PHẦN III

GIỚI THIỆU NGÔN NGỮ AUTOLISP TRONG AUTOCAD 2000

AutoLISP là ngôn ngữ lập trình có thể sử dụng các tính năng, các lệnh trong AutoCAD. AutoLISP dựa trên cơ sở ngôn ngữ lập trình LISP- ngôn ngữ lập trình trí tuệ nhân tạo(Artificial Interligence-AI).

Visual LISP (VLISP) là một phần mềm giúp phát triển chương trình AutoLISP. Nó giúp ta thực hiện hầu hết các thao tác cần thiết như soạn thảo, gỡ rối chương trình, giao tiếp với AutoCAD mà không cần thoát ra khỏi AutoCAD.

CHƯƠNG 28
MÔI TRƯỜNG PHÁT TRIỂN GIAO TIẾP AUTOLISP (VISUAL LISP)

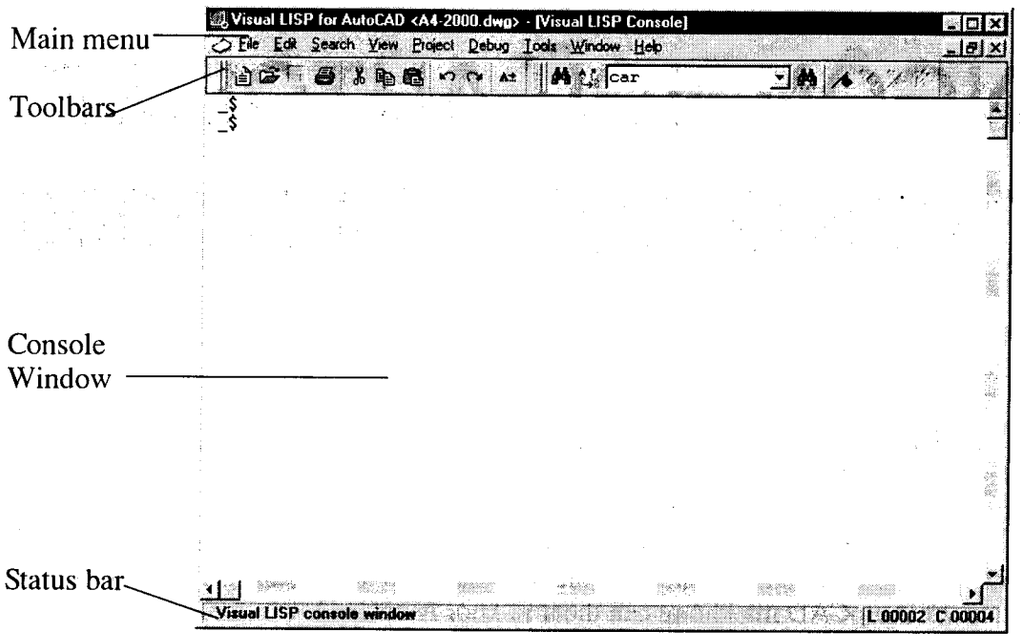
VLISP bao gồm các cửa sổ với các Menu riêng để hoạt động; nhưng VLISP chỉ hoạt động được khi AutoCAD khởi động. Cũng chú ý rằng một khi VLISP điều khiển thu nhỏ tối thiểu AutoCAD, thì ta phải kích hoạt cho cửa sổ AutoCAD hoạt động trở lại.

28.1. KHỞI ĐỘNG VISUAL LISP

Đang ở trong môi trường AutoCAD, để khởi động môi trường phát triển giao tiếp VLISP ta có 2 cách như sau:

1).Pull-down menu: Tools> AutoLISP> Visual LISP Editor
2).Gõ lệnh: **Vlisp**

Sau khi gọi lệnh sẽ xuất hiện màn hình **Visual LISP for AutoCAD** (hình 28.1).



Hình 28.1: Màn hình **Visual LISP for AutoCAD.**

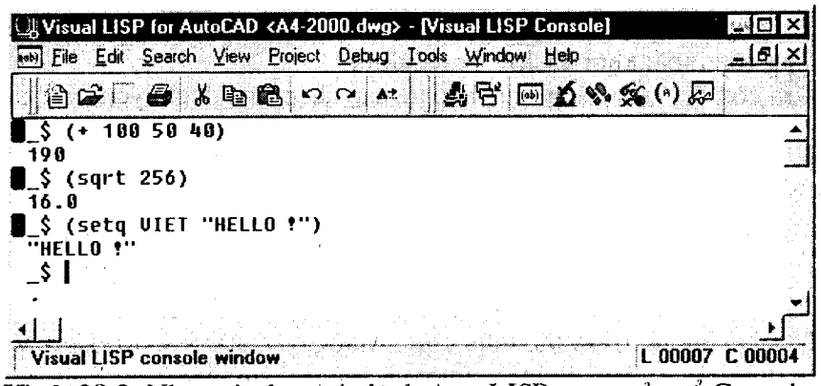
- Toolbars: VLISP có các thanh công cụ **Toolbars** như sau: *Standard, Debug, View, Search, Tools*. Để mở/đóng các **Toolbars** ta gọi: ***View> Toolbars*** và chọn thanh công cụ cần mở.

- Console Window: để nhập các lệnh của AutoLISP.

- Status bar: hiển thị các thông tin tóm tắt.

28.2. CONSOLE WINDOW

Trong cửa sổ Console (Console Window, hình 28.1) ta có thể nhập và chạy các lệnh của AutoLISP và xem kết quả(hình 28.2).



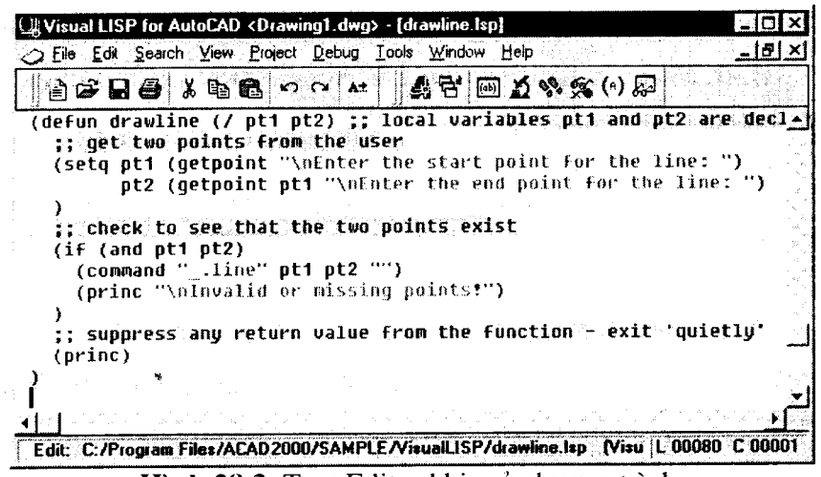
Hình 28.2: Nhập và chạy các lệnh AutoLISP trong cửa sổ Console.

Các tính năng thao tác của Console Window

- CTRL+ENTER: để nhập 1 biểu thức trên nhiều dòng.
- TAB: để truy xuất lại các lệnh đã nhập trong Console Window. Nếu ấn SHIFT+TAB thì đổi hướng truy xuất ngược lại.
- ESC: xoá các Text sau dấu nhắc Console(_∃).
- SHIFT+ESC: để lại Text bất kỳ sau dấu nhắc Console(_∃).
- Ấn phím phải chuột, hoặc SHIFT+F10: để hiện *Shortcut Menu*.

28.3. CỬA SỔ VISUAL LISP TEXT EDITOR

Vissual LISP Text Editor là công cụ để soan thảo, công cụ chính của môi trường phát triển giao tiếp VLISP. Để xem cửa sổ **Text Editor**, ta hãy mở 1 chương trình(hình 28.3).



Hình 28.3: Text Editor khi mở chương trình.

Các tính năng chủ yếu của **Visual LISP Text Editor**

- Tạo mã màu: các phần khác nhau của chương trình AutoLISP được VLISP Text Editor phân biệt và gán các mã màu khác nhau. Điều đó giúp ta tìm kiếm các phần của chương trình, các biến và lỗi chương trình một cách dễ dàng.

- Ghép đôi các dấu ngoặc trong chương trình: VLISP Text Editor có thể giúp ta dò tìm dấu ngoặc đơn còn thiếu bằng cách dò tìm cặp dấu ngoặc đóng liên kết với dấu ngoặc mở.

- Kiểm tra lỗi cú pháp mã AutoLISP: VLISP Text Editor còn có thể làm bật sáng các lỗi cú pháp trong chương trình.

- Thực hiện các biểu thức: VLISP Text Editor cho phép ta thử chạy các biểu thức mà không cần thoát khỏi VLISP Text Editor.

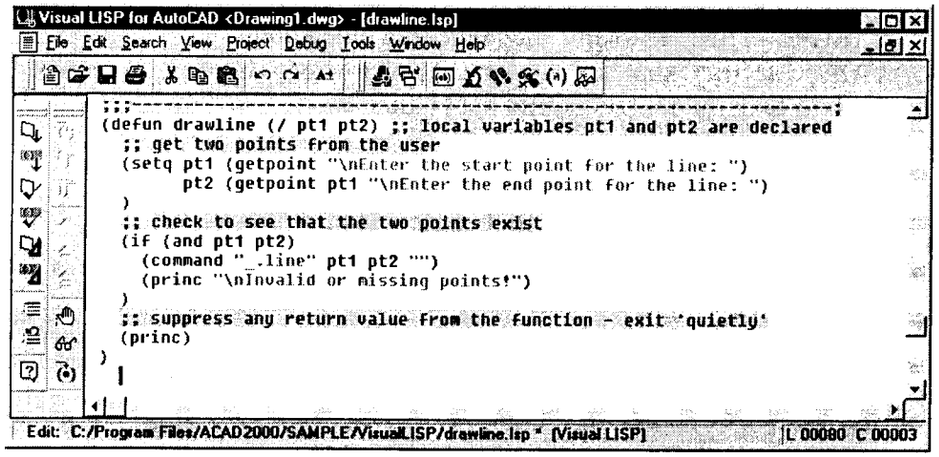
Ngoài ra còn các tính năng sao chép từ, sao chép file, di chuyển, tìm kiếm file …

28.4. TẢI VÀ CHẠY CHƯƠNG TRÌNH AUTOLISP

28.4.1. Chạy một chương trình

Trong VLISP Text Editor, nếu ta mở một chương trình AutoLISP thì ta có thể tải và chạy nó ngay tại đây. Ví dụ ta mở chương trình có tên là ***Drawline.LSP*** có trong thư mục SAMPLE của AutoCAD và chạy chương trình. Ta tiến hành theo trình tự sau:

1). Mở file chương trình theo địa chỉ: AutoCAD2000> SAMPLE> VisualLISP> ***drawline.lsp*** (hình 28.4). Sau đó kích chuột vào vùng soạn thảo Text Editor.

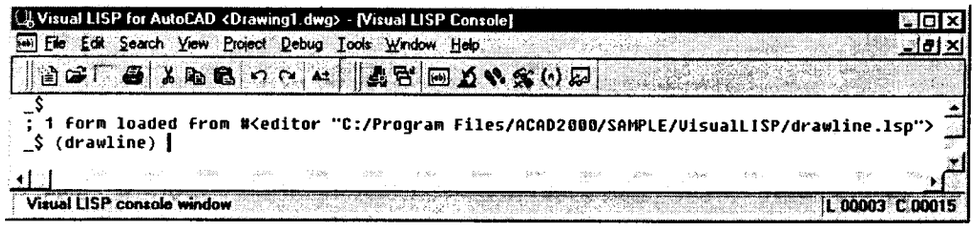


Hình 28.4: Mở file *Drawline.LSP* trong VLISP Text Editor.

2). Tải chương trình bằng cách:

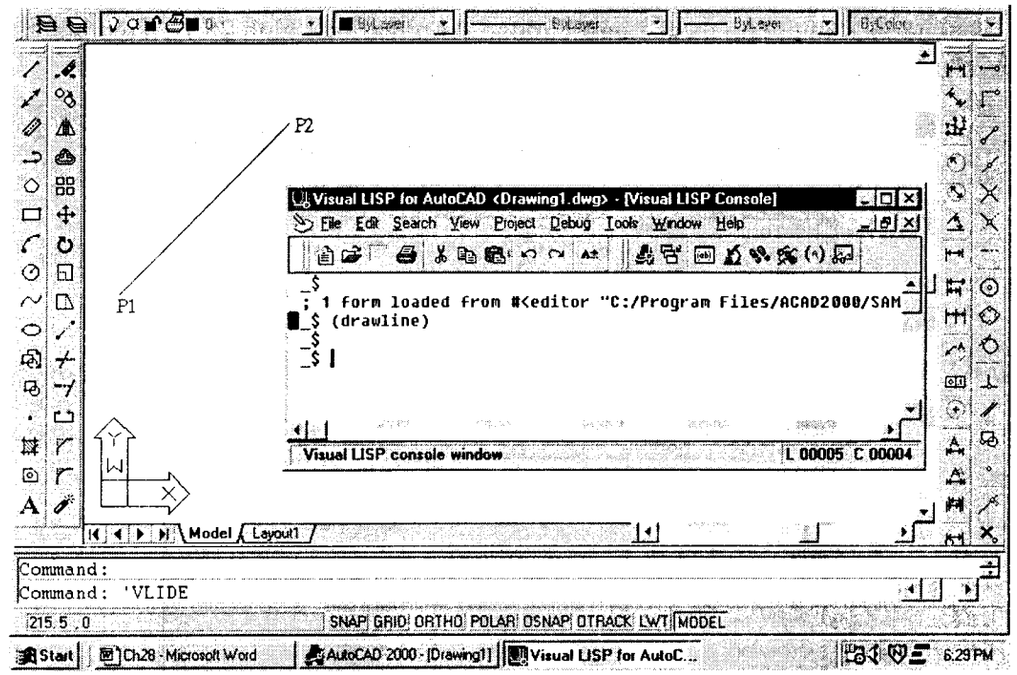
1).Pull-down menu: Tools> Load Text in Editor
2).Thanh Tools: chọn nút ***Load Active Edit Window:***

3). Chạy chương trình bằng cách nhập tên hàm trong dấu ngoặc đơn từ dấu nhắc (_∃) của Console Window (hình 28.5), tương tự như sau dấu nhắc (Command:) của AutoCAD.



Hình 28.5: Nhập tên hàm ***(drawline)*** trong Console Window.

4). Sau khi nhập tên hàm và ENTER thì cửa sổ Console thu lại để ta vẽ đoạn thẳng P1P2 trên màn hình AutoCAD. Vẽ xong P1P2 thì cửa sổ Console hiện trở lại(hình 28.6).



Hình 28.6

*** Chú ý:** để chuyển đổi giữa các cửa sổ AutoCAD và VLISP, ta chỉ việc kích chuột vào nút ***AutoCAD2000*** hoặc nút ***Visual LISP*** ở dòng đáy màn hình:

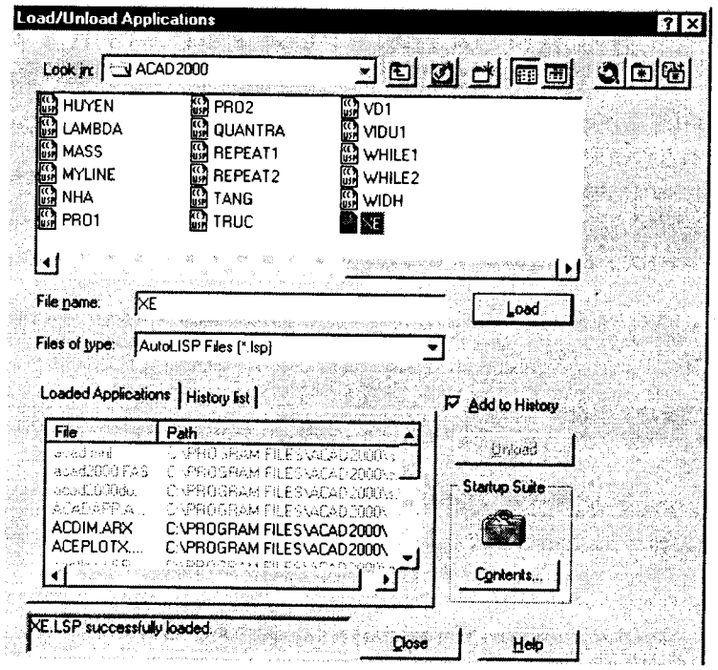
*** Cách thứ 2 tải và chạy chương trình AutoLISP tại cửa sổ AutoCAD:**

Ví dụ:

```
;; Chương trình vẽ xe đẩy (hình 28.8)
(Defun  C:XE ( )    ;; định nghĩa hàm và gán(nhập) dữ liệu
 (SETQ P1(GETPOINT  "\nCHO DINH 1 KHUNG XE-TAM BANH 1:"))
 (SETQ P2(GETPOINT  "\nCHO DINH 2 KHUNG XE-TAM BANH 2:"  P1))
 (SETQ P3(GETPOINT  "\nCHO DINH 3 KHUNG XE:"  P2))
 (SETQ P4(GETPOINT  "\nCHO DINH 4 KHUNG XE:"  P3))
 ;; sử dụng hàm commad của AutoLISP với các biến là các lệnh của AutoCAD:
 (COMMAND "PLINE" P1 P2 P3 P4 "c")
 (Command "PEDIT" Pause "W" "1.4" "")   ;; đổi bề rộng nét 1.4mm
 (command "COLOR" 5)    ;; đặt màu Blue vẽ các vòng tròn 2 bánh xe
 (COMMAND "CIRCLE" P1 PAUSE "CIRCLE" P1 PAUSE "CIRCLE" P2
                                          PAUSE "CIRCLE" P2 PAUSE)
 (command "COLOR" 3)     ;; đặt màu Green gạch trang trí cho thân xe
 (COMMAND "HATCH" "U" "90" "3" "N" PAUSE "")  ;; đặt màu Green
                                          ;; để gạch trang trí cho thân xe
 (command "COLOR" 1)       ;; đặt màu Red để gạch trang trí cho các bánh xe
 (COMMAND "HATCH" "U" "45" "2" "Y" PAUSE)
          ;; chọn đường bao thân xe và các vòng tròn bánh xe để gạch trang trí.
) ;; hết defun ----------
```

1). Tải chương trình: **Menu AutoCAD> Tools> Load Application**

Khi đó xuất hiện hộp thoại **Load/Unload Applications**(hình 28.7). Trên hộp thoai này ta chọn thư mục(ACAD2000) và tên file(XE.lsp) rồi ấn ***Load*** để tải.

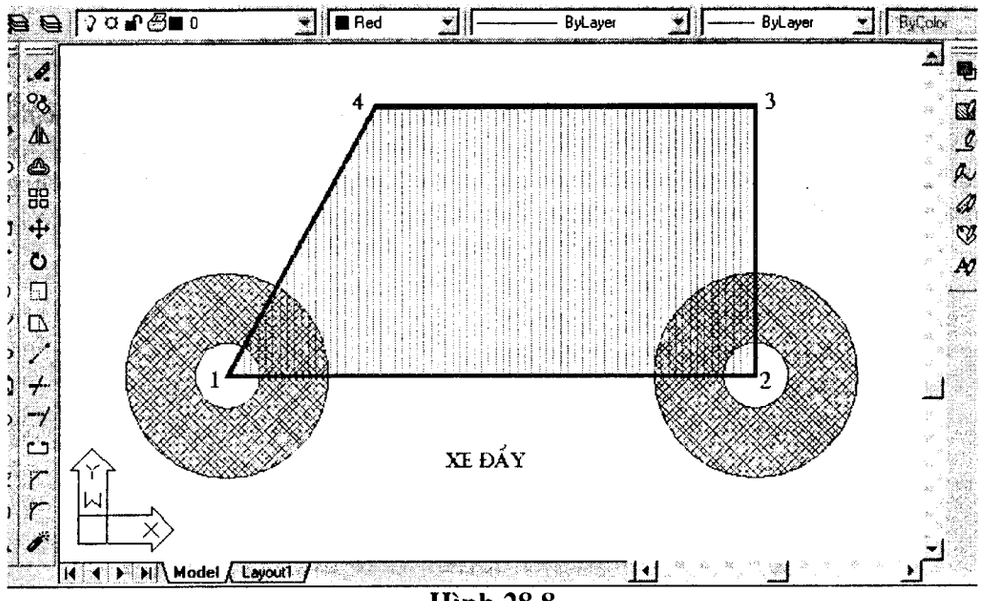


Hình 28.7: Hộp thoại Load/Unload Applications.

