TS. NGUYỄN HỮU LỘC, NGUYỄN THANH TRUNG

Lập trình thiết kế với AutoLISP và Visual LISP

TẬP 2

In lần thứ 2

NHÀ XUẤT BẢN THÀNH PHỐ HỒ CHÍ MINH - 2003

LỜI NÓI ĐẦU

Trong tập hai này chúng tôi tiếp tục biên soạn và giới thiệu 08 chương, bao gồm: truy xuất đối tượng trong cơ sở dữ liệu, quản lý file và môi trường làm việc, các bảng mô tả, các hàm xử lý màn hình và thiết bị nhập, tạo hộp thoại, các hàm điều khiển hộp thoại, các kỹ thuật lập trình và gỡ rối chương trình, giới thiệu môi trường lập trình **Visual LISP**, các ví dụ lập trình ứng dụng tạo các lệnh mới cho **AutoCAD** và chương trình vẽ thiết kế tự động. Trong phần phụ lục giới thiệu bảng tra cứu các mã DXF và bảng tra cứu các hàm **AutoLISP**.

Qua lần in thứ nhất (Ngôn ngữ lập trình **AutoLISP**) vào năm 2001 chúng tôi nhận được rất nhiều đóng góp bạn đọc xa gần. Trong lần xuất bản này chúng tôi thay đổi tên sách cho phù hợp nội dung và trong quá trình biên soạn để tái bản không thể tránh được thiếu sót. Chúng tôi xin thành thật cảm ơn các bạn có ý kiến đóng góp, phê bình những thiếu sót của sách để cho các lần xuất bản sau được hoàn thiện hơn. Mọi ý kiến đóng góp, phê bình và thắc mắc xin gửi đến địa chỉ:

Nguyễn Hữu Lộc, Trường Đại học Bách Khoa TP. Hồ Chí Minh, 268 Lý Thường Kiệt, Quận 10.

hoặc liên hệ trực tiếp qua email: ***nhlcad@yahoo.com***

TP Hồ Chí Minh, 07 - 2003

NỘI DUNG

Chương 13

TRUY XUẤT ĐỐI TƯỢNG TRONG CƠ SỞ DỮ LIỆU CỦA AutoCAD

Nội dung chương

1. Truy xuất dữ liệu của đối tượng trong cơ sở dữ liệu bản vẽ **AutoCAD**.
2. Hiệu chỉnh *record* dữ liệu và cập nhật vào cơ sở dữ liệu.
3. Truy xuất dữ liệu của các đối tượng con.
4. Tạo các đối tượng mới trực tiếp trong cơ sở dữ liệu.

Một trong những ưu điểm chính của chương trình **AutoLISP** là có thể thao tác trực tiếp cơ sở dữ liệu đối tượng của **AutoCAD**. Nhờ đó, tốc độ và hiệu quả của chương trình sẽ tăng lên so với việc sử dụng hàm **Command**.

Vì các dữ liệu mô tả đối tượng được **AutoLISP** biểu diễn ở dạng danh sách, nên chúng ta cần phải thông thạo các kỹ thuật xử lý danh sách của ngôn ngữ **AutoLISP**. Các kỹ thuật này đã được trình bày ở tập 1.

13.1 Truy xuất dữ liệu đối tượng

Trong chương trước, chúng ta đã biết cách tạo ra các tập hợp chọn và lưu giữ để sử dụng về sau. Tập hợp chọn đặc biệt có ích khi thao tác nhiều đối tượng cùng lúc. Trong chương này, chúng ta sẽ biết thêm các phương pháp khác để truy xuất đối tượng trong cơ sở dữ liệu của **AutoCAD**.

Hàm ENTNEXT

Khi một đối tượng mới được tạo ra, trong cơ sở dữ liệu của **AutoCAD** sẽ xuất hiện một *record* để lưu trữ thông tin về đối tượng này. Mỗi đối tượng được **AutoCAD** gán cho một *mã đối tượng (Entity name)* để quản lý (khác với tên đối tượng do người sử dụng đặt). Khi một đối tượng bị xóa, **AutoCAD** không xóa hẳn *record* của đối tượng này, mà chỉ đánh dấu *bị xóa*. Ta gọi đối tượng này là *đối tượng bị đánh dấu xóa*. Các đối tượng khác gọi là *không bị đánh dấu xóa*. Ta có thể phục hồi đối tượng bị đánh dấu xóa bằng lệnh **Oops** của **AutoCAD**.

Hàm **Entnext** (*ENTity NEXT*) lấy ra mã đối tượng (không bị đánh dấu xóa) kế tiếp đối tượng có mã là ENAME trong cơ sở dữ liệu.

(**Entnext** [ENAME])

Nếu không có tham số ENAME, hàm sẽ trả về mã đối tượng không bị đánh dấu xóa *đầu tiên* trong cơ sở dữ liệu.

Ví dụ:

Hãy tạo một bản vẽ mới, sau đó thực hiện các lệnh sau.

Command: **(entnext)** ↵	
nil	Không có đối tượng nào
Command: **Line** ↵	Đối tượng thứ nhất
From point: **2,2** ↵	
To point: **100,100** ↵	
To point: ↵	
Command: **Line** ↵	Đối tượng thứ hai
From point: **5,5** ↵	
To point: **10,10** ↵	
To point: ↵	

Command: **Circle** ↵ *3P/2P/TTR/<Center point>:* **10,10** ↵ *Diameter/<Radius>:* **50** ↵ *Command:*	Đối tượng thứ ba
Command: **(setq E1 (entnext))** ↵ *<Entity name: 8f0505d0>*	**Entnext** không tham số trả về mã của đối tượng đầu tiên.
Command: **(setq E2 (entnext E1))**↵ *<Entity name: 8f0505d8>*	Đối tượng kế tiếp đối tượng E1
Command: **(setq E3 (entnext E2))** ↵ *<Entity name: 8f0505e0>*	Đối tượng kế tiếp đối tượng E2
Command: **(setq E4 (entnext E3))** ↵ *nil*	Chỉ có 3 đối tượng, nên trả về *nil*

Ta có thể sử dụng vòng lặp **While** và hàm **Entnext** để duyệt từng đối tượng trong cơ sở dữ liệu từ đầu đến cuối. Vòng lặp kết thúc khi hàm **Entnext** trả về nil; tức là đã đến vị trí cuối cùng.

✌ **Ví dụ:**

Chương trình duyệt cơ sở dữ liệu. Kiểm tra có đối tượng nào không. Nếu không, thông báo trên màn hình. Nếu có, sử dụng hàm **Entnext** để lấy ra mã đối tượng. Sau đó sử dụng hàm **While** để duyệt tất cả các đối tượng.

```
;Tên file: PR-ENAME.LSP
;
(if (setq E (entnext))
  (progn
    (print E)
    (while (setq E (entnext E))
       (print E)
    )
  );Đóng progn
  (princ "\nNo entities found in database!")
);Đóng if
```

```
(princ)
;Kết thúc chương trình
```

Thi hành chương trình PR-ENAME.LSP:

Command: **(load "PR-ENAME")** ↵
<Entity name: 8f0505d0>
<Entity name: 8f0505d8>
<Entity name: 8f0505e0>
Command:

Hàm ENTGET

Hàm **Entget** (*ENTity GET*) trả về danh sách biểu diễn *record* dữ liệu của đối tượng có mã ENAME. Tham số tùy chọn APPLIST cho phép truy xuất đến phần dữ liệu mở rộng.

(**Entget** ENAME [APPLIST])

Mỗi *field* dữ liệu được biểu diễn bằng một danh sách con. Phần tử đầu tiên của danh sách con là một *mã DXF* (*DXF group code*), biểu diễn một thuộc tính của đối tượng. Phần tử thứ hai là giá trị của thuộc tính đó.

Ví dụ:

Command: **(entget (entnext))** ↵
((-1 . <Entity name: 2e10500>) (0 . "LINE") (5 . "20") (100 . "AcDbEntity") (67. 0) (8 . "0") (100 . "AcDbLine") (10 2.0 2.0 0.0) (11 100.0 100.0 0.0) (210 0.0 0.0 1.0))

Giải thích:

```
(
```

(-1 . <Entity name: 2e10500>)	Mã đối tượng
(0 . "LINE")	Kiểu đối tượng là đường thẳng
(5 . "20")	Giá trị *handle*
(100 . "AcDbEntity")	Đánh dấu phân loại các mã DXF theo sau
(67. 0)	Đối tượng thuộc không gian mô hình
(8 . "0")	Đối tượng thuộc lớp bản vẽ 0
(100 . "AcDbLine")	Đánh dấu phân loại các mã DXF theo sau

(10 2.0 2.0 0.0)	Điểm đầu của đoạn thẳng
(11 100.0 100.0 0.0)	Điểm cuối của đoạn thẳng
(210 0.0 0.0 1.0)	Hướng trục Z của đối tượng

)

Chú ý:

Phụ thuộc vào từng phần mềm **AutoCAD** cụ thể, mã đối tượng có thể khác nhau. Ngoài ra, tùy vào đặc điểm môi trường bản vẽ khi tạo đối tượng, các dữ liệu mô tả đối tượng có thể khác đi.

Ví dụ: Dữ liệu đối tượng đường tròn E3 trong ví dụ ở trên

Command*:* **(entget E3)** ↵

(

(-1 . <Entity name: 2e10518>)	Mã đối tượng
(0 . "CIRCLE")	Đối tượng có kiểu là đường tròn
(5 . "23")	Giá trị handle
(100 . "AcDbEntity")	Đánh dấu phân loại các mã DXF theo sau
(67 . 0)	Đối tượng thuộc không gian mô hình
(8 . "0")	Đối tượng thuộc lớp bản vẽ 0
(100 . "AcDbCircle")	Đánh dấu phân loại các mã DXF theo sau
(10 10.0 10.0 0.0)	Tọa độ tâm đường tròn
(40 . 50.0)	Bán kính
(210 0.0 0.0 1.0)	Hướng trục Z của đối tượng

)

Trong *record* dữ liệu đối tượng, có những *field* dữ liệu thường xuất hiện nhưng cũng có các *field* chỉ xuất hiện khi cần thiết. Ví dụ như màu sắc và dạng đường của đối tượng. Nếu màu sắc và dạng đường của đối tượng có giá trị mặc định là BYLAYER, thì các *field* dữ liệu này không xuất hiện. Nếu được gán giá trị khác, thì các *field* này sẽ xuất hiện.

Ví dụ:

. Xuất hiện thêm 3 *field*: 62 - màu sắc, 6 - dạng đường, 48 - tỉ lệ dạng đường.

Command: **Chprop** ↵

Select objects: **(entnext)** ↵

1 found
Select objects: ↵
Change what property (Color/LAyer/LType/ltScale/Thickness) ? **C** ↵
New color <BYLAYER>: **5** ↵
Change what property (Color/LAyer/LType/ltScale/Thickness) ? **LT** ↵
New linetype <BYLAYER>: **CENTER** ↵
Change what property (Color/LAyer/LType/ltScale/Thickness) ? **S** ↵
New linetype scale <1.0000>: **1.1**
Change what property (Color/LAyer/LType/ltScale/Thickness) ? ↵

Command: **(entget (entnext))** ↵
((-1 . <Entity name: 2e10520>) (0 . "LINE") (5 . "24") (100 . "AcDbEntity")
*(67 . 0) (8 . "0") **(62 . 5) (6 . "CENTER") (48 . 1.1)** (100 . "AcDbLine")*
(10 2.0 2.0 0.0) (11 100.0 100.0 0.0) (210 0.0 0.0 1.0))

Ta có thể dễ dàng lấy ra *field* dữ liệu mong muốn và giá trị của nó nhờ vào các *mã DXF* (xem chi tiết các mã DXF trong phần phụ lục).

Ví dụ: Lấy ra kiểu đối tượng

(assoc 0 (entget (entnext)))	trả về (0 . "LINE")
(cdr (assoc 0 (entget (entnext))))	trả về "LINE"

Truy xuất dữ liệu của đối tượng con

Đối tượng con (subentity) là một trong các đối tượng thành phần tạo nên đối tượng phức. Ví dụ: đỉnh của đa tuyến, thuộc tính của khối, ... Trong cơ sở dữ liệu **AutoCAD**, một đối tượng phức được mô tả bằng nhiều *record* nằm liên tiếp nhau. *Record* đầu tiên mô tả *đối tượng chính*. Các *record* tiếp theo mô tả các đối tượng con. *Record* cuối cùng luôn luôn được gọi là *record SEQEND* (*SEQuence END*), báo hiệu không còn đối tượng con nào nữa .

Ví dụ:

Khởi tạo bản vẽ mới, sau đó thực hiện các lệnh sau. Theo mặc định, **AutoCAD 14** và **2000** tạo các đa tuyến dạng *lightweight*, là loại đối tượng đơn không chứa dữ liệu các đỉnh. Để tạo đa tuyến theo dạng cũ sử dụng trong ví dụ này ta phải gán biến hệ thống PLINETYPE bằng 2.

Command: **Plinetype** ↵
New value for PLINETYPE <2>: **0** ↵
Command: **Pline** ↵
From point: **0,0** ↵
Current line-width is 0.0000
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **120,90** ↵
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **40,80** ↵
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **c** ↵
Command: **(sslength (ssget "x"))** ↵
1

Trong cơ sở dữ liệu đang chứa 1 *record* chính, và 4 *record* con. Tuy nhiên, vì hàm **Ssget** chỉ truy xuất được đối tượng chính, nên hàm **Sslength** trả về kết quả là 1 *record.* Để truy xuất các *record* con, ta bổ sung thêm hàm *ENTGET* vào trong chương trình PR-EDATA.LSP như sau:

```
;Tên file: PR-EDATA.LSP
;
(if (setq E (entnext))
  (progn
    (print (entget E))
    (while (setq E (entnext E))
       (print (entget E))
    )
  );Đóng progn
  (princ "\nNo entities found in database!")
);Đóng if
(princ)
;Kết thúc chương trình
```

Thi hành chương trình PR-EDATA.LSP:

Command: **(load "pr-edata")** ↵
((-1 . <Entity name: 34b0520>) (0 . "POLYLINE") (5 . "4C")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDb2dPolyline") (66 . 1)
(10 0.0 0.0 0.0) (70 . 1) (40 . 0.0) (41 . 0.0) (210 0.0 0.0 1.0) (71 . 0) (72 . 0)
(73 . 0) (74 . 0) (75 . 0))
((-1 . <Entity name: 34b0528>) (0 . "VERTEX") (5 . "4D") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex") (10 0.0 0.0 0.0)
(40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))

((-1 . <Entity name: 34b0530>) (0 . "VERTEX") (5 . "4E") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 120.0 90.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))
((-1 . <Entity name: 34b0538>) (0 . "VERTEX") (5 . "4F") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 40.0 80.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))
((-1 . <Entity name: 34b0540>) (0 . "SEQEND") (5 . "50") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (-2 . <Entity name: 34b0520>))

Giải thích record chính:

(	
(-1 . <Entity name: 34b0520>)	Mã đối tượng
(0 . "POLYLINE")	Kiểu đối tượng là đa tuyến
(5 . "4C")	Giá trị *handle*
(100 . "AcDbEntity")	Đánh dấu phân loại các mã DXF
(67 . 0)	Đối tượng thuộc không gian mô hình
(8 . "0")	Đối tượng thuộc lớp bản vẽ 0
(100 . "AcDb2dPolyline")	Đánh dấu phân loại các mã DXF
(66 . 1)	Cờ báo hiệu có đối tượng con
(10 0.0 0.0 0.0)	Tọa độ điểm đầu tiên
(70 . 1)	1 = closed (đa tuyến đóng)
(40 . 0.0)	Chiều rộng đầu phân đoạn
(41 . 0.0)	Chiều rộng cuối phân đoạn
(210 0.0 0.0 1.0)	Hướng véc tơ chiều dày.
(71 . 0)	Số các đỉnh theo chiều M (lưới 3D)
(72 . 0)	Số các đỉnh theo chiều N (lưới 3D)
(73 . 0)	Độ mịn bề mặt theo chiều M (lưới 3D)
(74 . 0)	Độ mịn bề mặt theo chiều N (lưới 3D)
(75 . 0)	Dạng đường cong
)	

Giải thích record SEQEND:

(	
(-1 . <Entity name: 34b0540>)	Mã đối tượng
(0 . "SEQEND")	Kiểu đối tượng là SEQEND
(5 . "50")	Giá trị *handle*
(100 . "AcDbEntity")	Đánh dấu phân loại các mã DXF
(67 . 0)	Đối tượng thuộc không gian mô hình
(8 . "0")	Đối tượng thuộc lớp bản vẽ 0

(-2 . <Entity name: 34b0520>))	Mã đối tượng chính. Cho biết các đối tượng con là của đối tượng chính nào, vì các đối tượng có thể lồng vào nhau, nghĩa là một đối tượng chính là con của một đối tượng chính khác. Ví dụ: Một đa tuyến chứa trong một khối.

Ví dụ:

Sau đây là chương trình lấy ra thông tin về một đa tuyến, gồm *record* chính và các *record* con.

```
;Tên file: SUBLST.LSP
;
(defun C:SUBLST (/ SS EN  EL ME SN SE)
  (princ "\nSelect a polyline: ")
  (setq SS (ssget)                ;Giả sử người sử dụng chọn một đa tuyến
                                  ;theo đúng yêu cầu.
        EN (ssname SS 0)          ;EN là mã đối tượng đầu tiên trong SS.
        EN (entnext EN)           ;EN là đối tượng tiếp theo trong CSDL
        EL '()                    ;Khai báo EL là danh sách rỗng, dùng để
                                  ;chứa mã các đối tượng con.
  )                               ;Đóng setq
  (while
    (/= "SEQEND"                  ;Duyệt các record, bắt đầu từ record EN
      (cdr (assoc 0 (entget EN))) ;cho đến khi gặp SEQEND thì dừng.
    )                             ; Đóng /=
    (setq EL (append EL (list EN))  ;Thêm mã đối tượng con
                                    ; vào danh sách EL.
          EN (entnext EN)           ;Chuyển qua record kế tiếp.
    )
   )                              ; Đóng while
   (setq SE EN
         ME (cdr (assoc -2 (entget EN)))   ;ME chứa mã đối tượng chính
                                           ;Trong record SEQEND, field
                                           ;-2 chứa mã đối tượng chính
   )                                       ; Đóng setq
   (princ "\nMain entity name: ")
   (princ ME)                              ;In mã đối tượng chính
   (foreach SN EL     ;Duyệt từng mã đối tượng con trong danh sách EL
```

```
    (princ "\nSubentity name: ")
    (princ SN)                          ;In mã đối tượng con
  )                                     ; Đóng Foreach
  (princ "\nSEQEND entity name: ")
  (princ SE)                            ;In mã đối tượng SEQEND
  (princ)
)                                       ;Đóng defun
;Kết thúc chương trình
```

Thi hành chương trình trên:

Command: **sublst** ↵
Select a polyline:
Select objects: 1 found
Select objects: ↵
Main entity name: <Entity name: 34a0520>
Subentity name: <Entity name: 34a0528>
Subentity name: <Entity name: 34a0530>
Subentity name: <Entity name: 34a0538>
Subentity name: <Entity name: 34a0540>

Ví dụ trên trình bày một phương pháp sử dụng vòng lặp duyệt cơ sở dữ liệu để tìm một đối tượng mong muốn. Nó cho phép tìm kiếm các đối tượng dựa theo các thuộc tính mà hàm **Ssget** "X" không lấy ra được.

Hàm ENTLAST

Hàm **Entlast** (*ENTity LAST*) trả về mã của đối tượng chính cuối cùng (không bị đánh dấu xóa) trong cơ sở dữ liệu. Hàm này không có tham số.

(**Entlast**)

Hàm này thường được sử dụng để truy xuất dữ liệu của đối tượng vừa mới được tạo ra. Tương tự như hàm **Entnext**, hàm **Entlast** có thể truy xuất các đối tượng của các lớp bị đóng băng hoặc bị tắt.

Ví dụ:

Command: **(entlast)** ↵
<Entity name: 34a0520>
Command:

Khi cần truy xuất các đối tượng con của đối tượng chính cuối cùng, ta sử dụng thêm hàm **Entnext**.

Ví dụ:

Command: **(entnext (entlast))** ↵
<Entity name: 34a0528>

Giá trị HANDLE và hàm HANDENT

Mã đối tượng (*entity name*) chỉ tồn tại khi bản vẽ còn đang mở. Nó sẽ bị mất đi khi ta đóng bản vẽ lại. Khi mở lại bản vẽ này, **AutoCAD** sẽ gán mã mới cho các đối tượng. Tuy nhiên, trong nhiều trường hợp, ta cần truy xuất cùng một đối tượng trong các lần đóng mở bản vẽ. Khi đó, ta sử dụng giá trị *handle* của đối tượng. Giá trị này tồn tại vĩnh viễn cho đến khi đối tượng bị xóa, không phụ thuộc vào việc đóng mở bản vẽ. *Field* có mã **DXF 5** lưu giữ giá trị *handle* của đối tượng.

Hàm **Handent** (*HANDle ENTity*) trả về mã đối tượng (*entity name*) có giá trị *handle* chứa trong tham số HANDLE.

(**Handent** HANDLE)

Ví dụ:

Command*:* **Line** ↵
From point: **0,0** ↵
To point: **50,50** ↵
To point: ↵
Command: **List** ↵
Select objects: **L** ↵
Select objects: ↵

Trên cửa sổ *text window* xuất hiện thông tin về đường thẳng vừa vẽ. Ta thấy giá trị *handle* của đường thẳng này bằng 52.

LINE Layer: 0
Space: Model space
Handle = 52
from point, X= 0.0000 Y= 0.0000 Z= 0.0000
to point, X= 50.0000 Y= 50.0000 Z= 0.0000
Length = 70.7107, Angle in XY Plane = 45
Delta X = 50.0000, Delta Y = 50.0000, Delta Z = 0.0000

Command: **(entlast)** ↵ ;Mã đối tượng vừa vẽ
<Entity name: 2df0550>
Command: **(handent "52")** ↵ ;Mã đối tượng có *handle* bằng 52
<Entity name: 2df0550> ;Ghi chú: *Handle* có kiểu dữ liệu là chuỗi

Tóm tắt

1. Hàm **Entnext** cho phép tìm kiếm trực tiếp trong cơ sở dữ liệu.
2. Hàm **Entnext** trả về mã đối tượng. Nó có thể truy xuất đối tượng chính lẫn đối tượng con.
3. Hàm **Entget** trả về toàn bộ record dữ liệu của đối tượng
4. Đối tượng **SEQEND** đánh dấu vị trí cuối cùng của các đối tượng con trong cơ sở dữ liệu.
5. Vòng lặp có điều kiện giúp tìm kiếm hiệu quả trong cơ sở dữ liệu, hoặc giúp trả về mã của các đối tượng con tạo thành đối tượng phức.
6. Hàm **Entlast** truy xuất đối tượng cuối cùng vừa vẽ, nhưng chỉ sử dụng được đối với các đối tượng chính.
7. Hàm **Handent** trả về mã của đối tượng thông qua giá trị *handle* của đối tượng đó.

13.2 Các hàm chọn đối tượng trực tiếp

Hàm **Ssget** cho phép người sử dụng chọn các đối tượng để xử lý. Tuy nhiên, hàm này không kiểm soát được số lượng các đối tượng được chọn, cũng như không có thông tin về điểm chọn trên đối tượng.

Các hàm chọn đối tượng trực tiếp cho phép kiểm soát tốt hơn quá trình chọn, cũng như biết được tọa độ điểm chọn trên đối tượng.

Hàm ENTSEL

Hàm **Entsel** *(ENTity SELect)* cho phép người sử dụng chọn đối tượng bằng cách chọn một điểm trên màn hình bằng chuột hoặc nhập tọa độ điểm chọn. Nếu không tìm được đối tượng nào tại điểm chọn này thì hàm trả về *nil*. Nếu có, hàm trả về một danh sách gồm 2 phần tử: mã đối tượng và tọa độ điểm chọn.

(**Entsel** [PROMPT])

Tham số tùy chọn PROMPT sử dụng để hiển thị dòng nhắc khi chọn đối tượng. Dòng nhắc mặc định khi không có PROMPT là *"Select object: "*.

Ví dụ:

Command: **(setq ES (entsel))** ↵
Select object: (Chọn một đối tượng)
(<Entity name: 2d70610> (-306.35 34.2936 0.0))

Command:

Command: **(entsel "\nPick a line entity: ")** ↵
Pick a line entity: (Chọn một đối tượng)
(<Entity name: 2d70610> (-233.757 39.4646 0.0))

Command: **(car ES)** ↵ ;Lấy ra mã đối tượng
<Entity name: 2d70610>

Command: **(cadr ES)** ↵ ;Lấy ra tọa độ điểm chọn
(-241.535 38.4304 0.0)

Command: **Copy** ↵
Select objects: **(entsel)** ↵ ;Sử dụng được tại các dòng nhắc
Select object: (Chọn một đối tượng) ;chọn đối tượng của các lệnh
Select objects: ;**AutoCAD**

Một đặc điểm có ích của hàm **Entsel** là chỉ cho phép chọn **một** đối tượng. Nếu không chọn được đối tượng nào thì trả về nil. Nhờ đó, ta có thể sử dụng làm biểu thức điều kiện trong vòng lặp **While**.

Ví dụ:

Vòng lặp sau đây yêu cầu người sử dụng phải chọn một đối tượng thì mới chịu kết thúc và tiếp tục chương trình.

```
(while (null (setq ES (entsel)))
  (princ "\nNo object found!")
)
```

Hàm NENTSEL

Hàm **Nentsel** cho phép chọn trực tiếp một đối tượng thuộc tính trong một khối hoặc một đỉnh trong một đa tuyến.

(**Nentsel** [PROMPT])

Tham số PROMPT tương tự như của hàm ENTSEL.

Đối với các đối tượng đơn, như đường thẳng, cung tròn... hàm **Nentsel** trả về dữ liệu giống như hàm **Entsel**. Đối với đối tượng phức, hàm **Nentsel** trả về mã của đối tượng con và tọa độ điểm chọn.

Hàm NENTSEL và đa tuyến

Khi đối tượng chọn là đa tuyến, hàm **Nentsel** trả về danh sách chứa mã đối tượng là đỉnh đầu tiên của đa tuyến và tọa độ điểm chọn.

Chú ý:

Hàm **Nentsel** không bao giờ trả về mã đối tượng **SEQEND** của đa tuyến.

Ví dụ:

Command: **(nentsel)** ↵
Select object: (Chọn phân đoạn đầu tiên của đa tuyến)
(<Entity name: 2d70528> (4.90568 4.80963 0.0))

Lấy ra *record* của đối tượng có mã do hàm **Nentsel** trả về:

Command: **(entget (car (nentsel)))** ↵
Select object: (Chọn phân đoạn đầu tiên của đa tuyến)
((-1 . <Entity name: 2d70528>) (0 . "VERTEX") (5 . "4D")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbVertex")
(100 . "AcDb2dVertex") (10 3.5626 3.78697 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(70 . 0) (50 . 0.0))

Khi cần lấy mã đối tượng chính của đa tuyến, ta sử dụng phương pháp duyệt từng *record* con bằng hàm **Entnext** cho đến khi gặp *record* SEQEND. Sau đó lấy ra mã đối tượng chính chứa trong *record* SEQEND này. Khi đã có mã đối tượng chính, ta sử dụng hàm **Entget** để lấy thông tin về đối tượng này.

Ví dụ:

```
(setq EN (car (nentsel "\nPick a polyline entity: ")))          ; [1]
(while (/= "SEQEND" (cdr (assoc 0 (entget EN))))                ; [2]
```

```
  (setq EN (entnext EN))                                    ; [3]
)                                                  ;Đóng while
(princ (cdr (assoc –2 (entget EN))))                        ; [4]
```

Giải thích:

[1] **(setq EN (car (nentsel "\nPick a polyline entity: ")))** Khi chọn một đa tuyến bằng hàm **Nentsel**, giá trị trả về là *record* con đầu tiên. Hàm **Car** lấy ra mã đối tượng con này.

[2] **(while (/= "SEQEND" (cdr (assoc 0 (entget EN))))** Hàm **Entget** lấy ra *record* tiếp theo. Hàm **Assoc** lấy ra *field* 0, là *field* chứa kiểu đối tượng. Hàm **Cdr** lấy ra kiểu đối tượng. Kiểu đối tượng sẽ được so sánh có phải là "SEQEND" hay không.

[3] **(setq EN (entnext EN))** Nếu không phải là "SEQEND", vòng lặp **While** sẽ chuyển qua *record* con kế tiếp bằng hàm **Entnext**.

[4] **(princ (cdr (assoc –2 (entget EN))))** Khi kết thúc vòng **While**, ta đang ở vị trí record "SEQEND". Hàm **Entget** lấy ra *record* này. Hàm **Assoc** lấy ra *field* –2, là *field* chứa mã đối tượng chính. Hàm **Cdr** lấy ra mã đối tượng chính. Mã này được in ra màn hình bằng hàm **Princ**.

Hàm NENTSEL và khối

Các thuộc tính của khối, tương tự như các đỉnh của đa tuyến, là các đối tượng con và không thể truy xuất bằng các hàm **Ssget** hoặc **Entsel**. Hàm **Nentsel** cho phép truy xuất đến các thuộc tính trong một khối.

Để nghiên cứu hàm này, chúng ta tạo một bản vẽ mới và lần lượt thực hiện các lệnh sau:

Command: **Circle** ↵
3P/2P/TTR/<Center point>: **4,4** ↵
Diameter/<Radius>: **50** ↵

Command: **Attdef** ↵
Attribute modes -- Invisible:N Constant:N Verify:N Preset:N
Enter (ICVP) to change, or press ENTER when done: ↵
Attribute tag: **TAG** ↵
Attribute prompt: **PROMPT?** ↵
Default attribute value: **DEFAULT** ↵

Justify/Style/<Start point>: **4,4** ↵
Height <0.2000>: **2.5** ↵
Rotation angle <0>: **0** ↵

Command: **Block** ↵
Block name (or ?): **CIR** ↵
Insertion base point: **4,4** ↵
Select objects: **W** ↵
First corner: **-70,60** ↵
Other corner: **70,-50** ↵
2 found
Select objects: ↵

Command: **Vpoint** ↵
Rotate/<View point> <0.0000,0.0000,1.0000>: **2.5, 2.5, -1.0** ↵
Regenerating drawing.

Command: **Ucs** ↵
Origin/ZAxis/3point/OBject/View/X/Y/Z/Prev/Restore/Save/Del/?/<World>: **v** ↵

Command: **insert** ↵
Block name (or ?): **CIR** ↵
Insertion point: **4,4** ↵
X scale factor <1> / Corner / XYZ: **1** ↵
Y scale factor (default=X): **1** ↵
Rotation angle <0>: **0** ↵
Enter attribute values
Prompt? <DEFAULT>: ↵

Command: **Zoom** ↵
All/Center/Dynamic/Extents/Previous/Scale(X/XP)/Window/<Realtime>: **e** ↵
Regenerating drawing.

Thông tin do hàm **Nentsel** trả về phụ thuộc vào ta sử dụng chuột chọn vào phần nào của khối.

Nếu chọn vào thuộc tính của khối, danh sách trả về chỉ chứa mã đối tượng con và tọa độ điểm chọn.

Ví dụ:

Command: **(nentsel)** ↵
Select object: (Chọn vào thuộc tính của khối)
(<Entity name: 34c0560> (17.9853 3.71537 0.0))

Để lấy ra toàn bộ *record* của đối tượng thuộc tính, ta sử dụng hàm **Entget**.

Ví dụ:

Command: **(entget (car (nentsel)))** ↵
Select object: (Chọn vào thuộc tính của khối)
((-1 . <Entity name: 34c0560>) (0 . "ATTRIB") (5 . "54")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbText")
(10 4.0 4.0 -6.66134e-016) (40 . 2.5) (1 . "DEFAULT") (50 . 0.0) (41 . 1.0)
(51 . 0.0) (7 . "STANDARD") (71 . 0) (72 . 0) (11 0.0 0.0 -6.66134e-016)
(210 0.680414 0.680414 -0.272166) (100 . "AcDbAttribute") (2 . "TAG")
(70 . 0) (73 . 0) (74 . 0))

Nếu chọn vào các thành phần khác của khối, thông tin trả về hoàn toàn khác hẳn.

Ví dụ:

Command: **(nentsel)** ↵
Select object: (Chọn vào đường tròn)
(<Entity name: 34c0548> (-38.5269 28.1936 0.0)
((-0.707107 0.707107 0.0) (0.19245 0.19245 0.96225)
(0.680414 0.680414 -0.272166) (-2.05863 3.59823 3.849))
(<Entity name: 34c0558>))

Giải thích:

(<Entity name: 34c0548>	Mã của đối tượng con được chọn
(-38.5269 28.1936 0.0)	Tọa độ điểm chọn
((-0.707107 0.707107 0.0) *(0.19245 0.19245 0.96225)* *(0.680414 0.680414 -0.272166)* *(-2.05863 3.59823 3.849)* *)*	Ma trận biến đổi tọa độ điểm từ hệ trục tọa độ đối tượng sang hệ trục tọa độ thực
(<Entity name: 34c0558>))	Mã đối tượng chính là khối chứa đối tượng con

Các khối có thể lồng nhau. Nghĩa là đối tượng con của một khối có thể lại là một khối khác. Khi đó, phần tử cuối cùng của danh sách trên là một danh sách chứa mã đối tượng các khối lồng vào trong đối tượng con được chọn.

(<Entity name: 34c0538>)) Mã đối tượng của khối ở mức sâu nhất
(<Entity name: 34c0578>)) . . . (<Entity name: 34c05c8>))
(<Entity name: 34c0558>)) Mã đối tượng của khối ở mức ngoài cùng (đối tượng chính)

Bởi vì hàm **Nentsel** trả về danh sách có phần tử đầu tiên chứa mã của đối tượng được chọn, nên *record* của đối tượng này có thể được lấy ra theo cách sau đây:

Command: **(entget (car (nentsel)))** ↵
Select object: (Chọn vào đường tròn)
((-1 . <Entity name: 34c0548>) (0 . "CIRCLE") (5 . "51")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 0.0 0.0 0.0)
(40 . 50.0) (210 0.0 0.0 1.0))

Ma trận biến đổi tọa độ điểm

AutoCAD xem khối như một đối tượng duy nhất, có một điểm đặc biệt là điểm chèn. Vị trí điểm chèn này trong hệ trục tọa độ thực WCS xác định vị trí cả khối. Để quản lý vị trí các đối tượng con bên trong một khối, **AutoCAD** sử dụng một hệ trục tọa độ khác, có gốc tọa độ trùng với điểm chèn của khối, gọi là hệ trục tọa độ đối tượng MCS (*Model Coordinate System*).

Xét lại ví dụ ở trên:

Command: **(entget (car (nentsel)))** ↵
Select object: (Chọn vào đường tròn)
((-1 . <Entity name: 34c0548>) (0 . "CIRCLE") (5 . "51")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 0.0 0.0 0.0)
(40 . 50.0) (210 0.0 0.0 1.0))

Mã DXF 10 xác định tọa độ tâm đường tròn (10 0.0 0.0 0.0). Vì tâm đường tròn trùng với điểm chèn của khối, nên trong hệ trục MCS nó có tọa độ là (0.0, 0,0, 0.0). Ta lấy ra tọa độ này bằng phương pháp sau:

Command: **(setq CP (cdr (assoc 10 (entget (car (nentsel))))))** ↵
(0.0 0.0 0.0)

Khi cần xác định vị trí của các đối tượng con trong hệ trục tọa độ thực, ta sử dụng một ma trận biến đổi tọa độ điểm từ hệ trục tọa độ đối tượng sang hệ trục tọa độ thực (*Model to World Transformation Matrix*):

$$\begin{bmatrix} X0 & Y0 & Z0 \\ X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{bmatrix}$$

Gọi X_{WCS}, Y_{WCS}, Z_{WCS} : tọa độ điểm trong hệ trục tọa độ WCS
Gọi X_{MCS}, Y_{MCS}, Z_{MCS} : tọa độ điểm trong hệ trục tọa độ MCS

Ta có công thức chuyển đổi tọa độ điểm từ MCS sang WCS:
X_{WCS} = (+ (* X1 X_{MCS}) (* X2 Y_{MCS}) (* X3 Z_{MCS}) X4)
Y_{WCS} = (+ (* Y1 X_{MCS}) (* Y2 Y_{MCS}) (* Y3 Z_{MCS}) Y4)
Z_{WCS} = (+ (* Z1 X_{MCS}) (* Z2 Y_{MCS}) (* Z3 Z_{MCS}) Z4)

Phần tử thứ ba trong danh sách do hàm **Nentsel** trả về chứa ma trận biến đổi này:

```
(
  (X0 Y0 Z0)
  (X1 Y1 Z1)
  (X2 Y2 Z2)
  (X3 Y3 Z3)
)
```

Ví dụ:
Command: **(setq MATRIX (caddr (nentsel)))** ↵
Select objects: (Chọn vào đường tròn)
((-0.707107 0.707107 0.0) (0.19245 0.19245 0.96225)
(0.680414 0.680414 -0.272166) (-2.05863 3.59823 3.849))

Viết lại ở dạng ma trận:

```
(
 (-0.707107   0.707107   0.0)
 (0.19245     0.19245    0.96225)
 (0.680414    0.680414  -0.272166)
 (-2.05863    3.59823    3.849)
)
```

Vì vậy ta có các biểu thức chuyển đổi tọa độ điểm từ MCS sang WCS như sau:

```
(setq WX
  (+
    (* (car (nth 0 MATRIX)) (car CP))
    (* (car (nth 1 MATRIX)) (cadr CP))
    (* (car (nth 2 MATRIX)) (caddr CP))
    (car (nth 3 MATRIX))
  )
)
(setq WY
  (+
    (* (cadr (nth 0 MATRIX)) (car CP))
    (* (cadr (nth 1 MATRIX)) (cadr CP))
    (* (cadr (nth 2 MATRIX)) (caddr CP))
    (cadr (nth 3 MATRIX))
  )
)
(setq WZ
  (+
    (* (caddr (nth 0 MATRIX)) (car CP))
    (* (caddr (nth 1 MATRIX)) (cadr CP))
    (* (caddr (nth 2 MATRIX)) (caddr CP))
    (caddr (nth 3 MATRIX))
  )
)
```

Áp dụng các biểu thức này và lưu kết quả vào một danh sách, ta được tọa độ điểm trong hệ trục WCS.

Command: **(setq W-XYZ (list WX WY WZ))** ↵
(-2.05863 3.59823 3.849)

Ta có thể kiểm tra tọa độ này bằng cách sử dụng lệnh **Id** của **AutoCAD**:

Command: **Ucs** ↵
Origin/ZAxis/3point/OBject/View/X/Y/Z/Prev/Restore/Save/Del/?/<World>:**W** ↵
Command: **id** ↵
Point: **CEN** ↵
of (Chọn vào đường tròn)
X = -2.05863 Y = 3.59823 Z = 3.849

Hàm NENTSELP

Tương tự như hàm **Nentsel**, hàm **Nentselp** cho phép truy xuất đến các đối tượng con của một khối. Tuy nhiên, hàm **Nentselp** còn cung cấp thêm các lựa chọn khác.

(**Nentselp** [PROMPT] [PT])

Tham số PROMPT

Chứa dòng nhắc (tương tự như hàm **Nentsel**).

Ví dụ:

Command: **(nentselp "\nPick a subentity: ")** ↵
Pick a subentity:

Nếu không có đối tượng nào được chọn, hàm trả về giá trị nil. Để bắt buộc người sử dụng phải chọn một đối tượng, ta sử dụng vòng lặp sau:

```
(while (null (setq NSP (nentselp "\nPick a subentity: ")))
  (princ "\nYou missed!")
)
```

Tham số PT

Là tùy chọn, không bắt buộc phải có, xác định tọa độ điểm chọn đối tượng. Khi đó, người sử dụng không cần phải chọn đối tượng.

Command: **(nentselp '(0 0))** ↵
nil ;Không có đối tượng nào tại điểm 0,0.

Hàm **Nentselp** trả về một danh sách tương tự như hàm **Nentsel**. Tuy nhiên, ma trận biến đổi tọa độ điểm có cấu trúc khác.

Command: **(nentselp)** ↵
Select object: (Chọn đường tròn)
(<Entity name: 2d705d0> (-43.9473 -3.67071 0.0)
((-0.707107 0.19245 0.680414 -2.05863)
(0.707107 0.19245 0.680414 3.59823) (0.0 0.96225 -0.272166 3.849)
(0.0 0.0 0.0 1.0)) (<Entity name: 2d70620>))

Ma trận có dạng như sau:

```
(
  (-0.707107  0.19245   0.680414  -2.05863)
  (0.707107   0.19245   0.680414   3.59823)
  (0.0        0.96225  -0.272166   3.849)
  (0.0        0.0       0.0        1.0)
)
```

Công thức chuyển đổi tọa độ điểm như sau:

X_{WCS} = (+ (* X1 X_{MCS}) (* Y1 Y_{MCS}) (* Z1 Z_{MCS}))
Y_{WCS} = (+ (* Y1 X_{MCS}) (* Y2 Y_{MCS}) (* Y3 Z_{MCS}))
Z_{WCS} = (+ (* Z1 X_{MCS}) (* Z2 Y_{MCS}) (* Z3 Z_{MCS}))

```
(setq WX
  (+
    (* (car (nth 0 MATRIX)) (car CP))
    (* (cadr (nth 0 MATRIX)) (cadr CP))
    (* (caddr (nth 0 MATRIX)) (caddr CP))
    (cadddr (nth 0 MATRIX))
  )
)
(setq WY
  (+
    (* (car (nth 1 MATRIX)) (car CP))
    (* (cadr (nth 1 MATRIX)) (cadr CP))
    (* (caddr (nth 1 MATRIX)) (caddr CP))
    (cadddr (nth 1 MATRIX))
  )
)
(setq WZ
  (+
    (* (car (nth 2 MATRIX)) (car CP))
    (* (cadr (nth 2 MATRIX)) (cadr CP))
    (* (caddr (nth 2 MATRIX)) (caddr CP))
    (cadddr (nth 2 MATRIX))
  )
)
```

Áp dụng các biểu thức này và lưu kết quả vào một danh sách, ta được tọa độ điểm trong hệ trục WCS.

Command: **(setq W-XYZ (list WX WY WZ))** ↵
(-2.05863 3.59823 3.849)

13.3 Hiệu chỉnh các record đối tượng

Để hiệu chỉnh giá trị các *field* của *record* đối tượng, ta tiến hành hai bước:

- Tạo ra *record* mới chứa các field đã thay đổi giá trị (bằng hàm **Subst**).
- Thay thế *record* cũ của đối tượng bằng *record* mới này (bằng hàm **Entmod**).

Hàm SUBST

Để tạo một *record* mới cho đối tượng, thay vì phải tạo từ đầu, ta có thể sử dụng hàm **Subst** để sao chép *record* cũ thành một *record* mới, và sửa các giá trị cần thay đổi.

(**Subst** NEW_ITEM OLD_ITEM LIST)

Hàm **Subst** sẽ thay thế mọi phần tử OLD_ITEM trong *record* LIST bằng NEW_ITEM và trả về *record* mới này. Nếu không tìm thấy OLD_ITEM, *record* mới sẽ giống với *record* cũ.

Hàm **Subst** sử dụng cho mọi kiểu danh sách của **AutoCAD**. Tuy nhiên, nó thường được sử dụng cho mục đích hiệu chỉnh *record*.

Ví dụ: Sử dụng cho danh sách bình thường

```
(setq DLST '(A 22 34 "YES" 7.0 B 22))
(subst 22 11 DLST)          trả về '(A 11 34 "YES" 7.0 B 11)
(subst 30 'A DLIST)         trả về '(30 22 34 "YES" 7.0 B 22)
(subst "OK" "YES" DLST)     trả về '(A 22 34 "OK" 7.0 B 22)
(subst "STOP" "GO" DLST)    trả về '(A 22 34 "YES" 7.0 B 22)
```

Ví dụ: Chuyển đối tượng từ lớp "0" sang lớp bản vẽ mới "NEWLAYER".

Command: **(setq EL (entget (entnext)))**↵

((-1 . <Entity name: 2d80500>) (0 . "LINE") (5 . "20") (100 . "AcDbEntity")
(67. 0) ***(8 . "0")*** *(100 . "AcDbLine") (10 225.67 204.913 0.0)*
(11 718.646 171.729 0.0) (210 0.0 0.0 1.0))

Command: **(setq EL (subst '(8 . "NEWLAYER") '(8 . "0") EL))**↵

((-1 . <Entity name: 2d80500>) (0 . "LINE") (5 . "20") (100 . "AcDbEntity")
(67 . 0) **(8 . "NEWLAYER")** (100 . "AcDbLine") *(10 225.67 204.913 0.0)*
(11 718.646 171.729 0.0) (210 0.0 0.0 1.0))

Sau khi sử dụng hàm **Subst**, *record* mới được tạo ra, nhưng đối tượng vẫn chưa được sửa đổi. Ta phải sử dụng hàm **Entmod** để cập nhật các thay đổi cho đối tượng này.

Hàm ENTMODE

Hàm ***Entmode*** *(ENTity MODify)* thay thế *record* cũ trong cơ sở dữ liệu bằng *record* mới.

(**Entmode** ELIST)

ELIST là một danh sách có dạng một *record* đối tượng. Mã đối tượng chứa trong mã DXF –1. Hàm **Entmode** sẽ tìm trong cơ sở dữ liệu *record* nào có cùng mã đối tượng này để thực hiện việc thay thế.

Command: **(entmod EL)** ↵
((-1 . <Entity name: 2d80500>) (0 . "LINE") (5 . "20") (100 . "AcDbEntity")
(67 . 0) (8 . "NEWLAYER") (100 . "AcDbLine") (10 225.67 204.913 0.0)
(11 718.646 171.729 0.0) (210 0.0 0.0 1.0))

Trong các trường hợp sau đây, việc thay thế không thực hiện được, hàm sẽ trả về nil:

- Thay đổi mã đối tượng (mã DXF –1 hoặc –2).
- Thay đổi kiểu đối tượng (mã DXF 0).
- Thay đổi giá trị *handle* của đối tượng (mã DXF 5).
- Hiệu chỉnh đối tượng viewport.
- Sử dụng các dạng đường, shape, kiểu chữ hoặc khối chưa tạo ra trong bản vẽ hiện hành. Tuy nhiên, với tên lớp bản vẽ, nếu chưa có thì nó sẽ được tạo ra với màu là WHITE, dạng đường là CONTINUOUS.

Khi hiệu chỉnh các đối tượng con của đối tượng phức, hàm **Entmod** không cập nhật lại đối tượng trên màn hình. Điều này cho phép người sử dụng thay đổi một loạt các đỉnh của đa tuyến hoặc các thuộc tính của khối mà không phải chờ đợi việc cập nhật lại màn hình cho mỗi lần thay đổi. Sau khi hiệu chỉnh xong, ta cập nhật đối tượng trên màn hình bằng lệnh **Entupd**.

Hàm ENTUPD

Hàm **Entupd** *(ENTity UPDate)* cập nhật lại đối tượng ENAME trên màn hình.

(**Entupd** ENAME)

Ta có thể cập nhật màn hình bằng lệnh **Regen** của **AutoCAD**, nhưng sẽ mất nhiều thời gian vì tất cả các đối tượng đều được cập nhật. Ngược lại, hàm **Entupd** chỉ cập nhật riêng một đối tượng.

Hàm **Entupd** có thể cập nhật mọi kiểu đối tượng, nhưng thường chỉ được sử dụng cho đa tuyến và khối.

Ví dụ:

Thay đổi đỉnh đầu tiên của một đa tuyến. Tạo bản vẽ mới rồi thực hiện các lệnh sau:

- Vẽ một đa tuyến qua các đỉnh (20,20), (40,20), (40,40), (20,40):

Command: **Pline** ↵
From point: **20,20** ↵
Current line-width is 0.0000
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **40,20** ↵
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **40,40** ↵
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **20,40** ↵
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: **c** ↵

- Mã của đối tượng đầu tiên:

Command: **(setq EN1 (entnext))** ↵
<Entity name: 2d80530>

- Kiểm tra đối tượng này có phải là đa tuyến không:

Command: **(entget EN1)** ↵
((-1 . <Entity name: 2d80530>) (0 . "POLYLINE") (5 . "26")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDb2dPolyline") (66 . 1)
(10 0.0 0.0 0.0) (70 . 1) (40 . 0.0) (41 . 0.0) (210 0.0 0.0 1.0) (71 . 0) (72 . 0)
(73 . 0) (74 . 0) (75 . 0))

- Đối tượng kế tiếp trong cơ sở dữ liệu là đỉnh đầu tiên của đa tuyến:

Command: **(setq EN2 (entnext EN1))** ↵
<Entity name: 2d80538>

- Để thay đổi tọa độ đỉnh này, trước tiên ta phải lấy *record* này ra:

Command: **(setq EG2 (entget EN2))** ↵
((-1 . <Entity name: 2d80538>) (0 . "VERTEX") (5 . "27")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbVertex")
(100 . "AcDb2dVertex") ***(10 20.0 20.0 0.0)*** *(40 . 0.0) (41 . 0.0) (42 . 0.0)*
(70 . 0) (50 . 0.0))

- Sau đó sử dụng hàm **Subst** để thay đổi tọa độ đỉnh. Biểu thức (assoc 10 EG2) trả về tọa độ cũ. Tọa độ mới là '(10.0 10.0 0.0):

Command: **(setq EG2 (subst '(10 10.0 10.0 0.0) (assoc 10 EG2) EG2))** ↵
((-1 . <Entity name: 2d80538>) (0 . "VERTEX") (5 . "27") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 10.0 10.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))

- Cập nhật thay đổi cho cơ sở dữ liệu bằng hàm **Entmod**. Tuy nhiên màn hình vẫn chưa được cập nhật.

Command: **(entmod EG2)** ↵
((-1 . <Entity name: 2d80538>) (0 . "VERTEX") (5 . "27") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 10.0 10.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))

- Cập nhật đa tuyến trên màn hình bằng hàm **Entupd**:

Command: **(entupd EN2)** ↵
<Entity name: 2d80538>

Tóm tắt

1. Hàm **Entsel** trả về danh sách chứa 2 phần tử: mã đối tượng và tập hợp điểm chọn. Để lấy ra mã đối tượng được chọn ta sử dụng biểu thức:

   (car (entsel))

2. Hàm **Nentsel** cho phép chọn trực tiếp một đối tượng con như đỉnh của đa tuyến, thuộc tính của khối hoặc các thành phần của khối. Khi chọn đối tượng đơn, hàm này có tác dụng tương tự hàm **Entsel**.
3. Hàm **Subst** sử dụng để thay thế dữ liệu trong danh sách mô tả đối tượng.
4. Khi một *record* đối tượng mới được tạo ra, hàm **Entmod** sẽ cập nhật những thay đổi vào cơ sở dữ liệu và cập nhật lại màn hình.

5. Hàm **Entmod** không thể cập nhật màn hình cho các đối tượng con. Ta phải sử dụng hàm **Entupd**.

Chương trình mẫu

Chương trình sau đây sử dụng nhiều kiến thức được trình bày ở trên. Chương trình này chuyển các đối tượng được chọn sang cùng lớp bản vẽ với một đối tượng được chọn khác.

```
;Tên file: CHL.LSP
;
(defun C:CHL (/ SS1 ENAM EGET EN EG COUNT)
  (princ "\nChange layer . . .\nSelect entities to change: ")
  (setq SS1 (ssget))                ;Tạo tập hợp các đối tượng cần thay đổi lớp
                                    ;bản vẽ
  (while
    (null (setq ENAM (car (entsel "\nPick an entity on the target layer: "))))
    (princ "\nYou missed.")         ;Chọn đối tượng của lớp bản vẽ mong muốn
  )                                 ;Vòng lặp WHILE bắt buộc người sử dụng
                                    ;phải chọn đối tượng, không được bỏ qua.
  (setq EGET (entget ENAM))         ;Gán record đối tượng cho biến  EGET.
  (setq COUNT 0)                    ;Bắt đầu với đối tượng đầu tiên trong SS1.
  (repeat (sslength SS1)            ;Lần lượt xét từng đối tượng trong SS1.
    (setq EN (ssname SS1 COUNT)     ;EN chứa mã đối tượng đang xét.
        EG (entget EN)              ;EG chứa record đối tượng
        EG (subst (assoc 8 EGET)    ;Thay thế lớp bản vẽ của đối tượng
                  (assoc 8 EG)      ;này bằng lớp bản vẽ của EG
                  EG                ;trong record EG
           )
        COUNT (1+ COUNT)            ;Chuyển sang đối tượng tiếp theo.
    )
    (entmod EG)                     ;Cập nhật cơ sở dữ liệu.
)                                   ;Đóng defun
;Kết thúc chương trình
```

13.4 Tạo đối tượng mới

Ngoài việc tạo đối tượng mới bằng các lệnh của **AutoCAD**, ta có thể tạo đối tượng mới trực tiếp trong cơ sở dữ liệu.

Hàm ENTMAKE

Hàm **Entmake** *(ENTity MAKE)* tạo đối tượng mới trực tiếp trong cơ sở dữ liệu. Tham số ELIST là danh sách chứa các dữ liệu mô tả đối tượng mới.

(**Entmake** [ELIST])

Các quy định của tham số ELIST:

- Tham số ELIST không nhất thiết phải chứa đủ các *field* sử dụng để mô tả đối tượng, nhưng phải chứa đủ các *field* bắt buộc.
- *Field* thứ nhất hoặc thứ hai phải là *field* kiểu đối tượng (mã DXF 0). Các *field* khác có thể có thứ tự bất kỳ.
- Mã đối tượng mới do **AutoCAD** tự động đặt ra, ta không được gán mã cho đối tượng mới.
- Một vài *field* như lớp bản vẽ, kiểu kích thước, kiểu chữ... là tùy chọn. Nếu trong tham số ELIST không có các *field* này, thì chúng sẽ được gán các giá trị mặc định như sau:

Lớp bản vẽ:	*CURRENT LAYER*
Kiểu kích thước:	**UNNAMED*
Kiểu chữ:	*STANDARD*
Hướng véc tơ trục Z:	*0,0,1*
Tọa độ Z của điểm:	*0*

Tự tạo tham số ELIST

Khi sử dụng hàm **Entmake** để tạo đối tượng mới, trước tiên ta nên xem *record* của kiểu đối tượng đó chứa những *field* nào, để tạo ra tham số ELIST cho phù hợp. Phương pháp này thường sử dụng cho các đối tượng đơn giản.

Ví dụ: Tạo đường thẳng mới có điểm đầu là 4,4 và điểm cuối là 50,60.

Command: **(entget (entlast))** ↵
((-1 . <Entity name: 34c0538>) (0 . "LINE") (5 . "4F") (100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbLine") (10 2.0 2.0 0.0) (11 40.0 40.0 0.0) (210 0.0 0.0 1.0))

Command: **(entmake '((0 . "LINE")(10 4 4)(11 50 60)))** ↵
((0 . "LINE") (10 4 4) (11 50 60))

Command: **(entget (entlast))** ↵
((-1 . <Entity name: 34c0540>) (0 . "LINE") (5 . "50") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbLine") (10 4.0 4.0 1.42251e-143)
(11 50.0 60.0 9.93543e-043) (210 0.0 0.0 1.0))

Vì ta chỉ cung cấp tọa độ điểm đầu (mã DXF 10) và điểm cuối (mã DXF 11), nên các *field* khác của đường thẳng mới sẽ lấy các giá trị mặc định. Vì tọa độ Z không được cung cấp nên giá trị mặc định Z = 0 trong WCS được sử dụng. Ngoài ra, WCS được sử dụng mặc định bởi vì ta không cung cấp hướng trục Z của đối tượng (mã DXF 210). Điều này đặc biệt quan trọng khi tạo các đối tượng trong không gian ba chiều.

Sử dụng record của đối tượng có sẵn

Ta sử dụng *record* của đối tượng có sẵn cùng kiểu, và thay thế các giá trị thích hợp để tạo ra *record* của đối tượng mới. Nhờ đó tiết kiệm được thời gian và ít gây ra lỗi khi tạo các đối tượng phức tạp.

Ví dụ:

```
(setq EG (enget (entlast))              ;gán record có sẵn cho biến EG
      EG (subst (cons 10 '(20 20))      ;Thay đổi tọa độ điểm đầu
                (assoc 10 EG)
                EG
         )
      EG (subst (cons 11 '(50 80))      ;Thay đổi tọa độ điểm cuối
                (assoc 11 EG)
                EG
         )
)
(entmake EG)                            ;Tạo đường thẳng mới đi qua 2 điểm
                                        ;(20,20) và (50,80)
```

Tạo đa tuyến mới

Vì đa tuyến bao gồm một *record* chính và nhiều *record* con, nên ta phải sử dụng liên tiếp nhiều biểu thức **Entmake** để tạo một đa tuyến mới.

Thứ tự tạo các *record*:
- *Record* chính
- Các *record* mô tả các đỉnh
- *Record* SEQEND.

Nếu không tạo theo đúng trình tự này, **AutoLISP** sẽ báo lỗi.

Ví dụ: Tạo đa tuyến đi qua 2 đỉnh 125,125 và 40,20

```
(entmake
   '((0 . "POLYLINE"))
)
(entmake
   '((0 . "VERTEX") (10 125 125 0))
)
(entmake
   '((0 . "VERTEX") (10 40 20 0))
)
(entmake
   '((0 . "SEQEND"))
)
```

Kiểm tra đa tuyến vừa tạo ra, ta thấy có nhiều *field* được gán giá trị mặc định:

Command: **(setq EN (entlast))** ↵
<Entity name: 34c05b0>
Command: **(while EN (print (entget EN))(setq EN (entnext EN)))** ↵
((-1 . <Entity name: 34c05b0>) (0 . "POLYLINE") (5 . "5E")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDb2dPolyline") (66 . 1)
(10 0.0 0.0 0.0) (70 . 0) (40 . 0.0) (41 . 0.0) (210 0.0 0.0 1.0) (71 . 0) (72 . 0)
(73 . 0) (74 . 0) (75 . 0))
((-1 . <Entity name: 34c05b8>) (0 . "VERTEX") (5 . "5F") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 125.0 125.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))
((-1 . <Entity name: 34c05c0>) (0 . "VERTEX") (5 . "60") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (100 . "AcDbVertex") (100 . "AcDb2dVertex")
(10 40.0 20.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 0) (50 . 0.0))
((-1 . <Entity name: 34c05c8>) (0 . "SEQEND") (5 . "61") (100 . "AcDbEntity")
(67 . 0) (8 . "0") (-2 . <Entity name: 34c05b0>))
nil

Tạo khối mới

Khi sử dụng lệnh **Block** hoặc **Bmake** ta tạo ra một *định nghĩa khối (block definition)* có kiểu đối tượng là "BLOCK". Sau đó, sử dụng lệnh **Insert** để chèn các khối vào bản vẽ, ta tạo ra các *khối được chèn (inserted*

block) có kiểu đối tượng là "INSERT". Khi tạo lại các định nghĩa khối, các khối được chèn tương ứng sẽ thay đổi theo.

Vì một định nghĩa khối bao gồm một *record* chính và nhiều *record* con, nên ta phải sử dụng liên tiếp nhiều biểu thức **Entmake** để tạo một định nghĩa khối mới.

Thứ tự tạo các *record*:
- *Record* chính có kiểu đối tượng là "BLOCK".
- Các *record* mô tả các đối tượng thành phần.
- *Record* ENDBLK.

Nếu không tạo theo đúng trình tự này, **AutoLISP** sẽ báo lỗi.

Ví dụ:

Tạo một định nghĩa khối có tên là "SQR" gồm một đa tuyến 4 đỉnh.

```
(entmake '((0 . "BLOCK") (2 . "SQR") (70 . 64) (10 0 0 0)))
(entmake '((0 . "POLYLINE")(70 . 1)))
(entmake '((0 . "VERTEX")(10 0 0 0)))
(entmake '((0 . "VERTEX")(10 20 0 0)))
(entmake '((0 . "VERTEX")(10 20 20 0)))
(entmake '((0 . "VERTEX")(10 0 20 0)))
(entmake '((0 . "SEQEND")))
(entmake '((0 . "ENDBLK")))
```

Field (70 . 64) bắt buộc phải có. Mã DXF 70 : loại block (*block type flag)*. Đối với các block tự tạo, giá trị này bằng 64.

Lúc này, ta có thể chèn khối này vào bản vẽ bằng hàm **Entmake**. Kiểu đối tượng là "INSERT". Điểm chèn là 2,2,0. Các giá trị khác như các hệ số chèn, góc quay được gán giá trị mặc định tương ứng là 1, 1, 1, 0.

```
(entmake '((0 . "INSERT")(2 . "SQR") (10 2 2 0)))
```

Chú ý:

Khi tạo một định nghĩa khối, ta cần kiểm tra khối đã có hay chưa, vì có thể ta sẽ gây ra lỗi định nghĩa lại khối có sẵn.

Ví dụ: Định nghĩa lại khối "SQR" ở ví dụ trên.

Command: **(entmake '((0 . "BLOCK")(2 . "SQR")(70 . 66)(10 0.0 0.0 0.0)))**↵
Command: **(entmake '((0 . "CIRCLE")(10 0 0 0)(40 . 0.5)))** ↵
Command: **(entmake '((0 . "ENDBLK")))** ↵

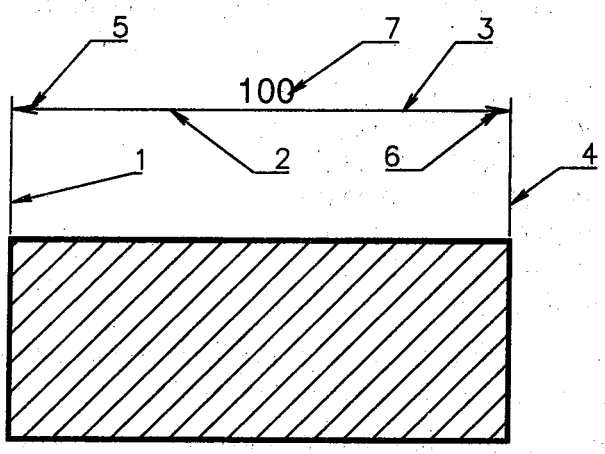
Lúc này các khối đã chèn vẫn không được cập nhật lại trên màn hình. Để thấy sự thay đổi này ta phải sử dụng lệnh **Regen**.

Tạo đối tượng kích thước (DIMENSION)

Đối tượng kích thước là kiểu *khối không tên (unnamed block)* có loại block (*block type flag)* là 1: mã DXF (70 . 1). Tuy nhiên, các đối tượng thành phần của kích thước phải được tạo ra riêng biệt bằng hàm **Entmake**.

Ví dụ: Tạo kích thước gồm có các thành phần như sau:

1. Đường gióng thứ nhất.
2. Nửa đường kích thước bên trái
3. Nửa đường kích thước bên phải
4. Đường gióng thứ hai
5. Mũi tên bên trái
6. Mũi tên bên phải
7. Chữ số kích thước



Command: (entmake '((0 . "BLOCK")(2 . "*Uxx")(70 . 1)(10 0.0 0.0 0.0))) ↵
Command: (entmake '((0 . "LINE")(10 0 0.0625 0)(11 0 2 0))) ↵ ; 1
Command: (entmake '((0 . "LINE")(10 0 1.875 0)(11 1.75 1.875 0))) ↵ ; 2
Command: (entmake '((0 . "LINE")(10 2.25 1.875 0)(11 4 1.875 0))) ↵ ; 3
Command: (entmake '((0 . "LINE")(10 4 0.0625 0)(11 4 2 0))) ↵ ; 4
Command: (entmake '((0 . "SOLID")(10 0 1.875 0)(11 0.12 1.905 0)
(12 0.12 1.845 0)(13 0.12 1.845))) ↵ ; 5
Command: (entmake '((0 . "SOLID")(10 4 1.875 0)(11 3.88 1.905 0)
(12 3.88 1.845 0)(13 3.88 1.845 0))) ↵ ; 6
Command: (entmake '((0 . "TEXT")(10 1.84 1.94 0)(40 . 0.12)(1 . "4.00")
(50 . 0.0)(41 . 1.0)(51 . 0.0)(71 . 0)(72 . 1)(11 2 1.875)(73 . 2))) ; 7
Command: (entmake '((0 . "POINT")(8 . "DEFPOINTS")(10 0 0 0)(50 . 0.0)))

Command: (entmake '((0 . "POINT")(8 . "DEFPOINTS")(10 4 0 0)(50 . 0.0)))
Command: (entmake '((0 . "POINT")(8 . "DEFPOINTS")(10 4 1.875 0)
(50 . 0.0))) ↵
Command: (entmake '((0 . "ENDBLK"))) ↵

Sau khi tạo định nghĩa block kích thước "*Uxx", ta có thể tạo ra các đối tượng kích thước bằng hàm **Entmake**.

Command: (entmake '((0 . "DIMENSION")(2 . "*U0")(10 4 1.875 0)
(11 2 1.875 0)(12 0 0 0)(13 0 0)(14 4 0 0)(40 . 0)(50 . 0)(52 . 0)
(53 . 0)(1 . "")(51 . 0)(70 . 0))) ↵

Chương trình mẫu

```
;Tên file: AutoPL.LSP
;
;Mục đích : Minh họa cách sử dụng hàm ENTMAKE để tạo một đa
;           tuyến.
;
(entmake '((0 . "POLYLINE")))
(setq VP (getpoint "\nFrom point: "))
(entmake (list (cons 0 "VERTEX")(cons 10 VP)))
(while (setq VP (getpoint VP "\nTo point or [ENTER] if done: "))
  (entmake (list (cons 0 "VERTEX")(cons 10 VP)))
)
(entmake '((0 . "SEQEND")))
```

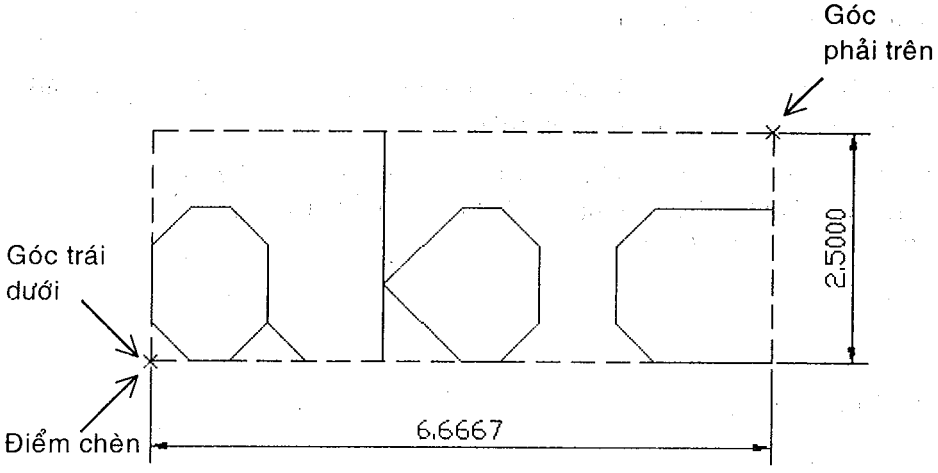
13.5 Hàm TEXTBOX

Hàm **Textbox** chỉ sử dụng riêng cho đối tượng là dòng chữ hoặc là thuộc tính của khối. Hàm này trả về tọa độ 2 đỉnh (góc trái dưới và góc phải trên) của hình chữ nhật bao dòng chữ.

(**Textbox** ELIST)

Tham số ELIST là danh sách phù hợp với đối tượng dòng chữ hoặc thuộc tính của khối. Để sử dụng hàm này với thuộc tính khối thì thuộc tính khối phải được lấy ra bằng hàm **Nentsel** hoặc **Nentselp**.

Ví dụ: Dòng chữ "abc"



Command: **Text** ↵
Justify/Style/<Start point>: **20,20** ↵
Height <0.2000>: **2.5** ↵
Rotation angle <0>: **0** ↵
Text: **abc** ↵

Sử dụng hàm **Entget** và **Entlast** để lấy ra record đối tượng:

Command: **(entget (entlast))** ↵
((-1 . <Entity name: 34c0520>) (0 . "TEXT") (5 . "4C") (100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbText") (10 20.0 20.0 0.0) (40 . 2.5) (1 . "abc") (50 . 0.0) (41 . 1.0) (51 . 0.0) (7 . "STANDARD") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0 1.0) (100 . "AcDbText") (73 . 0))

Sử dụng hàm **Textbox** để lấy ra tọa độ khung chữ nhật bao:

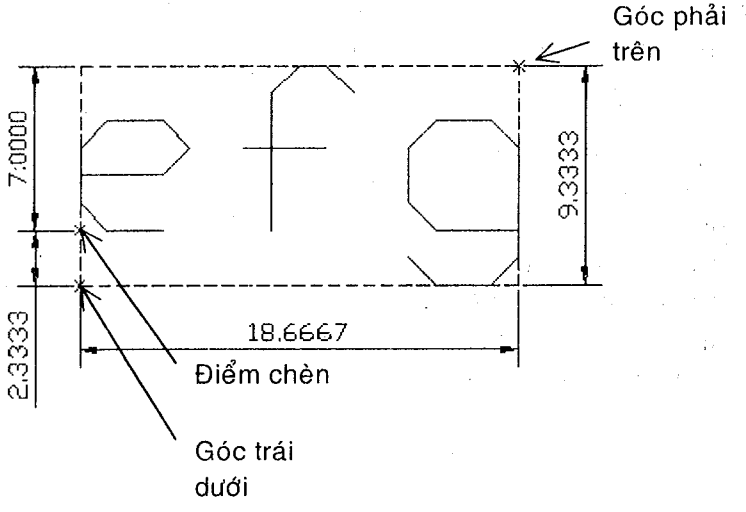
Command: (**textbox** (entget (entlast))) ↵
((0.0 0.0 0.0) (6.66667 2.5 0.0))

Danh sách tọa độ thứ nhất (0.0 0.0 0.0) là khoảng cách X, Y, Z giữa góc trái dưới với điểm chèn của dòng chữ. Danh sách tọa độ thứ hai (6.66667 2.5 0.0) cho biết chiều dài dòng chữ là 6.66667, chiều cao dòng chữ là 2.5. Trong ví dụ này, điểm chèn và góc trái dưới trùng nhau.

Ví dụ: Dòng chữ "efg"

Command: **Text** ↵
Justify/Style/<Start point>: **20,10** ↵
Height <2.5000>: **7** ↵
Rotation angle <0>: **0** ↵
Text: **egf** ↵

Command: **(textbox (entget (entlast)))** ↵
((0.0 -2.33333 0.0) (18.6667 7.0 0.0))



Trong ví dụ này, góc trái dưới thấp hơn điểm chèn. Chiều cao chữ tính từ điểm chèn đến đỉnh trên của chữ.

Tóm tắt

1. Hàm **Entmake** tạo đối tượng trực tiếp trong cơ sở dữ liệu của bản vẽ.
2. Nhiều *field* trong *record* đối tượng có thể bỏ qua. Khi đó chúng được gán các giá trị mặc định. Nếu bỏ qua các *field* bắt buộc, hàm sẽ tạo ra lỗi.
3. Để tạo các đối tượng phức phải sử dụng liên tiếp nhiều hàm **Entmake**
4. Ta có thể tạo *record* mới bằng cách lấy ra *record* có sẵn và thay đổi các giá trị mới bằng hàm **Subst**.

13.6 Các ví dụ mẫu

Ví dụ 1:

Chương trình thay đổi chiều rộng của đa tuyến. Người sử dụng nhập vào chiều rộng và chọn các đa tuyến cần thay đổi.

```
;Tên file: CLW.LSP
;
(defun C:CLW ()
  (setvar "cmdecho" 0)
  (setq LW (getdist "\nEnter new line width:  "))
  (setq A 1)
  (prompt "\nSelect Polylines to change. " )
  (while (/= A nil)
    (progn
      (setq A (entsel))
      (if (/= A nil)
        (progn
          (setq B (entget (car A)))
          (setq C (cdr (assoc 40 B)))
          (princ C)
          (command "pedit" A "W" LW "")
        )
      )
    )
  )
  (princ)
)
;Kết thúc chương trình
```

Ví dụ 2:

Chương trình kết nối các đối tượng (đường thẳng, cung tròn, đa tuyến) có các điểm chung thành đa tuyến.

```
;Tên file: PJ.LSP
(defun C:PJ (/ SS A B)
  (princ "\nSelect Lines/Arcs/Plines to be converted...")
  (setq SS (ssget '((-4 . "<OR")
    (0 . "POLYLINE")(0 . "LINE")(0 . "ARC")(-4 . "OR>")))
  )
```

```
(while (setq A (entsel "Pick one Line/PLine/Arc for each group..."))
  (setq B (cdr (assoc 0 (entget (car A)))))
  (if (member B '("LINE" "ARC" "POLYLINE"))
    (progn
      (command ".pedit" A)
      (if (member B '("LINE" "ARC"))
        (command "Y")
      )
      (command "J" SS "" "")
      (redraw (entlast) 3)
    )
  )
)
(redraw)
(princ)
)
;Kết thúc chương trình
```

13.7 Bài tập

1. Tạo một bản vẽ mới. Vẽ một đường thẳng từ điểm 2,2 đến điểm 40,40. Sử dụng hàm **Subst** và **Entmod** thay đổi điểm đầu của đường thẳng (2,2) thành (2,4). Sử dụng mã DXF 10.
2. Viết chương trình CHCIR.LSP cho phép người sử dụng thay đổi vị trí của đường tròn bằng cách chọn tâm đường tròn mới.
3. Viết chương trình COLINE.LSP, sử dụng hàm **Entmake** trong một vòng lặp liên tiếp tạo các đường thẳng có các màu khác nhau. Đường thẳng đầu tiên có màu là 1. Các màu tiếp theo tăng dần đến 16 thì quay về lại 1.
4. Viết chương trình LAYPICK.LSP tạo một hàm C:LP. Chương trình yêu cầu chọn một đối tượng, và tự động gán lớp bản vẽ của đối tượng đó thành lớp hiện hành.
5. Viết chương trình 16COLINE.LSP tạo một hàm C:CL. Chương trình yêu cầu người sử dụng chọn 2 điểm trên màn hình, sau đó sử dụng hàm **Entmake** để tạo đường thẳng đi qua 2 điểm này bao gồm 16 phân đoạn đường thẳng ghép lại. Đường thẳng đầu tiên có màu số 1, đường thẳng thứ hai có màu số 2... Như vậy người sử dụng vẽ được một đường thẳng 16 màu qua 2 điểm chọn trên màn hình.

6. Viết chương trình ENTCOUNT.LSP sử dụng vòng lặp và hàm **Entnext** để đếm tất cả các đối tượng chính và đối tượng con trong bản vẽ và thông báo kết quả cho người sử dụng.
7. Viết chương trình ENTLIST.LSP tạo một hàm C:ELS, yêu cầu người sử dụng chọn một đường thẳng, cung tròn hoặc đường tròn, sau đó tùy vào đối tượng chọn mà trả về kết quả trên màn hình như sau:

LINE	***ARC***	***CIRCLE***
Entity Type	*Entity Type*	*Entity Type*
Layer	*Layer*	*Layer*
Start point	*Start point*	*Start point*
End point	*End point*	*End point*

8. Viết chương trình P-LISP.LSP yêu cầu người sử dụng chọn một đa tuyến và trả về danh sách chứa tọa độ tất cả các đỉnh.
9. Viết chương trình TEXTFILL.LSP thực hiện các việc sau:
   a. Yêu cầu người sử dụng nhập vào một dòng chữ.
   b. Vẽ một hình chữ nhật quanh dòng chữ sao cho khoảng cách các cạnh so với khung chữ nhật bao là 1 đơn vị.
   c. Vẽ một hình chữ nhật tạm ngay tại vị trí khung chữ nhật bao dòng chữ, sau đó vẽ mặt cắt giữa 2 hình chữ nhật này.
   d. Xóa hình chữ nhật tạm.

13.8 Lời giải

1.

```
(setq EL (entget (entnext)))
(setq EL (subst  '(10 20.0 20.0 0.0) (assoc 10 EL) EL))
(entmod EL)
```

2.

```
;Tên file: CHCIR.LSP
;
(defun C:CHCIR (/ ENAM EREC)
  (while
    (null (setq ENAM (car (entsel "\nPick a circle: "))))
    (princ "\nYou missed.")
  )
  (setq EREC (entget ENAM))
  (if (= "CIRCLE" (cdr (assoc 0 EREC)) )
```

```
        (progn
           (initget 1)
          (setq EREC (subst  (cons 10 (getpoint "\nNew location: "))
                             (assoc 10 EREC) EREC)
           )
          (entmod EREC)
         );Đóng progn
        (princ "\nNot a circle")
  );Đóng if
;
  (princ)
);Đóng defun
;Kết thúc chương trình
```

3.

```
;Tên file: COLINE.LSP
;
(defun C:COLINE (/ VP1 VP2 COL)
  (setq VP1 (getpoint "\nFrom point: ")
        COL 1
  )
  (while (setq VP2 (getpoint VP1 "\nTo point or [ENTER] when done: "))
    (entmake (list '(0 . "LINE") (cons 10 VP1)(cons 11 VP2) (cons 62 COL)))
    (setq VP1 VP2
          COL (if (= COL 15) 1 (1+ COL))
    )
  );Đóng while
);Đóng defun
;Kết thúc chương trình
```

4.

```
;Tên file: LAYPICK.LSP
;
(defun C:LP (/ EN LAY)
  (setq EN (entget (car (entsel "\nPick a polyline: ")))
        LAY (cdr (assoc 8 EN))
  )
  (setvar "CLAYER" LAY)
  (princ)
); Đóng defun
;Kết thúc chương trình
```

5.

```
;Tên file: 16COLINE.LSP
;
(defun C:CL (/ LST PT1 PT2 PT DIST ANG COL LST)
  (setq  PT1 (getpoint "\nFirst point: ")
         PT2 (getpoint PT1 "\nSecond point: ")
          DIST (/ (distance PT1 PT2) 16)
         ANG (angle PT1 PT2)
  )
;
  (setq PT PT1
       COL 1
  )
  (repeat 16
        (setq PT (polar PT ANG DIST)
          LST (append LST (list PT))
         );Đóng setq
  );Đóng repeat
  (foreach PT LST
    (entmake (list '(0 . "LINE")(cons 10 PT1)(cons 11 PT)(cons 62 COL)))
    (setq PT1 PT
          COL (1+ COL)
    )
  );Đóng foreach
(princ)
); Đóng defun
;Kết thúc chương trình
```

6.

```
;Tên file: ENTCOUNT.LSP
;
(if (setq E (entnext))
  (progn
    (setq CNT 1)
    (while (setq E (entnext E))
       (setq CNT (1+ CNT))
    )
    (princ (strcat "\nThere are " (itoa CNT) " entities in database."))
  );Đóng progn
  (princ "\nNo entities found in database!")
);Đóng if
```

```
(princ)
;Kết thúc chương trình
```

7.

```
;Tên file : ENTLIST.LSP
;
(defun C:ELS (/ ENT)
   (setq ENT (entget (car (entsel "\nSelect a line, arc or circle: "))))
   (cond
     ((= "LINE" (cdr (assoc 0 ENT)))
      (princ (strcat "\nEntity type: " (cdr (assoc 0 ENT))))
      (princ (strcat "\nLayer:       " (cdr (assoc 8 ENT))))
      (princ "\nStart point: ")(princ (cdr (assoc 10 ENT)))
      (princ "\nEnd point:   ")(princ (cdr (assoc 11 ENT)))
     );Đóng "LINE"
     ((= "ARC" (cdr (assoc 0 ENT)))
      (princ (strcat "\nEntity type: " (cdr (assoc 0 ENT))))
      (princ (strcat "\nLayer:       " (cdr (assoc 8 ENT))))
      (princ "\nCenter point: ")(princ (cdr (assoc 10 ENT)))
      (princ "\nRadius:   ")(princ (cdr (assoc 40 ENT)))
     );Đóng "ARC"
     ((= "CIRCLE" (cdr (assoc 0 ENT)))
      (princ (strcat "\nEntity type: " (cdr (assoc 0 ENT))))
      (princ (strcat "\nLayer:       " (cdr (assoc 8 ENT))))
      (princ "\nCenter point: ")(princ (cdr (assoc 10 ENT)))
      (princ "\nRadius:   ")(princ (cdr (assoc 40 ENT)))
     );Đóng "CIRCLE"
     (T
      (princ "\nNot a line, arc or circle")
     )
   );Đóng cond
   (princ)
);Đóng defun
;Kết thúc chương trình
```

8.

```
;Tên file: P-LIST.LSP
   (setq ENT (entsel "\nPick a polyline: "))
   (while (/= "POLYLINE" (cdr (assoc 0 (entget(car ENT)))))
    (setq ENT (entsel "\nPick a polyline: "))
   )
```

```
  (setq EL '()
        ENT (entnext (car ENT))
  )
  (while
     (/= "SEQEND" (cdr (assoc 0 (entget ENT))))
     (setq EL (append EL (list (cdr (assoc 10  (entget ENT)))))
          ENT (entnext ENT)
     )
  ); Đóng while
  (princ "\nVertex point coordinates: \n")(princ EL)
  (princ)
;Kết thúc chương trình
```

9.

```
;Tên file: TEXTFILL.LSP
;
  (setq STR (getstring "\nEnter a string: ")
        HIG (getreal "\Height: ")
        POS (getpoint "Start point: ")
  )
  (command ".text"  POS HIG 0 STR "")
  (setq TB (textbox (entget (entlast)))
        LC (car TB)
        RC (cadr TB)
       PT1 (list (+ (car POS)(car LC)) (+ (cadr POS)(cadr LC)))
       PT2 (list (+ (car PT1)(car RC)) (+ (cadr PT1)(cadr RC)))
       PT3 (list (- (car PT1) 1) (- (cadr PT1) 1))
      PT4 (list (+ (car PT2) 1) (+ (cadr PT2) 1))
  )
  (command ".rectang" PT1 PT2
           ".rectang" PT3 PT4
  )
  (setq ENT1 (nentselp PT1)
       ENT2 (nentselp PT3)
  )
  (command  ".hatch" "s" ENT1 ENT2 ""   ".erase" ENT1 "" )
;Kết thúc chương trình
```

Chương 14

QUẢN LÝ FILE VÀ MÔI TRƯỜNG LÀM VIỆC

Nội dung chương

1. Tìm kiếm file trong đĩa bằng các hàm của **AutoLISP**.
2. Đọc và ghi dữ liệu vào các file văn bản.
3. Truy xuất các biến môi trường.