2). Chạy chương trình: ở ví dụ này trong file XE.lsp hàm Defun định nghĩa là **C:XE**, để tạo lệnh mới cho AutoCAD. Để chạy chương trình ta gõ **XE** vào như sau:

Command: **XE**
CHO DINH 1 KHUNG XE-TAM BANH 1: <điểm 1, hình 28.8>
CHO DINH 2 KHUNG XE-TAM BANH 2: <điểm 2, hình 28.8>
CHO DINH 3 KHUNG XE: <điểm 3, hình 28.8>
CHO DINH 4 KHUNG XE: <điểm 4, hình 28.8>
Specify radius of circle or [Diameter]: 48 <bán kính lớn bánh xe 1>
Specify radius of circle or [Diameter] <>: 16 <bán kính bé bánh xe 1>
Specify radius of circle or [Diameter] <48>: <bán kính lớn bánh xe 2>
Specify radius of circle or [Diameter] <16>: <bán kính bé bánh xe 2>
Select objects: <chọn đường bao thân xe để gạch trang trí>
Select objects: <chọn vòng tròn thứ 1 của bánh 1>
Select objects: <chọn vòng tròn thứ 2 của bánh 1>
Select objects: <chọn vòng tròn thứ 1 của bánh 2>
Select objects: <chọn vòng tròn thứ 2 của bánh 2>

Kết quả chạy chương trình **XE.lsp** ta có hình 28.8.

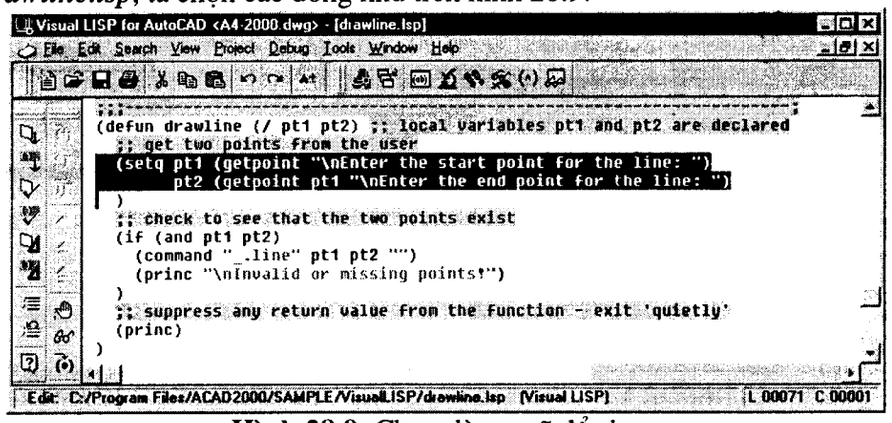


Hình 28.8

28.4.2. Chạy các dòng mã tuỳ ý chọn

Giả sử 1 chương trình được mở trong VLISP Text Editor, ta có thể chọn các dòng mã trong chương trình đó để chạy thử. Ta thực hiện như sau:

1). Chọn dòng mã trong chương trình (kích sáng dòng mã). Ví dụ trên với chương trình ***Drawline.lsp***, ta chọn các dòng như trên hình 28.9.



Hình 28.9: Chọn dòng mã để chạy.

2). Trên thanh ***Tools*** ta ấn nút ***Load Selection*** ( ) để tải và chạy. Khi ấy cửa sổ Console thu lại và AutoCAD nhắc ta nhập dữ liệu.

28.4.3. Tải các hàm mở rộng của AutoLISP

Chú ý rằng khi ta khởi động AutoCAD thì một số hàm mở rộng cho AutoLISP (các hàm được bắt đầu là Vir-, Vla-...) không được tự động tải vào.

- Hàm **Vir-** tạo hỗ trợ cho hoạt động của AutoCAD.
- Hàm **Vla-** tạo hỗ trợ ActiveX trong AutoLISP
- Hàm **Vlax-** cung cấp các tiện ích cho các hàm chuyển đổi dữ liệu, xử lý từ điển, đo các đường cong...

Cách tải các hàm mở rộng đó bằng cách nhập sau dấu nhắc Console Window:

(vl - load – com)

28.5. ĐỊNH DẠNG MÃ QUẢN LÝ CHƯƠNG TRÌNH

28.5.1. Tạo mã màu của VLISP

Trong các cửa sổ **Console** hay **Text Editor** khi ta nhập Text vào thì VLISP xác định xem Text mà ta nhập vào là 1 hàm, biến, chuỗi hay thành phần nào khác. Sau đó VLISP gán các màu cho mỗi loại thành phần đó để ta quản lý và phát hiện thiếu sót tên hàm, dấu ngoặc đơn trong chương trình. Dưới đây là các màu gán mặc định của VLISP:

Màu gán **Các thành phần của AutoLISP**

- Đỏ..... Dấu ngoặc đơn(Parentheses)
- Tím .....Lời chú thích
- Xanh lục.....Số nguyên(Interger)
- Số thực.....Màu Teal
- Đỏ tím.....Chuỗi(Strings)
- Xanh .....Các hàm tạo sẵn
- Màu đen.....Các thành phần riêng không nhận biết

Ta có thể đổi màu mặc định đó từ main menu VLISP như sau:

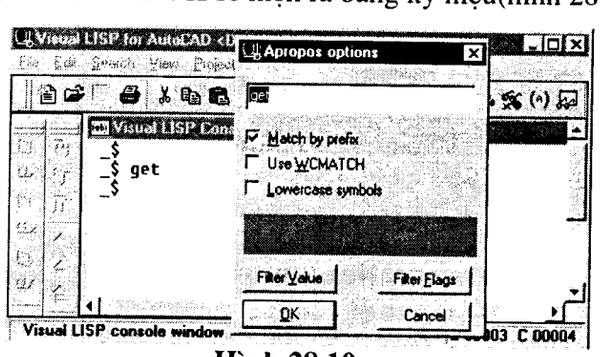
Tools> Window Attribute> Configure Current

28.5.2. Tìm kiếm bảng ký hiệu bằng APROPOS

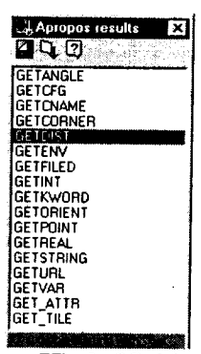
Khi thực hiện chương trình AutoLISP, sử dụng tính năng **Apropos** giúp ta tìm kiếm bảng ký hiệu một cách nhanh chóng. Để thực hiện **Apropos** ta làm như sau:

Từ menu VLISP> View> Apropos Window

Sau đó xuất hiện hộp thoại **Apropos options** (hình 28.10), trên đó hiện ra text như ta đã nhập "get". Tiếp theo lựa chọn: *Match by prefix, use WCMATCH hay Lowercase symbols* và ấn **OK** sẽ hiện ra bảng ký hiệu(hình 28.11).



Hình 28.10



Hình 28.11

28.5.3. Định dạng mã quản lý chương trình VLISP

Để tạo các khối Text, sắp xếp cách căn lề thụt đầu dòng cho chương trình ta sử dụng bộ định dạng mã mặc định VLISP. Ta cũng có thể định dạng lại Text trong cửa sổ Text Editor như sau:

1). Định dạng tất cả Text trong cửa sổ Text Editor:

Tools> Format code in Editor (từ menu VLISP)

2). Định dạng các thành phần của Text trong cửa sổ Text Editor:

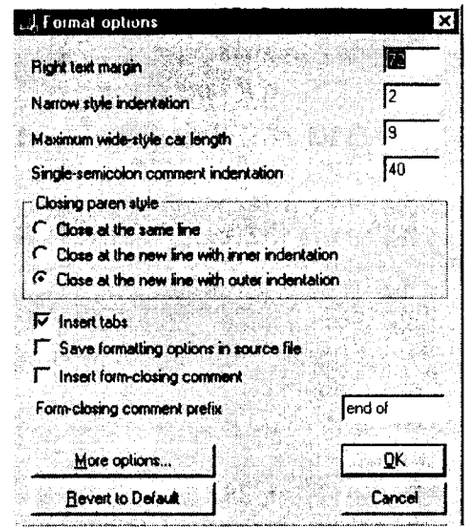
+ Chọn đoạn Text.

+ **Tools> Format code in Selection** (từ menu VLISP)

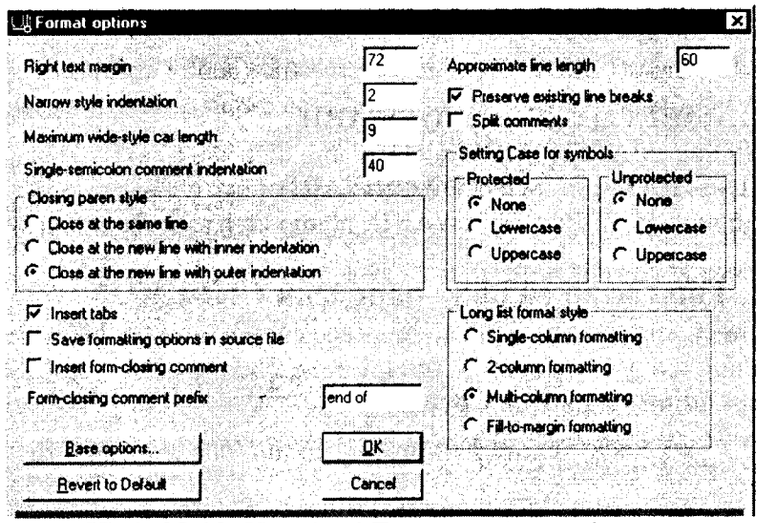
3). Định dạng qua hộp thoại **Format options** (hình 28.12). Để làm xuất hiện hộp thoại đó ta làm như sau(từ menu VLISP):

Tools> Environment Options> Visual LISP Format Options

Trên hộp thoại **Format options** ta có thể định dạng. Muốn định dạng chi tiết hơn ta ấn vào nút ***More Options*** sẽ xuất hiện hộp thoại **Format options** mở rộng (hình 28.13).



Hình 28.12:
Hộp thoại **Format options.**



Hình 28.13: Hộp thoại Format options mở rộng.

28.6. KIỂM TRA LỖI CÚ PHÁP(SYNTAX ERRORS)

28.6.1. Kiểm tra các cặp dấu ngoặc đơn

Lỗi cú pháp thường mắc nhất là số các dấu ngoặc đơn mở và đóng không cân bằng hoặc không liên kết thành cặp. Các lệnh liên kết các dấu ngoặc đơn:

*** CTRL+] :** Match Forward

+ Nếu đặt con trỏ trước dấu ngoặc mở(dòng 1, hình 28.14a) thì **CTRL+]** sẽ chuyển con trỏ tới sau dấu ngoặc đóng tương ứng trong chương trình(dòng 4, hình 28.14b).

```
1  |(if (and pt1 pt2)
2    (command "_.line" pt1 pt2 "")
3    (princ "\nInvalid or missing
       points!")
4  )
```

Hình 28.14a

```
1  (if (and pt1 pt2)
2    (command "_.line" pt1 pt2 "")
3    (princ "\nInvalid or missing
       points!")
4  )|
```

Hình 28.14b

+ Nếu đặt con trỏ sau dấu ngoặc mở thì **CTRL+]** sẽ chuyển con trỏ tới sau dấu ngoặc đóng gần nhất:

(| if (and pt1 pt2) (if (and pt1 pt2) |

*** CTRL+[:** Match Backward

+ Nếu đặt con trỏ ở sau dấu ngoặc đóng(dòng 6, hình 28.15a) thì **CTRL+[** sẽ chuyển con trỏ tới trước dấu ngoặc mở tương ứng trong chương trình(dòng 3, hình 28.15b).

```
3  (if (and pt1 pt2)
4    (command "_.line" pt1 pt2 "")
5    (princ "\nInvalid or missing
     points!")
6  )|
```

Hình 28.15a

```
3  |(if (and pt1 pt2)
4    (command "_.line" pt1 pt2 "")
5    (princ "\nInvalid or missing
     points!")
6  )
```

Hình 28.15b

+ Nếu đặt con trỏ ở trước dấu ngoặc đóng thì **CTRL+[** sẽ chuyển con trỏ tới trước dấu ngoặc mở gần nhất.

(if (and pt1 pt2 |) (if |(and pt1 pt2)

*** CTRL+SHIFT+] : Select FORWARD**

Tương tự như MATCH FORWARD, nhưng kích sáng Text ở giữa đầu và cuối.

*** CTRL+SHIFT+[: Select BACKWARD**

Tương tự như MATCH BACKWARD, nhưng kích sáng Text ở giữa đầu và cuối.

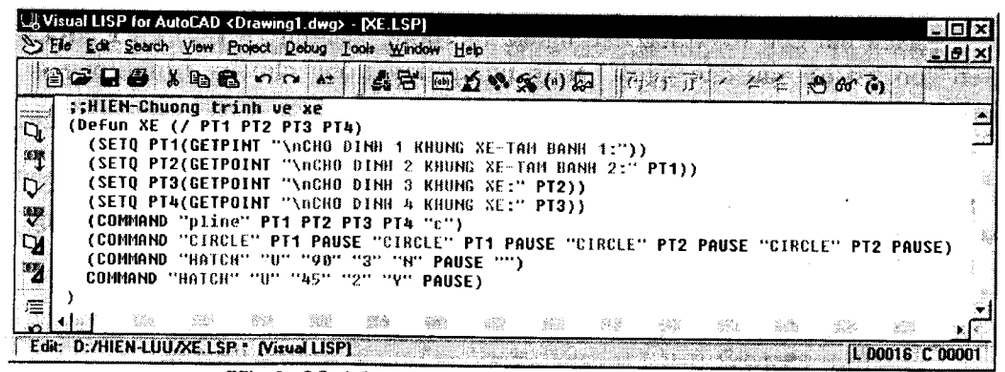
28.6.2. Tìm các lỗi cú pháp bằng lệnh CHECK

Lệnh **Check** của VLISP có thể dò tìm được các lỗi sau: tên biến sai, các đối số, cú pháp không đúng khi gọi hàm. Để dò tìm lỗi trong chương trình AutoLISP, trước tiên ta mở của sổ Text Editor và gọi lệnh **Check**:

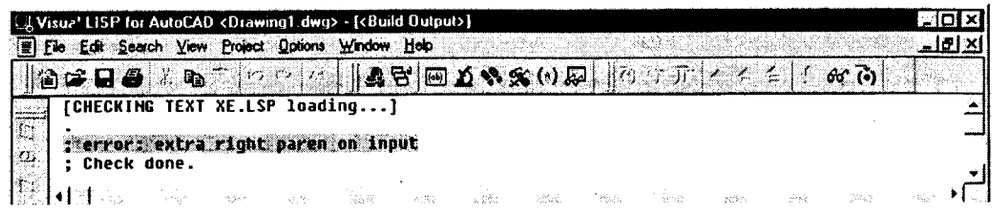
Tools > **Check Text in Editor**: dò tìm lỗi của toàn bộ file chương trình.

Check Selection: dò tìm lỗi của đoạn chương trình đã chọn.

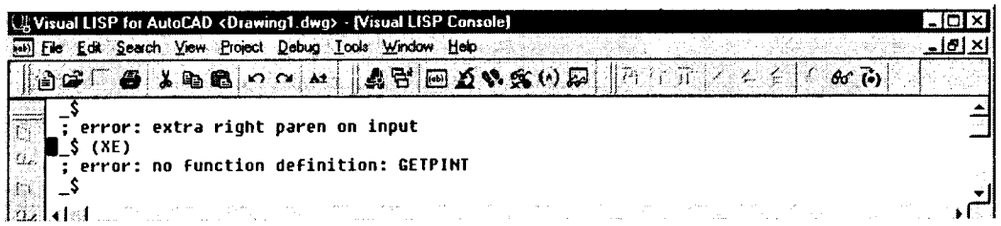
*** Ví dụ**: Hãy sử dụng lệnh **Check** để dò tìm lỗi của file chương trình ***XE.lsp*** (hình 28.16). Sau khi gọi lệnh Check để dò tìm lỗi cho toàn bộ file chương trình, VLISP đưa ra cửa sổ thông báo lỗi sau khi Check(hình 28.17), khi chạy chương trình vẫn còn lỗi cũng thông báo tiếp(hình 28.18).



Hình 28.16: File chương trình cần dò tìm lỗi.



Hình 28.17: Cửa sổ thông báo lỗi sau Check(thiếu dấu ngoặc mở).



Hình 28.18: Cửa sổ thông báo lỗi vẫn còn khi chạy chương trình(GETPINT là sai).

Sau khi biết được các lỗi đó ta có thể sửa ngay, hoặc khó tìm vị trí lỗi thì ta ***nháy đúp*** phím trái chuột vào dòng thông báo lỗi của cửa sổ ***Build Output,*** khi ấy VLISP mở cửa sổ Text Editor và đặt con trỏ ở đầu câu lệnh liên quan đến lỗi để ta tìm sửa.

28.6.3. Dò tìm lỗi cú pháp qua mã màu

Như đã nêu trong tiết 28.4.1 ở trên, VLISP gán các màu cho mỗi loại thành phần đó để ta quản lý và phát hiện thiếu sót tên hàm, dấu ngoặc đơn trong chương trình. Nghĩa là ta có thể dựa vào màu của từng thành phần trong chương trình để dò tìm lỗi.

28.7. TÍNH NĂNG GỠ RỐI CỦA VISUAL LISP

Khi chương trình dài có lỗi hoặc bị hỏng thì khó mà xác định nguyên nhân. Nhưng VLISP cung cấp nhiều tính năng tìm và xử lý sự cố của chương trình- gỡ rối chương trình.

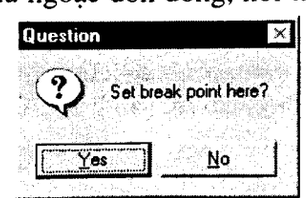
28.7.1. Cài đặt Breakpoint để ngắt việc thực hiện chương trình

Để gỡ rối chương trình ta thường phải kiểm tra từng đoạn, hay từng hàm mã chương trình. Cho nên VLISP cung cấp tính năng đặt **Breakpoint**(điểm ngắt) để việc chạy chương trình chỉ thực hiện tới điểm ngắt thì dừng lại để ta kiểm tra và xử lý.

Giả sử ta đang mở chương trình ***Yinyang.lsp*** (trong thư mục SAMPLE của AutoCAD 2000) trong cửa sổ Text Editor(hình 28.16). Để cài đặt **Breakpoint** ta làm như sau:

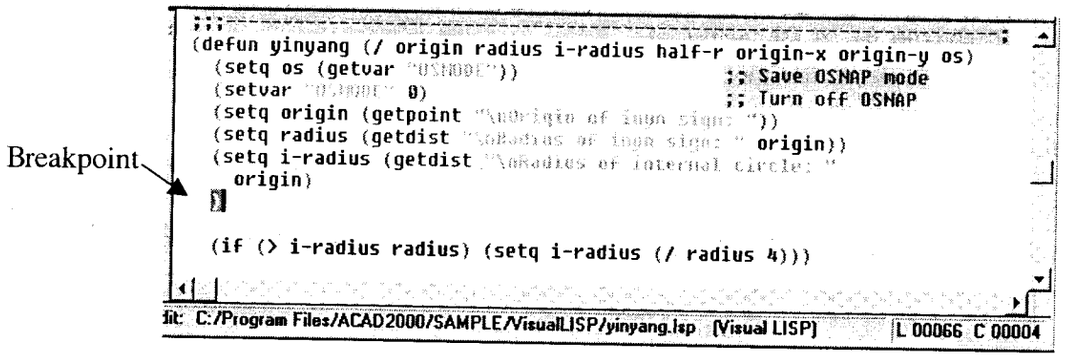
1). Đặt con trỏ vào trước dấu ngoặc đơn mở, hoặc sau dấu ngoặc đơn đóng, nơi ta muốn ngắt(hình 28.19).