Chúng ta đã biết nhiều phương pháp lưu trữ và rút trích dữ liệu. Đó là nhiệm vụ chính của các chương trình **AutoLISP**. Tuy nhiên, với một khối lượng dữ liệu lớn và nhiều kiểu khác nhau, ta cần phải có những phương pháp mạnh mẽ hơn. Đó là đọc (*read*) và chép (*write*) dữ liệu thành các file văn bản ASSCII.

Ngoài ra, dữ liệu lưu trữ trong các file có thể được sử dụng trong các bản vẽ khác. Ta có được phương tiện trao đổi dữ liệu giữa các ứng dụng, và khi cần thiết có thể sửa đổi các file hỗ trợ (*support file*) của **AutoCAD** cho phù hợp.

14.1 Quản lý file

14.1.1 Tìm kiếm file

Một trong những lỗi thường gặp của các chương trình **AutoLISP** là đọc những file không có trong đường dẫn thư mục mong muốn. Lỗi này có thể khắc phục nếu ta sử dụng các hàm tìm kiếm file để kiểm tra có file hay không.

Hàm FINDFILE

Hàm **Findfile** dùng để tìm kiếm file có trong các thư viện hay không. Nếu tìm thấy sẽ trả về đường dẫn thư mục của file này. Nếu không tìm thấy sẽ trả về nil.

(**Findfile** FILENAME)

Tham số FILENAME phải có đủ tên file và phần mở rộng. Nếu không chứa đường dẫn thư mục, hàm sẽ tìm kiếm file trong đường dẫn tìm kiếm thư viện *(library search path)* của **AutoCAD**. Nếu có chứa cả đường dẫn thư mục thì hàm này chỉ tìm kiếm trong thư mục đó mà thôi. Dấu ngăn cách thư mục là \\ hoặc /.

Ví dụ:

Command: **(findfile "ACAD.EXE")** ↵
C:\\Program Files\\AutoCAD R14\\ACAD.EXE
Command: **(findfile "ACAD.LIN")** ↵
C:\\Program Files\\AutoCAD R14\\SUPPORT\\acad.LIN
Command: **(findfile "WIN.EXE")** ↵
nil

Ví dụ: Chương trình kiểm tra có file hay không

```
;Tên file: LL.LSP
(defun C:LL (/ FN)
  (setq FN (getstring "\nAutoLISP filename to load: ")); Gán tên file cho FN
  (if
    (findfile (strcat FN ".LSP"))        ;Nếu tìm thấy FN
    (load FN)                            ;thì tải FN
```

```
    (princ                          ;Ngược lại thì in thông báo lên màn
                                    ;hình
      (strcat "\n" FN ".LSP not found!") )
  ); Đóng if
  (princ)
); Đóng defun
;Kết thúc chương trình
```

14.1.2 Yêu cầu người sử dụng tìm một file

Để yêu cầu người sử dụng nhập vào tên một file, ta có thể dùng hàm **Getstring** và sau đó dùng **Findfile** để kiểm tra tên file này. Ta cũng có thể dùng hàm **Getfiled** làm xuất hiện hộp thoại

Hàm GETFILED

Hàm **Getfiled** làm xuất hiện hộp thoại cho phép chọn một file có sẵn, hoặc chọn đường dẫn cho file mới. Nếu người sử dụng chọn tên file hợp lệ, hàm trả về đầy đủ tên file và đường dẫn. Ngược lại, hàm trả về nil.

(**Getfiled** TITLE DEFAULT EXT FLAGS)

Các tham số bắt buộc phải có. Nếu không muốn sử dụng tham số nào, ta phải gán giá trị rỗng "" cho tham số đó.

Tham số TITLE

Tiêu đề của hộp thoại

Tham số DEFAULT

Tên file mặc định xuất hiện sẵn trong hộp thoại. Nếu không muốn xuất hiện tên file mặc định, ta dùng chuỗi rỗng ("").

Chú ý:

Dùng dấu / hoặc \\ trong đường dẫn thư mục thay cho dấu \.

Tham số EXT

Phần mở rộng của tên file.

Tham số FLAGS

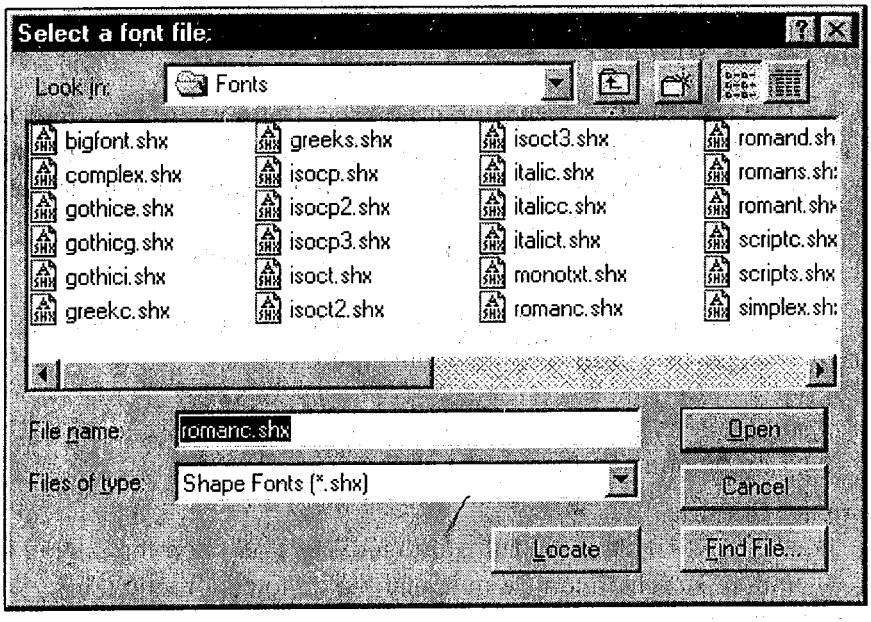
Xác định đặc điểm của hộp thoại. Các mã tạo ra giá trị cho tham số này như sau.

Mã	Ý nghĩa
1	Yêu cầu tạo ra file mới.
4	Cho phép nhập tên file có phần mở rộng khác với kiểu file trong tham số EXT.
8	Cho phép tìm kiếm trong đường dẫn thư viện.

Ví dụ: Yêu cầu người sử dụng chọn một file font chữ.

Command: **(getfiled "Select a font file: " "/Program Files/AutoCAD R14/Fonts/romanc" "shx" 8)**↵

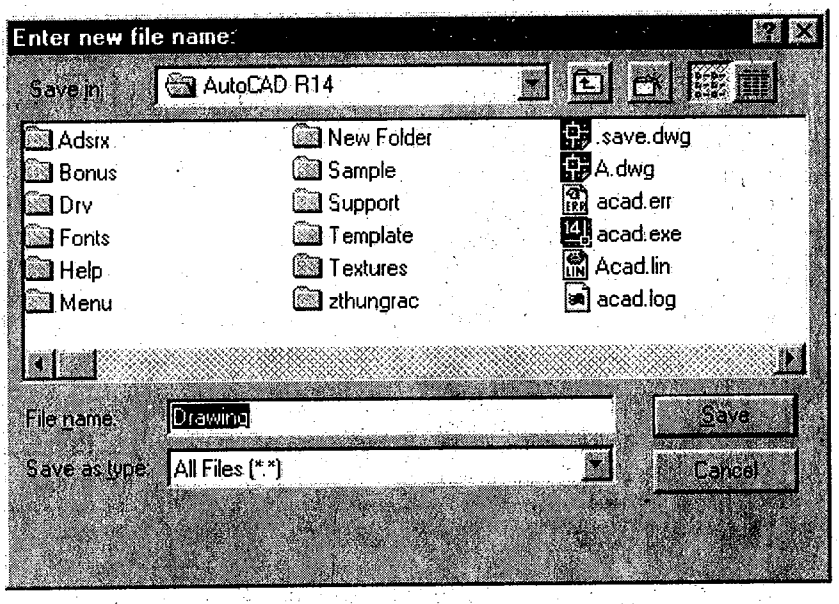
Xuất hiện hộp thoại **Select a font file:** như sau:



Ví dụ: Yêu cầu nhập vào tên file mới sẽ được tạo ra.

Command: **(getfiled "Enter new file name:" "" "" 1)** ↵

Xuất hiện hộp thoại **Enter new file name:** như sau:



Các tham số trong ví dụ trên:

Tham số DEFAULT = "" nên **AutoCAD** sử dụng thư mục mặc định *AutoCAD R14* khi hiện hộp thoại.

Tham số EXT = "" nên kiểu file là *All Files (*.*)*.

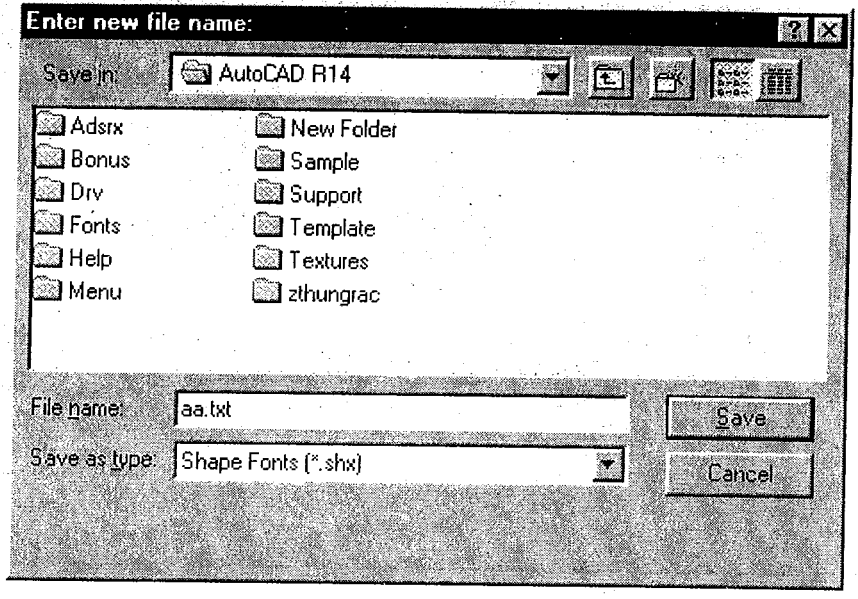
Tham số FLAGS = 1 nên xuất hiện nút *Save* trong hộp thoại.

Ví dụ:

Yêu cầu nhập tên file mới, có tên file mở rộng mặc định là .shx, nhưng người sử dụng có thể nhập tên file mở rộng khác.

Command: **(getfiled "Enter new file name:" "" "shx" 5)** ↵
"C:\\Program Files\\AutoCAD R14\\aa.txt"

Xuất hiện hộp thoại **Enter new file name:** như sau:



Trong ví dụ trên, tham số FLAGS = 5 vì 5 = 1 + 4:
1 – Xuất hiện hộp thoại tạo file mới.
4 – Cho phép nhập phần tên mở rộng bất kỳ.

Ví dụ:

Sử dụng tham số FLAGS = 8. Khi *không* dùng mã 8 cho tham số FLAGS, tên file trả về luôn chứa đường dẫn thư mục.

Command: **(getfiled "Enter file name:" "" "dwg" 0)** ↵
"C:\\Program Files\\AutoCAD R14\\SUPPORT\\ansi_a.dwg"

Khi *dùng* mã 8 cho tham số FLAGS, tên file trả về không chứa đường dẫn thư mục.

Command: **(getfiled "Enter file name:" "" "dwg" 8)** ↵
"ansi_a.dwg"

14.1.3 Đọc dữ liệu từ file

Quá trình đọc dữ liệu từ file gồm 3 bước:
- Mở file (hàm **Open**).
- Đọc dữ liệu từ file (hàm **Read-line**).
- Đóng file (hàm **Close**).

Hàm OPEN

Để đọc dữ liệu từ file, trước tiên ta phải mở file bằng hàm **Open**.

(**Open** FILENAME MODE)

Tham số MODE là một ký tự viết *thường* "r", "w" hoặc "a" xác định chế độ mở file.

"r" Mở file FILENAME để đọc dữ liệu. *Con trỏ file (file pointer)* nằm ở tại dòng đầu tiên của file. Nếu không tìm thấy file này, hàm trả về nil.

"w" Mở file FILENAME để chép dữ liệu vào file này. Nếu tìm thấy, file này sẽ được mở ra, dữ liệu có sẵn trong file sẽ bị chép đè lên. Nếu không tìm thấy, file mới sẽ được tạo ra. Con trỏ file nằm ở vị trí đầu file.

"a" Mở file FILENAME để chép dữ liệu nối tiếp vào phía cuối file. Nếu tìm thấy file, con trỏ file nằm tại vị trí cuối file, dữ liệu sẽ được chép nối vào cuối file. Nếu không tìm thấy, file mới sẽ được tạo ra, con trỏ file nằm ở vị trí đầu file.

Sau khi được mở ra, mỗi file sẽ được cấp một *thẻ file (file descriptor)*. Thẻ file này phải được gán ngay cho một biến. Biến này sẽ đại diện cho file trong quá trình đọc ghi dữ liệu.

Ví dụ:

Command: **(setq PR (open "C:/program files/autocad r14/ support/ acadr14. lsp" "r"))** ↵

<File: #2db23c2> ;Biến PR lưu trữ thẻ file

Số lượng các file mở cùng một lúc do hệ điều hành quy định trong file *CONFIG.SYS* bằng câu lệnh "FILES = nn".

Hàm READ-LINE

Hàm **Read-line** dùng để đọc dữ liệu của một *file văn bản ASSCII* có thẻ file là FILE-DESC.

(**Read-line** [FILE-DESC])

Nếu không có tham số FILE-DESC, hàm sẽ đọc chuỗi dữ liệu từ bàn phím. Hàm **Read-line** bắt đầu từ dòng đầu tiên của file văn bản. Sau đó con trỏ file chuyển qua dòng thứ hai. Nếu dùng hàm **Read-line** lần nữa thì dòng thứ hai được lấy ra, con trỏ file chuyển qua dòng thứ ba... Khi không còn dòng nào nữa thì hàm trả về nil.

Ví dụ:

Command: **(setq PR (open "C:/program files/autocad r14/ support/ acadr14. lsp" "r"))** ↵

<File: #2db23c2>

Command: **(read-line PR)** ↵

"; Next available MSG number is 86"

Command: **(read-line PR)** ↵

"; MODULE_ID ACADR13_LSP_"

Command: **(read-line PR)** ↵

";;; ACADR14.LSP Version 14.1 for Release 14 "

Kết hợp với vòng lặp **While** ta có thể đọc toàn bộ file văn bản:

```
(while (setq RL (read-line PR))
  (princ RL)
)
```

Hàm CLOSE

Hàm **Close** dùng để đóng file đang mở có thẻ file là FILE-DESC. Hàm này luôn trả về nil.

(**Close** FILE-DESC)

Ta phải đóng tất cả các file đang mở trước khi thoát **AutoCAD**. Nếu không, các file này có thể sẽ bị thất lạc vì hệ điều hành không thể quản lý chúng được.

Ví dụ:

Command: **(close PR)** ↵

nil

Ví dụ:

Dùng các hàm **Open**, **Read-line**, **Close** và vòng lặp **While** để in nội dung một file lên màn hình. Dùng các hàm **Getfiled** và **Findfile** để kiểm tra tên file trước khi mở file.

```
;Tên file: TYPE-IT.LSP
;
```

```
(defun C:TYP ()
  (if (not
       (setq FN (getfiled "Indicate file to read: " "" "" 10)        ;Chọn tên file
             FN (if( null FN) FN (findfile FN));Bổ sung đường dẫn thư mục
       )
      )
  (princ "\nFunction cancelled.")
  (progn
    (command ".textscr")
    (setq PR (open FN "r"))
    (while (setq RL (read-line PR))
       (princ RL)
       (princ "\n")
    )
    (close PR)
  ); Đóng progn
 ); Đóng if
 (princ)
); Đóng defun
;Kết thúc chương trình
```

14.1.4 Chép dữ liệu ra file

Khi cần chép dữ liệu ra file, ta mở file ở chế độ "w" hoặc "a", rồi dùng hàm **Write-line**.

Hàm WRITE-LINE

Hàm **Write-line** dùng để chép chuỗi STRING ra một *file văn bản ASSCII* có thể file là FILE-DESC.

(**Write-line** STRING [FILE-DESC])

Nếu không có tham số FILE-DESC, chuỗi sẽ được in ra dòng nhắc lệnh. Tương tự như hàm **Read-line**, hàm **Write-line** sẽ dịch chuyển con trỏ file sang dòng kế tiếp. Sau khi chép xong dữ liệu, file phải được đóng lại bằng hàm **Close**.

Ví dụ:

```
(setq PW (open "Sample.txt" "w"))                ;Mở file theo chế độ "w"
```

```
(write-line "OK" PW)          ;Chép chuỗi "OK" ra file
(close PW)                    ;Đóng file lại.
```

Hãy dùng hàm **Typ** tạo ra trong chương trình TYPE-IT.LSP ở trên để xem nội dung file "Sample.txt".

Ví dụ:

Mở lại file "Sample.txt" để điền thêm nội dung vào cuối file.

```
(setq PA (open "Sample.txt" "a"))   ;Mở file theo chế độ "a"
(write-line "YES" PA)               ;Chép chuỗi "YES" vào cuối file
(close PA)                          ;Đóng file lại.
```

Hãy dùng hàm **Typ** tạo ra trong chương trình TYPE-IT.LSP ở trên để xem lại nội dung file "Sample.txt".

14.1.5 Chép dữ liệu vào vị trí giữa file

Ta có thể phối hợp các hàm đã biết để chép thêm một dòng mới vào giữa file. Các bước như sau:

1. File ban đầu gọi là file nguồn. Tạo ra một file đích.
2. Chép vào file đích các dòng từ đầu file nguồn cho đến vị trí muốn chèn dòng mới.
3. Chép các dòng mới vào file đích.
4. Chép phần còn lại của file nguồn vào file đích.

Ví dụ:

```
;
;Tạo file nguồn "Source.txt" chứa các số từ 1 đến 10
(setq PR (open "Source.txt" "w"))
(setq CNT 1)
(repeat 10
  (write-line (itoa CNT) PR)
  (setq CNT (1+ CNT))
)
(close PR)
;
;Mở lại file "Source.txt" và tạo file đích "Target.txt"
(setq PR (open "Source.txt" "r"))
(setq PW (open "Target.txt" "w"))
; Chép các dòng từ file "Source.txt" sang file "Target.txt" cho đến khi gặp
```

```
;dòng chứa số 6 thì dừng.
(setq RL (read-line PR))
(while (/= (substr (strcase RL) 1 1) "6")
  (write-line RL PW)
  (setq RL (read-line PR))
)
; Chép tiếp các dòng chứa các số từ 51 đến 55 vào file "Target.txt"
(setq CNT 1)
(repeat 5
  (write-line (itoa (+ 50 CNT)) PW )
  (setq CNT (1+ CNT))
)
;Chép các dòng còn lại từ file "Source.txt" sang file "Target.txt"
(write-line RL PW)
(while (setq RL (read-line PR))
  (write-line RL PW)
)
;Đóng các file lại. Sử dụng hàm MAPCAR để nhóm các file cần đóng vào
;một danh sách để không bị sót.
(mapcar 'close (list PR PW))
;Kết thúc chương trình
```

Nội dung các file:

"Source.txt"	**"Target.txt"**
1	*1*
2	*2*
3	*3*
4	*4*
5	*5*
6	*51*
7	*52*
8	*53*
9	*54*
10	*55*
	6
	7
	8
	9
	10

Mặc dù ta có thể đọc và chép lại trên cùng một file, nhưng tốt nhất ta nên dùng 2 file đích và nguồn khác nhau. Nhờ đó, nếu quá trình chép dữ liệu bị lỗi và file bị đóng lại, dữ liệu ban đầu trong file nguồn vẫn còn.

Ví dụ:

Chương trình sau đây thêm một dòng định nghĩa dạng đường *HIDDEN4 vào file *ACAD.LIN*. Giả sử file này đang có tại thư mục hiện hành. Chú ý: Để an toàn, ta nên sao lưu lại file ACAD.LIN trước khi thực hiện chương trình này.

```
;Sao chép file ACAD.LIN thành file ACADSAV.LIN
;File nguồn: ACADSAV.LIN
;File đích: ACAD.LIN
(command "SHELL" "RENAME ACAD.LIN ACADSAV.LIN")
(setq PR (open "ACADSAV.LIN" "r"))
(setq PW (open "ACAD.LIN" "w"))
;
(setq RL (read-line PR))                    ;Đọc dòng đầu tiên
(while (/= (substr (strcase RL) 1 7) "*HIDDEN");Nếu chưa đến dòng định
  (write-line RL PW)                        ; nghĩa *HIDDEN thì chép các
  (setq RL (read-line PR))                  ; dòng sang file đích.
); Đóng while
;
(while                                      ;Chép tiếp các dòng
 (or                                        ; *HIDDEN sang file đích
    (= (substr (strcase RL) 1 7) "*HIDDEN")
    (= (substr RL 1 1) "A")
  )
  (write-line RL PW)
  (setq RL (read-line PR))
); Đóng while
;
;Chép các dòng định nghĩa dạng đường mới *HIDDEN vào file đích.
(write-line "*HIDDEN4,_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _" PW)
(write-line "A,.125,-.125" PW)
;
;
(write-line RL PW)                          ;Chép các dòng còn lại từ file
(while (setq RL (read-line PR))             ; nguồn sang file đích.
  (write-line RL PW)
```

```
)
;
(mapcar 'close (list PR PW))                ;Đóng các file lại.
;Kết thúc chương trình
```

14.1.6 Các hàm READ và WRITE khác

Trong một số trường hợp, ta cần phải đọc hoặc ghi từng ký tự ra file.

Hàm READ-CHAR

Hàm **Read-char** dùng để đọc một ký tự của một *file văn bản ASSCII* có thể file là FILE-DESC và trả về mã ASSCII của ký tự này. Mỗi lần, con trỏ file sẽ tăng lên một vị trí.

(**Read-char** [FILE-DESC])

Nếu không có tham số FILE-DESC, hàm sẽ đọc một ký tự từ bàn phím.

✌ **Ví dụ:**

Command: **(setq PR (open "C:/program files/autocad r14/ support/ acadr14. lsp" "r"))** ↵
<File: #36329a4>
Command: **(setq LTR (read-char PR))** ↵
59 ;Ký tự đầu tiên có mã ASCII là 59
Command: **(chr LTR)** ↵
";" ;Ký tự này là ";"

Command: **(repeat 36 (princ (chr (setq LTR (read-char PR)))))** ↵
Next available MSG number is 86 ;Đọc 36 ký tự còn lại của
"\n" ; dòng đầu tiên

Hàm WRITE-CHAR

Hàm **Write-char** biến một mã ASCII thành một ký tự và chép nó vào một file đang mở. Nếu bỏ qua tham số FILE-DESC, ký tự sẽ được in lên màn hình.

(**Write-char** NUM [FILE-DESC])

Ví dụ:

Command: **(write-char 65)** ↵
A65 ;in ký tự A lên mận hình. Trả về 65.
Command: **(write-char (ascii "A"))** ↵
A65 ;in ký tự A lên màn hình. Trả về 65.

Ví dụ:

Chép bảng chữ cái ABC ra một file ALPHA.BET

```
(setq PW (open "ALPHA.BET" "w")   ;Mở file
      N 65                        ;Gán cho N mã ASCII của ký tự "A"
)
(repeat 26                        ;Duyệt từng ký tự
  (write-char N PW)               ;Chép ký tự ra file
  (setq N (1+ N))                 ;Tăng N
)                                 ; Đóng REPEAT
(close PW)                        ;Đóng file
```

Hãy dùng hàm **Typ** trong file TYPE-IT.LSP để xem nội dung file này.

14.2 Lưu trữ dữ liệu của ứng dụng trong file ACAD14.CFG

Dữ liệu trong file cấu hình *ACAD14.CFG* dùng để thiết lập môi trường làm việc cho **AutoCAD** như màn hình, máy in, tên file bản vẽ mặc định... Mỗi khi khởi động, **AutoCAD** đọc nội dung file này để thiết lập môi trường làm việc phù hợp. Nếu trong quá trình làm việc, các thông số của môi trường bị thay đổi, chúng sẽ được lưu lại trong file này.

Nội dung file này chia thành nhiều phần (*section*), mỗi phần lưu trữ dữ liệu của mỗi công việc. Mỗi *section* bắt đầu bằng một tên (*section name*) chứa trong cặp dấu ngoặc vuông. Mỗi dòng trong một *section* chứa giá trị của một tham số trong chương trình.

Ví dụ: Sau đây là một đoạn trích trong file ACAD.CFG.

```
[AppData/DriverCustom]
fDither=1
```

fUseDefaultPrn=1
fDoSmartFills=1
fUpdatePens=1

dmColor=monochrome
dmOrientation=portrait
dmPaperSize=1
dmPaperLength=2794
dmPaperWidth=2159
dmScale=100
dmCopies=1
dmPrintQuality=200
dmYResolution=200
dmTTOption=2
dmDeviceName=PaperPort
dmDefaultSource=1
dmSpecVersion=778
dmDriverVersion=512
dmDriverExtra=0
dmFields=7f1f
dmDuplex=1

[Version]
Platform=Microsoft Windows
FileVersion=8
ExecutableDate&Time=97/05/06 03:01:20

[AutoCAD]
CfgStamp=FED1")W.
PlottersConfigured=6
AuthorizationCode=ZC+?PW:P
. . . . . .

Chúng ta có thể chép dữ liệu của chương trình do chúng ta viết vào file này. Các dữ liệu này mang tính chất quan trọng, dùng để thiết lập môi trường làm việc như tên người sử dụng, số lần tối đa chương trình được thực hiện, vị trí xuất hiện các hộp thoại tự tạo ...

Ta có thể sử dụng các hàm đọc ghi file đã biết để truy xuất dữ liệu trong file này. Tuy nhiên, vì tính chất quan trọng của file này, nên **AutoLISP** cung cấp 2 hàm đọc ghi riêng cho file này.

Hàm SETCFG

Hàm **Setcfg** *(SET ConFiGuration)* chép dữ liệu vào *section* AppData của file ACAD14.CFG. File này không cần phải mở ra bằng hàm **Open**.

(**Setcfg** CFGNAME CFGVAL)

Tham số CFGNAME chứa tên *section* và tên tham số tương ứng được gán dữ liệu chứa trong CFGVAL.

Tham số CFGVAL phải có kiểu chuỗi. Tham số CFGNAME có dạng như sau:

"AppData/application_name/section_name/. . ./parameter_name"

Hàm **Setcfg** lưu giá trị vào *section* tương ứng. Nếu *section* này chưa có thì nó sẽ được tạo ra.

Ví dụ:

Chương trình MakeStruct lưu trữ tên công ty, tên công trình, vật liệu, số nút trong file ACAD4.CFG.

Command: **(setcfg "AppData/MakeStruct/Name/StructName" "XYZ")** ↵
"XYZ"
Command: **(setcfg "AppData/MakeStruct/Name/UserName" "Nguyen Van A")** ↵
"Nguyen Van A"
Command: **(setcfg "AppData/MakeStruct/Material/Material" "Steel")** ↵
"Steel"
Command: **(setcfg "AppData/MakeStruct/Material/Nodes" "50")** ↵
"50"

Thoát khỏi chương trình **AutoCAD** để file ACAD14.CFG được cập nhật dữ liệu. Sau đó, xem nội dung file ACAD14.CFG, ta sẽ thấy 2 *section* mới tạo ra như sau:

[AppData/MakeStruct/Name]
StructName=XYZ

UserName=Nguyen Van A

[AppData/MakeStruct/Material]
Material=Steel
Nodes=50

Ví dụ: Chương trình chép dữ liệu vào file ACAD14.CFG.

```
;Tên file: SETDATA.LSP
;
(defun C:SETDATA (/ AN SN PN CV)
  (setvar "cmdecho" 0)
  (while (/= "" (setq AN
                  (getstring T "\nApplication Name [ENTER to exit]: ")))
    (setq SN (getstring T "\nSection Name: ")
          PN (getstring T "\nParameter Name: ")
          CV (getstring T "\nConfiguration Value: ")
    )
    (setcfg (strcat "AppData/" AN "/" SN "/" PN) CV)
  ); Đóng while
  (princ)
); Đóng defun
;Kết thúc chương trình
```

Thực hiện chương trình:

Command: **SETDATA** ↵
Application Name [ENTER to exit]: **MakeStruct** ↵
Section Name: **Name** ↵
Parameter Name: **CompanyName** ↵
Configuration Value: **ABC Co.Ltd** ↵
Application Name [ENTER to exit]: ↵
Command:

Thoát khỏi chương trình **AutoCAD** để file ACAD14.CFG được cập nhật dữ liệu. Sau đó, xem nội dung file ACAD14.CFG, ta sẽ thấy 1 dòng dữ liệu mới được thêm vào:

[AppData/MakeStruct/Name]
StructName=XYZ
UserName=Nguyen Van A

CompanyName=ABC Co.Ltd

[AppData/MakeStruct/Material]
Material=Steel
Nodes=50

Hàm GETCFG

Hàm **Getcfg** *(GET ConFiGuration)* đọc dữ liệu từ *section* AppData của file acad14.cfg. File này không cần phải mở ra bằng hàm **Open**.

(**Getcfg** CFGNAME)

Tham số CFGNAME chứa tên *section* và tên tham số tương ứng chứa dữ liệu cần lấy ra. Tham số CFGNAME có dạng như sau:

"AppData/application_name/section_name/. . ./parameter_name"

Ví dụ: Lấy thông tin đã ghi vào file ACAD14.CFG.

Command: **(getcfg "AppData/MakeStruct/Name/StructName")** ↵
"XYZ"
Command: **(getcfg "AppData/MakeStruct/Name/UserName")** ↵
"Nguyen Van A"

14.3 Truy xuất thông tin về môi trường làm việc

Môi trường làm việc có thể ảnh hưởng đến chương trình chúng ta viết. Đặc biệt là phiên bản **AutoLISP** đang sử dụng. Nếu chương trình sử dụng các hàm **AutoLISP** dành riêng cho release **14** hoặc **2000**, thì khi chạy trên các phiên bản trước đó nó sẽ gây ra lỗi.

Hàm VER

Hàm **Ver** *(VERsion)* cho biết phiên bản **AutoLISP** đang sử dụng. Hàm này không có tham số:

(**Ver**)

Ví dụ:
Command: **(ver)** ↵
"AutoLISP Release 14.0"

Các biến môi trường

Đặc điểm của môi trường làm việc chứa trong các *biến môi trường (environmental variable)*. Các biến này do hệ điều hành quản lý và gán giá trị. Chúng ta có thể xem chi tiết về các biến này trong phần *Help* của **AutoCAD**, mục *Installation Guide*. Ví dụ các biến: ACAD, ACADMAXMEM, ACADPAGEDIR, ACADCFG . . .

Chú ý:

Tên các biến môi trường có phân biệt dạng chữ hoa chữ thường.

Hàm GETENV

Hàm **Getenv** *(GET ENvironmental Variable)* trả về giá trị của biến VARIABLE-NAME.

(**Getenv** VARIABLE-NAME)

Ví dụ:

Command: **(getenv "ACADCFG")** ↵ ;File cấu hình của **AutoCAD**
"C:\\PROGRAM FILES\\AUTOCAD R14"

Command: **(getenv "ACAD")** ↵ ;Đường dẫn thư viện
"C:\\Program Files\\AutoCAD R14\\SUPPORT; C:\\PROGRAM FILES\\ AutoCAD R14\\fonts; C:\\PROGRAM FILES\\AUTOCAD R14\\help"

Command: **(getenv "ACADMAXMEM")** ↵
nil ;Không được gán giá trị

Tóm tắt

1. Hàm **Findfile** tìm một file có trong đường dẫn thư mục hay không. Vì hàm này trả về *nil* nếu không tìm thấy file, nên có thể dùng làm biểu thức điều kiện.
2. Để thực hiện việc đọc ghi dữ liệu ra file, file này phải được mở ra bằng hàm **Open**. Tham số MODE xác định chế độ mở file: "r" để đọc, "w" để chép, "a" để chép nối đuôi.

3. Hàm **Read-line** đọc từng dòng dữ liệu trong một file đang mở và làm cho con trỏ file chuyển đến dòng tiếp theo.
4. Hàm **Write-line** chép một dòng vào file đang mở và làm cho con trỏ file chuyển đến dòng tiếp theo.
5. Khi đã đọc ghi dữ liệu xong, hàm phải được đóng lại bằng hàm **Close**.
6. Hàm **Read-char** đọc từng ký tự và trả về mã ASCII của ký tự này.
7. Hàm **Write-char** chép một ký tự vào một file đang mở.
8. Hàm **Setcfg** và **Getcfg** lưu trữ và truy xuất dữ liệu trong file cấu hình ACAD.CFG.
9. Chương trình có thể kiểm tra môi trường làm việc nhờ các hàm **Ver** và **Getenv**.

14.4 Bài tập

1. Viết chương trình FILECOPY.LSP dùng để sao chép nội dung của một file bằng các hàm đọc ghi file. Chương trình yêu cầu người sử dụng nhập tên file nguồn và file đích.
2. Viết chương trình LISPLOAD.LSP tạo ra hàm C:LSP yêu cầu người sử dụng chọn một file **AutoLISP** để tải vào. Dùng hàm **Getfiled** để chọn file.
3. Viết chương trình TEXTIN.LSP yêu cầu người sử dụng chon một file văn bản và in từng dòng văn bản lên màn hình đồ họa bằng lệnh **Text** của **AutoCAD**. Dùng hàm **Getfiled** để chọn file.
4. Viết chương trình TEXTOUT.LSP cho phép người sử dụng chọn nhiều dòng chữ trên màn hình, sau đó chép chúng vào một file văn bản. Cần tạo ra một tập hợp chọn cho các dòng chữ này. Lưu ý: khi chọn đối tượng bằng cửa sổ chọn hoặc cửa sổ cắt, kết quả tạo ra trong file văn bản có thể không được như ý. Tốt nhất mỗi lần chọn một đối tượng hoặc nếu cần thì dùng hàm **Nentsel**.
5. Viết chương trình STORSET.LSP chép thành một file chứa các giá trị hiện hành của các biến: ORTHOMODE, GRIDMODE, SNAPMODE, LUPREC, AUPREC, LUNITS, AUNITS, BLIPMODE, APERTURE, PICKBOX, ANGDIR, ANGBASE. Dùng hàm **Getfiled** để chọn tên file mới và gán thêm phần mở rộng ".SET" cho tên file này.

6. Viết chương trình GETSET.LSP đọc dữ liệu chứa trong file tạo bởi chương trình STORSET.LSP ở trên, sau đó gán giá trị cho các biến của môi trường hiện hành. Chương trình chỉ chấp nhận file có phần mở rộng là ".SET".

14.5 Lời giải

1.

```
;Tên file: FILECOPY.LSP
;
(defun C:FCP (/ PR PW RL)
  (setq PR (getstring "Source file: ")
        PR (findfile PR)
        PW (getstring "Target file: ")
  ); Đóng setq
;
  (setq PR (open PR "r")
        PW (open PW "w")
  )
;
  (while (setq RL (read-line PR))
    (write-line RL PW)
  )

;
  (mapcar 'close (list PR PW))
  (princ)
); Đóng defun
;Kết thúc chương trình
```

2.

```
;Tên file: LISPLOAD.LSP
(defun C:LISPLOAD (/ FN)
  (setq FN
    (getfiled "\nAutoLISP filename to load: " "" "lsp" 10));
  (if (null FN)
    (princ "\nNo file selected")
    (progn
```

```
        (load FN)
        (princ (strcat "\nFile " FN "has loaded."))
    ); Đóng progn
  ); Đóng if
  (princ)
); Đóng defun
;Kết thúc chương trình
```

3.

```
;Tên file: TEXTIN.LSP
;
(defun C:TEXTIN (/ FN PR RL PT)
  (setq FN (getfiled "\nText file to print: " "" "txt" 0)); Gán tên file cho FN
  (if
    (null FN)                                  ;Nếu không tìm thấy file FN
    (princ "\nNo file selected")               ;thì in thông báo
    (progn                                     ;ngược lại, in nội dung file.
      (setq PR (open FN "r"))
      (setq PT (getpoint "\nSelect location to print text: "))
      (while (setq RL (read-line PR))          ;Đọc từng dòng trong file
          (command ".text" PT 2.5 0 RL)        ;In ra tại điểm PT
        (setq PT (polar PT (* 1.5 Pi) 3))      ;Điểm kế tiếp
      )                                        ; Đóng while
      (close PR)
    )                                          ; Đóng progn
  )                                            ; Đóng if
  (princ)
)                                              ; Đóng defun
;Kết thúc chương trình
```

4.

```
;Tên file: TEXTOUT.LSP
;
(defun C:TEXTOUT (/ FN EN TXT PW LST)
  (setq FN (getfiled "Select a file to write text" "" "txt" 1))
  (if (null FN)
    (princ "No file selected!")
    (progn
;Đưa các chuỗi TXT vào danh sách LST
```

```
      (while (setq EN (nentsel "\nSelect text: "))
        (setq TXT (cdr (assoc 1 (entget(car EN))))
              LST (append LST (list TXT) )
        )
      ); Đóng while
 ;Lần lượt lấy các chuỗi từ danh sách LST chép vào file PW.
       (setq PW (open FN "w"))
       (foreach TXT LST
          (write-line TXT PW)
       )
       (close PW)
      ); Đóng ELSE - progrn
   ); Đóng if
); Đóng defun
;Kết thúc chương trình
```

5.

```
;Tên file: STORSET.LSP
;
(defun C:STORSET (/ FN PW LST VAL)
;
;Chọn tên file cần lưu giá trị các biến
;
  (setq FN (getfiled "Select a file to store variables" "" "SET" 1))
   (if (null FN)
      (princ "No file selected!")
      (progn
;
;tạo danh sách LST chứa tên các biến cần lưu giá trị
       (setq LST (list "ORTHOMODE" "GRIDMODE" "SNAPMODE"
                       "LUPREC"  "AUPREC" "LUNITS" "AUNITS"
                       "BLIPMODE"  "APERTURE" "PICKBOX"
                       "ANGDIR" "ANGBASE"
             )
       )
;
;Mở file để lưu giá trị các biến
       (setq PW (open FN "w"))
       (foreach VAL LST
          (write-line (strcat VAL " = " (rtos (getvar VAL))) PW)
```

```
        )
        (close PW)
       ); Đóng ELSE - progrn
   ); Đóng if
;
;Kết thúc
  (princ (strcat "Variables has stored in file " FN))
  (princ)
); Đóng defun
;Kết thúc chương trình
```

Chương 15

CÁC BẢNG MÔ TẢ

Nội dung chương

1. Các hàm truy xuất các bảng mô tả của **AutoCAD**.
2. Truy xuất từ điển đối tượng của **AutoCAD**.
3. Các hàm xử lý đối tượng khung nhìn.

Các đối tượng hình học của **AutoCAD** (đường thẳng, đường tròn, đa tuyến ...) được lưu trữ trong cơ sở dữ liệu đối tượng. Ta có thể truy xuất các đối tượng này thông qua các hàm được trình bày ở chương 13.

Bên cạnh đó, **AutoCAD** còn có các *bảng mô tả* (như bảng mô tả khối, bảng mô tả lớp bản vẽ, bảng mô tả dạng đường ...) chứa dữ liệu mô tả cấu trúc của các khối, các lớp bản vẽ, các dạng đường... Ví dụ:

"APPID"	"STYLE"
"BLOCK"	"UCS"
"DIMSTYLE"	"VIEW"
"LAYER"	"VPORT"
"LTYPE"	

Ngoài ra, **AutoCAD** còn có *từ điển đối tượng* dùng để chứa các đối tượng Group và các kiểu đường mline.

15.1 Các hàm truy xuất các bảng mô tả của AutoCAD

Tương tự như cơ sở dữ liệu đối tượng, mỗi bảng mô tả chứa các record dữ liệu của các đối tượng tương ứng.

Hàm Tblnext

Hàm **Tblnext** duyệt từng phần tử trong bảng mô tả.

(**Tblnext** TABLE-NAME [REWIND])

Tham số TABLE-NAME chứa tên bảng mô tả. Nếu tham số REWIND khác nil, hàm này trả về phần tử đầu tiên. Nếu tham số REWIND bằng nil hoặc bị bỏ qua, hàm trả về phần tử kế tiếp trong bảng mô tả.

Ví dụ:

Command: **(tblnext "LAYER" T)** ↵

((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "CONTINUOUS"))

Giải thích:

(0 . "LAYER")	0 = Bảng mô tả	Bảng mô tả "LAYER"
(2 . "0")	2 = Tên đối tượng	Lớp bản vẽ "0"
(70 . 0)	70 = Cờ quy định trạng thái của lớp bản vẽ.	0 nghĩa là lớp bản vẽ đang ở trạng thái mở (On) và tan băng (Thaw).
(62 . 7)	62 = Mã màu	7 nghĩa là màu white. Nếu lớp bản vẽ đang bị tắt (Off), giá trị này là số âm.
(6."CONTINUOUS")	6 = Tên dạng đường	Dạng đường là "CONTINUOUS"

Ví dụ:

Command: **(tblnext "VIEW" T)** ↵	Không có khung nhìn View nào
nil	được tạo ra trong bản vẽ
Command: **View** ↵	Tạo View có tên Full
?/Delete/Restore/Save/Window: **S** ↵	

View name to save: **Full** ↵

Command: **(tblnext "VIEW" T)**↵ Đối tượng View đầu tiên trong bản vẽ

((0 . "VIEW") (2 . "FULL") (70 . 0) (40 . 7.09615) (10 7.82797 4.5) (41 . 15.6695) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Giải thích:

(0 . "VIEW")	Bảng mô tả
(2 . "FULL")	Tên đối tượng
(70 . 0)	Cờ trạng thái
(40 . 7.09615)	Chiều cao của khung nhìn View
(10 7.82797 4.5)	Tọa độ tâm khung nhìn View
(41 . 15.6695)	Chiều rộng của khung nhìn View
(11 0.0 0.0 1.0)	Điểm đỉnh véc tơ xác định hướng trục Z của WCS
(12 0.0 0.0 0.0)	Điểm gốc véc tơ xác định hướng trục Z của WCS
(42 . 50.0)	Tiêu cự
(43 . 0.0)	Mặt phẳng cắt trước trong hình chiếu phối cảnh
(44 . 0.0)	Mặt phẳng cắt sau trong hình chiếu phối cảnh
(50 . 0.0)	Góc quay ống kính trong hình chiếu phối cảnh
(71 . 0)	View mode

Ví dụ:

Command: **(tblnext "LTYPE" T)**↵	Truy xuất phần tử đầu tiên của bảng mô tả "LTYPE". Bảng "LTYPE" chỉ lưu trữ các dạng đường đã được tải vào bản vẽ.
((0 . "LTYPE")	Bảng mô tả dạng đường.
(2 . "CONTINUOUS")	Tên dạng đường
(70 . 0)	Cờ trạng thái
(3 . "Solid line")	Chuỗi mô tả dạng đường
(72 . 65)	Mã canh lề
(73 .0)	Số lượng các phân đoạn tạo thành *mẫu dạng đường* (*pattern*).

(40 . 0.0))	Chiều dài của một *mẫu dạng đường*
Command: **(tblnext "ltype")**↵	Truy xuất phần tử kế tiếp trong bảng mô tả "LTYPE"
((0 . "LTYPE")	Bảng mô tả dạng đường.
(2 . "ACAD_ISO02W100")	Tên dạng đường
(70 . 0)	Cờ trạng thái
(3 . "ISO dash __ __ __ __ __ __ __ __ __ __ __ __ __ __")	Chuỗi mô tả dạng đường
(72 . 65)	Mã canh lề
(73 . 2)	Số lượng các phân đoạn tạo thành *mẫu dạng đường* (1 phân đoạn nét liền, 1 phân đoạn khoảng trắng)
(40 . 15.0)	Chiều dài của một *mẫu dạng đường* (12.0 + 3.0)
(49 . 12.0)	Chiều dài phân đoạn 1 (nét liền 12.0)
(49 . -3.0))	Chiều dài phân đoạn 2 (khoảng trắng 12.0)
Command: **(tblnext "ltype")**↵	Truy xuất phần tử kế tiếp trong bảng mô tả "LTYPE"
nil	Vì không còn dạng đường nào nữa nên trả về *nil*.

Phân biệt giữa đối tượng định nghĩa khối và khối được chèn vào bản vẽ

✌ **Ví dụ:** Dùng lệnh **Bmake** tạo một khối có tên là "TRUC". Sau đó dùng lệnh **Insert** chèn khối này vào bản vẽ.

Command: **(setq A (tblnext "BLOCK" T))**↵	Truy xuất phần tử đầu tiên trong bảng mô tả "BLOCK"
((0 . "BLOCK")	Bảng mô tả
(2 . "TRUC")	Tên đối tượng *định nghĩa khối*

(70 . 0)	Kiểu khối (0 nghĩa là khối không chứa thuộc tính)
(10 0.0 0.0 0.0)	Tọa độ điểm gốc (X, Y, Z)
(-2 . <Entity name: 34b0560>))	Mã đối tượng dùng làm mẫu để **AutoCAD** đặt tên cho các khối khi chèn vào bản vẽ.
Command: **(setq S (entget (car (entsel)))) ↵**	
Select object:	Chọn khối được chèn trong bản vẽ
((-1 . <Entity name: 34b0578>)	Mã đối tượng
(0 . "INSERT")	Kiểu đối tượng là *khối được chèn*
(5 . "57")	Giá trị *handle* của đối tượng
(100 ."AcDbEntity")	Đánh dấu các mã DXF theo sau là các tính chất tổng quát của đối tượng
(67 . 0)	0 = Không gian mô hình; 1 = Không gian giấy vẽ
(8 . "0")	Lớp bản vẽ mà khối được chèn vào
(100 . "AcDbBlockReference")	Đánh dấu các mã DXF theo sau là các tính chất của kiểu đối tượng là khối chèn.
(2 . "TRUC")	Tên của đối tượng *định nghĩa khối*
(10 5.20871 4.51442 0.0)	Tọa độ điểm chèn (X, Y, Z)
(41 . 1.0)	Hệ số chèn theo trục X
(42 . 1.0)	Hệ số chèn theo trục Y
(43 . 1.0)	Hệ số chèn theo trục Z
(50 . 0.0)	Góc quay
(70 . 0)	Số cột khi chèn khối thành dãy
(71 . 0)	Số hàng khi chèn khối thành dãy
(44 . 0.0)	Khoảng cách giữa các cột
(45 . 0.0)	Khoảng cách giữa các dòng
(210 0.0 0.0 1.0))	Tọa độ điểm đỉnh của vec tơ xác định hướng trục Z

1. Mã đối tượng (*<Entity name: 34b0560>*) chứa trong bảng mô tả chỉ được dùng làm mẫu để **AutoCAD** đặt tên khi chèn các khối vào bản vẽ. Mã đối tượng do hàm **Entget** trả về (*<Entity name: 34b0578>*) mới thật sự chỉ đến các *record* trong cơ sở dữ liệu đối tượng.
2. Đối tượng *định nghĩa khối* có kiểu là "BLOCK". Đó là tên của bảng mô tả. Các *khối* chèn vào bản vẽ có kiểu là "INSERT", là tên của kiểu đối tượng bản vẽ.
3. Mã DXF 10 của bảng mô tả là *(10 . 0.0 0.0 0.0)* giữ nguyên không đổi. Điểm chèn của khối trong WCS là *(10 5.20871 4.51442 0.0)* sẽ thay đổi khi khối được di chuyển trong bản vẽ. Ngoài ra, tọa độ các điểm của khối do hàm **Nentsel** trả về được tính thông qua ma trận biến đổi tọa độ điểm từ hệ trục tọa độ đối tượng sang hệ trục tọa độ thực (*Model to World Transformation Matrix*).
4. Thuộc tính của khối là một đối tượng có kiểu là "ATTDEF". Đối twợng này được truy xuất bằng hàm **Nentsel**.
5. Mã DXF -2 của bảng mô tả chứa mã đối tượng dùng làm mẫu. Các đối tượng thành phần của khối được liệt kê tiếp theo trong bảng. Mã của các đối tượng này chỉ có tác dụng trong phạm vi bảng mô tả mà thôi.

Ví dụ: So sánh mã đối tượng do các hàm sau đây trả về:

Command: **(setq B (entget (entnext (cdr (assoc -2 A)))))**↵	Biến A chứa khối đầu tiên trong bảng mô tả *(assoc –2 A)* trả về mã DXF –2 *(cdr (assoc –2 A))* trả về mã đối tượng chính *(entnext . . .)* trả về mã đối tượng con thứ nhất *(entget . . .)* trả về dữ liệu của đối tượng con này
((-1 . <Entity name: 34b05d8>)	Mã đối tượng con
(0 . "LINE")	Kiểu đối tượng là đường thẳng
(5 . "63")	Giá trị *handle* của đối tượng
(100 . "AcDbEntity")	Đánh dấu các mã DXF theo sau là các tính chất tổng quát của đối tượng
(67 . 0)	0 = Không gian mô hình; 1 = Không

	gian giấy vẽ
(8 . "0")	Tên lớp bản vẽ
(100 . "AcDbLine")	Đánh dấu các mã DXF theo sau là các tính chất của riêng đối tượng đường thẳng
(10 -1.82099 -0.317308 0.0)	Tọa độ điểm đầu
(11 0.578091 1.32692 0.0)	Tọa độ điểm cuối
(210 0.0 0.0 1.0))	Tọa độ điểm đỉnh của véc tơ xác định hướng trục Z
Command: **(setq V (nentsel))**↵	Dữ liệu đối tượng con tạo ra bằng hàm **Nentsel**.
Select object:	Chọn đường thẳng là đối tượng con thứ nhất của khối (tương ứng với đường thẳng ở trên)
(<Entity name: 34b05d8>)	Mã đối tượng
(9.14313 8.87252 0.0)	Tọa độ điểm chọn trên đối tượng
((1.0 0.0 0.0)	Ma trận biến đổi tọa độ điểm
(0.0 1.0 0.0)	
(0.0 0.0 1.0)	
(9.5767 8.32444 0.0)	
)	
(<Entity name: 34b05b0>)	Mã của đối tượng chính (là khối chứa đường thẳng này)
)	

Ví dụ:

Sử dụng hàm **Tblnext** duyệt bảng mô tả "LTYPE" để hiện tên các dạng đường được tải vào bản vẽ.

```
;Tên file: LOADTYPE.LSP
; Mục đích: Chương trình liệt kê tất cả các dạng đường được tải vào bản vẽ
;          Kết quả hiện lên trong màn hình văn bản tương tự như lựa
;          chọn "?" của lệnh Linetype của AutoCAD .
;
```

```
(defun C:LOADTYPE (/ LTYP LST TYP DES GAP CTR SPA NAME
                     CHECK)
  (setvar "cmdecho" 0)
  (setq LTYP (tblnext "ltype" T) LST '() )                    ; [1]
  (while LTYP                                                 ; [2]
    (setq TYP  (cdr (assoc 2 LTYP))
          LST  (cons TYP LST)
          DES   (cdr (assoc 3 LTYP))
          LST   (cons DES LST)
          LTYP (tblnext "ltype")
    )                                  ; Đóng setq
  )                                    ; Đóng while
  (setq LST (reverse LST) GAP 19 CTR 0 SPA " ")               ; [3]
  (command ".textscr")
  (princ "\n_ _ _ _ LineType _ _ _ _ _ _ _ _Description _ _ _\n")  ; [4]
;
  (while (setq NAME (nth CTR LST))                            ; [5]
    (princ (strcat "\n"                                       ; [6]
                   NAME
                   (repeat (- GAP (strlen NAME))
                           (setq SPA (strcat " " SPA))
                   )
                   (nth (1+ CTR) LST)
           )                           ; Đóng strcat
    )                                  ; Đóng princ
    (setq CHECK (+ CTR 2))                                    ; [7]
    (if (= (gcd CHECK 10) 10) (princ "\n"))                   ; [8]
    (if (= (gcd CHECK 30) 30) (getstring "\n- -<Enter> for More- -")) ; [9]
    (setq SPA " " CTR CHECK)                                  ; [10]
  )                                    ; Đóng while
;
  (setvar "cmdecho" 1)
  (princ)
)
;Kết thúc chương trình
```

Giải thích:

[1] **(setq LTYP (tblnext "ltype" T) LST '())** Gán phần tử đầu tiên trong bảng mô tả "LTYPE" cho biến LTYP. Khởi tạo một danh sách rỗng

LST dùng để chứa tất cả các dạng đường và các chuỗi mô tả dạng đường tương ứng.

[2] **(while LTYP**
(setq TYP (cdr (assoc 2 LTYP))
LST (cons TYP LST)
DES (cdr (assoc 3 LTYP))
LST (cons DES LST)
LTYP (tblnext "ltype")
)

Trong quá trình duyệt các dạng đường, bổ sung tên dạng đường *(assoc 2 LTYP)* và chuỗi mô tả *(assoc 3 LTYP)* vào danh sách LST. Sau đó, biến LTYP được gán dạng đường kế tiếp trong bảng mô tả *(tblnext "LTYPE")*, và sẽ trả về *nil* sau khi đã duyệt tất cả các dạng đường. Giá trị *nil* này sẽ kết thúc vòng lặp **While** .

[3] **(setq LST (reverse LST) GAP 19 CTR 0 SPA " ")** Vì các dạng đường được đưa vào LST theo thứ tự ngược (bằng hàm **Cons**), nên phải đảo ngược danh sách này lại *(reverse LST)*. Khởi tạo các biến GAP, CTR, SPA. Biến GAP=19 là khoảng cách lớn nhất giữa tên dạng đường và chuỗi mô tả khi in lên màn hình. Biến đếm CTR là vị trí trong danh sách LST khi đang duyệt. SPA là khoảng cách chính xác giữa tên dạng đường và chuỗi mô tả (vì mỗi tên dạng đường có chiều dài khác nhau).

[4] **(princ "\n_ _ _ _ LineType _ _ _ _ _ _ _ _Description _ _ _\n")** Hiện dòng tiêu đề LineType và Description.

[5] **(while (setq NAME (nth CTR LST))** Khi vẫn còn tên dạng đường NAME trong danh sách LST, thì in các thông tin theo sau. Hàm **Nth** bắt đầu tại phần tử đầu tiên trong LST.

[6] Kết nối tên dạng đường NAME *(nth CTR LST)* và chuỗi mô tả *(nth (1+ CTR) LST)* và in chuỗi này lên màn hình. Hàm **Repeat** dùng để chèn các khoảng trắng vào giữa tên dạng đường và chuỗi mô tả. Số lượng các khoảng trắng là (- GAP (strlen NAME)). Tên dạng đường NAME càng dài thì GAP càng ngắn.

[7] **(setq CHECK (+ CTR 2))** Sau khi in từng cặp NAME và DES thì gán biến CHECK bằng (+ CTR 2).

[8] **(if (= (gcd CHECK 10) 10)** thì in một dòng trắng. Dùng để in một dòng trắng sau khi in 5 dòng dạng đường. Ghi chú: Hàm **Gcd** trả về True nếu CHECK đạt đến 10.

[9] **(if (= (gcd CHECK 30) 30)** thì hiện dòng thông báo - *-<Enter> for More-* - . Sau khi in 15 dòng dạng đường, chương trình dừng lại để người sử dụng quan sát màn hình. Ghi chú: Hàm **Getstring** không những hiện thông báo lên màn hình mà còn đợi người sử dụng nhập vào *Enter*.

[10] **(setq SPA " " CTR CHECK)** Biến SPA được khởi tạo lại để nhận giá trị tương ứng đối với biến NAME kế tiếp. Biến đếm CTR được gán bằng CHECK để duy trì vị trí đang truy xuất trong danh sách LST.

Thực hiện chương trình tại dòng nhắc lệnh:

Command: **loadtype** ↵

_ _ _ _ _ LineType _ _ _ _ _ _ _ _ _Description _ _ _

CONTINUOUS Solid line
ACAD_ISO02W100 ISO dash __ __ __ __ __ __ __ __ __ __ __ __ __
ACAD_ISO03W100 ISO dash space __ __ __ __ __ __
ACAD_ISO04W100 ISO long-dash dot ____ . ____ . ____ . ____ . _
ACAD_ISO05W100 ISO long-dash double-dot ____ .. ____ .. ____ .

ACAD_ISO06W100 ISO long-dash triple-dot ____ ... ____ ... ____
ACAD_ISO07W100 ISO dot . . . . . .
ACAD_ISO08W100 ISO long-dash short-dash ____ __ ____ __ ____ _
ACAD_ISO09W100 ISO long-dash double-short-dash ____ __ __ ____

Hàm TBLSEARCH

Hàm **Tblsearch** dùng để tìm một đối tượng trong bảng mô tả. Nếu tìm thấy sẽ trả về toàn bộ *record* dữ liệu này.

(**Tblsearch** TABLE-NAME SYMBOL [SETNEXT])

Tham số TABLE-NAME chứa tên bảng mô tả. Tham số SYMBOL chứa đối tượng cần tìm. Nếu tham số tùy chọn SETNEXT khác nil, con trỏ vị trí trong bảng mô tả sẽ chuyển sang đối tượng kế tiếp, và do đó sẽ ảnh hưởng đến lần gọi hàm **Tblnext** kế tiếp.

Ví dụ:

Command: **(setq Q (tblsearch "STYLE" "TCVN"))**↵

((0 . "STYLE") Bảng mô tả "STYLE"

(2 . "TCVN")	Tên đối tượng "TCVN"
(70 . 0)	Flag Linked with mã DXF 71
(40 . 0.0)	Chiều cao chữ = 0.0
(41 . 1.0)	Hệ số tỉ lệ chiều rộng 1.0
(50 . 0.0)	Góc nghiêng 0.0 radians
(71 . 0)	0 = bình thường, 2 = đối xứng gương theo phương thẳng đứng, 4 = đối xứng gương theo phương ngang.
(42 . 0.2)	Chiều cao của đối tượng text tạo ra cuối cùng trên bản vẽ (thuộc bất kỳ kiểu chữ nào) là 0.2
(3 . "txt.shx")	Tên file font chữ = "txt.shx"
(4 . "vn.shx")	Tên Bigfont file = "vn.shx"
)	

Ví dụ: Chương trình tìm kiếm đối tượng trong các bảng mô tả.

```
;Tên file: SEARCH.LSP
;
(defun SEARCH (DOG)
  (setvar "cmdecho" 0)
  (setq LST '("Layer" "LType" "Style" "Dimstyle" "Block"          ; [1]
              "Ucs" "Appid" "View" "Vport")
  )
  (command ".textscr")
  (foreach TNAME LST                                              ; [2]
    (cond ((null (setq ITEM (tblnext TNAME T)))                   ; [3]
           (princ (strcat "\nThere are no entries in the " TNAME " table."
                  )                          ; Đóng strcat
           )                                 ; Đóng princ
           )
           ((setq ITEM (Tblsearch TNAME DOG))
           (princ (strcat "\n" DOG " was found in the " TNAME "Table."
                  )                          ; Đóng strcat
           )                                 ; Đóng princ
           )
```

```
        (T
        (princ (strcat "\n" DOG " was NOT found in the " TNAME
                       "Table."
               )                    ; Đóng strcat
        )                           ; Đóng princ
        )
    )                               ; Đóng cond
  )                                 ; Đóng foreach
  (setvar "cmdecho" 1)
  (princ)
)
; Kết thúc chương trình
```

Giải thích:

[1] **(setq LST '("Layer" "LType" "Style" "Dimstyle" "Block" "Ucs" "Appid" "View" "Vport")** Biến LST chứa tên tất cả các bảng mô tả của **AutoCAD** . Các tên này sẽ được chuyển cho các hàm **Tblsearch** và **Tblnext**.

[2] Hàm **Foreach** lấy từng phần tử trong danh sách LST để chuyển cho hàm **Cond**.

[3] Có 3 trường hợp xảy ra khi tìm TNAME trong các bảng mô tả:

A. **(null (setq ITEM (tblnext TNAME T)))** Truy xuất phần tử đầu tiên của bảng mô tả TNAME. Nếu kết quả trả về là nil thì in thông báo không có phần tử nào trong bảng mô tả này.

B. **(setq ITEM (Tblsearch TNAME DOG))** Tìm phần tử DOG trong bảng mô tả TNAME. Nếu điều kiện này trả về True, thì hiện thông báo tìm thấy.

C. **T (True)** Khi cả hai điều kiện 1 và 2 đều trả về nil, thì thực hiện các biểu thức của điều kiện 3, in thông báo giá trị DOG không tìm thấy trong bảng mô tả này.

Thực hiện chương trình này tại dòng nhắc lệnh:

Command: **(search "TCVN")** ↵
TCVN was NOT found in the LayerTable.
TCVN was NOT found in the LTypeTable.
*TCVN was **found** in the StyleTable.*
TCVN was NOT found in the DimstyleTable.

TCVN was NOT found in the BlockTable.
There are no entries in the Ucs table.
TCVN was NOT found in the AppidTable.
There are no entries in the View table.
There are no entries in the Vport table.

Hàm TBLOBJNAME

Hàm **Tblobjname** dùng để tìm một đối tượng trong bảng mô tả, tương tự như hàm **Tblsearch,** nhưng giá trị trả về là mã đối tượng.

(**Tblobjname** TABLE-NAME SYMBOL)

Tham số TABLE-NAME chứa tên bảng mô tả. Tham số SYMBOL chứa đối tượng cần tìm. Nếu tìm thấy, hàm trả về mã đối tượng, ngược lại hàm trả về *nil*.

Để lấy thông tin về đối tượng này, ta dùng hàm **Enget**.

Ví dụ:

Command: **(setq LAY (Tblobjname "LAYER" "0"))** ↵	Tìm lớp "0" trong bảng mô tả "LAYER"
<Entity name: 2fa0478>	Tìm thấy và trả về mã đối tượng này
Command: **(entget LAY)** ↵	Lấy thông tin bằng hàm **Entget**
((-1 . <Entity name: 2fa0478>)	Mã đối tượng
(0 . "LAYER")	Kiểu đối tượng
(5 . "F")	Giá trị *handle* của đối tượng
(100 . "AcDbSymbolTableRecord")	Đánh dấu các mã DXF theo sau được lấy từ bảng mô tả.
(100 . "AcDbLayerTableRecord")	Đánh dấu các mã DXF theo sau được lấy từ bảng mô tả "LAYER".
(2 . "0")	Tên đối tượng
(70 . 0)	Cờ trạng thái của lớp bản vẽ
(62 . 7)	Màu của lớp bản vẽ
(6 . "CONTINUOUS")	Dạng đường của lớp bản vẽ
)	
Command: **(entget (Tblobjname**	

"STYLE" "STANDARD")) ↵

((-1 . <Entity name: 2fa0480>)	Mã đối tượng
(0 . "STYLE")	Kiểu đối tượng
(5 . "10")	Giá trị *handle* của đối tượng
(100 . "AcDbSymbolTableRecord")	Đánh dấu các mã DXF theo sau được lấy từ bảng mô tả.
(100 . "AcDbTextStyleTableRecord")	Đánh dấu các mã DXF theo sau được lấy từ bảng mô tả "STYLE".
(2 . "STANDARD")	Tên đối tượng
(70 . 0)	Cờ trạng thái của kiểu chữ
(40 . 0.0)	Chiều cao chữ
(41 . 1.0)	Hệ số tỉ lệ chiều rộng
(50 . 0.0)	Góc nghiêng
(71 . 0)	0 = bình thường, 2 = đối xứng theo phương thẳng đứng, 4 = đối xứng theo phương ngang.
(42 . 2.5)	Chiều cao của đối tượng text tạo ra cuối cùng trên bản vẽ (thuộc bất kỳ kiểu chữ nào) là 2.5
(3 . "txt")	Tên file font chữ = "txt.shx"
(4 . ""))	Tên Bigfont file = "vn.shx"

Hàm SNVALID

Hàm **Snvalid** (*Symbol Name VALID*) kiểm tra chuỗi chứa trong tham số SYM_NAME, biểu diễn tên đối tượng chứa trong các bảng mô tả, có hợp lệ hay không. Các ký tự hợp lệ là: các chữ cái, và các dấu $, –, _ .

(**Snvalid** SYM_NAME [FLAG])

Tham số SYM_NAME chứa tối đa 31 ký tự. Nếu tên này hợp lệ, hàm **Snvalid** trả về **T**, ngược lại trả về *nil*.

Tham số FLAG = 1 : Cho phép chứa các ký tự không hợp lệ khi tham khảo từ các file ngoài.

Tham số FLAG = 0 : (Mặc định) Không cho phép chứa các ký tự không hợp lệ khi tham khảo từ các file ngoài.

Ví dụ:

(snvalid "Truc1")	trả về T
(snvalid "Truc 1")	trả về nil. Không được chứa khoảng trắng
(snvalid "Truc-1")	trả về T
(snvalid "$---Truc1")	trả về T
(snvalid "XrefILayerName")	trả về nil. Không được chứa dấu "I"
(snvalid "XrefILayerName" 1)	trả về T. Cho phép chứa dấu "I"
(snvalid 3D_UCS)	trả về lỗi. Sai kiểu dữ liệu
(snvalid "3D_UCS")	trả về T

Ví dụ:

Chương trình sau đây sử dụng ưu điểm của hàm **Snvalid** để tạo một giao diện giữa chương trình với người sử dụng, cho phép thay đổi hệ trục tọa độ UCS.

```
;Tên file: SELUCS.LSP
;
;Mục đích: Sau khi giới thiệu các hệ trục UCS của bản vẽ hiện hành (bằng
;hộp thoại và hiện trên màn hình), chương trình yêu cầu người sử dụng
;chọn lấy một hệ trục UCS để chuyển thành UCS hiện hành.
;
(defun C:SELUCS (/ NEXT_UCS UCS_SET SETS NAME RES)
  (setvar "cmdecho" 0)
;
;Định nghĩa một hàm hiển thị tất các UCS được đặt tên lên màn hình.
  (defun SHOW ( )
    (foreach ITEM UCS_SET
      (princ (strcat "\nShowing UCS: " ITEM))
      (command ".ucs" "restore" ITEM        ;Hàm Foreach duyệt tất cả
               ".delay" 2000)               ;các UCS trong bảng mô tả
                                            ;"UCS
      )
  ) ; Đóng SHOW
;
;Duyệt các UCS trong bảng mô tả. Đưa tên các UCS vào một chuỗi để hiện
;ra hộp thoại thông báo. Đưa các UCS vào một danh sách UCS_SET để hàm
;SHOW hiển thị
;
(setq NEXT_UCS (tblnext "ucs" T)        ;Phần tử đầu tiên trong bảng mô tả
                                        ;"UCS"
```

```
    UCS_SET '()                          ;Khởi tạo danh sách rỗng UCS_SET
    SETS  "Current UCS's at this time"   ;Tiêu đề của hộp thông báo
)                                        ; tạo ra bằng lệnh Alert.
;
(if (null NEXT_UCS)                      ;IF không có UCS nào được đặt tên
    (princ "\nNo named UCS's at this time") ;THEN hiện thông báo
    (progn                               ; ELSE . . .
      (while NEXT_UCS                    ; Duyệt các UCS trong bảng mô tả
        (if
          (Snvalid (setq NAME (cdr (assoc 2 NEXT_UCS))))  ;Tên của UCS
          (setq SETS (strcat SETS "\n" NAME)  ;Thêm vào chuỗi thông báo
                UCS_SET (cons NAME UCS_SET)   ;Thêm vào tập hợp
                NEXT_UCS (tblnext "ucs")      ;Chuyển qua UCS kế tiếp
                                              ;trong bảng mô tả
          ); Đóng setq
          (princ (strcat "\nInvalid name: " NAME))
        ); Đóng if
      ); Đóng while
      (alert SETS)                       ;Hiện hộp thông báo.
      (setq UCS_SET (reverse UCS_SET))   ; Đảo ngược danh sách các UCS
      (command ".ucs" "world")
      (SHOW)                             ;Thực hiện hàm SHOW
;Yêu cầu người sử dụng nhập vào tên một UCS để biến thành UCS hiện
;hành, hoặc nhập ENTER để xem lại các UCS bằng hàm SHOW
      (if (= ""                          ;IF RES = Enter
            (setq RES (getstring "\nSelect named UCS/Display <enter>: "))
          )
          (progn                         ;THEN, SHOW và chọn UCS
            (SHOW)
            (alert SETS)
            (setq RES (getstring "\nSelect named UCS: "))
            (command ".ucs" "restore" RES)
          ); Đóng progn
          (command ".ucs" "restore" RES)  ;ELSE, phục hồi UCS cũ.
      )
    ); Đóng progn
  ); Đóng if
  (princ)
);Đóng SELUCS
;Kết thúc chương trình
```

Hàm WCMATCH

Hàm **Wcmatch** so sánh một chuỗi có thỏa mãn chuỗi ký tự đại diện hay không.

(**Wcmatch** STRING PATTERN)

Tham số STRING chứa chuỗi cần so sánh. Tham số PATTERN chứa chuỗi ký tự đại diện dùng để kiểm tra tham số STRING. Mỗi tham số chứa tối đa 500 ký tự. Nếu vượt quá giá trị này, các ký tự sẽ bị bỏ qua, không được so sánh.

Phép so sánh có **phân biệt dạng chữ hoa chữ thường**. Nếu so sánh trùng nhau, hàm sẽ trả về giá trị **T** (true).

Bảng liệt kê các ký tự đại diện

Ký tự đại diện	Ý nghĩa
#	Đại diện một ký tự là chữ số 0-9
@	Đại diện một ký tự là chữ cái a-z
.	Đại diện một ký tự không phải là chữ số hoặc chữ cái
*	Đại diện cho một nhóm nhiều ký tự, kể cả ký tự rỗng
?	Đại diện cho một ký tự duy nhất
~	Nếu ~ đứng ở đầu chuỗi, thì mọi chuỗi khác với chuỗi ký tự đại diện sẽ được chấp nhận. (Nếu ký tự này đứng giữa chuỗi ký tự đại diện thì xem như là ký tự bình thường)
[. . .]	So trùng với bất kỳ ký tự nào chứa trong dấu ngoặc vuông.
[~. . .]	So trùng với bất kỳ ký tự nào *không* chứa trong dấu ngoặc vuông.
-	Sử dụng trong dấu [] để biểu diễn miền giá trị của các ký tự. (Nếu ký tự này đứng ở vị trí đầu tiên hoặc sau cùng trong chuỗi ký tự đại diện thì xem như là ký tự bình thường)
,	Dấu phân cách hai ký tự đại diện
`	Đứng trước các ký tự điều khiển để các ký tự này được xem là ký tự bình thường. (Ghi chú: trên bàn phím, ký tự ` này nằm tại phím phía trên phím TAB)

Ví dụ:

(Wcmatch "TCVN" "**T***")	trả về T	Chấp nhận các chuỗi bắt đầu bằng ký tự "T" viết hoa, sau đó là bất kỳ các ký tự nào cũng được.
(Wcmatch "TCVN1" "**T@@@#**")	trả về T	Chấp nhận các chuỗi bắt đầu bằng ký tự "T" viết hoa, sau đó là 3 chữ cái và 1 chữ số.
(Wcmatch "TCVN1-a" "***?.?**")	trả về T	Chấp nhận các chuỗi có 3 ký tự cuối cùng thỏa mãn: 1 ký tự, 1 ký tự không phải chữ số hoặc chữ cái, 1 ký tự.
(Wcmatch "TCVN11a" "***?.?**")	trả về nil	Chữ số "1" không thỏa mãn ký tự đại diện "."
(Wcmatch "TCVN" "**~T***")	trả về nil	Chấp nhận các chuỗi *không* bắt đầu bằng ký tự "T".
(Wcmatch "TCVN2" "*[123]")	trả về T	Chấp nhận các chuỗi kết thúc bằng "1" hoặc "2" hoặc "3".
(Wcmatch "TCVN2" "*[~123]")	trả về nil	Chấp nhận các chuỗi *không* kết thúc bằng "1" hoặc "2" hoặc "3".
(Wcmatch "TCVN2" "*[1-5]")	trả về T	Chấp nhận các chuỗi kết thúc bằng chữ số từ 1 đến 5.
(Wcmatch "TCVN2" "S*,?????,*[1-5],T*")	trả về T	Có thể sử dụng nhiều tham số PATTERN. Các tham số này phân cách nhau bằng các dấu phẩy. Chuỗi "TCVN2" chỉ cần thỏa mãn **một** trong các điều kiện sau: - Bắt đầu bằng chữ "S" - Gồm tất cả là 5 ký tự - Kết thúc bằng một trong các số từ 1 đến 5. - Bắt đầu bằng chữ "T".

Có thể sử dụng các biến để cung cấp giá trị cho các tham số của hàm **Wcmatch**.

Ví dụ:

```
(setq STG (cdr (assoc 2 (tblnext "layer")))
      PAT (getstring "\nPattern argument: ")
)
(if (Wcmatch STG PAT) (setq NUM 5))
```

;Trả về tên lớp bản vẽ
;Chuỗi ký tự đại diện để so sánh
Nếu tên lớp bản vẽ STG trùng với biến PAT thì gán biến NUM bằng 5.

Hàm **Wcmatch** thường được sử dụng khi các chuỗi cần so sánh chứa trong các danh sách phức hợp.

Ví dụ:

Ví dụ sau đây sử dụng các hàm **Entget** và **Entlast** để lấy ra dữ liệu của đối tượng được tạo ra trong bản vẽ.

```
(setq OBJ (entget (entlast)))
(if (Wcmatch (cdr (assoc 8 OBJ)) "~0")
    (command ".chprop" "last" "" "la" "0" "")
)
```

Record dữ liệu của đối tượng
Hàm **Cdr** của mã DXF 8 trả về tên lớp bản vẽ của đối tượng. Nếu tên này khác "0" thì chuyển đối tượng này về lớp "0".

Trong một số trường hợp cần sử dụng các ký tự đặc biệt trong chuỗi ký tự đại diện, ta phải thêm dấu ` (phím phía trên phím TAB) ở phía trước ký tự này.

Ví dụ:

```
(Wcmatch "C:\\ACAD\\ACADLISP\\" "*`\\*")
```

Trả về T.
Trong **AutoLISP** ta phải dùng hai dấu gạch (\\) để biểu diễn đường dẫn thư mục: C:\ACAD\ACADLISP\. Chuỗi ký tự đại diện "*`*" cho phép các ký tự xuất hiện phía trước và phía sau dấu \

```
(Wcmatch "LISPFILE" "*`\\*")
```

Trả về nil
Vì chuỗi "LISPFILE" không chứa dấu \

Ví dụ:

Chương trình sau đây sử dụng ưu điểm của hàm **Wcmatch** để lấy ra các đối tượng trong bảng mô tả thỏa mãn chuỗi ký tự đại diện.

```
;Tên file: WC.LSP
;
;Mục đích: Duyệt bảng mô tả với chuỗi ký tự đại diện. Có thể sử dụng
;nhiều chuỗi ký tự đại diện cho hàm Wcmatch. Ghi nhớ: Chuỗi ký tự đại
;diện phân biệt dạng chữ hoa chữ thường. In kết quả lên màn hình văn bản.
;
(defun C:WC (/ RES TBL TLST NAMELST LST NAME)
  (setvar "cmdecho" 0)
  (setq RES (getstring "\nSpecify wild card search string: "))
;
  (initget 1 "Layer LType Style Dimstyle Block Ucs Appid View Vport")  ;[1]
;
  (prompt "\nSymbol table to search. . .")
  (setq TBL   (strcase (getkword
        "\nLayer LType/Style/Dimstyle/Block/Ucs/Appid/View/Vport: "))
        TLST  '("LAYER" "LTYPE" "STYLE" "DIMSTYLE"
                " BLOCK" "UCS" "APPID" "VIEW" "VPORT")
        NAMELST '()
  )
  (cond ((null (setq LST (tblnext TBL T)))                          ; [2]
         (princ (strcat "\nNo entries found in the " TBL "table!"))
         )
;
        (T                                                          ; [3]
         (while LST                                                 ; [4]
               (setq NAME (cdr (assoc 2 LST)))                      ; [5]
               (if (Wcmatch NAME RES)                               ; [6]
                  (setq NAMELST (cons NAME NAMELST))
               )
               (setq LST (tblnext TBL))                             ; [7]
         ) ; Đóng while
       ;
         (if (null NAMELST)                                         ; [8]
            (princ (strcat "\nNo W.C. matches for the " TBL " tablet!"))
            (progn
               (command ".textscr")
```

```
                (princ "\nMatching list\n- - - - - - - - - - - - -\n")
                (foreach N NAMELST
                  (princ N)
                  (princ "\n")
                ) ; Đóng foreach
              ) ; Đóng progn
            ) ; Đóng if
          ) ; Đóng T
        ) ; Đóng cond
        (setvar "cmdecho" 1)
        (princ)
); Đóng WC
;Kết thúc chương trình
```

Giải thích:

[1] (initget 1 "Layer LType Style Dimstyle Block Ucs Appid View Vport") Hàm **Initget** cung cấp danh sách các từ khóa hợp lệ là tên các bảng mô tả.

Sau khi người sử dụng nhập vào chuỗi ký tự đại diện và tên bảng mô tả, có hai khả năng sau đây xảy ra:

[2] (null (setq LST (tblnext TBL T))) Nếu không có đối tượng nào trong bảng mô tả thì thông báo cho người sử dụng.

[3] T (True) Truy xuất bảng mô tả, lấy ra các đối tượng có tên trùng với chuỗi ký tự đại diện.

[4] (While LST Duyệt từng đối tượng trong bảng mô tả.

[5] (setq NAME (cdr (assoc 2 LST))) Gán cho biến NAME tên của đối tượng (mã DXF 2).

[6] (if (Wcmatch NAME RES) Nếu NAME trùng với chuỗi ký tự đại diện, bổ sung NAME vào danh sách NAMELST.

[7] (setq LST (tblnext TBL)) Chuyển qua đối tượng kế tiếp trong bảng mô tả.

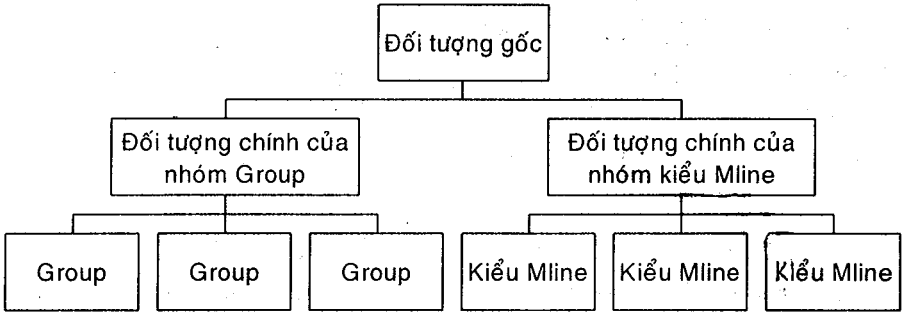
[8] (if (null NAMELST) Nếu không có đối tượng nào trong bảng mô tả trùng với chuỗi ký tự đại diện thì danh sách NAMELST bằng nil. Thông báo điều này cho người sử dụng. Nếu NAMELST có chứa các đối tượng thỏa điều kiện thì in chúng lên màn hình. Hàm **Foreach** duyệt từng phần tử trong danh sách NAMELST.

15.2 Từ điển đối tượng (Object dictionary)

Các bảng mô tả của **AutoCAD** là các đối tượng riêng biệt. Do đó, kể từ release 13 trở về sau, người ta tạo ra *từ điển đối tượng* (*Object dictionary*) để liên kết các bảng mô tả lại với nhau. Từ release 14 các đối tượng trong từ điển đối tượng được chia thành hai nhóm: chứa các dữ liệu mô tả các đối tượng *group* (là các tập hợp chọn có đặt tên, tạo ra bằng lệnh **Group** của **AutoCAD**) và các kiểu đường mline. Mỗi nhóm bắt đầu bằng một *đối tượng chính*. Trong các phiên bản sau của **AutoCAD**, các bảng mô tả khác sẽ được bổ sung thêm vào trong từ điển đối tượng.

Cấu trúc phân cấp của từ điển đối tượng:

- **Mức 1**: *Đối tượng gốc* của từ điển đối tượng, có kiểu là "DICTIONARY".
- **Mức 2**: *Đối tượng chính* của nhóm Group và *đối tượng chính* của nhóm kiểu đường Mline. Hai đối tượng này có kiểu là "DICTIONARY".
- **Mức 3**: Các đối tượng mô tả các Group và các kiểu đường Mline. Các đối tượng này có kiểu là "GROUP" và "MLINESTYLE".



Để truy xuất đến mã đối tượng gốc của từ điển đối tượng (mức 1), ta dùng hàm **Namedobjdict**. Sau đó dùng các hàm thích hợp (như **Entget**) để đi theo cấu trúc phân cấp này đến đối tượng mong muốn.

Hàm NAMEDOBJDICT

Hàm **Namedobjdict** (*NAMED Object DICTionary*) trả về mã đối tượng của từ điển đối tượng trong bản vẽ hiện hành. Cú pháp của hàm này như sau:

(**Namedobjdict**)

Ví dụ:

Thực hiện lần lượt các biểu thức **AutoLISP** sau:

Command: **(setq OD (namedobjdict))**↵	
<Entity name: 34e0460>	Mã đối tượng của từ điển đối tượng
Command: **(setq DICT (entget OD))**↵	Đối tượng gốc
((-1 . <Entity name: 34e0460>)	Mã đối tượng của từ điển đối tượng
(0 . "DICTIONARY")	Kiểu đối tượng "DICTIONARY"
(5 . "C")	Giá trị *handle* của đối tượng
(100 . "AcDbDictionary")	Đánh dấu vị trí các mã DXF của kiểu đối tượng "DICTIONARY"
(280 . 0)	Giá trị số nguyên 8 bit.
(3 . "ACAD_GROUP")	Đánh dấu vị trí các mã DXF của kiểu đối tượng "GROUP"
(350 . <Entity name: 34e0468>)	Mã *đối tượng chính* của nhóm Group
(3 . "ACAD_MLINESTYLE")	Đánh dấu vị trí các mã DXF của kiểu đối tượng "MLINESTYLES"
(350 . <Entity name: 34e0470>))	Mã *đối tượng chính* của nhóm kiểu đường mline.
Command: **(setq G-AND-M (member (assoc 3 DICT) DICT))**↵	
((3 . "ACAD_GROUP")	Đánh dấu GROUP
(350 . <Entity name: 34e0468>)	Mã *đối tượng chính* của nhóm Group
(3 . "ACAD_MLINESTYLE")	Đánh dấu vị trí MLINESTYLES
(350 . <Entity name: 34e0470>))	Mã *đối tượng chính* của nhóm kiểu đường mline.
Command: **(setq GROUP-ENT(cdr (assoc 350 G-AND-M)))**↵	
<Entity name: 34e0468>	Mã *đối tượng chính* của nhóm Group

Command: **(setq MLINE-AS (cddr G-AND-M))** ↵

((3 . "ACAD_MLINESTYLE") Đánh dấu vị trí MLINESTYLES

(350 . <Entity name: 34e0470>)) Mã *đối tượng chính* của nhóm kiểu đường mline.