Nếu ta không đặt con trỏ ở vị trí như vậy, mà đặt ở vị trí khác trên dòng Text, thì VLISP tự đặt con trỏ và đưa ra hộp thoại như hình bên để hỏi ta "Có đặt điểm ngắt ở đây không?".



2). Trên menu VLISP chọn: ***Debug> Togle Breakpoint,*** hoặc ấn nút ***Togle Breakpoint*** trên thanh Debug:

Lúc này xuất hiện ô chữ nhật màu đỏ chứa con trỏ trong đó(hình 28.19).



Hình 28.19: Đặt Breakpoint.

3). Sau đó ta chạy chương trình thì bị dừng lại tại **Breakpoint** đó. Các cách duyệt qua chương trình:

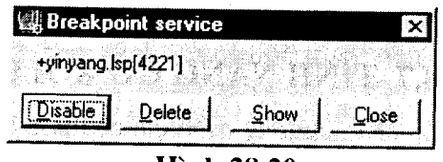
Debug > Step Info ( ), Step Over( ), Step Out( ), Continue( )...

4). **Xoá Breakpoint**

- Để xoá 1 **Breakpoint**: đặt con trỏ tại vị trí **Breakpoint**, sau đó ấn nút ***Togle Breakpoint*** trên thanh Debug(F9).
- Để xoá tất cả các **Breakpoints** mà ta đã cài đặt: ***Debug> Clear All Breakpoint*** trên menu VLISP.

5). **Tạm ngưng hoạt động của các Breakpoint**

Một hay nhiều Breakpoint có thể còn sử dụng sau này, do vậy ta cần tạm ngưng hoạt động của chúng. Muốn vậy trước tiên ta đặt con trỏ tại Breakpoint và ấn phím phải chuột, sau đó chọn ***Breakpoint service***. Lúc đó hiện ra hộp thoại **Breakpoint service**(hình 28.20), ta ấn nút ***Disable*** để tạm ngưng hoạt động của Breakpoint đó.



Hình 28.20

28.7.2. Chạy chương trình bởi chế độ ANIMATE

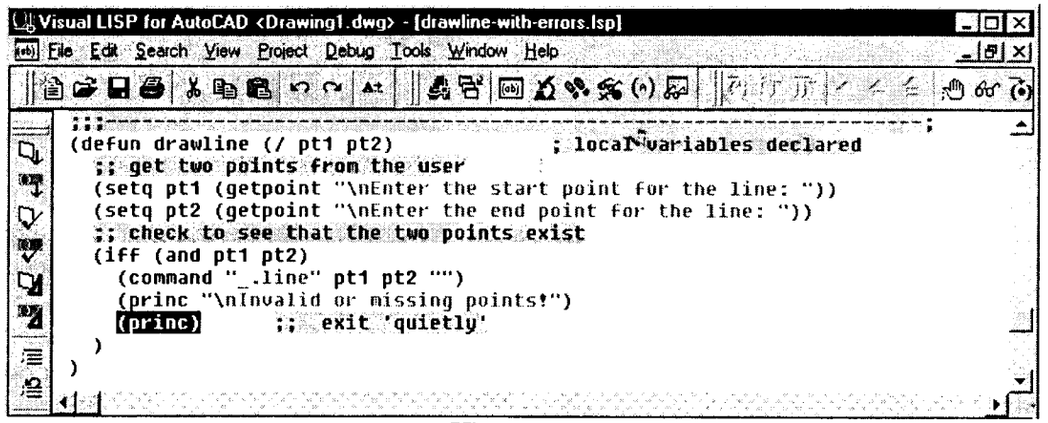
Chạy chế độ **Animate** thì VLISP trình diễn lần lượt qua mỗi hàm được kích sáng lên để ta theo dõi một cách dễ dàng.

Ví dụ ta mở file chương trình ***Drawline-with-errors.lsp*** có trong thư mục SAMPLE> Visual LISP của AutoCAD2000 để kiểm tra gỡ rối.

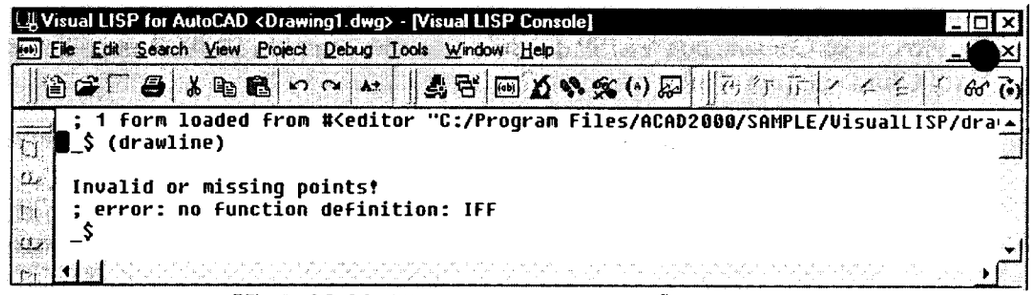
1). Gọi chế độ Animate từ menu VLISP: **Debug > Animate**

2). Chạy chương trình: (***drawline***)

VLISP trình diễn lần lượt qua mỗi hàm được kích sáng lên để ta theo dõi một cách dễ dàng(hình 28.21). Khi gặp lỗi thì việc chạy chương trình bị dừng lại và hiện hộp thoại thông báo lỗi (hình 28.22). Trên đó ta thấy lỗi là tên hàm IFF sai.



Hình 28.21



Hình 28.22: Hộp thoại thông báo lỗi(sai IFF).

3). Việc thực hiện chương trình sẽ dừng ở điểm ngắt **Breakpoint**; muốn VLISP tiếp tục chạy chương trình trong chế độ Animate, ta ấn nút **Continue**.

=== * ===

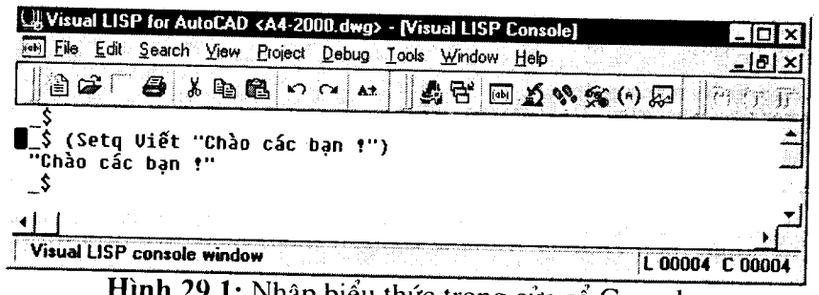
CHƯƠNG 29
CÁC VẤN ĐỀ CƠ BẢN CỦA AUTOLISP

29.1. MỞ ĐẦU AUTOLISP

29.1.1. Cấu tạo chương trình AutoLISP

Một chương trình AutoLISP là tập hợp các ***biểu thức***(Expresion) được đặt trong các cặp dấu ngoặc đơn mở và đóng(Perenthesis). Biểu thức của AutoLISP là tập hợp các con số, chuỗi ký tự và chúng được cách nhau bởi ký tự trống. Nếu ta nhập một biểu thức nằm trong cặp ngoặc đơn vào dòng nhắc **"Command:"** của AutoCAD, hay vào dấu nhắc **"_$"** của Console Window trong VLISP, thì máy thông dịch biểu thức đó rồi đưa kết quả ra màn hình và lưu kết quả đó trên bộ nhớ. Ví dụ:

1. Command: **(+ 100 50)** ↵
   150
2. Mở cửa sổ Console của VLISP để nhập biểu thức, sau khi ENTER ta có kết quả là dòng text **"Chào các bạn !"** như trên hình 29.1.



Hình 29.1: Nhập biểu thức trong cửa sổ Console.

• Có thể sử dụng các lệnh của AutoCAD trong chương trình:

(Command <đối số>)

Trong đó **<đối số>** có thể là ***tên lệnh*** của AutoCAD, là ***số nguyên, số thực, tên biến***.
Ví dụ chương trình MYLINE sau đây sử dụng lệnh LINE của AutoCAD:

```
(Defun MYLINE ( )
    (Setq P1(getpoint "Cho điểm thu 1:"))
    (setq P2(getpoint "Cho điểm thu 2:"))
    (Command "LINE" P1 P2)
)
```

*** Các quy ước viết các ký hiệu đặc biệt trong chương trình:**

- Viết " " trong hàm Command tương đương với ấn phím ENTER. ví dụ:
  (Command "SOLID" P1 P2 P3 P4 " ")
- Viết **(Command)** trong chương trình tương đương với CTRL+C (huỷ lệnh)
- Viết **PAUSE** trong lệnh Command của chương trình để dừng lại khi chạy chương trình để nhập dữ liệu. Ví dụ:
  (Command "CIRCLE" P1 PAUSE "CIRCLE" P2 PAUSE)
- Viết **;** trong chương trình để ghi chú dòng Text, AutoLISP không chạy dòng chương trình sau dấu ; này. Ví dụ:

```
; CT chuyển đổi từ độ thành radian
(defun RA-DO (x)      ; hàm RA-DO có 1 đối số x
   (* Pi (/ x 180))   ; biểu thức thân hàm
```


29.1.2. Soạn thảo chương trình

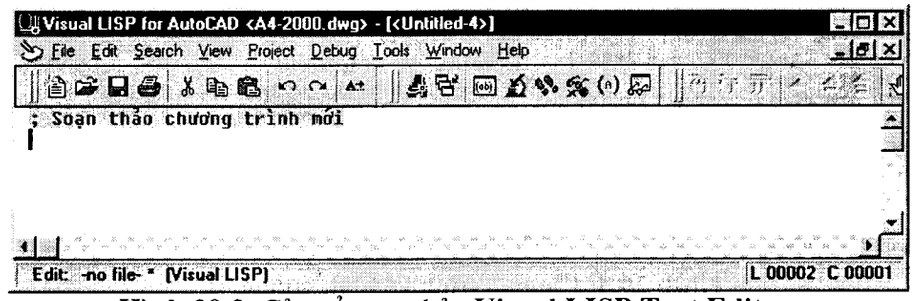
Muốn có một chương trình trước hết ta phải soạn thảo nó trong một phần mềm soạn thảo. Ví dụ đang ở trong AutoCAD, ta thoát ra tạm thời để soạn thảo trong EDIT của MS DOS như sau:

Command: SH
DOS command: EDIT

Đối với AutoCAD 2000 ta có thể soạn thảo trực tiếp trong cửa sổ **Text Editor** của **Visual LISP** (xem chương 28, mục 28.3). Cách mở cửa sổ **Text Editor**:

Tools> AutoLISP> Visual LISP Editor

Sau đó chọn ***New*** để soạn thảo chương trình mới, hoặc chọn ***Open*** để mở file chương trình đã có ra để hiệu chỉnh. Ta có cửa sổ soạn thảo mới như hình 29.2:



Hình 29.2: Cửa sổ soạn thảo **Visual LISP Text Editor**.

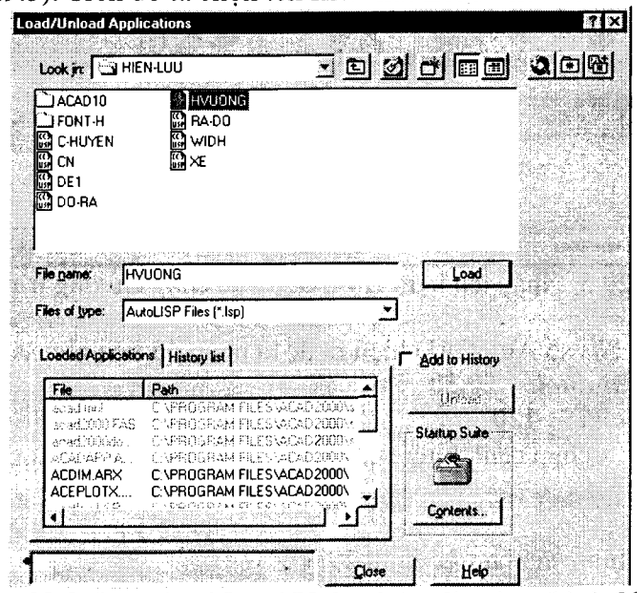
Sau khi soạn thảo xong, ta ghi thành file chương trình AutoLISP phải có đuôi **.LSP**, ví dụ: ***Myline.lsp, Truc.lsp, Qun-tra.lsp*** ...

29.1.2. Tải chương trình AutoLISP

Trước khi chạy chương trình cần phải tải nó vào AutoCAD. Có nhiều cách tải chương trình như sau:

1). Command: **(Load "<nhập tên file không đuôi>")**
Ví dụ: Command: (Load "Hvuong")

2). AutoCAD menu: **Tools> Load Application>** Hộp thoại Load/Unload Application (hình 29.3). Trên đó ta chọn tên file> nút Load> nút Close để tải.

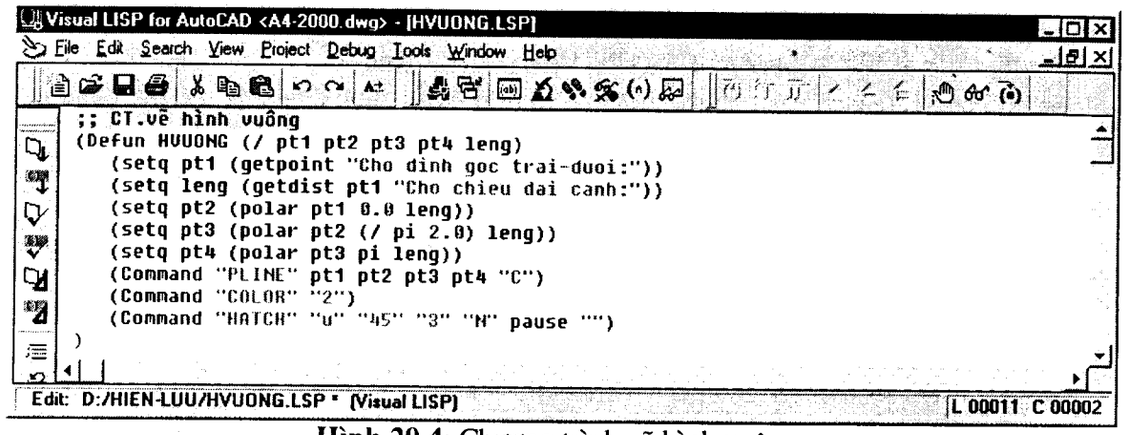


Hình 29.3: Hộp thoại Load/Unload Application (hình 29.3).

3). Dùng cửa sổ **Visual LISP Text Editor** để tải:

1).Pull-down menu: Tools> Load Text in Editor
2).Thanh Tools: chọn nút ***Load Active Edit Window:***

Ví dụ tải file chương trình ***Hvuong.lsp*** ra như hình 29.4.



Hình 29.4: Chương trình vẽ hình vuông.

29.1.3. Chạy chương trình AutoLISP

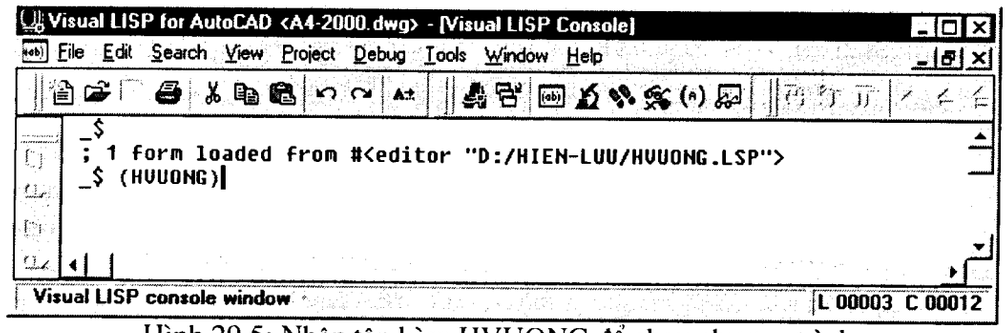
Sau khi tải mới có thể chạy chương trình được. Các cách chạy chương trình:

1). Chạy từ cửa sổ AutoCAD:

Command: (**<nhập tên hàm> <đối số>**)

Ví dụ: Command: (Canhhuyen 3 4) ⟶cho kết quả độ dài cạnh huyền là **5**

2). Chạy chương trình bằng cách nhập tên hàm trong dấu ngoặc đơn sau dấu nhắc (_∃) của **Console Window** (hình 29.5).



Hình 29.5: Nhập tên hàm HVUONG để chạy chương trình.

Sau khi ENTER cửa sổ AutoCAD hiện ra để ta nhập dữ liệu như sau:

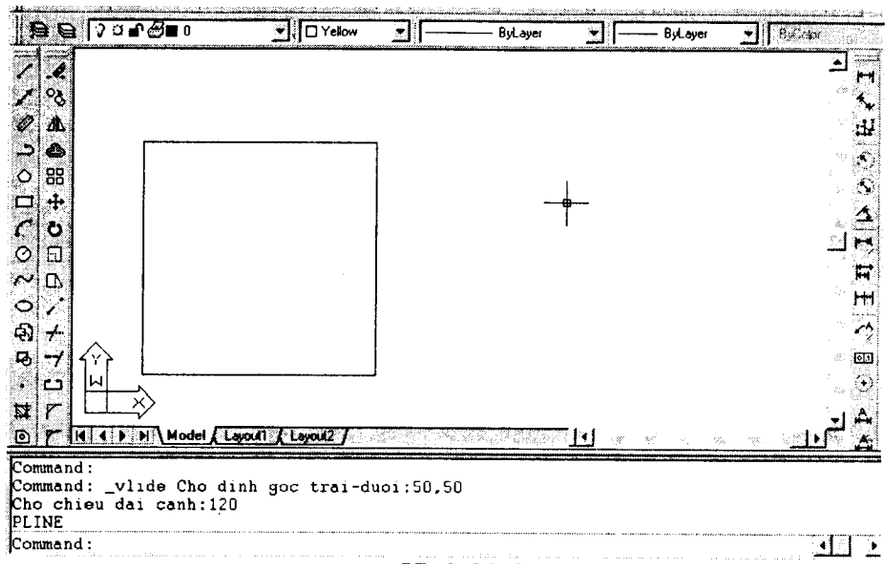
Cho dinh goc trai-duoi: **50,50**

Cho chieu dai canh: **120**

Select objects: <chọn cạnh hình vuông xác định biên gạch mặt cắt>

Select objects: ↵

Kết quả ta được hình vẽ như trên hình 29.6.



Hình 29.6

29.1.4. Tạo thêm lệnh cho AutoCAD

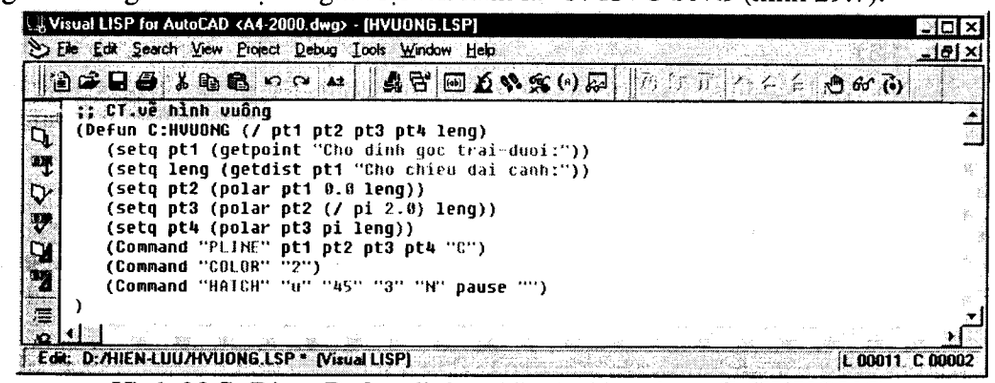
Ta có thể biến tên hàm được tạo do **Defun** thành 1 lệnh của AutoCAD bằng cách đặt tên hàm có dạng: **C: XXX**. Trong đó:

- Phần **C:** luôn bắt buộc phải có
- Phần **XXX** là tên lệnh được tạo ra cho AutoCAD, nên dùng chữ in hoa.