Command: **(SETQ mline-ent (CDR (ASSOC 350 mline-as)))** ↵

<Entity name: 34e0470> Mã *đối tượng chính* của nhóm kiểu đường mline.

Command: **(entget GROUP-ENT)** ↵

((-1 . <Entity name: 34e0468>) Mã *đối tượng chính* của nhóm Group

(0 . "DICTIONARY") Kiểu đối tượng "DICTIONARY"

(5 . "D") Giá trị *handle* của đối tượng

(102 . "{ACAD_REACTORS") Chuỗi đánh dấu

(330 . <Entity name: 34e0460>) Mã đối tượng của từ điển đối tượng

(102 . "}") Chuỗi đánh dấu

(100 . "AcDbDictionary") Đánh dấu các mã DXF mô tả đối tượng "DICTIONARY"

(280 . 0) Giá tri số nguyên 8 bit

(3 . "TRUC") Tên của đối tượng Group trong bản vẽ hiện hành

(350 . <Entity name: 34e0570>) Mã đối tượng của đối tượng Group ở trên.

(3 . "BANHRANG") Tên của đối tượng Group trong bản vẽ hiện hành

(350 . <Entity name: 34e0578>) Mã đối tượng của đối tượng Group ở trên.

(3 . "THEN") Tên của đối tượng Group trong bản vẽ hiện hành

(350 . <Entity name: 34e0580>)) Mã đối tượng của đối tượng Group ở trên.

Command: **(entget MLINE-ENT)** ↵

((-1 . <Entity name: 34e0470>) Mã *đối tượng chính* của nhóm kiểu đường mline.

(0 . "DICTIONARY")	Kiểu đối tượng "DICTIONARY"
(5 . "E")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34e0460>)	Mã đối tượng của từ điển đối tượng
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbDictionary")	Đánh dấu các mã DXF theo sau mô tả đối tượng "DICTIONARY"
(280 . 0)	Giá trị số nguyên 8 bit
(3 . "STANDARD")	Tên của kiểu đường mline trong bản vẽ hiện hành
(350 . <Entity name: 34e04e0>)	Mã đối tượng của kiểu đường mline ở trên.
(3 . "TUONG")	Tên của kiểu đường mline trong bản vẽ hiện hành
(350 . <Entity name: 34e0588>))	Mã đối tượng của kiểu đường mline ở trên.
Command: **(entget (cdr (assoc 350 (entget MLINE-ENT))))** ↵	Kiểu đường mline đầu tiên trong từ điển đối tượng
((-1 . <Entity name: 34e04e0>)	Mã đối tượng
(0 . "MLINESTYLE")	Kiểu đối tượng
(5 . "1C")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34e0470>)	Mã *đối tượng chính* của nhóm kiểu đường mline.
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbMlineStyle")	Đánh dấu các mã DXF theo sau mô tả kiểu đường mline.
(2 . "STANDARD")	Tên kiểu đường mline
(70 . 0)	Giá trị xác định cấu trúc đường mline
(3 . "")	Chuỗi mô tả kiểu đường Mline
(62 . 0)	Mã màu dùng để tô đường Mline.
(51 . 1.5708)	Góc bắt đầu (Start angle), tính bằng radian
(52 . 1.5708)	Góc kết thúc (End angle), tính bằng radian

(71 . 2)	Mã quy định cách vẽ đoạn thẳng nối đường mline tại điểm đầu (caps)
(49 . 0.5)	Khoảng cách của nét thứ nhất tính từ tâm
(62 . 256)	Mã màu của nét thứ nhất
(6 . "BYLAYER")	Dạng đường của nét thứ nhất
(49 . -0.5)	Khoảng cách của nét thứ hai tính từ tâm
(62 . 256)	Mã màu của nét thứ hai
(6 . "BYLAYER"))	Dạng đường của nét thứ hai

Hàm DICTNEXT

Hàm **Dictnext** (*DICTionary NEXT*) duyệt từng phần tử kiểu "DICTIONARY" trong từ điển đối tượng (tương tự như hàm **tblnext**).

(**Dictnext** ENAME [REWIND])

Tham số ENAME chứa mã tự điển đối tượng. Tham số tùy chọn REWIND chỉ có thể chứa hai giá trị hợp lệ là "*acad_mlinestyle*" và "*acad_group*". Nếu không có tham số REWIND, hàm này trả về đối tượng kế tiếp trong từ điển đối tượng. Ngược lại, trả về đối tượng đầu tiên.

Ví dụ:

Command: **(setq OD (namedobjdict))**↵

<Entity name: 34e0460>	Mã đối tượng của từ điển đối tượng

Command: **(setq GRPS (dictnext OD "acad_group"))** ↵

((-1 . <Entity name: 34e0468>)	Mã *đối tượng chính* của nhóm Group
(0 . "DICTIONARY")	Kiểu đối tượng "DICTIONARY"
(5 . "D")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34e0460>)	Mã đối tượng của từ điển đối tượng
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbDictionary")	Đánh dấu các mã DXF theo sau mô tả đối tượng

	"DICTIONARY"
(280 . 0)	Giá trị số nguyên 8 bit
(3 . "TRUC")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0570>)	Mã đối tượng của đối tượng Group ở trên.
(3 . "BANHRANG")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0578>)	Mã đối tượng của đối tượng Group ở trên.
(3 . "THEN")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0580>))	Mã đối tượng của đối tượng Group ở trên.

Command: **(setq GRP-1 (entget (cdr (assoc 350 GRPS))))**↵

((-1 . <Entity name: 34e0570>)	Mã đối tượng của group có tên là "TRUC"
(0 . "GROUP")	Kiểu đối tượng "GROUP"
(5 . "4E")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34e0468>)	Mã đối tượng chính của nhóm Group.
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbGroup")	Đánh dấu các mã DXF theo sau mô tả kiểu đối tượng Group
(300 . "Truc khuyu")	Chuỗi mô tả của Group
(70 . 0)	Cờ cho phép chọn Group
(71 . 1)	Cờ
(340 . <Entity name: 34e0520>)	Mã đối tượng thứ nhất trong Group
(340 . <Entity name: 34e0528>))	Mã đối tượng thứ hai trong Group

Command: **(setq ENT1_GRP-1 (entget (cdr (assoc 340 GRP-1))))**↵

((-1 . <Entity name: 34e0520>)	Mã đối tượng thứ nhất trong Group
(0 . "LINE")	Kiểu đối tượng "LINE"
(5 . "4C")	Giá trị *handle của đối tượng*
(100 . "AcDbEntity")	Đánh dấu các mã DXF theo sau mô tả tổng quát một đối

	tượng.
(67 . 0)	Không gian mô hình
(8 . "0")	Tên lớp bản vẽ
(100 . "AcDbLine")	Đánh dấu các mã DXF mô tả đối tượng kiểu "LINE"
(10 2.13873 4.44231 0.0)	Điểm bắt đầu
(11 4.8603 6.23077 0.0)	Điểm kết thúc
(210 0.0 0.0 1.0))	Tọa độ điểm đỉnh của véc tơ xác định hướng trục Z.

Hàm DICTSEARCH

Hàm **Dictsearch** (*DICTionary SEARCH*) dùng để tìm phần tử kiểu "DICTIONARY" trong từ điển đối tượng (tương tự như hàm **tblnext**).

(**Dictsearch** ENAME SYMBOL)

Tham số ENAME chứa mã tự điển đối tượng. Tham số SYMBOL chỉ có thể chứa 2 giá trị hợp lệ là "*acad_mlinestyle*" và "*acad_group*".

Ví dụ:

Command: **(setq OD (namedobjdict))**↵

<Entity name: 34e0460>	Mã đối tượng của từ điển đối tượng
Command: **(Dictsearch OD "acad_group")** ↵	
((-1 . <Entity name: 34e0468>)	Mã *đối tượng chính* của nhóm Group
(0 . "DICTIONARY")	Kiểu đối tượng "DICTIONARY"
(5 . "D")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34es0460>)	Mã đối tượng của từ điển đối tượng
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbDictionary")	Đánh dấu vị trí các mã DXF mô tả đối tượng kiểu "DICTIONARY"
(280 . 0)	Giá trị số nguyên 8 bit
(3 . "TRUC")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0570>)	Mã đối tượng của đối tượng

	Group ở trên.
(3 . "BANHRANG")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0578>)	Mã đối tượng của đối tượng Group ở trên.
(3 . "THEN")	Tên của đối tượng Group trong bản vẽ hiện hành
(350 . <Entity name: 34e0580>))	Mã đối tượng của đối tượng Group ở trên.
Command: **(Dictsearch OD "acad_mlinestyle")** ↵	
((-1 . <Entity name: 34e0470>)	Mã *đối tượng chính* của nhóm kiểu đường mline.
(0 . "DICTIONARY")	Kiểu đối tượng "DICTIONARY"
(5 . "E")	Giá trị *handle* của đối tượng
(102 . "{ACAD_REACTORS")	Chuỗi đánh dấu
(330 . <Entity name: 34e0460>)	Mã đối tượng của từ điển đối tượng
(102 . "}")	Chuỗi đánh dấu
(100 . "AcDbDictionary")	Đánh dấu các mã DXF theo sau mô tả đối tượng "DICTIONARY"
(280 . 0)	Giá trị số nguyên 8 bit
(3 . "STANDARD")	Tên của kiểu đường mline trong bản vẽ hiện hành
(350 . <Entity name: 34e04e0>)	Mã đối tượng của kiểu đường mline ở trên.
(3 . "TUONG")	Tên của kiểu đường mline trong bản vẽ hiện hành
(350 . <Entity name: 34e0588>))	Mã đối tượng của kiểu đường mline ở trên.

✌ **Ví dụ:** Chương trình liệt kê tất cả các đối tượng Group trong bản vẽ

```
;Tên file: DICTGRP.LSP
;
;Mục đích: Liệt kê tên và mã tất cả các đối tượng Group.
;Sau đó yêu cầu người sử dụng chọn tên một Group, chương trình sẽ liệt kê
;các đối tượng tạo nên Group này.
;
```

```
(defun C:DG (/ MASTER GRP GRP-LIST NE-LIST ARCH-LIST
               RES LST ENTS AS-340)
  (setvar "cmdecho" 0)
  (setq MASTER (namedobjdict)          ;Mã từ điển đối tượng của bản vẽ
        GRP    (dictsearch MASTER "acad_group") ; Tìm một đối tượng
                                                ;Group trong từ điển
  )
  (if GRP                              ; Nếu tìm thấy đối tượng Group
    (progn
      (setq GRP-LIST (member (assoc 3 GRP) GRP)   ; Lập danh sách tên
                                                  ;và mã của tất cả các
                                                  ;Group
            NE-LIST '()               ;Khởi tạo danh sách rỗng dùng để
                                      ;chứa mã của các Group
      )
      (foreach ITEM GRP-LIST                                    ; [1]
        (setq NE-LIST (cons (cdr ITEM) NE-LIST))
      )
      (setq NE-LIST (reverse NE-LIST)                           ; [2]
            ARCH-LIST NE-LIST
      )
      (princ (STRCAT "\nGroup names / Entity names: "
                     "\n- - - - - - - - - - - - - - - - - - - - - \n"))
      (while NE-LIST                                            ; [3]
        (princ (strcat "\tGroup name: " (car NE-LIST) "\t"))
        (princ (cadr NE-LIST))
        (princ "\n")
        (setq NE-LIST (cddr NE-LIST))
      )
      (setq RES  (strcase (getstring "\n\nGroup for Entity listing: "))
            LST  (entget (cadr (member RES ARCH-LIST)))         ; [4]
            ENTS (member (assoc 340 LST) LST)                   ; [5]
      )
      (while (setq AS-340 (car ENTS))                           ; [6]
        (setq LST (entget (cdr AS-340)))
        (princ (strcat "\tEntity type: " (cdr (assoc 0 LST))))
        (princ (strcat "\tLayer design: " (cdr (assoc 8 LST)) "\n"))
        (setq ENTS (cdr ENTS))
      ); Đóng while
    ) ; Đóng progn
```

```
  ); Đóng if
  (princ)
); Đóng defun
;Kết thúc chương trình
```

Giải thích:

[1] (foreach ITEM GRP-LIST (setq NE-LIST (cons (cdr ITEM) NE-LIST))) Danh sách GRP-LIST chứa các mã DXF tên đối tượng và mã đối tượng. Duyệt từng phần tử ITEM của danh sách này, lấy ra tên đối tượng (hoặc mã đối tượng), kết nối vào danh sách NE-LIST. Sau khi duyệt toàn bộ danh sách GRP-LIST, danh sách NE-LIST chỉ chứa tên hoặc mã của các Group, loại bỏ được các mã DXF 3 và 350.

[2] (setq NE-LIST (reverse NE-LIST) ARCH-LIST NE-LIST) Đảo ngược danh sách NE-LIST, vì ban đầu danh sách này được tạo ra theo thứ tự ngược. Danh sách lưu trữ cấu trúc mới của NE-LIST.

[3] (while NE-LIST (princ (strcat "\tGroup name: " (car NE-LIST) . . . Khi danh sách NE-LIST vẫn còn phần tử, in tên và các mã của các group trên cùng một dòng. Biểu thức (setq NE-LIST (cddr NE-LIST)) loại bỏ 2 phần tử đầu tiên của NE-LIST. Mỗi cặp phần tử gồm tên và mã của đối tượng Group.

[4] (setq LST (entget (cadr (member RES ARCH-LIST)))) Biến RES lưu trữ tên Group do người sử dụng chọn. Hàm **Member** trả về phần còn lại trong danh sách ARCH-LIST bắt đầu từ RES. Hàm **Cdr** trả về mã đối tượng của Group chứa trong RES. Lấy ra dữ liệu của đối tượng này và lưu trong biến LST.

[5] (setq ENTS (member (assoc 340 LST) LST)) Mã DXF 340 đánh dấu vị trí bắt đầu của tất cả các đối tượng tạo thành Group. Hàm **Member** lấy ra tất cả các mã DXF này và được lưu trong biến ENTS.

[6] (while (setq AS-340 (car ENTS)) (setq LST (entget (cdr AS-340))). . . Truy xuất từng mã DXF trong ENTS, và lấy ra kiểu đối tượng *(cdr (assoc 0 LST))* và tên lớp bản vẽ *(cdr (assoc 8 LST))* để in lên màn hình.

Thực hiện chương trình:

Command: **dg** ↵

Group names / Entity names:

- -

Group name: TRUC *<Entity name: 34e0570>*

Group name: BANHRANG <Entity name: 2df0550>
Group name: THEN <Entity name: 34e0580>

Group for Entity listing: **TRUC** ↵
Entity type: LINE Layer design: 0
Entity type: CIRCLE Layer design: 0

Hàm DICTRENAME

Hàm **Dictrename** (*DICTionary RENAME*) đổi tên một đối tượng của nhóm.

(**Dictrename** ENAME OLDSYM NEWSYM)

Tham số ENAME chứa mã đối tượng chính "*acad_group*" hoặc "*acad_mlinestyle*". OLDSYM chứa tên đối tượng muốn đổi tên. NEWSYM chứa tên mới. Nếu giá trị các tham số này không hợp lệ, hàm **Dictrename** trả về nil.

Ví dụ: Đổi tên group "TRUC" thành "TRUCKHUYU"

Command: **GROUP** ↵ Chọn các đối tượng trên màn hình để tạo Group có tên "TRUC"
Command: **(setq OD (namedobjdict))** ↵ Mã đối tượng chính
<Entity name: 34e0460>
Command: **(setq GROUP (dictsearch OD "acad_group"))**↵
((-1 . <Entity name: 34e0468>) (0 . "DICTIONARY") (5 . "D")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 34es0460>) (102 . "}")
(100 . "AcDbDictionary")(280 . 0) (3 . ***"TRUC"****)*
(350 . <Entity name: 34e0570>) (3 . "BANHRANG")
(350 . <Entity name: 34e0578>) (3 . "THEN")
(350 . <Entity name: 34e0580>))

Command: **(setq GROUP-ENAME (cdr (assoc –1 GROUP)))** ↵
(-1 . <Entity name: 34e0468>)

*Command:***(setq NEWNAME (getstring "\nNew name for group TRUC: ")**
New name for group TRUC: **TRUCKHUYU** ↵
"TRUCKHUYU"

Command: **(Dictrename GROUP-ENAME "TRUC" NEWNAME)** ↵

"TRUCKHUYU"
Command: **(setq GROUP (dictsearch OD "acad_group"))**↵
((-1 . <Entity name: 34e0468>) (0 . "DICTIONARY") (5 . "D")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 34es0460>) (102 . "}")
*(100 . "AcDbDictionary")(280 . 0) (3 . **"TRUCKHUYU"**)*
(350 . <Entity name: 34e0570>) (3 . "BANHRANG")
(350 . <Entity name: 34e0578>) (3 . "THEN")
(350 . <Entity name: 34e0580>))

Ví dụ: Chương trình đổi tên đối tượng Group hoặc kiểu đường mline.

```
;Tên file: DICTREN.LSP
;
;Mục đích: Cho phép người sử dụng chọn tên Group hoặc kiểu đường mline
;để thay đổi tên mới.
;
(defun C:DREN (/ MASTER GRP MLS GRP-E MLS-E FINAL
                 OBJ-LIST OLD NEW)
  (setvar "cmdecho" 0)
  (setq MASTER   (Namedobjdict)                          ; Mã từ điển
        GRP      (Dictsearch MASTER "acad_group")        ; Group
        MLS      (Dictsearch MASTER "acad_mlinestyle")   ; MlineStyle
        GRP-E    (cdr (assoc -1 GRP))                    ; [1]
        MLS-E    (cdr (assoc -1 MLS))                    ; [2]
        FINAL    '( )                    ; Khởi tạo danh sách FINAL
  ) ; Đóng setq
;
; - <Định nghĩa hàm NAMES để lưu trữ và in các đối tượng Group/Mline>-
;
(defun NAMES (S-NAME S-STR)                              ; [3]
  (setq OBJ-LIST '() )                   ; Khởi tạo danh sách dùng để
                                         ;lưu trữ chuỗi
  (foreach ITEM S-NAME                                   ; [4]
    (if (= (car ITEM) 3)
       (setq OBJ-LIST (cons (cdr ITEM) OBJ-LIST))
    ) ; Đóng if
  ) ; Đóng foreach
  (setq OBJ-LIST (reverse OBJ-LIST))       ; Đảo ngược danh sách
  (princ (strcat "\n AutoCAD " S-STR       ; In tiêu đề
       " names: \n - - - - - - - - - - - - - - - - - - -\n"))
```

```
  (foreach ITEM OBJ-LIST (princ (strcat "\t" ITEM "\n")))          ; [5]
  (setq FINAL (append OBJ-LIST FINAL))                             ; [6]
); Đóng defun NAMES
;
;
(NAMES GRP "Group")                                                ; [7]
(NAMES MLS "Mline")                                                ; [8]
(setq OLD (strcase (getstring "\nName to change: ")))
(while (not (member OLD FINAL))                                    ; [9]
  (princ "\nIncorrect name, please try again . . .")
  (setq OLD (strcase (getstring "\nName to change: ")))
) ; Đóng while
(setq NEW (strcase (getstring "\nNew name: ")))
(if (null (dictrename GRP-E OLD NEW))                              ; [10]
  (dictrename MLS-E OLD NEW)
); Đóng if
(princ)
); Đóng defun
;Kết thúc chương trình
```

Giải thích:

[1] (setq GRP-E (cdr (assoc –1 GRP))) Biến GRP chứa đối tượng chính của nhóm Group trong từ điển đối tượng. Biểu thức *(cdr (assoc –1 GRP))* trả về mã đối tượng chính của nhóm Group, và mã này được gán cho biến GRP-E.

[2] (setq MLS-E (cdr (assoc –1 MLS))) Biến MLS chứa đối tượng chính của nhóm kiểu Mline trong từ điển đối tượng. Biểu thức *(cdr (assoc –1 MLS))* trả về mã đối tượng chính của nhóm kiểu đường mline, và mã này được gán cho biến MLS-E.

[3] (defun NAMES (S-NAME S-STR) Hàm NAMES được tạo ra để lưu trữ và in tên các Group và các kiểu đường mline. Tham số S-NAME chứa tên đối tượng. Tham số S-STR chứa chuỗi biểu diễn kiểu đối tượng.

[4] (foreach ITEM S-NAME (if (= (car ITEM) 3) Với từng phần tử trong danh sách S-NAME, nếu mã DXF bằng 3 (tên của Group hoặc kiểu đường Mline) thì thêm tên này vào danh sách OBJ-LIST. Danh sách này sẽ chứa tất cả các tên cần in ra màn hình.

[5] (foreach ITEM OBJ-LIST (princ (strcat "\t" ITEM "\n"))) Với từng phần tử trong danh sách OBJ-LIST, in ra màn hình một khoảng TAB ("\t"), tên Group hoặc mline (ITEM), và xuống dòng ("\n").

[6] (setq FINAL (append OBJ-LIST FINAL)) Mỗi lần gọi hàm NAMES, danh sách OBJ-LIST lại bị gán bằng rỗng. Do đó, tên các Group hoặc mline cần được lưu lại trong danh sách FINAL để dùng về sau.

[7] (NAMES GRP "Group") In tên các Group lên màn hình.

[8] (NAMES MLS "Mline") In tên các mline lên màn hình.

[9] (while (not (member OLD FINAL)) (princ "\nIncorrect. . . Biến OLD chứa tên Group hoặc mline do người sử dụng nhập vào. Nếu tên này không có trong danh sách FINAL (do người sử dụng nhập sai), thì in thông báo yêu cầu nhập lại.

[10] (if (null (dictrename GRP-E OLD NEW)) (dictrename MLS-E OLD NEW) Nếu việc đổi tên Group không thực hiện được (trả về nil), thì thực hiện việc đổi tên kiểu đường mline.

Thực hiện chương trình:

Command: **dren** ↵

AutoCAD Group names:

\- - - - - - - - - - - - - - - - - - -

TRUC
BANHRANG
THEN

AutoCAD Mline names:

\- - - - - - - - - - - - - - - - - - -

TUONG
STANDARD

Name to change: **TRUC** ↵
New name: **TRUCKHUYU** ↵

Hàm Dictadd

Hàm **Dictadd** (*DICTionary ADD*) thêm đối tượng mới (Group hoặc kiểu đường Mline) vào từ điển đối tượng.

(**Dictadd** ENAME SYMBOL NEWOBJ)

Tham số ENAME chứa mã đối tượng chính "*acad_group*" hoặc "*acad_mlinestyle*". NEWOBJ chứa mã đối tượng của đối tượng được thêm vào. SYMBOL chứa dữ liệu của đối tượng này. Nếu giá trị các tham số này không hợp lệ, hàm **Dictadd** trả về nil.

Ví dụ:

Thêm group mới vào từ điển đối tượng. Hàm **Entmakex** (tương tự như hàm **Entmake**) dùng để tạo ra mã đối tượng mới, không trùng với các mã đối tượng đã có.

Command: **GROUP** ↵ Dùng lệnh **Group** tạo 2 group "LINES" và "CIRCLE"

Command: **(setq OD (namedobjdict))** ↵ Mã từ điển đối tượng
<Entity name: 2d90460>

Command: **(setq GROUP-SECTION (dictsearch OD "acad_group"))** ↵
((-1 . <Entity name: 2d90468>) (0 . "DICTIONARY") (5 . "D")
(102 ."{ACAD_REACTORS") (330 . <Entity name: 2d90460>) (102 . "}") (100 . "AcDbDictionary") (280 . 0) (3 . "LINES")
(350 . <Entity name: 2d90510>) (3 ."CIRCLES")
(350 . <Entity name: 2d90518>))

Command: **(setq GROUP-ENAME (cdr (assoc -1 GROUP-SECTION)))** ↵
<Entity name: 2d90468>

Command: **(setq GROUP1 (entget (cdr (assoc 350 GROUP-SECTION))))**↵
((-1 . <Entity name: 2d90510>) (0 . "GROUP") (5 . "22")
(102 ."{ACAD_REACTORS") (330 . <Entity name: 2d90468>) (102 . "}") (100 . "AcDbGroup") (300 . "") (70 . 0) (71 . 1)
(340 . <Entity name: 2d90500>))

Command:(setq NEWENT (entmakex GROUP1)) ↵ (Tạo mã đối tượng mới)
<Entity name: 2d90520>

Command: **(Dictadd GROUP-ENAME "newgroup" NEWENT)** ↵ (Thêm đối tượng mới)
<Entity name: 2d90520>

Command: **(entget GROUP-ENAME)** ↵ (Xem lại kết quả)

((-1 . <Entity name: 2d90468>) (0 . "DICTIONARY") (5 . "D")
(102 ."{ACAD_REACTORS") (330 . <Entity name: 2d90460>) (102 . "}") (100 . "AcDbDictionary") (280 . 0) (3 . "LINES")
(350 . <Entity name: 2d90510>) (3 ."CIRCLES")
(350 . <Entity name: 2d90518>) (3 ."NEWGROUP")
(350 . <Entity name: 2d90520>))

Hàm Dictremove

Hàm **Dictremove** (*DICTionary REMOVE*) loại bỏ một đối tượng của một nhóm trong từ điển đối tượng.

(**dictremove** ENAME SYMBOL)

Tham số ENAME chứa mã đối tượng chính "*acad_group*" hoặc "*acad_mlinestyle*". SYMBOL chứa tên đối tượng cần loại bỏ. Nếu giá trị các tham số này không hợp lệ, hàm **Dictremove** trả về nil.

Ta không thể loại bỏ một kiểu đường mline khi trong bản vẽ đã vẽ các đường mline thuộc kiểu này.

Ví dụ: Loại bỏ kiểu đường mline "M1"

Command: **(setq OD (namedobjdict))** ↵
<Entity name: 2da0460>

Command: **(setq MLINE-SECTION (dictsearch OD "acad_mlinestyle"))**↵
((-1 . <Entity name: 2da0470>) (0 . "DICTIONARY") (5 . "E") (102 . "{ACAD_REACTORS") (330 . <Entity name: 2da0460>) (102 . "}") (100 . "AcDbDictionary") (280 . 0) (3 . "M1") (350 . <Entity name: 2da0538>) (3 . "M4") (350 . <Entity name: 2da0558>) (3 . "STANDARD") (350 . <Entity name: 2da04e0>))

Command: **(setq MLINE-ENAME (cdr (assoc -1 MLINE-SECTION)))** ↵
<Entity name: 2da0470>

Command: **(dictremove MLINE-ENAME "M4")** ↵
<Entity name: 2da0558>

Command: **(setq MLINE-SECTION (dictsearch OD "acad_mlinestyle"))**↵
((-1 . <Entity name: 2da0470>) (0 . "DICTIONARY") (5 . "E") (102 . "{ACAD_REACTORS") (330 . <Entity name: 2da0460>) (102 . "}") (100 . "AcDbDictionary") (280 . 0) (3 . "M1") (350 . <Entity name: 2da0538>) (3 . "STANDARD") (350 . <Entity name: 2da04e0>))

Ví dụ:

```
;Tên file: DICTREM.LSP
;
;Mục đích: Cho phép người sử dụng chọn tên Group hoặc kiểu đường mline
;để loại bỏ khỏi bản vẽ.
```

```
;
(defun C:DREM (/ MASTER GRP MLS GRP-E MLS-E FINAL
                 OBJ-LIST SEL)
  (setvar "cmdecho" 0)
  (setq MASTER   (Namedobjdict)                          ; Mã từ điển
        GRP      (Dictsearch MASTER "acad_group")        ; Group
        MLS      (Dictsearch MASTER "acad_mlinestyle")   ; MlineStyle
        GRP-E    (cdr (assoc -1 GRP))                    ; [1]
        MLS-E    (cdr (assoc -1 MLS))                    ; [2]
        FINAL    '( )                  ; Khởi tạo danh sách FINAL
  ) ; Đóng setq
; - <Định nghĩa hàm NAMES để lưu trữ và in các đối tượng Group/Mline>-
(defun NAMES (S-NAME S-STR)                              ; [3]
  (setq OBJ-LIST '() )                  ; Khởi tạo danh sách dùng để
                                        ;lưu trữ chuỗi
  (foreach ITEM S-NAME                                   ; [4]
    (if (= (car ITEM) 3)
      (setq OBJ-LIST (cons (cdr ITEM) OBJ-LIST))
    ) ; Đóng if
  ) ; Đóng foreach
  (setq OBJ-LIST (reverse OBJ-LIST))    ; Đảo ngược danh sách
  (princ (strcat "\n AutoCAD " S-STR    ; In tiêu đề
       " names: \n - - - - - - - - - - - - - - - - - - -\n"))
  (foreach ITEM OBJ-LIST (princ (strcat "\t" ITEM "\n")))  ; [5]
  (setq FINAL (append OBJ-LIST FINAL))                   ; [6]
); Đóng defun NAMES
;
(NAMES GRP "Group")                                      ; [7]
(NAMES MLS "Mline")                                      ; [8]
(setq SEL (strcase (getstring "\nSelect entry to remove/Exit: ")))
(cond
  ((or (= SEL "E") (= SEL "EXIT"))      ;Nhập "E" hoặc "EXIT" để
                                        ; kết thúc chương trình.
   (princ "\nNo changes at this time")
   )
   (T
     (while (not (member SEL FINAL))                     ; [9]
       (princ "\nIncorrect name, please try again ...")
       (setq SEL (strcase (getstring "\nSelect entry to remove: ")))
      ); Đóng while
```

```
    (if (null (dictremove GRP-E SEL))                    ; [10]
      (dictremove MLS-E SEL)
    )
   ); Đóng T
  ); Đóng cond
  (princ)
)
; Kết thúc chương trình
```

Giải thích:

[1] (setq GRP-E (cdr (assoc –1 GRP))) Biến GRP chứa đối tượng chính của nhóm Group trong từ điển đối tượng. Biểu thức *(cdr (assoc –1 GRP))* trả về mã đối tượng chính của nhóm Group, và mã này được gán cho biến GRP-E.

[2] (setq MLS-E (cdr (assoc –1 MLS))) Biến MLS chứa đối tượng chính của nhóm kiểu mline trong từ điển đối tượng. Biểu thức *(cdr (assoc –1 MLS))* trả về mã đối tượng chính của nhóm kiểu đường mline, và mã này được gán cho biến MLS-E.

[3] (defun NAMES (S-NAME S-STR) Hàm NAMES được tạo ra để lưu trữ và in tên các Group và các kiểu đường mline. Tham số S-NAME chứa tên đối tượng. Tham số S-STR chứa chuỗi biểu diễn kiểu đối tượng.

[4] (foreach ITEM S-NAME (if (= (car ITEM) 3) Với từng phần tử trong danh sách S-NAME, nếu mã DXF bằng 3 (tên của Group hoặc kiểu đường mline) thì thêm tên này vào danh sách OBJ-LIST. Danh sách này sẽ chứa tất cả các tên cần in ra màn hình.

[5] (foreach ITEM OBJ-LIST (princ (strcat "\t" ITEM "\n"))) Với từng phần tử trong danh sách OBJ-LIST, in ra màn hình một khoảng TAB ("\t"), tên Group hoặc mline (ITEM), và xuống dòng ("\n").

[6] (setq FINAL (append OBJ-LIST FINAL)) Mỗi lần gọi hàm NAMES, danh sách OBJ-LIST lại bị gán bằng rỗng. Do đó, tên các Group hoặc mline cần được lưu lại trong danh sách FINAL để dùng về sau.

[7] (NAMES GRP "Group") In tên các Group lên màn hình.

[8] (NAMES MLS "Mline") In tên các mline lên màn hình.

[9] (while (not (member SEL FINAL)) (princ "\nIncorrect. . . Biến SEL chứa tên Group hoặc mline do người sử dụng nhập vào. Nếu tên này không có trong danh sách FINAL (do người sử dụng nhập sai), thì in thông báo yêu cầu nhập lại.

[10] **(if (null (dictremove GRP-E SEL)) (dictremove MLS-E SEL)** Nếu việc loại bỏ Group không thực hiện được (trả về nil), thì thực hiện việc đổi loại bỏ kiểu đường mline.

Thực hiện chương trình:

Command: **drem** ↵
AutoCAD Group names:
- - - - - - - - - - - - - - - - - - -
TRUC
BANHRANG
THEN
AutoCAD Mline names:
- - - - - - - - - - - - - - - - - - -
TUONG
STANDARD
Select entry to remove/Exit: **TRUC** ↵
Command: **drem** ↵
AutoCAD Group names:
- - - - - - - - - - - - - - - - - - -
BANHRANG
THEN
AutoCAD Mline names:
- - - - - - - - - - - - - - - - - - -
TUONG
STANDARD
Select entry to remove/Exit: **Exit** ↵
Command:

15.3 Đối tượng khung nhìn (*viewport)*

Trong **AutoCAD** có 2 loại khung nhìn: khung nhìn tĩnh (*tiled viewport)* và khung nhìn động (*floating viewport*). Khung nhìn tĩnh chỉ tồn tại trong không gian mô hình, khung nhìn động tồn tại trong không gian giấy vẽ. Ta có thể chia màn hình thành nhiều khung nhìn, nhưng tại một thời điểm chỉ có một khung nhìn hiện hành. Mỗi khung nhìn có một *số định danh* do **AutoCAD** quản lý. Ta không thể sửa đổi giá trị này.

Biến hệ thống TILEMODE điều khiển việc sử dụng không gian mô hình hoặc không gian giấy vẽ. Khi biến TILEMODE = 1, ta chỉ sử dụng không gian mô hình. Khi biến TILEMODE = 0, ta có thể sử dụng không gian giấy vẽ.

Biến hệ thống CVPORT chứa số định danh của khung nhìn hiện hành.

Hàm VPORTS

Hàm **Vports** trả về danh sách biểu diễn cách sắp xếp (*layout*) các khung nhìn trên màn hình hiện hành.

(**Vports**)

Giá trị trả về là một danh sách bao gồm nhiều danh sách con. Mỗi danh sách con tương ứng với một khung nhìn, gồm có: *số định danh*, tọa độ góc trái dưới, tọa độ góc phải trên của khung nhìn. Danh sách con đầu tiên tương ứng với khung nhìn hiện hành.

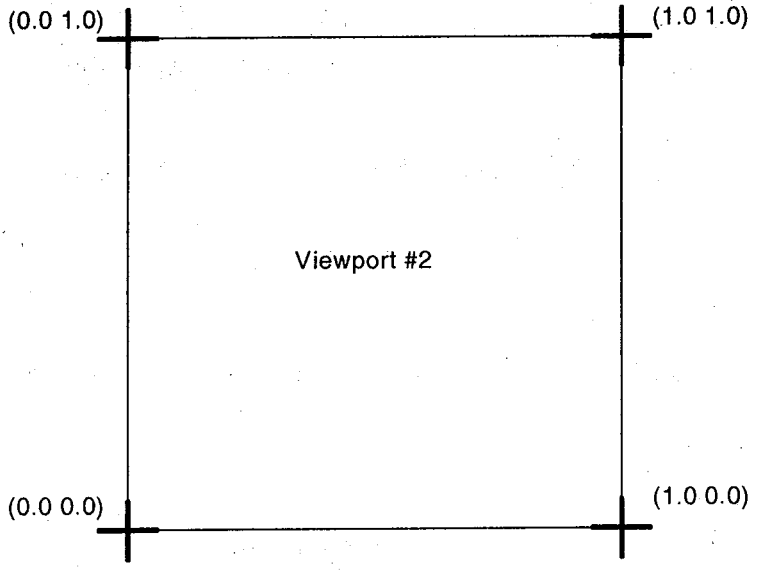
Khung nhìn tĩnh *(Tiled viewport)*

Khi biến TILEMODE = 1, màn hình được xem như là một cửa sổ có tọa độ góc trái dưới là (0.0, 0.0) và tọa độ góc phải trên là (1.0, 1.0). Tọa độ của các khung nhìn được tính tương ứng với hệ thống tọa độ này.

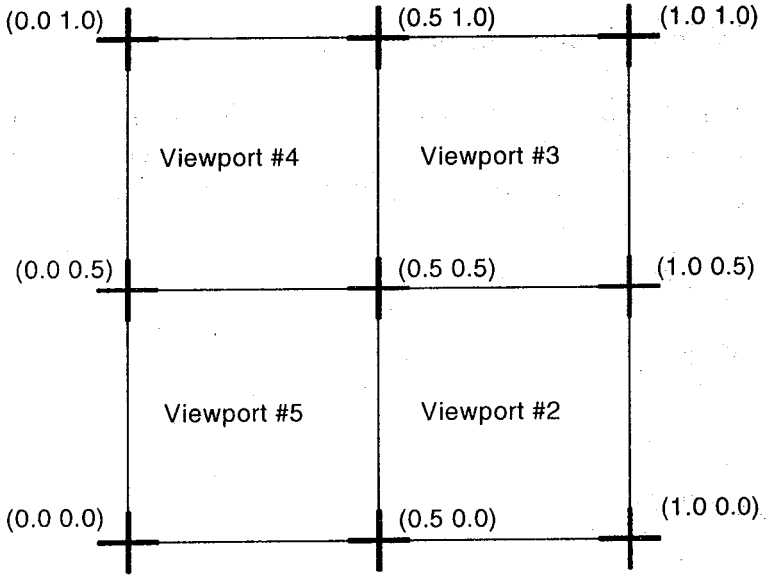
Ví dụ: Màn hình chỉ có 1 khung nhìn

Command: **Vports** ↵
Save/Restore/Delete/Join/SIngle/?/2/<3>/4: **SI** ↵

Command: **(vports)**↵
((2 (0.0 0.0) (1.0 1.0)))



Ví dụ: Màn hình có 4 khung nhìn



Command: **Vports** ↵
Save/Restore/Delete/Join/SIngle/?/2/<3>/4: **4** ↵

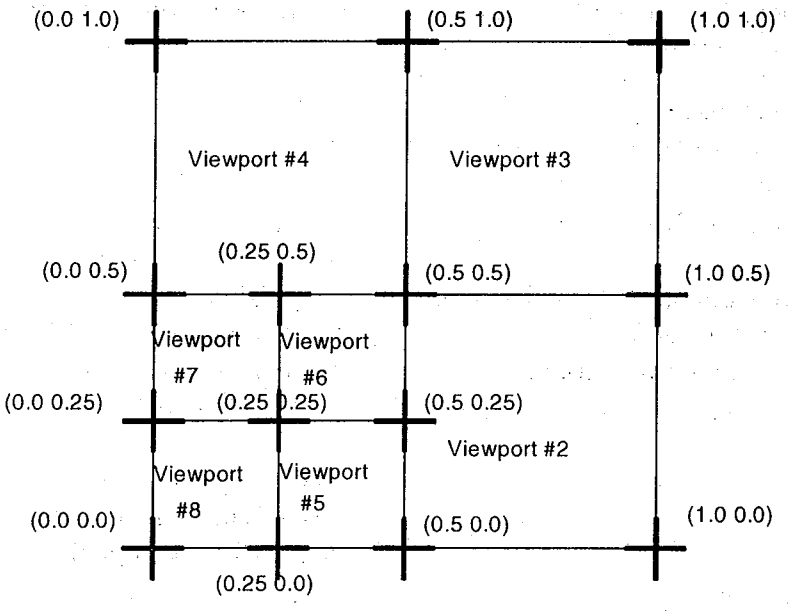
Command: **(vports)** ↵
((2 (0.5 0.0) (1.0 0.5)) (3 (0.5 0.5) (1.0 1.0)) (4 (0.0 0.5) (0.5 1.0)) (5 (0.0 0.0) (0.5 0.5)))

Viết lại cho dễ nhìn:

```
( (2 (0.5 0.0) (1.0 0.5))
  (3 (0.5 0.5) (1.0 1.0))
  (4 (0.0 0.5) (0.5 1.0))
  (5 (0.0 0.0) (0.5 0.5))
)
```

Các khung nhìn có số định danh từ 2 đến 5 và có các tọa độ tương ứng. Khung nhìn số 2 là khung nhìn hiện hành.

Ví dụ: Chia khung nhìn số 5 thành 4 khung nhìn.



Command: **Vports** ↵
Save/Restore/Delete/Join/SIngle/?/2/<3>/4: **4** ↵

```
Command: (vports) ↵
(  (5 (0.25 0.0) (0.5 0.25))
   (2 (0.5 0.0) (1.0 0.5))
   (3 (0.5 0.5) (1.0 1.0))
   (4 (0.0 0.5) (0.5 1.0))
   (6 (0.25 0.25) (0.5 0.5))
   (7 (0.0 0.25) (0.25 0.5))
   (8 (0.0 0.0) (0.25 0.25))
)
```

Tổng cộng có tất cả 7 khung nhìn. Các khung nhìn từ 2 đến 4 không thay đổi. Khung nhìn số 5 được chia thành 4 khung nhìn. Khung nhìn số 5 là khung nhìn hiện hành.

Các khung nhìn tĩnh được lưu trữ trong bảng mô tả "VPORT". Các đối tượng này có thể được truy xuất bằng các hàm truy xuất đối tượng.

Ví dụ:

Chương trình chuyển khung nhìn ở góc trái dưới màn hình thành khung nhìn hiện hành.

```
;Tên file: LLVP.LSP
(defun C:LLVP (/ CTR VP)
  (setq CTR 0
        VP (nth CTR (vports))                ;Khung nhìn đầu tiên
  )
  (while (not (equal (cadr VP) '(0.0 0.0)))  ;Trong khi chưa tìm thấy tọa
    (setq CTR (1+ CTR)                       ; độ góc trái dưới là (0.0 0.0)
          VP (nth CTR (vports))              ; thì chuyển qua khung nhìn
    ); Đóng setq                                    ; kế tiếp.
  ); Đóng while
  (setvar "cvport" (car VP))                 ;Tìm thấy khung nhìn ở góc
  (princ)                                    ; trái dưới. Chuyển nó thành
                                             ; khung nhìn hiện hành
);Kết thúc chương trình
```

Ví dụ: Chương trình cho phép chọn điểm nhìn cho các khung nhìn.

```
;Tên file: PORTS.LSP
```

```
;Mục đích: Cho phép tạo điểm nhìn cho 4 khung nhìn.
;
(defun C:PORTS (/ RES VPLIST PT)
  (setvar "cmdecho" 0)
;
;Tạo một khối vuông 3D với tiêu đề "3D-BOX"
;
  (command ".limits" '(0 0) '(420 297)
           ".zoom" "a"
           ".elev" 0 200
           ".pline" '(50 30) '(250 30) '(250 230) '(50 230) "c"
           ".elev" 200 0
           ".text" "m" '(150 130) 20 0 "3D-BOX"
           ".elev" 0 0
  ); Đóng command
;
;Yêu cầu người sử dụng chọn cấu hình cho 4 khung nhìn:
;       a/ Điểm nhìn đầu tiên là (0 0 1). Tự chọn điểm nhìn cho 3 khung
;          nhìn còn lại.
;       b/ Sử dụng các điểm nhìn theo tiêu chuẩn cho 4 khung nhìn.
;
  (initget "Specify")
  (setq RES (getkword "\nSpecify 3 vpoints/<ENTER> for standard cfg: ")
        VPLIST (list '(0 0 1))          ;Khởi tạo danh sách các điểm nhìn.
  )
;
; Nhập 3 điểm nhìn và tạo thành danh sách điểm nhìn.
;
  (if (= RES "Specify")
    (repeat 3                           ;Thực hiện 3 lần lệnh Vpoint.
      (command ".vpoint" "" pause       ;Dừng lại, sau đó hiển thị điểm nhìn
               ".delay" 500             ;được chọn.
      )
      (setq PT (getvar "viewdir")       ;Tọa độ điểm nhìn hiện hành
            VPLIST (append VPLIST (list PT)) ;Thêm điểm nhìn vào VPLIST
      )
    ); Đóng repeat
   ); Đóng if
  (command ".vports" 4)                 ; Tạo 4 khung nhìn
```

```
; Liên kết các khung nhìn với các điểm nhìn được chọn.
  (mapcar
    '(lambda (VPCFG VPLST)
      (setvar "cvport" (car VPCFG))   ;Duyệt 4 khung nhìn.
      (command ".vpoint" VPLST)       ;Sử dụng các điểm nhìn được
                                      ;cung cấp phía dưới
    )
    (vports)
    (if (=RES "Specify")              ;If "Specify",
      VPLIST                          ; then VPLIST,
      (list '(0 0 1) '(1 -1 1) '(-1 -1 1) '(1 1 1)) ; else các điểm nhìn theo tiêu
                                                    ;chuẩn
    ); Đóng if
  ); Đóng mapcar
  (setvar "cmdecho" 1)
  (princ)
); Đóng defun
;Kết thúc chương trình
```

Khung nhìn động *(Floating viewport)*

Khi biến hệ thống TILEMODE = 0 (không gian giấy vẽ), các khung nhìn động được tạo ra bằng lệnh **Mview**.

Danh sách trả về của hàm **Vports** trong không gian giấy vẽ khác với trong không gian mô hình.

Ví dụ: Chuyển qua không gian giấy vẽ, dùng lệnh **Mview** tạo 3 khung nhìn động như hình vẽ. Sau đó, gọi hàm (**vports**).

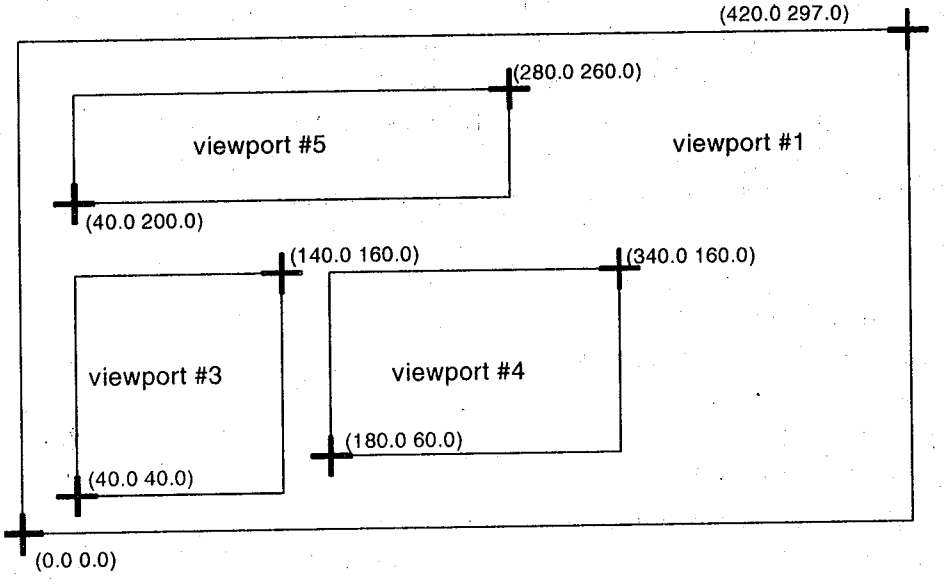
Command: **(vports)** ↵

((1 (0.0 0.0) (420.0 297.0)) (5 (40.0 200.0) (280.0 260.0)) (4 (180.0 60.0) (340.0 160.0)) (3 (40.0 40.0) (140.0 160.0)))

Viết lại cho rõ:

```
( (1 (0.0 0.0) (429.0 297.0))
  (5 (40.0 200.0) (280.0 260.0))
  (4 (180.0 60.0) (340.0 160.0))
  (3 (40.0 40.0) (140.0 160.0))
)
```

Khung nhìn số 1 luôn luôn là màn hình. Tọa độ các điểm góc trái dưới , góc phải trên của các khung nhìn biểu diễn trong hệ trục WCS của không gian giấy vẽ.



Ví dụ:

Chương trình xóa nhanh chóng tất cả các khung nhìn động của không gian giấy vẽ.

```
;Tên file: DELPORTS.LSP
(defun C:DELPORTS (/ PLST)
  (setvar "cmdecho" 0)
  (setq PLST (ssget "X" '((0 . "VIEWPORT"))))
  (if (= (getvar "TILEMODE") 0)
      (command ".erase" PLST "")
      (princ "\nSorry – Not in Paper Space!")
  ); Đóng if
  (setvar "cmdecho" 1)
  (princ)
); Đóng defun
;Kết thúc file DELPORTS.LSP
```

Hàm SETVIEW

Hàm **Setview** *(SET display VIEW)* dùng để gọi hiển thị một phần ảnh *(View)* lên màn hình hoặc khung nhìn.

(**Setview** VIEW_DESCRIPTOR [VPOTRS_ID])

Tham số VIEW_DESCRIPTOR là danh sách chứa tất cả các mã DXF cần thiết để tạo ra một phần ảnh của bản vẽ (*View*). Danh sách này có cấu trúc tương tự như cấu trúc của bảng mô tả "View". Tuy nhiên, giữa chúng có một số điểm khác nhau, ta sẽ xét trong ví dụ sau.

Tham số tùy chọn VPOTRS_ID chứa số định danh của khung nhìn sẽ được hiển thị phần ảnh View. Nếu bỏ qua tham số này, phần ảnh View sẽ hiển thị trên màn hình hoặc khung nhìn hiện hành.

Ví dụ:

Command: **view** ↵ Tạo phần ảnh có tên "View1"
?/Delete/Restore/Save/Window: **W** ↵
View name to save: **View1** ↵
First corner: **20,20** ↵
Other corner: **400,200** ↵

Command: **(tblsearch "view" "View1"))** ↵
((0 . "VIEW") (2 . "VIEW1") (70 . 0) (40 . 180.0) (10 210.0 110.0) (41 . 380.0) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Dữ liệu của View1 được lưu trữ trong bảng mô tả "View".

Command: **(setview (tblsearch "view" "View1"))** ↵
error: AutoCAD rejected function
(SETVIEW (TBLSEARCH "view" "View1"))
Cancel

Hàm **Setview** không chấp nhận chính xác cấu trúc dữ liệu của bảng mô tả "View" (do hàm **Tblsearch** trả về).

Command: **(setq V (tblsearch "view" "View1"))** ↵
((0 . "VIEW") (2 . "VIEW1") (70 . 0) (40 . 180.0) (10 210.0 110.0) (41 . 380.0) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Command: **(setq V (member (assoc 70 V) V))** ↵
((70 . 0) (40 . 180.0) (10 210.0 110.0) (41 . 380.0) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Loại bỏ mã DXF 0 (kiểu đối tượng) và mã DXF 2 (tên đối tượng) trong *record* dữ liệu đối tượng.

Command: **(setview V)** ↵
((70 . 0) (40 . 180.0) (10 210.0 110.0) (41 . 380.0) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Hàm **Setview** chấp nhận cấu trúc này.

Command: **(getvar "cvport")**↵
2

Biến hệ thống CVPORT chứa số định danh của khung nhìn hiện hành.

Command: **(setview V (getvar "cvport"))** ↵
((70 . 0) (40 . 180.0) (10 210.0 110.0) (41 . 380.0) (11 0.0 0.0 1.0) (12 0.0 0.0 0.0) (42 . 50.0) (43 . 0.0) (44 . 0.0) (50 . 0.0) (71 . 0))

Hiển thị phần ảnh View1 lên khung nhìn hiện hành.

Ví dụ:

```
;Tên file: SVIEW.LSP
;
;Mục đích: Chọn View và Viewport. Nếu không có View nào, chương trình
;sẽ hiện thông báo. Nếu có nhiều Viewport trong không gian mô hình,
;chương trình sẽ duyệt từng Viewport, rồi dừng lại cho phép người sử dụng
;chọn, hoặc chuyển qua Viewport khác.
;
(defun C:SV (/ NEXT_VIEW VIEW_SET TM SETS
               NAME NEW_VIEW V_DATA VP)
  (setvar "cmdecho" 0)
  (setq NEXT_VIEW (tblnext "view" T)        ; Phần tử đầu tiên trong bảng
                                            ;mô tả "VIEW"
        VIEW_SET  '()
        TM  (getvar "tilemode")
        SETS  (strcat "Current "                                    ; [1]
              (if (= TM 1) "Model" "Paper")
```

```
                " Space VIEWS:\n")
    ) ; Đóng setq
;
; Định nghĩa hàm DISPLAY để lưu trữ tên của các View
(defun DISPLAY ()
    (while NEXT_VIEW
      (if
        (/= (cdr (assoc 70 NEXT_VIEW)) (getvar "Titlemode"))      ; [2]
        (setq NAME    (cdr (assoc 2 NEXT_VIEW))                   ; [3]
              SETS    (strcat SETS "\n" NAME)                     ; [4]
              VIEW_SET (cons NAME VIEW_SET)                       ; [5]
        ) ; Đóng setq
      ) ; Đóng if
      (setq NEXT_VIEW (tblnext "view"))     ;Chuyển qua View kế tiếp
    ); Đóng while
    (alert SETS)                  ;Hiện hộp thoại chứa tên các View
) ; Đóng DISPLAY
;
;Định nghĩa hàm SELECT1 dùng cho việc chọn một View
;
(defun SELECT1 ()
  (cond
    ( (/= ""                      ;Nếu nhập tên một View (khác Enter)
        (setq NEW_VIEW (strcase
          (getstring "\nView name / <Enter>: "))) ;Yêu cầu nhập tên View
      )
      (while (not (member NEW_VIEW VIEW_SET))                     ; [6]
        (princ "\nIncorrect name, please try again ...")
        (alert SETS)
        (setq NEW_VIEW (strcase (getstring "\nView name: ")))
      ) ; Đóng while
      (setq V_DATA (tblsearch "view" NEW_VIEW)                    ; [7]
            V_DATA (member (assoc 70 V_DATA) V_DATA)              ; [8]
      ); Đóng setq
      (Setview V_DATA)                                            ; [9]
    )
  ); Đóng cond
) ; Đóng SELECT1
;
;
```

```
(cond ( (null NEXT_VIEW)                    ; Nếu không có View nào thì
        (alert "\nNo named VIEWS  exist!") ;thông báo lên màn hình
      )
      ( (= TM 0)                            ; Nếu Tilemode = 0 thì
        (DISPLAY)                           ; thực hiện hàm DISPLAY
        (if VIEW_SET                        ; Nếu VIEW_SET có View để chọn
            (SELECT1)                       ; thì thực hiện hàm SELECT1 để chọn
        ); Đóng if
      )
      ( (= TM 1)                            ; Nếu Tilemode = 1
        (DISPLAY)                           ; thực hiện hàm DISPLAY
        (if VIEW_SET                        ; Nếu VIEW_SET có View để chọn
          (repeat (length (setq VP (vports)))                    ; [10]
            (command ".cvport" (car (car VP)))                   ; [11]
            (SELECT1)              ;Thực hiện hàm SELECT1
            (setq VP (cdr VP))                                   ; [12]
          ) ; Đóng repeat
        ) ; Đóng if
      )
  ) ; Đóng cond
  (princ)
); Đóng SV
;Kết thúc file: SVIEW.LSP
```

Giải thích:

[1] (setq SETS (strcat "Current " (if (= TM 1) "Model" "Paper") " Space VIEWS:\n")) Biến SETS chứa nội dung thông báo của hộp thoại **Alert**. Chuỗi đầu tiên của thông báo này là *"Current Model Space VIEWS: "* hoặc *"Current Paper Space VIEWS: "* tùy theo biến TILEMODE bằng 1 hoặc bằng 0.

[2] (if (/= (cdr (assoc 70 NEXT_VIEW)) (getvar "Titlemode")) Mã DXF 70 của biến NEXT_VIEW chứa cờ báo hiệu View được tạo ra trong không gian mô hình (= 0) hoặc không gian giấy vẽ (= 1). Giá trị này ngược lại với biến TILEMODE (TILEMODE = 1 là không gian mô hình, TILEMODE = 0 là không gian giấy vẽ). Do đó ta sử dụng phép so sánh khác nhau /=.

[3] **(setq NAME (cdr (assoc 2 NEXT_VIEW)))** mã DXF 2 của biến NEXT_VIEW chứa tên của phần ảnh View. Tên này được lấy ra bằng hàm **Cdr**, sau đó được gán cho biến NAME.

[4] **(setq SETS (strcat SETS "\n" NAME))** Thêm tên này vào biến SETS ban đầu. Mỗi lần gọi hàm DISPLAY, biến SETS sẽ được bổ sung thêm tên của các View. "\n" dùng để ngăn cách tên của các View trên các dòng khác nhau.

[5] **(setq VIEW_SET (cons NAME VIEW_SET))** Bổ sung tên của View vào danh sách VIEW_SET.

[6] **(while (not (member NEW_VIEW VIEW_SET)) (princ "\nIncorrect name, please try again ...")** Nếu người sử dụng nhập sai tên View (không có trong danh sách VIEW_SET) thì hiện thông báo và yêu cầu nhập lại cho đến khi nhập đúng.

[7] **(setq V_DATA (tblsearch "view" NEW_VIEW))** Tìm NEW_VIEW trong bảng mô tả "View".

[8] **(setq V_DATA (member (assoc 70 V_DATA) V_DATA))** Chỉ giữ lại các mã DXF bắt đầu từ (assoc 70 V_DATA).

[9] **(Setview V_DATA)** Hàm **Setview** hiển thị phần ảnh View được chọn. Ghi chú: V_DATA đã được loại bỏ mã DXF kiểu đối tượng và tên đối tượng.

[10] **(repeat (length (setq VP (vports)))** Mỗi phần tử trong danh sách do hàm **Vports** trả về tương ứng với một khung nhìn Viewport. Hàm **Length** cho biết số phần tử (số khung nhìn) trong danh sách này. Hàm **Repeat** duyệt từng khung nhìn.

[11] **(command ".cvport" (car (car VP)))** Biểu thức *(car (car VP))* trả về số định danh của khung nhìn sẽ được chuyển thành khung nhìn hiện hành.

[12] **(setq VP (cdr VP))** Chuyển qua *record* dữ liệu kế tiếp. Tiếp tục duyệt.

Tóm tắt

1. Các hàm **Tblnext, Tblsearch** dùng để truy xuất các bảng mô tả của **AutoCAD**. Hàm **Tblnext** duyệt từng đối tượng trong bảng mô tả, hàm **Tblsearch** tìm trực tiếp một đối tượng trong bảng mô tả. Các bảng mô tả gồm có:

"APPID"	"STYLE"
"BLOCK"	"UCS"
"DIMSTYLE"	"VIEW"
"LAYER"	"VPORT"
"LTYPE"	

2. Hàm **Wcmatch** dùng để so sánh một chuỗi với các ký tự đại diện.
3. Hàm **Dictnext** và **Dictsearch** trả về mã đối tượng tìm thấy trong từ điển đối tượng. Các hàm **Dictrename**, **Dictadd**, và **Dictremove** dùng để đổi tên, thêm và xóa đối tượng trong từ điển đối tượng.
4. Hàm **Vports** trả về danh sách biểu diễn cách sắp xếp (*layout*) các khung nhìn trên màn hình hiện hành. Mỗi khung nhìn được biểu diễn bằng một số định danh, tọa độ góc trái dưới và tọa độ góc phải trên.
5. Hàm **Setview** dùng để gọi hiển thị một phần ảnh *(View)* lên màn hình hoặc khung nhìn.

15.4 Bài tập

1. Viết chương trình VIEWREP.LSP hiển thị các thông tin sau:
a/ Liệt kê các khung nhìn View có trong bản vẽ.
b/ Đối với mỗi khung nhìn, hiển thị tên và chiều cao khung nhìn.
c/ Hiện tiêu đề cho từng phần thông tin.

2. Viết chương trình SAVEFURN.LSP, duyệt tất cả các đối tượng trong các bảng mô tả "LAYER" và "BLOCK". Tạo các danh sách chứa các đối tượng này, ngoại trừ các đối tượng có tên bắt đầu bằng "FURN*". In các danh sách này lên màn hình.

3. Viết chương trình P15-3 thực hiện các công việc sau:
a/ Vẽ một hình chữ nhật có chiều dài bằng 150, chiều rộng bằng 100 (sử dụng các hàm **Getpoint** và **Polar**). Tất cả các đỉnh của hình chữ nhật được gán cho các biến. Hình chữ nhật được vẽ ở lớp "0".
b/ Dùng hàm **Tblsearch** để kiểm tra đã có lớp bản vẽ "DIM" hay chưa. Nếu chưa có, chương trình sẽ tạo ra lớp "DIM" có màu đỏ. Nếu có rồi, chuyển thành lớp hiện hành.
c/ Dùng hàm DIM để ghi kích thước dài và rộng cho hình chữ nhật.

4. Viết chương trình BLK-BOM lần lượt thực hiện các công việc sau:

a/ Gán đối tượng đầu tiên trong bảng mô tả "BLOCK" cho biến BL-ITEM (dùng hàm **Tblnext**).

b/ Nếu không có khối nào được định nghĩa trong bảng mô tả "BLOCK" (dùng hàm **Null**), thì thông báo lên màn hình (dùng hàm **Princ**). Nếu có, thực hiện các việc sau:

- Lấy ra từng tên định nghĩa khối trong bảng mô tả (**Cdr** của mã DXF 2)
- Tạo tập hợp chọn tất cả các khối chèn trong bản vẽ trùng với tên này (dùng hàm **Ssget**)
- Đếm số đối tượng trong tập hợp (dùng hàm **Sslength**)
- In tên khối và số lượng lên màn hình (dùng hàm **Princ** và **Strcat**)

15.5 Lời giải

1.

```
;Tên file: VIEWREP.LSP
;Mục đích: Hiển thị tên và chiều cao của các khung nhìn View trong bản vẽ
;
(defun C:VIEWREP (/ VIE VNAME VHEIGHT NAME)
  (setvar "cmdecho" 0)
  (setq VIE (tblnext "View" T) LST '() )
  (while VIE
    (setq VNAME   (cdr (assoc 2 VIE))
          LST     (cons VNAME LST)
          VHEIGHT (rtos (cdr (assoc 40 VIE)))
          LST     (cons VHEIGHT LST)
          VIE     (tblnext "View" )
    )                                   ; Đóng setq
  )                                     ; Đóng while
;
  (setq LST (reverse LST) GAP 19 CTR 0 SPA " ")
  (command ".textscr")
  (princ "\n_ _  ViewName _ _ _ _ ViewHeight _ _ _\n")
;
  (while (setq NAME (nth CTR LST))
```

```
    (princ (strcat "\n"
                   NAME
                   (repeat (- GAP (strlen NAME))
                           (setq SPA (strcat " " SPA))
                   )
                   (nth (1+ CTR) LST)
           )                                ; Đóng strcat

     )                                      ; Đóng princ
    ;
    (setq CHECK (+ CTR 2))
    (if (= (gcd CHECK 10) 10) (princ "\n"))
    (if (= (gcd CHECK 30) 30) (getstring "\n- -<Enter> for More- -"))
    (setq SPA " " CTR CHECK)
  )                                         ; Đóng while
;
  (setvar "cmdecho" 1)
  (princ)
)
;Kết thúc file VIEWREP.LSP
```

2.

```
;Tên file: SAVEFURN.LSP
; Mục đích: In tên các lớp bản vẽ và các khối
; (trừ các tên bắt đầu bằng "FURN*")
;
(defun C:SAVEFURN ()
  (setvar "cmdecho" 0)
;
;Định nghĩa hàm NAMES để in tên lên màn hình
  (defun NAMES (TNAME /  ITEM NAME LST)
   (setq ITEM (tblnext TNAME T)      ;đối tượng đầu tiên của bảng mô tả
                                     ;TNAME
       LST '()                       ;Khởi tạo danh sách chứa các đối
                                     ;tượng trong bảng mô tả
   ); Đóng setq
;Tạo danh sách chứa các đối tượng thỏa chuỗi ký tự đại diện.
   (while ITEM
```

```
      (setq NAME  (cdr (assoc 2 ITEM)))          ;Tên đối tượng
      (if (Wcmatch (strcase NAME) "~*FURN*") ; Nếu tên này trùng với chuỗi
         (setq LST (cons NAME LST))              ; ký tự đại diện thì đưa vào
      ); Đóng if                                 ; danh sách
      (setq ITEM  (tblnext TNAME))               ; Chuyển qua đối tượng kế
                                                 ; tiếp
   ); Đóng while
 ;
 ;In danh sách lên màn hình.
   (if (not (null LST))
    (progn
       (setq LST (reverse LST))                  ;Đảo ngược danh sách.
       (foreach ITEM LST                         ;Duyệt từng phần tử trong
         (princ "\n")                            ; danh sách để in
         (princ ITEM)
       ); Đóng foreach
    ); Đóng progn
  ); Đóng if
); Đóng defun NAMES
;
;
    (princ "\nLayer names: ")          ;Gọi hàm NAMES cho bảng mô tả
    (NAMES "layer")                    ; "layer"
    (princ "\nBlock names: ")          ;Gọi hàm NAMES cho bảng mô tả
    (NAMES "Block")                    ;'Block"
;
  (setvar "cmdecho" 1)
  (princ)
); Đóng defun
;Kết thúc file SAVEFURN.LSP
```

3.

```
;Tên file P15-3.LSP
(defun P15-3 (/ PT1 PT2 PT3 PT4)
  (setvar "cmdecho" 0)
;
;Vẽ hình chữ nhật
  (setq PT1 (getpoint "\nSelect first point: ")
      PT2 (Polar PT1 0 150)
```

```
        PT3 (polar PT2 (/ pi 2) 100)
        PT4 (polar PT3 pi 150)
   )
   (command ".pline" PT1 PT2 PT3 PT4 "C")
   (command ".change" "L" "" "P" "LA" "0" "")
;
;Kiểm tra lớp bản vẽ "DIM" đã có chưa và chuyển thành lớp hiện hành.
   (if (tblsearch "layer" "dim")
      (command "-layer" "S" "DIM" "")
      (command "-Layer" "M" "DIM" "")
   )
;
;Ghi kích thước
  (command ".dimlinear" PT1 PT2 (polar PT2 (- (/ pi 2)) 10)
           ".dimlinear"  PT2 PT3 (polar PT3 0 10)
  )
;
;
(setvar "cmdecho" 1)
(princ)
); Đóng defun
;Kết thúc file P15-3.LSP
```

4.

```
;Tên file: BLK-BOM.LSP
;In tên và số lượng các block trong bản vẽ.
(defun C:BLK-BOM (/ BL-ITEM BLKNAME SS LEN)
   (setvar "cmdecho" 0)
;
;  Phần tử đầu tiên trong bảng mô tả "BLOCK"
   (setq BL-ITEM (tblnext "BLOCK" T))
;
; Duyệt bảng mô tả "BLOCK" và in tên, số lượng từng block
;
   (if (Null BL-ITEM)
      (princ "\nNo block found!")
      (while BL-ITEM
        (setq BLKNAME (cdr (assoc 2 BL-ITEM))
              SS  (ssget "X" (list (cons 2 BLKNAME)))
              LEN (SSLength SS)
```

```
        )
        (princ (strcat "\nBlock " BLKNAME " : " (itoa LEN)))
        (setq BL-ITEM (tblnext "BLOCK"))
      ); Đóng while
    ); Đóng if
;
  (princ)
); Đóng defun
;Kết thúc file BLK-BOM.LSP
```

Chương 16

CÁC HÀM XỬ LÝ MÀN HÌNH ĐỒ HỌA VÀ THIẾT BỊ NHẬP

Nội dung chương:

1. Các hàm chuyển đổi giữa màn hình đồ họa và màn hình văn bản.
2. Các lựa chọn khác nhau khi sử dụng hàm **Redraw** để vẽ lại đối tượng trên màn hình.
3. Gọi hiển thị các menu bằng tên.
4. Xử lý màn hình đồ họa và các thiết bị nhập bằng các hàm có dạng GRxxxx.

Màn hình đồ họa của **AutoCAD** hiển thị rất nhiều thông tin đến cho người sử dụng. Thông tin luôn được cập nhật trên dòng trạng thái. Các menu luôn cung cấp các lệnh thích hợp. Màn hình có thể được sửa đổi sao cho phù hợp với sở thích của người sử dụng và khi cần có thể phục vụ cho yêu cầu của người lập trình.

16.1 Màn hình đồ họa

Hàm TEXTSCR

Hàm **Textscr** (*TEXT SCReen*) sử dụng để chuyển từ màn hình đồ họa sang màn hình văn bản (tương tự phím F2).

(Textscr)

Hàm **Textscr** luôn luôn trả về *nil*. Do đó, nó thường được sử dụng làm biểu thức điều kiện cho các hàm **If**, **Cond**, **While** ...

Hàm TEXTPAGE

Hàm **Textpage,** tương tự như hàm **Textscr**, sử dụng để chuyển từ màn hình đồ họa sang màn hình văn bản (tương tự phím F2). Ngoài ra, đối với phiên bản **AutoCAD** chạy trên hệ điều hành DOS, khi chuyển qua màn hình văn bản, hàm này sẽ xóa toàn bộ nội dung trên màn hình.

(Textpage)

Hàm GRAPHSCR

Hàm **Graphscr** (*GRAPHics SCReen*) sử dụng để chuyển từ màn hình văn bản sang màn hình đồ họa (tương tự phím F2).

(Graphscr)

Hàm **Graphscr** luôn luôn trả về *nil*. Do đó, nó thường được sử dụng làm biểu thức điều kiện cho các hàm **If**, **Cond**, **While** ...

Ví dụ: Chương trình hiển thị các thông tin trạng thái của bản vẽ.

```
;Tên file: STATS.LSP
;
(defun STATS (/ DATE)
  (setvar "cmdecho" 0)
  (setq DATE (rtos (getvar "cdate") 2)          ;Chuyển đổi giá trị ngày hiện
        DATE (strcat (substr DATE 7 2) "_"      ;hành về dạng:
                     (substr DATE 5 2) "_"      ; Ngày_Tháng_Năm
                     (substr DATE 1 4)
```

```
            ); Đóng strcat
  ); Đóng setq
  (Textpage)                          ;Chuyển sang màn hình văn bản
  (mapcar 'princ (list                ; In các thông tin về bản vẽ.
                  "\nDrawing info ...\n\n"
                  "Current Date:      " DATE "\n"
                  "Drawing Name:      " (getvar "Dwgname") "\n"
                  "Drawing Prefix:    " (getvar "Dwgprefix") "\n"
                  "Directory Path:    " (getvar "Acadprefix") "\n\n"
                  "L. L. Corner:      " (getvar "Limmin") "\n"
                  "U. R. Corner:      " (getvar "Limmax") "\n"
                 "The Last Point:     " (getvar "Lastpoint") "\n\n"
                  "Current Layer:     " (getvar "Clayer") "\n"
                  "Drawing Color:     " (getvar "Cecolor") "\n"
                  "Drawing LType:     " (getvar "Celtype") "\n\n"
                  "Text Style Name:   " (getvar "Textstyle") "\n"
                  "Current Menu:      " (getvar "Menuname") "\n\n"
                 ) ; Đóng list
  ); Đóng mapcar
  (getstring "Press Enter to continue: ")
  (graphscr)                          ;Chuyển về màn hình đồ họa.
  (setvar "cmdecho" 1)
  (princ)
); Đóng defun
;Kết thúc chương trình
```

Hàm REDRAW

Hàm **Redraw** sử dụng để làm nổi bật (*highlight*) hoặc vẽ lại (*redraw*) các đối tượng được chọn trên màn hình.

(**Redraw** [ENAME [MODE]])

Hàm **Redraw** không tham số sử dụng để xóa các dấu *blipmode* hoặc các véc tơ tạm (tạo ra bằng các lệnh **Grdraw, Grvecs**) trên màn hình.

Tham số ENAME là mã của đối tượng được chọn. Tham số MODE xác định chức năng hoạt động của hàm **Redraw**. Các giá trị sử dụng cho tham số MODE như sau:

Mode	Chức năng
1	Hiện đối tượng lên màn hình.
2	Không hiện đối tượng lên màn hình.
3	Làm nổi bật đối tượng bằng đường nét đứt (nếu đối tượng này đang hiện trên màn hình)
4	Đưa đối tượng trở về trạng thái bình thường (ngược với mode 3).

Ví dụ:

(redraw)	Xóa các dấu *Blipmode* trên màn hình.
(setq X (entlast))	Gán đối tượng được tạo ra cuối cùng cho biến A.
(redraw X 1)	Hiện đối tượng gán trong biến A lên màn hình. Quá trình này diễn ra rất nhanh, rất khó phát hiện, đặc biệt đối với một nhóm nhiều đối tượng (đối tượng *group*).
(redraw X 2)	Che đối tượng gán trong biến A, không xuất hiện trên màn hình. Đối tượng dường như đã bị xóa đi.
(redraw X 3)	Làm nổi bật đối tượng trong biến A Đối tượng sẽ được vẽ bằng đường nét đứt.
(redraw X 4)	Chuyển về trạng thái bình thường.

Ví dụ: Chương trình làm nhấp nháy đối tượng *Block* trên màn hình.

```
;Tên file: BLKBLINK.LSP
(defun C:BLKBLINK (/ RES SET BLK)
  (setvar "cmdecho" 0)
```

```
(setq RES (getstring "\nHigh last insert of which block: "))
(if (null (setq SET (ssget "X" (list (cons 2 RES))))) ; Tạo SET
  (princ "\nSorry – No block with that name was found!")
  (progn
    (setq BLK (ssname SET 0))      ;
    (repeat 3
      (redraw BLK 3)
      (command ".delay" 50)
      (redraw BLK 4)
      (command ".delay" 50)
    ) ; Đóng repeat
  ) ; Đóng progn
) ; Đóng if
(setvar "cmdecho" 1)
(princ)
); Đóng defun
;Kết thúc chương trình
```

16.2 Gọi hiển thị các menu

Hệ thống menu của **AutoCAD** bao gồm nhiều menu khác nhau. Các menu này được mô tả trong các *menu file*. Mỗi *menu file* chia thành nhiều đoạn (*section*) tương ứng với các menu thành phần của hệ thống. Mỗi *section* bắt đầu bằng một nhãn (*label*) có dạng ****section_name*.

Nhãn	Menu
***MENUGROUP	Tên của *menu group*
***BUTTONSn	Các *Button menu*. Đây là các menu xuất hiện khi nhấn các nút của các thiết bị trỏ (khác thiết bị con chuột).
***AUXn	Các *Auxiliary menu*. Đây là các menu xuất hiện khi nhấn các nút của con chuột (*mouse*).
***POPn	Các *Pull-down menu* và *Shortcut menu*. Các *Pull-down menu* xuất hiện khi ta chọn một mục trên thanh menu. *Shortcut menu* xuất hiện tại vị trí con trỏ trên màn hình.

***TOOLBARS	Các thanh công cụ (*Toolbar*).
***IMAGE	Các menu hình ảnh (*Image tile menu*)
***SCREEN	Menu màn hình (*Screen Menu*).
***TABLETn	Các *Tablet menu*. Đây là các menu sử dụng cho thiết bị nhập là bàn đồ họa (*digitizing tablet*).
***HELPSTRINGS	Chuỗi xuất hiện ở thanh trạng thái khi một mục của *Pull-down menu* hoặc *Shortcut menu* được chọn hoặc khi con trỏ chuột đang kéo ngang qua nút lệnh của thanh *Toolbar*.
***ACCELERATORS	Các phím tắt.

Mỗi menu có một tên gọi (*menu name*). Ta có thể sử dụng hàm **Menucmd** để gọi hiển thị các menu thông qua tên của chúng.

Hàm MENUCMD

Hàm **Menucmd** (*MENU CoMmanD*) sử dụng để làm xuất hiện các menu trên màn hình. Hàm này luôn luôn trả về nil. Nếu menu muốn gọi đến không có trong menu file, hàm **Menucmd** vẫn trả về nil mà không thông báo lỗi.

(**Menucmd** STRING)

Tham số STRING có dạng: *"menu_area = value"*

Menu area	Menu section
B1 - B4	Các *Button menu* từ 1 đến 4
P0 - P16	Các *Pull-down menu* từ 0 đến 16
I	Các menu hình ảnh
S	Menu màn hình.
T1 - T4	Các *Tablet menu.*
A1 - A4	Các *Auxiliary menu* từ 1 đến 4
M	Các biểu thức chuỗi DIESEL
Gmenugroup.nametag	Xác định bằng tên của *menu group* và tên của menu.

Ví dụ:

(menucmd "S=EDIT")	Làm xuất hiện menu EDIT tại vùng menu màn hình.
(menucmd "B1=BUTTON4")	Kích hoạt menu "BUTTON4" tương ứng với nút nhấn số 1 (B1) của thiết bị trỏ.
(menucmd "T1=BLOCKS")	Gán menu "BLOCKS" vào vùng số 1 của bàn đồ họa. Như vậy ta có thể thay đổi các menu thích hợp cho các vùng của bàn đồ họa khi cần thiết.
(menucmd "P5=LAYERS") (menucmd "P5=*")	Gán menu "LAYERS" vào vị trí của *Pull-down menu* thứ 5. Sau đó làm xuất hiện menu này. Dấu "*" được sử dụng để gọi xuất hiện các *Pull-down menu* và các menu hình ảnh.
(menucmd "P0=*")	Làm xuất hiện *Shortcut menu* tại vị trí con trỏ. Mặc định ban đầu là menu "SNAP" (chứa các chức năng truy bắt đối tượng).
(menucmd "P0=DRAW") (menucmd "P0=*")	Gán menu "DRAW" vào vị trí P0. Sau đó hiển thị menu này. Để ý rằng trên màn hình xuất hiện *Shortcut menu* chứa các lệnh vẽ của menu "DRAW".
(menucmd "P3.1=#?")	Trả về trạng thái của mục menu thứ 1 của *Pull-down menu* thứ 3. Mặc định ban đầu đó là mục menu "REDRAW" của menu "VIEW". • Nếu mục này có thể chọn được (*enabled*) giá trị trả về là: "P3.1=" • Nếu mục này không thể chọn được (*disabled*) giá trị trả về là: "P3.1=~". Các mục menu được đánh số thứ tự từ trên xuống (tính cả dấu gạch phân cách). Mục nào có menu con thì được tính tiếp tục các mục con

trước khi chuyển xuống mục kế tiếp.
Ví dụ:
P3.14 là mục "In" của menu con của mục "Zoom" của menu "VIEW".
P7.10 là mục "Polyline" của menu "DRAW".

(menucmd "P7.10=") Chuyển trạng thái mục menu P7.10 ("Polyline") thành *chọn được* (*enabled*)

(menucmd "P7.10=~") Chuyển trạng thái mục menu P7.10 ("Polyline") thành *không chọn được* (*disabled*).

Kể từ **AutoCAD release 12** có bổ sung ngôn ngữ DIESEL (*Direct Interpretively Evaluated String Expression Language*). Các biểu thức chuỗi DIESEL có thể sử dụng phối hợp với hàm **Menucmd**.

Ví dụ:

Command:(**menucmd** "M=$(edtime,$(getvar,date),DDDD DD MONTH YYYY)")↵

"Wednesday 31 January 2001" Ký tự M cho biết đang sử dụng biểu thức chuỗi DIESEL. Kết quả trả về là ngày tháng hiện hành ở dạng Thứ Ngày Tháng Năm.

Command: **(menucmd "M=$(<,$(getvar,filletrad),10)")**↵

"0" Nếu giá trị chứa trong biến FILLETRAD nhỏ hơn 10 thì trả về "1", ngược lại, thì trả về "0".

Ví dụ:

```
;Tên file: HOLEDRIL.LSP
;
(defun C:HOLEDRIL (/ PT1 SIZE CLAY FAC)
  (setvar "cmdecho" 0)
  (mapcar 'menucmd '("S=OSNAP" "P0=POP0" "P0=*"))          ; [1]
```

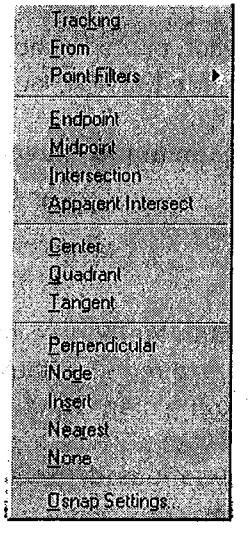
```
  (setq PT1 (getpoint "\nSpecify hole-center location: "))          ; [2]
  (setq SIZE (getreal "\nSpecify size for drill hole: ")            ; [3]
        CLAY (getvar "clayer")                                      ; [4]
  )
  (if (setq FAC (getreal "\nScaling [e.g. 0.25; default = 1]: "))
    (setq SIZE (* FAC SIZE))                                        ; [5]
  )
  (command ".layer" "s" "drill" ""                                  ; [6]
           ".insert" "thehole" PT1 SIZE SIZE 0
           ".layer" "s" CLAY ""
  )
  (menucmd "S=S")                                                   ; [7]
  (setvar "cmdecho" 1)
  (princ)
)
;Kết thúc chương trình
```

Giải thích:

[1] (mapcar 'menucmd '("S=OSNAP" "P0=POP0" "P0=*")) Hàm **Mapcar** cung cấp giá trị tham số cho hàm **Menucmd**. Xuất hiện menu OSNAP trên menu màn hình, và *Shortcut menu* tại vị trí con trỏ.

Dưới đây là *section* POP0 trong *menu file* ACAD.MNU của **AutoCAD** và *Shortcut menu* SNAP tương ứng:

```
***POP0
**SNAP
// Shift-right-click
             [&Object Snap Cursor Menu]
ID_Tracking  [Trac&king]_tracking
ID_From      [&From]_from
ID_MnPointFi [->Point Fi&lters]
ID_PointFilx   [.X].X
ID_PointFily   [.Y].Y
ID_PointFilz   [.Z].Z
             [--]
ID_PointFixy   [.XY].XY
ID_PointFixz   [.XZ].XZ
ID_PointFiyz   [<-.YZ].YZ
             [--]
ID_OsnapEndp [&Endpoint]_endp
ID_OsnapMidp [&Midpoint]_mid
ID_OsnapInte [&Intersection]_int
```



```
ID_OsnapAppa [&Apparent Intersect]_appint
             [--]
ID_OsnapCent [&Center]_cen
ID_OsnapQuad [&Quadrant]_qua
ID_OsnapTang [&Tangent]_tan
             [--]
ID_OsnapPerp [&Perpendicular]_per
ID_OsnapNode [No&de]_nod
ID_OsnapInse [In&sert]_ins
ID_OsnapNear [Nea&rest]_nea
ID_OsnapNone [&None]_non
             [--]
ID_Osnap     [&Osnap Settings...]'_osnap
```

[2] (setq PT1 (getpoint "\nSpecify hole-center location: ")) Yêu cầu nhập vị trí tâm lỗ khoan. Vì menu Object Snap đã xuất hiện trên màn hình, nên người sử dụng có thể chọn nhanh chóng các chức năng truy bắt đối tượng.

[3] (setq SIZE (getreal "\nSpecify size for drill hole: ") Yêu cầu nhập kích thước lỗ khoan.

[4] (setq CLAY (getvar "clayer") Tên lớp bản vẽ hiện hành được giữ lại trong biến CLAY.

[5] (if (setq FAC (getreal "\nScaling [e.g. 0.25; default = 1]: "))

(setq SIZE (* FAC SIZE))

)

Kích thước lỗ khoan được nhân với hệ số tỉ lệ FAC. Nếu người sử dụng nhập ENTER, kích thước lỗ khoan được giữ nguyên.

[6] (command ".layer" "s" "drill" ""

".insert" "thehole" PT1 SIZE SIZE 0

".layer" "s" CLAY ""

)

Gán lớp bản vẽ hiện hành là "drill", chèn khối "thehole" vào đúng vị trí và đúng kích thước. Sau đó, chuyển về lại lớp bản vẽ ban đầu chứa trong biến CLAY.

[7] (menucmd "S=S") Gọi lại menu màn hình trước đó.

Hàm MENUGROUP

Hàm **Menugroup** sử dụng để kiểm tra một *menu group* đã được tải hay chưa.

(**Menugroup** GROUPNAME)

Tham số GROUPNAME chứa tên của *menu group* cần kiểm tra. Nếu *menu group* này đã được tải, kết quả trả về là tên này. Ngược lại, trả về nil.

Ví dụ:

Command: **(menugroup "acad")** ↵ *"acad"*	*menu group* "acad" đã được tải.
Command: **(menugroup "ac_bonus")** ↵ *nil*	*menu group* "ac_bonus" chưa được tải.

16.3 Các hàm truy xuất màn hình đồ họa và thiết bị nhập

Phần này trình bày các hàm **AutoLISP** sử dụng để truy xuất và chỉnh sửa trực tiếp màn hình đồ họa của **AutoCAD** (ví dụ như truy xuất thanh trạng thái) và các thiết bị nhập (như con trỏ và *digitizing tablet*). Mặc dù các hàm này (có dạng GRxxxx) thường không được sử dụng rộng rãi, nhưng lại là lĩnh vực của những nhà lập trình kinh nghiệm.

Hàm GRCLEAR

Hàm **Grclear** (*GRaphics CLEAR*) sử dụng để che các đối tượng trên khung nhìn hiện hành.

(**Grclear**)

Hàm này chỉ có tác dụng đối với release 13 trở về trước. Từ release 14 về sau nó được giữ lại chỉ để tương thích với các phiên bản cũ. Ta có thể sử dụng hàm **Redraw** mode 2 để thay thế.

Ví dụ:

Chương trình che tất cả đối tượng trên màn hình, và hiện dòng chữ BREAK TIME. Khi nhấn Enter. Màn hình sẽ được vẽ lại như ban đầu (chỉ sử dụng cho release 13).

```
;Tên file: OTL.LSP
;
(defun C:OTL ()
  (setvar "cmdecho" 0)
  (setvar "blipmode" 0)
  (Grclear)                                  ;Xóa khung nhìn hiện hành
  (command ".text" "m" (getvar "viewctr")    ;Viết dòng chữ tại tâm màn
                                             ;hình.
        (/ (getvar "viewsize") 10.0)         ;Chiều cao chữ bằng 1/10
                                             ;chiều cao màn hình đồ họa
        0.0                                  ;Độ nghiêng dòng chữ
        "BREAK TIME"                         ;Nội dung dòng chữ
  ) ; Đóng command
  (getstring "\nPress Enter to continue editing: ")
  (command ".erase" "Last" ""
           ".redraw"
  )
  (setvar "cmdecho" 1)
  (princ)
) ; Đóng defun
;Kết thúc file OTL.LSP
```

Ví dụ:

Sửa lại chương trình trên, sử dụng hàm **Redraw** thay cho hàm **Grclear** (sử dụng cho release 14 về sau). Những thay đổi trong chương trình được in đậm để dễ nhận biết.

```
;Tên file: OTL.LSP
;
(defun C:OTL ()
  (setvar "cmdecho" 0)
  (setvar "blipmode" 0)
  (setq ENT (entnext))                  ;Duyệt từng đối tượng và
  (while ENT                            ; sử dụng hàm Redraw mode 2
    (redraw ENT 2)
```

```
    (setq ENT (entnext ENT))
  )
  (command ".text" "m" (getvar "viewctr")    ;Viết dòng chữ tại tâm màn
                                             ;hình.
        (/ (getvar "viewsize") 10.0)         ;Chiều cao chữ bằng 1/10
                                             ;chiều cao màn hình đồ họa
        0.0                                  ;Độ nghiêng dòng chữ
        "BREAK TIME"                         ;Nội dung dòng chữ
  ) ; Đóng command
  (getstring "\nPress Enter to continue editing: ")
  (command ".erase" "Last" "" )
  (setq ENT (entnext))                  ;Duyệt từng đối tượng và
  (while ENT                            ; sử dụng hàm Redraw mode 1
    (redraw ENT 1)
    (setq ENT (entnext ENT))
  )
  (setvar "cmdecho" 1)
  (princ)
) ; Đóng defun
;Kết thúc chương trình
```

Hàm GRDRAW

Hàm **Grdraw** sử dụng để vẽ một véc tơ đi qua 2 điểm trên màn hình trong hệ trục UCS hiện hành.

(**Grdraw** FROM TO COLOR [HIGHLIGHT])

Tham số FROM, TO là tọa độ (2D hoặc 3D) của điểm gốc và điểm ngọn của véc tơ. Các tham số COLOR và HIGHLIGHT xác định màu sắc và dạng đường của véc tơ.

- Véc tơ này không phải là đối tượng của bản vẽ. Ta không thể xóa, sao chép hoặc di chuyển nó được. Các lệnh có tác dụng vẽ lại màn hình như **Redraw**, **Regen**, **Zoom** ... sẽ xóa đi véc tơ này.
- Tham số COLOR chứa số nguyên xác định màu của véc tơ, tương tự như màu của lớp bản vẽ.
- Nếu không có tham số HIGHLIGHT hoặc tham số HIGHLIGHT = 0 thì véc tơ được vẽ bằng dạng đường liên tục. Nếu tham số HIGHLIGHT <> 0 (tham số này phải là một số nguyên), véc tơ được vẽ trên màn hình bằng dạng đường nét đứt.

- Nếu chiều dài véc tơ vượt quá kích thước khung nhìn hiện hành, véc tơ sẽ bị cắt xén 2 đầu cho vừa khít màn hình.

Ví dụ:

(Grdraw '(0 0) '(100 200) 1)	Véc tơ được vẽ trong không gian 2D, có tọa độ điểm gốc và điểm ngọn là (0 0) và (100 200). Véc tơ có màu đỏ.
(Grdraw '(0 0 0) '(100 200 50) 3)	Véc tơ được vẽ trong không gian 3D, có tọa độ điểm gốc và điểm ngọn là (0 0 0) và (100 200 50). Véc tơ có màu xanh lá cây.
(Grdraw '(50 50) '(200 200) 2 –1)	Véc tơ có tọa độ điểm gốc và điểm ngọn là (50 50) và (200 200), màu vàng, và được vẽ bằng đường nét đứt.

Ví dụ:

Chương trình vẽ đường viền dọc theo một đa tuyến (chỉ vẽ các đoạn thẳng lần lượt qua các đỉnh của đa tuyến).

```
;Tên file: CONTOUR.LSP
;
(defun C:CONTOUR (/ ENT CLR SPL PT1 PT2
                    CP1 CP2 VER BEG FINAL)
  (setvar "cmdecho" 0)
;
; Yêu cầu người sử dụng chọn một đường đa tuyến trên màn hình
  (while (/= (cdr
               (assoc 0
                 (setq ENT (entget (car (entsel "\nSelect contour: "))))
               ) ; Đóng assoc
             ); Đóng cdr
         "POLYLINE")       ; Đóng /=
        (princ "\nNot a Contour/Polyline... Try again")
  ); Đóng while
```

```
;
  (initget (+ 1 2 4))
  (setq CLR (getint "\nContour tracing color number: ")          ; [1]
        SPL  (cdr (assoc 70 ENT))                               ; [2]
        ENT (entget (entnext (cdr (assoc -1 ENT))))             ; [3]
        PT1  (cdr (assoc 10 ENT))                               ; [4]
        CP1  (cdr (assoc 70 ENT))                               ; [5]
        ENT (entget (entnext (cdr (assoc -1 ENT))))             ; [6]
        PT2  (cdr (assoc 10 ENT))                               ; [7]
        CP2  (cdr (assoc 70 ENT))                               ; [8]
        VER  (cdr (assoc 0 ENT))                                ; [9]
        BEG PT1                                                 ; [10]
  ); Đóng setq
;
;Định nghĩa hàm VECLOOP để vẽ các véc tơ dọc theo đa tuyến
  (defun VECTLOOP ()
    (while (/= VER "SEQEND")                                    ; [11]
      (if (and                                                  ; [12]
            (/= CP1 16)
            (/= CP2 16)
          ); Đóng and
          (Grdraw PT1 PT2 CLR)                                  ; [13]
      ) ; Đóng if
      (setq PT1 PT2                                             ; [14]
            CP1 CP2
            ENT (entget (entnext (cdr (assoc -1 ENT))))
            PT2  (cdr (assoc 10 ENT))
            CP2  (cdr (assoc 70 ENT))
            VER (cdr (assoc 0 ENT))
      ); Đóng setq
      (if (/= CP1 16)                                           ; [15]
        (setq FINAL PT1)
      ); Đóng if
    ); Đóng while
  ); Đóng defun VECLOOP
;
;
  (cond ( (or
            (= SPL 0)           ;Không phải là đường Spline và là đa
                                ;tuyến mở
            (= SPL 4)           ;Hoặc là đường Spline Spline và là đa
```

```
                                        ;tuyến mở
            )
            (VECTLOOP)                  ;Gọi VECTLOOP để vẽ các véc tơ
          ); Đóng điều kiện 1
          ( (= SPL 1)                   ;Không phải là đường Spline và là đa
                                        ;tuyến đóng
            (VECTLOOP)                  ;Gọi VECTLOOP để vẽ các véc tơ
          (Grdraw BEG FINAL CLR)        ; Tạo véc tơ cuối cùng
        ) ; Đóng điều kiện 2
        ( (= SPL 5)                     ; Là đường Spline và là đa
                                        ;tuyến đóng
          (setq BEG PT2)                ; Đỉnh đầu tiên
          (VECTLOOP)                    ;Gọi VECTLOOP để vẽ các véc tơ
          (Grdraw BEG FINAL CLR)        ; Tạo véc tơ cuối cùng
          ); Đóng điều kiện 3
      ); Đóng cond
;
  (setvar "cmdecho" 1)
  (princ)
); Đóng defun
;Kết thúc file CONTOUR.LSP
```

Giải thích

[1] (setq CLR (getint "\nContour tracing color number: ") Yêu cầu nhập màu cho đường viền và gán cho biến CLR.

[2] (setq SPL (cdr (assoc 70 ENT))) Mã DXF 70 cho biết đặc điểm của đường đa tuyến.

(= SPL 0) *Đa tuyến bình thường và là đa tuyến mở*
(= SPL 1) *Đa tuyến bình thường và là đa tuyến đóng*
(= SPL 4) *Đa tuyến đã được chuyển thành Spline và là đa tuyến mở*
(= SPL 5) *Đa tuyến đã được chuyển thành Spline và là đa tuyến đóng*

[3] (setq ENT (entget (entnext (cdr (assoc –1 ENT))))) Biến ENT chứa *record* dữ liệu của đỉnh đa tuyến đầu tiên.

[4] (setq PT1 (cdr (assoc 10 ENT))) Biến PT1 chứa tọa độ đỉnh đầu tiên của đa tuyến (do mã DXF 10 cung cấp).

[5] (setq CP1 (cdr (assoc 70 ENT))) Giá trị của mã DXF 70 được gán cho biến CP1. Sử dụng để kiểm tra điểm PT1 có là điểm *control point* của đường cong *spline* hay không.

[6] (setq ENT (entget (entnext (cdr (assoc -1 ENT))))) Biến ENT chứa *record* dữ liệu của đỉnh đa tuyến thứ hai.

[7] (setq PT2 (cdr (assoc 10 ENT))) Biến PT2 chứa tọa độ đỉnh thứ hai của đa tuyến (do mã DXF 10 cung cấp).

[8] (setq CP2 (cdr (assoc 70 ENT))) Giá trị của mã DXF 70 được gán cho biến CP2. Sử dụng để kiểm tra điểm PT2 có là điểm *control point* của đường cong *spline* hay không.

[9] (setq VER (cdr (assoc 0 ENT))) Mã DXF 0 chứa tên của đối tượng con trong cơ sở dữ liệu đối tượng. Tên đầu tiên được gán cho biến VER là "VERTEX".

[10] (setq BEG PT1) Sử dụng biến BEG để lưu trữ tọa độ đỉnh đầu tiên để vẽ véc tơ cuối cùng. Biến PT1 sẽ bị thay đổi trong quá trình vẽ các véc tơ.

[11] (while (/= VER "SEQEND") Duyệt các đỉnh của đa tuyến. Khi biến VER chứa tên đối tượng con vẫn còn khác giá trị "SEQEND" thì vẫn còn duyệt tiếp.

[12] (if (and (/= CP1 16)(/= CP2 16)) Nếu PT1 và PT2 đều là các đỉnh của đa tuyến (không phải là các điểm *control point* khi đa tuyến bị chuyển thành *spline*) thì thực hiện biểu thức THEN. Điều này giúp cho các véc tơ chỉ được vẽ qua các đỉnh của đa tuyến mà thôi.

[13] (Grdraw PT1 PT2 CLR) Vẽ véc tơ đi qua 2 điểm PT1 và PT2, có màu là CLR.

[14] Các giá trị trong PT2 và CP2 được chuyển cho PT1 và CP1. Sau đó, PT2 và CP2 lấy giá trị của đỉnh tiếp theo của đa tuyến. Biến VER cũng được gán lại giá trị để kiểm tra đã đạt đến "SEQEND" hay chưa.

[15] (if (/= CP1 16) (setq FINAL PT1)) Nếu CP1 khác 16 thì gán giá trị trong PT1 cho FINAL. Biến FINAL sẽ được cập nhật liên tục trong vòng lặp **While**, cho đến khi kết thúc vòng **While** thì FINAL chứa tọa độ đỉnh cuối cùng của đa tuyến.

Hàm GRVECS

Hàm **Grvecs** (*GRaphics VECtorS*) sử dụng để vẽ nhiều véc tơ trên màn hình cùng lúc.

(**Grvecs** VLIST [TRANS])

Tham số VLIST

([COLOR1] FROM1 TO1 [COLOR2] FROM2 TO2 ...)

Tham số VLIST là một danh sách lần lượt chứa mã màu của các véc tơ và tọa độ các điểm của các véc tơ. Mã màu có giá trị từ 0 đến 255. Nếu ta sử dụng mã màu có giá trị âm, véc tơ sẽ được vẽ bằng đường nét đứt.

Ví dụ:

```
(grvecs '(
          3 (10 10) (100 100)
          2 (150 100) (200 150)
         )          ;đóng VLIST
)                   ;đóng grvecs
```

2 véc tơ được vẽ trên màn hình. Véc tơ thứ 1 có màu là 3 (*green*), đi qua 2 điểm (10 10) và (100 100). Véc tơ thứ 2 có màu là 2 (*yellow*), đi qua 2 điểm (150 100) và (200 150).

```
(grvecs '(
          -5 (20 20) (130 200)
          15 (30 20) (100 250)
         )
)
```

2 véc tơ được vẽ trên màn hình. Véc tơ thứ 1 có màu 5 (*blue*) và được vẽ bằng nét đứt. Véc tơ thứ 2 có màu 15 (*gray*)

```
(grvecs '(
           1 (1 1 0) (50 70 0)
             (20 20 0) (120 120 0)
           6 (10 10 0) (50 200 0)
             (50 100) (100 200)
          )
)
```

4 véc tơ được vẽ trên màn hình. Véc tơ thứ 1 có màu đỏ. Véc tơ thứ 2 cũng có màu đỏ (do đó có thể bỏ mã màu cho véc tơ này). Véc tơ thứ 3 có màu tím. Véc tơ thứ 4 cũng có màu tím (do đó có thể bỏ mã màu cho véc tơ này).

```
(initget 1)
```

Không cho phép nhập giá trị rỗng.

```
(setq COL (getint "\nColor integer: ")
    PT1 (getpoint "\nFirst point: ")
    PT2 (getpoint "\nSecond point: ")
    PT3 (getpoint "\nThird point: ")
)
```

Nhập màu và tọa độ 3 điểm

```
(grvecs (list
          COL PT1 PT2
          COL PT2 PT3
          COL PT3 PT1)
        )
)
```

Sử dụng hàm **List** để tạo danh sách VLIST

Tham số TRANS

Tham số TRANS là một ma trận biến hình, cho phép thay đổi vị trí và tỉ lệ của các điểm trong danh sách VLIST. Ma trận này tương tự ma trận biến đổi tọa độ điểm trong hàm **Nentselp** (xem chương 13).

Ví dụ:

Sử dụng ma trận biến hình sau trong hàm **Grvecs:**

```
'(
  (1.0 0.0 0.0 70.0)
  (0.0 1.0 0.0 70.0)
  (0.0 0.0 1.0 0.0)
  (0.0 0.0 0.0 1.0)
 )
```

Hệ số tỉ lệ là 1.0. Độ dời là 0.0, 70.0, 0.0.

Command: **(grvecs '(7 (0 0) (200 0) (200 0) (200 150) (200 150) (0 150) (0 150) (0 0))) ↵**

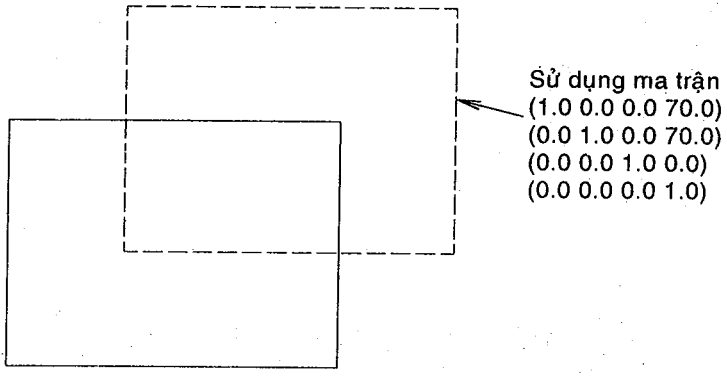
nil

4 véc tơ được vẽ tạo thành hình chữ nhật nét liền.

Command: **(grvecs '(-7 (0 0) (200 0) (200 0) (200 150) (200 150) (0 150) (0 150) (0 0)) '((1.0 0.0 0.0 70.0)(0.0 1.0 0.0 70.0)(0.0 0.0 1.0 0.0)(0.0 0.0 0.0 1.0))) ↵**

nil

4 véc tơ được vẽ tạo thành hình chữ nhật nét đứt. Ma trận biến hình làm dịch chuyển 4 véc tơ này theo độ dời là 0.0, 70.0, 0.0 . Vì hệ số tỉ lệ bằng 1.0, nên 2 hình chữ nhật này bằng nhau.



Ví dụ:

```
;Tên file: GRBOX.LSP
;Mục đích: Yêu cầu người sử dụng nhập vào 4 điểm bất kỳ để vẽ một hình
;         4 cạnh, và chọn màu cho hình này. Ngoài ra, người sử dụng còn
;         nhập vào các giá trị để tạo ra ma trận biến hình, áp dụng khi vẽ
;         hình 4 cạnh.
;
(defun C:GRBOX (/ PT1 PT2 PT3 PT4 COL FCOL
                  X-TRANS Y-TRANS Z-TRANS)
  (setvar "cmdecho" 0)
  (setq PT1 (getpoint "\nFirst corner: ")          ; Nhập 4 đỉnh
        PT2 (getpoint "\nSecond corner: ")
        PT3 (getpoint "\nThird corner: ")
        PT4 (getpoint "\nFourth corner: ")
  )                                                ;Đóng setq
  (command ".line" PT1 PT2 PT3 PT4 "c")            ;Vẽ hình 4 cạnh
;
;Chọn màu cho hình 4 cạnh
  (while
      (setq COL (getint "\nColor integer/<ENTER> when done: "))
      (grvecs (list                                ;Vẽ các véc tơ có màu
                COL PT1 PT2                        ; theo hình 4 cạnh
                COL PT2 PT3
                COL PT3 PT4
```

```
                COL PT4 PT1
              )
        )                                   ;Đóng Grvecs
      (setq FCOL COL)                       ;Màu được chọn
  )                                         ;Đóng While
;
;Nhập ma trận biến hình. Sau đó vẽ hình 4 cạnh thứ hai bằng các véc tơ, độ
;dời so với hình thứ nhất do ma trận biến hình xác định.
;
  (setq X-TRANS (getreal "\nX-transformation value: ")
        Y-TRANS (getreal "\nY-transformation value: ")
        Z-TRANS (getreal "\nZ-transformation value: ")
  )                                         ;Đóng setq
  (grvecs (list
            FCOL PT1 PT2
            FCOL PT2 PT3
            FCOL PT3 PT4
            FCOL PT4 PT1
          )
          (list
            (list 1.0 0.0 0.0 X-TRANS)
            (list 0.0 1.0 0.0 Y-TRANS)
            (list 0.0 0.0 1.0 Z-TRANS)
            (list 0.0 0.0 0.0 1.0)
          )
  )                                         ;Đóng Grvecs
;
  (setvar "cmdecho" 1)
  (princ)
)                                           ;Đóng defun
;Kết thúc chương trình
```

Hàm GRTEXT

Hàm **Grtext** (*GRaphics TEXT*) sử dụng để viết một chuỗi lên dòng trạng thái (*status line*) hoặc lên menu màn hình (*screen menu*).

(**Grtext** [BOX TEXT [HIGHLIGHT])

Tham số BOX

- -1 hoặc -2: Ghi chuỗi lên vị trí *mode* hoặc *coordinate* của dòng trạng thái. Giá trị –2 chỉ sử dụng cho **AutoCAD** chạy trên hệ điều hành DOS.
- Số nguyên > 0 : Ghi chuỗi lên menu màn hình. Giá trị của tham số BOX xác định vị trí mục menu. Vị trí các mục của menu màn hình được đánh số từ 0. Chỉ tiêu đề mục menu hiện lên màn hình bị thay thế. Chức năng của menu vẫn giữ nguyên.

Tham số TEXT

Chuỗi văn bản được ghi lên dòng trạng thái hoặc menu màn hình.

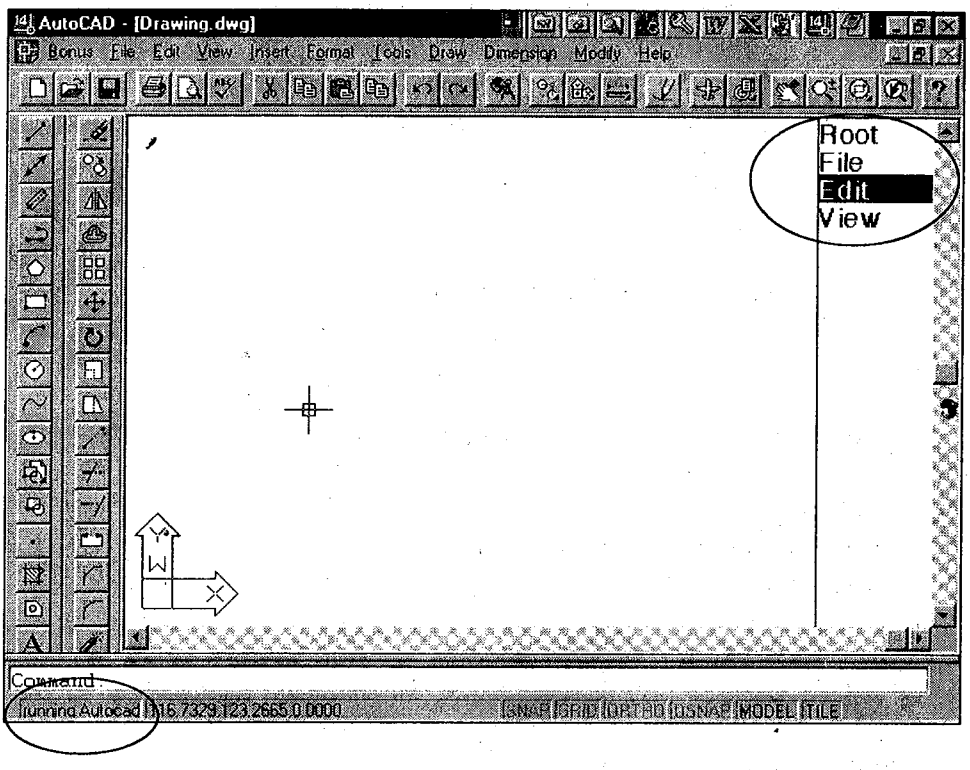
Tham số HIGHLIGHT

Chỉ sử dụng khi ghi chuỗi lên menu màn hình.

- 1: Mục menu BOX được chiếu sáng (*highlight*)
- 0: Mục menu BOX bị loại bỏ việc chiếu sáng (*dehighlight*)

Ví dụ:

(**Grtext** 0 "Root")	Ghi chuỗi "*Root*" lên mục menu thứ 1
(**Grtext** 1 "File")	Ghi chuỗi "*File*" lên mục menu thứ 2
(**Grtext** 2 "Edit")	Ghi chuỗi "*Edit*" lên mục menu thứ 3
(**Grtext** 3 "View")	Ghi chuỗi "*View*" lên mục menu thứ 4
(**Grtext** 2 "" 1)	Chiếu sáng mục menu thứ 3
(**Grtext** –1 "running Autocad")	Ghi chuỗi "*running Autocad*" lên vị trí *mode* của dòng trạng thái.
(**Grtext**)	Hàm **Grtext** không tham số sẽ trả lại các giá trị ban đầu cho dòng trạng thái và menu màn hình. Hàm **Redraw** và **Regen** chỉ có tác dụng đối với màn hình đồ họa, không thể sử dụng cho mục đích này.



Ví dụ:

Chương trình sau đây sử dụng hàm **Grtext** để tạo ra các mục của menu màn hình. Để ý cách sử dụng hàm **Read** để lấy ra tên biến từ một chuỗi, và sử dụng hàm **Set** để lấy giá trị các biến.

```
;Tên file: SCRVARS.LSP
;Mục đích: Chương trình cung cấp một menu màn hình tự tạo. Sau khi nhập
;         tên biến, người sử dụng chọn một hàm AutoLISP thích hợp từ
;         menu màn hình.
;
(defun C:SV (/ LST1 LST2 VN CTR BOX FIL)
  (setvar "cmdecho" 0)
  (setq LST1 '("getInt" "getReal" "getString" "getPoint" "getDist"      ;[1]
               "getAngle" "ssGet" "entSel" "nentSel" "findFile"
               "loadFile" "wordProc")
        LST2 '( (getint      (strcat "\nInteger value for " VN ": "))
```

```
                (getreal    (strcat "\nReal number value for " VN ":"))
                (getstring  T (strcat "\nText string value for " VN ": "))
                (getpoint   (strcat "\nPoint value for " VN ": "))
                (getdist    (strcat "\nDistance value for " VN ": "))
                (getangle   (strcat "\nAngle value for " VN ": "))
                (ssget)
                (entsel)
                (nentsel)
               )                                              ;đóng LST2
        VN     (getstring "\nVariable Name: ")
        CTR  3                                                ; [2]
    )
;
    (grtext 0 "-Quick-") (grtext 1 "AutoLISP" )  (grtext 2 "- - - - - - - -") ; [3]
;
    (repeat 12
       (grtext CTR (nth (- CTR 3) LST1))                      ; [4]
       (setq CTR (1+ CTR))
    )                                                         ; Đóng repeat
;
    (grtext 15 "- - - - - - - -") (grtext 16 "PrevMenu") (setq CTR 17)    ; [5]
;
    (repeat (- (getvar "screenboxes") 16)                     ; [6]
      (grtext CTR " ")
      (setq CTR (1+ CTR))
    )                                                         ; Đóng Repeat
;
    (setq BOX (cadr (grread)))                                ; [7]
    (cond
      ( (wcmatch (itoa BOX) "[0-2], 15")                      ; [8]
        (princ "\nInvalid box - exit function.")
        (grtext)
      )                                                       ; Đóng Test 1
      ( (= BOX 12)                                            ; [9]
        (setq FIL  (getstring "\nFile to search for: "))
        (set (read VN) (findfile FIL))
        (grtext)
      )                                                       ; Đóng Test 2
      ( (= BOX 13)                                            ; [10]
        (setq FIL  (getstring "\nAutoLISP File to search load: "))
        (load FIL)
```

```
      (grtext)
    )                                              ;Đóng Test 3
    ( (= BOX 14)                                          ; [11]
      (setq FIL  (getstring "\nFile to edit: "))
      (command "edit" FIL)
      (grtext)
    )                                              ;Đóng Test 4
    ( (= BOX 16)                          ;Nếu BOX = 16 thì xóa menu
      (grtext)
    )                                              ; Đóng Test 5
    ( T                                                   ; [12]
      (setq BOX (nth (- BOX 3) LST2))
      (set (read VN) (eval BOX))
      (grtext)
    )                                              ; Đóng T
  )                                                ; Đóng cond
;
  (princ)
)                                                  ; Đóng Defun
;Kết thúc chương trình
```

Giải thích:

[1] (setq LST1 ... LST2 ...) Danh sách LST1 chứa tên một số hàm **AutoLISP**. Tên các hàm này sẽ được ghi lên menu màn hình bằng hàm **Grtext**. Danh sách LST2 chứa các dòng nhắc tương ứng với các hàm trong LST1. Để ý rằng các danh sách LST2 là danh sách không định giá trị. Các phần tử của nó sẽ được định giá trị ở phần sau của chương trình.

[2] (setq CTR 3) 3 mục đầu tiên của menu (được đánh số thứ tự là 0, 1, 2) sử dụng để hiện các dòng tiêu đề. Do đó, biến CTR được gán bằng 3, để chỉ đến mục thứ 4 (có số thứ tự là 3) của menu khi sử dụng trong hàm **Repeat**.

[3] (grtext 0 "-Quick-") (grtext 1 "AutoLISP") (grtext 2 "- - - - - - - -") Hàm **Grtext** ghi các tiêu đề "-Quick-", "AutoLISP", "- - - - - - - -" lên 3 mục đầu tiên 0, 1, 2 của menu màn hình.

[4] (repeat 12 (grtext CTR (nth (- CTR 3) LST1)) Danh sách LST1 chứa 12 tên các hàm **AutoLISP**, do đó hàm **Repeat** phải lặp lại 12 lần để lấy ra các tên hàm. Biến CTR đang được gán bằng 3. Do đó, (- CTR 3)

bằng 0, chỉ đến phần tử đầu tiên trong LST1. Hàm **Grtext** lấy ra tên hàm tại vị trí thứ (- CTR 3) trong danh sách LST1 và ghi lên mục thứ CTR trên menu.

[5] **(grtext 15 "- - - - - - - -") (grtext 16 "PrevMenu") (setq CTR 17)** Hàm **Grtext** ghi các chuỗi "- - - - - - - -" và "PrevMenu" lên các mục menu thứ 15 và 16. Sau đó, CTR được gán bằng 17, để chuyển qua mục menu kế tiếp.

[6] **(repeat (- (getvar "screenboxes") 16) (grtext CTR " ")** Biến hệ thống "screenboxes" của **AutoCAD** chứa số thứ tự mục menu lớn nhất có thể có của menu màn hình. Lấy giá trị này trừ cho 16 mục menu đã sử dụng sẽ cho biết số lượng các mục menu còn lại. Hàm **Repeat** sẽ duyệt lần lượt các mục menu này để hàm **Grtext** ghi lên các chuỗi rỗng. Quá trình ghi này bắt đầu từ ô thứ 17 (vì CTR = 17).

[7] **(setq BOX (cadr (grread)))** Hàm **Grread** (sẽ được trình bày ở phần sau) sẽ đọc giá trị nhập vào từ thiết bị nhập và trả về một danh sách chứa 2 giá trị. Giá trị thứ nhất cho biết loại thiết bị nhập. Giá trị thứ hai cho biết số thứ tự của mục menu màn hình. Hàm **Cadr** lấy ra giá trị thứ hai và lưu trong biến BOX.

[8] **(wcmatch (itoa BOX) "[0-2], 15")** Chuyển số nguyên chứa trong biến BOX sang kiểu chuỗi. Hàm **Wcmatch** so sánh chuỗi này với chuỗi ký tự đại diện. Nếu chuỗi này trùng với "0", "1", "2", hoặc "15" thì người sử dụng đã chọn mục menu không phù hợp, do đó hiện thông báo lỗi.

[9] **(= BOX 12) (setq FIL (getstring "\nFile to search for: "))** Khi người sử dụng chọn mục menu thứ 12, thì yêu cầu nhập tên file để tìm. Sau đó hàm **Read** sẽ đọc giá trị chứa trong biến VN, và hàm **Set** sẽ gán giá trị này cho kết quả trả về của hàm **Findfile**.

[10] **(= BOX 13)(setq FIL (getstring "\nAutoLISP File to search load: "))** Khi người sử dụng chọn mục menu thứ 13, thì yêu cầu nhập tên file để tải vào. Nếu biến FIL chứa tên file hợp lệ thì tải file này.

[11] **(= BOX 14)(setq FIL (getstring "\nFile to edit: "))** Khi người sử dụng chọn mục menu thứ 14, thì yêu cầu nhập tên file để chỉnh sửa.

[12] **(T (setq BOX (nth (- BOX 3) LST2))** Khi người sử dụng chọn các mục menu khác với các mục trên, thì trừ biến BOX đi 3 vị trí để có được vị trí tương ứng trong danh sách LST2. Sử dụng hàm **Read** để đọc tên biến chứa trong biến VN, và gán nó cho giá trị trả về của biểu thức (eval BOX).

Hàm GRREAD

Để nhập dữ liệu cho chương trình ta có thể sử dụng các thiết bị nhập sau:

- Bàn phím (*Keyboard*): sử dụng để nhập ký tự.
- Thiết bị chuột (*Mouse*): sử dụng để nhập tọa độ điểm hoặc chọn lệnh trên menu.
- Bàn đồ họa (*Digitizing tablet*): thường được sử dụng để nhập các giá trị tọa độ X, Y, Z.
- Các thiết bị trỏ khác (*Pointing device*): sử dụng để nhập tọa độ điểm hoặc chọn lệnh trên menu.

Hàm **Gread** (*GRaphics READ*) sử dụng để nhận giá trị nhập từ các thiết bị nhập.

(**Grread** [TRACK] [ALLKEYS [CURTYPE]])

Hàm này trả về một danh sách gồm 2 phần tử:

- Phần tử thứ nhất cho biết loại thiết bị nhập.
- Phần tử thứ hai hoặc là một số nguyên hoặc là một danh sách tọa độ điểm.

Phần tử thứ nhất		**Phần tử thứ hai**	
Giá trị	**Kiểu thiết bị nhập**	**Giá trị**	**Ý nghĩa**
2	Bàn phím	Các mã ASCII	Mã ASCII của ký tự nhập
3	Điểm	Tọa độ điểm 3D	Tọa độ điểm 3D
4	Menu màn hình hoặc *Pull-down menu* (được chọn bằng chuột hoặc các thiết bị trỏ khác)	0-999 1001-1999 2001-2999 3001-3999 ... 16001-16999	Số thứ tự mục menu của menu màn hình. Số thứ tự mục menu của *Pull-down menu* POP1. Số thứ tự mục menu của *Pull-down menu* POP2. Số thứ tự mục menu của *Pull-down menu* POP3. ... Số thứ tự mục menu của *Pull-down menu* POP16
5	Thiết bị trỏ (chỉ khi sử dụng tham số TRACK)	Tọa độ điểm 3D	Tọa độ điểm khi ở chế độ kéo rê con trỏ (*drag mode*)

6	*Buttons menu*	0-999 1000-1999 2000-2999 3000-3999	Số thứ tự nút nhấn của menu BUTTONS1. Số thứ tự nút nhấn của menu BUTTONS2. Số thứ tự nút nhấn của menu BUTTONS3. Số thứ tự nút nhấn của menu BUTTONS4.
7	*Tablet1 menu*	0-32767	Số thứ tự ô dữ liệu của TABLET1
8	*Tablet2 menu*	0-32767	Số thứ tự ô dữ liệu của TABLET2
9	*Tablet3 menu*	0-32767	Số thứ tự ô dữ liệu của TABLET3
10	*Tablet4 menu*	0-32767	Số thứ tự ô dữ liệu của TABLET4
11	*Auxiliary menu*	0-999 1000-1999 2000-2999 3000-3999	Số thứ tự nút chuột của menu AUX1 Số thứ tự nút chuột của menu AUX2 Số thứ tự nút chuột của menu AUX3 Số thứ tự nút chuột của menu AUX4
12	Nút nhấn thiết bị trỏ (theo sau giá trị 6 và 11)	Tọa độ điểm 3D	Tọa độ điểm 3D

Ví dụ:

Command: **(gread)** ↵
(2 97)

Ký tự "a" (mã ASCII 97) được nhập vào từ bàn phím

Command: **(grread)**↵
(3 (171.027 141.837 0.0))

Tọa độ một điểm được chọn trên màn hình.

Command: **(grread)**↵
(4 5001)

Mục menu thứ 1 của *Pull-down menu* #5 được chọn.

Command: **(grread)**↵
(4 3)

Mục menu thứ 3 của menu màn hình được chọn.

Command: **(grread)**↵
(6 1)

Nút nhấn số 1 của thiết bị trỏ

Command: **(grread)**↵
(9 23)

Ô số 23 của vùng số 3 của bàn đồ họa.

Tham số TRACK

Khi tham số TRACK có giá trị khác nil, thiết bị trỏ chuyển qua chế độ kéo rê, mà không cần nhấn phím để chọn. Trả về tọa độ con trỏ trên màn hình.

Ví dụ:

Command: **(grread T)**↵ *(5 (119.53 35.2212 0.0))*	Tọa độ con trỏ trên màn hình (chế độ kéo rê).

Tham số ALLKEYS

Tham số ALLKEYS chỉ có tác dụng khi tham số TRACK = nil.

Tham số ALLKEYS có các giá trị sau đây:

Giá trị của ALLKEYS	Hàm Grread
1	Trả về tọa độ điểm 3D ở chế độ kéo rê (*drag mode*)
2	Trả về giá trị của tất cả các phím của bàn phím, kể cả các phím mũi tên.
4	Sử dụng tham số CURTYPE để biểu diễn hình ảnh con trỏ trên màn hình.
8	Không hiển thị dòng thông báo lỗi: *error: consol break*
16	Không cho phép sử dụng *Pull-down menu*.

Ví dụ:

Command: **(grread T)**↵ *(5 (119.53 35.2212 0.0))*	Tọa độ con trỏ trên màn hình (chế độ kéo rê).
Command: **(grread T 2)**↵ *(5 (119.53 35.2212 0.0))*	Mặc dù tham số ALLKEYS = 2 để đọc ký tự từ bàn phím, nhưng vì tham số TRACK khác nil, nên hàm **Grread** vẫn trả về tọa độ điểm.
Command: **(gread nil 2)** ↵	Đọc ký tự nhập từ bàn phím. Các

(2 97)

phím sau đây không sử dụng được:
- Ctrl
- Caps Lock
- Pause
- Print Screen
- Alt
- Shift
- Num Lock
- Scroll Lock

Command: **(grread nil 1)**↵
(5 (119.53 35.2212 0.0))

Tương tự như (grread T)

Tham số CURTYPE

Tham số CURTYPE quy định hình thức hiển thị con trỏ trên màn hình. Tham số này chỉ sử dụng được khi tham số ALLKEYS = 4.

Tham số này có các giá trị như sau:

Giá trị của CURTYPE	Hình dạng con trỏ
0	Con trỏ có hình dạng sợi tóc (*crosshairs*)
1	Không hiển thị con trỏ
2	Con trỏ có hình ô vuông chọn đối tượng □

Ví dụ:

(gread nil 4) — Không sử dụng tham số CURTYPE. Con trỏ có dạng sợi tóc

(gread nil 4 0) — Tham số CURTYPE = 0. Con trỏ có dạng sợi tóc

(gread nil 4 1) — Tham số CURTYPE = 1. Con trỏ không xuất hiện trên màn hình. Tuy nhiên vẫn có thể nhận giá trị nhập từ thiết bị trỏ.

(gread nil 4 2) — Tham số CURTYPE = 2. Con trỏ có dạng ô vuông chọn đối tượng.

Chú ý:
Vì các hệ điều hành nhận giá trị nhập từ các thiết bị nhập theo các cách khác nhau, nên hàm **Grread** có thể trả về các kết quả không như mong muốn.

Ví dụ:

Đoạn chương trình sau đây yêu cầu người sử dụng chọn một điểm trên màn hình hoặc nhấn phím ENTER, SPACEBAR hoặc phím chuột phải (đều mang cùng ý nghĩa với phím ENTER). Để ý cách sử dụng hàm **Grread** trong vòng lặp **While**.

```
(princ "\nPick object on target layer (or [ENTER] for current layer): "); [1]
(while                                                                   ; [2]
  (and                                                                   ; [3]
    (setq GRVAL (grread T 4 2))                                          ; [4]
    (/= 3 (car GRVAL))                                                   ; [5]
    (not (equal '(2 13) GRVAL))                                          ; [6]
    (not (equal '(2 32) GRVAL))                                          ; [7]
    (/= 11 (car GRVAL))                                                  ; [8]
))
```

Giải thích:

[1] (princ "\nPick object on target layer (or [ENTER] for current layer): ") Nhắc người sử dụng chọn một đối tượng hoặc nhấn ENTER

[2] (while Vòng lặp sử dụng để kiểm tra giá trị nhập

[3] (and Chỉ cần một biểu thức điều kiện không thỏa mãn, vòng lặp sẽ kết thúc.

[4] (setq GRVAL (grread T 4 2)) Biến GRVAL lưu trữ giá trị do hàm Grread trả về. Vì hàm này có sử dụng tham số TRACK nên nó chuyển qua chế độ *drag mode*. Khi con trỏ di chuyển, hàm này liên tục được gọi. Do đó ta phải sử dụng vòng lặp While để kiểm tra giá trị trả về cho đến khi thỏa điều kiện. Tham số CURTYPE = 2, nên con trỏ có dạng ô vuông chọn đối tượng.

[5] (/= 3 (car GRVAL)) Điều kiện này không thỏa mãn khi người sử dụng chọn một điểm bất kỳ.

[6] (not (equal '(2 13) GRVAL)) Điều kiện này không thỏa mãn khi người sử dụng nhấn ENTER.

[7] (not (equal '(2 32) GRVAL)) Điều kiện này không thỏa mãn khi người sử dụng nhấn SPACEBAR.

[8] (/= 11 (car GRVAL)) Điều kiện này không thỏa mãn khi người sử dụng nhấn phím chuột phải.

Ta có thể sử dụng hàm **Entsel** để yêu cầu chọn đối tượng, nhưng hàm này không có khả năng phân biệt người sử dụng không chọn một điểm hoặc chỉ nhấn phím ENTER, vì trong cả 2 trường hợp hàm **Entsel** đều trả về nil. Sử dụng hàm **Grread** như trong ví dụ trên giúp nhận biết rõ hơn giá trị nhập.

Ví dụ:

Chương trình sau đây sử dụng hàm (**Grread T**) trong vòng lặp **While** để kiểm tra liên tục vị trí con trỏ trên màn hình cho đến khi con trỏ di chuyển đến vị trí mong muốn (phía trong hình chữ nhật nằm ở tâm giới hạn bản vẽ, được vẽ bằng hàm **Grvecs**).

```
;Tên file: SCAN.LSP
;
;Mục đích: Gom các đối tượng được chọn vào giữa giới hạn bản vẽ.
;         Người sử dụng được yêu cầu chọn các đối tượng. Sau đó di
;         chuyển con trỏ vào trong một hình chữ nhật nằm ở tâm giới
;         hạn bản vẽ, để gom các đối tượng vào đó. Hình chữ nhật được
;         vẽ bằng hàm Grvecs, có kích thước bằng 1/10 giới hạn bản vẽ,
;         màu đỏ, nét đứt.
;
(defun C:SCAN(/ CTR SET LST LL UR CPT LR UL PIC ENT PT IT)
  (setvar "cmdecho" 0)
  (setvar "blipmode" 1)
  (setq CTR 0                                   ;Khởi tạo CTR bằng 0
        SET (ssadd)                             ;Khởi tạo tập hợp chọn SET
        LST '( )                                ;Khởi tạo danh sách rỗng LST
        LL  (getvar "Limmin")          ;Góc trái dưới của giới hạn bản vẽ.
        UR  (getvar "Limmax")          ;Góc phải trên của giới hạn bản vẽ
        CPT (list (/ (+ (car LL) (car UR)) 2)  ;Điểm tâm của giới hạn bản vẽ
              (/ (+ (cadr LL) (cadr UR)) 2)
            )
        LL  (list (- (car CPT) (/ (car CPT) 10.0))  ;Góc trái dưới của h.c.nhật
              (- (cadr CPT) (/ (cadr CPT) 10.0))
            )
        UR  (list (+ (car CPT) (/ (car CPT) 10.0)) ;Góc phải trên của h.c.nhật
              (+ (cadr CPT) (/ (cadr CPT) 10.0))
            )
        LR  (list (car UR) (cadr LL))                ;Góc phải dưới của h.c.nhật
```

```
      UL    (list (car LL) (cadr UR))          ;Góc trái trên của h.c.nhật
  )                                                    ; Đóng setq
;
  (prompt "\nSelect objects from screen/<ENTER> when done ...")
  (while (setq PIC (entsel "\nSelect object: "))     ;Chọn các đối tượng.
    (setq ENT (car PIC)                                ; Mã đối tượng
          PT    (cadr PIC)                           ; Điểm chọn trên đối tượng
          LST   (cons PT LST)          ;Kết nối PT vào danh sách LST
    )                                                  ; Đóng setq
    (ssadd ENT SET)                    ;Thêm mã đối tượng vào tập hợp SET
  )                                                    ; Đóng while
  (setq LST (reverse LST))       ;Đảo ngược danh sách LST để các đối tượng
                                 ;trong SET đúng thứ tự với các điểm chọn
                                 ;trong LST
;
  (prompt "\nReposition cursor inside highlighted box ...")
  (grvecs (list
      (1- CTR)  LL LR          ;Vẽ h.c.nhật có 4 đỉnh là LL LR UR UL
                LR UR          ; có màu đỏ, nét đứt.
                UR UL
                UL LL
      )
  )
;
;Vòng lặp While sẽ thực hiện liên tục cho đến khi con trỏ lọt vào trong
;h.c.nhật.
;Để ý cách sử dụng hàm (grread T) để theo dõi sự di chuyển của con trỏ.
;
  (while
    (or (> (car   (cadr (grread T))) (car UR))  ;Khi con trỏ còn nằm ngoài các
        (> (cadr (cadr (grread T))) (cadr UR)) ;góc của h.c.nhật thì vòng lặp
        (< (car   (cadr (grread T))) (car LL))    ;while còn tiếp tục.
        (< (cadr (cadr (grread T))) (cadr LL))
    )                                                  ;Đóng or
  )                                                    ;Đóng while
;
  (while (setq IT (ssname SET CTR))          ;Duyệt các đối tượng trong SET
    (command ".move" IT ""                   ;Di chuyển các đối tượng
          (nth CTR LST)                      ;từ điểm chọn của đối tượng
          (cadr (grread T))                  ;đến vị trí con trỏ trên m.hình
    )
```

```
      (setq CTR (1+ CTR))
   )                                          ; Đóng while
;
   (setvar "cmdecho" 1)
   (princ)
)                                             ; Đóng defun
; Kết thúc chương trình.
```

Tóm tắt

1. Hàm **Grclear** (*GRaphics CLEAR*) sử dụng để che các đối tượng trên khung nhìn hiện hành. Các hàm **Redraw** và **Regen** sẽ làm hiện lại các đối tượng này.
2. Hàm **Grdraw** sử dụng để vẽ một véc tơ đi qua 2 điểm trên màn hình trong hệ trục UCS hiện hành. Các hàm **Redraw** và **Regen** sẽ xóa đi véc tơ này.
3. Hàm **Grvecs** sử dụng để vẽ nhiều véc tơ trên màn hình cùng lúc. Tham số VLIST cung cấp màu và tọa độ các điểm để vẽ các véc tơ. Tham số TRANS cung cấp ma trận biến hình để thay đổi vị trí và tỉ lệ các điểm trong VLIST.
4. Hàm **Grtext** sử dụng để viết một chuỗi lên dòng trạng thái hoặc lên menu màn hình. Hàm (**Grtext**) không tham số sẽ trả về các giá trị mặc định ban đầu cho dòng trạng thái và menu màn hình.
5. Hàm **Gread** sử dụng để nhận giá trị nhập từ các thiết bị nhập như các thiết bị trỏ hoặc bàn đồ họa. Tham số TRACK cho phép theo dõi sự di chuyển của con trỏ mà không cần nhấn phím.

16.4 Bài tập

1. Bổ sung các hàm **Textscr**, **Textpage**, **Graphscr** vào các chương trình ở phần bài tập chương 15 ở những vị trí cần thiết để chuyển đổi từ màn hình đồ họa sang màn hình văn bản, và ngược lại.
2. Tự chọn 4 biến hệ thống chứa giá trị là chuỗi. Viết chương trình GROUT.LSP ghi lên menu màn hình giá trị của 4 biến này.
3. Viết chương trình NUM-TXT.LSP thực hiện các bước sau:

a/ Chuyển qua màn hình đồ họa. Tạo vòng lặp **While** để người sử dụng liên tục nhập vào các số. Nhấn ENTER để kết thúc vòng lặp. Cộng các số đã nhập và gán cho biến SUM.

b/ In kết quả lên màn hình văn bản.

4. Viết chương trình P16-4.LSP thực hiện các bước sau:

a/ Yêu cầu người sử dụng nhập vào giá trị cho các tham số COLOR và HIGHLIGHT để xác định màu sắc và dạng đường cho các véc tơ sẽ vẽ.

b/ Sử dụng các hàm **Grdraw** và **Grvecs** để vẽ các véc tơ bao quanh các giới hạn bản vẽ.

5. Viết chương trình P16-5.LSP thực hiện các bước sau:

a/ Tạo các biến lưu giá trị các biến hệ thống sau đây:

- DIMASZ
- DIMCEN
- DIMDLI
- DIMSCALE
- DIMTXT

b/ Sử dụng hàm **Grtext** in tên và giá trị các biến lên menu màn hình:

```
DIMVARS
- - - - - - - -
Dimasz
3.0
- - - - - - - -
Dimcen
2.5
- - - - - - - -
Dimdli
3.75
- - - - - - - -
Dimscale
1
- - - - - - - -
Dimtxt
3.0
```

16.4 Lời giải

2.

```
;Tên file: GROUT.LSP
;Mục đích: Hiện giá trị 4 biến hệ thống của AutoCAD lên menu màn hình.
```

```
;
(defun C:GROUT (/ LST CTR)
  (setvar "cmdecho" 0)
  (setq LST '("CECOLOR" "CELTYPE" "CLAYER" "CMLSTYLE")
        CTR  0
  )
;
  (repeat 4
    (grtext CTR (GETVAR (nth CTR LST)))
    (setq CTR (1+ CTR))
  )                                          ; Đóng repeat
)                                            ; Đóng defun
;Kết thúc chương trình.
```

3.

```
;Tên file: NUM-TXT.LSP
;
(defun C:NUM-TXT (/ NUM SUM)
  (setvar "cmdecho" 0)
  (graphscr)
  (setq SUM 0)
  (while (setq NUM (getreal "\nEnter a number (or ENTER when done): "))
    (setq SUM (+ SUM NUM))
  )                                          ; Đóng while
  (textscr)
  (princ (strcat "\nThe summation is: " (rtos SUM)))
  (setvar "cmdecho" 1)
  (princ)
)                                            ; Đóng defun
;Kết thúc chương trình
```

4.

```
;Tên file: P16-4.LSP
;
(defun C:P16-4 (/ COL HL LL UR LR UL)
  (initget 1)
  (setq COL (abs (getint "\nColor integer: ")))
  (initget 1 "Yes No")
  (setq HL (getkword "Highlight <Yes/No>: "))
```

```
  (if (= HL "Yes") (setq COL (- COL)))
;
  (setq LL  (getvar "Limmin")          ;Góc trái dưới của giới hạn bản vẽ.
        UR  (getvar "Limmax")          ;Góc phải trên của giới hạn bản vẽ
        LR  (list (car UR) (cadr LL))  ;Góc phải dưới của giới hạn bản vẽ
        UL  (list (car LL) (cadr UR))  ;Góc trái trên của giới hạn bản vẽ.
  )
;
  (command ".zoom" "a")
  (grvecs (list COL
            LL LR              ;Vẽ h.c.nhật có 4 đỉnh là LL LR UR UL
            LR UR
            UR UL
            UL LL
      )
  )
  (princ)
)                                                  ;Đóng defun
;Kết thúc chương trình
```

5.

```
;Tên file: P16-5.LSP
;
(defun C:P16-5()
  (setq DA "Dimasz"
        DC "Dimcen"
        DD "Dimdli"
        DS "Dimscale"
        DT "Dimtxt"
  )
  (grtext 0 "DIMVARS")
  (grtext 1 "- - - - - - - -")
  (grtext 2 DA )
  (grtext 3 (rtos (getvar DA)))
  (grtext 4 "- - - - - - - -")
  (grtext 5 DC )
  (grtext 6 (rtos (getvar DC)))
  (grtext 7 "- - - - - - - -")
  (grtext 8 DD )
  (grtext 9 (rtos (getvar DD)))
```

```
   (grtext 10 "- - - - - - - -")
   (grtext 11 DS )
   (grtext 12 (rtos (getvar DS)))
   (grtext 13 "- - - - - - - -")
   (grtext 14 DT )
   (grtext 15 (rtos (getvar DT)))
)
```

;Kết thúc chương trình.

Chương 17

TẠO HỘP THOẠI

Nội dung chương:

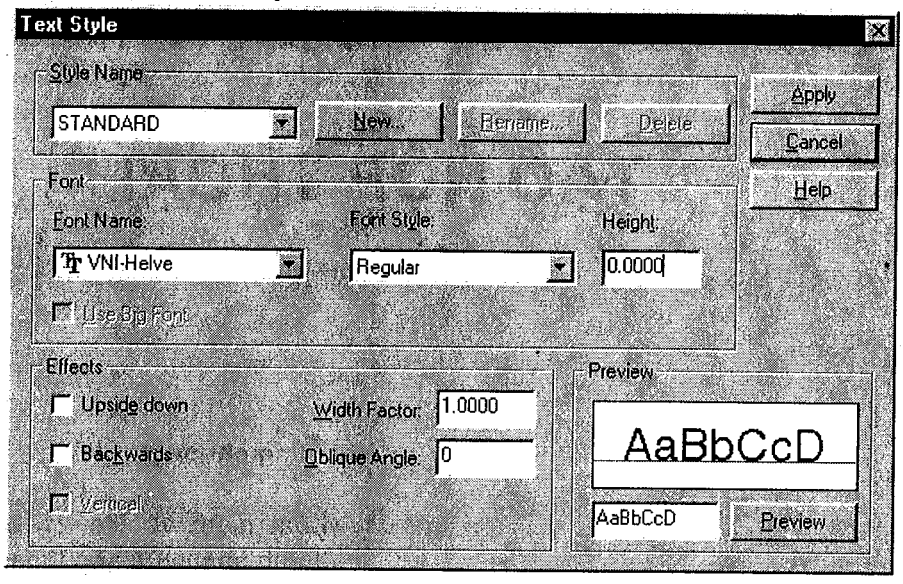
1. Khái niệm về file .DCL và các thành phần của hộp thoại.
2. Các *tile* và các thuộc tính của *tile*.

Người sử dụng thường xuyên phải giao tiếp, trao đổi số liệu với phần mềm **AutoCAD** như nhập các giá trị, các tọa độ điểm. Việc giao tiếp thông qua các hộp thoại tỏ ra dễ dàng, thuận lợi hơn so với tại dòng nhắc lệnh.

Chương này sẽ trình bày cấu trúc của một hộp thoại và các thành phần của nó. Chương tiếp theo sẽ trình bày cách sử dụng hộp thoại trong các ứng dụng **AutoLISP**.

Hình dạng các hộp thoại phụ thuộc vào hệ điều hành đang sử dụng. Các hộp thoại của **AutoCAD** tương tự như hầu hết các ứng dụng chạy trên hệ điều hành Windows.

Ví dụ: Hộp thoại **Text Style** của **AutoCAD**.



Để tạo ra các hộp thoại, trước tiên ta sử dụng ngôn ngữ *Dialog Control Language (DCL)* để mô tả đặc điểm cấu trúc của hộp thoại, sau đó sử dụng ngôn ngữ **AutoLISP** để hiển thị và lấy thông tin từ hộp thoại.

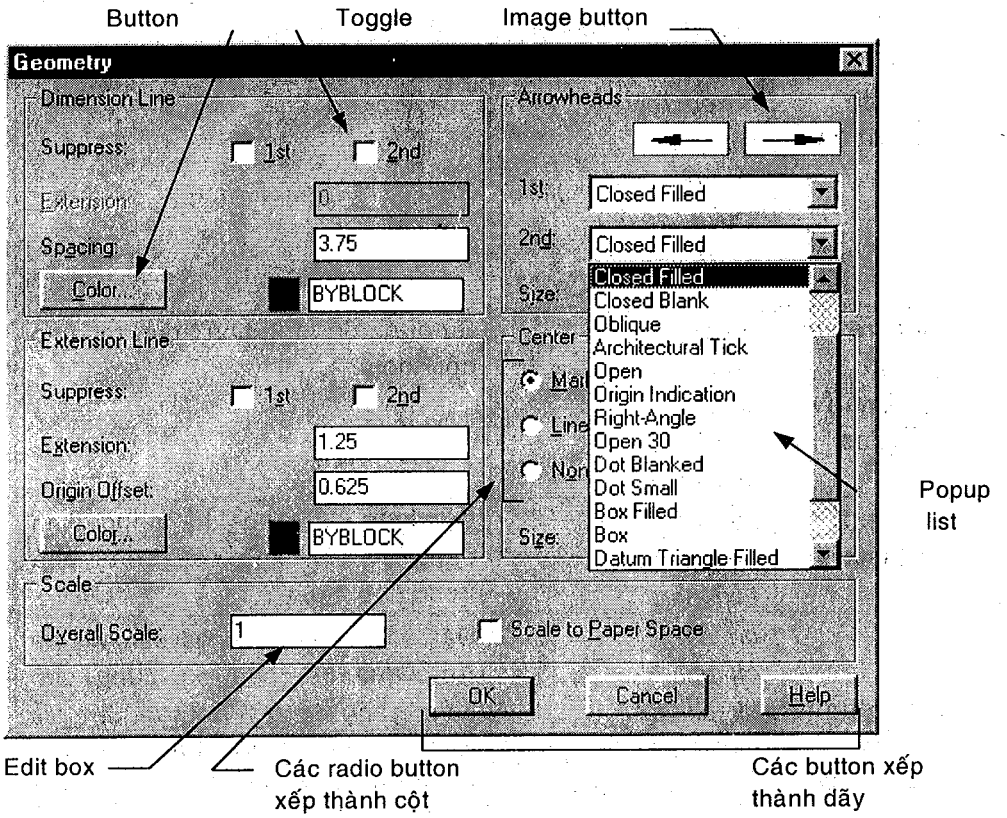
Các giá trị mô tả cấu trúc của hộp thoại được lưu trữ trong file *.DCL*.

17.1 Khái niệm về file .DCL và các thành phần của hộp thoại

File .DCL sử dụng để mô tả cấu trúc của hộp thoại. File này có dạng file văn bản ASCII tương tự như file chương trình **AutoLISP**. Các hộp thoại của **AutoCAD** được mô tả trong file acad.dcl (trong thư mục SUPPORT). Ta có thể xem nội dung file này bằng các phần mềm như **Notepad**, **MS Word**...

Các thành phần của hộp thoại như các *nút lệnh (button)* hoặc các *hộp văn bản* (*edit box)* ... , gọi là các *tile*. Hình dáng, kích thước, chức năng ... của các *tile* được xác định bởi các *thuộc tính* (*attribute*) của chúng. Ngoài ra, ta có thể sử dụng các *tile* để tạo ra các *Prototype* hoặc các *Subassembly* để sử dụng được nhiều lần trong các file DCL khác nhau (tương tự như tạo các khối để sử dụng nhiều lần trong các bản vẽ).

Ví dụ: Các *tile* của hộp thoại **Geometry** khi thực hiện lệnh **Ddim** trong **AutoCAD 14**.



AutoCAD đã mô tả cấu trúc mặc định cho các *tile*, ví dụ như kích thước hộp thoại và cách sắp xếp các thành phần của nó được thực hiện tự động. Do đó, ta chỉ khai báo giá trị cho những thuộc tính nào cần thiết mà thôi.

Phân loại các *tile*

Phân loại	Các tile	Mô tả
Active Tiles	*button* *edit_box* *image_button* *list_box* *popup_list*	Có khả năng nhận được con trỏ, do đó nhận được giá trị nhập của người sử dụng. Là các *tile* cơ sở, có thể sử dụng để tạo ra các *tile* phức tạp hoặc các *cluster*.

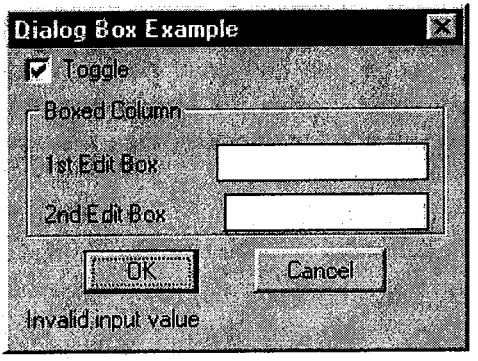
	radio_button *slider* *toggle*	
Tile Clusters	*boxed_column* *boxed_radio_column* *boxed_radio_row* *boxed_row* *column* *dialog* *radio_column* *radio_row* *row*	Sử dụng để nhóm các *tile* thành một hàng hoặc một cột.
Decorative & Informative Tiles	*image* *text* *spacer* *spacer_0* *spacer_1*	Sử dụng để tăng tính thẩm mỹ cho hộp thoại.
Text Clusters	*concatenation* *paragraph* *text_part*	Sử dụng để tăng khả năng định dạng và sắp xếp các dòng chữ trong hộp thoại.
Exit buttons and Error *Tile*	*err_tile* *ok_only* *ok_cancel* *ok_cancel_help* *ok_cancel_help_* *errtile* *ok_cancel_help_info*	Đây là các *subassembly* do **AutoCAD** tạo sẵn sử dụng cho chức năng đóng hộp thoại. *err_tile* là một vùng ở góc trái dưới hộp thoại sử dụng để hiện thông báo lỗi khi người sử dụng đưa vào dữ liệu không thích hợp.

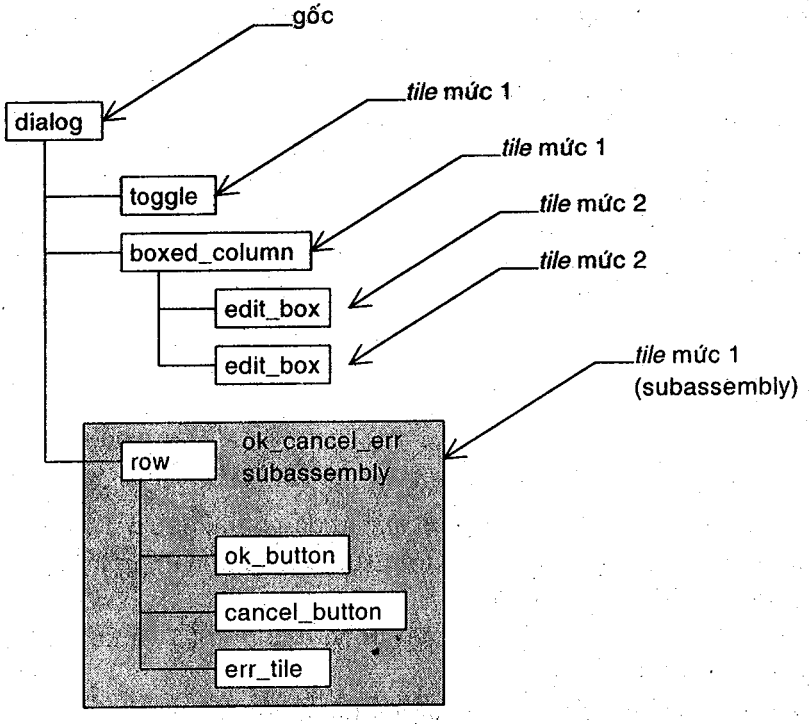
Cấu trúc cây trong mô tả cấu trúc hộp thoại

Trong file DCL, để mô tả cấu trúc của một hộp thoại, ta liệt kê lần lượt các *tile* và các thuộc tính của chúng theo thứ tự từ trên xuống, theo cấu trúc *cây*.

Ví dụ:

Cấu trúc cây của hộp thoại *sample*, các dòng mô tả trong file *sample.DCL* và chương trình **AutoLISP** làm xuất hiện hộp thoại này.





Mô tả cấu trúc cây:

- Mức cao nhất của cây, gọi là *gốc (root)*, luôn luôn là *dialog*.
- Các *tile* mức 1: *toggle, boxed_column, row* gọi là *con (children)* của hộp thoại.
- Các *tile* mức 2: *edit_box, edit_box* là con của *boxed_column*.
- *row:* là một *subassembly* chứa các *tile*: *ok_button, cancel_button, err_tile*.

Các dòng mô tả hộp thoại:

```
// Tên file: SAMPLE.DCL
// Sample Dialog Box Code
sample : dialog {
        label           = "Dialog Box Example";
        : toggle {
          label         = "Toggle";
          value         = "1";
        }
        : boxed_column {
          label         = "Boxed Column";
          : edit_box {
            label       = "1st Edit Box";
            key         = "edit_1";
          }
          : edit_box {
            label       = "2nd Edit Box";
            key         = "edit_2";
          }
        }
        ok_cancel_err;
    }
```

Ghi chú:

- Tên các *tile* phải viết bằng chữ *thường*, bắt đầu bằng dấu hai chấm (:)
- Các *tile* được liệt kê theo thứ tự từ trên xuống như trong hộp thoại.
- Các dòng chú thích bắt đầu bằng //
- Các thuộc tính của một *tile* được đặt trong cặp dấu { } và kết thúc mỗi thuộc tính là dấu chấm phẩy (;) .
- Các hộp thoại phải chứa một *exit tile* để đóng hộp thoại.

Sau đây là chương trình **AutoLISP** để làm xuất hiện hộp thoại trên:

```
;Tên file:  SAMPLE.LSP
;Mục đích: Làm xuất hiện hộp thoại Dialox Box Sample trong file
;           SAMPLE.DCL
;           File sample.dcl phải được đặt trong đường dẫn thư viện, hoặc
;           phải bổ sung đường dẫn cho tên file trong hàm load_dialog.
;
```

```
(setq DCL_ID (load_dialog "Sample.DCL"))
(new_dialog "sample" DCL_ID)
(start_dialog)
;Kết thúc chương trình
```

Tóm tắt

1. Các hộp thoại cung cấp một giao diện với người sử dụng tốt hơn là các chuỗi xuất hiện tại dòng nhắc lệnh.
2. Các mã DCL mô tả cấu trúc, hình dáng của hộp thoại bằng cách gán giá trị cho các thuộc tính như tiêu đề, vị trí và các giá trị mặc định cho các *tile*.
3. Các file DCL ở dạng file văn bản ASCII và có tên mở rộng là *.DCL*
4. Dấu chấm phẩy báo hiệu kết thúc một thuộc tính.
5. Các dòng chú thích trong file DCL bắt đầu bằng 2 dấu gạch //
6. Mỗi hộp thoại đều phải có một *exit tile* để đóng hộp thoại.

17.2 Các *tile* và các thuộc tính của *tile*

Các thuộc tính sử dụng để xác định hình dạng, kích thước, vị trí của các *tile*. Một số thuộc tính sử dụng được cho nhiều *tile*, một số thuộc tính chỉ sử dụng riêng cho một *tile* mà thôi.

Bảng liệt kê các thuộc tính của các *tile*:

Thuộc tính	*Tile*	Giá trị	Mô tả
action	Tất cả các *active tile*	"(Biểu Thức)"	Biểu thức **AutoLISP** liên kết với *tile* để tạo ra chức năng cho *tile*.
alignment	Tất cả các *tile*	*left, right, centered, top, bottom* Mặc định: *left* (ngang) hoặc *centered* (đứng)	Xác định vị trí canh lề của *tile* khi được đặt trong một *cluster* (theo phương nằm ngang hoặc thẳng đứng)

allow_accept	*edit_box*, *image_button*, *list_box*	*true*, *false* Mặc định: *false*	Nếu bằng *true*, khi *tile* đang được chọn, nếu nhấn phím ENTER, thì *button* mặc định sẽ được "nhấn".
aspect_ratio	*image*, *image_button*	Số thực Mặc định: không có	Tỉ lệ giữa chiều rộng và chiều cao của hình ảnh.
big_increment	*slider*	Số nguyên. Mặc định: 1/10 tổng miền giá trị	Độ dời khi nhấn vào phần thân của thanh *slider*.
children_ alignment	*row*, *column*, *radio_row*, *radio_column*, *boxed_row*, *boxed_column*, *boxed_radio_row*, *boxed_radio_ column*	*left*, *right*, *centered*, *top*, *bottom* Mặc định: *left* (ngang) hoặc *centered* (đứng)	Xác định vị trí canh lề *mặc định* của các *tile* trong *cluster* (theo phương nằm ngang hoặc thẳng đứng).
children_fixed_ height	*row*, *column*, *radio_row*, *radio_column*, *boxed_row*, *boxed_column*, *boxed_radio_row*, *boxed_radio_ column*	*true*, *false* Mặc định: *false*	Nếu bằng *true*, chiều cao các *tile* trong *cluster* sẽ giữ cố định.
children_fixed_ width	*row*, *column*, *radio_row*, *radio_column*, *boxed_row*, *boxed_column*, *boxed_radio_row*, *boxed_radio_ column*	*true*, *false* Mặc định: *false*	Nếu bằng *true*, chiều rộng các *tile* trong *cluster* sẽ giữ cố định.

color	*image*, *image_button*	Số nguyên hoặc các từ dành riêng Mặc định: 7	Màu nền của hình ảnh
edit_limit	*edit_box*	Số nguyên Mặc định: 132	Số lượng tối đa các ký tự nhập vào.
edit_width	*edit_box*, *popup_list*	Số nguyên hoặc số thực	Chiều rộng của phần nhập ký tự.
fixed_height	tất cả các *tile*	*true, false* Mặc định: *false*	Nếu bằng *true*, chiều cao của *tile* sẽ giữ cố định.
fixed_width	tất cả các *tile*	*true, false* Mặc định: *false*	Nếu bằng *true*, chiều rộng của *tile* sẽ giữ cố định.
fixed_width_font	*list_box*, *popup_list*	*true, false* Mặc định: *false*	Nếu bằng *true*, hiển thị chữ bằng font có chiều rộng các ký tự bằng nhau.
height	tất cả các *tile*	Số nguyên hoặc số thực	Chiều cao của *tile*.
initial_focus	*dialog*	"*key*" Mặc định: không có	Xác định *tile* nào nhận con trỏ khi hộp thoại xuất hiện. *Key* là tên của *tile* nhận con trỏ.
is_bold	*text*	*true, false* Mặc định: *false*	Nếu bằng *true*, chữ sẽ được in đậm.
is_cancel	*button*	*true, false* Mặc định: *false*	Nếu bằng *true*, *button* sẽ được "nhấn" khi người sử dụng nhấn phím ESC. Chỉ có duy nhất một *button* có thuộc tính này bằng *true*.

is_default	*button*	*true, false* Mặc định: *false*	Nếu bằng *true*, *button* sẽ được "nhấn" khi người sử dụng nhấn phím ENTER. Chỉ có duy nhất một *button* có thuộc tính này bằng *true.*
is_enabled	Tất cả các *active tile*	*true, false* Mặc định: *true*	Nếu bằng *false*, *tile* sẽ bị mờ đi, không chọn được.
is_tab_stop	Tất cả các *active tile*	*true, false* Mặc định: *true*	Nếu bằng *false*, *tile* sẽ không nhận được con trỏ khi người sử dụng di chuyển giữa các *tile* bằng cách nhấn phím TAB.
key	Tất cả các *active tile*	"Chuỗi" Mặc định: không có	Tên của *tile* được chương trình sử dụng. Phân biệt dạng chữ hoa chữ thường.
label	*boxed_row, boxed_column, boxed_radio_row, button, boxed_radio_ column, text, dialog, edit_box, list_box, popup_list, radio_button, toggle*	"string" Mặc định: chuỗi rỗng	Dòng tiêu đề của *tile*. Ký tự nào của dòng tiêu đề có dấu & ở phía trước sẽ là ký tự "nóng" (khi xuất hiện trên màn hình, ký tự này có dấu gạch dưới) . Sử dụng chung với phím ALT+Ký tự, sẽ chuyển con trỏ về *tile* này.
layout	*slider*	*horizontal, vertical* Mặc định: *horizontal*	Hướng của thanh *slider* (nằm ngang hoặc thẳng đứng)

list	*list_box, popup_list*	"Chuỗi" Mặc định: không có	Các giá trị xuất hiện trong danh sách. Các giá trị được ngăn cách nhau bằng ký tự xuống dòng "\n".
max_value	*slider*	Số nguyên. Mặc định: 10000	Giá trị cực đại của *slider*.
min_value	*slider*	Số nguyên: Mặc định: 0	Giá trị cực tiểu của *slider*.
mnemonic	Tất cả các *tile*	"Ký tự"	Xác định ký tự nóng (khi xuất hiện trên tiêu đề của *tile*, ký tự này có dấu gạch dưới). Sử dụng chung với phím ALT+Ký tự, sẽ chuyển con trỏ về *tile* này.
multiple_select	*list_box*	*true, false* Mặc định: *false*	Nếu bằng *true*, có thể chọn nhiều mục cùng lúc trên danh sách.
password_char	*edit_box*	"Ký tự"	Ký tự "mật mã". Trên màn hình không xuất hiện các ký tự nhập từ bàn phím, mà chỉ xuất hiện ký tự mật mã.
small_ increment	*slider*	Số nguyên. Mặc định: 1/100 tổng miền giá trị	Độ dời khi nhấn vào phần mũi tên của thanh *slider*.
tabs	*list_box, popup_list*	"Chuỗi"	Xác định vị trí các TAB cho danh sách dạng cột. Đơn vị tính là chiều rộng ký tự. Ví dụ: Chuỗi "8 16 24 32" xác định 4 vị trí TAB. Mỗi vị trí cách nhau 8 ký tự.

tab_truncate	*list_box, popup_list*	*true, false* Mặc định: *false*	Nếu bằng *true*, các chuỗi xuất hiện trên danh sách sẽ bị xén bớt khi cần thiết.
value	*text*, tất cả các *active tile* (ngoại trừ *button* và *image_button).*	"Chuỗi"	Giá trị ban đầu của *tile.*
width	Tất cả các *tile.*	Số nguyên hoặc số thực	Chiều rộng của *tile.*

Giá trị của các thuộc tính phải thuộc một kiểu dữ liệu. Có 4 kiểu dữ liệu cơ bản là:

- Số nguyên (*integer*)
- Số thực (*real*)
- Chuỗi (*string*): Phân biệt dạng chữ hoa chữ thường.
- Từ dành sẵn (*reserved word*): **Không** được đặt trong dấu nháy chuỗi. Phân biệt dạng chữ hoa chữ thường. Ví dụ: *horizontal*, *true*, *false* (chứ không phải Horizontal, True, False).

Chú ý:

Tên thuộc tính (cũng như các từ dành sẵn) có phân biệt dạng chữ hoa chữ thường. Tất cả các tên thuộc tính đều được viết bằng chữ thường và dấu gạch dưới (_).

Các Active *tile*

Là các thành phần cơ bản để thiết kế các hộp thoại. Chúng có thể được kết hợp để tạo ra các *cluster* và *subassembly.*

Phần tiếp theo sẽ lần lượt mô tả về các *tile* và sơ lược về các thuộc tính thường dùng của chúng.

Dialog

```
: dialog {
        initial_focus  label  value  key
}
```

Dialog là *tile* sử dụng để mô tả bản thân hộp thoại. Các *tile* con của *dialog* được sắp xếp thành một cột theo thứ tự từ trên xuống.

initial_focus Tên của *tile* được nhận con trỏ khi hộp thoại xuất hiện.

label Tiêu đề của hộp thoại (xuất hiện trên thanh tiêu đề).

value Tiêu đề của hộp thoại. Thường được sử dụng trong thời gian chạy (hộp thoại đã xuất hiện và ta muốn thay đổi tiêu đề của nó).

Button

: button {
action alignment fixed_height fixed_width height is_cancel is_default is_enabled key is_tabstop label mnemonic width
}

Nút lệnh, có hình dạng một nút nhấn. Khi người sử dụng nhấn vào nút lệnh này, nó sẽ thực hiện một chức năng (là một hàm **AutoLISP**) được liên kết với nó. Mỗi hộp thoại phải có ít nhất một *button* để đóng hộp thoại.

label Tiêu đề của *button* (ví dụ: "OK").

Ví dụ:

```
: button {
   label      = "&OK";
   key        = "btn_1";
 }
```

Edit box

: edit_box {
action alignment allow_accept fixed_height edit_limit edit_width fixed_width height is_enabled is_tab_stop key label mnemonic

password value width
}

Edit_box cho phép người sử dụng nhập và sửa nội dung của chuỗi xuất hiện trên hộp thoại.

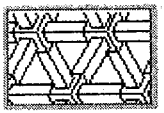
label Tiêu đề của *edit\box* (ví dụ: "Linetype scale").

value Chuỗi xác định giá trị mặc định ban đầu của *edit_box*.

Ví dụ:

```
: edit_box {
   label      = "Linetype &Scale";
   value      = "1.0000";
   key        = "edit_1";
}
```

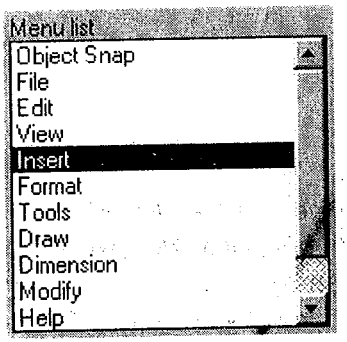
Image button



```
: image_button {
action alignment allow_accept aspect_ratio
color fixed_height fixed_width height
is_enabled is_tab_stop key label
mnemonic width
}
```

Image_button hiển thị hình ảnh và hoạt động giống như *button*. Hình ảnh sử dụng có thể là một slide của **AutoCAD** hoặc là các véc tơ. Ta có thể xác định được tọa độ điểm chọn trên hình ảnh, nhờ đó mà thực hiện các lệnh thích hợp. Các thuộc tính chiều cao và chiều rộng bắt buộc phải có hoặc thông qua thuộc tính tỉ lệ giữa chúng *aspect_ratio*.

List Box




```
: list_box {
action alignment allow_accept fixed_height
fixed_width_font tabs fixed_width height
is_enabled is_tab_stop key label list
mnemonic multiple_select tab_truncate
```

value width
}

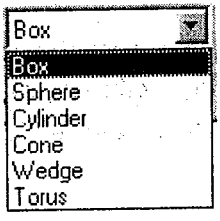
List_box sử dụng để cung cấp cho người sử dụng một danh sách các chuỗi sắp xếp theo cột để lựa chọn. Khi có nhiều phần tử không thể xuất hiện đủ trên danh sách, sẽ xuất hiện thanh cuộn theo phương thẳng đứng.

label	Tiêu đề xuất hiện phía trên danh sách.
list	Chuỗi trong dấu nháy kép chứa nội dung các phần tử xuất hiện trong danh sách. Mỗi phần tử được ngăn cách nhau bằng ký tự xuống dòng "\n".
value	Chuỗi trong dấu nháy kép chứa số nguyên xác định phần tử được chọn trong danh sách. Các phần tử được đánh số thứ tự bắt đầu từ 0.

Ví dụ:

```
: list_box {
  label    = "Menu list";
  list     = "Object Snap \nFile \nEdit \nView \nInsert \nFormat
             \nTools\nDraw \nDimension \nModify \nHelp
             \nBonus \nExpress";
  value    = "4";
  key      = "Lst_1";
}
```

Popup list



: popup_list {
action alignment edit_width fixed_height fixed_width_font tabs fixed_width height is_enabled is_tab_stop key label list

```
    mnemonic tab_truncate value width
}
```

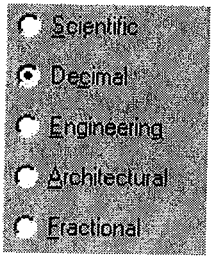
Popup_list tương tự như *list_box*, ngoại trừ danh sách bị che đi, chỉ xuất hiện khi người sử dụng chọn vào mũi tên bên phải của *popup_list*. *Popup_list* không cho phép chọn nhiều giá trị cùng lúc.

label	Tiêu đề xuất hiện bên trái của *popup_list*.
edit_width	Chiều rộng của phần xuất hiện chữ (không kể phần tiêu đề, mũi tên và thanh cuộn). Giá trị này là số thực hoặc số nguyên.
value	Chuỗi trong dấu nháy kép chứa số nguyên xác định phần tử được chọn trong danh sách. Các phần tử được đánh số thứ tự bắt đầu từ 0.

Ví dụ:

```
: popup_list {
  label    = "Solids";
  list     = "Box \nSphere \nCylinder \nCone \nWedge \nTorus ";
  value    = "0";
  key      = "pop_1";
}
```

Radio_button




```
: radio_button {
  action alignment fixed_height fixed_width
  height is_enabled is_tab_stop key label list
  mnemonic value width
}
```

Các *radio_button* cung cấp một số lựa chọn, và tại một thời điểm chỉ có một trong số đó được chọn. Các *radio_button* không được đặt trực tiếp vào trong *dialog* mà chỉ có thể được đặt trong *radio_row* hoặc *radio_column* mà thôi.

label	Tiêu đề xuất hiện bên phải *radio_button*.

value Chuỗi trong dấu nháy kép chứa số nguyên xác định trạng thái ban đầu của *radio_button*. Giá trị "0" nghĩa là không được chọn. "1" nghĩa là được chọn.