Chú ý rằng hàm **C:XXX** không có đối số toàn cục nhưng phải có dấu ( ) sau tên hàm **XXX**, và có thể có đối số cục bộ(đặt sau dấu /).

Khi một hàm của AutoLISP được đặt có dạng **C: XXX** thì **XXX** là tên 1 lệnh mới của AutoCAD. Cho nên để tải và chạy chương trình có tên hàm như vậy, ta chỉ cần nhập tên **XXX** sau dòng nhắc lệnh Command: của AutoCAD.

*** Ví dụ:** Chương trình vẽ hình vuông được tô mặt cắt như ví dụ ở phần trên đây, bây giờ ta dùng **Defun** định nghĩa lại tên hàm là **C: HVUONG** (hình 29.7).



Hình 29.7: Dùng Defun định nghĩa tên hàm dạng **C:HVUONG.**

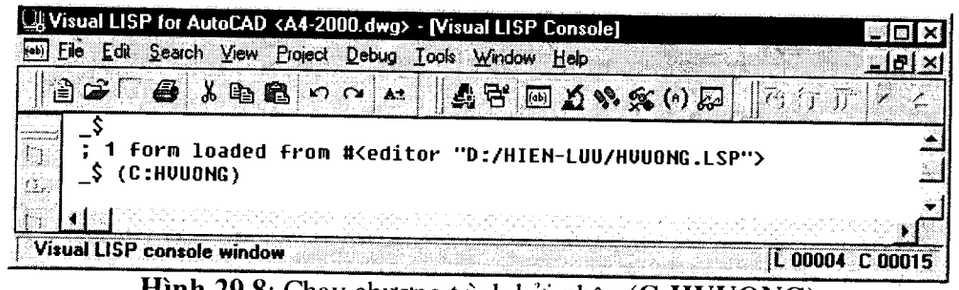
Để chạy chương trình đó, ta có hai cách như sau:

1). Chạy từ cửa sổ AutoCAD vì tên HVUONG đã là tên lệnh của AutoCAD:

Command: HVUONG

2). Chạy trong cửa sổ **Console** của VLISP: (hình 29.8)

_$(C:HVUONG)



Hình 29.8: Chạy chương trình bởi nhập (**C:HVUONG**).

29.2. CÁC KHÁI NIỆM CƠ SỞ TRONG AUTOLISP

29.2.1. Biểu thức trong AutoLISP

Một chương trình AutoLISP bao gồm chuỗi các **biểu thức**(Expresion) được đặt trong các cặp dấu ngoặc đơn mở và đóng.

Biểu thức của AutoLISP là tập hợp các số, chuỗi ký tự mà chúng đặt cách nhau bởi khoảng trống. Biểu thức của AutoLISP có dạng như sau:

(Function Arguments)

+ Function: tên hàm
+ Arguments: đối số

Mỗi biểu thức được bắt đầu bởi dấu ngoặc đơn mở: (, sau đó là tên hàm, rồi các đối số của hàm đó, kết thúc biểu thức là dấu ngoặc đơn đóng:) . Mỗi đối số của hàm có thể là một biểu thức. Do vậy một biểu thức có thể chứa các biểu thức con bên trong nó. Ví dụ các biểu thức sau:

1). Command: **(+ 9 1)**
10
Đó là 1 biểu thức: hàm cộng, có 2 đối số là 9 và 1, kết quả hoàn trả là 10.

2). **_$(* 2(+ 8 7))**
30
Đây là các biểu thức lồng nhau: biểu thức con bên trong chứa hàm cộng (+) 2 đối số là 8 và 7. Kết quả của biểu thức hàm cộng đó sẽ là 1 trong 2 đối số của hàm nhân (*).

3). **_$(List (+ 2 1) (* 3 4))**
(3 12)
Đây là các biểu thức lồng nhau: có 2 biểu thức con bên trong (hàm + và hàm *) mà kết quả của chúng sẽ là 2 đối số cho hàm *List* của biểu thức ngoài. Kết quả hoàn trả sẽ là danh sách List: (3 12).

• Các biểu thức đơn giản như trên(hay các mã) của AutoLISP không cần phải soạn thảo như một chương trình, cho nên ta có thể nhập vào sau dòng nhắc lệnh **Command:** của AutoCAD(ví dụ 1) hoặc nhập vào sau dấu nhắc lệnh **_$** của cửa sổ Console trong VLISP(ví dụ 2, 3 ở trên). AutoLISP sẽ xử lý và hoàn trả kết quả ngay tại các dấu nhắc đó.

• Nếu ta nhập vào số lượng các dấu ngoặc đơn đóng chưa đúng, AutoLISP sẽ hiện ra dấu nhắc nhở ở đầu dòng như sau: **(_>**

29.2.2. Các kiểu dữ liệu của AutoLISP

a). Intergers (kiểu số nguyên) và Reals(kiểu số thực)

• **Intergers** là các số nguyên không chứa dấu chấm thập phân và chọn trong khoảng +2 147 483 647 đến – 2 147 483 648. Nếu ta nhập vào con số lớn hơn số cực đại đó, hoặc

kết quả của phép tính giữa các số cho phép là con số lớn hơn số cực đại thì AutoLISP sẽ đưa ra kết quả sai. Ví dụ:

1). **_$(/ 9 2)**
4
Kết quả hàm chia(/) đó đã bỏ đi phần lẻ, không phải số nguyên.

2). **_$(List 5 12 -7)**
(5 12 -7)

• **Reals** là số có chứa dấu chấm thập phân. Nếu Reals nằm giữa –1 và +1 thì phải có số 0 ở đầu trước dấu chấm thập phân. Ví dụ: 0.5, 0.123... Ví dụ:

3). **_$(* 2.5 10.7)**
26.75

4). Trong ví dụ 1, muốn kết quả hàm chia giữa 2 số nguyên 9 và 2 sẽ là số thực thì ta phải nhập các số ở dạng số thực như sau:

_$(/ 9.0 2.0)
4.5

b). List (kiểu danh sách)

List là kiểu dữ liệu đặc trưng của AutoLISP. List có thể bao gồm các giá trị cụ thể, các biến, các hàm. Ví dụ:

1). **_$(6 4.5 -7)**
(6 4.5 -7)

2). **_$(x "Lim" -90)**
(x "Lim" -90)

*** Chú ý:**

• Hàm **QUOTE** (có thể viết tắt là dấu ‘) được dùng để tạo ra một hàm **LIST** gồm toàn là các giá trị. Ví dụ:

_$(quote (1 3 -4.7))
(1 3 -4.7)

Hoặc:

_$(‘(1 3 –4.7))
(1 3 –4.7)

• Có thể thay hàm **QUOTE** bởi hàm **LIST** ; mà hàm LIST gồm các phần tử là các giá trị và cả các biến nữa. Ví dụ:

_$ (setq A (getpoint "Cho điểm A:"))
(50.0 100.0 0.0)
_$ (List A)
((50.0 100.0 0.0))

c). Strings (kiểu chuỗi ký tự)

Strings là chuỗi các ký tự đặt bên trong cặp dấu ngoặc kép " ". Chuỗi ký tự không được dài quá **132** ký tự. Ví dụ:

1). **_$ (setq VIET "Chương trình vẽ nhà")**
"Chương trình vẽ nhà"

2). **_$ (setq STRING (getstring "Họ & Tên:"))**
"Ng.V.HIEN"

*** Chú ý:** Nếu đặt ký tự đặc biệt ^ đứng trước các ký tự khác (\, n, e, r, t, nnn ",\ ...) sẽ cho ta các ý nghĩa khác như sau:

\ n → để xuống dòng
\ e → ESC
\ r → ENTER
\ t → TAB
\ nnn → ký tự mã hệ 8 là nnn
\ " → "
**** → \
\ U+xxxx → ký tự mã ASCII là xxxx
\ M+nxxxx → ký tự mã UNICODE là xxxx

d). Ngoài ra còn các dữ liệu kiểu

- Symbols: để mô tả các ký tự ASCII mà chúng chứa một thông tin nào đó.

- Kiểu đối tượng vẽ VLA(Visual LISP ActiveX): các đối tượng đặt tên của bản vẽ AutoCAD có thể là dữ liệu trong AutoLISP.

- Kiểu File: AutoLISP có thể dùng file descriptor mô tả dưới dạng đối số cho hàm AutoLISP.

29.2.3. Các biến trong AutoLISP

Một biến AutoLISP nhận các giá trị kiểu dữ liệu đã gán cho nó cho đến khi gán giá trị mới. Các biến trong ngôn ngữ lập trình bậc cao khác thường phải khai báo kiểu biến trước khi sử dụng, còn AutoLISP tự động xác định kiểu biến theo giá trị đã gán cho nó.

Đặc biệt biến trong đo góc của AutoLISP phải dùng đơn vị là ***Radian(RA);*** biến số pi được gán sẵn giá trị **3.14159**. Một biến không được gán giá trị thì gọi là ***Nil***. Biến với ***Nil*** khi giá trị của nó không còn cần thiết nữa, sẽ giảm việc sử dụng bộ nhớ của máy.

• Hàm **Setq** được sử dụng để gán giá trị cho các biến. Nó có dạng sau:

(Setq <tên biến 1> <giá trị 1> <tên biến 2> <giá trị 2>...)

Hàm **Setq** sẽ gán giá trị 1 cho biến 1, gán giá trị 2 cho biến 2...Nhưng AutoLISP chỉ hoàn trả giá trị của biểu thức biến cuối cùng, như ví dụ sau:

```
_$ (Setq  a  5.0   b  9   c  -122)
-122
```

Chú ý rằng lệnh Setq được dùng với các cách bố trí các biến khác nhau nhưng kết quả hoàn trả giống nhau, ví dụ:

1). Viết riêng:

```
_$ (Setq  k  3.5)
3.5
_$ (Setq  l  -7)
-7
_$ (Setq  r  "HELLO!")
"HELLO!"
```

2). Viết gộp lại:

```
_$ (Setq  k  3.5  l -7  r "HELLO!")
"HELLO!"
```


• **Cách hiển thị giá trị một biến:**

1).
```
_$ k
3.5
_$ r
"HELLO!"
```

2).
```
Command:  !k
3.5
Command:  !r
"HELLO!"
```

• **Các biến hệ thống trong AutoLISP**

Để truy cập đến các biến hệ thống AutoCAD, ta sử dụng 2 hàm AutoLISP như sau:
Tra cứu giá trị hiện thời biến hệ thống:

(getvar "tên biến hệ thống")

Ví dụ:

1).
```
Command: (getvar "LTSCALE")
13.0
```

2).
```
_$ (getvar "FILLETRAD")
0.0
```

Để gán giá trị mới cho biến hệ thống:

(setvar "tên biến hệ thống" <giá trị mới>)

Ví dụ:

1).
```
Command: (setvar "LTSCALE" 1)
1
```

2).
```
_$ (setvar "DIMCEN" 3)
3
```

29.2.4. Các hàm(hay lệnh) trong AutoLISP

a). Cú pháp của hàm

Các hàm AutoLISP có cú pháp được mô tả như sau:

(foo *string [number ...]*)

Trong đó:

- **foo**: tên hàm
- ***string***: đối số(biến) theo yêu cầu của hàm **foo**
- ***number***: đối số tỳ ý(có thể không có)

Ví dụ:

1).
```
_$ (getpoint  Point  [dòng nhắc])
```

Trong đó hàm **getpoint** để nhập 1 điểm, đối số **Point** là toạ độ 1 điểm, còn đối số ở trong [ ] để ghi chú, có thể cho hoặc không.

```
_$ (getpoint  P1  "Cho toạ độ điểm 1:")
Cho toạ độ điểm 1:
```

2).
```
_$ (quote (+ 4  6))
(+ 4  6)
```

b). Định nghĩa hàm và thực thi hàm:

Phần này ta xét cách định nghĩa một hàm và gọi hàm ra chạy như thế nào.

• Cú pháp định nghĩa một hàm:

(defun <tên hàm> <các biến/biến cục bộ> <biểu thức thân hàm>)

Trong đó:

+ Thành phần đầu tiên trong cặp ( ) là từ khoá **defun**

+ Thành phần thứ hai là tên hàm nhập vào

+ Thành phần thứ ba là biến toàn cục, hay biến cục bộ(đặt sau dấu /). Biến cục bộ chỉ có tác dụng trong một hàm con.

+ Thành phần cuối cùng là biểu thức thân hàm.

Ví dụ:

1).
```
;;CT.đổi từ ĐỘ thành RADIAN
(defun  DO-RA  (x)      ;;;hàm DO-RA này có 1 đối số x
  (* pi (/  x  180))    ;;;biểu thức thân hàm
)
```

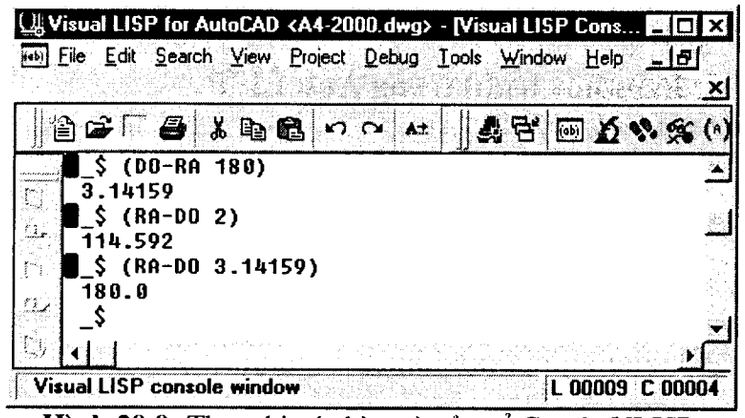
2).
```
;;;CT.đổi RADIAN thành ĐỘ
(defun RA-DO  (x)
  (/ (*  x  180)  pi)
)
```

• Để chạy chương trình ta có các cách(xem mục 29.1.3):

1). Chạy từ dòng nhắc lệnh của AutoCAD: nhập trong cặp ngoặc đơn ( ) tên hàm và sau đó là đối số(biến) cách nhau khoảng trống:

```
Command:  (DO-RA  180)
3.14159
Command:  (RA-DO  3)
171.887
```

2). Chạy từ dấu nhắc của cửa sổ Console VLISP: (hình 29.9)



Hình 29.9: Thực thi các hàm từ cửa sổ Cosole VLISP.

• Hàm **C:XXX** tạo thêm lệnh cho AutoCAD: xem mục 29.1.4 đã trình bày ở trên.

Có nhiều loại hàm: hàm gán giá trị, hàm danh sách LIST, hàm toán, hàm nhập điểm, hàm nhập khoảng cách, hàm chuỗi ký tự...mà ta xét cụ thể dưới đây.

29.2.5. Các hàm(hay lệnh) cơ bản của AutoLISP

1). Hàm QUOTE

Hàm này dùng để hoàn trả biểu thức mà không tính toán. Cú pháp của lệnh Quote:

(quote <biểu thức>) hoặc **'<biểu thức>**

Chú ý rằng kiểu viết thứ hai của hàm Quote không được thực thi tại dòng nhắc Command: của AutoCAD.

Ví dụ:

1). **Command: (quote (ab))**
(AB)

2). **_$ (quote (+ 1 2))**
(+ 1 2)
_$ '(+ 1 2)
(+ 1 2)
_$ (quote m)
M
_$ 'm
M

2). Hàm SETQ

để gán(nhập) giá trị cho biến(xem mục 29.2.3). Nó có dạng sau:

(Setq <tên biến 1> <giá trị 1> <tên biến 2> <giá trị 2>...)

Ví dụ: chương trình tính cạnh huyền của tam giác vuông, biết 2 cạnh x, y.

```
(defun  HUYEN ( )       ;; định nghĩa hàm HUYEN
 (setq x (getreal  "Cho canh x:"))  ;; nhập độ lớn cạnh x(số thực)
 (setq y (getreal  "Cho canh y:"))  ;; nhập độ lớn cạnh y(số thực)
 (setq k (+  (* x  x)  (* y  y)))   ;; k là tổng bình phương 2 cạnh x, y
 (princ  "\nGia tri l=")            ;; in kết quả độ dài l cạnh huyền tính được
 (setq l (sqrt  k))         ;; l bằng căn bậc 2 của k
)
```

Sau khi chạy ví dụ với x=3, y=4 cho kết quả cạnh huyền bằng 5.

3). Hàm tra cứu và gán giá trị cho biến hệ thống của AutoCAD

(xem mục 29.2.3)

Để truy cập đến các biến hệ thống AutoCAD, ta sử dụng 2 hàm AutoLISP như sau:

Tra cứu giá trị hiện thời biến hệ thống:

(getvar "tên biến hệ thống")

Để gán giá trị mới cho biến hệ thống:

(setvar "tên biến hệ thống" <giá trị mới>)

4). Hàm nhập dữ liệu

• **Lệnh PROMPT:** để hiện dòng ký tự ra màn hình. Cú pháp:

(prompt "dòng ký tự")

Ví dụ:

1). **Command: (prompt "HELLO!!!")**
HELLO!!! Nil

2).Chương trình sau đây sử dùng hàm **Prompt**

```
(defun DOLA ( )
 (prompt "\n15000$")
```

```
    (prompt "\nGET!!!")(princ)
  )
```

Khi chạy hàm DOLA ta có kết quả hoàn trả :

```
Command: (DOLA)
1500$
GET!!!
```

• **Lệnh nhập điểm(GETPOINT)**

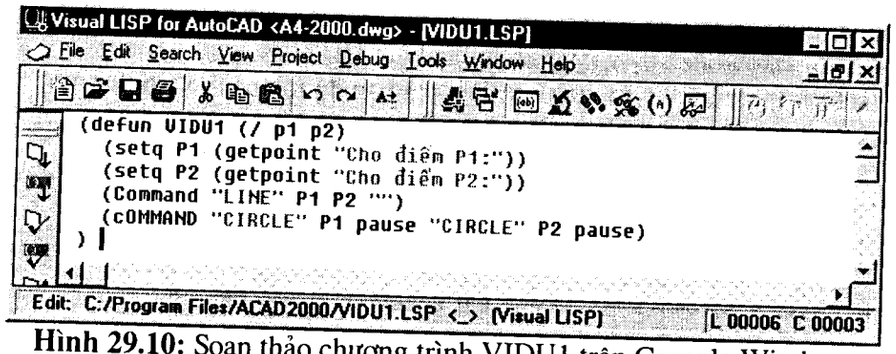
Cú pháp hàm **Getpoint**:

(getpoint [toạ độ điểm] [dòng nhắc])

Ví dụ:

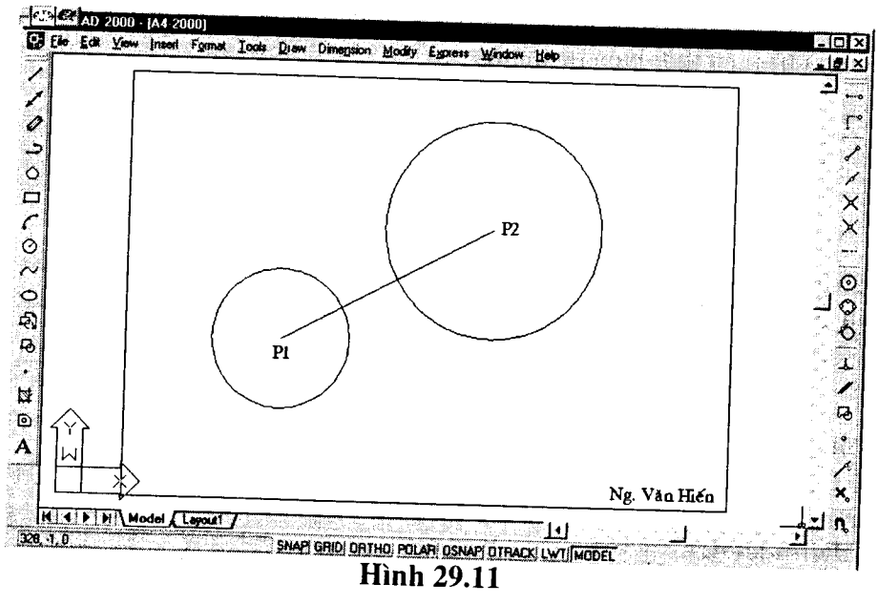
1). **(setq O1(getpoint "Cho tâm đường tròn O1:"))**
Cho tâm đường tròn O1: 100, 80

2). Bây giờ ta soạn thảo một chương trình đơn giản có sử dụng hàm **Getpoint**: nhập 2 điểm P1, P2 và vẽ đường thẳng P1P2, 2 đường tròn tâm P1 và P2, các bán kính sẽ nhập vào khi chạy chương trình. Mở cửa sổ Console của VLISP để soạn thảo chương trình (hình 29.10).