Ví dụ:

```
:radio_button {
  label  ="&Scientific";
  value ="0";
}
:radio_button {
  label  ="De&cimal";
  value ="1";
}
```

Slider

```
: slider {
  action  alignment big_increment
  fixed_height  fixed_width  key  label  layout
  max_value  min_value  mnemonic
  small_increment  value  width
}
```

Slider có chức năng tương tự thanh cuộn (*scroll bar*). Khi nhấn vào mũi tên, con trượt sẽ di chuyển với độ dời nhỏ *small_increment*. Khi nhấn vào thân thanh cuộn, con trượt sẽ di chuyển với độ dời lớn *big_increment*.

value Chuỗi chứa số nguyên xác định giá trị ban đầu của *slider*.

max_value Giá trị cực đại của *slider*

min_value Giá trị cực tiểu của *slider*.

Ví dụ:

```
: slider {
  min_value = 1;
  max_value = 100;
```

```
    value = 25;
  }
```

Toggle


```
: toggle {
  action  alignment  fixed_height  fixed_width
  height label is_enabled is_tab_stop  value  width
}
```

Các *toggle* cho phép người sử dụng lựa chọn đồng thời nhiều mục. Khi sử dụng chuột nhấn vào *toggle* thì giá trị của nó sẽ thay đổi. Khi được chọn, sẽ xuất hiện dấu gạch chéo **X** hoặc ✓.

label	Tiêu đề xuất hiện bên phải *toggle*
value	Chuỗi trong dấu nháy kép chứa số nguyên xác định trạng thái ban đầu của *radio_button*. Giá trị "0" nghĩa là không được chọn. "1" nghĩa là được chọn.

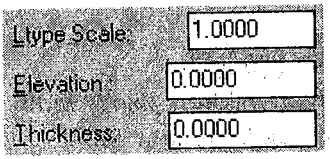
Ví dụ:

```
: toggle {
   label   = "&Endpoint";
   value ="1";
}
: toggle {
   label   = "&Midpoint";
   value ="0";
}
```

Các tile cluster

Các *tile cluster* sử dụng để nhóm các *tile* có chức năng liên quan với nhau cho dễ quản lý và sử dụng trong hộp thoại. Ta không thể sử dụng chuột chọn được chính các *cluster* mà chỉ có thể chọn các *tile* bên trong của nó. Ngoại trừ các *radio_row* và *radio_column*, ta không thể gán chức năng (biểu thức **AutoLISP**) cho các *cluster*.

Column




```
: column {
  alignment children_alignment
  children_fixed_height hildren_fixed_width
  fixed_height fixed_width height label
  width
}
```

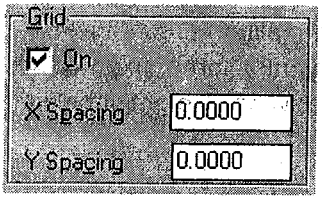
Các *tile* được sắp xếp thành một cột theo thứ tự từ trên xuống. Chúng được xem là con của *column*. Do đó, các thuộc tính như *children_alignment* sẽ có tác dụng đối với tất cả các *tile* con. *Column* có thể chứa tất cả các loại *tile*, ngoại trừ *radio_button* đứng một mình.

Chú ý: Nếu sử dụng thuộc tính *label*, một khung chữ nhật sẽ được vẽ bao quanh *column tile*, nhưng trong một số trường hợp, khung này vẽ không được chính xác. Do đó, ta nên sử dụng *boxed_column* để thay thế.

Ví dụ:

```
: column {
        : edit_box {
          label  = "&Ltype Scale:";
          value = "1.0000";
          key    = "edit_1";
        }
        : edit_box {
          label  = "&Elevation :";
          value = "0.0000";
          key    = "edit_2";
        }
        : edit_box {
          label  = "&Thickness:";
          value  = "0.0000";
          key    = "edit_3";
        }
}
```

Boxed_column




```
: box_column {
   alignment children_alignment
   children_fixed_height hildren_fixed_width
   fixed_height fixed_width height label
   width
}
```

Boxed_column có chức năng tương tự như *column*. Bao quanh nó là một khung chữ nhật và một tiêu đề.

Ví dụ:

```
: boxed_column {
      label  = "&Grid";
      : toggle {
        label  = "On";
        value = "1";
        key    = "tgl_1";
      }
      : edit_box {
        label  = "X S&pacing";
        value  = "0.0000";
        key    = "edit_1";
      }
      : edit_box {
        label  = "Y Spa&cing";
        value  = "0.0000";
        key    = "edit_2";
      }
}
```

Row


```
: row {
   alignment children_alignment
   children_fixed_height hildren_fixed_width
   fixed_height fixed_width height label
   width
}
```

Row có chức năng tương tự *column*, ngoại trừ các *tile* con được sắp xếp theo hàng ngang theo thứ tự từ trái sang phải.

Ví dụ:

```
: row {
  : toggle {
    label = "&Double";
    value = "0";
    key   = "tgl_1";
  }
  : toggle {
    label = "&Exploded";
    value = "0";
    key   = "tgl_2";
  }
}
```

Boxed_row


```
: boxed_row {
alignment children_alignment
children_fixed_height children_fixed_width
fixed_height fixed_width height label width
}
```

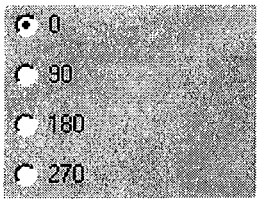
Boxed_row có chức năng tương tự như *row*. Bao quanh nó là một khung chữ nhật và một tiêu đề.

Ví dụ:

```
: boxed_row {
   label    = "Plot origin";
   : edit_box {
     label  = "&X origin: ";
     value  = "0.00";
     key    = "edit_1";
   }
```

```
    :edit_box {
      label  = "&Y origin: ";
      value  = "0.00";
      key    = "edit_2";
    }
  }
```

Radio column




```
: radio_column {
  alignment  children_alignment
  children_fixed_height  children_fixed_width
  fixed_height  fixed_width  height
  label  width
}
```

Radio_column là *cluster* sử dụng để chứa các *radio_button*. Các *radio_button* được sắp xếp thành một cột. Ta có thể gán chức năng (biểu thức **AutoLISP**) cho *radio_column*.

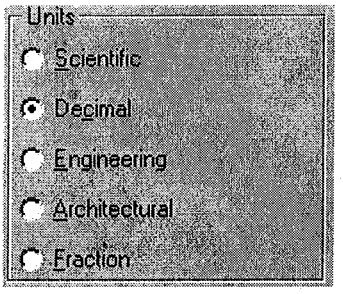
value Chuỗi trong dấu nháy kép chứa tên (*key*) của *radio_button* có giá trị bằng "1" (on).

Ví dụ:

```
:radio_column {
   :radio_button {
     label  = "0";
     value  = "1";
     key    = "rad_1";
   }
   :radio_button {
     label  = "90";
     value  = "0";
     key    = "rad_2";
   }
   :radio_button {
     label  = "180";
     value  = "0";
     key    = "rad_3";
```

```
    }
    :radio_button {
      label   = "270";
      value   = "0";
      key     = "rad_4";
    }
  }
```

Boxed radio column




```
: boxed_radio_column {
  alignment  children_alignment
  children_fixed_height  children_fixed_width
  fixed_height  fixed_width  height
  label  width
}
```

Box_radio_column có chức năng tương tự như *radio_column*, ngoại trừ nó có thêm khung viền hình chữ nhật và tiêu đề.

value Chuỗi trong dấu nháy kép chứa tên (*key*) của *radio_button* có giá trị bằng "1" (on).

Ví dụ:

```
: boxed_radio_column {
  label     = "Units";
  : radio_button {
    label   = "&Scientific";
    value   = "0";
    key     = "rad_1";
  }
  : radio_button {
    label   = "De&cimal";
    value   = "1";
    key     = "rad_2";
  }
```

```
    : radio_button {
      label  = "&Engineering";
      value  = "0";
      key    = "rad_3";
    }
    : radio_button {
      label  = "&Architectural";
      value  = "0";
      key    = "rad_4";
    }
    : radio_button {
      label  = "&Fraction";
      value  = "0";
      key    = "rad_5";
    }
}
```

Radio_row


```
: radio_row {
  alignment  children_alignment
  children_fixed_height  children_fixed_width
  fixed_height  fixed_width  height
  label  width
}
```

Radio_row là *cluster* sử dụng để chứa các *radio_button*. Các *radio_button* được sắp xếp thành hàng ngang theo thứ tự từ trái sang phải. Ta có thể gán chức năng (biểu thức **AutoLISP**) cho *radio_column*.

Nếu có thể được, ta nên sử dụng *radio_column* thay cho *radio_row* vì cách sắp xếp của *radio_column* giúp ta di chuyển con trỏ ít hơn.

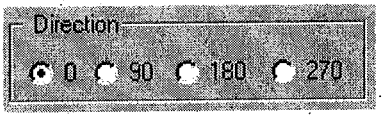
value Chuỗi trong dấu nháy kép chứa tên (*key*) của *radio_button* có giá trị bằng "1" (on).

Ví dụ:

```
:radio_row {
  :radio_button {
    label  = "0";
    value  = "1";
```

```
    key     = "rad_1";
  }
   :radio_button }
    label   = "90";
    value   = "0";
    key     = "rad_2";
  }
   :radio_button {
    label   = "180";
    value   = "0";
    key     = "rad_3";
  }
   :radio_button {
    label   = "270";
    value   = "0";
    key     = "rad_4";
  }
}
```

Boxed radio row




```
: boxed_radio_row {
  alignment  children_alignment
  children_fixed_height
  children_fixed_width
  fixed_height  fixed_width  height
  label  width
}
```

Boxed_radio_row có chức năng tương tự như *radio_row*, ngoại trừ nó có thêm khung viền hình chữ nhật và tiêu đề.

Ta nên sử dụng *boxed_radio_column* thay cho *boxed_radio_row* để giảm sự di chuyển con trỏ.

value Chuỗi trong dấu nháy kép chứa tên (*key*) của *radio_button* có giá trị bằng "1" (on).

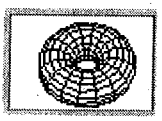
Ví dụ:

```
: boxed_radio_row {
    label = "Direction";
    :radio_button {
      label  = "0";
      value  = "1";
      key    = "rad_1";
    }
    :radio_button {
      label  = "90";
      value  = "0";
      key    = "rad_2";
    }
    :radio_button {
      label  = "180";
      value  = "0";
      key    = "rad_3";
    }
    :radio_button {
      label  = "270";
      value  = "0";
      key    = "rad_4",
    }
}
```

Informative Tile

Các *informative tile* sử dụng để cung cấp thông tin cho người sử dụng thông qua hình ảnh và các dòng chữ. Các *tile* này không thể chọn được và không tạo ra một hành động nào. Chúng chỉ sử dụng để cung cấp thông tin.

Image



```
: image {
    action  alignment  aspect_ratio  color
    fixed_height  fixed_width  height
    is_enabled  is_tab_stop  key  label
    mnemonic  value  width
}
```

Image sử dụng để hiển thị hình ảnh chứa trong một *slide file*, hoặc hình ảnh được vẽ bằng các véc tơ. Với các hình ảnh đơn giản, như để biểu diễn tọa độ các điểm đầu và cuối, ta có thể sử dụng các véc tơ. Với các hình ảnh phức tạp, ta sử dụng các lệnh vẽ của **AutoCAD** để vẽ, sau đó ghi thành một *slide file*.

Ta phải khai báo chính xác các kích thước dài và rộng cho *image tile*, hoặc một trong hai kích thước này cùng với hệ số tỉ lệ *aspect_ratio*.

Text


```
: text {
  alignment fixed_height fixed_width
  height is_bold key label value
  width
}
```

Text sử dụng để hiển thị thông tin ở dạng chuỗi, thường là các thông tin động, thay đổi trong khi thực hiện chương trình. Ví dụ: Đường dẫn thư mục hiện hành *"C:\Program Files\AutoCAD\Help"*, hoặc thời gian hiện hành : "*The time is now 0800 hours and 37 seconds*".

Ta không nên sử dụng *text* để hiển thị tiêu đề của các *tile* khác, vì mỗi *tile* đã có thuộc tính *label* để hiển thị tiêu đề.

Ta không nên sử dụng *text* để hiện các thông báo lỗi, mà nên sử dụng *error tile* để hiện thông báo lỗi ở góc trái dưới của hộp thoại, tương tự như các hộp thoại của **AutoCAD**.

label Cách thứ nhất để hiện chuỗi. Sử dụng để hiện nội dung ban đầu cho *text*. Chiều dài của chuỗi chứa trong *label* sẽ được so sánh với giá trị của thuộc tính *width*. Giá trị lớn hơn sẽ được sử dụng làm chiều rộng của *text*. Do đó, ta phải khai báo ít nhất một trong hai giá trị này.

value Cách thứ hai để hiện chuỗi. Sử dụng để thay đổi nội dung khi chương trình đang chạy. Thuộc tính này không ảnh hưởng đến chiều rộng của *text*.

Ví dụ:

```
: text {
    label = "C:\\Program Files\\AutoCAD\\Help";
}
```

Text Part


```
:text_part {
  label
}
```

Text_part sử dụng để hiển thị các thông tin *động*, kết nối với các thông tin tĩnh tạo thành nội dung của *text tile*. Ví dụ: Trong chuỗi "*The time is now 0800 hours and 37 seconds*", ta sử dụng *text_part* để hiển thị các số *0008* và *37*. Các giá trị này thay đổi trong khi chạy chương trình.

Ta có thể sắp xếp các *text_part* theo phương thẳng đứng để tạo ra dòng chữ theo phương thẳng đứng.

Text_part là một *prototype,* được mô tả trong file *base.dcl* của **AutoCAD**.

Concatenation


```
:concatenation {
}
```

Sử dụng để kết hợp nhiều *text_part* theo hàng ngang tạo thành một dòng chữ.

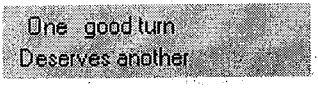
Concatenation là một *prototype,* được mô tả trong file *base.dcl* của **AutoCAD**.

Ví dụ:

```
: concatenation {
  : text_part {
  label = "Text_part1 and";
```

```
        }
        : text_part {
          label = "text_part2";
        }
    }
```

Paragraph




```
: paragraph {

}
```

Paragraph là một *cluster* sử dụng để chứa các *text_part* và *concatenation* và sắp xếp chúng thành một cột theo thứ tự từ trên xuống.

Ví dụ:

```
: paragraph {
    : concatenation {
        : text_part {
            label = "One";
            }
        : text_part {
            label = "good turn";
         }
    }
    : text_part {
            label = "Deserves another";
    }
}
```

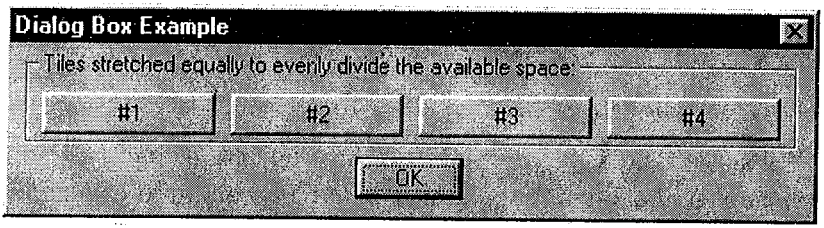
Spacer

```
: spacer {
        alignment fixed_height fixed_width
        height width
}
```

Khi thiết kế hộp thoại, chúng ta nên để cho **AutoCAD** tự động điều chỉnh khoảng cách giữa các *tile* trong hộp thoại. Tuy nhiên, chúng ta vẫn có thể điều chỉnh khoảng cách và cách sắp xếp giữa các *tile* bằng cách sử dụng *spacer. Spacer* không xuất hiện trên màn hình và người sử dụng không thể chọn được.

Ví dụ:

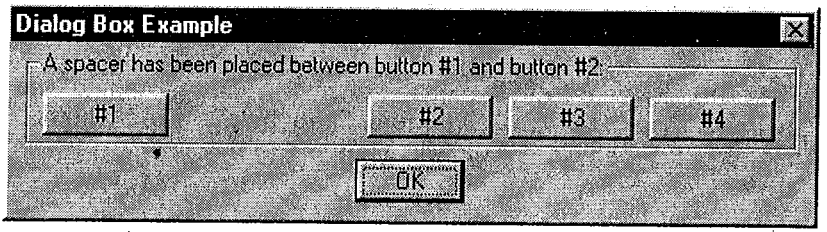
Hộp thoại sau đây có 4 *button* #1 #2 #3 và #4. Các *button* được kéo dãn ra cho vừa khít các khoảng trống của *boxed_row.*




```
sample : dialog {
          value = "Dialog Box Example";
       :boxed_row {
          label= "Tiles stretched equally to evenly divide the available space:";
          : button {label = "#1";}
          : button {label = "#2";}
          : button {label = "#3";}
          : button {label = "#4";}
       }
       ok_only;
    }
```

Ví dụ:

Trong hộp thoại sau đây, có một *spacer* (chiều rộng width = 10) đặt giữa *button* #1 và #2.



```
sample : dialog {
          value = "Dialog Box Example";
        :boxed_row {
         label= "A spacer has been placed between button #1 and button #2:";
         : button {label = "#1";}
         : spacer {width = 10;}
         : button {label = "#2";}
         : button {label = "#3";}
         : button {label = "#4";}
        }
        ok_only;
    }
```

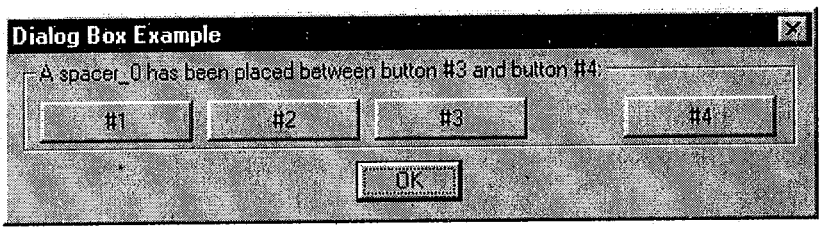
Spacer_0

: spacer_0;

Spacer_0 được thiết kế sẵn trong file *base.dcl* của **AutoCAD**. Mặc định ban đầu nó có chiều rộng bằng 0. Khi các *tile* trong hộp thoại được kéo dãn, *spacer_0* cũng được kéo dãn thích hợp.

Ví dụ:

Hộp thoại sau đây có một *spacer_0* đặt giữa *button* #3 và #4.



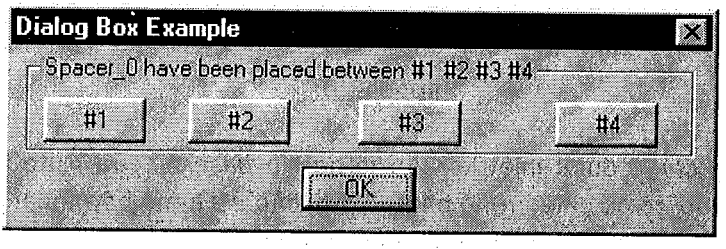

```
sample : dialog {
        value  = "Dialog Box Example";
        :boxed_row {
         label= "A spacer_0 has been placed between button #3 and button
                 #4:";
        : button {label = "#1";}
        : button {label = "#2";}
```

```
    : button {label = "#3";}
      spacer_0;
     : button {label = "#4";}
    }
    ok_only;
  }
```

Ví dụ:

Hộp thoại sau đây có:

- 1 spacer_0 đặt giữa #1 và #2
- 2 spacer_0 đặt giữa #2 và #3
- 3 spacer_0 đặt giữa #3 và #4




```
sample : dialog {
    value  = "Dialog Box Example";
     :boxed_row {
        label= "Spacer_0 have been placed between #1 #2 #3 #4";
     : button {label = "#1";}
     spacer_0;
     : button {label = "#2";}
     spacer_0; spacer_0;
     : button {
       label = "#3";}
     spacer_0; spacer_0; spacer_0;
     : button {label = "#4";}
    }
    ok_only;
  }
```

Space_1

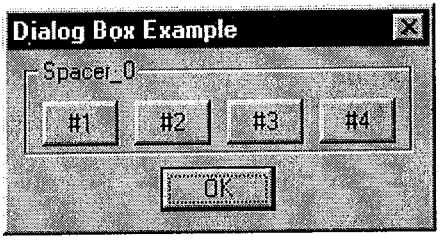
spacer_1;

Spacer_1 được thiết kế sẵn trong file *base.dcl* của **AutoCAD**. Nó có chiều rộng và chiều cao ban đầu bằng 1. Khi các *tile* trong hộp thoại được kéo dãn, *spacer_1* cũng được kéo dãn thích hợp giống như *spacer_0*. Tuy nhiên, chiều rộng của *spacer_0* có thể thu hẹp bằng 0, còn chiều rộng của *spacer_1* chỉ có thể thu hẹp nhỏ nhất bằng 1, vừa đủ để người sử dụng nhận biết được khoảng trống này.

Hãy so sánh 2 ví dụ sau đây:

Ví dụ:

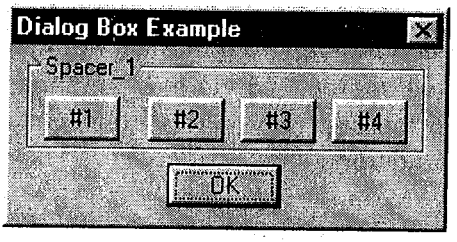
Hộp thoại sau đây có một *spacer_0* giữa #1 và #2. Nhưng nó bị thu hẹp bằng 0.




```
sample : dialog {
        value    = "Dialog Box Example";
        :boxed_row {
         label= "Spacer_0";
         : button {label = "#1";}
         spacer_0;
        : button {label = "#2";}
        : button {label = "#3";}
        : button {label = "#4";}
       }
       ok_only;
    }
```

Ví dụ:

Hộp thoại sau đây có một *spacer_1* ở giữa button #1 và #2. *Spacer_1* không thể thu hẹp lại bằng 0.




```
sample : dialog {
        value    = "Dialog Box Example";
        :boxed_row {
         label= "Spacer_1";
         : button {label = "#1";}
         spacer_1;
         : button {label = "#2";}
         : button {label = "#3";}
         : button {label = "#4";}
        }
        ok_only;
    }
```

Các Error tile và Exit tile

Tất cả các hộp thoại đều phải có ít nhất một *exit tile*. Thông thường đó là các button: OK và Cancel.

Error tile là một *tile* xuất hiện tại góc trái dưới của hộp thoại. ta có thể lập trình để nó hiển thị thông báo lỗi khi dữ liệu nhập cho các *tile* không phù hợp.

Bởi vì các *tile* này hầu như có mặt trong tất cả các hộp thoại, nên **AutoCAD** đã tạo sẵn các *subassembly* để dễ sử dụng. Ta chỉ cần gọi tên các *subassembly* này mà không cần phải khai báo các thuộc tính.

Errtile

errtile;

Errtile sử dụng để hiện các thông báo lỗi. *Errtile* nên được đặt ở góc trái dưới của hộp thoại (xuất hiện cuối cùng trong file DCL).

Errtile được khai báo trong file base.dcl và đã được gán sẵn tên (*key*) là *"error"*. Khi nào cần làm xuất hiện thông báo lỗi, ta sử dụng hàm **Set_tile** của **AutoLISP**.

Ví dụ:

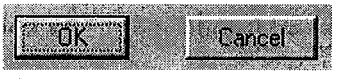
```
(setq DCL_ID (load_dialog "Sample.DCL"))
(new_dialog "sample" DCL_ID)
(set_tile "error" "Invalid input value")
(start_dialog)
```

Ok_only

ok_cancel;

Nút **OK** thường được sử dụng trong các hộp thoại hiển thị thông tin hoặc thông báo lỗi. Nó được khai báo trong file Base.dcl và có khóa (*key*) là *"accept"*.

Ok_cancel

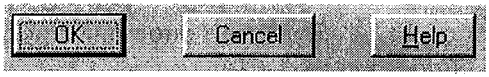


ok_cancel;

Hai nút **OK** và **Cancel** thường được sử dụng trong các hộp thoại cho phép người sử dụng thoát khỏi một chức năng đang thực hiện, hoặc thi hành chức năng này.

Nút **OK** có khóa là *"accept"*. Nút **Cancel** có khóa là *"cancel"*.

Ok_cancel_help

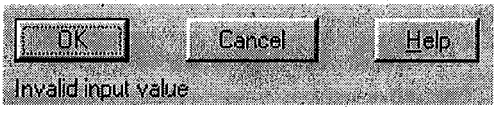


ok_cancel_help;

Trong các hộp thoại phức tạp thường có thêm nút **Help.** Khóa của nút này là *"help"*.

Ta phải liên kết nút này với các biểu thức **AutoLISP** để hiển thị hộp thoại Help của **AutoCAD**.

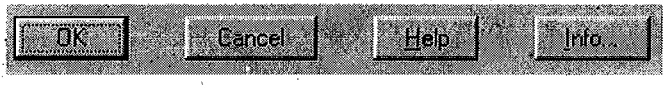
Ok_cancel_help_errtile



ok_cancel_help_errtile;

Subassembly này có tất cả 4 *tile*: *OK*, *Cancel*, *Help* và *errtile*.

Ok_cancel_help_info



ok_cancel_help_info;

Info button thường được người lập trình sử dụng để hiển thị các thông tin như logo, địa chỉ của công ty, phiên bản của phần mềm, địa chỉ liên lạc...

Khóa của nút Info là *"info"*.

Các thuộc tính của *tile*

Phần này sẽ trình bày chi tiết hơn về các thuộc tính của các *tile*.

action = "(setq VAR 3)";

Biểu thức **AutoLISP** chứa trong thuộc tính *action* sẽ được thực hiện khi *tile* được chọn (con trỏ chuyển đến *tile* này). Đối với một vài *tile*, biểu thức chứa trong thuộc tính *action* cũng sẽ được thực hiện khi người sử dụng chuyển qua *tile* khác (con trỏ rời khỏi *tile* này và chuyển sang *tile* khác).

Thông thường trong thuộc tính *action*, ta chỉ nên gán biểu thức đơn giản. Với các biểu thức phức tạp, ta sử dụng hàm **Action_tile** trong chương trình **AutoLISP**.

Chú ý:

Biểu thức **AutoLISP** phải được chứa trong dấu ngoặc kép.

alignment = left;

Thuộc tính *alignment* xác định vị trí canh lề theo phương nằm ngang hoặc thẳng đứng của *tile* khi nó được đặt trong một *cluster*.

Các giá trị có thể gán cho thuộc tính này là các từ dành riêng sau đây:

- Khi chứa trong *column: left, right, centered* (mặc định: *left*).
- Khi chứa trong *row*: *top, bottom, centered* (mặc định: *centered)*.

Để sắp xếp vị trí cho các *tile* theo phương thẳng đứng (khi chứa trong *column*) hoặc theo phương ngang (khi chứa trong *row)*, ta phải sử dụng các *spacer*, mà không thể sử dụng thuộc tính này được.

allow_accept = true

Mặc định là *false*.

Khi giá trị này bằng *true* và nếu *tile* đang nhận con trỏ và ta nhấn phím ENTER, thì:

- Biểu thức chứa trong thuộc tính *action* của *tile* này (nếu có) sẽ được thi hành.
- Sau đó, *tile* nào có khóa là *accept* (thường là nút OK) sẽ được chọn để đóng hộp thoại.

aspect_ratio = 1.0;

Tỉ lệ giữa chiều rộng và chiều cao của hình ảnh xuất hiện trên *image tile* hoặc *image_button*.

Khi thuộc tính này bằng 0, kích thước *tile* sẽ đúng bằng kích thước hình ảnh. Lúc này, kích thước hình ảnh sẽ do chương trình **AutoLISP** xác định khi gọi hiển thị hộp thoại.

Thuộc tính này không có giá trị mặc định, và nếu không gán giá trị cho thuộc tính này, ta phải cung cấp đầy đủ chiều cao và chiều rộng hình ảnh.

big_increment = 10;

Xác định độ dời con trượt khi ta nhấn chuột trên thân của *slider*.

Giá trị mặc định bằng 1/10 miền giá trị xác định bằng 2 thuộc tính: *max_value* và *min_value*.

children_alignment = right;

Thuộc tính này sử dụng để xác định vị trí canh lề theo phương nằm ngang hoặc thẳng đứng của các *tile* con trong *cluster*, khi các *tile* con không được khai báo thuộc tính *alignment* của mình.

Các giá trị có thể gán cho thuộc tính này là các từ dành riêng sau đây:

- Khi chứa trong *column: left, right, centered* (mặc định: *left*).
- Khi chứa trong *row*: *top, bottom, centered* (mặc định: *centered).*

Để sắp xếp vị trí cho các *tile* theo phương thẳng đứng (khi chứa trong *column*) hoặc theo phương ngang (khi chứa trong *row),* ta phải sử dụng các *spacer*, mà không thể sử dụng thuộc tính này được.

children_fixed_height = true;

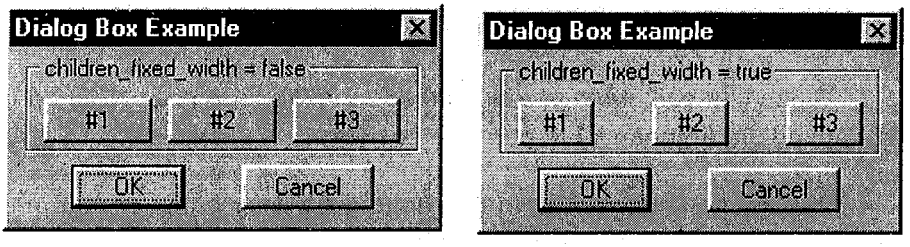
children_fixed_width = true;

Khi thuộc tính này bằng *true*, các *tile* con sẽ giữ cố định chiều cao (hoặc chiều rộng). Khi bằng *false*, các *tile* con sẽ thay đổi chiều cao hoặc chiều rộng cho phù hợp với các khoảng trống trong *cluster*.

Thuộc tính này không có tác dụng khi các *tile* con được khai báo chính xác chiều cao *height* (hoặc chiều rộng *width*).

Ví dụ:

Sau đây là 2 hộp thoại, có thuộc tính *children_fixed_width* được khai báo tương ứng bằng *false* và *true*. Các *button* #1, #2, #3 trong trường hợp thứ nhất được tự động kéo rộng ra cho phù hợp với kích thước của *boxed_row*. Trong trường hợp thứ hai, chiều rộng các *button* không đổi.



color = dialog_background;

color = white;

Xác định màu nền của *image tile*. Các giá trị có thể gán cho thuộc tính này được liệt kê trong bảng sau đây:

Giá trị	Ý nghĩa
dialog_line	Màu đường viền của hộp thoại
dialog_foreground	Màu chữ trong hộp thoại.
dialog_background	Màu nền của hộp thoại.
graphics_foreground	Màu hiện hành của các đối tượng vẽ trên màn hình đồ họa của **AutoCAD**.
graphics_background	Màu nền của màn hình đồ họa của **AutoCAD**

	(thường bằng 0).
black	= 0 (màu đen)
red	= 1 (màu đỏ)
yellow	= 2 (màu vàng)
green	= 3 (màu xanh lá cây)
cyan	= 4 (màu lục)
blue	= 5 (màu xanh dương)
magenta	= 6 (màu đỏ tươi)
white	= 7 (màu trắng)

edit_limit = 32;

Số nguyên xác định số lượng lớn nhất các ký tự có thể nhập vào *edit_box*. Giá trị mặc định là 132. Giá trị lớn nhất cho phép là 256.

edit_width = 16;

Một *edit_box* gồm 2 phần: phần tiêu đề và phần nhập ký tự.

Thuộc tính *edit_width* xác định chiều rộng (số lượng các ký tự có thể nhìn thấy) của phần nhập ký tự của *edit_box*.

Nếu không khai báo giá trị này hoặc giá trị này bằng 0, phần nhập ký tự sẽ mở rộng hết không gian cho phép.

fixed_height = true;

fixed_width = true;

Khi thuộc tính này bằng *true*, *tile* sẽ giữ cố định chiều cao (hoặc chiều rộng). Khi bằng *false*, *tile* sẽ thay đổi chiều cao hoặc chiều rộng cho phù hợp.

Thuộc tính này không có tác dụng nếu ta khai báo chiều cao *height* hoặc chiều rộng *width* của *tile*.

fixed_width_font = true;

Nếu bằng *true*, thuộc tính này làm cho các *list_box* và *popup_list* hiển thị chữ bằng font có chiều rộng các ký tự bằng nhau. Nhờ đó ta dễ canh lề cho các ký tự TAB và các khoảng trắng.

height = 3;

Thuộc tính này chứa số nguyên hoặc số thực để xác định chiều cao của *tile*. Đơn vị tính chiều cao trong trường hợp này là *chiều cao ký tự* (chiều cao của ký tự trên màn hình, kể cả khoảng cách giữa 2 dòng).

initial_focus = "tile_key";

Xác định *tile* được nhận con trỏ ban đầu, khi hộp thoại vừa xuất hiện.

is_bold = true;

Khi bằng *true*, thuộc tính này làm cho các chữ xuất hiện được in đậm.

is_cancel = true;

Khi thuộc tính này bằng *true*, *button* này sẽ được "nhấn", khi người sử dụng nhấn phím ESC. Khi đó, biểu thức **AutoLISP** chứa trong thuộc tính *action* của *button* này sẽ được thi hành. Nếu biểu thức này không có lệnh đóng hộp thoại, thì hộp thoại cũng được tự động đóng lại sau khi biểu thức được tính toán xong.

is_default = true;

Nếu bằng *true*, *button* sẽ được "nhấn" khi người sử dụng nhấn phím ENTER. Chỉ có duy nhất một *button* có thuộc tính này bằng *true*. Giá trị mặc định là *false*.

is_enabled = false;

Xác định trạng thái ban đầu của *tile*, và có thể thay đổi bằng chương trình **AutoLISP**. Nếu bằng *true*, trạng thái là *enabled* . Nếu bằng *false*, trạng thái là *disabled* (bị làm mờ đi).

is_tab_stop = *true*;

Khi hộp thoại đang xuất hiện trên màn hình, nếu ta liên tục nhấn phím TAB, con trỏ sẽ lần lượt di chuyển qua các *tile*. Trong quá trình này, con trỏ sẽ bỏ qua các *tile* nào có thuộc tính *is_tab_stop = false*. Giá trị mặc định là *true*.

key = "key_name";

Tên (khóa) của *tile*, sử dụng để truy xuất khi chạy chương trình. Trong một hộp thoại, tên của các *tile* không được trùng nhau. Tên này có kiểu chuỗi, và phân biệt dạng chữ hoa chữ thường.

label = "Label text";

Xác định tiêu đề của *tile*. Ta có thể xác định phím nóng cho *tile* bằng cách đặt dấu (&) trước ký tự phím nóng. Ví dụ:

label = "&Pick point<"

Trên màn hình sẽ xuất hiện tiêu đề "Pick point<". Khi nhấn tổ hợp phím ALT+P, con trỏ sẽ chuyển về *tile* này.

Ta cũng có thể xác định phím nóng cho *tile* bằng thuộc tính *mnemonic*.

layout = vertical;

Xác định hướng của thanh *slider* (nằm ngang hoặc thẳng đứng). Mặc định là nằm ngang *horizontal*.

list = "List_item1\nList_item2\nList_item3";

Chuỗi chứa trong dấu nháy kép, xác định nội dung các mục ban đầu cho *list_box* và *popup_list*. Tất cả các mục đều được đặt trong một chuỗi, ngăn cách nhau bằng ký tự xuống dòng "\n". Cho phép sử dụng ký tự TAB "\t" trong chuỗi.

max_value = 1000;

min_value = 1;

Hai thuộc tính này xác định giá trị lớn nhất và nhỏ nhất của thanh *slider*. Các giá trị mặc định là *max_value = 1000* và *min_value = 1*.

Các giá trị có thể gán cho các thuộc tính này từ –32768 đến 32767.

mnemonic = "d";

Xác định ký tự sử dụng làm phím nóng. Tuy nhiên ta nên gán phím nóng trong thuộc tính *label*.

multiple_select = true;

Khi thuộc tính này bằng *true*, ta có thể chọn nhiều mục trong *list_box* cùng một lúc. Giá trị mặc định là *false*.

password_char = "*";

Trong một số trường hợp, ta cần giấu các ký tự nhập vào từ bàn phím. Ví dụ như khi người sử dụng nhập vào mật khẩu. Khi đó, ta có thể sử dụng các ký tự thay thế (thông thường là "*" hoặc "-") để hiển thị các ký tự thật do người sử dụng nhập vào.

Thuộc tính *password_char* sử dụng để chứa ký tự thay thế này.

small_increment = 1;

Xác định độ dời của con trượt khi ta nhấn chuột vào các mũi tên của *slider.*

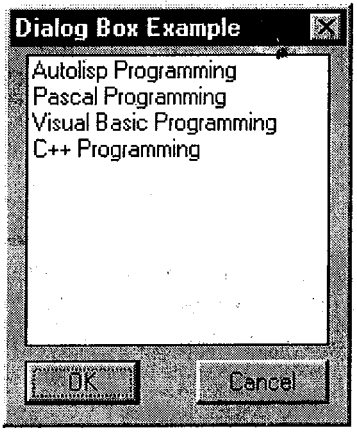
Giá trị mặc định bằng 1/100 miền giá trị xác định bằng 2 thuộc tính: *max_value* và *min_value.*

tabs = "4 8 12 16";

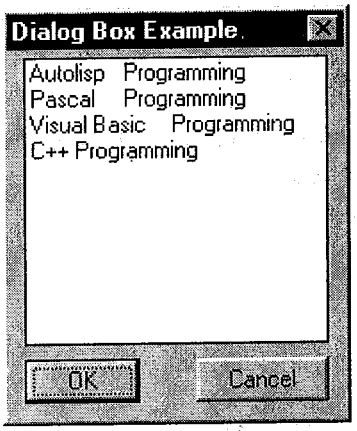
Xác lập vị trí các TAB trong danh sách của *list_box* và *popup_list,* chia danh sách thành nhiều cột. Đơn vị tính là chiều rộng ký tự. Ví dụ: Chuỗi "4 8 12 16" xác định 4 vị trí TAB. Mỗi vị trí cách nhau 4 ký tự.

Ví dụ:

Sau đây là 2 hộp thoại không sử dụng và có sử dụng các ký tự TAB. Trong hộp thoại thứ hai, chuỗi "AutoLISP" và "Pascal" có chiều dài vượt quá vị trí tab 4, nên chuỗi "Programming" bắt đầu tại vị trí tab 8. Chuỗi "Visual Basic" có chiều dài vượt quá vị trí tab 8, nên chuỗi "Programming" bắt đầu tại vị trí tab 16. Chuỗi "C++" không vượt quá vị trí tab 4, nên chuỗi "Programming" bắt đầu tại vị trí tab 4.




```
:list_box {
        list = "Autolisp Programming
                \nPascal Programming
                \nVisual Basic Programming
                 \nC++ Programming";
        }
```



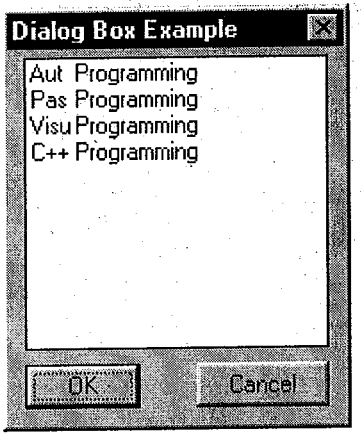

```
:list_box {
        list = "Autolisp \tProgramming
                \nPascal\tProgramming
                \nVisual Basic\tProgramming
                \nC++\tProgramming";
        tabs = "4 8 12 16";
 }
```

tab_truncate = *true*;

Khi bằng *true*, nếu chuỗi dài hơn khoảng cách giữa 2 TAB thì chuỗi sẽ bị xén bớt.

Ví dụ:

Trong hộp thoại sau, vì các chuỗi "Autolisp", "Pascal" và "Visual Basic" dài hơn 4 ký tự (khoảng cách của tab đầu tiên), nên chúng bị xén bớt. Chuỗi "C++" chỉ có 3 ký tự nên không bị xén bớt. Các chuỗi "Programming" đều bắt đầu tại tab 4.




```
:list_box {
        list = "Autolisp \tProgramming
                \nPascal\tProgramming
                \nVisual Basic\tProgramming
                \nC++\tProgramming";
        tabs = "4 8 12 16";
        tab_truncate = true;
        }
```

value = "1";

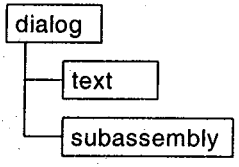
Kiểu **chuỗi**, xác định giá trị ban đầu của *tile*.

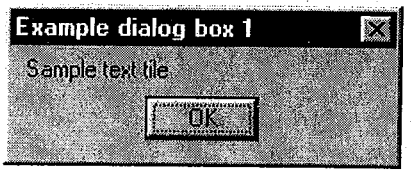
width = 8;

Thuộc tính này chứa số nguyên hoặc số thực để xác định chiều rộng của *tile*. Đơn vị tính chiều rộng trong trường hợp này là *ký tự*.

17.3 Các ví dụ mẫu

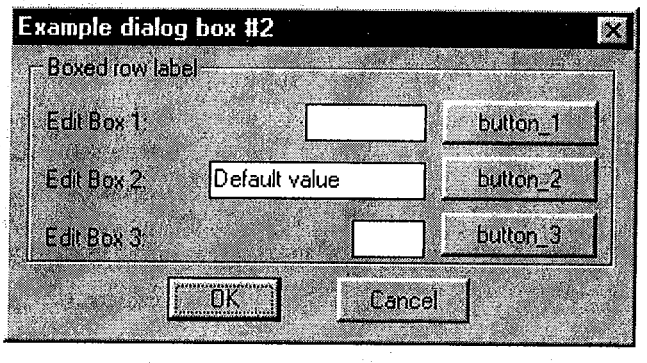
Ví dụ 1:





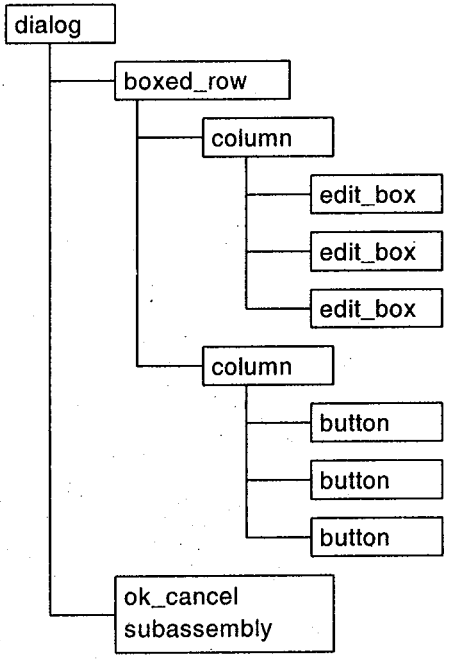

```
// Sample Dialog Box Code
example_1 : dialog {
    label = "Example dialog box 1";
    : text {
      label = "Sample text tile";
    }
    ok_only;
{
```

Ví dụ 2:



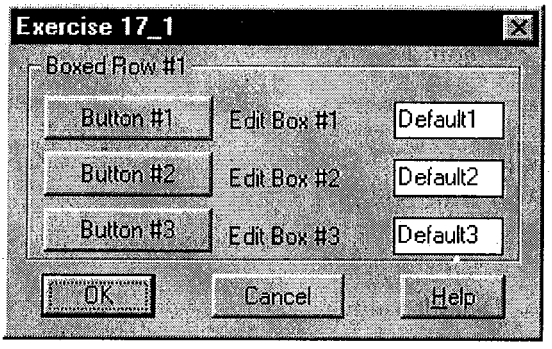

```
example_2: dialog {
        label = "Example dialog box #2";
        : boxed_row {
```

```
    label = "Boxed row label";
    : column {
      :edit_box {
       label      = "Edit Box 1: ";
       edit_width = 8;
       key        = "edit_1";
      }

      :edit_box {
       label      = "Edit Box 2: ";
       edit_width = 16;
       key        = "edit_2";
       value  = "Default value";
      }

      :edit_box {
       label      = "Edit Box 3: ";
       edit_width = 4;
       key        = "edit_3";
      }
    }

    : column {
      : button {
        label     = "button_1";
        key       = "btn_1";
      }
      : button {
       label     = "button_2";
        key       = "btn_2";
      }
      : button {
        label     = "button_3";
        key       = "btn_3";
      }
    }
  }
  ok_cancel;
}
```



17.4 Bài tập

1. Tạo file EX17-1.DCL, mô tả hộp thoại sau đây. Tên của hộp thoại là "dgl_1"



17.5 Lời giải

1.

```
//Tên file EX17-1.DCL
//
dlg_1 : dialog {
        label = "Exercise 17_1";
        : boxed_row {
          label = "Boxed Row #1";
          : column {
            : button {label = "Button #1"; }
            : button {label = "Button #2"; }
            : button {label = "Button #3"; }
          }                              /* Dong column #1 */
          : column {
            : edit_box {label = "Edit Box #1"; value = "Default1"; }
            : edit_box {label = "Edit Box #2"; value = "Default2"; }
            : edit_box {label = "Edit Box #3"; value = "Default3"; }
          }                                        /* Dong column #2 */
        }                                          /* Dong boxed_row */
        ok_cancel_help;
}
// Kết thúc file
```

Chương 18

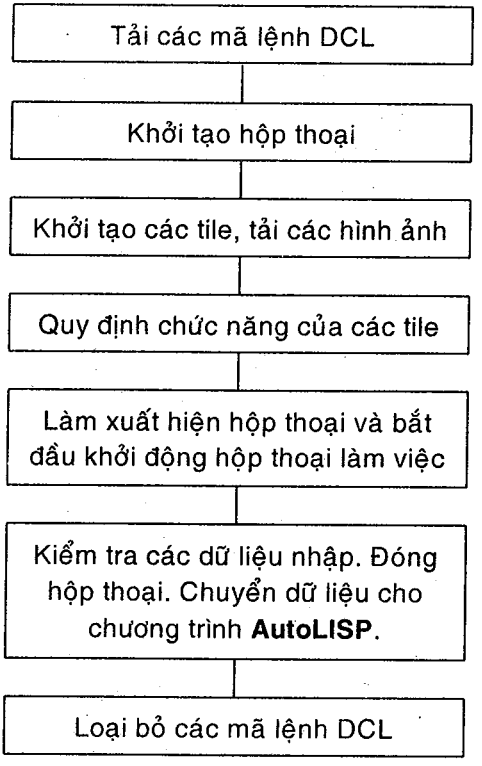
CÁC HÀM ĐIỀU KHIỂN HỘP THOẠI

Nội dung chương:

1. Các hàm điều khiển hộp thoại.
2. Thiết kế và điều khiển chức năng của các *tile*.
3. Cập nhật giá trị cho hộp thoại và chuyển các giá trị của hộp thoại cho chương trình.

AutoLISP cung cấp một số hàm để điều khiển hộp thoại như: tải các mã lệnh DCL, kiểm tra chức năng của các *tile*, hiển thị và cập nhật các hình ảnh, dòng chữ xuất hiện trên hộp thoại, xử lý số liệu do người sử dụng nhập vào, đóng hộp thoại . . .

Để làm xuất hiện hộp thoại, chương trình **AutoLISP** thực hiện các bước cơ bản sau:



18.1 Các hàm điều khiển hộp thoại

Hàm LOAD_DIALOG

Cấu trúc của các hộp thoại được mô tả trong các file DCL. Trước tiên, các mã này phải được tải vào bộ nhớ. Sau đó, **AutoLISP** mới có thể làm xuất hiện và điều khiển hộp thoại trên màn hình.

Hàm **Load_dialog** dùng để mở file DCL và chuyển các dòng mô tả hộp thoại và các tile của hộp thoại vào bộ nhớ.

(**Load_dialog** DCLFILE)

Tham số DCLFILE là chuỗi chứa tên file DCL muốn tải. Tên mở rộng mặc định của file là .DCL. Nếu file chứa trong đường dẫn thư viện thì

không cần phải ghi rõ đường dẫn thư mục. Ngược lại, ta phải ghi rõ đường dẫn thư mục (nhớ sử dụng ký hiệu \\ cho đường dẫn).

Giá trị trả về của hàm bằng:

- -1, nếu **không** tải được file DCL vào bộ nhớ. Các lỗi có thể là do sai đường dẫn, các mã DCL sai cú pháp . . .
- số nguyên lớn hơn 0, nếu tải được file DCL vào bộ nhớ. Giá trị này sử dụng để truy xuất đến các mã lệnh DCL.

✌ **Ví dụ:** Tải file SAMPLE.DCL vào bộ nhớ.

Command: **(setq DCL_ID (Load_dialog "Sample.dcl"))** ↵
1

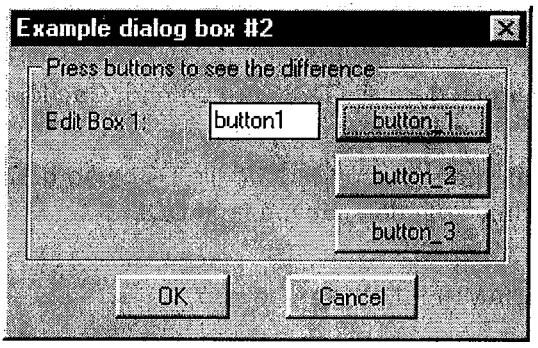
Hàm NEW_DIALOG

Hàm **New_dialog** dùng để khởi tạo một hộp thoại có cấu trúc được mô tả bằng các mã DCL đã được tải vào bộ nhớ.

(**New_dialog** DLGNAME DCL_ID [ACTION-EXPRESSION [SCREEN-PT]])

DLGNAME	Tên của hộp thoại.
DCL_ID	Số nguyên do hàm **Load_dialog** trả về.
ACTION-EXPRESSION	Chuỗi chứa biểu thức **AutoLISP**. Các tile của hộp thoại không được khai báo chức năng trong thuộc tính *action* sẽ sử dụng biểu thức này để làm chức năng mặc định cho mình.
SCREEN-PT	Vị trí xuất hiện hộp thoại trên màn hình. Tham số này chứa tọa độ *góc trái trên* của hộp thoại, được biểu diễn trong hệ trục tọa độ màn hình của hệ điều hành Window: gốc tọa độ (0,0) trùng với góc trái trên của màn hình, trục X hướng về bên trái, trục Y hướng xuống dưới. Theo mặc định, hộp thoại sẽ xuất hiện chính giữa màn hình.

Ví dụ:



File EX18-1.DCL

```
example_2: dialog {
        label = "Example dialog box #2";
        : boxed_row {
          label = "Press buttons to see the difference";
          : column {
            :edit_box {label= "Edit Box 1: "; key = "edit_1"; }
          }
          : column {
            : button {label= "button_1"; key = "btn_1";
                      action = "(set_tile \"edit_1\" \"button1\")";
                     }
            : button {label= "button_2"; key = "btn_2";
                      action = "(set_tile \"edit_1\" \"button2\")";
                     }
            : button {label= "button_3"; key = "btn_3"; }
          }
        }
        ok_cancel;
}
```

File EX18-1.LSP

```
(setq DCL_ID (load_dialog "EX18-1.DCL"))
(new_dialog "example_2" DCL_ID "(set_tile \"edit_1\" \"Default\")")
(start_dialog)
```

Giải thích

Thuộc tính *action* của các button_1 và button_2 chứa biểu thức **AutoLISP** sử dụng để gán giá trị cho edit_box là "button1" và "button2" tương ứng. Do đó, khi ta nhấn nút button_1, tại edit_box xuất hiện chuỗi "button1". Khi ta nhấn nút button_2, tại edit_box xuất hiện chuỗi "button2".

Thuộc tính *action* của button_3 **không** chứa biểu thức **AutoLISP**. Tuy nhiên, vì tham số ACTION-EXPRESSION của hàm **New_dialog** có chứa biểu thức gán giá trị cho edit_box là "Default", nên khi ta nhấn nút button_3, tại edit_box xuất hiện chuỗi "Default". Tham số này không ảnh hưởng đến các nút button_1 và button_2.

Hàm START_DIALOG

Hàm **Start_dialog** làm xuất hiện hộp thoại được tạo ra bằng hàm **New_dialog** và bắt đầu khởi động hộp thoại làm việc.

Hàm này không có tham số:

(**Start_dialog**)

Hộp thoại sẽ xuất hiện trên màn hình cho đến khi được đóng lại bằng hàm **Done_dialog**. Giá trị trả về của hàm **Start_dialog** tùy thuộc vào kết quả của hàm **Done_dialog** được gọi.

Hàm DONE_DIALOG

Hàm **Done_dialog** dùng để đóng hộp thoại. Hàm này có thể được chứa trong thuộc tính *action*, hoặc được gọi bởi hàm **Action_*tile***, hoặc được tự động gọi bởi các *exit tile* như OK, CANCEL.

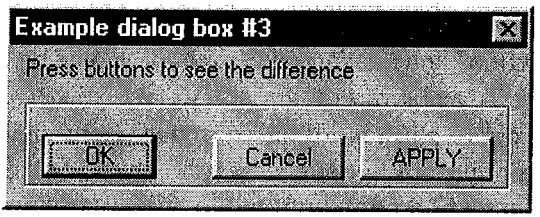
(**done_dialog** [STATUS])

Tham số STATUS là một số nguyên. Khi hàm **done_dialog** được gọi, số nguyên này sẽ được truyền cho hàm **Start_dialog**. Hàm **Start_dialog** sẽ trả về giá trị này cho chương trình **AutoLISP**. Nhờ đó, chương trình có thể nhận biết được người sử dụng đã đóng hộp thoại bằng cách nào để xử lý cho phù hợp.

Chú ý:

Khi nhấn nút OK, giá trị trả về là 1. Khi nhấn nút CANCEL, giá trị trả về là 0.

Ví dụ:



File EX18-2.DCL

```
example_3: dialog {
            label = "Example dialog box #3";
            : text {label = "Press buttons to see the difference"; }
            : boxed_row {
            ok_cancel;
            : button {label= "APPLY"; key = "btn_1";
                    action = "(done_dialog 2)";
                   }
            }
          }
```

File EX18-2.LSP

```
(setq DCL_ID (load_dialog "EX18-2.DCL"))
(new_dialog "example_3" DCL_ID)
(setq RES (start_dialog))          ;Làm xuất hiện hộp thoại và lưu giữ giá trị
                                   ;trả về khi đóng hộp thoại trong biến RES
(cond                    ;Kiểm tra biến RES để biết nút nào được nhấn
  ((= RES 0) (princ "\nCANCEL has been pressed."))  ;Nhấn nút CANCEL
  ((= RES 1) (princ "\nOK has been pressed."))      ;Nhấn nút OK
  ((= RES 2) (princ "\nAPPLY has been pressed."))   ;Nhấn nút APPLY
)
```

Bản thân hàm **Done_dialog** trả về tọa độ điểm xác định vị trí của hộp thoại trên màn hình tại thời điểm đóng hộp thoại. Ta có thể truyền tọa độ điểm này cho hàm **New_dialog** (tham số SCREEN-PT) để hiển thị hộp thoại kế tiếp tại đúng vị trí này.

Hàm TERM_DIALOG

Vì mỗi hộp thoại có thể gọi xuất hiện hộp thoại khác, nên trên màn hình có thể xuất hiện cùng lúc nhiều hộp thoại. Để nhanh chóng đóng tất cả các hộp thoại, ta sử dụng hàm **Term_dialog**.

(**Term_dialog**)

Hàm này không có tham số và luôn luôn trả về *nil*.

Tuy nhiên, ta nên đóng lần lượt từng hộp thoại để tránh gây ra lỗi.

Hàm UNLOAD_DIALOG

Sau khi đóng hộp thoại, ta nên loại bỏ các mã lệnh của file DCL không còn sử dụng trong bộ nhớ nữa. Nhờ đó, tránh được các trường hợp gây lỗi do có nhiều file DCL tải vào bộ nhớ không tương thích với nhau.

(**Unload_dialog** DCL-ID)

Tóm tắt

1. Để làm xuất hiện và làm việc với hộp thoại, ta cần thực hiện các bước sau đây:
   a) Tải file DCL chứa các dòng mô tả cấu trúc hộp thoại.
   b) Khởi tạo một hộp thoại mới theo cấu trúc này.
   c) Khởi tạo các *tile* và tải các hình ảnh.
   d) Quy định chức năng (*action*) cho hộp thoại và các *tile*.
   e) Làm xuất hiện hộp thoại trên màn hình và bắt đầu khởi động hộp thoại làm việc.
   f) Kiểm tra các dữ liệu nhập. Đóng hộp thoại. Chuyển dữ liệu cho chương trình **AutoLISP**.
   g) Loại bỏ file DCL ra khỏi bộ nhớ.
2. Hàm **Load_dialog** dùng để mở file DCL và chuyển các dòng mô tả cấu trúc hộp thoại và các tile của hộp thoại vào bộ nhớ.
3. Hàm **New_dialog** dùng để khởi tạo hộp thoại. Hàm **Start_dialog** khởi động hộp thoại làm việc. Tất cả các lệnh bổ sung chức năng, giá trị cho các *tile* phải được đặt ở đoạn giữa hai hàm này.

4. Hàm **Done_dialog** chỉ đóng hộp thoại hiện hành. Hàm **Term_dialog** đóng tất cả các hộp thoại đang xuất hiện trên màn hình.
5. Việc loại bỏ các mã lệnh DCL khỏi bộ nhớ sau khi đóng hộp thoại ngăn ngừa được các lỗi gây ra do ảnh hưởng qua lại của các file DCL được tải vào bộ nhớ cùng lúc.

Ví dụ mẫu

Tạo 2 file EX4.DCL và EX4.LSP như sau:

File EX18-3.DCL

```
// Tên file: EX18-3.DCL                                    //[1]
//Mục đích: Mô tả hộp thoại hiển thị thông báo: Dialog session example
EX3 : dialog {                                            //[2]
      label  = "Dialog Handling Example";
       : text {                                           //[3]
         label = "Dialog session example";
       }
       ok_only;                                           //[4]
    }
// Kết thúc file EX18-3.DCL
```

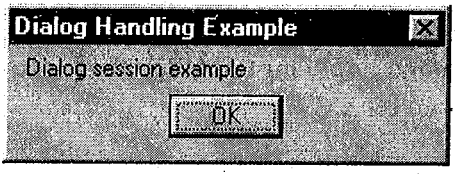
File EX18-3.LSP

```
;Tên file : EX18-3.LSP                                    ; [5]
;Mục đích: Hiển thị hộp thoại EX3 chứa trong file EX18-3.DCL
(defun C:EX3 (/ DCL_ID)                                   ; [6]
  (setq DCL_ID (load_dialog "EX18-3.DCL"))                ; [7]
  (if (not (new_dialog "EX3" DCL_ID)) (exit))             ; [8]
  (start_dialog)                                          ; [9]
  (unload_dialog DCL_ID)                                  ; [10]
  (princ)
); Kết thúc file EX18-3.LSP
```

Tại dòng nhắc lệnh của **AutoCAD** thực hiện lệnh sau:

Command: **(load "EX18-3.LSP")** ↵
C:EX3
Command: **ex3** ↵

Trên màn hình xuất hiện hộp thoại sau đây:



Giải thích

[1] **// Tên file: EX18-3.DCL** Các dòng chú thích bắt đầu bằng 2 dấu gạch chéo //

[2] **EX3 : dialog {**
label = "Dialog Handling Example"; Tên và tiêu đề của hộp thoại. Để ý rằng dòng mô tả thuộc tính kết thúc bằng dấu chấm phẩy.

[3] **: text {**
label = "Dialog session example";
} Dòng mô tả *text tile* và tiêu đề của *text tile*.

[4] **ok_only;** Khai báo *subassembly* chỉ chứa một nút OK.

[5] **;Tên file : EX18-3.LSP**
;Mục đích: Hiển thị hộp thoại EX3 chứa trong file EX18-3.DCL
Các dòng chú thích của chương trình **AutoLISP**.

[6] **(defun C:EX3 (/ DCL_ID)** Định nghĩa hàm EX3 dùng để hiển thị hộp thoại.

[7] **(setq DCL_ID (load_dialog "EX18-3.DCL"))** Tải file EX18-3.DCL vào bộ nhớ, giá trị trả về được lưu trong biến DCL_ID.

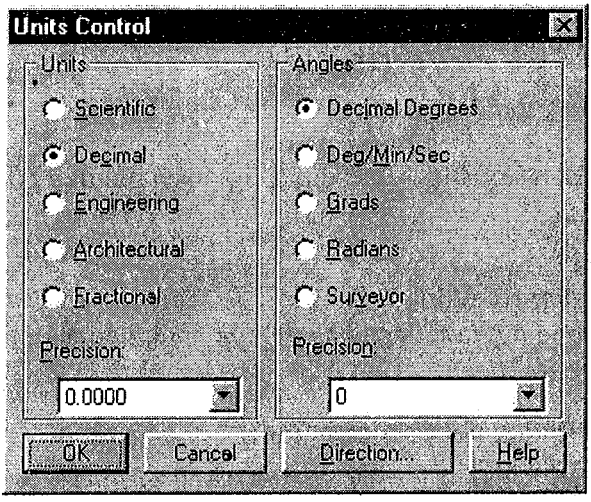
[8] **(if (not (new_dialog "EX3" DCL_ID)) (exit))** Phương pháp thường được sử dụng để bẫy các lỗi xuất hiện khi chạy chương trình. Nếu không khởi tạo được hộp thoại thì kết thúc chương trình.

[9] **(start_dialog)** Hiển thị hộp thoại và bắt đầu làm việc. Chương trình **AutoLISP** dừng lại đợi cho đến khi hộp thoại đóng lại.

[10] **(unload_dialog DCL_ID)** Khi người sử dụng đóng hộp thoại, chương trình tiếp tục. Biểu thức này sẽ loại bỏ các mã lệnh DCL khỏi bộ nhớ.

18.2 Các hàm điều khiển các tile

Ta có thể sử dụng các thuộc tính như *value, is_enabled, action* để gán các giá trị mặc định, trạng thái ban đầu và chức năng của các *tile*. Tuy nhiên, trong một số trường hợp, như trong hộp thoại sau, phương pháp này không sử dụng được.



Mỗi khi làm xuất hiện hộp thoại này, **AutoCAD** sẽ tham khảo giá trị của các biến LUNITS, LUPREC, AUNITS, AUPREC để gán giá trị ban đầu cho các *tile*. Các giá trị ban đầu này thay đổi tùy theo tình trạng của môi trường **AutoCAD**.

Trong các trường hợp này, ta có thể sử dụng các hàm điều khiển các *tile* của **AutoLISP**, sẽ được trình bày sau đây.

Hàm SET_TILE

Quá trình khởi tạo các *tile* bao gồm việc gán giá trị mặc định, gán vị trí con trỏ, xác định trạng thái *enabled* hoặc *disabled* ...

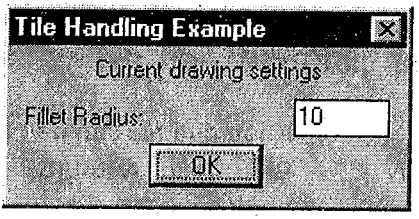
Hàm **Set_tile** dùng để gán giá trị ban đầu cho các *tile* hoặc thay đổi giá trị trong thời gian chạy chương trình.

(**Set_tile** KEY VALUE)

KEY Khóa của *tile* cần gán giá trị (được khai báo trong thuộc tính *key*).
VALUE Giá trị gán cho *tile*.

Ví dụ:

Hộp thoại hiển thị giá trị hiện hành của biến FILLETRAD.



File EX18-4.DCL

```
//Tên file: EX18-4.DCL
//Mục đích: Hiển thị giá trị hiện hành của biến FILLETRAD.
EX4 : dialog {
      label = "Tile Handling Example";
      :text {
        label = "Current drawing settings";
        alignment = centered;
      }
       : edit_box {
        label = "Fillet Radius:";
        edit_width = 6;
        key = "f_rad";
      }
      ok_only;
   }
// Kết thúc file EX18-4.DCL
```

File EX18-4.LSP

```
;Tên file : EX18-4.LSP
;Mục đích: Hiển thị hộp thoại EX4 chứa trong file EX18-4.DCL
(defun C:EX4 (/ DCL_ID)
  (setq DCL_ID (load_dialog "EX18-4.DCL"))
  (if (not (new_dialog "EX4" DCL_ID)) (exit))
  (set_tile "f_rad" (rtos (getvar "FILLETRAD")))
  (start_dialog)
  (unload_dialog DCL_ID)
  (princ)
); Kết thúc file EX18-4.LSP
```

Hàm ACTION_TILE

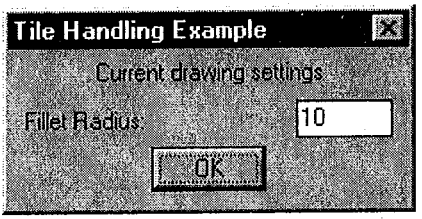
Để gán cho *tile* chức năng thực hiện một công việc, ta sử dụng hàm **Action_tile** để gán cho *tile* một biểu thức **AutoLISP** (tương tự biểu thức chứa trong thuộc tính *action*).

(**Action_tile** KEY ACTION_EXPRESSION)

KEY	Khóa của *tile* cần gán biểu thức.
ACTION_EXPRESSION	Biểu thức **AutoLISP** chứa trong cặp dấu nháy chuỗi " ".

Ví dụ:

Trong hộp thoại của ví dụ EX18-4, ta gán thêm chức năng sau đây cho *edit_box* "f_rad". Khi người sử dụng nhập giá trị mới, giá trị này sẽ được gán cho biến FILLETRAD.



File EX18-4.LSP

```
;Tên file: EX18-4.LSP
;Mục đích: Hiển thị hộp thoại EX4 chứa trong file EX18-4.DCL
(defun C:EX4 (/ DCL_ID)
  (setq DCL_ID (load_dialog "EX18-4.DCL"))
  (if (not (new_dialog "EX4" DCL_ID)) (exit))
  (set_tile "f_rad" (rtos (getvar "FILLETRAD")))
  (action_tile "f_rad"
               "(setvar \"FILLETRAD\" (atof $value))")  ; $value lấy ra giá
                                                        ; trị chứa trong edit_box
  (start_dialog)
  (unload_dialog DCL_ID)
  (princ)
); Kết thúc file EX18-4.LSP
```

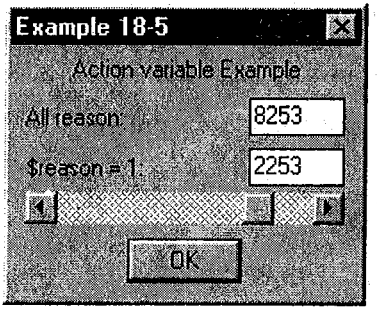
Trong ví dụ trên, có sử dụng biến *$value* (sử dụng trong biểu thức *action*) để lấy ra giá trị hiện hành của *tile*.

Bảng liệt kê các biến sử dụng trong biểu thức *action*

Tên biến	Ý nghĩa
$key	Khóa (*key*) của *tile* đang được chọn hiện hành.
$value	Giá trị của *tile* đang được chọn hiện hành (luôn luôn có kiểu chuỗi).
$data	Sử dụng với hàm client_data_tile.
$reason	Cho biết chức năng *action* của *tile* đã được kích hoạt bằng phương pháp nào (thường chỉ sử dụng cho *edit_box*, *list_box*, *image_button*, *slider*). 1 = Người sử dụng nhấn phím Enter hoặc dùng chuột chọn, 2 = Con trỏ rời khỏi *edit_box*. 3 = Người sử dụng kéo rê con trượt của thanh *slider*. 4 = Khi một mục của *list_box* được chọn và người sử dụng nhấn phím Enter.
$x $y	Tọa độ điểm được chọn trên *image_button*.

Ví dụ:

Trong hộp thoại sau đây, khi di chuyển thanh *slider* (bằng bất kỳ cách nào) thì giá trị của nó sẽ được gán cho *edit_box* 1. Trong khi đó, *edit_box* 2 chỉ được gán giá trị khi ta kéo rê con trượt của thanh *slider* (tương ứng với *$reason* = 1*)*.



File EX18-5.DCL

```
//Tên file: EX18-5.DCL
//Mục đích: Minh họa các biến $key, $value, $reason.
EX5 : dialog {
```

```
        label = "Example 18-5";
        :text {
          label = "Action variable Example";
          alignment = centered;
        }
        : edit_box {
          label = "All reason:";
          edit_width = 6;
          key = "edit_1";
        }
        : edit_box {
          label = "$reason = 1:";
          edit_width = 6;
          key = "edit_2";
        }
        : slider {
        label = "slider:";
        key ="sld_1";
        }
       ok_only;
     }
// Kết thúc file EX18-5.DCL
```

File EX18-5.LSP

```
;Tên file: EX18-5.LSP
;Mục đích: Hiển thị hộp thoại EX5 chứa trong file EX18-5.DCL
(defun C:EX5 (/ DCL_ID)
  (setq DCL_ID (load_dialog "EX18-5.DCL"))
  (if (not (new_dialog "EX5" DCL_ID)) (exit))
  (action_tile "sld_1"
   (set_tile \"edit_1\" $value) (if (= 3 $reason)(set_tile \"edit_2\" $value))"
   )
  (start_dialog)
  (unload_dialog DCL_ID)
  (princ)
); Kết thúc file EX18-5.LSP
```

Các hàm **AutoLISP** không thể sử dụng trong các biểu thức *action* bao gồm các hàm thay đổi màn hình và các hàm yêu cầu người sử dụng nhập dữ liệu:

command	getdist	grclear	prompt
entdel	getint	grdraw	redraw
entmake	getkword	grread	ssget (nếu có yêu cầu người sử dụng chọn đối tượng)
entmode	getorient	grtext	
entsel	getpoint	grvecs	textpage
entupd	getreal	menucmd	textscr
getangle	getstring	nentsel	
getcorner	graphscr	osnap	

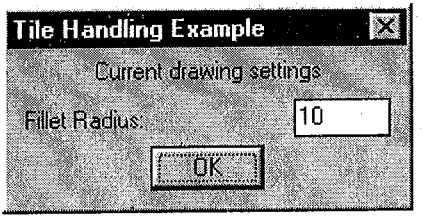
Hàm MODE_TILE

Hàm **Mode_tile** dùng để thay đổi trạng thái của các *tile* trong khi chạy chương trình.

(**Mode_tile** KEY STATE)

KEY Khóa của *tile* cần thay đổi trạng thái
STATE Trạng thái gán cho *tile*.

Ví dụ: Sử dụng hàm **Mode_tile** để di chuyển con trỏ của hộp thoại.
Trong hộp thoại của ví dụ EX18-4, ta sử dụng biến $reason để phát hiện nếu sau khi nhập giá trị mới cho *edit_box*, người sử dụng nhấn Enter, thì con trỏ sẽ tự động chuyển về nút OK.

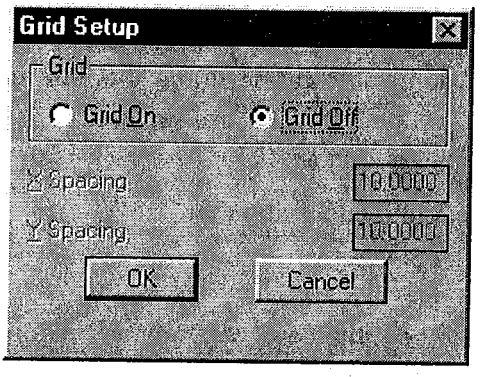



```
;Tên file: EX18-4.LSP
;Mục đích: Hiển thị hộp thoại EX4 chứa trong file EX18-4.DCL
(defun C:EX4 (/ DCL_ID)
  (setq DCL_ID (load_dialog "EX18-4.DCL"))
  (if (not (new_dialog "EX4" DCL_ID)) (exit))
  (set_tile "f_rad" (rtos (getvar "FILLETRAD")))
  (action_tile "f_rad"
     "(setvar \"FILLETRAD\" (atof $value)) (if (= 1 $reason)(mode_tile
\"accept\" 2))")
  (start_dialog)
  (unload_dialog DCL_ID)
```

```
    (princ)
); Kết thúc file EX18-4.LSP
```

Ví dụ:

Sử dụng hàm **Mode_tile** để thay đổi trạng thái *enabled* hoặc *disabled* của các *tile*. Trong hộp thoại sau đây, khi nhấn vào nút *radio* "Grid off" thì các *edit_box* bị mờ đi. Khi nhấn vào nút *radio* "Grid on" thì các *edit_box* xuất hiện lại để nhập giá trị.



File EX18-6.DCL

```
//Tên file: EX18-6.DCL
//Mục đích: Sử dụng hàm Mode_tile để thay đổi trạng thái của các
edit_box.
EX6: dialog {
     label = "Grid Setup";
     :boxed_row {
      label = "Grid";
      : radio_button { label = "Grid &On"; key = "radio_on"; }
      : radio_button { label = "Grid &Off"; key = "radio_off"; }
     }
     : edit_box { label = "&X Spacing"; edit_width = 6; key = "edit_x";}
     : edit_box { label = "&Y Spacing"; edit_width = 6; key = "edit_y";}
     ok_cancel_err;
   }
//Kết thúc file EX18-6.DCL
```

File EX18-6.LSP

```
;Tên file: EX18-6.LSP
;Mục đích: Hiển thị hộp thoại EX6 chứa trong file EX18-6.DCL
```

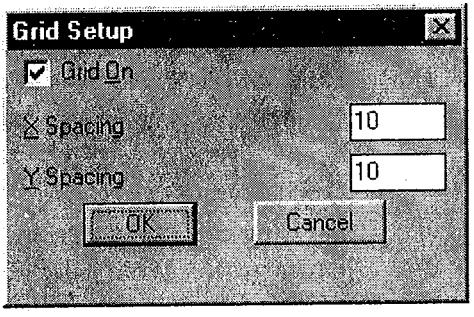
```
(defun C:EX6 (/ DCL_ID)
 (setq DCL_ID (load_dialog "EX18-6.DCL"))
 (if (not (new_dialog "EX6" DCL_ID)) (exit))
 (action_tile "radio_off" "(mode_tile \"edit_x\" 1) (mode_tile \"edit_y\" 1)")
 (action_tile "radio_on" "(mode_tile \"edit_x\" 0) (mode_tile \"edit_y\" 0)")
 (start_dialog)
 (unload_dialog DCL_ID)
 (princ)
); Kết thúc file: EX18-6.LSP
```

Hàm Get_tile

Hàm **Get_tile** dùng để lấy ra giá trị hiện hành của *tile* có khóa chứa trong tham số KEY.

(**Get_tile** KEY)

Ví dụ:



File EX18-7.DCL

```
//Tên file: EX18-7.DCL
//
EX7: dialog {
      label = "Grid Setup";
      : toggle {
        label = "Grid On";
        mnemonic = "O";
        key = "gr_tog";
      }
      : edit_box {
```

```
        label = "X Spacing";
        edit_width = 6;
        key = "gr_x";
        mnemonic = "X";
      }
      : edit_box {
        label = "Y Spacing";
        edit_width = 6;
        key = "gr_y";
        mnemonic = "Y";
      }
      ok_cancel_err;
    }
//Kết thúc file EX18-7.DCL
```

File EX18-7.LSP

```
;Tên file: EX18-7.LSP
;
(defun GRID_TOG (val)          ;Hàm này được gọi bởi toggle "Grid On"
  (if (zerop (atoi VAL))       ;Khi toggle bằng Off, thì VAL bằng 0
    (progn
      (mode_tile "gr_x" 1)     ;Làm mờ edit_box X_Spacing
      (mode_tile "gr_y" 1)     ;Làm mờ edit_box Y_Spacing
      (mode_tile "accept" 2)   ;Chuyển con trỏ cho nút OK
    )
    (progn                     ;Toggle bằng On
      (mode_tile "gr_x" 0)     ;Bật sáng edit_box X_Spacing
      (mode_tile "gr_y" 0)     ;Bật sáng edit_box Y_Spacing
      (mode_tile "gr_x" 2)     ;Chuyển con trỏ cho edit_box X_Spacing
      (mode_tile "gr_x" 3)     ;Chiếu sáng toàn bộ nội dung của edit_box
                               ;X_Spacing
    )
  )
)                              ;Đóng defun GRID_TOG

(defun SET_GRX (why)           ;Hàm này được gọi bởi edit_box X_Spacing
                               ;Biến $reason được truyền cho tham số WHY
  (if (= WHY 1)                ;Nếu người sử dụng nhấn Enter ($reason =1)
    (progn                     ;thì
      (set_tile "gr_x" (rtos (atof (get_tile "gr_x"))))  ;gán giá trị nhập cho tile
      (mode_tile "gr_y" 2)     ;Chuyển con trỏ sang edit_box Y_Spacing
```

```
      (mode_tile "gr_y" 3)      ;Chiếu sáng nội dung của edit_box
                                ;Y_Spacing
    )
  )
)

(defun SET_GRY (why)            ;Hàm này được gọi bởi edit_box Y_Spacing
                                ;Biến $reason được truyền cho tham số WHY

  (if (= WHY 1)                 ;Nếu người sử dụng nhấn Enter ($reason =1)
    (progn                      ; thì . . .
      (set_tile "gr_y" (rtos (atof (get_tile "gr_y")))) ;gán giá trị nhập cho tile
      (mode_tile "accept" 2)    ; Chuyển con trỏ sang nút OK
    )
  )
)

(defun CHECKOUT ()              ;Hàm này được gọi bằng nút OK. Hàm này
                                ;lấy giá trị của các tile và gán cho các biến hệ
                                ;thống tương ứng.
  (if (zerop (atoi (get_tile "gr_tog")))   ;Kiểm tra giá trị của toggle
    (setvar "GRIDMODE" 0)                  ;Nếu là Off, thì không hiển thị lưới
    (progn
     (setvar "GRIDMODE" 1)                 ;Nếu là On, thì hiển thị lưới
     (setvar "GRIDUNIT" (list (atof (get_tile "gr_x")) (atof (get_tile "gr_y"))))
    )
  )
)

(defun C:EX7 ()                         ;Hàm gọi xuất hiện hộp thoại.
  (setq DCL_ID (load_dialog "EX18-7.DCL"))
  (if (not (new_dialog "EX7" DCL_ID)) (exit))
  (set_tile "gr_tog" (rtos (getvar "GRIDMODE"))); Giá trị của toggle phụ
                                ;thuộc vào tình trạng hiện hành của lưới.
  (set_tile "gr_x" (rtos (car (getvar "GRIDUNIT")))) ;Giá trị ban đầu của các
  (set_tile "gr_y" (rtos (cadr (getvar "GRIDUNIT")))) ;edit_box phụ thuộc vào
                                                ;giá trị hiện hành của
                                                ;các biến hệ thống tương ứng.

  (if (zerop (getvar "GRIDMODE"))   ;Nếu lưới đang bị tắt thì . . .
```

```
    (progn
      (mode_tile "gr_x" 1)          ;làm mờ edit_box X_Spacing
      (mode_tile "gr_y" 1)          ;làm mờ edit_box Y_Spacing
    )
  )
  (action_tile "gr_tog" "(GRID_TOG $value)") ;Gán action cho toggle
  (action_tile "gr_x" "(SET_GRX $reason)")    ;Gán action cho "X_Spacing"
  (action_tile "gr_y" "(SET_GRY $reason)")    ;Gán action cho "Y_Spacing"
  (action_tile "accept" "(CHECKOUT) (done_dialog)") ;Gán action cho "OK"
  (start_dialog)
  (unload_dialog DCL_ID)
  (command "redraw")
  (princ)
)
;Kết thúc file EX18-7.LSP
```

Kiểm tra giá trị nhập và thông báo lỗi

Hộp thoại ở trên không hiển thị thông báo lỗi khi giá trị nhập ở các *edit_box* không phải là các con số. Để hiển thị thông báo lỗi ở *errtile* (có khóa là "error"), ta bổ sung thêm hàm VALIDATE để kiểm tra giá trị nhập, và sửa lại các biểu thức action của các *edit_box* để gọi đến hàm VALIDATE này.

```
;Tên file: EX18-7.LSP

(defun GRID_TOG (val)          ;Hàm này được gọi bởi toggle "Grid On"
  (if (zerop (atoi VAL))       ;Khi toggle bằng Off, thì VAL bằng 0
    (progn
      (mode_tile "gr_x" 1)     ;Làm mờ edit_box X_Spacing
      (mode_tile "gr_y" 1)     ;Làm mờ edit_box Y_Spacing
      (mode_tile "accept" 2)   ;Chuyển con trỏ cho nút OK
    )
    (progn                     ; Toggle bằng On
      (mode_tile "gr_x" 0)     ; Bật sáng edit_box X_Spacing
      (mode_tile "gr_y" 0)     ; Bật sáng edit_box Y_Spacing
      (mode_tile "gr_x" 2)     ;Chuyển con trỏ cho edit_box X_Spacing
      (mode_tile "gr_x" 3)     ;Chiếu sáng toàn bộ nội dung của edit_box
                               ;X_Spacing
    )
  )
```

```
)                                   ; Đóng defun GRID_TOG
(defun SET_GRX (why)                ;Hàm này được gọi bởi edit_box X_Spacing
                                    ;Biến $reason được truyền cho tham số WHY
  (if (= WHY 1)                     ;Nếu người sử dụng nhấn Enter ($reason =1)
    (progn                          ; thì
      (set_tile "gr_x" (rtos (atof (get_tile "gr_x"))))  ;gán giá trị nhập cho tile
      (mode_tile "gr_y" 2)          ; Chuyển con trỏ sang edit_box Y_Spacing
      (mode_tile "gr_y" 3)          ; Chiếu sáng nội dung của edit_box
                                    ;Y_Spacing
    )
  )
)

(defun SET_GRY (why)                ;Hàm này được gọi bởi edit_box Y_Spacing
                                    ;Biến $reason được truyền cho tham số WHY

  (if (= WHY 1)                     ;Nếu người sử dụng nhấn Enter ($reason =1)
    (progn                          ; thì . . .
      (set_tile "gr_y" (rtos (atof (get_tile "gr_y"))))  ;gán giá trị nhập cho tile
      (mode_tile "accept" 2)        ; Chuyển con trỏ sang nút OK
    )
  )
)

;Hàm kiểm tra giá trị nhập
;
(defun VALIDATE (val why key)
  (if (= "." (substr VAL 1 1)) (setq VAL (strcat "0" VAL))) ;Nếu giá trị nhập
                                    ;bắt đầu bằng dấu chấm (.) thì thêm
                                    ;số 0 vào đằng trước
  (if (not                          ; Kiểm tra giá trị nhập có phải là
      (or                           ; số thực hoặc số nguyên hay không
        (= (type (read VAL)) 'REAL)
        (= (type (read VAL)) 'INT)
      )
    )
    (progn                          ;Nếu không thì
      (mode_tile KEY 3)             ;chuyển con trỏ về lại và
      (set_tile "error" "Invalid entry"); hiện thông báo lỗi.
    )
```

```
      (progn                          ;Nếu phải (giá trị nhập hợp lệ) thì
        (set_tile "error" "")         ; xóa thông báo lỗi
        (if (= KEY "gr_x")            ;và gọi đến các hàm tương ứng.
          (SET_GRX WHY)
          (SET_GRY WHY)
        )
      )
    )
)                                     ;Đóng hàm defun VALIDATE

(defun CHECKOUT () ;Hàm này được gọi bằng nút OK. Hàm này
                   ;lấy giá trị của các tile và gán cho các biến hệ
                   ;thống tương ứng.
  (if (zerop (atoi (get_tile "gr_tog")))   ;Kiểm tra giá trị của toggle
    (setvar "GRIDMODE" 0)                  ;Nếu là Off, thì không hiển thị lưới
    (progn
     (setvar "GRIDMODE" 1)                 ;Nếu là On, thì hiển thị lưới
     (setvar "GRIDUNIT" (list (atof (get_tile "gr_x")) (atof (get_tile "gr_y"))))
    )
  )
)

(defun C:EX7 ()                           ;Hàm gọi xuất hiện hộp thoại.
  (setq DCL_ID (load_dialog "EX18-7.DCL"))
  (if (not (new_dialog "EX7" DCL_ID)) (exit))
  (set_tile "gr_tog" (rtos (getvar "GRIDMODE"))); Giá trị của toggle phụ
                              ;thuộc vào tình trạng hiện hành của lưới.
  (set_tile "gr_x" (rtos (car (getvar "GRIDUNIT")))) ;Giá trị ban đầu của các
  (set_tile "gr_y" (rtos (cadr (getvar "GRIDUNIT")))) ;edit_box phụ thuộc vào
                                                      ;giá trị hiện hành của
                                                      ;các biến hệ thống tương ứng.

  (if (zerop (getvar "GRIDMODE"))         ;Nếu lưới đang bị tắt thì . . .
    (progn
      (mode_tile "gr_x" 1)                ;làm mờ edit_box X_Spacing
      (mode_tile "gr_y" 1)                ;làm mờ edit_box Y_Spacing
    )
  )
  (action_tile "gr_tog" "(GRID_TOG $value)")
  (action_tile "gr_x" "(VALIDATE (get_tile \"gr_x\") $reason \"gr_x\")")
  (action_tile "gr_y" "(VALIDATE (get_tile \"gr_y\") $reason \"gr_y\")")
```

```
  (action_tile "accept" "(CHECKOUT) (done_dialog)")
  (start_dialog)
  (unload_dialog DCL_ID)
  (command "redraw")
  (princ)
)
;Kết thúc file  EX18-7.LSP
```

Tóm tắt

1. Hàm **Set_tile** dùng để gán giá trị ban đầu cho các *tile* hoặc thay đổi giá trị trong thời gian chạy chương trình.
2. Để gán cho *tile* chức năng thực hiện một công việc, ta sử dụng hàm **Action_tile** để gán cho *tile* một biểu thức **AutoLISP**.
3. Thuộc tính *key* giúp cho các *tile* có thể truy xuất được bằng các hàm **AutoLISP**. Thuộc tính *key* phân biệt dạng chữ hoa chữ thường.
4. Các hàm thay đổi màn hình và các hàm yêu cầu người sử dụng nhập dữ liệu không thể sử dụng được trong các biểu thức *action.*
5. Hàm **Mode_tile** được dùng để thay đổi trạng thái của các *tile.*
6. Các biến liên quan đến hộp thoại trong thời gian chạy chương trình là: *$key, $value, $data, $reason, $x, $y.*
7. Cần phải kiểm tra dữ liệu nhập vào cho hộp thoại để chương trình không gây ra lỗi.

18.3 List box và popup list

Có ba hàm **AutoLISP** liên quan đến List box và Popup list là: **Start_list, Add_list** và **End_list.**

Hàm START_LIST

Hàm **Start_list** dùng để khởi tạo một danh sách hoặc mở danh sách có sẵn để thêm hoặc thay đổi các mục chọn cho *list_box* hoặc *popup_list.*

(**Start_list** KEY [OPERATION [INDEX]])

KEY Khóa của *list_box* hoặc *popup_list*.

OPERATION Cho biết cách thức tạo danh sách. Các giá trị như sau:

1: Thay đổi nội dung các mục chọn có sẵn.
2: Thêm các mục chọn mới.
3: (mặc định) Xóa danh sách cũ, tạo ra danh sách mới.

INDEX Số thứ tự của mục chọn được thay đổi nội dung. Các mục chọn được đánh số bắt đầu từ 0. Chỉ sử dụng tham số INDEX khi tham số OPERATION = *1*. Nếu không xác định tham số INDEX, thì giá trị mặc định là 0.

Ví dụ:

(**Start_list** "lst_1")	Tạo danh sách mới (mặc định OPERATION = 3)
(**Start_list** "lst_1" 1 2)	Mở danh sách "lst_1" để thay đổi nội dung mục chọn thứ 3 (có số thứ tự là 2)

Hàm ADD_LIST

Hàm **Add_list** dùng để thay đổi hoặc thêm mục chọn mới vào danh sách tùy theo cách mở danh sách bằng hàm **Start_list**.

(**Add_list** ITEM)

ITEM Nội dung mục chọn mới.

Hàm END_LIST

Hàm **End_list** dùng để kết thúc quá trình tạo hoặc chỉnh sửa danh sách.

(**end_list**)

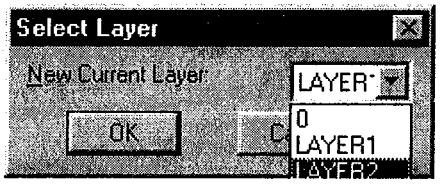
Ví dụ:

(**start_list** "lst_1" 1 2) (**add_list** "Layer 0") (**end_list**)	Sửa nội dung mục chọn thứ 3 thành "Layer 0".

(**start_list** "lst_1" 2) (**add_list** "Layer 5") (**add_list** "Layer 6") (**end_list**)	Thêm 2 mục chọn mới vào phía cuối của danh sách
(**start_list** "lst_2" 2) (**mapcar** '**add_list** (list "A" "B" "C" "D")) (**end_list**)	Thêm 4 mục chọn vào danh sách "lst_2". Sử dụng hàm **Mapcar** để đơn giản các dòng lệnh.

Ví dụ:

Chương trình sau đây hiển thị hộp thoại cho phép người sử dụng chọn một lớp bản vẽ trong *popup_list*, sử dụng làm lớp bản vẽ hiện hành.



File EX18-8.DCL

```
//Tên file: EX18-8.DCL
//Mục đích: Hộp thoại cung cấp danh sách các layer.
//
EX8 : dialog {
    label = "Select Layer";
    : popup_list {
      label = "New Current Layer:";
      mnemonic = "N";
      key = "lay_pop";
      allow_accept = true;
      width = 32;
    }
    ok_cancel;
}
//Kết thúc file: EX18-8.DCL
```

File EX18-8.LSP

```
;Tên file : EX18-8.LSP
;Mục đích: Hiển thị hộp thoại EX8 chứa trong file EX18-8.DCL, cho phép
;          người sử dụng chọn lớp bản vẽ hiện hành.
;
;Định nghĩa hàm CHECKOUT được gọi bởi nút OK
(defun CHECKOUT ()
  (setq LD (tblsearch "LAYER" (nth (atoi (get_tile "lay_pop")) LL)) ; Biến LD
                                    ; chứa record dữ liệu của lớp bản vẽ
                                    ; được chọn trong popup_list.
        LN (cdr (assoc 2 LD))       ; Biến LN chứa tên lớp bản vẽ
        LS (cdr (assoc 70 LD))      ; Biến LS chứa trạng thái lớp bản vẽ
  )                                 ; Đóng setq
  (if (and                          ; Nếu LS khác 1 và khác 65, nghĩa là
        (/= 1 LS)                   ; nếu lớp bản vẽ không bị đóng băng
        (/= 65 LS)
      )
      (prog             ; Gán lớp bản vẽ hiện hành và đóng hộp thoại.
        (setvar "CLAYER" (nth (atoi (get_tile "lay_pop")) LL))
        (done_dialog)
      )
      (alert "Selected layer is frozen!")  ; ngược lại, thì thông báo và quay về
  )                                 ; lại hộp thoại.
)                                   ;Đóng defun CHECKOUT
;
; Định nghĩa hàm EX8 sử dụng để hiển thị hộp thoại.
;
  (defun C:EX8 ()
    (setq DCL_ID (load_dialog "EX18-8.DCL"))    ;Tải file DCL vào bộ nhớ
    (if (not (new_dialog "EX8" DCL_ID)) (exit))
    (start_list "lay_pop")          ; Bắt đầu tạo các mục chọn cho "lay_pop"
    (setq LL '()                    ; Khởi tạo danh sách chứa tên các lớp bản vẽ.
          NL (tblnext "LAYER" T)
          IDX 0
    )
    (while NL                       ; Duyệt các lớp bản vẽ trong bảng "LAYER"
      (if (= (getvar "CLAYER") (cdr (assoc 2 NL))); Tại mỗi bước, NL được
                                    ; kiểm tra có phải là lớp bản vẽ hiện hành
                                    ;hay không.
          (setq CL IDX)             ; Nếu phải, CL chứa vị trí này
          (setq IDX (1+ IDX))       ; Nếu không, tăng IDX
```

```
    )                                   ; Đóng if
    (setq LL (append LL (list (cdr (assoc 2 NL))))   ; Thêm tên lớp bản vẽ
                                                     ;mới vào LL
        NL (tblnext "LAYER")            ;Chuyển lớp·bản vẽ kế tiếp
    )
)                                       ; Đóng while
(mapcar 'add_list LL)       ; Thêm các lớp bản vẽ trong LL vào popup_list
(end_list)                  ; Đóng popup_list
(set_tile "lay_pop" (itoa CL))          ; Gán giá trị hiện hành của popup_list
                                        ;bằng tên lớp bản vẽ hiện hành
(action_tile "lay_pop" "(if (= $reason 4) (mode_tile \"accept\" 2))")
(action_tile "accept" "(CHECKOUT)")
(start_dialog)
(unload_dialog DCL_ID)
(princ)
); Kết thúc file EX18-8.LSP
```

18.4 Image và Image Button

Image và *Image Button* sử dụng để biểu diễn hình ảnh trên hộp thoại. *Image Button* còn có chức năng tương tự như một *button.*

Các hình ảnh có thể được tạo ra từ các *slide file* (.sld) của **AutoCAD** hoặc được vẽ trực tiếp bằng các véc tơ.

Các hàm **AutoLISP** liên quan đến *Image* hoặc *Image Button* là **Start_image** và **Fill_image.**

Hàm START_IMAGE

Việc tạo và quản lý các *image tile* phải tiến hành bằng nhiều bước. Bước đầu tiên là phải "mở" *image tile* (tương tự các *list_box*) bằng hàm **Start_image**.

(**Start_image** KEY)

KEY Khóa của *image tile* được tạo hình ảnh.

Khi mới khởi tạo, *image tile* chưa có hình ảnh và màu nền của nó phụ thuộc vào màu nền hiện hành của **AutoCAD**.

Hàm END_IMAGE

Hàm **End_image** kết thúc quá trình khởi tạo và điền hình ảnh cho *image tile.*

(**End_image**)

Hàm FILL_IMAGE

Hàm **Fill_image** dùng để tạo màu nền cho *image tile.* Hàm này sẽ vẽ một hình chữ nhật trong phạm vi của *image tile* và tô màu cho hình chữ nhật này.

(**Fill_image** X Y WID HGT COLOR)

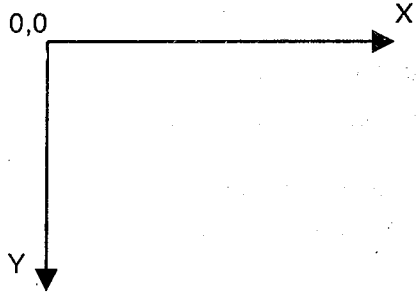
X, Y Tọa độ đỉnh thứ nhất hình chữ nhật.
WID Chiều rộng hình chữ nhật.
HGT Chiều cao hình chữ nhật.
COLOR Màu nền hình chữ nhật. Ngoài các mã màu của **AutoCAD**, ta còn có thể sử dụng các mã màu sau:

Mã màu	Ý nghĩa
-2	Màu nền màn hình đồ họa của **AutoCAD.**
-15	Màu nền hộp thoại.
-16	Màu chữ xuất hiện trên hộp thoại.
-18	Màu đường nét vẽ trên hộp thoại.

Chú ý:

Hệ trục tọa độ trong trường hợp này được xác định như sau:

- Gốc tọa độ (0,0): Góc **trái trên** của *image tile.*
- Trục X: Hướng từ trái sang phải.
- Trục Y: Hướng từ trên xuống dưới.



Chiều rộng và chiều cao hình chữ nhật phụ thuộc vào kích thước của *image tile*. Chiều rộng và chiều cao của *image tile* có thể lấy ra bằng các hàm **Dimx_tile** và **DimY_tile**.

(**DimX_tile** KEY)
(**DimY_tile** KEY)

Ví dụ:

Đoạn chương trình sau tô màu đỏ cho *image tile* có khóa là "img_1"

```
(start_image "img_1")
(setq WID (dimx_tile "img_1")
      HGT (dimy_tile "img_1")
      CLR 1
)
(fill_image 0 0 WID HGT CLR)
(end_image)
```

Hàm VECTOR_IMAGE

Đối với các hình ảnh đơn giản, ta có thể sử dụng hàm **Vector_image** để vẽ trực tiếp các véc tơ lên *image tile*.

(**Vector_image** X1 Y1 X2 Y2 COLOR)

X1, Y1	Tọa độ điểm đầu của véc tơ.
X2, Y2	Tọa độ điểm cuối của véc tơ.
COLOR	Màu của véc tơ.

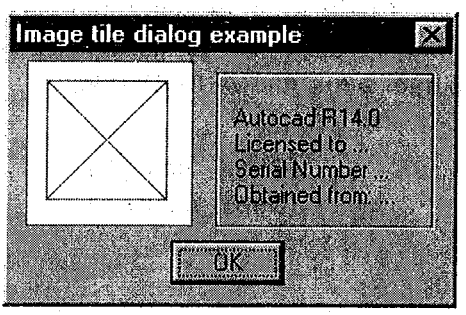
Ví dụ:

Hộp thoại sau đây có một *image tile* chứa hình vẽ tượng trưng cho logo của công ty. Hình được vẽ trực tiếp bằng các véc tơ. Các điểm của véc tơ được biểu diễn bằng *tọa độ tuyệt đối* trong hệ trục tọa độ của *image tile*.

File EX18-9.DCL

```
//Tên file: EX18-9.DCL
```

```
//
 EX9: dialog {
    label = "Image tile dialog
                example";
    : row {
      : image_button {
        key = "logo";
        width = 14;
        fixed_width = true;
        fixed_height = true;
        aspect_ratio = 1;
      }
      : boxed_column {
        width = 16;
         : paragraph {
           : text_part {value = "Autocad R14.0";}
           : text_part {value = "Licensed to ...";}
           : text_part {value = "Serial Number ...";}
           : text_part {value = "Obtained from: ...";}
         }
       }
    }
    ok_only;
}
//Kết thúc file EX18-9.DCL
```



File EX18-9.LSP

```
;Tên file: EX18-9.LSP
;
(defun C:EX9 ()
  (setq DCL_ID (load_dialog "EX18-9.DCL"))
  (new_dialog "EX9" DCL_ID)
  (start_image "logo")                          ; Khởi tạo hình ảnh
  (fill_image 0 0 (dimx_tile "logo") (dimy_tile "logo") 7)      ;Tô màu nền
  (mapcar 'vector_image                         ;Vẽ 5 véc tơ tạo hình biểu
        (list 10 10 70 70 10 70)                ; diễn logo
        (list 10 10 10 70 10 10)                ; Hàm Mapcar thực hiện lệnh
        (list 10 70 70 10 70 10)                ; vector_image 5 lần
        (list 70 10 70 70 70 70)
        (list 1 1 1 1 1 1)
```

```
  )
  (end_image)                                  ; Kết thúc quá trình tạo ảnh
  (start_dialog)
  (unload_dialog DCL_ID)
)
; Kết thúc file EX18-9.LSP
```

Tuy nhiên, vì mỗi hệ điều hành (Windows, DOS, Windows NT ....) có cách tính tọa độ khác nhau, nên hình ảnh có thể không được biểu diễn chính xác trên các hệ điều hành khác nhau. Trong trường hợp này, các tọa độ điểm nên được tính theo *tỉ lệ phần trăm* so với kích thước của *image tile*. Trong đó, kích thước *image tile* có thể lấy ra bằng các hàm **Dimx_tile** và **DimY_tile**

Ví dụ:

Vẽ hình logo trong hộp thoại EX18-9 ở ví dụ trên bằng cách tính tọa độ điểm theo kích thước *image tile*.

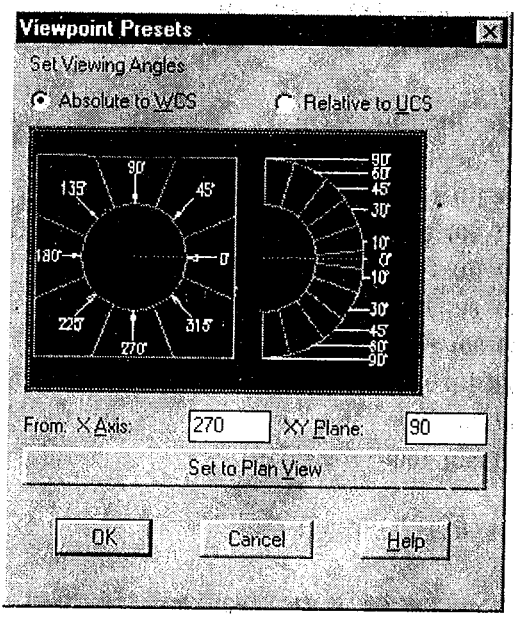
```
;Tên file: EX18-9-1.LSP
;
(defun C:EX9 ()
  (setq DCL_ID (load_dialog "EX18-9.DCL"))
  (new_dialog "EX9" DCL_ID)
  (start_image "logo")
  (setq WID (dimx_tile "logo")
        HGT (dimy_tile "logo")
        P1_X (/ (* WID 10) 100)
        P1_Y (/ (* HGT 10) 100)
        P2_X (/ (* WID 10) 100)
        P2_Y (/ (* HGT 80) 100)
        P3_X (/ (* WID 80) 100)
        P3_Y (/ (* HGT 10) 100)
        P4_X (/ (* WID 80) 100)
        P4_Y (/ (* HGT 80) 100)
  )
  (fill_image 0 0 WID HGT 7)
 (mapcar 'vector_image
        (list P1_X P1_X P3_X P4_X P1_X P2_X)
        (list P1_Y P1_Y P3_Y P4_Y P1_Y P2_Y)
        (list P2_X P3_X P4_X P2_X P4_X P3_X)
```

```
        (list P2_Y P3_Y P4_Y P2_Y P4_Y P3_Y)
        (list 1 1 1 1 1 1)
    )
    (end_image)
    (start_dialog)
    (unload_dialog DCL_ID)
); Kết thúc file EX18-9-1.LSP
```

Khi hình ảnh được tạo bằng *image_button*, ta có thể biết được tọa độ điểm chọn trên hình ảnh nhờ các biến $X và $Y. Thông qua đó, chương trình có những đáp ứng thích hợp.

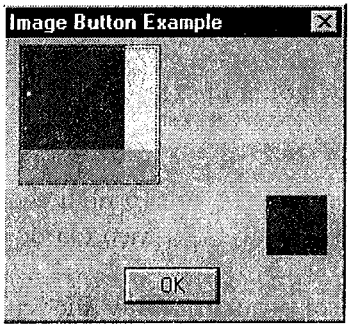
Ví dụ:

Hộp thoại DDVPOINT của **AutoCAD**. Trên hộp thoại có một *image_button* biểu diễn vị trí của điểm nhìn hiện hành gồm 2 hình: hình bên trái biểu diễn góc tạo bởi điểm nhìn và trục X, hình bên phải biểu diễn góc tạo bởi điểm nhìn và mặt phẳng XY. Khi nhấn chuột trên các hình này, điểm chọn trên hình vẽ sẽ được xác định tọa độ và vị trí điểm nhìn sẽ được thay đổi phù hợp.



Ví dụ:

Hộp thoại sau đây minh họa cách tạo bảng màu của **AutoCAD**. Khi chọn một điểm trên *image_button* "boxes", màu sắc tại điểm chọn này sẽ được nhận biết và gán cho *image* "col_swatch".



File EX18-10.DCL

```
//Tên file: EX18-10.DCL
//
EX10: dialog {
    label = "Image Button Example";
    : row {
      : image_button {
        key = "boxes";
        width =14;
        fixed_width = true;
        fixed_height = true;
        aspect_ratio = 1;
      }
    }
    : row {
      spacer_1;
      : image {
        key = "col_swatch";
        width = 6;
        fixed_width = true;
        fixed_height = true;
        aspect_ratio = 1;
      }
```

```
    }
    ok_only;
}
//Kết thúc file EX18-10.DCL
```

File EX18-10.LSP

```
;Tên file: EX18-10.LSP
;
;Định nghĩa hàm PICK_COL được gọi bởi image_button "boxes"
(defun PICK_COL (X Y)
  (start_image "col_swatch")
  (cond                                    ;Tô màu cho "col_swatch" tùy
                                           ;theo tọa độ điểm chọn X, Y.
    ( (and (<= X 63) (<= Y 63))
      (fill_image 0 0 (dimx_tile "col_swatch") (dimy_tile "col_swatch") 1)
    )
    ( (and (> X 63) (<= Y 63))
      (fill_image 0 0 (dimx_tile "col_swatch") (dimy_tile "col_swatch") 2)
    )
    ( (and (<= X 63) (> Y 63))
      (fill_image 0 0 (dimx_tile "col_swatch") (dimy_tile "col_swatch") 3)
    )
    ( (and (> X 63) (> Y 63))
      (fill_image 0 0 (dimx_tile "col_swatch") (dimy_tile "col_swatch") 4)
    )
  )                                        ; Đóng Cond
  (end_image)
)                                          ;Đóng defun PICK_COL
;
;Định nghĩa hàm EX10 dùng để hiển thị hộp thoại.
(defun C:EX10 ()
  (setq DCL_ID (load_dialog "EX18-10.DCL"))
  (new_dialog "EX10" DCL_ID)
  (start_image "boxes")
  (fill_image 0 0 (dimx_tile "boxes") (dimy_tile "boxes") 7)      ;Tô màu nền
  (mapcar 'fill_image                      ;Tô 4 ô màu
          (list 0 63 0 63)
          (list 0 0 63 63)
          (list 63 62 63 62)
          (list 63 63 62 62)
```

```
        (list 1 2 3 4)
  )
  (end_image)
  (start_image "col_swatch")
  (fill_image 0 0 (dimx_tile "col_swatch") (dimy_tile "col_swatch") -15)
  (end_image)
  (action_tile "boxes" "(PICK_COL $x $y)")   ;Truyền tọa độ điểm chọn cho
                                             ;hàm PICK_COL để tô màu
                                             ;cho "col_swatch"
  (start_dialog)
)
;Kết thúc file EX18-10.LSP
```

Hàm SLIDE_IMAGE

Các véc tơ và các hình chữ nhật được tô màu chỉ sử dụng cho các hình ảnh đơn giản. Đối với các hình ảnh phức tạp, trước tiên ta sử dụng các lệnh vẽ của **AutoCAD** để vẽ hình, sau đó tạo thành *slide file* (bằng lệnh **Mslide**), rồi sử dụng hàm **Slide_image** để hiện *slide file* này trên các *image tile*.

(**Slide_image** X1 Y1 WID HGT SLDNAME)

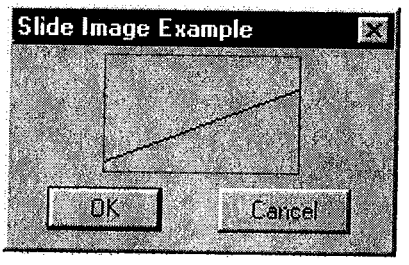
X1, Y1	Tọa độ góc trái trên của vùng xuất hiện hình ảnh.
WID, HGT	Chiều rộng, chiều cao của vùng xuất hiện hình ảnh.
SLDNAME	Tên của *slide file*.

Chú ý:

- Trên cùng một *image tile* có thể hiển thị nhiều hình *slide*. Tại các phần trùng nhau, *slide* ở bên trên không che khuất được *slide* ở bên dưới. Nếu cần, ta phải sử dụng hàm **Fill_image** để che khuất *slide* bên dưới rồi mới cho xuất hiện *slide* bên trên.
- Để có được hình ảnh đẹp, khi tạo *slide file* ta nên thay đổi kích thước màn hình đồ họa của **AutoCAD** theo tỉ lệ phù hợp với kích thước của *image tile*. Sau đó thực hiện lệnh **Zoom** sao cho hình vẽ phóng to hết màn hình, rồi tạo *slide file*.
- Khi cần có thể sử dụng lệnh **Shade** để tô bóng, sau đó tạo *slide file*. Hình ảnh sẽ đẹp hơn.

Ví dụ:

Hộp thoại cho phép lựa chọn vẽ đường thẳng, cung tròn hoặc đường tròn. Nhấn chuột trên *image_button*, hình ảnh sẽ lần lượt thay đổi là đường thẳng, cung tròn hoặc đường tròn. Sau đó nhấn OK để thực hiện lệnh vẽ tương ứng. Ghi nhớ: Trước khi hiện *slide* mới, ta sử dụng lệnh **Fill_image** để che *slide* cũ.



File EX18-11.DCL

```
//Tên file: EX18-11.DCL
//
EX11 : dialog {
      label = "Slide Image Example";
      : row {
        spacer_0;
        : image_button {
          color = dialog_background;
          width = 10;
          aspect_ratio = 1;
          key = "img_1";
          alignment = centered;
        }
        spacer_0;
      }
      ok_cancel;
}
//Kết thúc file EX18-11.DCL
```

File EX18-11.LSP

```
;Tên file : EX18-11.LSP
;Định nghĩa hàm NXTIMG dùng để chuyển đổi slide hiện trên image_button.
(defun NXTIMG ()
```

```
  (if (= 2 IMAGE_COUNTER)          ;Nếu là hình cuối cùng
    (setq IMAGE_COUNTER 0)         ; thì chuyển về hình đầu tiên
    (setq IMAGE_COUNTER
          (1+ IMAGE_COUNTER))      ; nếu không thì
                                   ;chuyển qua hình kế tiếp
  )                                ; Đóng If
  (start_image "img_1")
  (fill_image 0 0 (dimx_tile "img_1")   ; Che slide đang có bằng
            (dimy_tile "img_1") -15)    ; hàm fill_image
  (slide_image 0 0 (dimx_tile "img_1") (dimy_tile "img_1")
     (nth IMAGE_COUNTER IMAGE_LIST)  ;Hiện slide tương ứng
  )
  (end_image)
  (setq OPTION (nth IMAGE_COUNTER IMAGE_LIST)); Cập nhật biến
                                   ; OPTION chứa tên slide hiện hành
)                                  ; Đóng defun NXTIMG
;
; Định nghĩa hàm EX11 để hiển thị hộp thoại
(defun C:EX11 ()
  (setq IMAGE_COUNTER 0                    ; Khởi tạo các biến
      IMAGE_LIST (list "LINE" "ARC" "CIRCLE")
      IMAGE_CURRENT (nth IMAGE_COUNTER IMAGE_LIST)
      OPTION (car IMAGE_LIST)
  )                                        ; Đóng setq
  (setq DCL_ID (load_dialog "EX18-11.DCL"))
  (new_dialog "EX11" DCL_ID)
  (start_image "img_1")
  (slide_image 0 0 (dimx_tile "img_1")     ; Hiện slide file "LINE"
           (dimy_tile "img_1") "LINE")
  (end_image)
  (action_tile "img_1" "(NXTIMG)")         ; Gán biểu thức action cho "img_1"
  (start_dialog)
  (cond                                    ; Sau khi đóng hộp thoại, phụ thuộc
   ((= "LINE" OPTION)                      ;vào biến OPTION mà thực hiện lệnh
    (command ".LINE")                      ; vẽ thích hợp (LINE, ARC hoặc
   )                                       ; CIRCLE)
   ((= "ARC" OPTION)
    (command ".ARC")
   )
   ((= "CIRCLE" OPTION)
    (command ".CIRCLE")
```

```
    )
  )                                    ;Đóng cond
)
;Kết thúc file: EX18-11.LSP
```

Tóm tắt

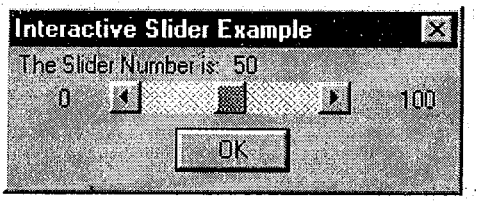
1. Các hàm **AutoLISP** thường dùng để điều khiển các *list_box* và *popup_list* là **Start_list**, **Add_list** và **End_list**.
2. Hình ảnh trong các hộp thoại tăng cường sự tương tác giữa chương trình và người sử dụng.
3. Hàm **Start_image** bắt đầu quá trình tạo hình ảnh cho các *image tile*.
4. Hàm **Fill_image** dùng để gán màu nền cho toàn bộ hoặc một phần của *image tile*.
5. Hàm **Dimx_tile** và **Dimy_tile** cho biết chiều rộng và chiều cao của *image tile*.
6. Hàm **Slide_image** thường được sử dụng để hiện các *slide file* của **AutoCAD** trên hộp thoại. Với các hình đơn giản tạo thành từ những đường thẳng, có thể sử dụng hàm **Vector_image** để vẽ trực tiếp.

18.5 Thanh Slider

Các thanh *slider* giúp cho người sử dụng nhập các giá trị kiểu số cho chương trình một cách dễ dàng và hiệu quả. Ta có thể gán cho *slider* một biểu thức *action* tương tự như đối với các *tile* khác. Thông thường, biểu thức này sử dụng để gán giá trị của *slider* cho một *tile* khác trên hộp thoại.

Ví dụ:

Sử dụng thanh *slider* trong hộp thoại sau, người sử dụng có thể nhập các giá trị từ 0 đến 100. Giá trị này được hiện ra trên hộp thoại và được lưu trữ trong biến VALUE sau khi hộp thoại được đóng lại.



File EX18-12.DCL

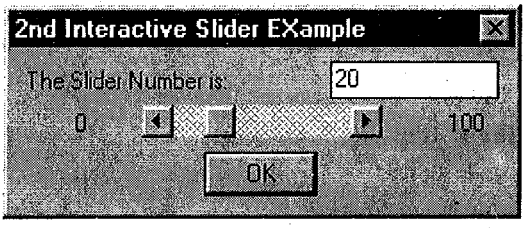
```
//Tên file: EX18-12.DCL
//Mục đích: Minh họa cách sử dụng thanh slider.
EX12: dialog {
    label = "Interactive Slider Example";
    : column {
      : row {
        : text_part {
          value = "The Slider Number is: ";
        }
        : text_part {
          value = 50;
          key = "numtxt";
        }
      }
      : row {
        spacer_0;
        : text_part {
          value = "0";
          align = right;
        }
        : slider {
          key = "sld";
          value = "50";
          max_value = 100;
          min_value = 0;
          big_increment = 10;
          small_increment = 1;
          fixed_width = true;
          width = 20;
        }
        : text_part {
          value = "100";
          align = left;
        }
      }
    }
    ok_only;
}
//Kết thúc file EX18-12.DCL
```

File EX18-12.LSP

```
;Tên file: EX18-12.LSP
;Mục đích: Hiển thị hộp thoại EX12 trong file EX18-12.DCL
(defun C:EX12 ()
  (setq DCL_ID (load_dialog "EX18-12.DCL"))
  (new_dialog "EX12" DCL_ID)
  (action_tile "sld" "(set_tile \"numtxt\" (setq VAL $value))")
  (start_dialog)
  (alert (strcat "Your final slider value was: " VAL))
  (princ)
)
;Kết thúc file EX18-12.LSP
```

Ví dụ:

Khi *slider* được liên kết với một *edit_box*, người sử dụng có thể nhập giá trị bằng một trong 2 cách: sử dụng *slider* hoặc sử dụng *edit_box*. Khi thay đổi giá trị của *slider* , giá trị của *edit_box* cũng thay đổi theo tương ứng, và ngược lại khi thay đổi giá trị của *edit_box*, giá trị của *slider* cũng thay đổi theo tương ứng.



File EX18-13.DCL

```
//Tên file: EX18-13.DCL
//Mục đích: Minh họa việc liên kết giữa slider và edit_box.
EX13: dialog {
    label = "2nd Interactive Slider Example";
     : column {
     : row {
       : edit_box {
         label = "The Slider Number is: ";
         value = "50";
         key = "edit_1";
       }
```

```
    }
    : row {
      spacer_0;
      : text_part {
        value = "0";
        align = right;
      }
      : slider {
        key = "sld";
        value = "50 ;
        max_value = 100;
        min_value = 0;
        big_increment = 10;
        small_increment = 1;
        fixed_width = true;
        width = 20;
      }
      : text_part {
        value = "100";
        align = left;
      }
    }
  }
  ok_only;
}
//Kết thúc file EX18-13.DCL
```

File EX18-13.LSP

```
;Tên file: EX18-13.LSP
;Mục đích: Hiển thị hộp thoại EX13 trong file EX18-13.DCL
(defun C:EX13 ()
  (setq DCL_ID (load_dialog "EX18-13.DCL"))
  (new_dialog "EX13" DCL_ID)
  (action_tile "sld" "(set_tile \"edit_1\" (setq VAL $value))")
  (action_tile "edit_1" "(set_tile \"sld\" (setq VAL $value))")
  (start_dialog)
  (alert (strcat "Your final slider value was: " VAL))
  (princ)
)
;Kết thúc file EX18-13.LSP
```

18.6 Một số đặc điểm cần chú ý khi thiết kế hộp thoại

Khi thiết kế các hộp thoại, chúng ta nên cố gắng làm cho chúng giống với các hộp thoại của **AutoCAD** mà người sử dụng đã quen sử dụng. Nhờ đó, người sử dụng dễ dàng nắm vững chương trình do chúng ta viết ra.

Các phím nóng

Đây là chức năng quan trọng mà người sử dụng thường dùng, đặc biệt là trong các trường hợp không dùng được con chuột. Khi sử dụng phím nóng, người sử dụng dễ dàng chuyển con trỏ đến vị trí *tile* mong muốn, nhanh hơn là sử dụng con chuột.

Mỗi *tile* có một phím nóng riêng, không trùng với phím nóng của các *tile* khác. Ký tự sử dụng làm phím nóng phải gợi nhớ đến chức năng của *tile*.

Các Tab Stop

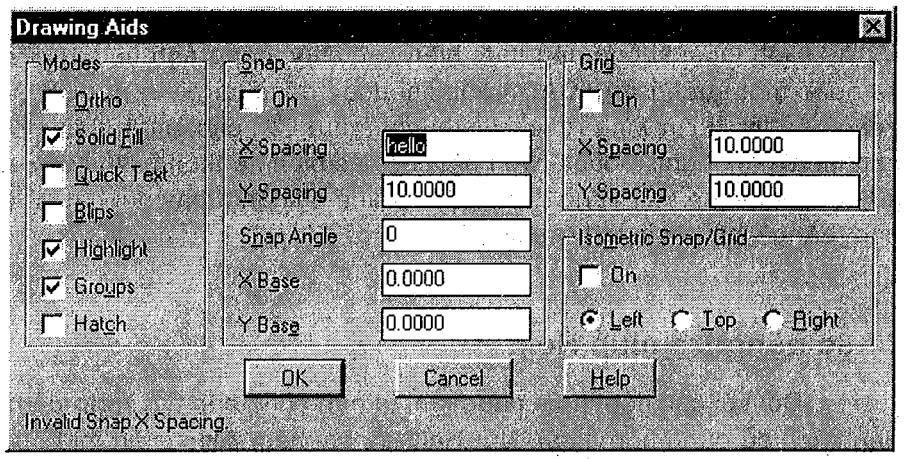
Chức năng này cho phép người sử dụng di chuyển lần lượt qua các *tile* bằng cách nhấn phím TAB. Con trỏ chỉ dừng lại ở các *active tile* (*button, edit_box, image_button, list_box, popup_list, radio_button, slider, toggle)* và thuộc tính *is_tab_stop* của chúng phải bằng *true* (mặc định).

Thông thường, thứ tự di chuyển qua các *tile* giống thứ tự mô tả các *tile* trong file DCL. Trong một vài trường hợp đặc biệt, ta có thể thay đổi thứ tự này. Trong biểu thức *action*, ta kiểm tra giá trị của biến *$reason* để quyết định con trỏ sẽ tiếp tục di chuyển đến đâu bằng hàm **Mode_tile** (xem ví dụ EX18-7.LSP trong chương này).

Kiểm tra lỗi

Việc kiểm tra lỗi rất cần thiết khi trong hộp thoại có chứa các *edit_box*. Vì *edit_box* chấp nhận mọi giá trị nhập vào, nên sẽ xảy ra trường hợp cần phải nhập vào một số nhưng người sử dụng lại nhập vào một chuỗi. Sau đó, các biểu thức sử dụng giá trị này sẽ bị lỗi vì kiểu dữ liệu không phù hợp.

Ta nên sử dụng phương pháp mà **AutoCAD** sử dụng trong việc kiểm tra lỗi. Xét ví dụ sau:



Thay vì nhập một số vào *edit_box* "X Spacing", người sử dụng lại nhập chuỗi "hello" và nhấn Enter. Khi đó, thông báo lỗi sẽ xuất hiện tại *error tile* "*Invalid Snap X Spacing*". Tuy nhiên, người sử dụng không nhất thiết phải sửa ngay lỗi này mà vẫn có thể di chuyển qua các *tile* khác để nhập giá trị. Chỉ khi người sử dụng nhấn nút OK để đóng hộp thoại, việc kiểm tra lỗi lần cuối mới được thực hiện. Nếu có lỗi, hộp thoại sẽ không được đóng lại, và con trỏ sẽ chuyển đến nơi bị lỗi, nội dung bị lỗi sẽ được chiếu sáng. Chỉ khi tất cả các lỗi đã được sửa xong, hộp thoại mới được đóng lại.

Trong một số trường hợp với các lỗi nghiêm trọng, ta sử dụng hàm **Alert** để hiện thông báo lỗi. Khi đó, chỉ khi người sử dụng đọc xong nội dung thông báo lỗi và nhấn nút OK thì mới tiếp tục làm việc với hộp thoại được.

Ngoài ra, ta phải nhớ xóa nội dung của *errtile* khi lỗi đã được sửa.

Sắp xếp các tile

- Các *tile* liên quan với nhau nên được sắp xếp gần nhau. Các *tile* được canh lề thẳng hàng, khi cần thì sử dụng thêm các *spacer* để điều chỉnh khoảng cách.
- Các *tile* quan trọng nên đặt ở các vị trí nổi bật, dễ nhìn thấy.
- Nên sử dụng các *column cluster* thay cho các *row cluster.*
- Các dòng chữ xuất hiện trên hộp thoại nên được định dạng giống các dòng chữ trên các hộp thoại của **AutoCAD**.

Đóng hộp thoại

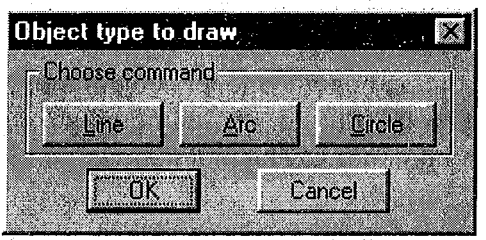
Hộp thoại không nên thực hiện các chức năng ảnh hưởng đến bản vẽ hoặc môi trường **AutoCAD** trước khi được đóng lại bằng nút OK. Ví dụ, trong hộp thoại Drawing Adis, chỉ khi nhấn nút OK, các giá trị thay đổi trên hộp thoại mới thực sự được cập nhật cho các biến hệ thống. Như vậy, người sử dụng vẫn còn khả năng đóng hộp thoại mà không thay đổi bất cứ giá trị nào (bằng cách nhấn nút Cancel).

Chức năng trợ giúp Help

Đối với các hộp thoại phức tạp, ta nên cung cấp nút **Help** để hướng dẫn cách sử dụng hộp thoại. Nút **Help** sẽ gọi hàm **Alert** hiển thị hộp thoại chứa nội dung trợ giúp, hoặc làm xuất hiện hộp thoại "**AutoCAD Help**" của **AutoCAD**.

18.7 Bài tập

1. Hãy tạo các file DCL và LSP làm xuất hiện hộp thoại sau. Khi chọn các nút, hộp thoại sẽ đóng lại và các lệnh tương ứng được thực hiện.



2. Bổ sung cho hộp thoại trong ví dụ EX18-13 chức năng kiểm tra lỗi cho giá trị nhập vào *edit_box* phải lớn 0 và nhỏ hơn 100.

18.8 Lời giải

1.

File BT18-1.DCL

```
//Tên file: BT18-1.DCL
//
```

```
BT1 : dialog {
        label = "Object type to draw";
        : boxed_row {
          label = "Choose command";
          :button {
           label = "&Line";
           width = 10;
           key = "btn_line";
          }
          :button {
           label = "&Arc";
           width = 10;
           key = "btn_arc";
          }
          :button {
           label = "&Circle";
           width = 10;
           key = "btn_circle";
          }
         }
        ok_cancel;
}
//Kết thúc file BT18-1.DCL
```

File BT18-1.LSP

```
;Tên file: BT18-1.LSP
;Mục đích: Hiển thị hộp thoại BT1 trong file BT18-1.DCL
(defun C:BT1 (/ COM)
  (setq DCL_ID (load_dialog "BT18-1.DCL"))
  (new_dialog "BT1" DCL_ID)
  (action_tile "btn_line" "(done_dialog) (setq COM \".line\")")
  (action_tile "btn_arc" "(done_dialog) (setq COM \".arc\")")
  (action_tile "btn_circle" "(done_dialog) (setq COM \".circle\")")
  (start_dialog)
  (command COM)
  (princ)
)
;Kết thúc file BT18-1.LSP
```

2.

File EX18-13.DCL

```
//Tên file: EX18-13.DCL
//Mục đích: Minh họa việc liên kết giữa slider và edit_box.
EX13: dialog {
    label = "2nd Interactive Slider Example";
     : column {
     : row {
       : edit_box {
         label = "The Slider Number is: ";
         value = "50";
         key = "edit_1";
       }
     }
     : row {
       spacer_0;
       : text_part {
         value = "0";
         align = right;
       }
       : slider {
         key = "sld";
         value = "50";
         max_value = 100;
         min_value = 0;
         big_increment = 10;
         small_increment = 1;
         fixed_width = true;
         width = 20;
       }
       : text_part {
         value = "100";
         align = left;
       }
     }
   }
   ok_cancel_err;
}
//Kết thúc file EX18-13.DCL
```

File BT18-2.LSP

```
;Tên file: BT18-2.LSP
;Mục đích: Hiển thị hộp thoại EX13 trong file EX18-13.DCL
;
;Định nghĩa hàm VALIDATE kiểm tra giá trị nhập
;
(defun VALIDATE (val key)
  (if (= "." (substr VAL 1 1)) (setq VAL (strcat "0" VAL))
   )                                    ;Nếu giá trị nhập
                                        ;bắt đầu bằng dấu chấm (.) thì thêm
                                        ;số 0 vào đằng trước
  (cond
   ( (not                               ; Kiểm tra giá trị nhập có phải là
      (or                               ; số thực hoặc số nguyên hay không
       (= (type (read VAL)) 'REAL)
       (= (type (read VAL)) 'INT)
      )
     )
      (set_tile "error" "Invalid entry"); hiện thông báo lỗi.
   )                                    ;Đóng điều kiện 1

   ( (or                                ;Kiểm tra số nhập vào có nằm trong
      (< (atof VAL) 1)                  ; miền giá trị từ 0 đến 100 hay không
      (> (atof VAL) 100)
     )
      (set_tile "error" "Invalid entry")  ; hiện thông báo lỗi.
     )                                  ;Đóng điều kiện 2
   ( T
     (progn                             ;Nếu phải (giá trị nhập hợp lệ) thì
       (set_tile "error" "")            ; xóa thông báo lỗi
       (set_tile "sld" val)             ;Thay đổi thanh slider cho phù hợp
      )
    )                                   ;Đóng điều kiện 3
  )                                     ; Đóng  cond
)                                       ;Đóng hàm defun VALIDATE
;
(defun EDIT_CALL ()
  (VALIDATE (get_tile "edit_1") "edit_1") ; Kiểm tra giá trị nhập
)
;
(defun OK_CALL ()
```

```
   (VALIDATE (get_tile "edit_1") "edit_1")      ;Kiểm tra giá trị nhập
   (mode_tile "edit_1" 2)                       ;Chuyển con trỏ về edit_box.
   (if (= (get_tile  "error") "")               ;Nếu không có lỗi thì
      (progn                                    ;
        (setq RES (get_tile "edit_1"))          ; Gán giá trị nhập cho RES
        (done_dialog)                           ; và đóng hộp thoại.
      )
   )
)
;
(defun SLD_CALL ()
  (set_tile "error" "")                         ; xóa thông báo lỗi
  (set_tile "edit_1" $value)                    ; Gán giá trị tương ứng cho edit_box.
)
;
(defun C:BT2 (/ RES)
   (setq DCL_ID (load_dialog "EX18-13.DCL"))
   (new_dialog "EX13" DCL_ID)
   (action_tile "sld" "(SLD_CALL)")
   (action_tile "edit_1" "(EDIT_CALL)")
   (action_tile "accept" "(OK_CALL)")
   (start_dialog)
   (if (not (null RES)) (alert (strcat "Your final slider value was: " RES)))
   (unload_dialog DCL_ID)
   (princ)
)
;Kết thúc file BT18-2.LSP
```

Chương 19

KỸ THUẬT LẬP TRÌNH VÀ GỠ RỐI CHƯƠNG TRÌNH

Nội dung chương

1. Cách trình bày chương trình cho dễ đọc và dễ tìm lỗi.
2. Các phương pháp tìm lỗi trong chương trình.
3. Chia nhỏ chương trình để dễ viết và dễ tìm lỗi.
4. Tạo cho chương trình một giao diện thân thiện với người sử dụng.

Tương tự như với các ngôn ngữ lập trình khác, khi lập trình bằng ngôn ngữ **AutoLISP** ta thường gặp phải các lỗi. Theo thống kê, những người mới học lập trình phải bỏ ra khoảng 75% thời gian cho việc tìm và sửa lỗi chương trình. Những người có kinh nghiệm hơn cũng phải mất khoảng 40-50% thời gian. Do đó, ta phải có những phương pháp hiệu quả trong việc tìm và sửa lỗi chương trình.

19.1 Cách trình bày chương trình

Một trong những phương pháp hiệu quả để dễ sửa lỗi chương trình là trình bày chương trình sao cho dễ đọc, dễ theo dõi ý nghĩa của các dòng lệnh, các biến ...

Canh lề

Việc canh lề các dòng lệnh trong chương trình giúp chúng ta dễ theo dõi các đoạn chương trình lồng nhau và tránh được các lỗi dư hoặc thiếu các dấu đóng ngoặc. Cần chú ý:

- Các dấu đóng ngoặc ")" phải trên cùng một cột với các dấu mở ngoặc "(" tương ứng.
- Các biểu thức ở mức sâu hơn phải được canh lề ở các cột thụt vào so với các biểu thức ở mức ngoài.

Ví dụ:

```
(defun C:COUNT ()
   (princ
      (strlen
         (getstring T "\nEnter string to count: ")
      )                                   ;Đóng strlen
   )                                      ;Đóng princ
   (princ)
)                                         ;Đóng defun
```

- Với các biểu thức ngắn, chứa đủ trên một dòng và chỉ có hai hoặc ba mức lồng nhau, ta có thể viết trên một dòng cho dễ đọc.

Ví dụ:

```
(defun DTR (a) (* pi (/ 180.0 a )))
```

✌ **Ví dụ:**

Trong hai cách trình bày chương trình sau đây, cách thứ hai dễ đọc hơn vì được chú thích đầy đủ, canh lề đúng đắn và còn dùng phương pháp viết hoa viết thường thích hợp (xem phần kế tiếp).

Cách 1:

```
(DEFUN PARSE (STR / REF N CHAR LSTR LADD LST) (SETQ REF
(STRLEN STR) N 1) (WHILE (>= REF N) (SETQ CHAR (SUBSTR STR N 1))
(WHILE (AND (/= CHAR ",") (/= CHAR "")(/= CHAR " ") ) (IF (NULL
LSTR) (SETQ LSTR CHAR) (SETQ LSTR (STRCAT LSTR CHAR)) )(SETQ N
(1+ N))(SETQ CHAR (SUBSTR STR N 1)) ) WHILE (SETQ LADD LSTR LSTR
NIL) (SETQ N (1+ N)) (IF (/= LADD NIL) (SETQ LST (APPEND LST
(LIST LADD) )))))
```

Cách 2:

```
;Tên file:    PARSE.LSP
;
;Mục đích: Đọc vào một chuỗi gồm nhiều từ ngăn cách nhau bằng dấu
;          phẩy (,) hoặc bằng khoảng trắng và trả về một danh sách gồm
;          các từ tạo nên chuỗi này.
;
;Ví dụ về cách gọi hàm:  (PARSE "How are you, Mary?")
;Trả về danh sách:       ("How" "are" "you" "Mary?")
;
(defun PARSE (STR / REF N CHAR LSTR LADD LST)
  (setq REF (strlen STR) N 1)
  (while (>= REF N)
    (setq CHAR (substr STR N 1))
    (while
      (and
        (/= CHAR ",")
        (/= CHAR "")
        (/= CHAR " ")
      )                          ; đóng AND
      (if
        (null LSTR)
        (setq LSTR CHAR)
        (setq LSTR (strcat LSTR CHAR))
      )                          ; đóng IF
      (setq N (1+ N))
      (setq CHAR (substr STR N 1))
    )                            ; đóng WHILE
```

```
    (setq LADD LSTR LSTR nil)
    (setq N (1+ N))
    (if
      (/= LADD nil)
      (setq LST (append LST
                        (list LADD)
                )                    ; đóng APPEND
      )                              ; đóng SETQ
    )                                ; đóng IF
  )                                  ; đóng WHILE
)                                    ; đóng DEFUN
;Kết thúc file  PARSE.LSP
```

Viết hoa viết thường

Sử dụng cách viết hoa viết thường hợp lý sẽ giúp chúng ta dễ đọc cấu trúc của chương trình. Ta có thể sử dụng quy tắc sau:

- Tên hàm: viết thường (ví dụ: **defun**)
- Tên biến và tham số: viết hoa (ví dụ: PARSE, LSTR)

19.2 Tìm các lỗi trong chương trình

Ngoài cách trình bày cho chương trình dễ đọc, người lập trình nên hiểu ý nghĩa các thông báo lỗi của **AutoLISP** để có phương án sửa lỗi phù hợp.

Sau đây là bảng liệt kê một số thông báo lỗi thường gặp và nguyên nhân gây ra các lỗi này.

Thông báo lỗi	Nguyên nhân
malformed list	Thiếu các dấu đóng ngoặc
malformed string	Thiếu dấu nháy đóng chuỗi
bad argument type	Tham số cung cấp khi gọi hàm không đúng kiểu dữ liệu mà hàm yêu cầu.
bad function	Tên hàm bị sai, thường là do lỗi chính tả.
null function	Tương tự như *bad function.* Và khi

	AutoLISP định giá trị hàm này thì kết quả bằng *nil*.
too many arguments	Số lượng các tham số cung cấp khi gọi hàm nhiều hơn cần thiết.
too few arguments	Cung cấp không đủ các tham số mà hàm bắt buộc phải có.
incorect number of arguments to a function	Cung cấp sai số lượng các tham số khi gọi hàm tự tạo.
extra right parenthesis	Hoặc thiếu các dấu mở ngoặc, hoặc dư các dấu đóng ngoặc.

Khi xuất hiện lỗi, chương trình sẽ ngừng việc tính toán và hiện biểu thức gây ra lỗi lên màn hình. Tiếp theo là các biểu thức lồng bên ngoài biểu thức lỗi, giúp chúng ta định được vị trí chính xác của biểu thức gây ra lỗi. Số lượng các biểu thức này nhiều hay ít tùy thuộc vào độ phức tạp của chương trình.

Sau đây là một chương trình có chứa lỗi, và phương pháp tìm ra lỗi này.

Ví dụ:

```
(defun C:DRAWBOX (/ P1 D P4 P3 P2)
  (setq P1 (getpoint "\nLower left corner: "))
  (setq D (getdist "\nLength of one side: "))
  (setq P4
    (polar
      (setq P3
        (polar
          (setq P2
           (polar P1 0 DD)        ; Biểu thức gây lỗi. Biến D bị lộn thành DD
          )
          (/ pi 2)
          D
        )
      )
      pi
      D
    )
  )
```

```
  (command "LINE" P1 P2 P3 P4 "c")
  (princ)
)
```

- Khi thực hiện chương trình, thông báo lỗi xuất hiện:

  Command: **(load "DRAWBOX")** ↵
  C:DRAWBOX
  Command: **DRAWBOX** ↵
  Lower left corner: (Chọn một điểm trên màn hình)
  Length of one side: **200** ↵
  error: bad argument type — Thông báo lỗi.
  (POLAR P1 0 DD) — Biểu thức gây ra lỗi.
  (SETQ P2 (POLAR P1 0 DD)) — Biểu thức lồng ở ngoài một mức
  (POLAR (SETQ P2 (POLAR P1 0 DD)) (/ PI 2) D) — Biểu thức lồng ở ngoài ở mức tiếp theo (và tiếp tục như vậy...)
  (SETQ P3 (POLAR (SETQ P2 (POLAR P1 0 DD)) (/ PI 2) D))
  (POLAR (SETQ P3 (POLAR (SETQ P2 (POLAR P1 0 DD)) (/ PI 2) D)) PI D)
  (SETQ P4 (POLAR (SETQ P3 (POLAR (SETQ P2 (POLAR P1 0 DD)) (/ PI 2) D)) PI D))
  (C:DRAWBOX)
  Cancel — Biểu thức ngoài cùng, thông thường là hàm được gọi.

  Vì các biến P1, D, P4, P3 và P2 được khai báo là biến cục bộ, nên giá trị của chúng bị mất đi khi chương trình kết thúc. Để xác định nguyên nhân gây lỗi, ta cần phải biết giá trị các biến này tại thời điểm xuất hiện lỗi.

- Sửa chương trình lại như sau (thêm dấu đóng ngoặc và dấu chấm phẩy) để các biến trở thành biến toàn cục và chạy lại chương trình.

```
(defun C:DRAWBOX ( ) ;/ P1 D P4 P3 P2)
```

Sau khi chạy lại chương trình, giá trị các biến được giữ lại tại thời điểm gặp lỗi. Ta sẽ xem xét các giá trị này để tìm lỗi. Lỗi ở đây là sai kiểu dữ liệu (*bad argument type*) và biểu thức bị lỗi là *(POLAR P1 0 DD)*.

- Kiểm tra kiểu của biến P1 có phải là tọa độ điểm hay không.

Command: **!P1** ↵
(101.69 130.153 0.0) Đúng. Đây là tọa độ điểm

- Kiểm tra giá trị **0** có đúng là số 0 hay bị nhập vào nhầm là chữ O.
- Kiểm tra kiểu của biến DD có phải là một số hay không.

Command: **!DD** ↵
nil Không. Vậy đây là lỗi.Tại sao? Vì biến DD không được định nghĩa trong chương trình nên nó có giá trị là *nil*.

- Sửa lại biến DD thành D, chương trình sẽ chạy đúng.

Trong ví dụ này, lỗi xuất hiện do tên biến bị viết sai. Trong các trường hợp khác, lỗi có thể là do gán giá trị sai, biểu thức không đúng... Nhờ phát hiện tên biến gây ra lỗi, ta có được dấu vết để đi tìm lỗi.

19.3 Chia chương trình thành các hàm nhỏ hơn

Với các chương trình lớn, ta rất khó theo dõi diễn tiến của chương trình và do đó rất khó tìm ra vị trí lỗi. Cách tốt nhất là chia chương trình thành nhiều hàm nhỏ hơn, mỗi hàm giải quyết một vấn đề riêng. Nhờ đó, phạm vi tìm lỗi được thu hẹp. Ngoài ra các hàm có thể được sử dụng nhiều lần, làm giảm việc lặp đi lặp lại các đoạn lệnh giống nhau.

Ví dụ:

Trong ví dụ sau đây, biểu thức **entmake** có chứa một lỗi. Nếu chương trình chứa hàng trăm dòng lệnh thì ta sẽ không dễ dàng tìm ra vị trí lỗi như trong trường hợp này.

```
(defun C:CLINE ()
  (setq FPT (getpoint "From point: ")
        CLR 1
  )
  (while (and
           FPT
           (setq TPT (getpoint FPT "\nTo point: "))
```

```
        )
      (entmake (list '(0 . "LINE") (cons 10 FPP) (cons 11 TPT) (cons 62 CLR)))
      (if ( < CLR 16)
         (setq CLR (1+ CLR))
         (setq CLR 1)
      )
       (setq FPT TPT)
   )
   (princ)
)
```

Command: **Cline** ↵
From point: (Chọn một điểm)
To point: error: bad entmake list
(ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR)))
(WHILE (AND FPT (SETQ TPT (GETPOINT FPT "\nTo point: "))) (ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR))) (IF (< CLR 16) (SETQ CLR (1+ CLR)) (SETQ CLR 1)) (SETQ FPT TPT))
(C:CLINE)
Cancel

Chương trình trên được viết lại bằng cách chia thành nhiều hàm nhỏ.

```
;Định nghĩa hàm DRAWCLINE vẽ một đường thẳng qua 2 điểm P1 và P2
;và có màu là CV.
;
(defun DRAWCLINE (P1 P2 CV)
   (entmake (list '(0 . "LINE") (cons 10 FPP) (cons 11 TPT) (cons 62 CLR)))
)
;
;Định nghĩa hàm LOOPINC  dùng để tăng tham số NUM thêm 1. Nếu vượt
;quá giá trị MAXVAL thì gán NUM về lại bằng 1
;
(defun LOOPINC (NUM MAXVAL)
   (if (< NUM MAXVAL)
```

```
    (setq NUM (1+ NUM))
    (setq NUM 1)
  )
)
;
;Hàm CLINE vẽ các đường thẳng nhiều màu
;
(defun C:CLINE ()
  (setq FPT (getpoint "From point: ")
        CLR 1
  )
  (while (and
           FPT
           (setq TPT (getpoint FPT "\nTo point: "))
         )
    (DRAWCLINE FPT TPT CLR)
    (setq CLR (LOOPING CLR 16)
          FPT TPT
    )
  )
  (princ)
)
```

Command: **Cline** ↵
From point: (Chọn một điểm trên màn hình)
To point: error: bad entmake list
(ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR)))
(DRAWCLINE FPT TPT CLR)
(WHILE (AND FPT (SETQ TPT (GETPOINT FPT "\nTo point: "))) (DRAWCLINE FPT TPT CLR) (SETQ CLR (LOOPING CLR 16) FPT TPT))
(C:CLINE)
Cancel

Thông báo lỗi trong trường hợp này cho biết lỗi xuất hiện trong biểu thức **Entmake** khi hàm DRAWCLINE được gọi. Nhờ vậy, vị trí tìm lỗi được thu hẹp: hoặc tại vị trí gọi hàm hoặc trong phần định nghĩa hàm.

19.4 Chức năng đánh dấu chương trình (*Tracer*)

Trong một số trường hợp, mặc dù chương trình không gây ra lỗi nào khi chạy, nhưng chương trình vẫn không tạo ra kết quả như mong muốn. Lỗi này có thể do chúng ta tạo ra các hàm có kết quả tính toán không chính xác. Để tìm ra các lỗi này, chúng ta có thể sử dụng chức năng đánh dấu chương trình.

Chức năng này cho phép đánh dấu các hàm cần theo dõi kết quả tính toán khi chạy chương trình. Nhờ đó có thể phát hiện được các hàm tạo ra kết quả không đúng.

Hàm TRACE

Hàm **Trace** dùng để đánh dấu các hàm cần theo dõi kết quả. Khi đó, mỗi lần gọi đến các hàm này, **AutoLISP** sẽ thông báo trên màn hình và sau đó hiện kết quả tính toán.

(**Trace** FUNCTION . . .)

Ví dụ:

Trong chương trình sau đây, hàm **dtr** bị định nghĩa sai, do đó chương trình tạo ra kết quả không đúng.

```
;Định nghĩa hàm đổi đơn vị đo góc từ độ sang radian
(defun dtr (a) (/ pi (* a 180)) )
;
(defun C:BOXES ()
  (setq SZ (getdist "\nLength of one side of box: "))
  (setq P1 (getpoint "\nLower left corner: "))
  (setq ANG (getreal "\nAngle for box: "))
  (setq P2 (polar P1 (dtr ANG) SZ))
  (setq P3 (polar P2
                  (+ (dtr ANG) (* pi 0.5))
                   SZ
           )
  )
  (setq P4 (polar P3
                  (+ (dtr ANG) pi)
```

```
                    SZ
                )
    )
    (command ".LINE" P1 P2 P3 P4 "c")
    (princ)
)
```

Command: **boxes** ↵
Length of one side of box: **100** ↵
Lower left corner: (Chọn một điểm trên màn hình)
Angle for box: **45** ↵
Command:

Mặc dù chương trình vẫn vẽ được một hình vuông trên màn hình, nhưng nó không nghiêng góc 45^0 như mong muốn. Do đó ta cần kiểm tra hàm liên quan là hàm **dtr** bằng cách dùng hàm **Trace**.

Command: **(trace dtr)** ↵
DTR
Command: **boxes** ↵
Length of one side of box: **100** ↵
Lower left corner: (Chọn một điểm trên màn hình)
Angle for box: **45** ↵
Entering DTR: 45.0
Result: 0.000387851
Entering DTR: 45.0
Result: 0.000387851
Entering DTR: 45.0
Result: 0.000387851
Command:

Nhờ các thông báo về kết quả sử dụng hàm **dtr**, ta phát hiện được hàm này đã được định nghĩa sai, vì góc 45^0 tính bằng đơn vị radian sẽ bằng 0.25 x pi = 0.875.

Hàm UNTRACE

Hàm **Untrace** dùng để loại bỏ việc đánh dấu của các hàm trong chương trình.

(**Untrace** FUNCTION . . .)

Ví dụ:

Loại bỏ việc đánh dấu hàm **Dtr**

Command: **(untrace dtr)** ↵
dtr

19.5 Sử dụng hàm Alert để bẫy lỗi

Hàm **Alert** dùng để dừng chương trình, hiện hộp thoại thông báo và đợi người sử dụng nhấn nút OK để tiếp tục chạy chương trình. Trong việc tìm lỗi chương trình, ta dùng hàm **alert** để đánh dấu các thời điểm kiểm tra chương trình và hiện giá trị các biến tại các thời điểm kiểm tra này.

Hiện giá trị các biến

Ta dùng hàm **Alert** để theo dõi sự thay đổi giá trị của các biến trong chương trình.

Ví dụ:

Chương trình sau đây dùng để vẽ các đường thẳng với các màu khác nhau. Tuy nhiên, vì có lỗi nên chỉ vẽ được các đường thẳng màu đỏ.

```
;Định nghĩa hàm DRAWCLINE vẽ một đường thẳng qua 2 điểm P1 và P2
;và có màu là CV.
;
(defun DRAWCLINE (P1 P2 CV)
  (entmake (list '(0 . "LINE") (cons 10 FPP) (cons 11 TPT) (cons 62 CLR)))
)
;
;Định nghĩa hàm LOOPINC  dùng để tăng tham số NUM thêm 1. Nếu vượt
;quá giá trị MAXVAL thì gán NUM về lại bằng 1
;
(defun LOOPINC (NUM MAXVAL)
```

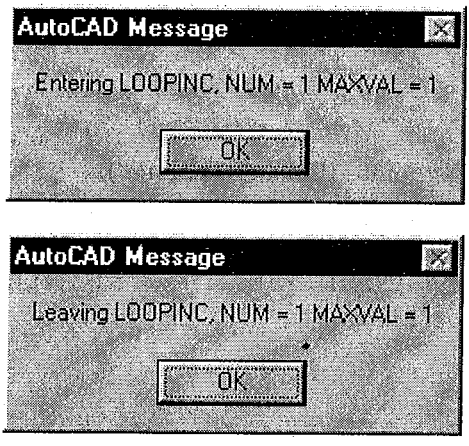
```
  (if (< NUM MAXVAL)
    (setq NUM (1+ NUM))
    (setq NUM 1)
  )
)
;
;Hàm CLINE vẽ các đường thẳng nhiều màu
;
(defun C:CLINE ()
  (setq FPT (getpoint "From point: ")
        CLR 1
  )
  (while (and
           FPT
           (setq TPT (getpoint FPT "\nTo point: "))
         )
    (DRAWCLINE FPT TPT CLR)
    (setq CLR (LOOPINC CLR 1)                    ;Vị trí gây ra lỗi
          FPT TPT
    )
  )
  (princ)
)
```

Vì màu không thay đổi, nên đầu tiên chúng ta cần kiểm tra hàm LOOPINC. Bổ sung hàm **Alert** vào hàm LOOPINC như sau:

```
(defun LOOPINC (NUM MAXVAL)
  (alert (strcat "Entering LOOPINC, NUM = "
                 (itoa NUM) "MAXVAL = " (itoa MAXVAL))
  )
  (if (< NUM MAXVAL)
    (setq NUM (1+ NUM))
    (setq NUM 1)
  )
  (alert (strcat "Leaving LOOPINC, NUM = "
                 (itoa NUM) "MAXVAL = " (itoa MAXVAL))
  )
```

```
    (eval NUM)                    ;Ghi chú: Vì hàm LOOPINC cần phải
                                  ;trả về giá trị của biến NUM
)
```

Khi thực hiện chương trình, 2 hộp thoại **Alert** sau đây sẽ xuất hiện. Nhờ đó, ta phát hiện ra giá trị của biến NUM không được hàm LOOPINC tăng lên, lý do là biến MAXVAL = 1. Lỗi ở biểu thức gọi hàm LOOPINC.



Tạo các điểm kiểm tra chương trình

Ta có thể dùng hàm **Alert** để tạo ra các điểm kiểm tra chương trình, chia chương trình thành nhiều đoạn khi chạy. Nhờ đó, dễ phát hiện lỗi xảy ra ở đoạn nào của chương trình.

Cách này thường dùng để phát hiện các lỗi *"Malformed list"* hoặc *"Extra right parenthese"*... vì **AutoLISP** không thông báo được vị trí lỗi trong những trường hợp này.

Ví dụ:

```
;Tên file: FASTKEYS.LSP
;Mục đích: Tạo các lệnh tắt cho AutoCAD.
```

```
;
(defun C:AR () (command ".ARRAY") (princ))
(defun C:B () (command ".BREAK") (princ))
;
(defun C:BA (/ SS BP)
  (setvar "osmode" 512)
  (setq SS (getpoint "\nSelectline to break: "))    ; Chọn đường thẳng
  (redraw (ssname (setq SST (ssget SS)) 0) 3)       ; Chiếu sáng đường thẳng.
  (if (not (or (= "LINE" (cdr (assoc 0 (entget (ssname SST 0)))))
               (= "ARC" (cdr (assoc 0 (entget (ssname SST 0)))))
               (= "POLYLINE" (cdr (assoc 0 (entget (ssname SST 0)))))
           )
      )
      (progn
          (redraw (ssname SST 0) 1)
          (princ "\nCan't BREAK that entity.")
      )
     (progn
        (setvar "OSMODE" 32)                    ; Phương pháp truy bắt INT
        (setq BP (getpoint "\nIntersection to BREAK: ")); Chọn điểm INT
        (command "BREAK" SS "F" BP "@")  ;Thực hiện lệnh BREAK
        (setvar "OSMODE" 0)
        (setvar "HIGHLIGHT" 1)
        (princ)
     )
  )
  (princ)
)
;
;Hàm BYL dùng để thay đổi màu sắc và dạng đường của các đối tượng
;chọn thành BYLAYER.
(defun C:BYL (/ SS)
  (if (setq SS (ssget))
    (command "CHANGE" SS "" "P" "C" "BYL" "LT" "BYLAYER")
    (princ "\nNo entities selected!")
  )
  (princ)
)
;
;
(defun C:CG () (command "CHANGE") (princ)
```

```
(defun C:CX () (command "COPY") (princ))
(defun C:EX () (command "EXTEND") (princ))
(defun C:F () (command "FILLET") (princ))
(defun C:FR (/ r) (setq r (getdist "\nFillet radius: "))
  (command "FILLET" "R" r) (C:F)
)
(defun C:FX () (command "FILLET" "R" "0")(C:F))
(defun C:HO ()
  (initget 3)
  (setq OV (getreal "\nOffset to halve: "))
  (command ".OFFSET" (* 0.5 OV))
)
(defun C:M () (command "MOVE") (princ))
(defun C:MI () (command "MIRROR" "AU" ) (princ))
(defun C:O () (command "OFFSET" ) (princ))
```

Khi tải chương trình trên, xuất hiện lỗi như sau:

```
Command: (load "fastkeys") ↵
error: malformed list
(LOAD "fastkeys")
*Cancel*
```

Thêm các biểu thức **Alert** tạo các điểm kiểm tra chương trình như sau:

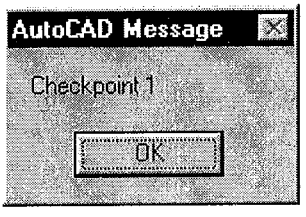
```
;Tên file: FASTKEYS.LSP
;Mục đích: Tạo các lệnh tắt cho AutoCAD.
;
(defun C:AR () (command ".ARRAY") (princ))
(defun C:B () (command ".BREAK") (princ))
(alert "Checkpoint 1")
;
(defun C:BA (/ SS BP)
  (setvar "osmode" 512)
  (setq SS (getpoint "\nSelectline to break: "))          ; Chọn đường thẳng
  (redraw (ssname (setq SST (ssget SS)) 0) 3)     ; Chiếu sáng đường thẳng.
```

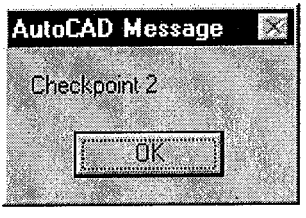
```
   (if (not (or (= "LINE" (cdr (assoc 0 (entget (ssname SST 0)))))
                (= "ARC" (cdr (assoc 0 (entget (ssname SST 0)))))
                (= "POLYLINE" (cdr (assoc 0 (entget (ssname SST 0)))))
       )     )
       (progn
             (redraw (ssname SST 0) 1)
             (princ "\nCan't BREAK that entity.")
       )
       (progn
          (setvar "OSMODE" 32)                    ; Phương pháp truy bắt INT
          (setq BP (getpoint "\nIntersection to BREAK: ")); Chọn điểm INT
          (command "BREAK" SS "F" BP "@")  ;Thực hiện lệnh BREAK
          (setvar "OSMODE" 0)
          (setvar "HIGHLIGHT" 1)
          (princ)
   ) )
   (princ)
)
;
;Hàm BYL dùng để thay đổi màu sắc và dạng đường của các đối tượng
;chọn thành BYLAYER.
(defun C:BYL (/ SS)
   (if (setq SS (ssget))
      (command "CHANGE" SS "" "P" "C" "BYL" "LT" "BYLAYER")
      (princ "\nNo entities selected!")
   )
   (princ)
)
;
;
(alert "Checkpoint 2")
(defun C:CG () (command "CHANGE") (princ)
(defun C:CX () (command "COPY") (princ))
(defun C:EX () (command "EXTEND") (princ))
(defun C:F () (command "FILLET") (princ))
(defun C:FR (/ r) (setq r (getdist "\nFillet radius: "))
   (command "FILLET" "R" r) (C:F)
)
(defun C:FX () (command "FILLET" "R" "0")(C:F))
(defun C:HO ()
```

```
  (initget 3)
  (setq OV (getreal "\nOffset to halve: "))
  (command ".OFFSET" (* 0.5 OV))
)
(defun C:M () (command "MOVE") (princ))
(defun C:MI () (command "MIRROR" "AU" ) (princ))
(defun C:O () (command "OFFSET" ) (princ))
(alert "Checkpoint 3")
```

Chạy lại chương trình này, lần lượt xuất hiện 2 hộp thoại **Alert**, sau đó là thông báo lỗi. Nhờ đó, ta biết lỗi xuất hiện ở đoạn giữa hai điểm kiểm tra 2 và 3.

Command: **(load "fastkeys")**↵





error: malformed list
(LOAD "fastkeys")
Cancel

Tạo thêm các điểm kiểm tra khác để thu hẹp phạm vi chứa lỗi, ta phát hiện được lỗi thiếu dấu đóng ngoặc tại biểu thức định nghĩa hàm C:CG.

19.6 Giao diện chương trình thân thiện

Giao diện chương trình nên thân thiện, tương tự các lệnh của **AutoCAD**. Nhờ đó người sử dụng cảm thấy thoải mái, quen thuộc khi chạy chương trình và nhập dữ liệu chính xác.

Giao diện chương trình nên tuân theo các quy định sau:

- Chỉ hiện kết quả cuối cùng, không hiện kết quả các hàm trung gian lên màn hình. Biểu thức sau đây nên đặt ở đầu chương trình:

```
(setvar "CMDECHO" 0)
```

- Sử dụng giá trị mặc định cho các dòng nhắc yêu cầu người sử dụng nhập dữ liệu. Các hàm thường dùng là **Initget** và **Getkword**. Tuy nhiên, vì hàm **Getkword** trả về giá trị *nil* khi người sử dụng không nhập dữ liệu mà chỉ nhấn phím ENTER để chấp nhận giá trị mặc định, nên ta phải kiểm tra giá trị nhập vào. Sau đây là một ví dụ:

```
(initget "Yes No")
(setq ANS (getkword "\nOk to proceed? <Y>: "))
(if (null ANS) (setq ANS "Yes"))
```

- Sử dụng các mã *ANSI escape code* trong các dòng nhắc và các thông báo. Nhờ đó các dòng chữ có thể in đậm hoặc có màu để tạo sự chú ý. Xem thêm phần phụ lục để hiểu về các mã này.
- Sử dụng menu để gọi lệnh. Xem giải thích về hàm **Menucmd** ở chương 16.
- Kiểm soát việc hiện các thông báo lỗi. Theo mặc định, khi gặp lỗi, **AutoLISP** sẽ hiện thông báo lỗi, sau đó là một loạt các dòng lệnh **AutoLISP**. Điều này có thể làm cho người sử dụng bối rối.

Ví dụ:

Command: **cline** ↵
From point:
To point: error: bad entmake list

(ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR)))
(WHILE (AND FPT (SETQ TPT (GETPOINT FPT "\nTo point: "))) (ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR))) (IF (< CLR 16) (SETQ CLR (1+ CLR)) (SETQ CLR 1)) (SETQ FPT TPT))
(C:CLINE)
Cancel

Ta có thể điều khiển việc hiển thị thông báo lỗi cho phù hợp với đối tượng sử dụng chương trình. Tạo hàm ***error*** như sau:

```
;Tên file: ERR.LSP
(defun *error*  (msg)
  (princ \nError: ")
  (princ msg)
  (princ)
)
```

Tải chương trình này vào bộ nhớ:

Command: **(load "err")** ↵
ERROR

Khi gặp lỗi, **AutoLISP** sẽ truyền chuỗi thông báo lỗi cho tham số MSG của hàm ***error*** và sau đó gọi hàm này để hiện thông báo lên màn hình.

Command: **cline** ↵
From point:
To point:
*Error: bad entmake list*Cancel**

Để trở về lại cách hiện thông báo lỗi mặc định của **AutoLISP** ta gán giá trị cho biến *error* bằng nil.

Command: **(setq *error* nil)** ↵
nil

Command: **cline** ↵
From point:
To point: error: bad entmake list
(ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR)))
(WHILE (AND FPT (SETQ TPT (GETPOINT FPT "\nTo point: "))) (ENTMAKE (LIST (QUOTE (0 . "LINE")) (CONS 10 FPP) (CONS 11 TPT) (CONS 62 CLR))) (IF (< CLR 16) (SETQ CLR (1+ CLR)) (SETQ CLR 1)) (SETQ FPT TPT))
(C:CLINE)
Cancel

Phương phàp hiệu quả nhất để quản lý lỗi là ngăn cản chúng xuất hiện. Một trong những nguyên nhân gây ra lỗi là dữ liệu nhập của người sử dụng không đúng. Nếu không thể ngăn cản chúng trực tiếp bằng hàm **Initget**, ta có thể dùng các phương pháp khác để kiểm tra.

Ví dụ:

Nếu chương trình yêu cầu phải chọn chính xác đối tượng là đường tròn, thì người sử dụng sẽ bị "khóa" lại trong vòng lặp cho đến khi chọn đúng.

```
(setq ES (entsel "\nSelect a circle: "))
(while
  (or
    (null ES)
    (/= "CIRCLE" (cdr (assoc 0 (entget (car ES)))))
  )
  (princ "\nInvalid selection. Try again ...")
  (setq ES (entsel "\nSelect a circle: "))
)
```

Tóm tắt

Sau đây là một vài hướng dẫn giúp ta tạo ra các chương trình tốt:

1. **Phác thảo các nét chính của chương trình**
   Trước khi bắt tay vào viết các mã lệnh **AutoLISP**, ta nên phác thảo các nhiệm vụ của chương trình, mô tả các hàm sẽ được viết.
2. **Nên có các sách tham khảo**
   Khi viết chương trình ta nên tham khảo sách để viết các lệnh cho đúng, giảm bớt thời gian tìm và sửa lỗi.
3. **Viết nhiều hàm nhỏ thay cho một chương trình lớn**
   Khi chia một chương trình lớn thành nhiều chương trình nhỏ, việc lập trình trở nên đơn giản hơn. Các chương trình nhỏ dễ viết hơn các chương trình lớn. Các chương trình nhỏ dễ kiểm tra hơn trước khi được kết nối vào chương trình lớn. Các chương trình nhỏ dễ sửa lỗi hơn vì phạm vi tìm kiếm lỗi nhỏ hơn.
4. **Viết chương trình sao cho dễ đọc**
   Chương trình sẽ dễ đọc hơn nếu ta sử dụng phương pháp canh lề để biểu diễn các biểu thức lồng nhau, hoặc dùng cách viết chữ hoa chữ thường để tạo sự tương phản giữa các lệnh và các biến. Ngoài ra chương trình cần được chú thích đầy đủ. Điều này giúp ta tiết kiệm thời gian tìm và sửa lỗi.
5. **Nếu được, không khai báo các biến cục bộ cho đến khi chương trình chạy hoàn chỉnh**
   Trong thời gian viết và chạy thử chương trình, ta nên khai báo các biến là toàn cục để dễ theo dõi giá trị của chúng nếu có lỗi xuất hiện. Sau khi hoàn tất chương trình và đưa vào sử dụng, ta sẽ sửa lại các biến là cục bộ.
6. **Hiểu ý nghĩa các thông báo lỗi của AutoLISP**
   Hiểu các thông báo lỗi của **AutoLISP** giúp ta dễ tìm ra vị trí gây lỗi. Các thông báo lỗi thường gặp là *"malformed list"* hoặc *"bad argument type"*.
7. **Sử dụng các phương pháp quản lý lỗi phù hợp**
   Khi đang viết chương trình, ta thường sử dụng cách quản lý lỗi mặc định của **AutoLISP**. Khi chương trình đã hoàn tất, ta nên bổ sung các

phương pháp quản lý lỗi phù hợp. Hoặc kiểm tra ngay lúc người sử dụng nhập dữ liệu, hoặc sau khi người sử dụng nhập dữ liệu thì dùng vòng lặp để kiểm tra và yêu cầu nhập lại nếu cần.

19.7 Bài tập

1. Viết lại chương trình sau cho dễ đọc, và sửa lỗi cho chương trình này.

```
;Tên file: LP.LSP
;
;Mục đích: Chương trình cho phép người sử dụng chọn một đối tượng
;         để chuyển lớp bản vẽ của đối tượng này thành lớp bản vẽ
;         hiện hành.
(defun c:lp (/ nl) (prompt "\nSelect entity on target layer: ")(if (setq es
(entsel))(progn (setq nl (cdr (assoc 8 (entget (cdr es)))))(command "layer"
"s" nl ""))(princ "\nNo entities selected!")(princ))
```

2. Sửa lại các chương trình đã viết trong các chương 13, 14, 15, 16, bổ sung các kiến thức về viết và sửa lỗi chương trình được đề cập đến trong chương này

19.8 Lời giải

1.

```
;Tên file: LP.LSP
;
;Mục đích: Chương trình cho phép người sử dụng chọn một đối tượng
;         để chuyển lớp bản vẽ của đối tượng này thành lớp bản vẽ
;         hiện hành.
(defun C:LP (/ NL)
  (prompt "\nSelect entity on target layer: ")
  (if (setq ES (entsel))
    (progn
      (setq NL (cdr (assoc 8 (entget (car ES)))))
      (command "layer" "s" NL "")
    )
    (princ "\nNo entities selected!")
```

```
  )
  (princ)
)
;Kết thúc file: LP.LSP
```

Chương 20

GIỚI THIỆU VỀ MÔI TRƯỜNG LẬP TRÌNH Visual LISP

Nội dung chương

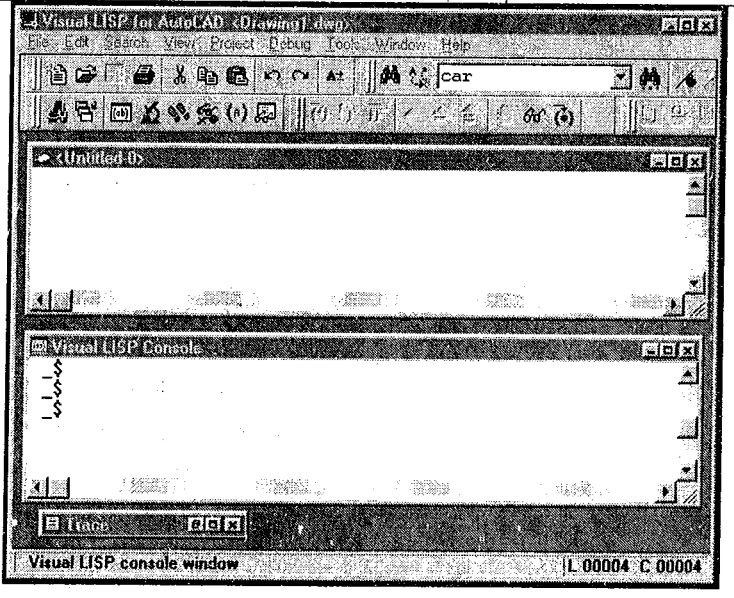
1. Viết, kiểm tra cú pháp và chạy các chương trình trong **Visual LISP**.
2. Các chức năng gỡ rối chương trình.
3. Tạo Project quản lý nhiều file.
4. Chức năng xem trước giao diện của các hộp thoại **DCL**.