Hình 29.10: Soạn thảo chương trình VIDU1 trên Console Window.

Sau khi chạy chương trình ta thu được hình vẽ của đường thẳng P1P2 và 2 đường tròn tâm P1, P2 (hình 29.11).



Hình 29.11

• **Lệnh nhập đỉnh hình chữ nhật(GETCORNER)**

Hàm này giống như hàm **Getpoint**, nhưng phải cho toạ độ đỉnh cụ thể. Cú pháp:

(getcorner point1 [dòng nhắc])

Ví dụ: **(getcorner '(55 48) "Cho đỉnh đối của chữ nhật:")**
Cho đỉnh đối của chữ nhật:

• **Lệnh nhập khoảng cách(GETDIST)**

Để nhập khoảng cách giữa 2 điểm. Cú pháp:

(getdist [điểm gốc] [dòng nhắc])

Ví dụ: **_$ (getdist '(50 60) "Cho điểm thứ 2:")**
124.827
_$ (setq DIST (getdist '(22 88) "Cho điểm thứ 2:"))
199.081

Khoảng cách đó sẽ được gán cho biến.

• **Lệnh nhập góc(GETANGLE)**

a). Nhập góc theo UNITS đã đặt trong bản vẽ hiện thời:

(getangle [cho góc] [dòng nhắc])

Ví dụ: **_$ (getangle "Cho goc:")**
Cho goc: 45
0.785398

b). Nhập góc theo 2 điểm xác định đường thẳng nghiêng với trục x:

(getangle [điểm gốc] [dòng nhắc])

Ví dụ: **_$ (getangle "Cho diem 1:")**
Cho diem 1: 40, 50
Cho diem 2: 140, 150
0.785398
_$ (getangle '(10 20) "Diem thu 2:")
Diem thu 2: 200, 88
0.343691

• **Nhập số nguyên(GETINT), số thực(GETREAL)**

(getint [dòng nhắc])
(getreal [dòng nhắc])

Ví dụ: **_$ (setq TUOI(getint "Tuoi ban la:"))**
Tuoi ban la: 35
35
_$ (setq REAL(getreal "Cho so:"))
Cho so: 50.84
50.84

• **Nhập chuỗi ký tư (GETSTRING)**

Hàm **Getstring** chỉ cho phép nhập chuỗi ký tự có chiều dài tối đa 132 ký tự.

(getstring [T] [dòng nhắc])

+ Nếu đặt dấu \ ở trước một số ký tự khác(xem mục 29.2.2) sẽ tạo ra các ký tự mới.
+ Trong cú pháp nhập chuỗi ký tự ở trên mà có mặt **T** thì có thể nhập ký tự trống trong chuỗi. Còn trong cú pháp nhập chuỗi ký tự ở trên mà không có mặt **T** thì không thể nhập ký tự trống trong chuỗi.
+ Chú ý rằng chuỗi ký tự phải đặt trong cặp dấu ngoặc kép " ".

Ví dụ:
```
_$ (setq S(getstring "Họ & Tên:"))
Họ & Tên:    NG-Van-HIEN
"NG-Van-HIEN"
```

Sau hàm **getstring** ta không cho biến **T**, do vậy dòng ký tự nhập vào không được có khoảng trống

```
_$ (setq  S(getstring T  "Họ & Tên:"))
Họ & Tên:    Nguyen  Van  HIEN
"Nguyen  Van  HIEN"
```

Ở đây sau hàm **getstring** ta cho biến **T**, do vậy dòng ký tự nhập vào có khoảng trống

- **Nhập từ khoá(GETKWORD)**

Hàm **Getkword** để nhập các từ khoá. Danh sách các từ khoá do hàm **Initget** xác định. Cho nên phải gọi hàm **Initget** trước khi nhập từ khoá **Getkword**.

(initget [bit] [dòng nhắc])

Bit = 1 không được nhập ký tự trống
2 không được nhập số 0
4 không được nhập số âm
8 có thể nhập điểm ngoài Limits
64 chỉ dùng hàm getdist khi bỏ qua toạ độ z.

Cú pháp nhập từ khoá:

(getkword [dòng nhắc])

Ví dụ:
```
_$ (initget  1 "True False")
nil
_$ (setq x(getkword  "True or False:"))
True or False:    T
"True"
```

5). Các hàm toán

- ***Các toán tử số học:***

(+ <num1> <num2>...) ; toán tử cộng
(- <num1> <num2>...) ; toán tử trừ
(* <num1> <num2>...) ; toán tử nhân
(/ <num1> <num2>...) ; toán tử chia
(max <num1> <num2>...) ; tìm giá trị cực đại
(min <num1> <num2>...) ; tìm giá trị cực tiểu

- ***Các hàm số học và đại số:***

(1+ <number>) ; đếm tăng lên 1 đơn vị, ví dụ:
```
_$ (setq A(1+ 9))
10
```
(1- <number>) ; đếm giảm đi 1 đơn vị
(get <số nguyên> <số nguyên>...) ; tìm mẫu số chung lớn nhất

(rem <number1> <number2>...) ; tìm phần dư của phép chia
(sqrt <number>) ; khai căn bậc 2 ($\sqrt{\ }$)
(expt x p) ; tính x mũ p, ví dụ: (expt 2 3) → 8
(exp p) ; tính e mũ p
(abs <number>) ; lấy giá trị tuyệt đối của số thực, ví dụ: (abs -45.2) → 45.2
(log x) ; tính logarit tự nhiên cơ số e của x
(float x) ; chuyển số x từ nguyên sang số thực
(fix x) ; chuyển số x từ thực sang số nguyên
(sin alpha) ; tính sin góc alpha (radian)
(cos alpha) ; tính cosin góc alpha (radian)
(atan x) ; tính actang của x (radian)
(angle p1 p2) ; tính góc xác định bởi đường p1p2 và trục x(theo radian)
(distance p1 p2) ; tính khoảng cách p1p2
(polar p <alpha> d) ; tính toạ độ 1 điểm mà cách điểm cũ p một khoảng d và nằm trên đường thẳng nghiêng với trục x góc alpha

6). Các hàm xử lý chuỗi ký tư(STRING)

(strcat <string1> <string2>...) ; ghép nối các chuỗi thành 1 chuỗi
Ví dụ:
```
_$ (strcat "Hà Nội"  "-"  "Việt Nam")
"Hà Nội-Việt Nam"
```
(strcase <string> [T]) ; chuyển các ký tự trong chuỗi string thành chữ in hoa, nếu không cho T, nếu cho T thì chuyển thành chữ thường.
(strlen <string>) ; sẽ hoàn trả số ký tự có trong string.
(substr <string> <bắt đầu n> <chiều dài l>) ; chọn ra chuỗi con trong chuỗi string, từ phần tử bắt đầu n với chiều dài l.

Ví dụ:
```
_$ (substr "AUTOLISP"  5  3)
"LIS"
_$ (substr "AUTOLISP"  5)
"LISP"
```
(chr <số nguyên>) ; hoàn trả ký tự ứng với số nguyên là mã ASCII
Ví dụ:
```
_$ (chr  65)
"A"
```
(acii <ký tự>) ; chuyển 1 ký tự thành mã ASCII, ví dụ: (acii "C") → 99
(itoa <số nguyên>) ; chuyển 1 số thành số nguyên
Ví dụ:
```
_$ (atoi  "12.3")
12
```

7). Hàm xử lý danh sách(LIST)

Ngôn ngữ LISP nói chung và cả AutoLISP đều trên nền tảng là danh sách các thông tin. Một biểu thức chính là một danh sách. Chương trình AutoLISP bao gồm các biểu thức, mà mỗi biểu thức được bắt đầu bằng dấu ngoặc đơn mở và kết thúc bằng dấu ngoặc đơn đóng. Các biểu thức có thể lồng vào nhau.

AutoLISP có sẵn các hàm để xử lý với các danh sách.

- ***Hợp nhất các biểu thức thành một danh sách:***

(list <biểu thức1> <biểu thức 2>...)

Ví dụ:
```
_$ (list (+ 1  2) (- 10  3))
(3  7)
```

```
_$ (setq CHAO(list "Xin chào" "các bạn!"))
("Xin chào" "các bạn!")
```

Chú ý: Một điểm trong AutoLISP được gán bởi lệnh **List**; tức là các toạ độ (x,y) trong 2D hay (x,y,z) trong 3D được viết dưới dạng List như sau:

```
Command: (list 20 35)
(20 35)

Command: (setq P2(list 34.5 75 80))
(34.5 75 80)
```

• ***Các lệnh lựa chọn phần tử trong LIST và xử lý LIST:***

List là tập hợp nhiều phần tử bên trong. Đôi khi ta cần lựa chọn lấy 1 số phần tử trong danh sách List đó. AutoLISP đưa ra các hàm lựa chọn họ **Car** như sau:

(car <list>) ; chỉ lấy ra (hoàn trả) *phần tử thứ 1* trong danh sách List
(cadr <list>) ; chỉ lấy ra (hoàn trả) *phần tử thứ 2* trong danh sách List
(caddr <list>) ; chỉ lấy ra (hoàn trả) *phần tử thứ 3* trong danh sách List

** Chú ý: Trong thực tiễn người ta thường sử dụng* **Car** *để chọn toạ độ x của điểm,* **Cadr** *để chọn toạ độ y của điểm,* **Caddr** *để chọn toạ độ z của điểm.*

Ví dụ: Giả sử ta có 1 điểm Pt được gán toạ độ x là 10, toạ độ y là -11, toạ độ z là 12.5 bởi biểu thức sau:

```
_$ (setq Pt (list 10 -11 12.5))
(10 -11 12.5)
```

Ta sử dụng các lệnh **car, cadr, caddr** để chọn toạ độ x,y,z như sau:

```
_$ (setq x (car Pt))
10
_$ (setq y (cadr Pt))
-11
_$ (setq z (caddr Pt))
12.5
```

(cdr <list>) ; bỏ phần tử đầu tiên, chỉ lấy ra các phần tử thứ 2 trở đi.
* Chú ý: Các hàm **Car, Cdr** có thể lồng vào nhau để tạo thành các tên mới:
(caar <list>) = (car (car <list>))
(cddr <list>) = (cdr (cdr <list>))
(cadar <list>) = (car (cdr (car <list>)))
(cdar <list>) = (cdr (car <list>))

(last <list>) ; chỉ chọn phần tử cuối cùng trong List.

(nth <số nguyên n> <list>) ; chọn phần tử thứ n trong List.

Ví dụ:
```
_$ (setq L (list 1 2 3 4 5))
(1 2 3 4 5)
_$ (last L)
5
_$ (nth 0 L)
1
_$ (nth 3 L)
4
_$ (cdr L)
(2 3 4 5)
```


(length <list>) ; hoàn trả số lượng phần tử có trong List.

Ví dụ:
```
_$ (length '(m n (p q)))
3
```

(reverse <list>) ; hoàn trả đổi ngược các phần tử từ đầu xuống cuối trong List.

Ví dụ:
```
_$ (reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

(cons <biểu thức> <list/Symbol>) ; để tạo ra một List mới bằng cách nối hai thành phần phía sau tên hàm lại với nhau.

+ Nếu thành phần cuối là **List** *thì tạo ra List mới nối phần tử đầu<biểu thức> với phần tử cuối<list> với nhau:*

```
_$ (setq A (list 1 2 3) B "object")
"object"
_$ (cons (* 5 5) A)
(25 1 2 3)
_$ (cons (+ 1 9) A)
(10 1 2 3)
```

+ Nếu thành phần cuối là **Symbol** *thì tạo ra List mới nối phần tử đầu<biểu thức> với phần tử cuối<Symbol> với nhau và cách nhau dấu chấm • (Dotted pair):*

```
_$ (cons (+ 1 9) B)
(10 . "object")
```

(append <list1> <list2>) ; hàm này nối các phần tử List2 vào cuối List1.

Ví dụ:
```
_$ (append '(1 2 3) (list "M" "N" "P" "Q"))
(1 2 3 "M" "N" "P" "Q")
```

(subst <phần tử mới> <phần tử cũ> <list>) ; hàm này thay thế một phần tử cũ trong List bằng một phần tử mới.

Ví dụ:
```
_$ (subst "NEW" 12 (list 10 11 12 13))
(10 11 "NEW" 13)
```

(mapcar <hàm func> <list1> <list2>...) ; hàm **mapcar** sắp xếp và hoàn trả các thành phần được xử lý giữa các List1, List2... bởi hàm **Func.**

Ví dụ:
```
_$ (mapcar '1+ '(10 20 30))
(11 21 31)
```

Hoàn trả kết quả là 1 List do hàm cộng thêm 1 đơn vị "1+" tạo ra.

```
_$ (setq A (mapcar '+ (list 7 8 9) '(1 2 3)))
(8 10 12)
```

Hoàn trả kết quả là 1 List do hàm cộng "+" tạo ra.

(apply <func> <list>) ; để thực thi hàm **func** với các phần tử của List.

Ví dụ:
```
_$ (apply 'strcat (list "GOOD" " MORNING" "!"))
"GOOD MORNING!"
```

(lambda <args> <biểu thức>) ; để định nghĩa một hàm không cần tên, chỉ dùng một lần tiết kiệm bộ nhớ, theo hàm này mà hàm **mapcar** thực thi. Ví dụ:

```
(defun LAM ()
 (mapcar '(lambda (x y z)
    (* x (- y z)))
```

```
        '(2 3) '(20 30) '(122 -13.3)
      )
    )
```

Hoàn trả kết quả là List: **(-204 129.9)**

8). Các hàm logic

Các hàm so sánh giữa 2 biến hay 2 phần tử gồm có:

(eq <biến1> <biến2>) ; nếu biến1 giống hệt biến2 sẽ hoàn trả **T**, còn biến1 khác biến2 thì hoàn trả **Nil**. Ví dụ:

```
_$ (eq 3.5 3.5)
T
_$ (eq x z)
nil
```

(equal <biểu thức1> <biểu thức2> <độ chính xác so sánh>) ; nếu biểu thức1 bằng biểu thức2 với độ chính xác đã định thì hoàn trả **T**.

Ví dụ:
```
_$ (equal 5.555 5.553 0.01)
T
```

(= <phần tử1> <phần tử2>) ; nếu phần tử1 giống hệt phần tử 2 thì hoàn trả **T**.Ví dụ:

```
_$ (= 5.66 5.66)
T
_$ (= B B)
T
_$ (= 2 y)
nil
```

** Chú ý: Đối với số thực máy tính xem số 0 là 1 số xấp xỉ 0, cho nên khi dùng hàm "=" để so sánh có khi lại hoàn trả* **Nil**. *Trường hợp này nên dùng hàm* **Equal** *để so sánh.*

(/= <biểu thức1> <biểu thức2>) ; nếu 2 biểu thức khác nhau sẽ hoàn trả **T**. Ví dụ:

```
_$ (/= (* 2 3) 9)
T
_$ (/= (list a b c) (list a b))
T
```

(>= <số thực1> <số thực2>) ; nếu số thực1 lớn hơn hay bằng số thực2 thì hoàn trả **T**, ngược lại sẽ hoàn trả **Nil.**

(<= <số thực1> <số thực2>) ; nếu số thực1 nhỏ hơn hay bằng số thực2 thì hoàn trả **T**, ngược lại sẽ hoàn trả **Nil.**

(> <số thực1> <số thực2>) ; nếu số thực1 lớn hơn số thực2 thì hoàn trả **T**, ngược lại sẽ hoàn trả **Nil.**

(< <số thực1> <số thực2>) ; nếu số thực1 nhỏ hơn hay số thực2 thì hoàn trả **T**, ngược lại sẽ hoàn trả **Nil.**

(and <giá trị1> <giá trị2>...) ; hàm **AND** sẽ hoàn trả **Nil** nếu bất kỳ 1 giá trị nào là **Nil**; và hoàn trả **T** khi tất cả các giá trị không có **Nil**. Ví dụ:

```
_$ (SETQ A 5 B NIL C "STRING")
"STRING"
_$ (and A B)
```



```
nil
_$ (and A C)
T
_$ (and B C)
nil
_$ (and A B C)
nil
```

(or <giá trị1> <giá trị2>...) ; hàm **OR** xử lý từ trái sang phải, nếu tìm thấy 1 giá trị khác **Nil** thì hoàn trả **T**, nếu tìm thấy tất cả các giá trị là **Nil** sẽ hoàn trả **Nil**. Ví dụ:

```
_$ (or 9 8 '() 7)
T
_$ (or Nil 3 4 m n)
T
_$ (or Nil '() Nil)
nil
```

(not <giá trị>) ; hàm **NOT** hoàn trả **T** nếu giá trị cho là **Nil**, còn giá trị cho khác **Nil** sẽ hoàn trả **Nil**. Ví dụ:

```
_$ (setq x 5 y "Blue" z Nil)
nil
_$ (not z)
T
_$ (not x)
nil
_$ (not y)
nil
```

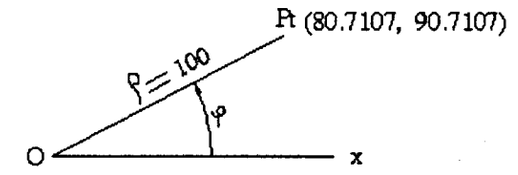
9). Các hàm hình học và lượng giác

(distance <P1> <P2>) ; tính khoảng cách giữa hai điểm P1 và P2.

(angle <P1> <P2>) ; tính góc tạo bởi đường P1P2 với trục x(Radian).

(polar <điểm Pt> <góc φ> <khoảng cách ρ>) ; hàm **polar** định toạ độ một điểm mới, so với điểm chuẩn Pt cách đoạn ρ, nằm trên đường thẳng nghiêng với trục x góc φ.

Ví dụ: **(polar '(10 20) 0.785398 100)**
(80.7107 90.7107)



(inters <P1> <P2> <P3> <P4> [seg]) ; hàm **Inters** hoàn trả toạ độ của giao điểm giữa P1P2 với P3P4. Nếu biến [seg] là **Nil** thì hai đường thẳng đó xem là dài vô tận và hoàn trả toạ độ giao điểm của chúng. Nếu [seg] nhận giá trị **T** thì hoàn trả **Nil** khi hai đoạn thẳng không cắt ở bên trong của chúng.

Ví dụ:
```
_$ (setq A '(0 0) B '(50 50))
(50 50)
_$ (setq C '(10 0) D '(50 10))
```

```
(50  10)
_$ (setq  M '(0  50)  N '(50  20))
(50  20)
_$ (inters A B C D)
nil
_$ (inters A B C D T)
nil
_$ (inters A B C D NIL)
(-3.33333  -3.33333)
_$ (inters A B M N)
(31.25  31.25)
```

(trans <Pt> <hệ cũ> <hệ mới>) ; lệnh **Trans** để chuyển hệ toạ độ cũ sang hệ toạ độ mới của toạ độ một điểm Pt. Trong đó các biến <hệ cũ> và <hệ mới> lấy giá trị như sau:

0 ứng với hệ WCS
1 ứng với hệ UCS

Ví dụ: ta chuyển điểm Pt(150, 0, 0) từ hệ UCS hiện thời sang hệ WCS:

```
_$ (trans '(150  0  0)  1  0)
(292.449   72.963   3.552)
```

(sin <góc radian>) ; hoàn trả giá trị sin góc đó.
(cos <góc radian>) ; hoàn trả giá trị cosin góc đó.
(cvunit <số> <Units cũ> <Units mới>) ; hàm **Convert units** dùng để chuyển đổi một số từ hệ đo này sang hệ đo khác. Ví dụ:

```
_$ (cvunit  25.4  "mm"  "in")
1.0
_$ (cvunit  1  "in"  "mm")
25.4
```

10). Hàm kiểm soát lỗi

Đây là cú pháp của một hàm sử lý lỗi:

(*error* <string>)

Trong đó String là câu giải thích của AutoLISP về lỗi đã phạm. Ví dụ:

```
(defun *error* (msg)
 (princ "error: ")
 (princ  msg)
 (princ)
)
```

Ta thường nhận được thông báo lỗi chương trình với các dạng sau:

```
; ***ERROR:  too few arguments
; ***ERROR:  bad arguments type:  2D/3D point
```

29.2.6. Nhập và xuất

Cũng như các ngôn ngữ lập trình khác, AutoLISP sử dụng hai dạng quản lý dữ liệu: ghi vào bộ nhớ của máy, cách thứ hai là ghi dữ liệu vào file ngoại vi trên đĩa cứng, đĩa mềm không sử dụng bộ nhớ của máy. Trong quá trình làm việc, các file dữ liệu đó được mở ra để ghi hoặc nhập vào khi AutoLISP yêu cầu.

1). Lệnh mở/đóng file:

(open <tên file> <mode>) ; lệnh mở file theo **mode**:
"r" chỉ để đọc file
"w" ghi thành file mới
"a" ghi tiếp vào file hiện thời. Khi xong việc phải đóng file lại.
(close <mô tả file>) ; lệnh đóng file
Ví dụ:

```
(setq  fil "SOMEFILE.TXT")
(setq  x (open fil "r")  ct  0)
(while (read-line x)
 (setq  ct (1+  ct))
)
(close x)
```

2). Lệnh nhập dữ liệu từ file và ghi dữ liệu vào file

Trước khi nhập hay ghi dữ liệu cần quay lại với màn hình hay máy in, chọn chế độ màn hình Tex hay Graphics:

(graphscr) ; chuyển màn hình sang chế độ đồ hoạ Graphics
(textscr) ; chuyển màn hình sang chế độ văn bản Text
(textpage) ; chuyển màn hình sang chế độ văn bản Text và xoá sạch màn hình, đưa con trỏ lên góc trái trên màn hình.

- Nhập dữ liệu từ file:

(read-line <mô tả file>) ; có mô tả file sẽ nhập dữ liệu từ file; không có mô tả file sẽ nhập dữ liệu từ bàn phím.
(read-char <mô tả file>) ; nhập mã ASCII của ký tự đọc từ file mô tả, nếu không cho mô tả file sẽ đọc từ bàn phím.

- Xuất(ghi) dữ liệu vào 1 file:

(write-line <chuỗi ký tự> <mô tả file>) ; cho mô tả file sẽ ghi chuỗi ký tự vào file, không cho mô tả file thì thì ghi chuỗi ký tự ra màn hình. Ví dụ: (write-line "Test" f)
(write-char <mã số> <mô tả file>) ; cho mô tả file thì ghi ký tự ứng với mã ASCII vào file; không cho mô tả file thì ghi ký tự ứng với mã ASCII ra màn hình.
Ví dụ: (write-char 65) → A65

3). Các hàm họ PRINT

Các hàm họ Print sẽ xuất ra màn hình, máy in hoặc ghi vào file dữ liệu.
(princ <biểu thức> [mô tả file]) ; nếu có [mô tả file] thì nội dung biểu thức được ghi vào file đó; nếu không hàm Princ sẽ in kết quả và hoàn trả biểu thức ra màn hình.
(prin1 <biểu thức> [mô tả file]) ; hàm sẽ in biểu thức ra màn hình, nếu có [mô tả file] thì ý nghĩa như đã nói ở trên. Khi in nếu là chuỗi ký tự thì đặt trong dấu "". Ví dụ:

```
Command: (setq  a 123   b '(a))
(A)
Command: (prin1 'a)
AA                      ; in nội dung biểu thức và hoàn trả
Command: (prin1 a)
123123                  ; in nội dung biểu thức và hoàn trả
Command: (prin1 b)
(A)(A)                  ; in nội dung biểu thức và hoàn trả
Command: (prin1 "Hello"  f)
```

```
"Hello"                    ; in nội dung biểu thức là dòng ký tự đặt trong ""
```

(print <biểu thức> [mô tả file]) ; giống như prin1, hàm **print** sẽ in biểu thức ra màn hình, nếu có [mô tả file] thì ý nghĩa như đã nói ở trên. Sau khi in nếu là chuỗi ký tự thì dành một khoảng trống ở cuối dòng. Ví dụ:

```
_$ (princ  "\t AutoLISP")
AutoLISP"\t AutoLISP"    ; hàm in và hoàn trả.

_$ (print  "\t AutoLISP")
        ; hàm để lại 1 dòng trống không in
"\t AutoLISP" "\t AutoLISP"    : hàm in và hoàn trả.

_$ (princ  (+ 200  55))
255 255                  ; hàm in và hoàn trả

_$ (print  (+ 200  55))
; hàm để lại 1 dòng trống không in
255  255                  ; hàm in và hoàn trả.

_$ (setq f (open "Textfile.txt"  "w"))    ; mở file
_$ (princ "Đây là file văn bản"  f)       ; ghi biểu thức vào file.
```

* Chú ý:

+ Hàm **Princ** sẽ in ra chuỗi ký tự không có dấu ngoặc kép "", còn hàm **Print** sẽ in ra chuỗi ký tự ở trong dấu ngoặc kép "". Ví dụ:

```
_$ (princ "Hà Nội")
Hà Nội"Hà Nội"        ; Princ  in ra và hoàn trả biểu thức
_$ (print "Hà Nội")
                      ; bỏ trống 1dòng trước khi in ra
"Hà Nội" "Hà Nội"     ; Print  in ra và hoàn trả biểu thức
```

+ Hàm **(princ)** hay được dùng để xoá màn hình AutoCAD khi chạy chương trình.

*** Ví dụ 1:** viết chương trình giải phương trình bậc 2 có các hệ số a, b, c.

```
(defun  BAC2  (a  b  c  / delta)
 (setq a (getreal  "Cho a:"))
 (setq b (getreal  "Cho b:"))
 (setq c (getreal  "Cho c:"))
 (setq delta (- (* b  b) (* 4  a  c)))
 (if (>= delta 0)
  (progn          ;;nếu Delta>=0 là đúng sẽ tiếp tục
   (princ "\nCó nghiệm thực")
   (setq b (- 0  b))
   (princ "\nX1=")
   (princ (/ (+ b  (sqrt  delta))  (* 2   a)))
   (princ "\nX2=")
   (princ (/ (- b (sqrt  delta)) (* 2  a)))
  )               ;; kết thúc Progn
 ;;; bây giờ nếu Delta<0 sẽ tiếp như sau
  (princ "\nKhông có nghiệm thực")
 )                                    ; kết thúc if
) ;;;;;;;;;;;; Hết chương trình.-------------------------
```


*** Ví dụ 2:** dưới đây là 1 chương trình mở file để đọc và in ra màn hình.

```
(defun  Printf  (s / c  i)
  (setq  i (open   s  " r"))    ; mở file để đọc
  (if  i
      (progn
          (while  (setq  c (read-line  i))    ; đọc 1 dòng
          (princ  "\n")
          )   ; kết thúc while
      (close  i)   ; đóng file
      )   ; kết thúc Progn
  (princ  (strcat  "Không mở được file"  s))
  )   ; kết thúc if
 (princ)   ; xoá màn hình AutoCAD trước khi chạy chương trình
 )  ;;;;;;;;;; Hết chương trình----------------------------------------
```

*** Ví dụ 3:** chương trình tính cạnh huyền của tam giác vuông, biết 2 cạnh x, y.

```
(defun  HUYEN ( )       ;; định nghĩa hàm HUYEN
 (setq x (getreal  "Cho canh x:"))  ;; nhập độ lớn cạnh x(số thực)
 (setq y (getreal  "Cho canh y:"))  ;; nhập độ lớn cạnh y(số thực)
 (setq k (+  (* x  x)  (* y  y)))   ;; k là tổng bình phương 2 cạnh x, y
 (princ  "\nGia tri l=")            ;; in kết quả độ dài l cạnh huyền tính được
 (setq l (sqrt  k))         ;; l bằng căn bậc 2 của k
)
```

Sau khi chạy ví dụ với x=3, y=4 cho kết quả:

```
_$ (huyen)
Gia tri l=5.0
```

*** Chú ý:** chương này ta đã khảo sát các lệnh cơ bản hay dùng nhất của AutoLISP. Các bạn có thể tìm hiểu thêm các lệnh khác trong mục **Help** của AutoLISP như sau:

Menu VLISP> Help> Visual LISP Help Topics(F1)>
>Help Topics> AutoLISP Reference

Phần trên ta đã sử dụng màn hình là một thiết bị in ra, tức là sử dụng thiết bị xuất mặc định là **CON** (màn hình). Nếu ta muốn xuất tài liệu ra máy in(Printer), hoặc cửa song song thì cần sử dụng thêm lệnh khác sau đây:

Thiết bị xuất:	**Phần cứng thiết bị:**
PRN	màn hình, bàn phím
COM1, COM2	cổng nối tiếp 1, 2
LPT1 LPT2	cổng song song 1, 2
NUL	không có thiết bị xuất /nhập

Giả sử muốn xuất ra máy in ta cần đăng ký máy in giống như mở 1 file(thay tên file bằng tên máy in). Ví dụ:

```
(setq A (open  "PRN"  "w")
;;;;; thực hiện các lệnh khác
(close  A)  ;;;;; đóng máy in
```

=== * ===

CHƯƠNG 30
CẤU TRÚC TRONG CHƯƠNG TRÌNH AUTOLISP

Cấu trúc trong chương trình AutoLISP cũng có đủ các đặc trưng như trong các ngôn ngữ lập trình khác: *If, While, Cond, Progn.* Ta khảo sát từng dạng sau đây.

30.1. HÀM ĐIỀU KIỆN (IF)

If là hàm điều kiện rất thường dùng trong chương trình AutoLISP. Ý nghĩa của hàm đó là: ***Nếu điều kiện nêu ra là đúng(T) thì thực hiện theo biểu thức đầu; còn điều kiện nêu ra là sai(Nil) thì thực hiện biểu thức sau.*** Cú pháp hàm **If**:

(if <điều kiện> <thực hiện biểu thức nếu đúng> <thực hiện nếu sai>)

* Ví dụ1: **Command: (if (= 1 3) "YES!!" "no.")**
"no."
Command: (if (= 2 (+ 1 1)) "YES!!")
"YES!!"
_$ (if (< 5 9) "True" "Fale")
"True"

* Ví dụ 2:

```
;;; DRAWLINE.LSP là chương trình vẽ đoạn thẳng
;;; Function: drawline
(defun drawline (/ pt1 pt2)  ;; các biến cục bộ pt1 and pt2 được khai báo
 ;; người sử dụng gán 2 điểm
 (setq pt1 (getpoint "\nCho điểm đầu của line: ")
       pt2 (getpoint pt1 "\nCho điểm cuối của line: ")
 ) ;;; hết setq
 ;; hàm if để kiểm tra xem 2 điểm có tồn tại không
 (if (and pt1 pt2)
  (command "_.line" pt1 pt2 "")
  (princ "\nInvalid or missing points!")
 ) ;;; hết if
 ;; bỏ qua bất cứ giá trị hoàn trả nào của hàm-thoát ra nhẹ nhàng- exit 'quietly
 (princ)
)
```

* Ví dụ 3: viết chương trình tính nghiệm của phương trình bậc 2: $ax^2 + bx + c = 0$

```
(defun bac2 (a b c / delta)
 (setq a (getreal "Cho a:"))
 (setq b (getreal "Cho b:"))
 (setq c (getreal "Cho c:"))
 (setq delta (- (* b b) (* 4 a c)))
 (if (>= delta 0)
  (progn          ;nếu Delta>=0 là đúng sẽ tiếp tục
   (princ "\nCó nghiệm thực")
   (princ "\nX1=")
   (princ (/ (+ (- 0 b) (sqrt delta)) (* 2 a)))
   (princ "\nX2=")
```



```
   (princ (/ (- (- 0 b) (sqrt delta)) (* 2 a)))
  ) ;;; kết thúc Progn
 ;;; bây giờ nếu Delta<0 sẽ tiếp như sau
  (princ "\nKhông có nghiệm thực")
 ) ;;; kết thúc if
) ;;; hết chương trình ----------------------------
```

Ví dụ 4: hàm If được mô tả trong chương trình sau đây so sánh 2 khoảng cách xuất phát từ 2 điểm p1 và p2 trên màn hình. Chương trình viết trong cửa sổ Text Editor của VLISP (hình 30.1).

TESTTING.LSP

```
(deffun TESTTING (/ x y)
        (setq p1 (getpoint "Diem dau tren man hinh:"))
        (setq p2 (getpoint "Diem thu 2 tren man hinh:"))
        (setq x (getdist p1 "Khoang cach tu p1:"))
        (setq y (getdist p2 "Khoang cach tu p2:"))
        (iff (= x y)
             (prompt "\nChieu dai bang nhau")
             (prompt "\nChieu dai khong bang nhau")
        )  ;; hết if
        (princ)
) ;; hết deffun
```

Hình 30.1: Chương trình Testting.

Chạy chương trình:

_$ (TESTTING)
Diem dau tren man hinh: **0,0**
Diem thu 2 tren man hinh: **100,0**
Khoang cach tu p1: **50**
Khoang cach tu p2: **60**
Chieu dai khong bang nhau.

_$ (TESTTING)
Diem dau tren man hinh: **0,0**
Diem thu 2 tren man hinh: **100,0**
Khoang cach tu p1: **99**
Khoang cach tu p2: **99**
Chieu dai bang nhau.

30.2. HÀM LỰA CHỌN (COND)

Hàm **Cond** dùng tương tự như hàm **If** nhưng hàm Cond có thể xử lý với nhiều điều kiện thử. Nghĩa là hàm Cond có thể thay cho nhiều hàm If một lúc và hoàn trả giá trị tính sau cùng. Cú pháp:

(cond
điều kiện 1 đúng thì thực hiện lệnh
điều kiện 2 đúng thì thực hiện lệnh
.....
điều kiện n đúng thì thực hiện lệnh
)

* Ví dụ 1:

```
;;;;; chương trình tính giá trị tuyệt đối của số a
_$ (defun abso (a)
   (cond
    ((minusp a) (- a))
    (t a)
   )
  ) ;;;;; hết chương trình
```

Bây giờ chạy chương trình:

```
_$ (abso -58)
58
_$ (abso 99)
99
```

* Ví dụ 2: giả sử ta muốn biết chương trình cần tải (defun C:XE ...) đã có chưa:

```
;;; dùng if sẽ viết như sau ----------------------------------
(if (not C:XE) (load "BAC2") (princ "XE đã có"))
;;; nếu dùng hàm Cond sẽ viết như sau----------------------
(cond
 ((not C:XE) (load "BAC2"))
 (C:XE ( princ "BAC2"))
)
```

* Ví dụ 3:

```
;;; khai căn bậc 2 ------------------------------------------------------------
(defun can2 (x / y c cl) ;;; hàm có 1 biến chung và 3 biến cục bộ
 (if (or (= 'REAL (type x)) (= 'INT (type x)))
  (progn
   (cond (( minusp x) 'Bien am')
    (( zerop x) 0.0)
    (t (setq y (/ (+ 0.154116 (* x 1.893872)) (+ 1.0 (* x 1.04788))))
     (setq c (/ (- y (/ x y)) 2.0))
     (setq cl 0.0)
      (while (not (equal c cl))
      (setq y (- y c))
      (setq cl c)
      (setq c (/ (- y (/ x y)) 2.0))
      ) ;;; hết while
    )
   ) ;;; hết cond
  ) ;;; hết progn
   (Progn
    (princ "Tham số sai kiểu")
    (princ)
   ) ;;; hết progn
) ;;; hết if
) ;;; hết defun
;;; ---------------------------------------------
```


30.3. HÀM PROGN

Hàm **Progn** sẽ gộp các biểu thức lại thành một biểu thức cùng thực hiện theo một cách thức mà Progn quy định. Hàm progn cho phép tiến hành nhiều hàm **If** ở trong đó. Cú pháp:

(progn <biểu thức>...)

* Ví dụ 1:

```
(defun pro1 (a b)
 (if (= a b)
  (progn
   (princ "\nA = B ")
   (setq a (+ a 10) b (- b 10))
  ) ;;; hết progn
 ) ;;; hết if
) ;;; hết defun ----------------------------------------
```

Chạy chương trình ta có:

```
_$ (pro1 8 8)
A = B -2
_$ (pro1 2 9)
nil
_$ (pro1 m m)
A = B
```

* Ví dụ 2:

```
(defun C:PRO2 ( )
 (if test      ;;; test là điều kiện cho hàm if
  (progn     ;;; nếu là T sẽ tiếp tục
   (setvar "ORTHOMODE" 1)
   "TRUE"    ;;; hoàn trả TRUE
  ) ;;; hết progn
  (progn          ;;; nếu là Nil sẽ tiếp tục
   (setvar "ORTHOMODE" 0)
   "FALSE"     ;;; hoàn trả FALSE
  ) ;;; hết progn
 )    ;;; hết if
) ;;; hết defun ------------------------------
```

Chạy chương trình ta có:

```
_$ (C:PRO2)
"FALSE"
```

30.4. VÒNG LẶP HỮU HẠN(REPEAT)

AutoLISP có hai loại vòng lặp hay dùng là: lặp biết trước số vòng lặp n (Repeat), lặp không biết trước số lần(While). Cú pháp lệnh **Repeat**:

(repeat <số lần lặp n> <biểu thức>)

* Ví dụ 1:

```
(defun C:REPEAT1 ()
 (setq m 1 n 100)
 (repeat 5
  (setq m (+ m 1))
  (setq n (+ n 100))
 ) ;;; hết repeat
)
```

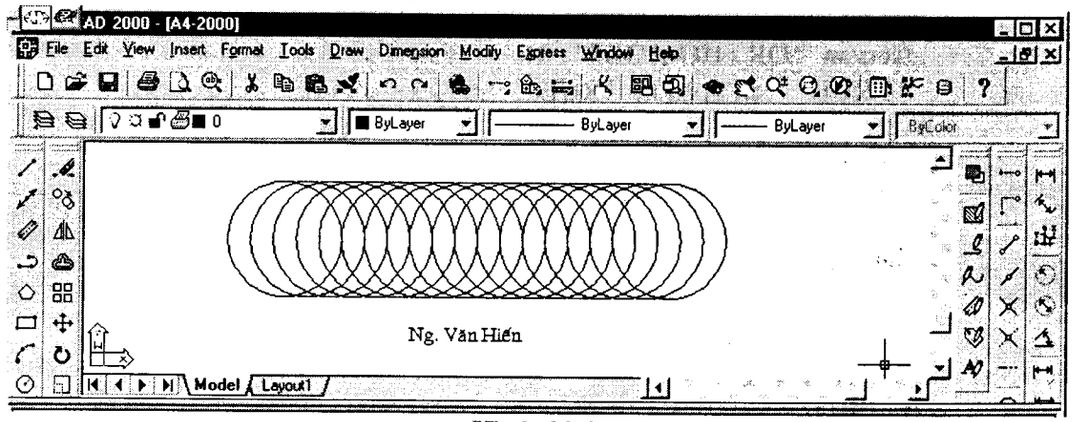
Chạy chương trình:

```
_$ (C:REPEAT1)
600
```

* Ví dụ 2:

```
;;;;;;; chương trình vẽ 18 vòng tròn với các tâm trên x và cách nhau 10 đơn vị,
;;;;;;; bán kính cùng bằng 25 đơn vị.
(defun repeat2 ()
 (setq pt '(50 50 0))  ;;; vẽ từ gốc điểm (50, 50, 0)
 (repeat 18            ;;; lệnh lặp 18 lần
  (command "COLOR" 3)          ;;; định màu vẽ 3=green
  (command "CIRCLE" pt 25)  ;;; vẽ tròn tâm pt, bán kính 25 đơn vị
  (setq pt (list (+ 10 (car pt)) 50 0))   ;;; các tâm tròn tăng x là 10 đơn vị
 ) ;;; hết repeat
 (command "MOVE" "ALL" "" '(50 50 0) PAUSE)
 (COMMAND "ZOOM" "ALL")
 (princ "Kết thúc. Chào bạn!")
)
```

Chạy chương trình ta có kết quả như trên hình 30.2.



Hình 30.2

30.5. VÒNG LẶP KHÔNG BIẾT SỐ LẦN(WHILE)

Hàm lặp có điều kiện **While** khảo sát điều kiện cho, nếu điều kiện khác **Nil** thì thực hiện biểu thức. Công việc tiến hành lặp lại mãi cho tới khi điều kiện là **Nil** thì dừng lại.

(while <điều kiện> <biểu thức>...)

* Ví dụ 1:

```
(defun C:WHILE1 (*)
 (setq count 2 data '(a b c d))
 (while (< count 6)
  (princ (nth count data))
  (setq count (1+ count))
 ) ;;; hết while
) ;;; hết defun ----------
```

Chạy chương trình:

```
_$ (C:WHILE1)
CDnilnil6
```

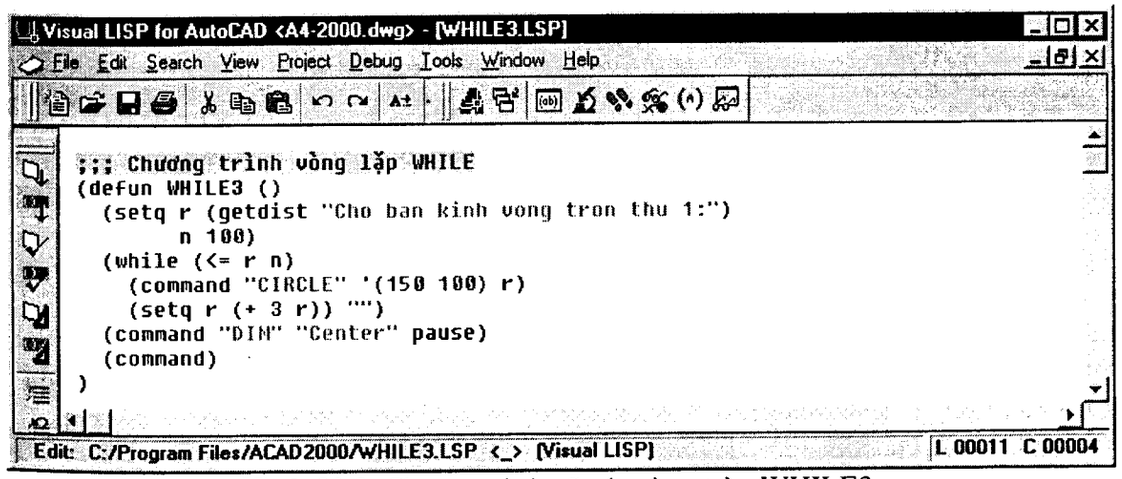
* Ví dụ 2:

```
(defun while2 ( )
 (setq pt '(0 0) i 1)
 (while (<= (setq i (1+ i)) 15)  ;;; lặp khi i<= 15
  (command "COLOR" 2)
  (command "CIRCLE" pt 30)
  (setq pt (list (+ 40 (car pt)) 0))
 ) ;;; hết while
 (command "MOVE" "All" "" '(0 0) pause)
 (command "ZOOM" "All")
 (princ "Vẽ xong")
) ;;; hết defun ----------
```

Chạy chương trình:

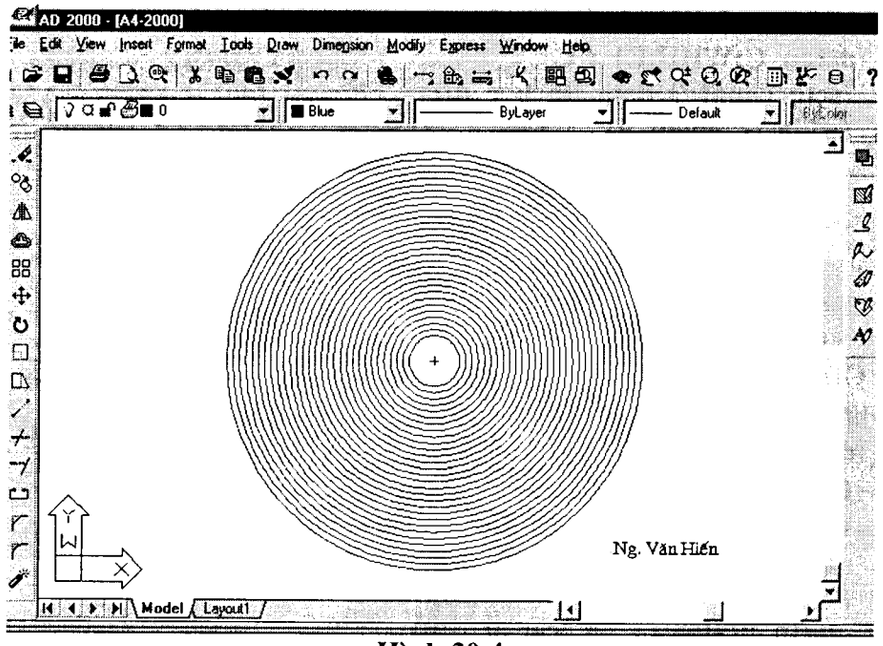
```
_$ (while2)
Vẽ xong"Vẽ xong"
```

* Ví dụ 3: điều kiện được cho trước, vòng lặp của chương trình sẽ tiếp tục thực hiện cho đến khi nào không còn thoả mãn điều kiện đã cho nữa. Ví dụ dưới đây là chương trình vẽ các vòng tròn đồng tâm với bán kính thay đổi từ 10 đơn vị đến 100 đơn vị (hình 30.3).



Hình 30.3: Chương trình vẽ các vòng tròn WHILE3.

Sau khi chạy chương trình cho r1=12 ta có hình vẽ trên hình 30.4.



Hình 30.4

30.6. HÀM FOREACH (XỬ LÝ LIST)

Hàm **Foreach** tách từng phần tử theo thứ tự trong danh sách và dùng nó lần lượt làm biến cho biểu thức, hoàn trả kết quả của biểu thức cuối cùng. Cú pháp:

(foreach <symbol> <list> <biểu thức>)

* Ví dụ 1:

```
Command: (foreach n '(a b c) (print n))
A
B
C C
```

Hàm này in ra từng phần tử trong List và hoàn trả c. Lệnh này tương đương 3 hàm:

```
(print a)
(print b)
(print c)
```

* Ví dụ 2:

```
(setq A (list 100 "Chào" (/ 45 9)))
(foreach W A (princ W))
```

Thực thi các biểu thức đó ta có:

```
(100 "Chào" 5) 100Chào5
5
```

30.7. LỆNH COMMAND TRONG AUTOLISP

Biểu thức **Command** trong AutoLISP dùng để gọi các lệnh của AutoCAD. Các ví dụ ở các phần trên ta đã sử dụng hàm Command để gọi các lệnh: LINE, CIRCLE, MOVE... của AutoCAD. Cú pháp:

(command "tên lệnh" <các lựa chọn của lệnh>)

Trong "tên lệnh" có thể là String, số hoặc một điểm.

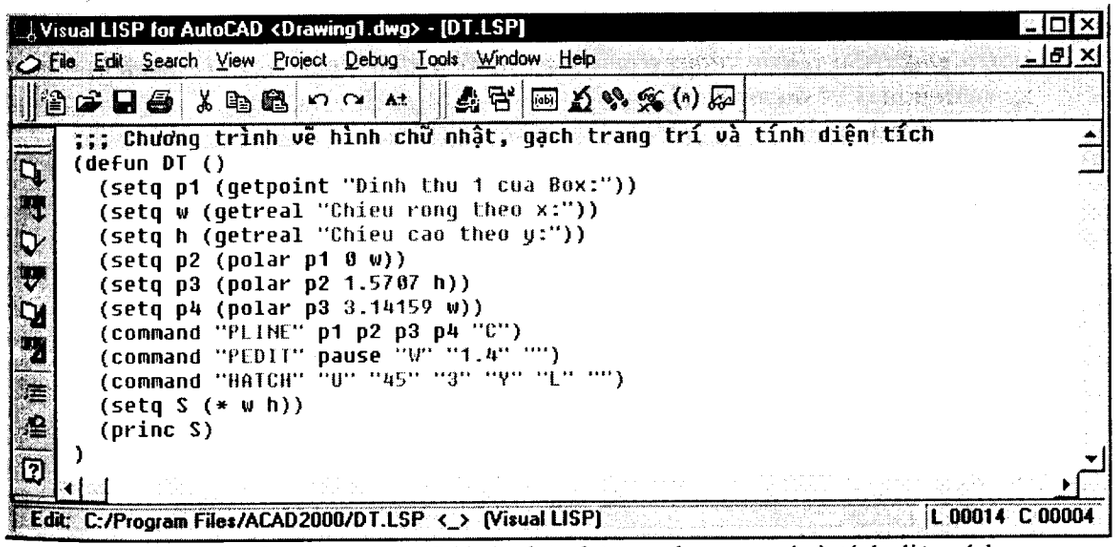
* Ví dụ1:

```
;;; vẽ đoạn thẳng từ (0,0,0) tới (100, 50, 0)
(command "LINE" '(0 0 0) '(100 50 0))
```

* Ví dụ 2:

```
;;; vẽ đường gấp khúc thẳng
(defun HLINE (/ pt)
 (setq pt (getpoint "\nFirst point:"))
 (command "COLOR" 3)
 (COMMAND "LINE")
 (while (/= pt Nil)   ;;; điều kiện pt /= Nil
  (command pt)        ;;; nhập điểm đầu cho Line
  (setq pt (getpoint pt "\nNext point:"))   ;;; nhập điểm cho biến pt
 )  ;;; hết while
 (command "")   ;;; kết thúc lệnh Line
)   ;;; hết defun ----------
```

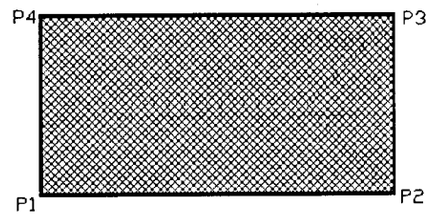
* Ví dụ 3: dưới đây là chương trình có sử dụng lệnh Command, các hàm nhập dữ liệu để vẽ hình chữ nhật, tô đậm, gạch trang trí và tính diện tích của nó. Chương trình được soạn thảo trong cửa sổ Text Editor của VLISP (hình 30.5).



Hình 30.5: Chương trình vẽ hình chữ nhật, gạch trang trí và tính diện tích.

Sau khi chạy chương trình ta được thông báo kết quả tính diện tích(S) như dòng dưới đây và kết quả hình vẽ như hình 30.6.

```
_$ (DT)
30000.0
```



Hình 30.6

*** Chú ý:**

1). **(command "")** ;;; tương đương với ENTER

2). **(command)** ;;; tương đương với CTRL+C để kết thúc lệnh.

3). **PAUSE** đưa vào làm biến của hàm Command để tạm dừng thực thi chương trình chờ người sử dụng nhập dữ liệu(khi biến TEXTEVAL ≠ 0). Ví dụ:

_$ (command "CIRCLE" '(10 88) pause)

4). Muốn cho AutoCAD thực hiện các lệnh của nó mà không cần thông báo gì, ta đặt biến hệ thống: **Cmdecho = 0**

5). Chú ý có một số lệnh của AutoCAD: DTEXT, SKETCH, PLOT, SCRIPT lại đọc dữ liệu từ bàn phím hoặc từ Digitizer trực tiếp. Do vậy chúng không thể là đối số cho lệnh Command trong AutoLISP.

30.8. CÁC VÍ DỤ VỀ CÁC CHƯƠNG TRÌNH ĐỒ HOẠ

*** Ví dụ 1:** trong một chương trình AutoLISP có thể gồm nhiều chương trình con(*Subroutine*). Mỗi Subroutine thực hiện một công việc cụ thể và chương trình chính chạy sẽ gọi từng Subroutine ra để xử lý. Dưới đây là chương trình vẽ các đường tròn với các đường kính ghi trong 1 danh sách. Như vậy ta cần các Subroutines là: chương trình lập ra danh sách đường kính **List**, chương trình đọc đường kính ra từ danh sách **Read** và chương trình vẽ đường tròn **Draw**.

```
;;; Chương trình vẽ các đường tròn với các đường kính ghi trong 1 danh sách.
;;; subroutine1 vẽ đường tròn ----------
(defun draw ( )
 (readd)
 (foreach pt (listd)
  (command "CIRCLE" pt "D" #readd)
 ) ;;; hết foreach
)  ;;; hết defun
;;; subroutine2 lập danh sách ----------
(defun listd (/ pt pl)
 (while (setq pt (getpoint "\nInput point:"))
  (setq pl (append pt))
 ) ;;; hết while
)  ;;; hết defun
(if #readd Nil (setq #readd 0.5))
;;; subroutine3 đọc đường kính ra----------
(defun readd (/ tmdia)
 (prompt "\nReadd:")
 (princ #readd)
 (setq tmdia (getdist "Distance:"))
 (if tmdia (setq #readd tmdia))
 ) ;;; hết prompt
;;; chương trình chính ----------
(listd)
(draw)
;;; hết ----------
```


*** Ví dụ 2:** Sau khi vẽ các đối tượng của bản vẽ, ta thường phải thay đổi bề rộng nét vẽ(tô đậm) của 1 hay 1 nhóm đối tượng. Chương trình sau đây giúp ta làm việc đó.

```
;;; Thay đổi bề rộng nét vẽ width(WD).
(defun C:WD ( )
 (setvar "cmdecho" 0)
 (GC)
 (setq s(ssget))
 (setq n(sslength s))
 (setq z(getreal " New width:"))
 (setq m 1)
 (setq d 0)
 (While (<= m n)
  (setq e1(ssname s d))
  (command "SELECT" e1 "")
  (setq CH(ENTGET e1))
  (setq K(CDR(ASSOC 0 ch)))
;;;----------
  (cond ((= K "LINE")
   (PROGN
   (command "PEDIT" (SSGET "P") "" "W" Z "X")
   (setq m(+ 1 m))
   (setq d(+ 1 d))
   ) ;; hết progn
  )
;;; ----------
  ((= K "ARC")
   (PROGN
    (command "PEDIT" (SSGET "P") "" "W" Z "X")
    (setq m(+ 1 m))
    (setq d(+ 1 d))
   ) ;; hết PROGN
  )
;;; ----------
  ((= K "POLYLINE")
   (PROGN
   (command "PEDIT" (SSGET "P") "W" Z "X")
   (setq m(+ 1 m))
   (setq d(+ 1 d))
   ) ;; hết PROGN
  )
;;;----------
  ((= K "CIRCLE")
   (PROGN
   (setq p1 '(0.10 0.001))
   (setq p2 '(0.10 0.0))
   (command "BREAK" (SSGET "P") p1 p2)
```

```
        (command "PEDIT" (SSGET "P") ""  "cl" "W"  Z "X")
        (setq m(+ 1  m))
        (setq d(+ 1  d))
        ) ;; hết PROGN
       )
;;;-------------------------------------------------------------------
      ((= K  "ELLIPSE")
       (PROGN
       (setq p1  '(0.10  0.001))
       (setq p2  '(0.10  0.0))
       (command "BREAK" (SSGET "P") p1  p2)
       (command "PEDIT" (SSGET "P") "" "cl" "W"  Z "X")
       (setq m(+ 1  m))
       (setq d(+ 1  d))
       ) ;; hết PROGN
      )
      (T  (progn (prompt "\n not change" )
            (setq m(+ 1  m))
            (setq d(+ 1  d))
          ) ;; hết progn
      )
    ) ;; hết cond
  ) ;; hết while
  (command  "REDRAW")
  (princ)
 ) ;; hết defun----------------------------------------------------------------------
```

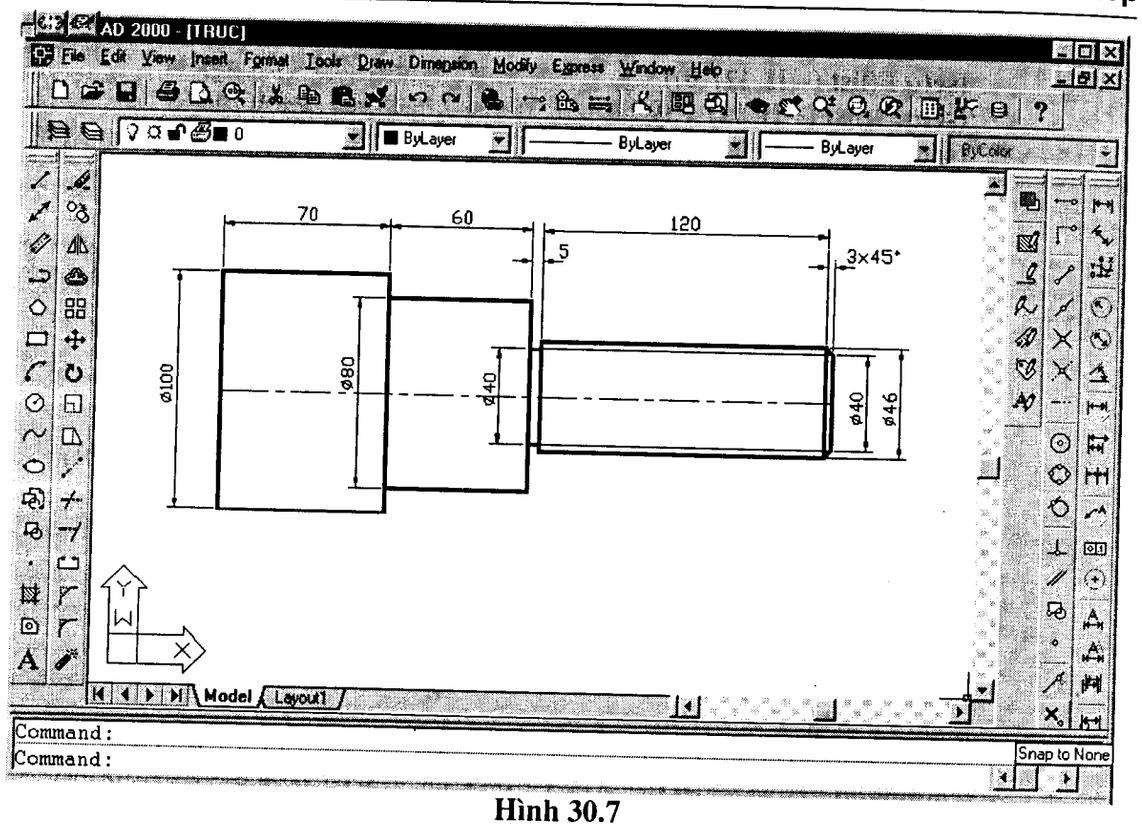
* **Ví dụ 3:** ở ví dụ này bạn chú ý theo dõi cách nhập dữ liệu và cách nhập các biến là các lệnh AutoCAD cho hàm Command của AutoLISP.