Kể từ phiên bản **AutoCAD 2000** trở đi, ta có thể viết các chương trình **AutoLISP** trong môi trường lập trình **Visual LISP** (**VLISP**). Môi trường này cung cấp nhiều công cụ hỗ trợ cho việc viết và gỡ rối chương trình, nhất là đối với các chương trình phức tạp. Tuy nhiên, điều quan trọng vẫn là nắm vững cú pháp của ngôn ngữ lập trình **AutoLISP** như đã trình bày trong các chương trước của cuốn sách này.

20.1 Giới thiệu

Môi trường **Visual LISP for AutoCAD** (hình vẽ) có thể được mở ra bằng cách nhập lệnh hoặc gọi từ thanh menu:

Menu bar	**Nhập lệnh**
Tools>AutoLISP>Visual LISP Editor	*VLISP* hoặc *VLIDE*



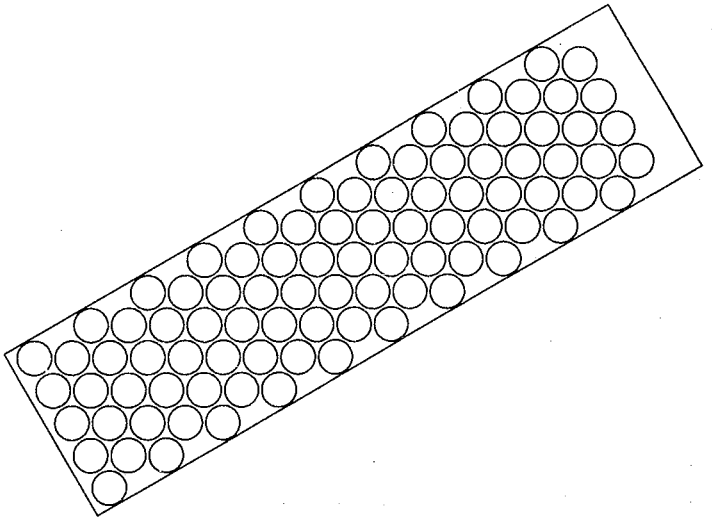
Môi trường này có 3 cửa sổ chính là:

- Các cửa sổ chương trình (ví dụ cửa sổ <Untitled-0>): dùng để mở và sửa đổi các file văn bản như .lsp, .dcl, ...
- Cửa sổ **Visual LISP Console** tương tự cửa sổ **Command:** của **AutoCAD**.
- Cửa sổ **Trace:** dùng để gỡ rối chương trình đang chạy.

Ta sẽ tìm hiểu cách sử dụng **VLISP** thông qua quá trình viết hàm **GPath** dùng để vẽ con đường rải sỏi trong vườn.

Ví dụ

Gpath vẽ một con đường rải sỏi như trên hình vẽ. Thông tin yêu cầu từ người dùng bao gồm: điểm đầu, điểm cuối, chiều rộng con đường, kích thước (bán kính) các viên sỏi và khoảng cách giữa chúng.



Đường rải sỏi vẽ bằng hàm GPath

20.2 Viết chương trình trong VLISP

Bắt đầu thực hiện chương trình theo trình tự sau:

1. Trong cửa sổ chương trình **Visual LISP**, chọn menu *File>New File...*
2. Nhập đoạn chương trình sau đây vào cửa sổ soạn thảo (là cửa sổ có tiêu đề *"<Untitled-0>"*).
3. Trên **File** menu chọn *Save As* để lưu chương trình thành file *gpmain.lsp*.

```
(defun GetPointInput ( )
  (alert
    "Function GetPointInput will get user drawing input"
  )
  T
) ;_ end of defun
```

```
(defun GetDialogInput ( )
 (alert
  "Function GetDialogInput will get user choices via a dialog"
 )
 T
) ;_ end of defun

(defun DrawOutline ( )
 (alert
  (strcat "This function will draw the outline of the polyline "
          "\n and return a polyline entity name/pointer."
  ) ;_ end of strcat
 ) ;_ end of alert
) ;_ end of defun

(defun C:GPath ( )
 (if (GetPointInput)
  (if   (GetDialogInput)
   (progn
       (setq PolylineName (DrawOutline))
       (princ "\nThe DrawOutline function returned <")
       (princ PolylineName)
       (princ ">")
       (Alert "Congratulations - your program is complete!")
   ) ; end of progn
   (princ "\nFunction cancelled.")
  ) ; end of if
  (princ "\nIncomplete information to draw a boundary.")
 ) ; end of if
 (princ)
) ; end of defun

(princ "\nType GPATH to draw a garden path.")
(princ)
```

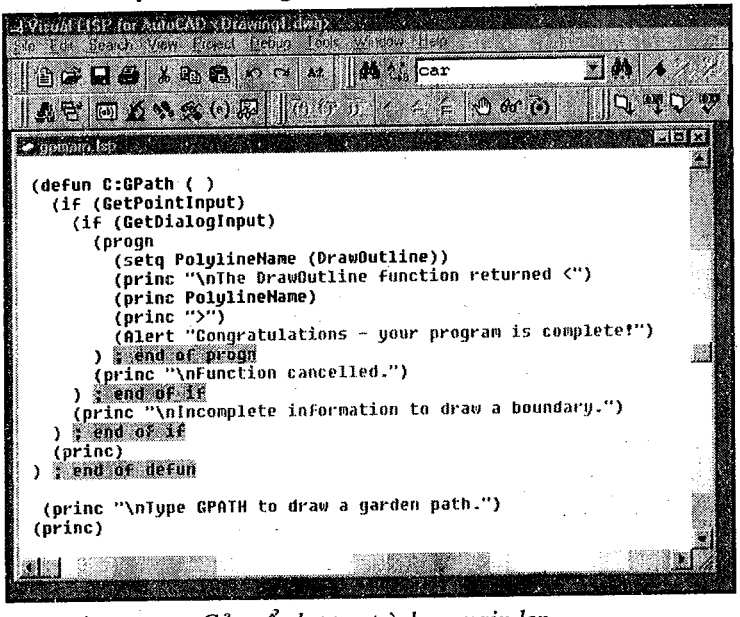
Màu các dòng chữ

Để giảm bớt lỗi khi nhập chương trình từ bàn phím, **VLISP** sẽ đổi màu tương ứng cho các dòng chữ trong chương trình **AutoLISP**. Nhờ đó, giả sử khi ta bỏ sót dấu nháy kết thúc một chuỗi ", thì các ký tự nhập vào sau đó vẫn tiếp tục có màu tím (là màu quy định của một chuỗi) chứ không đổi

Các quy định về màu sắc mặc định như sau:

Xanh dương	Các từ khóa của **AutoLISP** (defun, if, princ, ...).
Tím	Chuỗi ký tự trong dấu nháy kép " ".
Xanh lá cây	Số nguyên và số thực.
Màu tím trên nền xám	Các dòng chú thích.
Đỏ	Các dấu đóng ngoặc), mở ngoặc (.
Đen	Tất cả các ký hiệu còn lại (biến tự tạo , ...).

Giải thích sơ lược chương trình



Cửa sổ chương trình gpmain.lsp

- Chương trình chính **C:GPath** tạo ra một lệnh **Gpath** có thể gọi trực tiếp tại dòng nhắc lệnh *Command:*
- Hàm **GetPointInput** và **GetDialogInput** nhận thông tin yêu cầu từ người dùng. Hàm **DrawOutline** vẽ con đường rải sỏi. Các hàm này chỉ mới được viết các dòng thông báo trả về cho hàm chính, chưa có các lệnh cụ thể. Điều này cho phép kiểm tra từng phần chương trình (trong trường hợp này nhằm kiểm tra hàm chính *C:GPath*). Các hàm **GetPointInput** và **GetDialogInput** tạm thời trả về kết quả là **T**, tức là chạy hoàn tất.
- Nếu chương trình chạy gặp lỗi (ví dụ như người dùng nhấn ENTER thay vì dùng chuột chọn điểm trên màn hình), hàm **GetPointInput** trả về giá

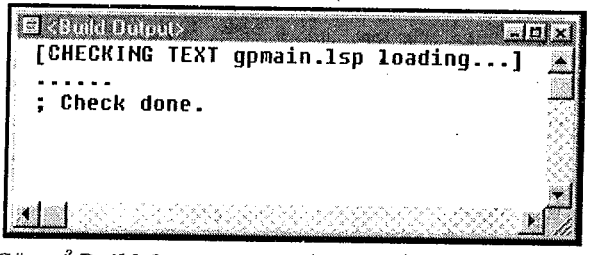
trị nil cho hàm chính **C:GPath**. Hàm này sẽ in dòng chữ *"Incomplete information to draw a boundary."*

Kiểm tra cú pháp của chương trình

Thực hiện chức năng kiểm tra cú pháp trước khi chạy chương trình sẽ giúp ta phát hiện các lỗi cú pháp thường gặp như thiếu các dấu ngoặc, các dấu nháy chuỗi...

Thực hiện theo trình tự:

1. Nhấn chuột vào thanh tiêu đề của cửa sổ chứa chương trình *gpmain.lsp* để bảo đảm đây là cửa sổ hiện hành.
2. Chọn menu **Tools**> *Check Text in Editor.*
3. Cửa sổ **Build Output** xuất hiện chứa kết quả kiểm tra cú pháp. Nếu **Visual LISP** không tìm thấy lỗi, nội dung hiện ra như trên hình vẽ.



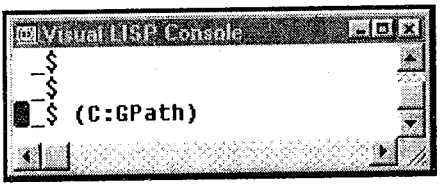
*Cửa sổ **Build Output** chứa kết quả kiểm tra cú pháp*

Chạy chương trình trong môi trường VISUAL LISP

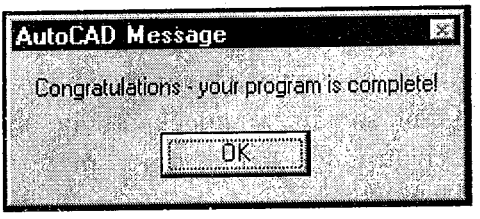
Sau khi đã kiểm tra cú pháp, ta cho chạy thử chương trình trong môi trường **VLISP** để dễ dàng theo dõi và gỡ rối chương trình khi gặp lỗi.

Thực hiện theo trình tự:

1. Nhấn chuột vào thanh tiêu đề của cửa sổ chứa chương trình gpmain.lsp để bảo đảm đây là cửa sổ hiện hành.
2. Chọn menu **Tools**> *Load Text in Editor.*
3. Tại cửa sổ **Visual LISP Console** nhập (C:Gpath). Ghi chú: Phải nhập đầy đủ cả các dấu ngoặc.



4. Nhấn ENTER để chạy chương trình, các thông báo sẽ lần lượt xuất hiện. Thông báo cuối cùng là "*Congratulations - your program is complete!*".

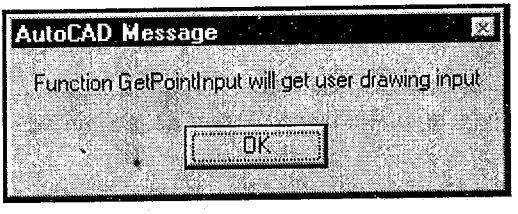


20.3 Các chức năng gỡ rối chương trình (debug)

20.3.1 Cửa sổ Visual LISP Console

Cửa sổ **Visual LISP Console** dùng để gọi thi hành các hàm **AutoLISP** (tương tự như cửa sổ **Command:** của **AutoCAD**). Ta có thể dùng nó để kiểm tra các hàm tự tạo trước khi gắn chúng vào chương trình chính.

Gọi thi hành hàm **GetPointInput** đã viết ở các ví dụ trên tại cửa sổ **Visual LISP Console** sẽ làm xuất hiện hộp thoại như sau:



Bổ sung thêm các lệnh cho hàm **GetPointInput** như sau:

1. Nhập các lệnh của hàm này trực tiếp tại cửa sổ **Visual LISP Console**:

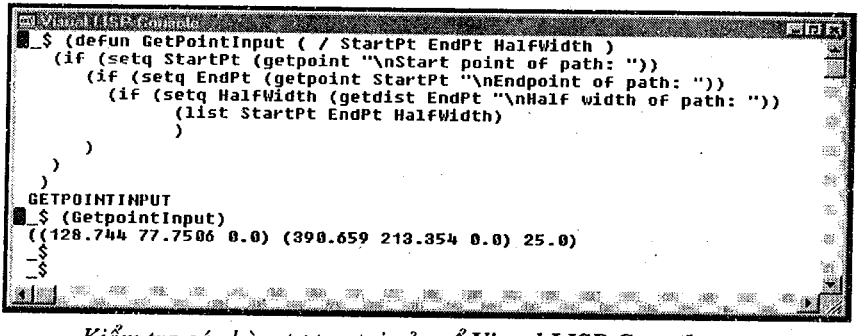
```
(defun GetPointInput ( / StartPt EndPt HalfWidth )
 (if (setq StartPt (getpoint "\nStart point of path: "))
```

```
        (if (setq EndPt (getpoint StartPt "\nEndpoint of path: "))
          (if (setq HalfWidth (getdist EndPt "\nHalf width of path: "))
              (list StartPt EndPt HalfWidth)
              )
          )
        )
      )
```

2. Di chuyển con trỏ đến nằm trong phạm vi các lệnh của hàm này rồi nhấn ENTER, hàm **GetPointInput** mới này sẽ thay thế cho hàm cũ.
3. Gọi thi hành hàm mới này bằng cách nhập **(GetPointInput)**↵
4. Chương trình sẽ yêu cầu người sử dụng nhập các điểm đầu, cuối và nửa chiều rộng. Sau đó trả về một danh sách tương tự như sau (hình vẽ):

((128.744 77.7506 0.0) (390.659 213.354 0.0) 25.0)

(128.744 77.7506 0.0)	toạ độ điểm đầu con đường
(390.659 213.354 0.0)	toạ độ điểm cuối con đường
25.0	chiều rộng con đường



*Kiểm tra các hàm tự tạo tại cửa sổ **Visual LISP Console***

Các giá trị do người dùng nhập vào sẽ được hàm **GetPointInput** đưa vào một danh sách để trả về cho chương trình chính là hàm **GPath**. Để thuận tiện xử lý trong chương trình chính, ta có thể dùng hàm **Cons** để xây dựng một danh sách phức hợp (hàm **cons** được trình bày ở chương 10, tập 1. Các mã DXF được liệt kê trong phụ lục 1).

Hàm **GetPointInput** được cải tiến như sau:

```
(defun GetPointInput   (/ StartPt EndPt HalfWidth)
 (if (setq StartPt (getpoint "\nStart point of path: "))
  (if    (setq EndPt (getpoint StartPt "\nEndpoint of path: "))
   (if (setq HalfWidth (getdist EndPt "\nHalf width of path: "))
       (list
        (cons 10 StartPt)
        (cons 11 EndPt)
        (cons 40 (* HalfWidth 2.0))
        (cons 50 (angle StartPt EndPt))
        (cons 41 (distance StartPt EndPt))
       )
    )
  )
 )
)
```

Thực hiện theo trình tự:

1. Nhập các lệnh của hàm **GetPointInput** cải tiến trực tiếp tại cửa sổ **Visual LISP Console**.
2. Di chuyển con trỏ đến nằm trong phạm vi các lệnh của hàm này rồi nhấn ENTER, hàm **GetPointInput** mới này sẽ thay thế cho hàm cũ.
3. Gọi thi hành hàm mới này bằng cách nhập

   (setq PathData (GetPointInput))↵

   Danh sách trả về sẽ được chứa trong biến PATHDATA.

4. Nhập tên biến PATHDATA tại cửa sổ **Console** để xem giá trị của nó, có dạng một danh sách phức hợp:

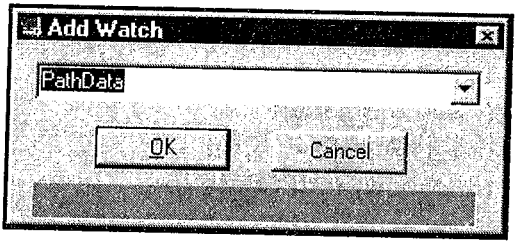
((10 168.585 88.8052 0.0) (11 361.885 197.877 0.0) (40 . 50.0)
(50 . 0.513726) (41 . 221.95))

20.3.2 Các cửa sổ Watch và Inspect

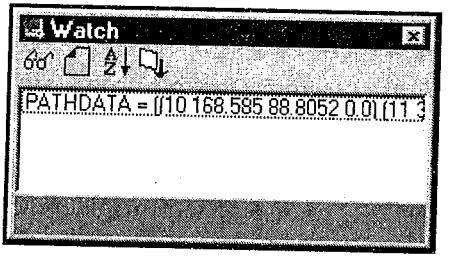
Ta có thể sử dụng cửa sổ **Watch** và cửa sổ **Inspect** để xem chi tiết hơn giá trị của các biến.

Thực hiện theo trình tự:

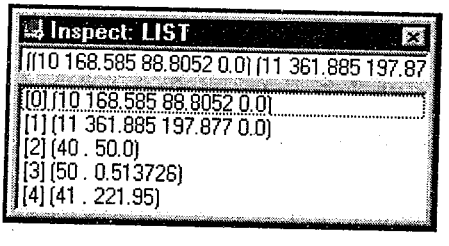
1. Chọn menu **Debug**> *Add Watch* để xuất hiện hộp thoại **Add Watch**.



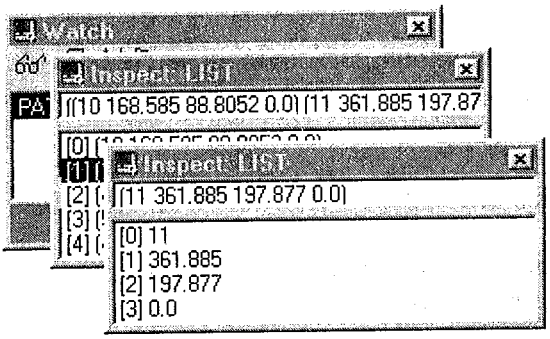
2. Nhập tên biến cần xem giá trị (ví dụ như biến PATHDATA). Nhấn nút *OK* để hiện cửa sổ **Watch** chứa giá trị của biến này (là một danh sách rất dài).



3. Nhắp kép vào tên biến PATHDATA trong cửa sổ **Watch** để mở cửa sổ **Inspect**. Cửa sổ này hiện các danh sách con trên nhiều dòng giúp ta quan sát dễ dàng hơn.



4. Nhắp kép vào dòng chứa danh sách có mã DXF là 11 sẽ hiện ra một cửa sổ **Inspect** khác tương ứng với danh sách này.



Sau khi đã kiểm tra và chạy thử hàm **GetPointInput** trong cửa số **Console,** ta chép hàm này vào thay thế cho hàm **GetPointInput** cũ trong file *gpmain.lsp.* Lệnh gọi hàm này trong hàm **C:GPath( )** được sửa lại như sau:

(if (GetPointInput) sửa thành (if (setq PathData (GetPointInput))

20.3.2 Các điểm dừng chương trình (Breakpoint)

Trong chương trình nguồn, ta có thể chèn các điểm dừng tại các vị trí ta muốn chương trình chạy đến đó sẽ dừng lại. Khi chương trình dừng lại, ta có thể thực hiện một số việc nhằm mục đích kiểm tra lỗi như sau:

- Cho chương trình chạy từng bước (từng dòng lệnh, từng hàm, từng biểu thức,...)
- Theo dõi giá trị hiện hành của các biến tại cửa sổ **Watch**.
- Thay đổi giá trị các biến, tác động đến các kết quả trung gian của chương trình đang chạy.
- Cho chương trình tiếp tục chạy đến điểm dừng kế tiếp, hoặc đến hết chương trình.

Ta sẽ gọi đến các chức năng này từ các nút lệnh trên thanh công cụ **Debug**.

Thanh công cụ ***Debug***

Dừng chương trình và sửa đổi giá trị cho biến được thực hiện như sau:

Chèn điểm dừng

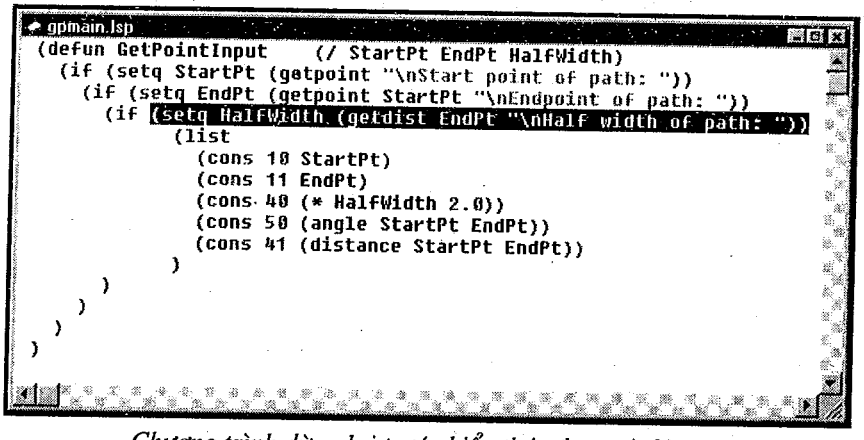
1. Trong cửa sổ chương trình *gpmain.lsp*, di chuyển con nháy (vị trí chèn ký tự) đến trước lệnh **(setq Halfwidth (getdist EndPt "\nHalf width of path: "))**

2. Nhấn nút *Toggle Breakpoint* để chèn một điểm dừng tại đây. 

3. Nhấn nút *Load Active Edit Window* để tải lại chương trình. 

4. Chạy hàm **(C:GPath)** trong cửa sổ **VLISP Console**.

Chương trình sẽ yêu cầu nhập vào 2 điểm đầu, cuối, sau đó dừng lại (hình vẽ) và tô đậm biểu thức **(setq Halfwidth (getdist EndPt "\nHalf width of path: "))**.



Chương trình dừng lại trước biểu thức được tô đậm

Chạy từng bước

1. Nhấn phím *Step Over*. 

Toàn bộ biểu thức **(setq Halfwidth (getdist EndPt "\nHalf width of path: "))** được thực hiện, chương trình yêu cầu nhập nửa chiều rộng con đường và gán cho biến HALFWIDTH.

2. Nhấn phím *Step Over*. 

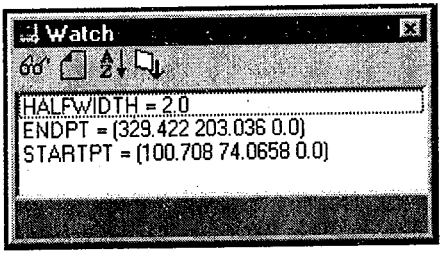
Biểu thức **(list (cons 10 StartPt) (cons .....)** được tô đậm.

3. Nhấn phím *Step Into*. 

Chương trình bước vào trong biểu thức list. Biểu thức **(cons 10 StartPt)** được tô đậm.

Theo dõi giá trị hiện hành của các biến

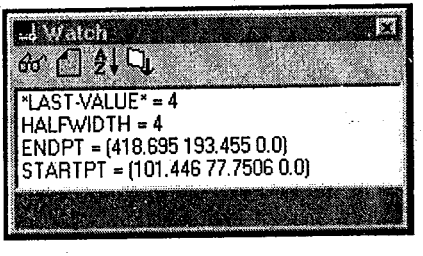
1. Hiện cửa sổ **Watch** bằng menu **View**>*Watch Window* hoặc bằng nút lệnh  . Xóa trắng cửa sổ bằng nút lệnh *Clear Window*  của cửa sổ này.
2. Lần lượt đưa các biến STARTPT, ENDPT, HALFWIDTH vào cửa sổ **Watch** để xem xét giá trị hiện hành của chúng (nhắp kép vào tên biến, rồi nhấn nút ).



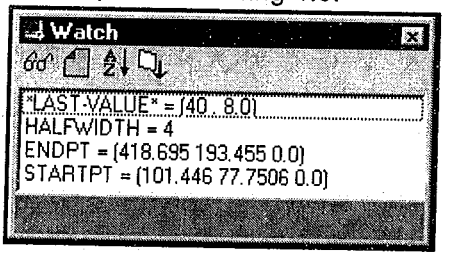
Thay đổi giá trị hiện hành của các biến

Giả sử ta cần sửa lại giá trị của biến HALFWIDTH bằng 4.

1. Thi hành biểu thức **(setq halfwidth 4.0)** tại cửa sổ **VLISP Console**. Biến HALFWIDTH sẽ được gán lại bằng 4.0. Ta có thể nhận thấy điều này trong cửa sổ **Watch**. Ta sẽ kiểm tra xem giá trị mới 4.0 này có được nhận biết tại biểu thức **(cons 40 (* HalfWidth 2.0))** hay không?
2. Chọn menu **Debug**>*Watch Last Evaluation*. Cửa sổ **Watch** sẽ được đưa thêm vào biến *Last-Value*. Đây là một biến toàn cục của **VLISP**, chứa giá trị của biểu thức được tính toán cuối cùng (tương tự như biến @ chứa toạ độ điểm được chọn cuối cùng trên bản vẽ).



3. Tiếp tục chạy chương trình từng bước (nhấn nút *Step Over* hoặc *Step Into*) cho đến khi biểu thức **(cons 40 (* HalfWidth 2.0))** được tính. Khi đó, biến *Last-Value* trong cửa sổ **Watch** có giá trị bằng (40. 8.0). Nghĩa là biến HALFWIDTH đã được hiểu là bằng 4.0.



Chạy hoàn tất chương trình

1. Nhấn nút *Continue* để chương trình chạy liên tục cho đến hết  .
2. Chọn menu **Debug**>*Clear All Breakpoints* để xóa tất cả các điểm dừng.

20.4 Tạo project bao gồm nhiều file chương trình

Để chương trình dễ đọc, dễ sửa chữa, ta chia chương trình thành nhiều hàm. Ví dụ như file *gpath.lsp* chứa 4 hàm: **C:GPath**, **GetPointInput**, **GetDialogInput**, **DrawOutline**. Tuy nhiên, khi số lượng hàm quá lớn, ta cũng sẽ gặp nhiều khó khăn khi sửa chữa các câu lệnh. Do đó, ta nên chia chương trình thành nhiều file nhỏ. Mỗi file chỉ chứa một số hàm có liên quan mật thiết với nhau.

Ví dụ:

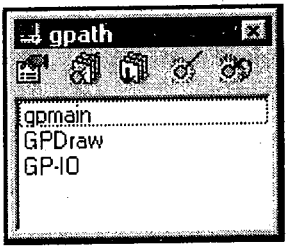
Tạo thư mục GPATH. Chia file *GPMAIN.LSP* thành 3 file nhỏ hơn chứa trong thư mục này:

GP-IO.LSP	Chứa các hàm nhập xuất của chương trình **(GetPointInput, GetDialogInput, ...)**.
GPDRAW.LSP	Chứa các hàm vẽ đường rải sỏi (**DrawOutline**, ...).
GPMAIN.LSP	Chứa hàm chương trình chính (**C:GPath**).

Ta sẽ tạo ra một **Project** chứa các file này. Khi đó, ta chỉ cần mở *Project* này trong **VLISP**, chứ không cần mở tất cả các file.

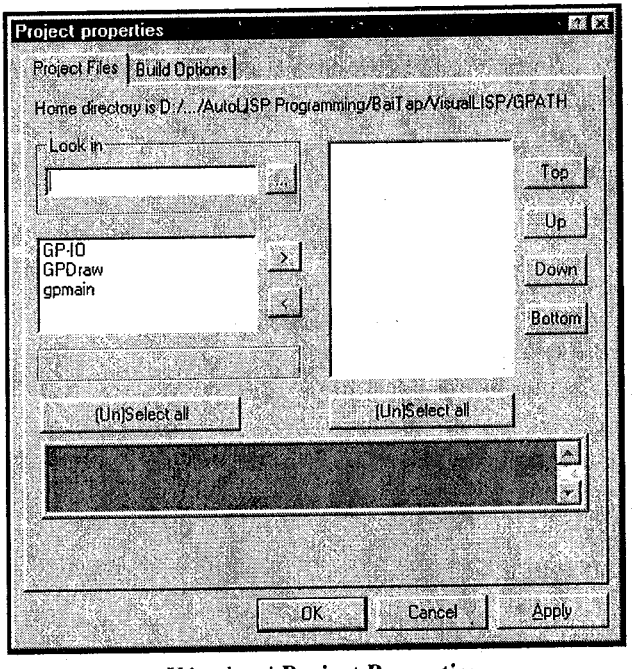
1. Chọn menu **Project**>*New Project* trên thanh menu.

2. Lưu *Project* này thành file *GPath.prj* trong thư mục GPATH. Xuất hiện hộp thoại **Project Properties** (xem hình 20.8).
3. Đưa các file *GP-IO.LSP, GPDRAW.LSP, GPMAIN.LSP* vào *Project:* chọn lần lượt từng file ở danh sách bên trái, và nhấn nút  để đưa sang danh sách bên phải.
4. Nhấn *OK*. Cửa sổ **Project** mang tên *gpath* xuất hiện, liệt kê các file của *Project* này.



Nhắp kép vào tên bất kỳ file nào trong cửa sổ này, thì file đó sẽ được mở ra trên màn hình.

5. Để chạy chương trình, ta nhấn nút lệnh *The Load Source Files* của cửa sổ *project*  . Sau đó, nhập (**C:GPath**) tại cửa sổ **Console**.



Hộp thoại ***Project Properties***

20.5 Xem trước các hộp thoại DCL

Ta có thể nhận thông tin từ người sử dụng thông qua các hộp thoại. Chương 17, tập 2 có trình bày cách tạo các hộp thoại bằng các file DCL. Trong môi trường **VLISP**, ta có thể xem trước giao diện của hộp thoại trước khi gán các chức năng của chương trình cho các hộp thoại này.

Ví dụ:

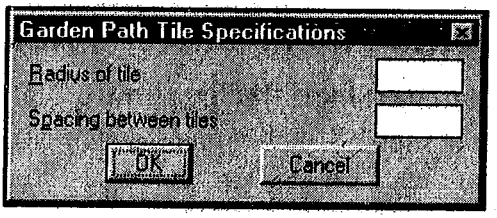
Trong ví dụ gpath đang xét, ta cần thông tin để vẽ các viên sỏi như: bán kính, khoảng cách giữa các viên sỏi...

1. Từ **File** menu chọn *New File...* để tạo một file mới.
2. Nhập các lệnh mô tả hộp thoại rồi lưu thành file *GP-DIA.DCL* trong thư mục GPATH đang xét.

```
gp_mainDialog : dialog {
label = "Garden Path Tile Specifications";
: edit_box {      // defines the Radius of Tile edit box
  label = "&Radius of tile";
  key = "gp_trad";
  edit_width = 6;
}
: edit_box {      // defines the Spacing Between Tiles edit box
  label = "S&pacing between tiles";
  key = "gp_spac";
  edit_width = 6;

}

: row {          // defines the OK/Cancel button row
  : spacer { width = 1; }
  : button {     // defines the OK button
    label = "OK";
    is_default = true;
    key = "accept";
    width = 8;
    fixed_width = true;
  }
  : button {     // defines the Cancel button
    label = "Cancel";
```

```
    is_cancel = true;
    key = "cancel";
    width = 8;
    fixed_width = true;
   }
   : spacer { width = 1;}
  }
  }
```

3. Từ **Tools** menu chọn *Interface Tools Preview DCL in Editor* để xem trước giao diện của hộp thoại (hình vẽ).



Hộp thoại nhập bán kính và khoảng cách giữa các viên sỏi

4. Sửa lại hàm **GetDialogInput** như sau để hiện hộp thoại vừa được tạo nên:

```
(defun GetDialogInput ( )
 (setq DCL_ID (load_dialog "GP-DIA.DCL"))
  (new_dialog."gp_mainDialog" DCL_ID)
  (start_dialog)
) ;_ end of defun
```

Trong chương này, ta đã làm quen với môi trường **VLISP** thông qua việc viết chương trình *GPath*. Ở đây chỉ trình bày minh hoạ một số hàm như C:GPMAIN, GETPOINTINPUT, GETDIALOGINPUT. Các hàm còn lại của chương trình chúng tôi không trình bày tại đây.

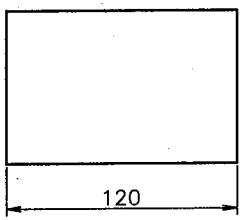
CHƯƠNG TRÌNH ỨNG DỤNG

Trong chương này chúng tôi giới thiệu các chương trình ứng dụng lập trình bằng **AutoLISP** và **Visual LISP** để thêm lệnh cho **AutoCAD** và các chương trình thiết kế tự động.

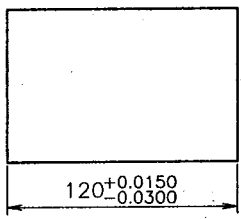
21.1 Lập chương trình thêm lệnh cho AutoCAD

21.1.1 Thêm dung sai vào kích thước sẵn có

Trong mục này ta tạo các chương trình có tên DDIMTOLR.LSP và DDIMTOLR.DCL để thêm dung sai vào kích thước sẵn có. Chiều cao dung sai phụ thuộc vào biến DIMTFAC.



a) Trước Ddimtolr



b) Sau Ddimtolr

File DIMTOLR.LSP

```
;Tên file:DDIMTOLR.LSP
;  Chương trình thêm giá trị dung sai vào kích thước chưa bị phá vỡ sẵn có. Ngoài ra
;bạn có thể bắt buộc chữ số kích thước
;nằm trong hai đường gióng
 (defun tolr_err (MSG)
  (setq *error* OLDERR)
  (if (not TOLRERR)
    (princ (strcat "\nDDimtolr error: " MSG))
    (princ TOLRERR)
  ); Đóng if
  (if OLDTOL (setvar "DIMTOL" OLDTOL))
  (if OLDLIM (setvar "DIMLIM" OLDLIM))
  (if OLDTIX (setvar "DIMTIX" OLDTIX))
```

```
  (if OLDCMDE (setvar "CMDECHO" OLDCMDE))
  (princ)
);Đóng defun
;; -------------------------------------------------------
;; Dòng nhắc để người sử dụng chọn kích thước
(defun tolruser ()
  (setq I 1)
  (while (= I 1)
    (setq ENTPICK (car (entsel "\nSelect a DIMENSION: ")))
    (while (= ENTPICK nil)
      (prompt "Nothing picked, try again! ")
      (setq ENTPICK (car (entsel "\nSelect a DIMENSION: ")))
    ); Đóng while
    (setq ETTYPE (cdr (assoc 0 (entget entpick))))
    (if (= ETTYPE "DIMENSION")
      (setq I 2)
      (prompt "Selection was not a DIMENSION! ")
    );Đóng if
  ); Đóng while
  (princ)
); Đóng defun tolruser
; -------------------------------------------------------
; Gán các biến được sử dụng trên LISP
(defun tolrset ()
  (if (= OLDTP nil)
    (setq OLDTP 0)
  );if
  (if (= Oldtm nil)
    (setq OLDTM 0)
  ); Đóng if
  (setq NEWTM OLDTM)
  (setq NEWTP OLDTP)
  (if (and (/= OLDTOL 1) (/= OLDLIM 1))
    (progn
      (setq OLDNONE 1)
      (setq TTOLR 1)
    ); Đóng progn
    (setq OLDNONE nil)
  ); Đóng if
  (if (= OLDTOL 1)
    (setq TTOLR 2)
  ); Đóng if
  (if (= OLDLIM 1)
    (setq TTOLR 3)
  ); Đóng if
  (if (= OLDTIX 1)
```

```
    (progn
      (setq OLDTIXYES 1)
      (setq OLDTIXNO nil)
      (setq TTIX 1)
    ); Đóng progn
    (progn
      (setq OLDTIXYES nil)
      (setq OLDTIXNO 1)
      (setq TTIX 2)
    ); Đóng progn
  ); Đóng if
  (princ)
); Đóng defun tolrset
; ---------------------------------------------------
;Gọi file DDIMTOLR.DCL
(defun tolr_dialog ()
  (setq DCLYES nil)
  (setq DCL_ID (load_dialog "DDIMTOLR.DCL"))
  (if (not (new_dialog "dim_tolr" DCL_ID)) (exit))
  (if (= OLDNONE 1)
    (progn
      (set_tile "tolr_none" "1")
      (mode_tile "tolr_tp" 1)
      (mode_tile "tolr_tm" 1)
    ); Đóng progn
  ); Đóng if
  (if (= OLDTOL 1)
    (progn
      (set_tile "tolr_tol" "1")
    ); Đóng progn
  ); Đóng if
  (if (= OLDLIM 1)
    (progn
      (set_tile "tolr_lim" "1")
    ); Đóng progn
  ); Đóng if
  (set_tile "tolr_tp" (strcat (rtos OLDTP)))
  (set_tile "tolr_tm" (strcat (rtos OLDTM)))
  (if (= OLDTIXYES 1)
    (set_tile "tix_yes" "1")
  ); Đóng if
  (if (= OLDTIXNO 1)
    (set_tile "tix_no" "1")
  ); Đóng if
  (action_tile "tolr_none"
    (strcat
```

```
      "(progn (mode_tile \"tolr_tp\" 1)"
      "(mode_tile \"tolr_tm\" 1)"
      "(setq TTOLR 1))"
     ); Đóng strcat
    ); Đóng action_tile
    (action_tile "tolr_tol"
     (strcat
      "(progn (mode_tile \"tolr_tp\" 0)"
      "(mode_tile \"tolr_tm\" 0)"
      "(setq TTOLR 2))"
     ); Đóng strcat
    ); Đóng action_tile
    (action_tile "tolr_lim"
     (strcat
      "(progn (mode_tile \"tolr_tp\" 0)"
      "(mode_tile \"tolr_tm\" 0)"
      "(setq TTOLR 3))"
     ); Đóng strcat
    ); Đóng action_tile
    (action_tile "tix_yes" "(setq TTIX 1)" )
    (action_tile "tix_no" "(setq TTIX 2)" )
    (action_tile "accept"
     (strcat
      "(progn (setq NEWTP (atof (get_tile \"tolr_tp\")))"
      "(setq NEWTM (atof (get_tile \"tolr_tm\")))"
      "(done_dialog 1))"
     ); Đóng strcat
    ); Đóng action_tile
    (if (equal (start_dialog) 1)
     (setq DCLYES 1)
    ); Đóng if
    (unload_dialog dcl_id)
    (princ)
); Đóng defun tolr_dialog
; ---------------------------------------------------
; Chương trình chính của LISP
(defun tolrwork ()
 (if (= TTOLR 1)
  (progn
   (setvar "DIMTOL" 0)
   (setvar "DIMLIM" 0)
  ); Đóng progn
 ); Đóng if
 (if (= TTOLR 2)
  (progn
   (setvar "DIMTOL" 1)
```

```
    (setvar "DIMLIM" 0)
   ); Đóng progn
  ); Đóng if
  (if (= TTOLR 3)
   (progn
    (setvar "DIMTOL" 0)
    (setvar "DIMLIM" 1)
   ); Đóng progn
  ); Đóng if
  (setvar "DIMTP" NEWTP)
  (setvar "DIMTM" NEWTM)
  (if (= TTIX 1)
   (setvar "DIMTIX" 1)
   (setvar "DIMTIX" 0)
  ); Đóng if
  (command "DIM1" "UPDATE" ENTPICK "")
); Đóng defun tolrwork
;; -------------------------------------------------------
;; Phần chính của chương trình LISP
(prompt "Loading.....")
(defun C:DDIMTOLR ()
  (setq OLDERR *error*
    *error* TOLR_ERR
    OLDTOL nil
    OLDLIM nil
    OLDTIX nil
    OLDHIGH nil
    OLDCMDE nil
    TOLRERR nil
  ); Đóng setq
  (tolruser)
  (setq OLDHIGH (getvar "HIGHLIGHT")
    OLDCMDE (getvar "CMDECHO")
    OLDTP (getvar "dimtp")
    OLDTM (getvar "dimtm")
    OLDLIM (getvar "dimlim")
    OLDTOL (getvar "dimtol")
    OLDTIX (getvar "dimtix")
  ); Đóng setq
  (tolrset)
  (tolr_dialog)
  (if (= DCLYES 1)
   (progn
    (setvar "HIGHLIGHT" 0)
    (setvar "CMDECHO" 0)
    (tolrwork)
```

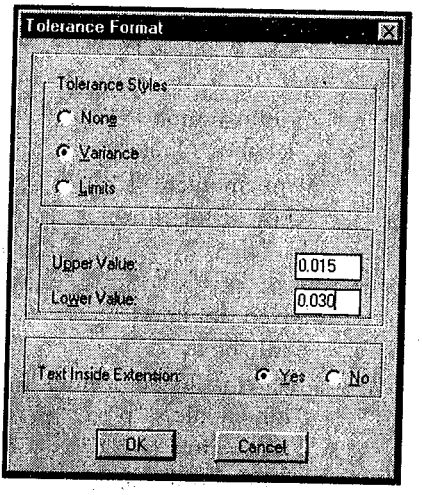
```
  ); Đóng progn
 ); Đóng if
 (setvar "HIGHLIGHT" OLDHIGH)
 (setvar "CMDECHO" OLDCMDE)
 (setvar "DIMTOL" OLDTOL)
 (setvar "DIMLIM" OLDLIM)
 (setvar "DIMTIX" OLDTIX)
 (setq *error* OLDERR)
 (princ)
); Đóng defun C:DDIMTOLR
(prompt "Done!")
(prompt "\nType DDIMTOLR to run.")
(princ); Kết thúc  DDIMTOLR.LSP
```

File DIMTOLR.DCL

```
//DDIMTOLR.DCL
dim_tolr : dialog {
 label = "Tolerance Format";
 : boxed_column {
    : column {
     : boxed_radio_column {
       label = "Tolerance Styles";
       : radio_button {
         label = "None";
         mnemonic = "e";
         key = "tolr_none";
       }
       : radio_button {
         label = "Variance";
         mnemonic = "V";
         key = "tolr_tol";
       }
       : radio_button {
         label = "Limits";
         mnemonic = "L";
         key = "tolr_lim";
       }
     }
     : boxed_column {
       : edit_box {
         label = "Upper Value: ";
         mnemonic = "p";
         key = "tolr_tp";
         edit_width = 6;
       }
```

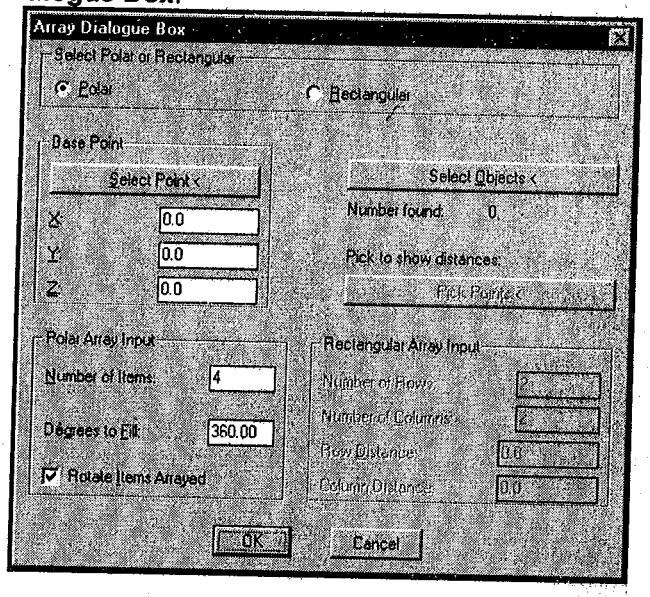
```
        : edit_box {
          label = "Lower Value: ";
          mnemonic = "w";
          key = "tolr_tm";
          edit_width = 6;
        }
      }
    }
  }
  : boxed_row {
    : text {
      label = "Text Inside Extension:";
    }
    : radio_row {
      : radio_button {
        label = "Yes";
        mnemonic = "Y";
        key = "tix_yes";
      }
      : radio_button {
        label = "No";
        mnemonic = "N";
        key = "tix_no";
      }
    }
  }
  : text {
    label = " ";
  }
  ok_cancel;
} // Kết thúc DDIMTOLR.DCL
```

Sau khi gọi chương trình ta nhập vào dòng lệnh **Ddimtolr**, đầu tiên ta chọn kích thước cần thêm dung sai, sau đó hộp thoại **Tolerance Format** sẽ xuất hiện. Ta nhập các số liệu như trên hộp thoại.



21.1.2 Thay đổi các dòng nhắc lệnh Array bằng hộp thoại

Trong **AutoCAD** khi thực hiện lệnh **Array** ta nhập các dữ liệu vào các dòng nhắc lệnh. Trong chương trình này thay các dòng nhắc lệnh bởi hộp thoại **Array Dialogue Box.**



Ta cần phải tạo hai file DDARRAY.LSP và DDARRAY.DCL.

File DDARRAY.LSP

```
;Tên file DDARRAY.LSP
;File .dcl kèm theo là DDARRAY.DCL.
;Thiết lập hàm defaults.
(defun defaults ()
  (set_tile "x_pt" x_pt)
  (set_tile "y_pt" y_pt)
  (set_tile "z_pt" z_pt)
  (set_tile "how_many"
    (if (/= selection_set nil)
      (rtos (sslength selection_set) 2 0)
    "0" )
  )
  (setq n_temp (atoi n_items))
  (setq n_items (rtos n_temp 2 0))
  (set_tile "n_items" n_items)
  (set_tile "fill_deg" fill_deg)
  (if (= 0 rot_item)
    (set_tile "rot_item" "0")
```

```
    (progn
      (set_tile "rot_item" "1")
      (setq rot_item 1)
    )
  )
  (if (= 0 pol_rect)
    (set_tile "rect" "1")
    ()
  )
  (if (= 1 pol_rect)
    (set_tile "polr" "1")
    ()
  )
  (setq r_temp (atoi rows))
  (setq rows (rtos r_temp 2 0))
  (set_tile "rows" rows)
  (setq c_temp (atoi cols))
  (setq cols (rtos c_temp 2 0))
  (set_tile "cols" cols)
  (set_tile "r_dist" r_dist)
  (set_tile "c_dist" c_dist)
)
;; Kết thúc hàm defuults.
;; Tạo hàm p_switch.
(defun p_switch ()
  (setq pol_rect 1)
  (mode_tile "pick_pt" 0)
  (mode_tile "pick_xy" 1)
  (mode_tile "x_pt" 0)
  (mode_tile "y_pt" 0)
  (mode_tile "z_pt" 0)
  (mode_tile "n_items" 0)
  (mode_tile "fill_deg" 0)
  (mode_tile "rot_item" 0)
  (mode_tile "rows" 1)
  (mode_tile "cols" 1)
  (mode_tile "r_dist" 1)
  (mode_tile "c_dist" 1)
)
;; Kết thúc hàm p_switch.
;; Tạo hàm r_switch.
(defun r_switch ()
  (setq pol_rect 0)
  (mode_tile "pick_pt" 1)
  (mode_tile "pick_xy" 0)
  (mode_tile "x_pt" 1)
```

```
  (mode_tile "y_pt" 1)
  (mode_tile "z_pt" 1)
  (mode_tile "n_items" 1)
  (mode_tile "fill_deg" 1)
  (mode_tile "rot_item" 1)
  (mode_tile "rows" 0)
  (mode_tile "cols" 0)
  (mode_tile "r_dist" 0)
  (mode_tile "c_dist" 0)
)
;Kết thúc hàm r_switch.
;Tạo hàm run_it.
(defun run_it ()
  (if (< (setq dcl_id (load_dialog "ddarray.dcl")) 0) (exit))
  (setq x_pt "0.0")          ;Tọa độ điểm chuẩn đầu tiên
  (setq y_pt "0.0")          ;
  (setq z_pt "0.0")          ; x, y, và z = 0.
  (setq n_items "4")         ;Polar array với 4 items.
  (setq fill_deg "360.00")   ;polar array với  360 degress.
  (setq rot_item 1)          ;Quay các đối tượng khi polar array.
  (setq rows "2")            ;2 rows cho rectangular array.
  (setq cols "2")            ;2 columns cho rectangular array.
  (setq r_dist "0.0")        ; row distance = 0
  (setq c_dist "0.0")        ; column distance = 0
  (setq pol_rect 1)          ; Bắt đầu dialog là polar array.
  (setq selection_set nil)
  (setq what_next 5)
  (while (< 2 what_next)
    (if (not (new_dialog "ddarray" dcl_id)) (exit))
    (defaults)
    (if (= 1 pol_rect);
(p_switch)
      (r_switch)
    )                          ;Kết thúc "if".
    (action_tile "pick_pt"    "(done_dialog 4)")
    (action_tile "sel_objs"   "(done_dialog 3)")
    (action_tile "pick_xy"    "(done_dialog 6)")
    (action_tile "x_pt"       "(setq x_pt $value)")
    (action_tile "y_pt"       "(setq y_pt $value)")
    (action_tile "z_pt"       "(setq z_pt $value)")
    (action_tile "n_items"    "(setq n_items $value)")
    (action_tile "fill_deg"   "(setq fill_deg $value)")
    (action_tile "rot_item"   "(setq rot_item (atoi $value))")
    (action_tile "rows"       "(setq rows $value)")
    (action_tile "cols"       "(setq cols $value)")
    (action_tile "r_dist"     "(setq r_dist $value)")
```

```
    (action_tile "c_dist"     "(setq c_dist $value)")
    (action_tile "polr"       "(p_switch)")
    (action_tile "rect"       "(r_switch)")
    (setq what_next (start_dialog))
  (cond                              ; Quyết định làm gì tiếp theo...
                                     ; Nếu dùng phím chọn để chọn đối tượng...
     ((= what_next 3)
      (setq selection_set (ssget))
      (setq ssflag 1)
     )                               ; Nếu chọn để định x,y row và column distances...
     ((= what_next 6)
      (initget 1)
      (setq pick_1st (getpoint "Pick first point: "))
      (setq pick_2nd (getcorner pick_1st "Pick second point: "))
      (setq r_dist (rtos (- (cadr pick_2nd) (cadr pick_1st)) 2 4))
      (setq c_dist (rtos (- (car pick_2nd) (car pick_1st)) 2 4))
     )    ;Nếu base point được chọn...
     ((= what_next 4)
      (initget 1)
      (setq pick_pt (getpoint "Polar array base point: "))
      (setq x_pt (rtos (car pick_pt) 2 4))
      (setq y_pt (rtos (cadr pick_pt) 2 4))
      (setq z_pt (rtos (caddr pick_pt) 2 4))
     )
   )                                 ;Kết thúc "cond"
  )                                  ;Kết thúc vòng lặp while.
  (action_tile "accept"    "(done_dialog)")
  (start_dialog)
  (unload_dialog dcl_id)
);Kết thúc hàm run_it
;Thiết lập MAIN FUNCTION
(defun C:DDARRAY (/ x_pt y_pt z_pt n_items fill_deg rot_item rows cols
   r_dist c_dist pol_rect selection_set what_next
  how_many pick_pt n_temp r_temp c_temp y_or_n )
  (setq scmde (getvar "cmdecho"))
  (setvar "cmdecho" 0)
  (run_it)
 (defaults ;Sau khi hộp thoại xuất hiện thực hiện hàm default
  ;để dữ liệu nhập rows, cols,  _items ... là các số nguyên.
  ; Khai báo IF / AND để cho phép thực hiện lệnh Array
  ;
 (if (and (= 1 what_next) (/= selection_set nil))
   (if (= 1 pol_rect)
    (progn
     (if (= 1 rot_item)
```

```
        (setq y_or_n "n")
       )
       (setq pick_pt (list (atof x_pt) (atof y_pt) (atof z_pt)))
       (command "array" "prev" "" "p" pick_pt n_items fill_deg y_or_n)
      )
      (progn
       (if (= "1" rows)
        (command "array" "prev" "" "r" rows cols c_dist)
       )
       (if (= "1" cols)
        (command "array" "prev" "" "r" rows cols r_dist)
       )
       (if (and (/= "1" rows) (/= "1" cols))
        (command "array" "prev" "" "r" rows cols r_dist c_dist)
       )
      )
    )                ; Kết thúc khai báo pol_rect if.
    ()               ; Kết thúc what_next = 1 'OK' và
  )                  ; kết thúc khai báo selection_set /= nil if
  (setvar "cmdecho" scmde)
  (princ)
);Kết thúc file DDARRAY.LSP
```

File DDARRAY.DCL

```
//DDARRAY.DCL  tạo hộp thoại cho lệnh Ddarray
ddarray : dialog {
 label = "Array Dialogue Box";
 : boxed_radio_row {
  label = "Select Polar or Rectangular";
  : radio_button {
    label = "Polar";
    mnemonic = "P";
    key = "polr";
  }
  : radio_button {
    label = "Rectangular";
    mnemonic = "R";
    key = "rect";
  }
}
spacer;
spacer;
: row {
  : boxed_column {
    label = "Base Point";
```

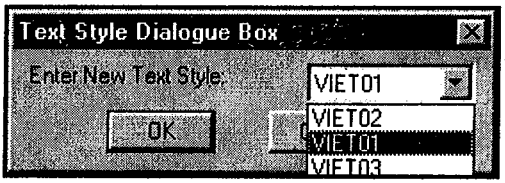
```
    : button {
      label = "Select Point <";
      mnemonic = "S";
      key = "pick_pt";
    }
    : edit_box {
      label = "X:";
      mnemonic = "X";
      key = "x_pt";
      edit_width = 10;
    }
    : edit_box {
      label = "Y:";
      mnemonic = "Y";
      key = "y_pt";
      edit_width = 10;
    }
    : edit_box {
      label = "Z:";
      mnemonic = "Z";
      key = "z_pt";
      edit_width = 10;
    }
  }
spacer_1;
  : column {
    spacer_1;
    : button {
      label = "Select Objects <";
      mnemonic = "O";
      key = "sel_objs";
    }
    : concatenation {
      : text_part {
        label = "Number found: ";
      }
      : text_part {
        key = "how_many";
        width = 5;
      }
    }
spacer_1;
    : text {
      label = "Pick to show distances:";
    }
    : button {
```

```
        label = "Pick Points <";
        mnemonic = "i";
        key = "pick_xy";
      }
    }
  }  /* end row assembly for base point & select objects */
  spacer;
  spacer;
  :row {
    : boxed_column {
      label = "Polar Array Input";
      : edit_box {
        label = "Number of Items:";
        mnemonic = "N";
        key = "n_items";
        edit_width = 6;
      }
      : edit_box {
        label = "Degrees to Fill:";
        mnemonic = "F";
        key = "fill_deg";
        edit_width = 6;
      }
      : toggle {
        label = "Rotate Items Arrayed";
        mnemonic = "I";
        key = "rot_item";
      }
    }
  spacer;
    : boxed_column {
      label = "Rectangular Array Input";
      : edit_box {
        label = "Number of Rows:";
        mnemonic = "m";
        key = "rows";
        edit_width = 8;
      }
      : edit_box {
        label = "Number of Columns:";
        mnemonic = "C";
        key = "cols";
        edit_width = 8;
      }
      : edit_box {
        label = "Row Distance:";
```

```
        mnemonic = "D";
        key = "r_dist";
        edit_width = 10;
      }
      : edit_box {
        label = "Column Distance:";
        mnemonic = "l";
        key = "c_dist";
        edit_width = 10;
      }
    }
  }
  spacer;
  spacer;
  ok_cancel;
}
```

21.1.3 Thay đổi kiểu chữ (text style) cho dòng text

Ta tạo hai file CHSTYLE.LSP và CHSTYLE.DCL để thay đổi kiểu chữ cho các dòng text trên bản vẽ. Khi thực hiện sẽ xuất hiện hộp thoại **Text Style Dialogue Box** cho phép ta chọn các kiểu chữ cần thay đổi.



File CHSTYLE.LSP

```
;Tên file  CHSTYLE.LSP
;Thay đổi Text Styles
(defun c:chstyle ()
  (SETVAR "CMDECHO" 0)
  (setq a (ssget))
  (setq num (load_dialog "chstyle"))
  (new_dialog "chstyle" num)
  (setq style_list (list (getvar "textstyle")))
  (setq test 1)
  (tblnext "style" 1)
  (while (/= test nil)
    (setq test (tblnext "STYLE"))
    (if (/= test nil)
      (progn
        (setq test (list (cdr (assoc 2 test))))
```

```
      (if (/= (nth 0 test) (getvar "textstyle"))
        (setq style_list (append style_list test))
      )
    )
  )
)
(action_tile "accept" "(chg_styl) (done_dialog)")
(start_list "tname")
(foreach n style_list (add_list n))
(end_list)
(start_dialog)
(unload_dialog num)
(setq n (sslength a))
(setq index 0)
(repeat n
  (setq b1 (entget (ssname a index)))
  (setq index (1+ index))
  (setq b (assoc 0 b1))
  (if (/= "MLINE" (cdr b))
    (progn
      (setq c (assoc 7 b1))
      (setq d (cons (car c) ns))
      (setq b2 (subst d c b1))
      (entmod b2)
    )
  )
)
(SETVAR "CMDECHO" 1)
(princ)
)
(defun chg_styl ()
  (setq ns (atoi (get_tile "tname")))
  (setq ns (nth ns style_list))
); Kết thúc file CHSTYLE.LSP
```

File CHSTYLE.DCL

```
//Tên file CHSTYLE.DCL
chstyle :dialog {
    label="Text Style Dialogue Box";
    :popup_list {
        label="Enter New Text Style:";
        key="tname";
        value= "0";
        edit_width=12;
        }
```

```
ok_cancel;
}
```

Ngôn ngữ lập trình AutoLISP

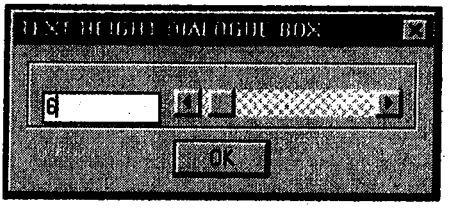
a) Kiểu chữ Viet01 với font VNI-Oxford

Ngôn ngữ lập trình AutoLISP

b) Kiểu chữ Viet02 với font VNI-Rush

21.1.4 Thay đổi chiều cao cho dòng chữ

Ta tạo hai file CHSIZE.LSP và CHSIZE.DCL để thay đổi chiều cao cho các dòng chữ trên bản vẽ. Khi thực hiện sẽ xuất hiện hộp thoại **Text Height Dialogue Box** cho phép ta nhập giá trị chiều cao mới hoặc thay đổi động bằng cách kéo thanh trượt ngang trên hộp thoại.



File CHSIZE.LSP

```
;Tên file  CHSIZE.LSP
;Thay đổi chiều cao text
(defun C:chsize (/ a n index b1 b c d b2)
  (SETVAR "CMDECHO" 0)
  (setq a (ssget))
  (setq num (load_dialog "CHSIZE"))
  (new_dialog "chsize" num)
  (mode_tile "slider_info" 2)
  (if (/= ht nil) (set_tile "slider_info" (rtos ht 2 2)))
  (action_tile "my_slider" "(slider_action $value $reason)")
  (action_tile "slider_info" "(ebox_action $value $reason)")
  (action_tile "accept" "(chg_ht) (done_dialog)")
  (start_dialog)
  (unload_dialog num)
  (setq n (sslength a))
  (setq index 0)
  (repeat n
    (setq b1 (entget (ssname a index)))
```

```
    (setq b (assoc 0 b1))
    (if (or (= "TEXT" (cdr b)) (= "MTEXT" (cdr b)))
      (progn
        (setq c (assoc 40 b1))
        (setq d (cons (car c) ht))
        (setq b2 (subst d c b1))
        (entmod b2)
      )
    )
  )
  (SETVAR "CMDECHO" 1)
  (princ)
)
(defun slider_action (val why)
  (if (or (= why 2) (= why 1))
    (set_tile "slider_info" val)
  )
)
(defun ebox_action (val why)
  (if (or (= why 2) (= why 1))
    (set_tile "my_slider" val)
  )
)
(defun chg_ht ()
  (setq ht1 (get_tile "slider_info"))
  (setq ht (atof ht1))
);Kết thúc CHSIZE.LSP
```

File CHSIZE.DCL

```
//Tên file  CHSIZE.DCL
dcl_settings : default_dcl_settings { audit_level = 3; }
chsize : dialog {
  label = "TEXT HEIGHT DIALOGUE BOX" ;
  : boxed_row {
    : edit_box {
      allow_accept = true ;
      value = "0" ;
      key = "slider_info" ;
      is_tab_stop = false ;
    }
    : slider {
      max_value = 100 ;
      min_value = 0 ;
      small_increment = 1 ;
      big_increment = 5;
```

```
        key = "my_slider" ;
        width = 20 ;
      }
    }
    ok_only ;
}
```

Ngôn ngữ lập trình AutoLISP

a) Chiều cao 12

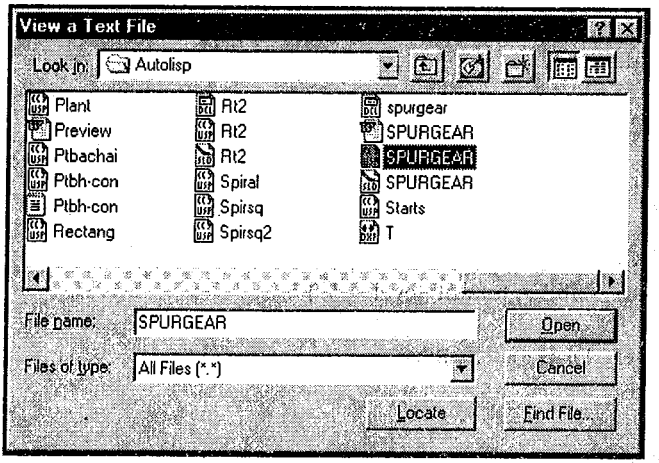
Ngôn ngữ lập trình AutoLISP

b) Chiều cao 6

21.1.5 Xem file văn bản trong AutoCAD

Tạo hai file VIEWTEXT.LSP và WIEWTEXT.DCL để xem các file văn bản trong **AutoCAD**.

Sau khi tải file VIEWTEXT.LSP, ta nhập tại dòng lệnh VIEWTEXT sẽ xuất hiện hộp thoại **View a Text file**. Trên hộp thoại này ta chọn file cần xem và nhấp *Open*.



Hộp thoại **Noi dung text file** sẽ xuất hiện. Trên hộp thoại này hiển thị file văn bản mà ta vừa chọn ở trên.

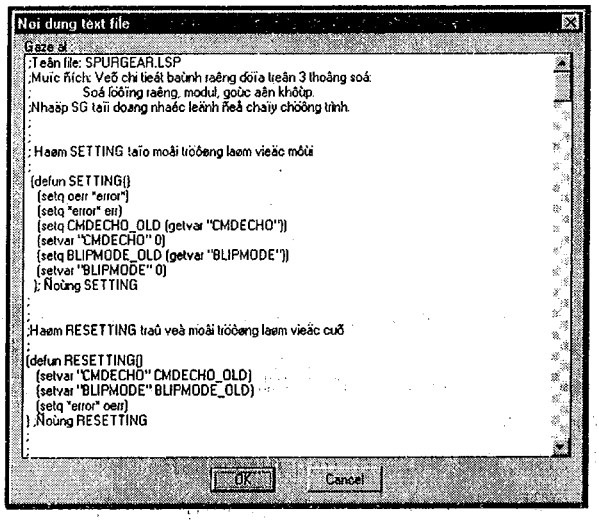
File VIEWTEXT.LSP

```
;Tên file VIEWTEXT.LSP Xem file văn bản
(defun C:VIEWTEXT (/ FNAME FSTRING)
  (setq FNAME (open (getfiled "View a Text File" "" "" 0) "r"))
  (setq ID (load_dialog "viewtext.dcl"))
  (if (not (new_dialog "gaze" ID))(exit))
  (start_list "gazebox")
```

```
  (while (setq FSTRING (read-line FNAME))(add_list FSTRING))
  (end_list)
  (close FNAME)
  (start_dialog)
  (term_dialog)
  (unload_dialog ID)
  (princ)
); Kết thúc file VIEWTEXT.LSP
```

File VIEWTEXT.DCL

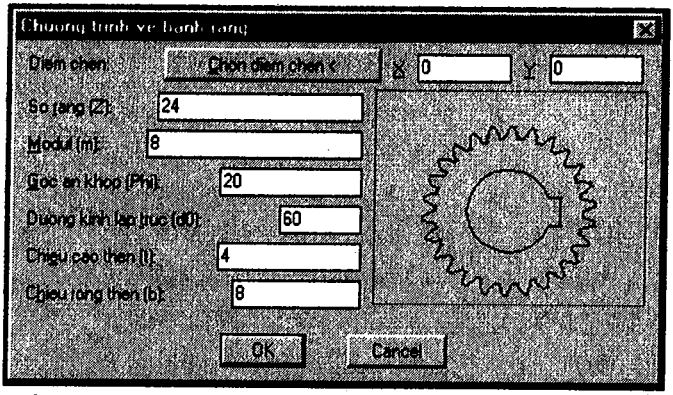
```
// VIEWTEXT.DCL
viewtext:   dialog {
      label = "Noi dung  text file";
      :list_box {
            key = "gazebox";
            label = "Gaze at :";
            width = 80;
            height = 25;
          }
      ok_cancel;
```



21.2 Chương trình vẽ biên dạng răng Spurgear.lsp

Chương trình SPURGEAR.LSP, SPURGEAR.DCL dùng để vẽ biên dạng răng thân khai dựa vào ba tham số: số răng z, modun m và góc ăn khớp α...

Sau khi tải file SPURGEAR.LSP ta nhập SG tại dòng nhắc lệnh và nhập các giá trị trên hộp thoại.

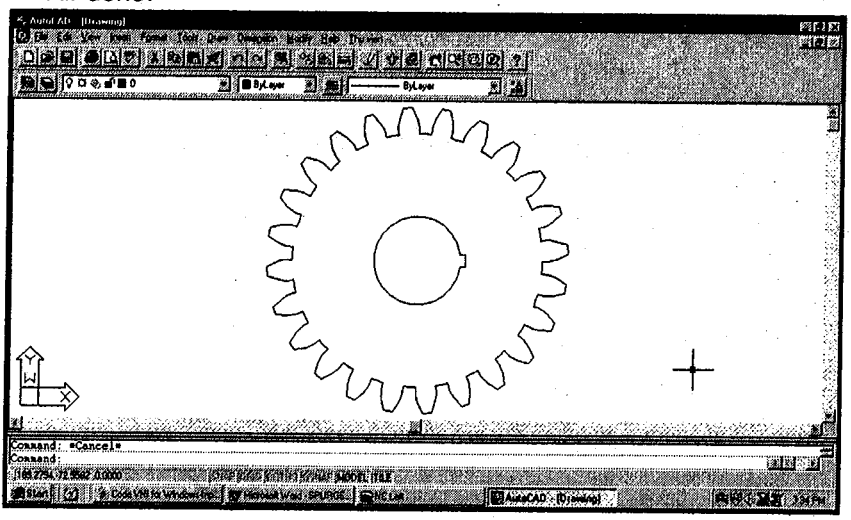


Ta có thể chọn điểm chèn bất kỳ (là tâm của vòng tròn chia) bằng cách chọn vào nút *Chọn điểm chèn <.*

Các dòng thông báo xuất hiện khi quá trình vẽ hoàn tất:

Converting to POLYLINE, please wait ...

All done!



Chương trình gồm 3 file:

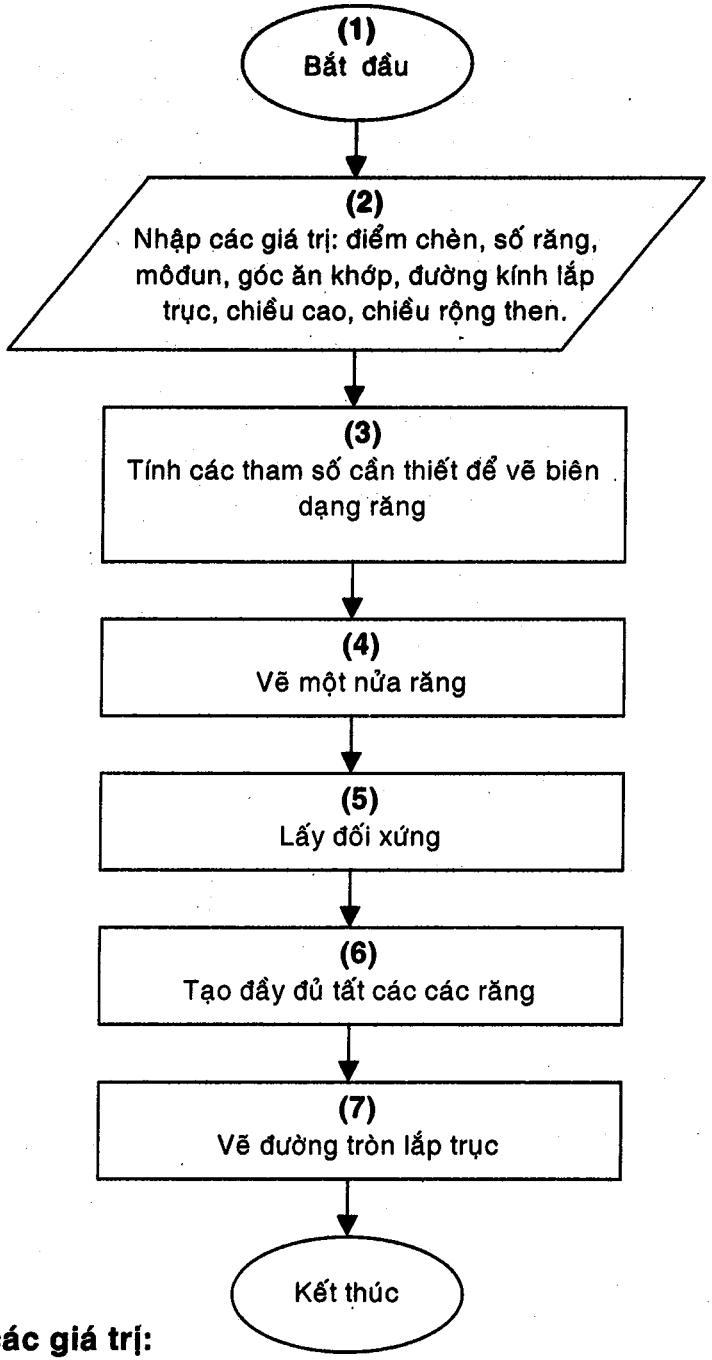
- SPURGEAR.LSP: File chương trình **AutoLISP**.
- SPURGEAR.DCL: File mô tả hộp thoại nhập số liệu.
- SPURGEAR.SLD: File chứa hình slide bánh răng, xuất hiện trên hộp thoại.

(1) Lưu đồ chương trình

Chú ý:

Để tạo file SUPRGEAR.SLD chứa hình slide của bánh răng:

- Chạy chương trình SUPRGEAR.LSP để vẽ hình bánh răng.
- Dùng lệnh **Mslide** để tạo lại slide file SUPRGEAR.SLD chứa hình bánh răng.



(2) Nhập các giá trị:

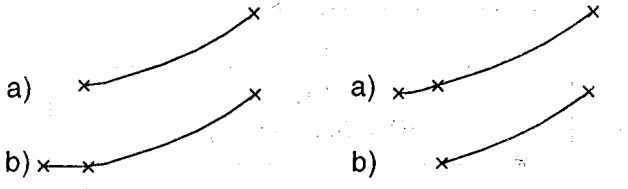
- Hàm GETDATA gán giá trị nhập trong hộp thoại cho các biến tương ứng

(3) Tính các tham số cần thiết để vẽ biên dạng răng:

- Đoạn đầu của hàm DRAW-GEAR tính toán các giá trị cần thiết theo tiêu chuẩn để vẽ biên dạng răng: đường kính vòng chia, đường kính vòng đỉnh, chiều cao đỉnh răng, chiều cao đáy răng, đường kính vòng đáy, đường kính vòng cơ sở.
- Nếu cần, ta có thể thay đổi cách tính các giá trị này để có biên dạng răng phù hợp.

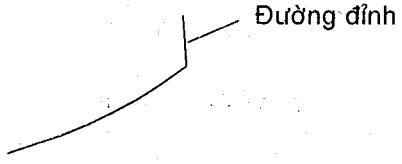
(4) Vẽ một nửa răng:

- Hàm DRAW-INV dựa trên các điểm được xác định bằng hàm INVOLUTE để vẽ phần đường cong của biên dạng răng (hình a).
- Hàm EXT_TRIM tùy theo kích thước của đường kính vòng cơ sở (biến DB) và đường kính vòng đáy (biến DR) sẽ thêm vào hoặc xén bớt phần đường cong do hàm INVOLUTE vẽ.



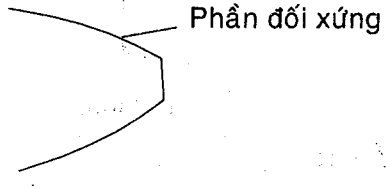
Thêm vào (DR<DB) Xén bớt (DR>DB)

- Hàm DRAW-TOP-LINE vẽ đường đỉnh



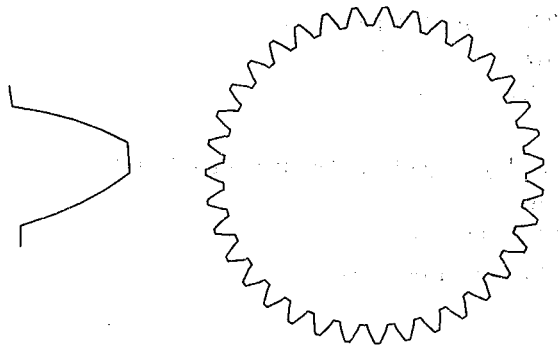
(5) Lấy đối xứng:

- Hàm MIR-IT lấy đối xứng tạo ra một nửa răng còn lại



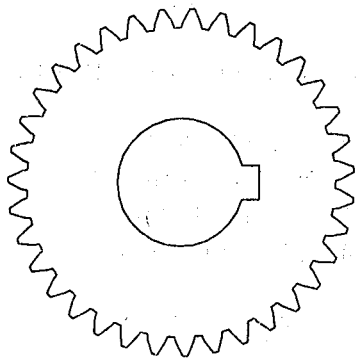
(6) Tạo đầy đủ tất cả các răng:

- Hàm SEGMENT tạo thêm đường tròn chân răng, sau đó dùng lệnh **Array** quanh tâm để tạo ra các răng còn lại.



(7) Vẽ đường tròn lắp trục:

- Hàm SEGMENT2 sử dụng các tham số đường kính lắp trục, chiều cao then, chiều rộng then để vẽ đường tròn lắp trục.



File chương trình SPURGEAR.LSP

```
;Tên file: SPURGEAR.LSP
;Mục đích: Vẽ biên dạng bánh răng dựa trên 3 thông số:
;          Số răng, modun, góc ăn khớp.
;Nhập SG tại dòng nhắc lệnh để chạy chương trình.
; Hàm SETTING tạo môi trường làm việc mới
(defun SETTING()
  (setq oerr *error*)
  (setq *error* err)
  (setq CMDECHO_OLD (getvar "CMDECHO"))
  (setvar "CMDECHO" 0)
  (setq BLIPMODE_OLD (getvar "BLIPMODE"))
  (setvar "BLIPMODE" 0)
```

```
;
;Hàm RESETTING trả về môi trường làm việc cũ
(defun RESETTING()
  (setvar "CMDECHO" CMDECHO_OLD)
  (setvar "BLIPMODE" BLIPMODE_OLD)
  (setq *error* oerr)
) ;Đóng RESETTING
;
;Hàm DXF dùng để lấy ra giá trị tương ứng của mã DXF
;trong record dữ liệu
(defun DXF (CODE ENAME)
  (cdr(assoc CODE(entget ENAME)))
) ;Đóng DXF
;
;Hàm SPURGEAR chứa tất cả các dòng lệnh vẽ biên dạng răng
(defun SPURGEAR (/ INS D Z PHI D0 CT RT DO RO A B
          DR DB INV-PLST P1 TRIMCODE INVENT P0 P
          CURVENT LINENT LINENT2 ENT2 P2)
;
;Hàm DIA hiển thị hộp thoại mô tả trong file SPURGEAR.DCL
(defun DIA (/ DCL_ID)
  (setq DCL_ID (load_dialog "SPURGEAR.DCL"))
  (if (not (new_dialog "spurgear" DCL_ID)) (exit))
  (set_tile "edit_X" (if (null INS) "" (rtos (car INS))) )
  (set_tile "edit_Y" (if (null INS) "" (rtos (cadr INS))) )
  (set_tile "edit_Z" Z)
  (action_tile "edit_Z" "(setq Z (get_tile \"edit_Z\"))")
  (set_tile "edit_M" M)
  (action_tile "edit_M" "(setq M (get_tile \"edit_M\"))")
  (set_tile "edit_PHI" PHI)
  (action_tile "edit_PHI" "(setq PHI (get_tile \"edit_PHI\"))")
  (set_tile "edit_D0" D0)
  (action_tile "edit_D0" "(setq D0 (get_tile \"edit_D0\"))")
  (set_tile "edit_CT" CT)
  (action_tile "edit_CT" "(setq CT (get_tile \"edit_CT\"))")
  (set_tile "edit_RT" RT)
  (action_tile "edit_RT" "(setq RT (get_tile \"edit_RT\"))")
  (action_tile "btn_INS" "(done_dialog 2)")
  (action_tile "accept" "(GETDATA) (done_dialog 1)" )
  (start_image "img1")
  (slide_image 0 0 (dimx_tile "img1") (dimy_tile "img1")
            "SPURGEAR.SLD" )
```

```
  (setq RES (start_dialog))
  (if (= RES 2)
    (progn
      (setq INS (getpoint))
      (dia)
    )
  )
  (unload_dialog DCL_ID)
) ; Đóng DIA
;
; Hàm GETDATA được gọi bởi nút lệnh OK của hộp thoại. Hàm này gán
; giá trị của hộp thoại cho các biến trong chương trình.
(defun GETDATA ()
  (setq X_COOR (atof (get_tile "edit_X"))
        Y_COOR (atof (get_tile "edit_Y"))
        INS (list X_COOR Y_COOR)
        Z   (atoi (get_tile "edit_Z"))
        M   (atof (get_tile "edit_M"))
        PHI (atof (get_tile "edit_PHI"))
        PHI (* (/ PHI 180) pi)
        D0  (atof (get_tile "edit_D0"))
        CT  (atof (get_tile "edit_CT"))
        RT  (atof (get_tile "edit_RT"))
  ) ; Đóng SETQ
) ; Đóng GETDATA
;
; Hàm DRAW-GEAR vẽ biên dạng răng. Hàm này chỉ được gọi khi
; người sử dụng chọn nút OK trên hộp thoại.
(defun DRAW-GEAR ()
  (setq D (* M Z)                          ; Đường kính vòng chia
        DO (* D (+ (/ 2.0 Z) 1.0))         ; Đường kính ngoài
        RO (/ DO 2.0)                      ; Bán kính ngoài
        A (/ D Z)                          ; Chiều cao đỉnh răng
        B (* 1.25 A)                       ;Chiều cao đáy răng
        DR (- D (* B 2.0))                 ;Đường kính vòng đáy
        DB (* D (cos PHI))                 ;Đường kính vòng tròn cơ sở
        INV-PLST (involute DB Z PHI)       ;Các điểm đường thân khai
        TRIMCODE nil
  ) ; Đóng Setq
  (command "Zoom" (list 0 (- B)) (list RO(/ RO 1.5)) )
  (setq INVENT (draw-inv INV-PLST)) ; (1)
  (setq P0 (car INV-PLST)
```

```
  )
  (if (and TRIMCODE (= TRIMCODE 0))
    (progn
      (setq P(list (/ DR 2.0)0))
      (command "Pedit" P "Y" "J" INVENT "" "X")
      (setq CURVENT (entlast))
    ) ; Đóng Progn
   (setq CURVENT (entlast))
  ); Đóng if
  (if (null TRIMCODE)(setq CURVENT INVENT))
  (setq LINENT (draw-top-line D DB Z RO)) ; (3) Vẽ đường đỉnh
  (command "copy" LINENT "" "0,0" "0,0")
  (setq LINENT2 (entlast))
  (setq ENT2 (mir-it CURVENT LINENT));(4) Vẽ biên dạng răng đối xứng
  (command "Pedit" CURVENT "J" LINENT ENT2 "" "X")
  (segment DR Z LINENT2)
  (setq P1 (list (- RO) (- RO)))
  (setq P2 (list RO RO))
  (command "Zoom" P1 P2)
  (prompt "\nConverting to POLYLINE, please wait ...")
  (command  "Pedit" (entlast) "J" "C" P1 P2 "" "X")
  (command "Move" (entlast) "" "0,0" INS) ;Chuyển về điểm chèn INS
  (command "Zoom" "C" INS "")
;
  (segment2)                          ; Vẽ đường tròn lắp trục
  (prompt "\nAll done!")
); Đóng DRAW-GEAR
; Bắt đầu hàm SPURGEAR
;
 (setq Z "" M "" PHI "" INS '(0 0) D0 "" CT "" RT "")
 (DIA)                                ; Gọi hiển thị hộp thoại.
 (if (= RES 1) (DRAW-GEAR))           ; Nếu chọn OK thì tiếp tục vẽ
) ; Đóng SPURGEAR
;
; Hàm INVOLUTE tính các điểm trên biên dạng răng
(defun INVOLUTE (DB Z PHI / NUMER DENOM FRAC THETA2MAX
    THETAMAX THETA-INC THETA PLIST RB XVAL YVAL P)
 (setq INVFACT 3)
 (setq NUMER (+ Z 2.0)
     DENOM (* Z (cos PHI))
     FRAC (/ NUMER DENOM)
     THETA2MAX (- (* FRAC FRAC) 1)
```

```
        THETA-INC (/ THETAMAX (float INVFACT))
        THETA 0
        PLIST NIL
        RB  (/ DB 2.0)
    ) ; Đóng setq
    (repeat (1+ INVFACT)
      (setq XVAL (do-x RB THETA)
            YVAL (do-y RB THETA)
            P (list XVAL YVAL)
            PLIST (append PLIST (list P))
      ) ; Đóng setq
     (setq THETA (+ THETA THETA-INC))
    ); Đóng repeat
    PLIST                          ; Trả danh sách tính được cho INV-PLST
);ĐóngINVOLUTE
;
; Hàm DO-X
(defun  DO-X (RB THETA)
   (* RB(+ (cos THETA) (* THETA (sin THETA))))
) :Đóng DO-X
;
;  Hàm DO-Y
(defun  DO-Y (RB THETA)
   (* RB(- (sin THETA) (* THETA (cos THETA))))
) ;Đóng DO-Y
;
;Định nghĩa hàm DRAW-INV
(defun DRAW-INV (INV-PLST / DIRPT PLIST P)
   (command "Pline" (nth 0 INV-PLST))
   (setq DIRPT (polar (nth 0 INV-PLST) 0 1))
   (command "A" "D" DIRPT)
   (setq PLIST (cdr INV-PLST))
   (foreach P PLIST (command P))
   (command  "")
   (entlast)
) ; Đóng DRAW-INV
;
; Hàm EXT-TRIM
(defun EXT-TRIM (P0 DR D / TRIMCODE DIST ENDR)
  (if (>(car P0)(/ DR 2.0))
     (progn
      (command "Line" (list (/ DR 2.0) 0) P0 "" )
```

```
      )
    )
    (if (< (car P0)(/ DR 2.0))
      (progn
        (command "Circle"  "0,0"  "D"  DR)
        (setq DIST(- (/ D 2.0) (car P0)))
        (command "Zoom" P0 (polar P0 0.6 DIST))
        (setq ENDR (entlast))
        (command  "Trim" ENDR "" P0 "")
        (command "Zoom" "P")
        (entdel ENDR)
        (setq TRIMCODE 1)
      ) ;Đóng progn
    );Đóng if
    TRIMCODE
) ; Đóng EXT-TRIM
;
; Hàm DRAW-TOP-LINE để vẽ đường đỉnh
(defun DRAW-TOP-LINE (D DB Z RO / THETA-P XP YP ALPHA
                      BETA TANG ANGEND INV-ENDPT LEND)
  (setq THETA-P (sqrt (- (* (/ D DB) (/ D DB)) 1.0))
        XP(DO-X (/ DB 2.0)THETA-P)
        YP(DO-Y (/ DB 2.0)THETA-P)
        ALPHA (ATAN YP XP)
        ABETA (ANGLE (list 0 0) (last INV-PLST))
        BETA (- ABETA ALPHA)
        TANG (/ pi Z)
        ANGEND (- (+ ALPHA TANG)BETA)
        INV-ENDPT (last INV-PLST)
        LEND (polar (list 0 0) ANGEND RO)
  ); Đóng setq
  (command "line" INV-ENDPT LEND  "")
  (redraw)
  (entlast)
); ĐóngDRAW-TOP-LINE
;
; Hàm MIR-IT để vẽ nửa đường cong đối xứng của biên dạng răng
(defun MIR-IT (CVENT LINENT / PT)
  (setq PT (dxf 11 LINENT))
  (command "Mirror" CVENT ""  "MID" PT "0,0" "")
  (entlast)
) ; Đóng MIR-IT
```

```
; Hàm SEGMENT dùng để hoàn tất biên dạng răng.
(defun SEGMENT (DR Z EN / P1 P2 ANG DIST MIDP P0 PANG
        PANG2 P P3 ENT3 ENT11 ENT12 EN1 EN2)
  (setq P1(dxf 10 EN)
        P2(dxf 11 EN)
        ANG (angle P1 P2)
        DIST (/ (distance P1 P2) 2.0)
        MIDP (polar P1 ANG DIST)
        P0 (list 0 0)
        PANG (angle P0 MIDP)
        PANG2 (/ pi Z)
        P (polar P0 PANG (/ DR 2.0))
        P1 (polar P0 (- PANG PANG2)(/ DR 2.0))
        P2 (polar P0 (+ PANG PANG2)(/ DR 2.0))
        P3 (polar P0 (+ PANG PANG2 PANG2)(/ DR 2.0))
        ENT3 (entlast)                    ; Biên dạng răng là đường pline
  ) ;Đóng setq
  (command "Zoom" "W" P3 P1)
  (command "Circle" "0,0" "D" DR)         ;Đường tròn chân răng
  (command  "Trim"  "all" "" P "")        ;Xén đường tròn chân răng
  (command "Zoom" "p")
  (command "line"  P0 P1 "")
  (setq ENT11 (entlast))
  (command "line" P0 P2 "")
  (setq ENT12 (entlast))
  (command "trim" ENT11 ENT12 "" P3 "");(5) Hoàn chỉnh tiết diện răng
  (entdel ENT11)
  (entdel ENT12)
  (entdel EN)
  (command "Zoom" "W" P3 P1)
  (command "pedit" P1 "Y"  "X")
  (setq EN1 (entlast))
  (command "pedit" P2 "Y"  "X")
  (setq EN2 (entlast))
  (command "pedit" EN1 "J" MIDP EN2 "" "X")
  (command "zoom" "P")
  (command "array" P1 "" "P" "0,0" Z "360" "Y");(6)Vẽ hoàn chỉnh ;bánh răng
);Đóng SEGMENT
;
; Hàm SEGMENT2 vẽ đường tròn lấp trục
(defun SEGMENT2 (/ POINT1 POINT2 POINT3 POINT4 POINTA POINTB)
  (command "Circle" INS "D" D0)           ; Vẽ đường tròn lấp trục
```

```
  (command "Move" (entlast) "" (list 0 (/ RT 2)) "")
  (setq L1 (entlast))
  (setq POINTA (polar INS 0 (/ D0 2))
        POINTB (polar INS (/ Pi 2) (/ RT 2))
  )
  (Command "ID" "int" POINTA POINTB)
  (setq POINT1 (getvar "lastpoint")
        POINT2 (polar POINT1 0 CT)
        POINT3 (polar POINT2 (/ (- Pi) 2) RT)
        POINT4 (polar POINT3 Pi CT)
  )
  (command "PLine" POINT1 POINT2 POINT3 POINT4 "")
  (entdel L1)
  (command "trim" POINT2 "" POINTA "")
  (command "Pedit" POINT2 "J" POINT1 "" "")
); Đóng SEGMENT2
;
;Chương trình chính C:SG được gọi từ dòng nhắc lệnh của AutoCAD.
(defun C:SG()
  (setting)
  (spurgear)
  (resetting)
  (princ)
);Đóng C:SG
(prompt "\nEnter SG to start")
; Kết thúc file SPURGEAR.LSP
```

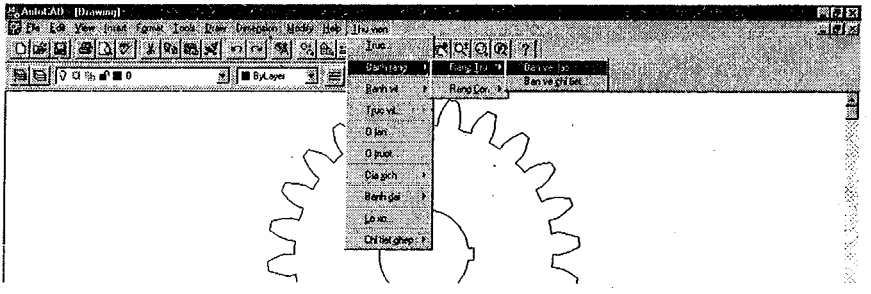
File mô tả hộp thoại SPURGEAR.DCL

```
//Tên file: SPURGEAR.DCL
//Dùng để nhập số liệu cho chương trình SPURGEAR.LSP
//
spurgear: dialog {
    label = "Chuong trinh ve banh rang";
      : row {
      : text {label = "Diem chen:"; }
      : button {key = "btn_INS"; label = "&Chon diem chen <";}
      : edit_box {key = "edit_X"; label = "&X"; }
      : edit_box {key = "edit_Y"; label = "&Y"; }
    }
    :row {
      : column {
        : edit_box {key = "edit_Z"; label = "So &rang (Z): ";}
```

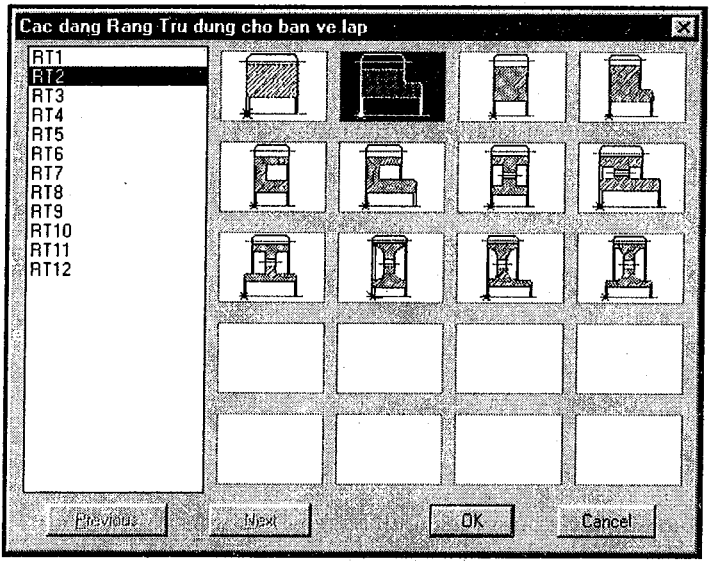
```
        : edit_box {key = "edit_PHI"; label = "&Goc an khop (Phi): ";}
        : edit_box {
            key = "edit_D0";
            label = "Duong kinh lap &truc (d0): ";
        }
        : edit_box {
            key = "edit_CT";
            label = "Chi&eu cao then (t): ";
        }
        : edit_box {
            key = "edit_RT";
            label = "C&hieu rong then (b): ";
        }
      }
      :image_button {key = "img1" ; width = 30; height = 5;
                color = dialog_background;}
    }
    spacer_1;
    ok_cancel;
}
// Kết thúc file SPURGEAR.DCL
```

21.3 Chương trình tự động vẽ các chi tiết máy

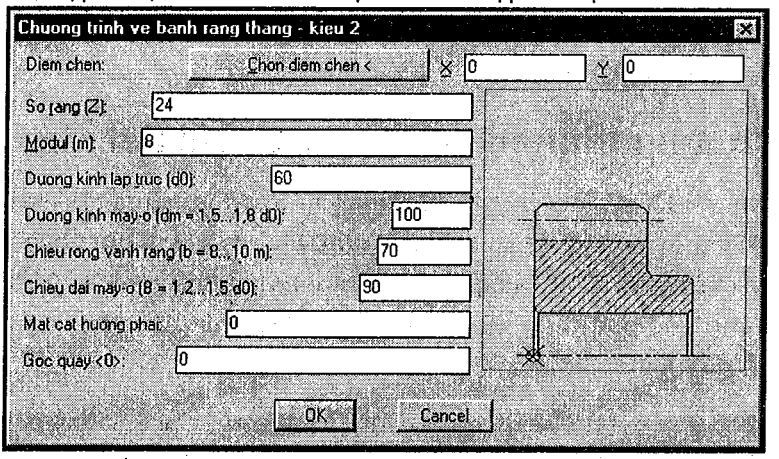
Chúng tôi đã tạo các chương trình tự động vẽ các chi tiết máy, các chương trình này có thể sử dụng trên **AutoCAD 14, AM 2000, 2002** và **2004**. Hiệu nay chúng tôi đang hoàn thiện dần các chương trình này để có thể giúp ích cho các nhà thiết kế cơ khí cũng như sinh viên làm luận văn tốt nghiệp và đồ án môn học.



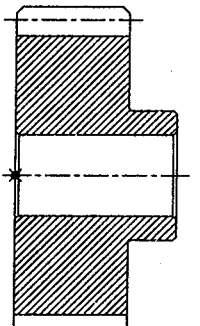
Vì dụ muốn vẽ bánh răng trụ răng thẳng ta chọn *Thuvien> Banh rang> Rang tru> Banvelap...* sẽ xuất hiện hộp thoại sau:



Trên hộp thoại này ta chọn dạng răng RT2 như hình vẽ. Sau khi chọn RT2, hộp thoại sau sẽ xuất hiện để ta nhập dữ liệu vào:



Sau đó ta chọn vào nút *Chon diem chen>* để định điểm chèn cho bản vẽ trên màn hình đồ họa. Bản vẽ có dạng sau:



Để vẽ bánh răng trên ta tạo hai file RT2.LSP và RT2.DCL.

File RT2.LSP

```
; ********** BANH RANG TRU - RANG THANG  *************
;Hàm DIA hiển thị hộp thoại mô tả trong file RT2.DCL
;Người viết: N. H. Luân, hiệu chỉnh: N. T. Trung
(defun DIA (/ DCL_ID)
  (setq DCL_ID (load_dialog "RT2.DCL"))
  (if (not (new_dialog "RT2" DCL_ID)) (exit))
  (set_tile "edit_X" (if (null P) "" (rtos (car P))) )
  (set_tile "edit_Y" (if (null P) "" (rtos (cadr P))) )
  (set_tile "edit_Z" Z)
  (action_tile "edit_Z" "(setq Z (get_tile \"edit_Z\"))")
  (set_tile "edit_M" M)
  (action_tile "edit_M" "(setq M (get_tile \"edit_M\"))")
  (set_tile "edit_DT" DT)
  (action_tile "edit_DT" "(setq DT (get_tile \"edit_DT\"))")
  (set_tile "edit_DM" DM)
  (action_tile "edit_DM" "(setq DM (get_tile \"edit_DM\"))")
  (set_tile "edit_B" B)
  (action_tile "edit_B" "(setq B (get_tile \"edit_B\"))")
  (set_tile "edit_LM" LM)
  (action_tile "edit_LM" "(setq LM (get_tile \"edit_LM\"))")
  (set_tile "edit_MC" MC)
  (action_tile "edit_MC" "(setq MC (get_tile \"edit_MC\"))")
  (set_tile "edit_ANG" ANG)
  (action_tile "edit_ANG" "(setq ANG (get_tile \"edit_ANG\"))")
  (action_tile "btn_INS" "(done_dialog 2)")
  (action_tile "accept" "(GETDATA) (done_dialog 1)" )
  (start_image "img1")
  (slide_image 0 0 (dimx_tile "img1") (dimy_tile "img1")
                   "RT2.SLD" )
  (end_image)
  (setq RES (start_dialog))
  (if (= RES 2)
    (progn
      (setq P (getpoint))
      (dia)
    )
  )
  (unload_dialog DCL_ID)
) ; Đóng DIA
;
; Hàm GETDATA được gọi bởi nút lệnh OK của hộp thoại. Hàm này gán
; giá trị của hộp thoại cho các biến trong chương trình.
```

```
 (setq  X_COOR (atof (get_tile "edit_X"))
        Y_COOR (atof (get_tile "edit_Y"))
        P (list X_COOR Y_COOR)
        Z   (atof (get_tile "edit_Z"))
        M   (atof (get_tile "edit_M"))
        DT  (atof (get_tile "edit_DT"))
        DM  (atof (get_tile "edit_DM"))
        B   (atof (get_tile "edit_B"))
        LM  (atof (get_tile "edit_LM"))
        MC  (atof (get_tile "edit_MC"))
        ANG  (atof (get_tile "edit_ANG"))
  ) ; Đóng SETQ
) ; Đóng GETDATA
(Defun DRAW-RT2()
(prompt "\n          CHUONG TRINH VE BANH RANG THANG - KIEU 2 ")
 (setq R(/ (* M Z) 2)
       rt(/ DT 2)
       rm(/ DM 2)
 )
 (cond ((null ANG)
          (setq ANG 0)
         )
 )
(cond ((null MC)
          (setq MC 0)
         )
 )
 (setq VAT(/ DT 25)
         V(/ pi 2)
         RA(+ R M)
         rf(- r (* 1.25 m))
         p1(polar p v (+ r (/ m 2)))
      p2(polar p1 (/ pi 4) (* (/ m 2) (sqrt 2)))
      p3(polar p2 0 (- b m))
      p4(polar p3 (- (/ pi 4)) (* (/ m 2) (sqrt 2)))
      p5(polar p4 (- v) (- (+ r (/ m 2)) rm))
      p6(polar p5 0 (- lm b))
        p65(polar p6 pi vat)
      p67(polar p6 (- v) vat)
      p7(polar p6 (- v) rm)
      p8(polar p7 pi vat)
      p9(polar p8 v rt)
      p10(polar p9 (/ pi 4) (* vat (sqrt 2)))
      p11(polar p 0 vat)
      p12(polar p11 v rt)
      p13(polar p12 (- pi (/ pi 4)) (* vat (sqrt 2)))
```

```
      p15(polar p14 0 b)
        p18(polar p5 v (* 2 vat))
        w1(polar p6 v (* 4 vat))
)
(Defun chontu(tentt / tentt2 tchon)
  (setq tentt2 tentt tchon(ssadd tentt2))
  (while (setq tentt2(entnext tentt2))
  (ssadd tentt2 tchon)
  )
  (princ tchon)
)
(command "layer" "m" "duongbao" "co" "white" "" ""
        "pline" p14 p1 p2 p3 p4 p15 ""
)
            (setq tentt(entlast))
(command "pline" p13 p14 p15 p5 p65 p67 p10 p9 p12 p13 ""
           "zoom" "w" p13 w1
           "fillet" "r" vat
           "fillet" p65 p18
           ;"pline" p16 p13 p12 p9 p10 p17 ""
        "line" p9 p8 ""
        "line" p7 p10 ""
        "line" p11 p12 ""
        "line" p p13 ""
)
(setq ;g1(polar p16 0 (/ b 2))
     ;g2(polar g1 (- v) (+ dt t))
     dx5(polar p5 (- v) (* rm 2))
     cen1(polar p pi m )
     cen2(polar p7 0 m )
        cen3(polar cen1 v r)
        cen4(polar cen3 0 (+ b (* 2 m)))
        z2(polar p2 (- v) (* 2 ra))
)
(command "layer" "m" "duongtam" "lt" "center" "" "co" "red" "" ""
           "zoom" "all"
           "ltscale" 12
        "line" cen3 cen4 ""
        "mirror" (chontu tentt) "" p p8 ""
           "line" cen1 cen2 ""
           ;"layer" "s" "rt2" ""
           ;"line" p16 p17 ""
)
 (command "zoom" "w" z2 p4
           "layer" "m" "matcat" "co" "magenta" "" ""
        "hatch" "ansi31" 1 mc p5 dx5 ""
```

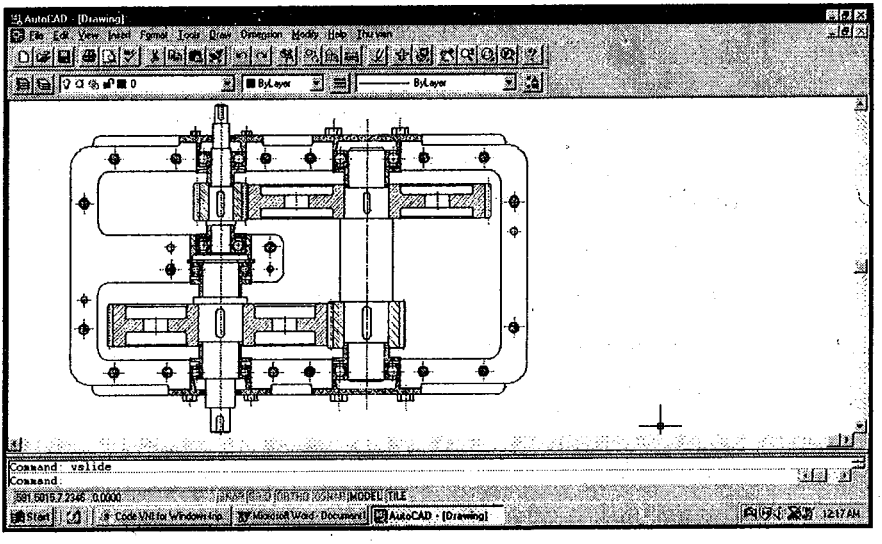
```
            "layer" "m" "point" "co" "blue" "" ""
            "pdmode" 35
            "pdsize" (/ dt 10)
            "point" p
            "layer" "s" "0" ""
          "zoom" "e"
  )
)
;
(defun C:RT2()
  (setq Z "" M ""  P '(0 0) DT "" DM "" B "" LM "" MC "0" ANG "0")
  (DIA)                                  ; Gọi hiển thị hộp thoại.
  (if (= RES 1) (DRAW-RT2))              ; Nếu chọn OK thì tiếp tục vẽ
)
(prompt "\n      Chuong trinh duoc lam boi Nguyen Hoang Luan ")
(prompt "\n        Ban go 'RT2' de chay chuong trinh        ")
```

File RT2.DCL

```
//Tên file: RT2.DCL
//Dùng để nhập số liệu cho chương trình RT2.LSP
//
RT2: dialog {
     label = "Chuong trinh ve banh rang thang - kieu 2";
     : row {
       : text {label = "Diem chen:"; }
       : button {key = "btn_INS"; label = "&Chon diem chen <";}
       : edit_box {key = "edit_X"; label = "&X"; }
       : edit_box {key = "edit_Y"; label = "&Y"; }
     }
     :row {
     : column {
       :  edit_box {key = "edit_Z"; label = "So &rang (Z): ";}
       :  edit_box {key = "edit_M"; label = "&Modul (m): ";}
       :  edit_box {
              key = "edit_DT";
              label = "Duong kinh lap &truc (d0): ";
      }
       :  edit_box {
              key = "edit_DM";
              label = "Duong kinh may-o (dm = 1,5...1,8 d0): ";
      }
       :  edit_box {
              key = "edit_B";
              label = "Chieu rong vanh rang (b = 8...10 m): ";
      }
       :  edit_box {
```

```
            label = "Chieu dai may-o (B = 1,2...1,5 d0): ";
        }
        : edit_box {
            key = "edit_MC";
            label = "Mat cat huong phai: ";
        }
        : edit_box {
            key = "edit_ANG";
            label = "Goc quay <0>: ";
        }
    }
    :image_button {key = "img1" ; width = 30; height = 5;
        color = dialog_background;}
    }
    spacer_1;
    ok_cancel;
}
// Kết thúc file RT2.DCL
```

Dưới đây trình bày kết quả sử dụng thư viện vẽ để tạo hình chiếu bằng của hệ thống truyền động cơ khí.



BẢNG TRA CỨU CÁC MÃ DXF (DXF GROUP CODE)

Các mã DXF trình bày trong phụ lục này được sử dụng trong các *record* dữ liệu của cơ sở dữ liệu đối tượng của **AutoCAD**.

Bảng I.1 liệt kê các mã DXF theo thứ tự. Các bảng I.2 và I.3 trình bày chi tiết về cách sử dụng các mã này.

Các mã DXF trong bảng I.2 được dùng trong hầu hết các loại đối tượng. Các mã DXF trong bảng I.3 chỉ dùng riêng cho một số đối tượng.

Bảng I.1 Các mã DXF

Mã DXF	Ý nghĩa
-4	Điều kiện so sánh (Chỉ dùng với hàm **Ssget**).
-2	Mã đối tượng
0	Kiểu đối tượng
1	Giá trị chính (kiểu chuỗi) của đối tượng
2	Tên: Tên khối, tên thuộc tính v.v...
3-4	Các chuỗi phụ
6	Tên dạng đường
7	Tên kiểu chữ
8	Tên lớp bản vẽ
10	Tọa độ điểm chính của đối tượng (điểm đầu của đường thẳng, tâm đường tròn ...)
11-18	Tọa độ các điểm khác
39	Chiều dày
40-48	Các giá trị số thực (hệ số tỉ lệ của khối, chiều cao chữ ...)
49	Các giá trị được sử dụng nhiều lần (như chiều dài nét liền của dạng

	đường)
50-58	Các giá trị góc
62	Mã màu (0 = BYBLOCK, 256 = BYLAYER).
66	Cờ xác định các đối tượng theo sau (các thuộc tính của khối ...)
67	Không gian vẽ (1 = Không gian giấy vẽ, 0 = Không gian mô hình)
70-78	Các số nguyên xác định các bộ đếm, cờ trạng thái
90-99	Các giá trị số nguyên khác
100	Đánh dấu phân loại các mã DXF theo sau
102	Chuỗi đánh đấu "*{arbitrary name*" hoặc "*}*".
105	Mã đối tượng xác định vị trí các biến kích thước trong bảng mô tả
210	Véc tơ định hướng trục Z của đối tượng (danh sách gồm 3 số thực)
280-289	Các giá trị số nguyên
300-309	Các chuỗi *arbitrary*
310-319	*Arbitrary binary chunks.*
320-329	*Arbitrary object handles.*
330-339	Giá trị trỏ đến các đối tượng khác trong bản vẽ
340-349	Giá trị trỏ đến các đối tượng khác trong bản vẽ
350-359	Giá trị xác định việc sở hữu các đối tượng khác trong bản vẽ
360-369	Giá trị xác định việc sở hữu các đối tượng khác trong bản vẽ
999	Các ghi chú

Bảng I.2 Mã DXF dùng cho tất cả các đối tượng

Mã DXF	Ý nghĩa	Giá trị mặc định
-1	Mã đối tượng (thay đổi mỗi lần bản vẽ được mở lại)	Không có
0	Kiểu đối tượng	Không có
5	Giá trị *handle* của đối tượng	Không có
100	Đánh dấu phân loại các mã DXF theo sau.	Không có
67	Xác định không gian bản vẽ. 0 (hoặc không có) = Không gian mô hình 1 = Không gian giấy vẽ	0
8	Tên lớp bản vẽ chứa đối tượng	Không có
6	Tên dạng đường của đối tượng (mã DXF này chỉ	"BYLAYER"

	xuất hiện nếu tên dạng đường khác "BYLAYER")	
62	Mã màu của đối tượng (mã DXF này chỉ xuất hiện nếu mã màu khác 256, "BYLAYER")	"BYLAYER"
48	Hệ số tỉ lệ dạng đường	1.0
60	Tính chất trông thấy của đối tượng 0 = trông thấy (*visible*) 1 = không trông thấy (*invisible*)	0

Bảng I.3 Mã DXF của riêng từng kiểu đối tượng

Kiểu đối tượng	Mã DXF	Ý nghĩa
3DFACE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbFace)
	10	Tọa độ đỉnh đầu tiên (trong hệ trục WCS)
	20	Thành phần Y của tọa độ đỉnh đầu tiên (trong hệ trục WCS)
	30	Thành phần Z của tọa độ đỉnh đầu tiên (trong hệ trục WCS)
	11	Tọa độ đỉnh thứ hai (trong hệ trục WCS)
	21	Thành phần Y của tọa độ đỉnh thứ hai (trong hệ trục WCS)
	31	Thành phần Z của tọa độ đỉnh thứ hai (trong hệ trục WCS)
	12	Tọa độ đỉnh thứ ba (trong hệ trục WCS)
	22	Thành phần Y của tọa độ đỉnh thứ ba (trong hệ trục WCS)
	32	Thành phần Z của tọa độ đỉnh thứ ba (trong hệ trục WCS)
	13	Tọa độ đỉnh thứ tư (trong hệ trục WCS)
	23	Thành phần Y của tọa độ đỉnh thứ tư (trong hệ trục WCS)
	33	Thành phần Z của tọa độ đỉnh thứ tư (trong hệ trục WCS)
	70	Tính chất trông thấy của các cạnh (mặc định = 0) 1 = Cạnh đầu tiên không trông thấy 2 = Cạnh thứ hai không trông thấy 4 = Cạnh thứ ba không trông thấy 8 = Cạnh thứ tư không trông thấy

3DSOLID	100	Đánh dấu phân loại các mã DXF theo sau (AcDbFace)
	70	Số phiên bản dạng thức sử dụng
	1	Các dữ liệu riêng của **AutoCAD**, không được cung cấp bản quyền sử dụng (chứa nhiều dòng, mỗi dòng ít hơn 255 ký tự)
	3	Các dòng bổ sung (trong trường hợp chuỗi chứa trong mã DXF 1 lớn hơn 255 ký tự)
ARC	100	Đánh dấu phân loại các mã DXF theo sau (AcDbCircle)
	39	Chiều dày của đối tượng (mặc định = 0)
	10	Tọa độ tâm đường tròn (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ tâm đường tròn (biểu diễn trong OCS)
	30	Thành phần Z của tọa độ tâm đường tròn (biểu diễn trong OCS)
	40	Bán kính
	100	Đánh dấu phân loại các mã DXF theo sau (AcDbArc)
	50	Góc tiếp tuyến tại điểm đầu tiên
	51	Góc tiếp tuyến tại điểm cuối cùng
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
ATTDEF	100	Đánh dấu phân loại các mã DXF theo sau (AcDbText)
	39	Chiều dày (tùy chọn; mặc định = 0)
	10	Tọa độ điểm canh lề thứ nhất của dòng chữ (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ điểm bắt đầu dòng chữ (biểu diễn trong OCS)
	30	Thành phần Z của tọa độ điểm bắt đầu dòng chữ (biểu diễn trong OCS)
	40	Chiều cao dòng chữ
	1	Giá trị mặc định (kiểu chuỗi)

100	Đánh dấu phân loại các mã DXF theo sau (AcDbAttributeDefinition)
50	Độ nghiêng của dòng chữ (tùy chọn; mặc định = 0)
41	Hệ số tỉ lệ theo trục X (tùy chọn; mặc định = 1)
51	Góc nghiêng của chữ (tùy chọn; mặc định = 0)
7	Tên kiểu chữ (tùy chọn; mặc định = "STANDARD")
71	Hướng tạo dòng chữ (mặc định = 0) 2 = Backward (đối xứng gương theo trục X) 4 = Upside down (đối xứng gương theo trục Y)
72	Kiểu canh lề chữ theo phương ngang (mặc định=0) 0 = Left 1 = Center 2 = Right 3 = Aligned 4 = Middle 5 = Fit
11	Tọa độ điểm canh lề thứ hai của dòng chữ (biểu diễn trong OCS).(tùy chọn; chỉ xuất hiện khi mã DXF 72 hoặc 74 có giá trị khác 0)
21	Giá trị Y của tọa độ điểm canh lề thứ hai (biểu diễn trong OCS)
31	Thành phần Z của tọa độ điểm canh lề thứ hai (biểu diễn trong OCS)
210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
100	Đánh dấu phân loại các mã DXF theo sau (AcDbAttributeDefinition)
3	Dòng nhắc khi chèn khối với thuộc tính
2	Attribute tag
70	Cờ trạng thái của thuộc tính 1 = Invisible 2 = Constant 4 = Verification 8 = Preset
73	Chiều dài dòng chữ (Tùy chọn; mặc định = 0)

	74	Kiểu canh lề dòng chữ theo phương thẳng đứng (mặc định = 0) 0 = Baseline 1 = Bottom 2 = Middle 3 = Top
ATTRIB	100	Đánh dấu phân loại các mã DXF theo sau (AcDbText)
	39	Chiều dày (tùy chọn; mặc định = 0)
	10	Tọa độ điểm đầu của dòng chữ (biểu diễn trong hệ trục tọa độ đối tượng OCS – Objects Coordinate System)
	20	Thành phần X trong tọa độ điểm đầu của dòng chữ (biểu diễn trong OCS)
	30	Thành phần Y trong tọa độ điểm đầu của dòng chữ (biểu diễn trong OCS)
	40	Chiều cao của chữ
	1	Giá trị (kiểu chuỗi)
	100	Đánh dấu phân loại các mã DXF theo sau (AcDbAttribute)
	2	Attribute tag (kiểu chuỗi)
	70	Cờ trạng thái của thuộc tính 1 = Invisible 2 = Constant 4 = Verification 8 = Preset
	73	Chiều dài dòng chữ (Tùy chọn; mặc định = 0)
	50	Độ nghiêng của dòng chữ (tùy chọn; mặc định = 0)
	41	Hệ số tỉ lệ thao trục X (tùy chọn; mặc định = 1)
	51	Góc nghiêng của chữ (tùy chọn; mặc định = 0)
	7	Tên kiểu chữ (tùy chọn; mặc định = "STANDARD")
	71	Hướng tạo dòng chữ (mặc định = 0) 2 = Backward (đối xứng gương theo trục X) 4 = Upside down (đối xứng gương theo trục Y)
	72	Kiểu canh lề chữ theo phương ngang (mặc định=0) 0 = Left 1 = Center 2 = Right 3 = Aligned 4 = Middle 5 = Fit

	74	Kiểu canh lề dòng chữ theo phương thẳng đứng (mặc định = 0) 0 = Baseline 1 = Bottom 2 = Middle 3 = Top
	11	Tọa độ điểm canh lề thứ hai của dòng chữ (biểu diễn trong OCS) (tùy chọn; chỉ xuất hiện khi mã DXF 72 hoặc 74 có giá trị khác 0)
	21	Thành phần Y của tọa độ điểm canh lề thứ hai (biểu diễn trong OCS)
	31	Thành phần Z của tọa độ điểm canh lề thứ hai (biểu diễn trong OCS)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
BODY	100	Đánh dấu phân loại các mã DXF theo sau (AcDbModelerGeometry)
	70	Số phiên bản của dạng thức sử dụng
	1	Các dữ liệu riêng của **AutoCAD**, không được cung cấp bản quyền sử dụng (chứa nhiều dòng, mỗi dòng ít hơn 255 ký tự)
	3	Các dòng bổ sung (trong trường hợp chuỗi chứa trong mã DXF 1 lớn hơn 255 ký tự)
CIRCLE	10	Tọa độ tâm đường tròn
	40	Bán kính
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
DIMENSION	100	Đánh dấu phân loại các mã DXF theo sau (AcDbDimesion)
	1	Chữ số kích thước do người sử dụng nhập vào (mặc định = giá trị kích thước thật)
	2	Tên của một khối giả, dùng để chứa hình ảnh các thành phần của đối tượng kích thước
	3	Tên kiểu kích thước
	10	Tọa độ điểm chuẩn khi ghi kích thước (biểu diễn

11	Tọa độ điểm *Middle* của chữ số kích thước
12	Tọa độ điểm chuẩn khi ghi kích thước theo kiểu *Baseline* và *Continue*)
13	Tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc
14	Tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc
15	Tọa độ điểm chuẩn khi ghi kích thước đường kính, bán kính hoặc kích thước góc
16	Tọa độ điểm xác định vị trí đường kích thước (là cung tròn) khi ghi kích thước góc
20	Thành phần Y của tọa độ điểm chuẩn khi ghi kích thước (biểu diễn trong hệ trục WCS)
30	Thành phần Z của tọa độ điểm chuẩn khi ghi kích thước (biểu diễn trong hệ trục WCS)
21	Thành phần Y của tọa độ điểm *Middle* của chữ số kích thước (biểu diễn trong OCS)
31	Thành phần Z của tọa độ điểm *Middle* của chữ số kích thước (biểu diễn trong OCS)
22	Thành phần Y của tọa độ điểm chuẩn khi ghi kích thước theo kiểu *Baseline* và *Continue* (trong OCS)
32	Thành phần Z của tọa độ điểm chuẩn khi ghi kích thước theo kiểu *Baseline* và *Continue* (trong OCS)
23	Thành phần Y của tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc (trong WCS)
33	Thành phần Z của tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc (trong WCS)
24	Thành phần Y của tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc (trong WCS)
34	Thành phần Z của tọa độ điểm chuẩn khi ghi kích thước thẳng và kích thước góc (trong WCS)
25	Thành phần Y của tọa độ điểm chuẩn khi ghi kích thước đường kính, bán kính hoặc kích thước góc (trong WCS)
35	Thành phần Z của tọa độ điểm chuẩn khi ghi kích thước đường kính, bán kính hoặc kích thước góc (trong WCS)
26	Thành phần Y của tọa độ điểm xác định vị trí đường kích thước (là cung tròn) khi ghi kích thước góc (trong OCS)

	36	Thành phần Z của tọa độ điểm xác định vị trí đường kích thước (là cung tròn) khi ghi kích thước góc (trong OCS)
	40	Chiều dài đoạn dẫn của đường kích thước ghi khi kích thước bán kính và đường kính
	50	Giá trị góc nghiêng của đường kích thước so với đường chuẩn khi ghi các kích thước thẳng
	51	Chiều của phương nằm ngang
	52	Giá trị góc nghiêng của đường gióng so với đường chuẩn khi ghi các kích thước thẳng
	53	Góc quay của chữ số kích thước
	70	Kiểu kích thước (từ 0-6 là các giá trị số nguyên; từ 32-128 là các *bit code)* 0 = Nằm ngang (*horizontal*), thẳng đứng (*vertical*) và quay (*rotated)* 1 = Song song đường chuẩn (*Aligned)* 2 = Góc (*Angular)* 3 = Đường kính *(Diameter)* 4 = Bán kính *(Radius)* 5 = Góc (chọn 3 điểm) 6 = Tọa độ một điểm *(Ordinate)* 32 = Tham khảo đến một khối 64 = Thành phần X của tọa độ điểm và vị trí mặc định 128= Thành phần X của tọa độ điểm và vị trí do người sử dụng chọn
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
ELLIPSE	100	Đánh dấu phân loại các mã DXF theo sau (*AcDbEllipse*) .
	10	Tọa độ điểm tâm (biểu diễn trong WCS)
	20	Thành phần Y của tọa độ điểm tâm (biểu diễn trong WCS)
	30	Thành phần Z của tọa độ điểm tâm (biểu diễn trong WCS)
	11	Tọa độ điểm cuối của trục chính tương ứng với

	21	Thành phần Y của tọa độ điểm cuối của trục chính tương ứng với điểm tâm
	31	Thành phần Z của tọa độ điểm cuối của trục chính tương ứng với điểm tâm
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	40	Tỉ lệ giữa trục phụ và trục chính
	41	Tham số đầu (0.0 = vẽ nguyên ellipse)
	42	Tham số cuối (2π = vẽ nguyên ellipse)
HATCH	100	Đánh dấu phân loại các mã DXF theo sau (AcDbHatch)
	10	Tọa độ điểm xác định cao độ của đối tượng (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ điểm xác định cao độ của đối tượng (biểu diễn trong OCS)
	30	Thành phần Z của tọa độ điểm xác định cao độ của đối tượng (biểu diễn trong OCS)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	2	Tên mẫu mặt cắt
	70	Xác định trạng thái tô bằng mẫu solid 1 = Tô bằng mẫu solid 0 = Tô bằng mẫu mặt cắt
	71	Xác định trạng thái liên kết 1 = Tạo mặt cắt liên kết 0 = Tạo mặt cắt không liên kết
	91	Số lượng các đường biên kín

92	Quy định kiểu đường biên 0 = Default 1 = External 2 = Polyline 4 = Derived 8 = Textbox 16 = Outermost
93	Số lượng các cạnh tạo thành đường biên
72	Kiểu của cạnh tạo thành đường biên 1 = Đường thẳng 2 = Cung tròn 3 = Cung ellipse 4 = Đường cong Spline
97	Số lượng các đối tượng tạo thành đường biên
340	Tham khảo (cứng) đến các đối tượng tạo thành đường biên
75	Kiểu tô mặt cắt 0 = Normal style 1 = Outer style 2 = Ignore style
76	Kiểu mẫu mặt cắt 0 = User-defined 1 = Predefined 2 = Custom
52	Góc quay của mẫu mặt cắt
41	Hệ số tỉ lệ của mẫu mặt cắt hoặc là khoảng cách giữa các đường nét trong kiểu User-defined
77	Quy định vẽ nét đôi trong kiểu User-defined 1 = Vẽ nét đôi 2 = Vẽ nét đơn
78	Số lượng các đường nét tạo thành mẫu mặt cắt
53	Góc của đường nét tạo thành mẫu mặt cắt
43	Thành phần X của điểm gốc của đường nét tạo thành mẫu mặt cắt
44	Thành phần Y của điểm gốc của đường nét tạo thành mẫu mặt cắt
45	Thành phần X của giá trị khoảng cách (offset) của đường nét tạo thành mẫu mặt cắt
46	Thành phần Y của giá trị khoảng cách (offset) của đường nét tạo thành mẫu mặt cắt

	79	Số lượng các nét liền của đường nét tạo thành mẫu mặt cắt
	49	Chiều dài nét liền
	47	Kích thước của một pixel
	98	Số lượng các điểm dùng để dò tìm các đường biên kín
IMAGE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbRasterImage)
	90	Số phiên bản sử dụng
	10	Tọa độ điểm chèn (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ điểm chèn (trong OCS)
	30	Thành phần Z của tọa độ điểm chèn (trong OCS)
	11	Véc tơ U của pixel (trong OCS)
	21	Thành phần Y của véc tơ U của pixel (trong OCS)
	31	Thành phần Z của véc tơ U của pixel (trong OCS)
	12	Véc tơ V của pixel (trong OCS)
	22	Thành phần Y của véc tơ V của pixel (trọng OCS)
	32	Thành phần Z của véc tơ V của pixel (trong OCS)
	13	Kích thước hình ảnh, tính bằng pixel (gồm giá trị U và V)
	23	Giá trị V của kích thước hình ảnh, tính bằng pixel
	340	Tham khảo (cứng) đến đối tượng tạo hình ảnh
	70	Cách hiển thị hình ảnh 1 = Hiện hình ảnh 2 = Hiện hình ảnh không phụ thuộc vào cách canh lề của màn hình 3 = Sử dụng đường bao cắt 4 = Bật chế độ trong suốt
	280	Trạng thái cắt hình: 0 = off, 1 = on
	281	Độ sáng (0-100, mặc định = 50)
	282	Độ tương phản (0-100, mặc định = 0)
	283	Độ nhòa (0-100, mặc định = 0)
	360	Tham khảo (cứng) đến đối tượng *imagedef_reactor*
	71	Kiểu đường bao (1= hình chữ nhật, 2=hình đa giác)
	91	Số lượng các đỉnh của đường bao cắt
	14	Tọa độ đỉnh của đường bao cắt (trong OCS)

INSERT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbBlockReference)
	2	Tên khối
	10	Tọa độ điểm chèn của khối (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ điểm chèn của khối (biểu diễn trong OCS)
	30	Thành phần Z của tọa độ điểm chèn của khối (biểu diễn trong OCS)
	41	Hệ số chèn theo trục X (mặc định = 1)
	42	Hệ số chèn theo trục Y (mặc định = 1)
	43	Hệ số chèn theo trục Z (mặc định = 1)
	44	Khoảng cách giữa các cột (khi dùng lệnh Minsert chèn khối theo dãy; mặc định = 0)
	45	Khoảng cách giữa các hàng (khi dùng lệnh Minsert chèn khối theo dãy; mặc định = 0)
	50	Góc quay khi chèn khối (mặc định = 0)
	70	Số lượng các cột (mặc định = 1)
	71	Số lượng các hàng (mặc định =1)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
LEADER	100	Đánh dấu phân loại các mã DXF theo sau (AcDbLeader)
	3	Tên kiểu kích thước
	71	Cờ quy định việc xuất hiện mũi tên kích thước 0 = Không hiện mũi tên 1 = Hiện mũi tên
	72	Kiểu đường dẫn của kích thước 0 = Các phân đoạn đường thẳng 1 = Đường spline
	73	Cờ quy định việc tạo kích thước kiểu Leader 0 = Kèm theo chữ số kích thước 1 = Kèm theo ký hiệu dung sai 2 = Kèm theo một khối 3 = Không có chữ số kích thước

74	Xác định hướng của đường thẳng nằm dưới dòng chữ 0 = Ngược chiều với véc tơ xác định hướng nằm ngang của đường leader 1 = Cùng chiều
75	Xác định việc vẽ đường thẳng nằm dưới dòng chữ 0 = Không vẽ 1 = Vẽ
40	Chiều cao dòng chữ
41	Chiều rộng dòng chữ
76	Số lượng các đỉnh của đường kích thước leader
10	Tọa độ các đỉnh
20	Thành phần Y của tọa độ đỉnh
30	Thành phần Z của tọa độ đỉnh
77	Mã màu của đường Leader
340	Tham khảo (cứng) đến đối tượng được liên kết làm dòng chữ (mtext, dung sai hoặc khối)
210	Hướng véc tơ chuẩn (sử dụng khi đường leader trong không gian 3D)
220	Thành phần Y của của véc tơ chuẩn
230	Thành phần Z của của véc tơ chuẩn
211	Véc tơ xác định hướng nằm ngang của đường leader (khi đường leader trong không gian 3D)
221	Thành phần Y của véc tơ xác định hướng nằm ngang của đường leader
231	Thành phần Z của véc tơ xác định hướng nằm ngang của đường leader
212	Độ dời giữa điểm chèn của khối làm dòng chữ và điểm cuối cùng của đường leader
222	Thành phần Y của độ dời giữa điểm chèn của khối làm dòng chữ và điểm cuối cùng của đường leader
232	Thành phần Z của độ dời giữa điểm chèn của khối làm dòng chữ và điểm cuối cùng của đường leader
213	Độ dời giữa điểm chèn dòng chữ và điểm cuối cùng của đường leader
223	Thành phần Y của độ dời giữa điểm chèn dòng chữ và điểm cuối cùng của đường leader
233	Thành phần Z của độ dời giữa điểm chèn dòng chữ và điểm cuối cùng của đường leader

LINE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbLine)
	39	Chiều dày (mặc định = 0)
	10	Tọa độ điểm đầu của đường thẳng (biểu diễn trong hệ trục tọa độ WCS)
	20	Thành phần Y của tọa độ điểm đầu (trong WCS)
	30	Thành phần Z của tọa độ điểm đầu (trong WCS)
	11	Tọa độ điểm cuối của đường thẳng (trong WCS)
	21	Thành phần Y của tọa độ điểm cuối (trong WCS)
	31	Thành phần Z của tọa độ điểm cuối (trong WCS)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
LWPOLYLINE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbPolyline)
	90	Số lượng các đỉnh
	70	Cờ đánh dấu (mặc định = 0) 0 = Đa tuyến mở 1 = Đa tuyến đóng 128 = Kiểu Linetypegen bằng ON
	43	Chiều rộng (mặc định = 0)
	38	Cao độ (mặc định = 0)
	39	Chiều dày (mặc định = 0)
	10	Tọa độ đỉnh (trong WCS)
	20	Thành phần Y của tọa độ đỉnh (trong WCS)
	40	Chiều rộng ở đầu một phân đoạn (mặc định = 0)
	41	Chiều rộng ở cuối một phân đoạn (mặc định = 0)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
MLINE	100	Đánh dấu phân loại các mã DXF theo sau

2	Tên kiểu đường mline (tối đa 32 ký tự)
340	Mã đối tượng của kiểu đường mline chứa trong từ điển đối tượng
40	Hệ số tỉ lệ
70	Cách định vị trí đường mline 0 = Bằng đường bên trái (top) 1 = Bằng đường tâm (zero) 2 = Bằng đường bên phải (bottom)
71	Cờ xác định đường mline đóng hay mở 1 = Mở 3 = Đóng
72	Số lượng các đỉnh
73	Số lượng các đường nét thành phần tạo thành đường mline
10	Tọa độ điểm bắt đầu (trong WCS)
20	Thành phần Y của tọa độ điểm bắt đầu (trong WCS)
30	Thành phần Z của tọa độ điểm bắt đầu (trong WCS)
210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
11	Tọa độ các đỉnh
21	Thành phần Y của tọa độ đỉnh
31	Thành phần Z của tọa độ đỉnh
12	Véc tơ xác định hướng của phân đoạn bắt đầu tại đỉnh chứa trong mã DXF 11
22	Thành phần Y của véc tơ xác định hướng của phân đoạn bắt đầu tại đỉnh chứa trong mã DXF 11
32	Thành phần Z của véc tơ xác định hướng của phân đoạn bắt đầu tại đỉnh chứa trong mã DXF 11
13	Véc tơ xác định hướng của đường nét thành phần tại đỉnh chứa trong mã DXF 11
23	Thành phần Y của véc tơ xác định hướng của đường nét thành phần tại đỉnh chứa trong mã DXF 11

	33	Thành phần Z của véc tơ xác định hướng của đường nét thành phần tại đỉnh chứa trong mã DXF 11
	74	Số lượng các tham số cho thành phần này
	41	Các tham số của thành phần này
	75	Số lượng các tham số xác định miền được tô của thành phần này
	42	Các tham số xác định miền tô
MTEXT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbMText)
	10	Tọa độ điểm chèn
	20	Thành phần Y của tọa độ điểm chèn
	30	Thành phần Z của tọa độ điểm chèn
	40	Chiều cao mặc định của chữ
	41	Chiều rộng của hình chữ nhật bao
	3	Dòng chữ bổ sung
	7	Tên kiểu chữ (mặc định = STANDARD)
	1	Nội dung dòng chữ
	3	Dòng chữ bổ sung
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
	11	Véc tơ hướng trục X (trong WCS)
	21	Thành phần Y của véc tơ hướng trục X (trong WCS)
	31	Thành phần Z của véc tơ hướng trục X (trong WCS)
OLEFRAME	100	Đánh dấu phân loại các mã DXF theo sau (AcDbOleFrame)
	70	Số phiên bản OLE
	90	Chiều dài của chuỗi dữ liệu nhị phân
	310	Dữ liệu nhị phân
	1	Đánh dấu vị trí kết thúc dữ liệu nhị phân

OLE2FRAME	100	• Đánh dấu phân loại các mã DXF theo sau (AcDbOle2Frame)
	70	Số phiên bản OLE
	3	Chiều dài chuỗi dữ liệu nhị phân
	10	Tọa độ góc trái trên (trong WCS)
	20	Thành phần Y của tọa độ góc trái trên (trong WCS)
	30	Thành phần Z của tọa độ góc trái trên (trong WCS)
	11	Tọa độ góc phải dưới (trong WCS)
	21	Thành phần Y của tọa độ góc phải dưới (trong WCS)
	31	Thành phần Z của tọa độ góc phải dưới (trong WCS)
	71	Kiểu đối tượng OLE 1 = Link 2 = Embedded 3 = Static
	72	Xác định không gian bản vẽ mà đối tượng được chèn vào 0 = Không gian mô hình 1 = Không gian giấy vẽ
	90	Chiều dài của chuỗi dữ liệu nhị phân
	310	Chuỗi dữ liệu nhị phân
	1	Đánh dấu vị trí kết thúc dữ liệu nhị phân
POINT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbPoint)
	10	Tọa độ điểm (biểu diễn trong WCS)
	20	Thành phần Y của tọa độ điểm (trong WCS)
	30	Thành phần Z của tọa độ điểm (trong WCS)
	39	Chiều dày (mặc định = 0)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
	50	Góc xác định trục X của điểm khi vẽ (mặc định = 0)

POLYLINE	100	Đánh dấu phân loại các mã DXF theo sau (AcDb2dPolyline hoặc AcDb3dPolyline)
	10	Luôn luôn bằng 0
	20	Luôn luôn bằng 0
	30	Cao độ của đa tuyến (2D = OCS, 3D = WCS)
	39	Chiều dày (mặc định = 0)
	70	Cờ xác định trạng thái của đa tuyến (mặc định = 0) 1 = Đa tuyến đóng 2 = Đa tuyến được chuyển sang dạng Curve fit 4 = Đa tuyến được chuyển sang dạng Spline-fit 8 = Đa tuyến 3D 16 = Lưới cạnh đa tuyến 3D 32 = Lưới cạnh đa tuyến đóng theo hướng N 64 = Lưới mặt đa giác 128 = Đa tuyến có LineType gen bằng ON
	40	Chiều rộng ở đầu một phân đoạn (mặc định = 0)
	71	Số lượng các đỉnh của lưới đa tuyến theo hướng M (mặc định = 0)
	72	Số lượng các đỉnh của lưới đa tuyến theo hướng N (mặc định = 0)
	73	Độ mịn của bề mặt lưới theo hướng M (mặc định = 0)
	74	Độ mịn của bề mặt lưới theo hướng N (mặc định = 0)
	75	Kiểu đường cong và kiểu mặt cong (mặc định = 0) 0 = Không 5 = Mặt cong B-spline bậc 2 6 = Mặt cong B-spline bậc 3 8 = Mặt cong Bezier
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
RAY	100	Đánh dấu phân loại các mã DXF theo sau (AcDbRay)
	10	Tọa độ điểm đầu của đường thẳng (trong WCS)
	20	Thành phần Y của tọa độ điểm đầu (trong WCS)

	11	Véc tơ xác định hướng (trong WCS)
	21	Thành phần Y của véc tơ xác định hướng (trong WCS)
	31	Thành phần Z của véc tơ xác định hướng (trong WCS)
REGION	100	Đánh dấu phân loại các mã DXF theo sau (AcDbRegion)
	70	Số phiên bản sử dụng
	1	Các dữ liệu riêng của **AutoCAD**, không được cung cấp bản quyền sử dụng (chứa nhiều dòng, mỗi dòng ít hơn 255 ký tự)
	3	Các dòng bổ sung (trong trường hợp chuỗi chứa trong mã DXF 1 lớn hơn 255 ký tự)
SEQEND	-2	Tên của đối tượng trong cơ sở dữ liệu đối tượng
SHAPE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbShape)
	39	Chiều dày (mặc định = 0)
	10	Tọa độ điểm chèn (trong WCS)
	20	Thành phần Y của tọa độ điểm chèn (trong WCS)
	30	Thành phần Z của tọa độ điểm chèn (trong WCS)
	40	Kích thước
	2	Tên của Shape
	50	Góc quay khi chèn (mặc định = 0)
	41	Hệ số tỉ lệ theo trục X (mặc định = 1)
	51	Góc nghiêng (mặc định = 0)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
SOLID	100	Đánh dấu phân loại các mã DXF theo sau (AcDbSolid)
	10	Tọa độ điểm đầu tiên
	20	Thành phần Y của tọa độ điểm đầu tiên
	30	Thành phần Z của tọa độ điểm đầu tiên
	11	Tọa độ điểm thứ hai

	31	Thành phần Z của tọa độ điểm thứ hai
	12	Tọa độ điểm thứ ba
	22	Thành phần Y của tọa độ điểm thứ ba
	32	Thành phần Z của tọa độ điểm thứ ba
	13	Tọa độ điểm thứ tư
	23	Thành phần Y của tọa độ điểm thứ tư
	33	Thành phần Z của tọa độ điểm thứ tư
	39	Chiều dày (mặc định = 0)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
SPLINE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbSpline)
	210	Tọa độ điểm xác định hướng véc tơ chuẩn (không có nếu đường spline được vẽ trong không gian)
	220	Thành phần Y của tọa độ điểm xác định hướng véc tơ chuẩn
	230	Thành phần Z của tọa độ điểm xác định hướng véc tơ chuẩn
	70	Cờ xác định kiểu spline 1 = Đường spline đóng 2 = Đường spline tuần hoàn 4 = Đường spline hữu tỉ 8 = Đường spline phẳng 16 = Đường spline tuyến tính
	71	Bậc đường cong của đường spline
	72	Số lượng các *knot*
	73	Số lượng các điểm *control point*
	74	Số lượng các điểm *fit point*
	42	Dung sai của các nút (mặc định = 0.0000001)
	43	Dung sai các điểm *control point* (mặc định = 0.0000001)
	44	Dung sai các điểm *fit point* (mặc định = 0.0000000001)

	12	Tọa độ điểm xác định tiếp tuyến tại điểm đầu (trong WCS)
	22	Thành phần Y của tọa độ điểm xác định tiếp tuyến tại điểm đầu (trong WCS)
	32	Thành phần Z của tọa độ điểm xác định tiếp tuyến tại điểm đầu (trong WCS)
	13	Tọa độ điểm xác định tiếp tuyến tại điểm cuối (trong WCS)
	23	Thành phần Y của tọa độ điểm xác định tiếp tuyến tại điểm cuối (trong WCS)
	33	Thành phần Z của tọa độ điểm xác định tiếp tuyến tại điểm cuối (trong WCS)
	40	Giá trị của *knot*
	41	Trọng lượng của nút (chỉ xuất hiện nếu khác giá trị mặc định là 1)
	10	Tọa độ các điểm *control point* (trong WCS)
	20	Thành phần Y của tọa độ các điểm *control point* (trong WCS)
	30	Thành phần Z của tọa độ các điểm *control point* (trong WCS)
	11	Tọa độ các điểm *fit point*
	21	Thành phần Y của tọa độ các điểm *fit point* (trong WCS)
	31	Thành phần Z của tọa độ các điểm *fit point* (trong WCS)
TEXT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbText)
	10	Tọa độ điểm canh lề thứ nhất (biểu diễn trong hệ trục tọa độ đối tượng OCS)
	20	Thành phần Y của tọa độ điểm canh lề thứ nhất (biểu diễn trong OCS)
	30	Thành phần Z của tọa độ điểm canh lề thứ nhất (biểu diễn trong OCS)
	39	Chiều dày (mặc định = 0)
	40	Chiều cao chữ
	1	Nội dung dòng chữ
	50	Góc quay của dòng chữ (mặc định = 0)
	41	Hệ số tỉ lệ theo phương X (mặc định = 1)

	51	Góc nghiêng của chữ (mặc định = 0)
	7	Tên kiểu chữ (mặc định = STANDARD)
	71	Hướng tạo dòng chữ (mặc định = 0) 2 = Backward (đối xứng gương theo trục X) 4 = Upside down (đối xứng gương theo trục Y)
	72	Kiểu canh lề chữ theo phương ngang (mặc định = 0) 0 = Left 1 = Center 2 = Right 3 = Aligned 4 = Middle 5 = Fit
	73	Kiểu canh lề dòng chữ theo phương thẳng đứng (mặc định = 0) 0 = Baseline 1 = Bottom 2 = Middle 3 = Top
	11	Tọa độ điểm canh lề thứ hai (trong WCS)
	21	Thành phần Y của tọa độ điểm canh lề thứ hai (trong WCS)
	31	Thành phần Z của tọa độ điểm canh lề thứ hai (trong WCS)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
TOLERANCE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbFcf)
	3	Tên kiểu kích thước
	10	Tọa độ điểm chèn (trong WCS)
	20	Thành phần Y của tọa độ điểm chèn (trong WCS)
	30	Thành phần Z của tọa độ điểm chèn (trong WCS)
	11	Véc tơ xác định hướng trục X của hình biểu diễn dung sai (trong WCS)
	21	Thành phần Y của véc tơ xác định hướng trục X của hình biểu diễn dung sai (trong WCS)

	31	Thành phần Z của véc tơ xác định hướng trục X của hình biểu diễn dung sai (trong WCS)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
TRACE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbTrace)
	10	Tọa độ đỉnh đầu tiên (biểu diễn trong hệ trục tọa độ OCS)
	20	Thành phần Y của tọa độ đỉnh đầu tiên (trong OCS)
	30	Thành phần Z của tọa độ đỉnh đầu tiên (trong OCS)
	11	Tọa độ đỉnh thứ hai (biểu diễn trong hệ trục tọa độ OCS)
	21	Thành phần Y của tọa độ đỉnh thứ hai (trong OCS)
	31	Thành phần Z của tọa độ đỉnh thứ hai (trong OCS)
	12	Tọa độ đỉnh thứ ba (biểu diễn trong hệ trục tọa độ OCS)
	22	Thành phần Y của tọa độ đỉnh thứ ba (trong OCS)
	32	Thành phần Z của tọa độ đỉnh thứ ba (trong OCS)
	13	Tọa độ đỉnh thứ tư (biểu diễn trong hệ trục tọa độ OCS)
	23	Thành phần Y của tọa độ đỉnh thứ tư (trong OCS)
	33	Thành phần Z của tọa độ đỉnh thứ tư (trong OCS)
	39	Chiều dày (mặc định = 0)
	210	Tọa độ điểm xác định hướng trục Z của đối tượng (mặc định = (0 0 1))
	220	Thành phần Y của tọa độ điểm xác định hướng trục Z của đối tượng
	230	Thành phần Z của tọa độ điểm xác định hướng trục Z của đối tượng
VERTEX	100	Đánh dấu phân loại các mã DXF theo sau (AcDbVertex)

	100	Đánh dấu phân loại các mã DXF theo sau (AcDb2dVertex hoặc AcDb3dPolyline Vertex)
	10	Tọa độ điểm (2D = OCS, 3D = WCS)
	20	Thành phần Y của tọa độ điểm (2D = OCS, 3D = WCS)
	30	Thành phần Z của tọa độ điểm (2D = OCS, 3D = WCS)
	40	Chiều rộng đầu phân đoạn (mặc định = 0)
	41	Chiều rộng cuối phân đoạn (mặc định = 0)
	70	Cờ xác định (mặc định = 0) 1 = Đỉnh bổ sung để tạo đường đa tuyến kiểu *fit* 2 = Tiếp tuyến của đường đa tuyến kiểu *fit* 4 = Không sử dụng 8 = Đỉnh của spline khi đường đa tuyến có dạng spline 16 = Các đỉnh điều khiển khung của spline 32 = Đỉnh của đa tuyến 3D 64 = Đỉnh của lưới cạnh đa tuyến 3D 128 = Đỉnh của lưới mặt đa tuyến 3D
	50	Hướng tiếp tuyến của đường đa tuyến kiểu *fit*
	71	Số thứ tự đỉnh lưới mặt đa tuyến 3D (chỉ xuất hiện khi giá trị này khác 0)
	72	Số thứ tự đỉnh lưới mặt đa tuyến 3D (chỉ xuất hiện khi giá trị này khác 0)
	73	Số thứ tự đỉnh lưới mặt đa tuyến 3D (chỉ xuất hiện khi giá trị này khác 0)
	74	Số thứ tự đỉnh lưới mặt đa tuyến 3D (chỉ xuất hiện khi giá trị này khác 0)
VIEWPORT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbViewport)
	10	Tọa độ điểm tâm (trong WCS)
	20	Thành phần Y của tọa độ điểm tâm (trong WCS)
	30	Thành phần Z của tọa độ điểm tâm (trong WCS)
	40	Chiều rộng (theo đơn vị tính của không gian giấy vẽ)
	41	Chiều cao (theo đơn vị tính của không gian giấy vẽ)
	68	Trạng thái khung nhìn (1 = On, 0 = off)
	69	Mã số thứ tự của khung nhìn (thay đổi mỗi lần mở

XLINE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbXline)
	10	Tọa độ điểm đầu tiên (trong WCS)
	20	Thành phần Y của tọa độ điểm đầu tiên (trong WCS)
	30	Thành phần Z của tọa độ điểm đầu tiên (trong WCS)
	11	Véc tơ xác định hướng (trong WCS)
	21	Thành phần Y của véc tơ xác định hướng (trong WCS)
	31	Thành phần Z của véc tơ xác định hướng (trong WCS)

Bảng I.4 Mã DXF của các đối tượng trong từ điển đối tượng

Kiểu đối tượng	Mã DXF	Ý nghĩa
DICTIONARY	-1	Mã đối tượng
	0	Kiểu đối tượng
	5	Giá trị *handle*
	100	Đánh dấu phân loại các mã DXF theo sau (AcDbDictionary)
	3	Đánh dấu vị trí bắt đầu của một nhóm đối tượng
	350	Mã đối tượng chính bắt đầu một nhóm
ACAD_GROUP	-1	Mã đối tượng
	0	Kiểu đối tượng
	5	Giá trị *handle*
	102	Bắt đầu chuỗi đánh dấu, luôn luôn là "{ACAD_REACTORS"
	330	Mã đối tượng của từ điển đối tượng
	102	Kết thúc chuỗi đánh dấu, luôn luôn là "}"
	100	Đánh dấu phân loại các mã DXF theo sau (AcDbGroup)
	300	Chuỗi mô tả Group

	70	Cờ xác định Group có được đặt tên chưa 1 = Chưa đặt tên 0 = Đã đặt tên
	71	Cờ xác định Group có chọn được hay không 1 = Chọn được 2 = Không chọn được
	340	Giá trị handle của Group
ACAD_MLINE STYLE	-1	Mã đối tượng
	0	Kiểu đối tượng
	5	Giá trị *handle*
	102	Bắt đầu chuỗi đánh dấu, luôn luôn là "{ACAD_REACTORS"
	330	Mã đối tượng của từ điển đối tượng
	102	Kết thúc chuỗi đánh dấu, luôn luôn là "}"
	100	Đánh dấu phân loại các mã DXF theo sau (AcDbMlineStyle)
	2	Tên kiểu đường mline
	70	Cờ xác định 1 = Tô màu đường mline 2 = Hiện các đường nối giữa các phân đoạn 16 = Nối điểm đầu của đường mline bằng đoạn thẳng 32 = Nối các nét bên trong tại điểm đầu của đường mline bằng cung tròn 64 = Nối các nét bên trong tại điểm đầu của đường mline bằng cung tròn 256 = Nối điểm cuối của đường mline bằng đoạn thẳng 512 = Nối các nét bên trong tại điểm cuối của đường mline bằng cung tròn 1024 =Nối các nét bên trong tại điểm cuối của đường Mline bằng cung tròn
	3	Chuỗi mô tả kiểu đường mline
	62	Màu tô của đường nét thành phần
	51	Góc bắt đầu phân đoạn
	52	Góc kết thúc phân đoạn
	71	Số lượng các đường nét thành phần
	49	Độ dời của đường nét thành phần so với đường tâm
	62	Màu của đường nét thành phần

6	Dạng đường của đường nét thành phần

Bảng I.5 Mã DXF dùng cho tất cả các bảng mô tả

Mã DXF	Ý nghĩa
-1	Mã đối tượng (thay đổi mỗi lần bản vẽ được mở lại)
0	Kiểu đối tượng của bảng mô tả
2	Tên bảng mô tả
5	Giá trị *handle* của đối tượng
100	Đánh dấu phân loại các mã DXF theo sau (AcDbSymbolTable)
70	Số lượng tối đa các đối tượng chứa trong bảng mô tả

Bảng I.6 Mã DXF dùng riêng cho từng bảng mô tả

Kiểu đối tượng	Mã DXF	Ý nghĩa
APPID	100	Đánh dấu phân loại các mã DXF theo sau (AcDbRegAppTableRecord)
	2	Tên ứng dụng được đăng ký
	70	Các giá trị cờ tiêu chuẩn
BLOCK	100	Đánh dấu phân loại các mã DXF theo sau (AcDbBlockTableRecord)
	2	Tên khối
	70	Cờ xác định kiểu khối 1 = Khối không tên 2 = Chứa các thuộc tính theo sau 4 = XREF 8 = Không sử dụng 16 = Phụ thuộc ngoài 32 = Phụ thuộc XREF 64 = Đã có khối chèn vào bản vẽ
	1	Đường dẫn của XREF
DIMSTYLE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbDimStyleTableRecord)
	2	Tên kiểu kích thước
	70	Các cờ tiêu chuẩn
	170	DIMALT

171	DIMALTD
143	DIMALTF
274	DIMALTTD
286	DIMALTTZ
273	DIMALTU
285	DIMALTZ
4	DIMAPOST
41	DIMASZ
275	DIMAUNIT
5	DIMBLK
6	DIMBLK1
7	DIMBLK2
141	DIMCEN
176	DIMCLRD
177	DIMCLRE
178	DIMCLRT
271	DIMDEC
46	DIMDLE
43	DIMDLI
44	DIMEXE
42	DIMEXO
287	DIMFIT
147	DIMGAP
280	DIMJUST
144	DIMLFAC
72	DIMLIM
3	DIMPOST
45	DIMRND
173	DIMSAH
40	DIMSCALE
281	DIMSD1
282	DIMSD2
75	DIMSE1
76	DIMSE2

	77	DIMTAD
	272	DIMTDEC
	146	DIMTFAC
	73	DIMTIH
	174	DIMTIX
	48	DIMTM
	172	DIMTOFL
	74	DIMTOH
	71	DIMTOL
	283	DIMTOLJ
	47	DIMTP
	142	DIMTSZ
	145	DIMTVP
	340	DIMTXSTY
	140	DIMTXT
	284	DIMTZIN
	270	DIMUNIT
	288	DIMUPT
	78	DIMZIM
LAYER	100	Đánh dấu phân loại các mã DXF theo sau (AcDbSymbolTableRecord)
	2	Tên lớp bản vẽ
	70	Các cờ tiêu chuẩn 1 = Bị đóng băng 2 = Bị đóng băng khi tạo khung nhìn mới 3 = Bị khóa
	62	Mã màu
	6	Dạng đường
LTYPE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbLineTypeTableRecord)
	2	Tên dạng đường
	70	Các cờ tiêu chuẩn
	3	Chuỗi mô tả dạng đường
	72	Mã *Alignment* (luôn luôn bằng 65)
	73	Số lượng các nét gạch tạo thành dạng đường
	40	Chiều dài tổng cộng của mẫu dang đường

	49	Chiều dài của nét gạch, dấu chấm hoặc khoảng trắng
	74	Kiểu phần tử của dạng đường 0 = Bình thường 2 = Dòng chữ 3 = hình ảnh shape
	75	Mã shape
	340	Con trỏ chỉ đến đối tượng kiểu chữ STYLE
	46	S = Hệ số tỉ lệ
	50	R = Góc quay
	44	X = Độ dời theo trục X
	45	Y = Độ dời theo trục Y
	9	Nội dung dòng chữ
STYLE	100	Đánh dấu phân loại các mã DXF theo sau (AcDbTextStyleTableRecord)
	2	Tên kiểu chữ
	70	Các cờ tiêu chuẩn 1 = Mô tả hình shape 4 = Dòng chữ viết theo phương thẳng đứng
	40	Chiều cao chữ
	41	Hệ số tỉ lệ của chiều rộng
	50	Góc nghiêng của chữ
	71	Hướng tạo dòng chữ (mặc định = 0) 2 = *Backward* (đối xứng gương theo trục X) 4 = *Upside down* (đối xứng gương theo trục Y)
	42	Chiều cao chữ vừa mới được sử dụng
	3	Tên *font file* chính
	4	Tên *big font file*
UCS	100	Đánh dấu phân loại các mã DXF theo sau (AcDbUCSTableRecord)
	2	Tên UCS
	70	Các giá trị cờ tiêu chuẩn
	10	Gốc tọa độ (biểu diễn trong WCS)
	20	Thành phần Y của gốc tọa độ (trong WCS)
	30	Thành phần Z của gốc tọa độ (trong WCS)
	11	Véc tơ xác định hướng trục X (trong WCS)

	21	Thành phần Y của véc tơ xác định hướng trục X (trong WCS)
	31	Thành phần Z của véc tơ xác định hướng trục X (trong WCS)
	12	Véc tơ xác định hướng trục Y (trong WCS)
	22	Thành phần Y của véc tơ xác định hướng trục Y (trong WCS)
	32	Thành phần Z của véc tơ xác định hướng trục Y (trong WCS)
VIEW	100	Đánh dấu phân loại các mã DXF theo sau (AcDbViewTableRecord)
	2	Tên phần ảnh View
	70	Các cờ tiêu chuẩn (1 = Không gian giấy vẽ)
	40	Chiều cao phần ảnh View
	10	Tọa độ điểm tâm (2D)
	20	Thành phần Y của tọa độ điểm tâm
	41	Chiều rộng
	11	Hướng nhìn từ tiêu điểm (trong WCS)
	21	Thành phần Y của hướng nhìn từ tiêu điểm (trong WCS)
	31	Thành phần Z của hướng nhìn từ tiêu điểm (trong WCS)
	12	Tọa độ tiêu điểm (trong WCS)
	22	Thành phần Y của tọa độ tiêu điểm (trong WCS)
	32	Thành phần Z của tọa độ tiêu điểm (trong WCS)
	42	Tiêu cự (trong hình chiếu phối cảnh)
	43	Mặt phẳng cắt trước (trong hình chiếu phối cảnh)
	44	Mặt phẳng cắt sau (trong hình chiếu phối cảnh)
	50	Góc quay ống kính (trong hình chiếu phối cảnh)
	71	View mode
VPORT	100	Đánh dấu phân loại các mã DXF theo sau (AcDbViewportTableRecord)
	2	Tên khung nhìn
	70	Các cờ tiêu chuẩn
	10	Tọa độ góc trái dưới (2D)
	20	Thành phần Y của tọa độ góc trái dưới (2D)
	11	Tọa độ góc phải trên (2D)

21	Thành phần Y của tọa độ góc phải trên (2D)
12	Tọa độ điểm tâm (2D)
22	Thành phần Y của tọa độ điểm tâm
13	Điểm gốc của snap (2D)
23	Thành phần Y của điểm gốc snap
14	Bước nhảy snap X và Y
24	Thành phần Y của của bước nhảy snap
15	Khoảng cách lưới X và Y
25	Thành phần Y của khoảng cách lưới
16	Hướng nhìn từ tiêu điểm (trong WCS)
26	Thành phần Y của hướng nhìn từ tiêu điểm
36	Thành phần Z của hướng nhìn từ tiêu điểm
17	Tiêu điểm (trong WCS)
27	Thành phần Y của tiêu điểm
37	Thành phần Z của tiêu điểm
40	Chiều cao
41	Tỉ lệ của khung nhìn
42	Tiêu cự
43	Mặt phẳng cắt trước (trong hình chiếu phối cảnh)
44	Mặt phẳng cắt sau (trong hình chiếu phối cảnh)
50	Góc quay của *snap*
51	Góc quay ống kính (trong hình chiếu phối cảnh)
71	Viewmode
74	Biểu tượng UCSICON
75	Bật tắt Snap
76	Bật tắt lưới
77	Kiểu Snap
78	Snap kiểu hình chiếu trục đo

BẢNG TRA CỨU HÀM AutoLISP, Tập 2

Tên hàm	Ý nghĩa	Trang
(**Action_tile** KEY ACTION_EXPRESSION)	Gán cho *tile* một biểu thức **AutoLISP**.	230
(**Add_list** ITEM)	Thay đổi hoặc thêm mục chọn mới vào danh sách tùy theo cách mở danh sách bằng hàm **Start_list**.	242
(**Close** FILE-DESC)	Đóng file đang mở có thẻ file là FILE-DESC.	59
(**Dictadd** ENAME SYMBOL NEWOBJ)	Thêm đối tượng mới (Group hoặc kiểu đường Mline) vào từ điển đối tượng.	109
(**Dictnext** ENAME [REWIND])	Duyệt từng phần tử kiểu "DICTIONARY" trong từ điển đối tượng.	100
(**dictremove** ENAME SYMBOL)	Loại bỏ một đối tượng của một nhóm trong từ điển đối tượng.	111
(**Dictrename** ENAME OLDSYM NEWSYM)	Đổi tên một đối tượng của nhóm trong tự điển đối tượng.	106
(**Dictsearch** ENAME SYMBOL)	Tìm phần tử kiểu "DICTIONARY" trong từ điển đối tượng	102
(**done_dialog** [STATUS])	Đóng hộp thoại.	223
(**End_image**)	Kết thúc quá trình khởi tạo và điền hình ảnh cho *image tile*.	245
(**end_list**)	Kết thúc quá trinh tạo hoặc chỉnh sửa danh sách.	242

(**Entget** ENAME [APPLIST])	Trả về danh sách biểu diễn *record* dữ liệu của đối tượng có mã ENAME.	12
(**Entlast**)	Trả về mã đối tượng chính cuối cùng (không bị đánh dấu xóa) trong cơ sở dữ liệu.	18
(**Entmake** [ELIST])	Tạo đối tượng mới trực tiếp trong cơ sở dữ liệu.	36
(**Entmode** ELIST)	Thay thế *record* cũ trong cơ sở dữ liệu bằng *record* mới.	32
(**Entnext** [ENAME])	Lấy ra mã đối tượng (không bị đánh dấu xóa) kế tiếp đối tượng có mã là ENAME trong cơ sở dữ liệu.	10
(**Entsel** [PROMPT])	Cho phép người sử dụng chọn đối tượng bằng cách chọn một điểm trên màn hình bằng chuột hoặc nhập tọa độ điểm chọn.	21
(**Entupd** ENAME)	Cập nhật lại đối tượng ENAME trên màn hình.	33
(**Fill_image** X Y WID HGT COLOR)	Vẽ một hình chữ nhật trong phạm vi của *image tile* và tô màu cho hình chữ nhật này.	245
(**Findfile** FILENAME)	Tìm kiếm file có trong các thư mục hay không và trả về đường dẫn thư mục của file này.	52
(**Get_tile** KEY)	Lấy ra giá trị hiện hành của *tile* có khóa chứa trong tham số KEY.	235
(**Getcfg** CFGNAME)	Đọc dữ liệu từ *section* AppData của file acad14.cfg.	68
(**Getenv** VARIABLE-NAME)	Trả về giá trị của biến VARIABLE-NAME.	69
(**Getfiled** TITLE DEFAULT EXT FLAGS)	Làm xuất hiện hộp thoại cho phép chọn một file có sẵn, hoặc chọn đường dẫn cho file mới.	53
(**Graphscr**)	Chuyển từ màn hình văn bản sang màn hình đồ họa.	134
(**Grclear**)	Che các đối tượng trên khung nhìn hiện hành.	143
(**Grdraw** FROM TO COLOR [HIGHLIGHT])	Vẽ một véc tơ đi qua 2 điểm trên màn hình trong hệ trục UCS hiện	145

	hành.	
(**Grread** [TRACK] [ALLKEYS [CURTYPE]])	Nhận giá trị nhập từ các thiết bị nhập.	159
(**Grtext** [BOX TEXT [HIGHLIGHT])	Viết một chuỗi lên dòng trạng thái hoặc lên menu màn hình.	153
(**Grvecs** VLIST [TRANS])	Vẽ nhiều véc tơ trên màn hình cùng lúc.	150
(**Handent** HANDLE)	Trả về mã đối tượng có giá trị *handle* chứa trong tham số HANDLE.	19
(**Load_dialog** DCLFILE)	Mở file DCL và chuyển các dòng mô tả hộp thoại và các tile của hộp thoại vào bộ nhớ.	220
(**Menucmd** STRING)	Làm xuất hiện các menu trên màn hình	138
(**Menugroup** GROUPNAME)	Kiểm tra một *menu group* đã được tải hay chưa.	143
(**Mode_tile** KEY STATE)	Thay đổi trạng thái của các *tile* trong khi chạy chương trình.	233
(**Namedobjdict**)	Trả về mã đối tượng của từ điển đối tượng trong bản vẽ hiện hành.	96
(**Nentsel** [PROMPT])	Cho phép chọn trực tiếp một đối tượng thuộc tính trong một khối hoặc một đỉnh trong một đa tuyến.	22
(**New_dialog** DLGNAME DCL_ID [ACTION-EXPRESSION [SCREEN-PT]])	Khởi tạo một hộp thoại có cấu trúc được mô tả bằng các mã DCL đã được tải vào bộ nhớ.	221
(**Open** FILENAME MODE)	Mở file	57
(**Read-char** [FILE-DESC])	Đọc một ký tự của một *file văn bản ASSCII* có thẻ file là FILE-DESC và trả về mã ASSCII của ký tự này.	63
(**Read-line** [FILE-DESC])	Đọc dữ liệu của một *file văn bản ASSCII* có thẻ file là FILE-DESC.	57
(**Redraw** [ENAME [MODE]])	Làm nổi bật (*highlight*) hoặc vẽ lại (*redraw*) các đối tượng được chọn trên màn hình.	134
(**Set_tile** KEY VALUE)	Gán giá trị ban đầu cho các *tile* hoặc thay đổi giá trị trong thời gian chạy	228

	chương trình.	
(**Setcfg** CFGNAME CFGVAL)	Chép dữ liệu vào *section* AppData của file acad14.cfg.	66
(**Setview** VIEW_DESCRIPTOR [VPOTRS_ID])	Gọi hiển thị một phần ảnh *(View)* lên màn hình hoặc khung nhìn.	122
(**Slide_image** X1 Y1 WID HGT SLDNAME)	Hiện *slide file* trên các *image tile*.	253
(**Snvalid** SYM_NAME [FLAG])	Kiểm tra chuỗi chứa trong tham số SYM_NAME, biểu diễn tên đối tượng chứa trong các bảng mô tả, có hợp lệ hay không.	88
(**Start_dialog**)	Làm xuất hiện hộp thoại được tạo ra bằng hàm **New_dialog** và bắt đầu khởi động hộp thoại làm việc.	223
(**Start_image** KEY)	"Mở" *image tile* để điền hình ảnh.	245
(**Start_list** KEY [OPERATION [INDEX]])	Khởi tạo một danh sách hoặc mở danh sách có sẵn để thêm hoặc thay đổi các mục chọn cho *list_box* hoặc *popup_list*.	241
(**Subst** NEW_ITEM OLD_ITEM LIST)	Thay thế mọi phần tử OLD_ITEM trong *record* LIST bằng NEW_ITEM và trả về *record* mới này.	31
(**Tblnext** TABLE-NAME [REWIND])	Duyệt từng phần tử trong bảng mô tả.	76
(**Tblobjname** TABLE-NAME SYMBOL)	Tìm một đối tượng trong bảng mô tả. Giá trị trả về là mã đối tượng.	87
(**Tblsearch** TABLE-NAME SYMBOL [SETNEXT])	Tìm một đối tượng trong bảng mô tả. Nếu tìm thấy sẽ trả về toàn bộ *record* dữ liệu này.	84
(**Term_dialog**)	Đóng tất cả các hộp thoại.	225
(**Textbox** ELIST)	Trả về tọa độ 2 đỉnh (góc trái dưới và góc phải trên) của hình chữ nhật bao dòng chữ.	41
(**Textpage**)	Chuyển từ màn hình đồ họa sang màn hình văn bản.	134
(**Textscr**)	Chuyển từ màn hình đồ họa sang màn hình văn bản.	134

(**Trace** FUNCTION . . .)	Đánh dấu các hàm cần theo dõi kết quả.	276
(**Unload_dialog** DCL-ID)	Loại bỏ các mã lệnh của file DCL ra khỏi bộ nhớ.	225
(**Untrace** FUNCTION . . .)	Loại bỏ việc đánh dấu các hàm trong chương trình.	277
(**Vector_image** X1 Y1 X2 Y2 COLOR)	Vẽ trực tiếp các véc tơ lên *image tile.*	247
(**Ver**)	Cho biết phiên bản **AutoLISP** đang sử dụng.	68
(**Vports**)	Trả về danh sách biểu diễn cách sắp xếp các khung nhìn	115
(**Wcmatch** STRING PATTERN)	So sánh một chuỗi STRING có thỏa mãn chuỗi ký tự đại diện PATTERN hay không.	91
(**Write-char** NUM [FILE-DESC])	Chuyển một mã ASCII thành một ký tự và chép nó vào một file đang mở.	63
(**Write-line** STRING [FILE-DESC])	Chép chuỗi STRING ra một *file văn bản ASSCII* có thẻ file là FILE-DESC.	59

Lập trình thiết kế với AutoLISP và Visual LISP

Tập 2

NGUYỄN HỮU LỘC – NGUYỄN THANH TRUNG

Chịu trách nhiệm xuất bản : Trần Đình Việt
Biên tập : Đức Nhân – Trung Hiếu
Sửa bản in : Dương Ly – Thiên Hương
Trình bày : Hữu Lộc, Thành Trung
Bìa : Mai Quế Vũ

NHÀ XUẤT BẢN THÀNH PHỐ HỒ CHÍ MINH

62 Nguyễn Thị Minh Khai, Quận 1.
Điện thoại: 8225340 – 8296764 – 8220405 – 8222762 – 8296731- 8223637- 8250616. **FAX**: 84.8.8222726
Email: nxbtphcm@bdvn.vnd.net

In lần thứ hai, Số lượng: 1000 cuốn, khổ 16x24 cm.
Tại: Xí nghiệp In số 5.
Số XB: 399-196/XB-QLXB cấp ngày: 11-04-2003
In xong và nộp lưu chiểu: 08-2003.