```
;;; Chương trình vẽ trục bậc cơ khí.
(defun TRUC ()      ;; định nghĩa hàm và nhập dữ liệu
 (setq r1(getdist "Cho r1:"))
 (setq r2(getdist "Cho r2:"))
 (setq r3(getdist "Cho r3:"))
 (setq r4(getdist "Cho r4:"))
 (setq r5(getdist "Cho r5:"))
 (setq l1(getdist "Cho l1:"))
 (setq l2(getdist "Cho l2:"))
 (setq l3(getdist "Cho l3:"))
 (setq l4(getdist "Cho l4:"))
 (setq l5(getdist "Cho l5:"))
  (setq O(list 0  0))
  (setq A(list 0  r1))
  (setq B(list l1  r1))
  (setq C(list l1  0))
  (setq D(list l1  r2))
```



```
  (setq E(list (+ l1  l2)  r2))
  (setq F(list (+ l1  l2)  0))
  (setq G(list (+ l1  l2)  r3))
  (setq H(list (+ l1  l2  l3  l4  l5)  r5))
  (setq I(list (+ l1  l2  l3  l4  l5)  0))
  (setq K(list (+ l1  l2  l3)  0))
  (setq L(list (+ l1  l2  l3)  r4))
  (setq M(list (+ l1  l2  l3  l4)  r4))
  (setq N(list (+ l1  l2  l3  l4)  0))
  (setq P(list (+ l1  l2  l3)  r3))
 ;; vẽ hình ----------------------------------------------------------------
 (COMMAND "PLINE" O "W" "1" "1" "")
 (COMMAND "PLINE" O A B C D E F G P K L M N "" "")
 (command "PLINE" M H I "" "")
 (command "MIRROR" "ALL" "" O I "N" "")
  (command "LINE" P H "" "")
  (command "MIRROR" "L" "" O I "N" "")
 (command "LINE" O I "")
 (command "CHANGE" "L" "" "P" "LT" "center" "")
 ;; ghi kích thước ---------------------------------------------------------------
 (command "DIM" "HOR" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "HOR" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "HOR" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "HOR" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "HOR" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "VER" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "VER" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "VER" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "VER" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command "DIM" "VER" "INT" PAUSE "INT" PAUSE PAUSE "")
 (command)
(command "DDEDIT" PAUSE PAUSE PAUSE PAUSE PAUSE
                                                PAUSE "")
  ;; dùng lệnh DDEDIT để sửa đổi chữ số kích thước: Φ100, Φ80, Φ40....
) ;; hết ----------------------------------------------------------------------
```

Sau khi chạy chương trình ta có bản vẽ trục bậc như hình 30.7.



Hình 30.7

*** Ví dụ 4:**

```
;;; chương trình vẽ đường cong CYCLOID.
(defun CYCLO ( )
 (setq r (getreal "Radius of circle:"))
 (setq k 0)
 (setq x 0)
 (setq m1 (list 0 0))
 (command "CIRCLE" (list 0 r) r)
 (while (<= x (* 2 pi r))
  (setq x (* r (- k (sin k))))
  (setq y (* r (- 1 (cos k))))
  (setq m2 (list x y))
  (command "LINE" m1 m2 "")
  (setq m1 m2)
  (setq x (+ x 0.1))
  (setq k (+ k 0.1))
 ) ;; hết while
) ;; hết defun
;;; ----------------------------------------------------------------------
```

Sau khi chạy chương trình (cho bán kính đường tròn sinh Φ30) ta có đường Cycloit như hình 30.8.

Hình 30.8

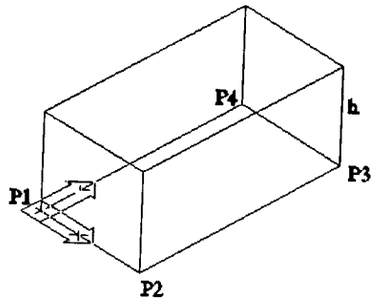
* CHƯƠNG TRÌNH AUTOLISP TRONG 3D

Ta có thể sử dụng hầu như tất cả các hàm của AutoLISP trong không gian3 chiều(3D). Do đó viết chương trình AutoLISP 3D cũng tương tự như trong 2D. Để tạo một mô hình vật thể 3D có nhiều cách khác nhau. Dưới đây nêu ra 1 cách thông qua các ví dụ:

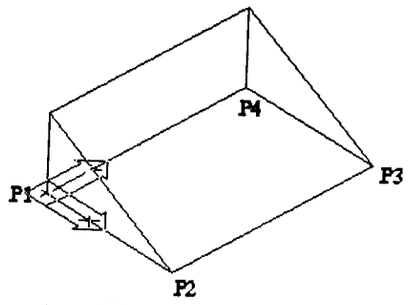
*** Ví dụ 5:**

```
;;; Lập chương trình 3D vẽ khối hộp BOX.
;;; chương trìn con 1: nhập dữ liệu ----------------------------------
(defun NHAP ( )
 (setq p1 (getpoint "Cho diem p1:")
       p3 (getpoint "Cho diem doi dien cua hinh chu nhat day Box:"))
)
;;; chương trìn con 2: nhập dữ liệu cần tính toán ---------------------
(defun TINH ( )
 (setq p2 (list (car p3) (cadr p1))
       p4 (list (car p1) (cadr p3)))
)
;;; chương trìn con 3: vẽ hình chữ nhật đáy BOX -------------------
(defun DAYBOX ( )
 (command "LINE" p1 p2 p3 p4 "C")
)
;;; chương trình chính ------------------------------------------------
(defun BOX1 (/ p1 p2 p3 p4)
 (NHAP)    ;; chương trình con1
 (TINH)    ;; chương trình con2
 (setq h (getreal "Cho chieu cao Box:"))
 (command "CHANGE" "Last" "" "Properties" "Thickness" h "")
                          ;; nâng chữ nhật đáy Box lên chiều cao h
 (command "3DFACE" p1 p2 p3 p4 ""
    "3DFACE" ".xy" p1 h ".xy" p2 h ".xy" p3 h ".xy" p4 h "")
 (command "LINE" p1 ".xy" p1 h ""
    "LINE" p2 ".xy" p2 h ""
    "LINE" p3 ".xy" p3 h ""
    "LINE" p4 ".xy" p4 h "")
)
;;; hết ----------------------------------------------------------------------
```

Sau khi chạy chương trình ta có hình Box như hình 30.9.



Hình 30.9: Box



Hình 30.10: Wedge

Đối với hình nêm-hay hình lăng trụ (Wedge), ta cũng sử dụng các chương trình con: **NHAP, TINH, DAYBOX** để nhập và tính toán các dữ liệu và vẽ hình chữ nhật đáy như của chương trình AutoLISP vẽ Box ở trên. Chương trình vẽ Wedge phần đầu có 3 chương trình con đó, sau đó là chương trình chính tiếp theo như sau:

```
;;; chương trình chính vẽ hình nêm Wedge ----------------------------------
(defun WEDGE1 (/ p1 p2 p3 p4)
 (NHAP)     ;; chương trình con1
 (TINH)     ;; chương trình con2
 (setq h (getreal "Cho chieu cao Box:"))
 (command "CHANGE" "Last" "" "Properties" "Thickness" h "")
     ;; nâng cạnh chữ nhật đáy Box  p1p4  lên chiều cao h
 (command "3DFACE" p1 p4 ".xy" p4 h ".xy" p1 h ".xy" p2 ".xy" p3 ""
      "3DFACE" p1 p2 ".xy" p1 h p1 "" "COPY" "L" "" P1 P4
      "LINE" p2 p3 "")
)
;;; hết ----------------------------------------------------------------------
```

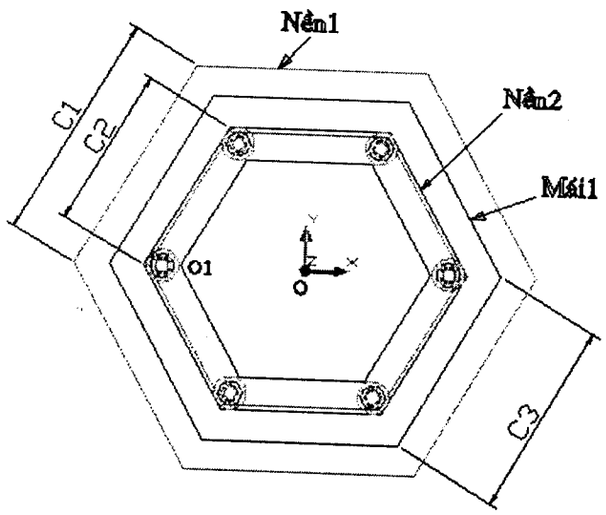
Sau khi chạy chương trình ta có hình Wedge như hình 30.10 ở trên.

*** Ví dụ 6:** ở ví dụ này bạn chú ý theo dõi cách nhập dữ liệu và cách nhập các biến là các lệnh AutoCAD cho hàm Command của AutoLISP.

```
;;; Chương trình vẽ quán trà công viên --------(hình 30.12)
(defun QUANTRA ( )
 (setq c1 (getdist "Cho canh day da giac nen nha thu 1:"))
 (setq c2 (* 0.7 c1))  ;; cạnh đáy đa giác nền thứ 2 ------------(hình 30.11)
 (setq r1 (/ c2 10))   ;; bán kính đế trụ 1 của cột
 (setq r2 (* r1 (sin (/ pi 4))))  ;; bán kính đế trụ 2 của cột
 (setq r3 (* r2 (sin (/ pi 4))))  ;; bán kính đế trụ 3 của cột
 (setq r4 (/ (* r3 (sqrt 2)) 2))  ;; bán kính cột
 (setq c3 (/ (+ c1 c2) 2))  ;; cạnh đa giác mái 1
 (setq c4 (/ (* 8 c3) 10))  ;; cạnh đa giác mái 2
 (setq c5 (* 0.8 c4))       ;; cạnh đa giác mái 3
 (setq h1 (getdist "Chieu cao nen 1:"))
 (setq o (list 0 0))   ;; tâm đa giác nền và mái
 (setq o1 (list (- c2 (/ r1 (sin (/ pi 3)))) 0))  ;; o1 là tâm chân cột ở bên trái
```



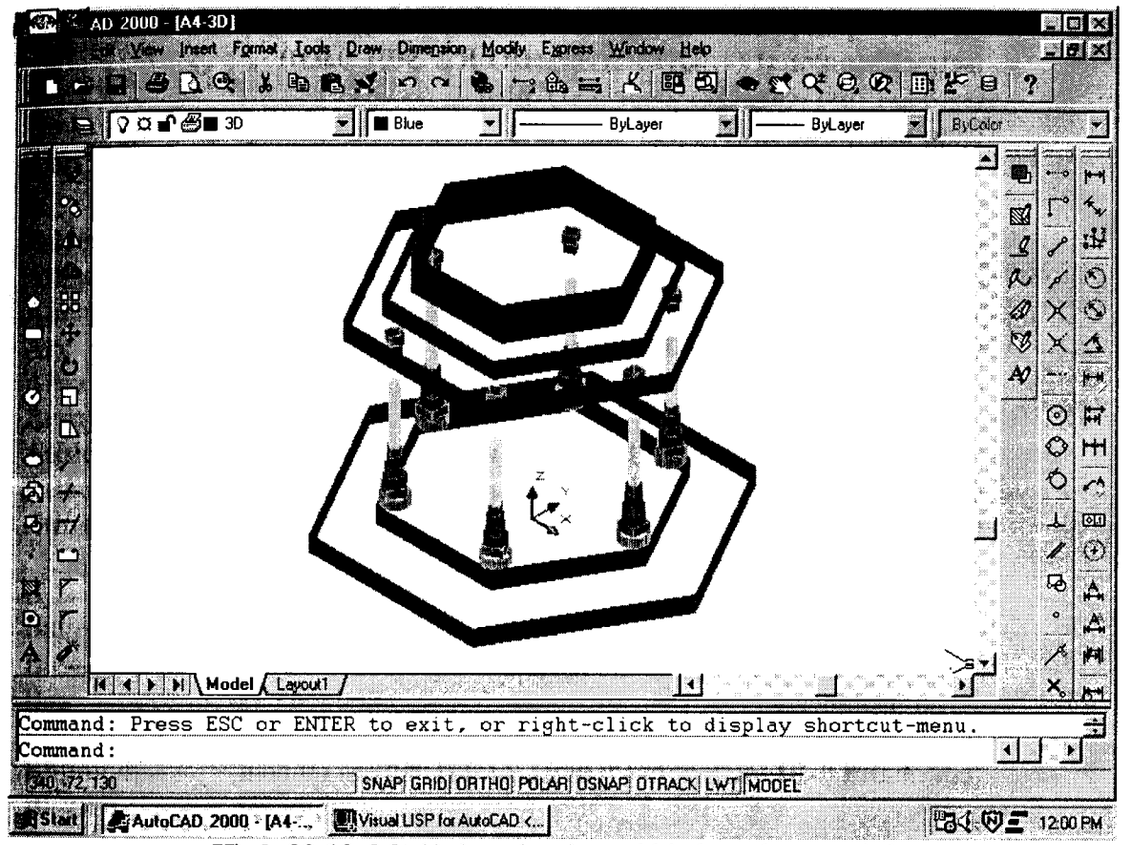
```
;;; subroutine1 vẽ nền 1 ----------------------------------------------------
 (command "ELEV" "0" h1)  ;; cho z=0 và nâng lên cao khoảng h1
 (command "COLOR" 1)      ;; màu nền dưới 1 là đỏ
 (command "POLYGON" "6" o "I" c1)  ;; vẽ bát giác nền1
;;; subroutine2 vẽ nền 2 ----------------------------------------------------
 (command "ELEV" h1 h1)   ;; cho z=h1 và nâng lên cao khoảng h1
 (command "COLOR" 5)      ;; màu nền dưới 2
 (command "POLYGON" "6" o "I" c2)   ;; vẽ bát giác nền2
;;; subroutine3 vẽ chân đế trụ1 của cột -------------------------------------
 (command "ELEV" (* 2 h1) (/ h1 2))   ;; cho z=2h1 và nâng lên cao  h1/2
 (command "COLOR" 3)
 (command "CIRCLE" o1 r1)
;;; subroutine4 vẽ chân đế vuông1 của cột -----------------------------------
 (command "ELEV" (+ (* 2 h1) (/ h1 2)) (/ h1 2))
 (command "COLOR" 4)
 (command "POLYGON" "4" o1 "I" r1)
;;; subroutine5 vẽ chân đế trụ2 của cột -------------------------------------
 (command "ELEV" (* 3 h1) (/ h1 2))
 (command "COLOR" 5)
 (command "CIRCLE" o1 r2)
;;; subroutine6 vẽ chân đế vuông2 của cột -----------------------------------
 (command "ELEV" (+ (* 3 h1) (/ h1 2)) (/ h1 2))
 (command "COLOR" 6)
 (command "POLYGON" "4" o1 "I" r2)
;;; subroutine7 vẽ chân đế trụ3 của cột -------------------------------------
(command "ELEV" (* 4 h1) (/ h1 2))
 (command "COLOR" 7)
 (command "CIRCLE" o1 r3)
;;; subroutine8 vẽ chân đế vuông3 của cột -----------------------------------
 (command "ELEV" (* 4.5 h1) (/ h1 2))
 (command "COLOR" 8)
 (command "POLYGON" "4" o1 "I" r3)
;;; subroutine9 vẽ cột ------------------------------------------------------
 (command "ELEV" (* 5 h1) (/ c1 3))
 (command "COLOR" 9)
 (command "CIRCLE" o1 r4)
;;; subroutine10 vẽ đầu đỡ vuông1 phía trên cột -----------------------------
 (command "ELEV" (+ (* 5 h1) (/ c1 2)) (/ h1 2))
 (command "COLOR" 10)
 (command "POLYGON" "4" o1 "C" r4)
;;; subroutine11 vẽ đầu đỡ vuông2 phía trên cột -----------------------------
 (command "ELEV" (+ (* 5.5 h1) (/ c1 2)) (/ h1 2))
 (command "COLOR" 11)
 (command "POLYGON" "4" o1 "C" r3)
;;; subroutine12 Copy thành 6 cột -------------------------------------------
 (command "ARRAY" "ALL" "" "P" o "6" "" "")
```



Hình 30.11:
Mặt bằng bản vẽ quán trà.

```
;;; subroutine13 vẽ đa giác mái1 ----------------------------------------------------
 (command "ELEV" (+ (* 6 h1) (/ c1 2)) h1)
 (command "COLOR" 5)
 (command "POLYGON" "6" o "I" c3)
;;; subroutine14 vẽ đa giác mái2 ----------------------------------------------------
 (command "ELEV" (+ (* 7 h1) (/ c1 2)) h1)
 (command "COLOR" 1)
 (command "POLYGON" "6" o "I" c4)
;;; subroutine15 vẽ đa giác mái3 ----------------------------------------------------
 (command "ELEV" (+ (* 8 h1) (/ c1 2)) (* 2 h1))
 (command "COLOR" 5)
 (command "POLYGON" "6" o "I" c5)
;;; subroutine16 xem ảnh 3D ----------------------------------------------------
 (command "DVIEW" "ALL" "" "CA" "12" "12" "X")
 (command "SHADE")
) ;;; hết ----------------------------------------------------------------------
```

Sau khi chạy chương trình với: cạnh đa giác nền1 là C1=100, chiều cao nền1=12, ta có bản vẽ quán trà như hình 30.12 dưới đây.



Hình 30.12: Mô hình quán trà sau khi chạy chương trình.

*** Ví dụ 7:**

```
;;; Vẽ cầu thang xoắn.
;;; chương trình con 1:  đổi độ thành radian ------------------------------------
(Defun DO-RA (x)
  (* pi (/ x 180))
)
;;; chương trình con 2:   hàm gv  và gán giá trị cho biến-------------------------
(defun  gv ( )
 (setq rp (fix (/ h  hs)))
 (setq a1 (polar  a (/ (DO-RA  ia)  2)  r))
 (setq a2 (polar a1 (/ (DO-RA  ia)  2)  1))
 (setq a3 (polar a (- (/ (DO-RA  ia)  2))  r))
 (setq a4 (polar a3 (- (/ (DO-RA  ia)  2))  1))
 ;;;
 (setq tl (+ r  (- l  0.04)))
 (setq t1 (polar a  0  (- tl  0.04)))
 (setq t2 (list (car t1)  hb))
 (setq t3 (polar a  (DO-RA  ia) (- tl  0.04)))
 (setq t4 (list (car t3) (cadr t3) (+ hs  hb)))
 ;;;
 (setq t11 (polar t1  0  0.03))
```

```
 (setq t12 (polar t11 (+ (DO-RA ia) (DO-RA 90)) 0.03))
 (setq t13 (polar t1 (+ (DO-RA ia) (DO-RA 90)) 0.03))
 (setq t21 (list (car t11) (cadr t11) hb))
 (setq t31 (polar t3 (DO-RA ia) 0.04))
 (setq t41 (list (car t31) (cadr t31) (+ hb hs)))
 ;;;
 (command "ELEV" 0 h)
 (command "CIRCLE" a r)
 ;;;
 (command "ELEV" 0 th)
 (command "PLINE" a1 a2 a4 a3 "")
 ;;;
 (command "ELEV" 0 hb)
 (command "PLINE" t1 t11 t12 t13 "C")
 (command "ELEV" hb 0.02)
 (command "3DLINE" t2 t4 "")
 (command "3DLINE" t21 t41 "")
)
;;; chương trình con 3: dựng Block chân thang ----------
(defun upblock ( )
 (setq ang ( ))
 (command "-BLOCK" "s1" a a1 t4 t21 "")
)
;;; chương trình chính như sau: ----------
(defun C:DO ( )
 (setq a (getpoint "Point to be showed on:")) ;; điểm sẽ vẽ trên màn hình
 (setq r (getreal "Radius of col:")) ;; bán kính trục cầu thang xoắn
 (setq h (getreal "Height of floor:")) ;; chiều cao trần nhà
 (setq hb (getreal "Height of bar steel:")) ;; chiều cao tay vịn cầu thang
 (setq l (getreal "Large-step number:")) ;; chiều rộng bậc thang
 (setq th (getreal "Thicknees-step number:")) ;; độ dày bậc thang
 (setq ia (getreal "Include angle of step:")) ;; góc xoay khi dựng cầu thang
 (setq hs (getreal "Height of step:")) ;; khoảng cách giữa các bậc
 (gv)
 (upblock)
 ;;;
 (setq hh ( ))
 (while (< hh h)
  (setq c (list (car a) (cadr a) hh))
  (command "INSERT" "s" c "" "" ang)
  (setq hh (+ hb hs))
  (setq ang (+ ang ia))
 ) ;; hết while
) ;; hết defun ----------
```

=== * ===

PHỤ LỤC
BẢNG TRA CỨU CÁC LỆNH AUTOCAD

=== * ===