TRƯỜNG ĐẠI HỌC NÔNG NGHIỆP I - HÀ NỘI
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

TS. DƯƠNG XUÂN THÀNH

Giáo trình
LẬP TRÌNH NÂNG CAO

(Trên ngôn ngữ Pascal)

(Soạn theo chương trình đã được Bộ GD&ĐT phê chuẩn)

Hà nội, 2005

Lời mở đầu

Cuốn giáo trình này được biên soạn theo đúng đề cương chi tiết môn học đã được Bộ Giáo dục và Đào tạo phê chuẩn. Thời gian học môn học này là 60 tiết trong đó có 10 tiết thực hành trên máy. Tác giả là người đã trực tiếp giảng dạy lập trình Pascal trong nhiều năm cho sinh viên chuyên tin và sinh viên các ngành khác.

Đối tượng sử dụng giáo trình là sinh viên chuyên ngành Tin học hệ đại học chính quy, tuy nhiên giáo trình cũng có thể sử dụng như là một tài liệu tham khảo cho sinh viên chuyên Tin hệ cao đẳng và những người muốn nghiên cứu nâng cao về lập trình.

Mục đích biên soạn cuốn giáo trình là cung cấp cho người đọc một tài liệu đơn giản, cô đọng những kiến thức về lập trình nâng cao. Người đọc có thể tự học mà không nhất thiết phải có thày hướng dẫn.

Giáo trình bao gồm 6 chương và 4 phụ lục.

Chương 1: Chương trình con - Thủ tục và hàm, sinh viên đã được học qua trong chương trình Tin học đại cương, do vậy ở đây chủ yếu đi sâu vào khái niệm tham số, cách thức mà hệ thống dành bộ nhớ cho việc lưu trữ các tham số và việc gọi chương trình con từ chương trình con khác.

Chương 2: Các kiểu dữ liệu có cấu trúc, tập trung vào các kiểu dữ liệu mà sinh viên chưa được học như bản ghi có cấu trúc thay đổi, tập hợp..

Chương 3: Đơn vị chương trình và thư viện chuẩn, là chương chưa được học ở Tin học đại cương , ở đây hướng dẫn cách thiết kế các Đơn vị chương trình (Unit), cách thức sử dụng các Unit và tạo lập thư viện chương trình .

Chương 4: Con trỏ và cấu trúc động, là một chương khó, vì nó vừa liên quan đến quản lý bộ nhớ, vừa liên quan đến kiến thức của môn học Cấu trúc dữ liệu và Giải thuật do vậy trong chương này đã trình bày nhiều ví dụ để người đọc tham khảo.

Chương 5: Giải thuật đệ quy, được trình bày "hơi dài dòng" do đặc thù của tính đệ quy. Bài toán Tháp Hanoi được mô tả khác hoàn toàn so với tất cả các sách về Pascal đã có.

Chương 6: Đồ hoạ, ngoài việc giới thiệu các thủ tục vẽ thông thường, còn dành một phần trọng tâm cho việc xử lý ảnh Bitmap. Trong chương này có sử dụng một vài ví dụ của các tác giả khác (xem phần tài liệu tham khảo) nhưng đã được cải tiến đi rất nhiều.

Phụ lục 1: Bảng mã ASCII

Phụ lục 2: Tóm tắt các thủ tục và hàm của Turbo Pascal 7.0

Phụ lục 3: Định hướng biên dịch

Phụ lục 4: Thông báo lỗi

Các phụ lục đưa ra nhằm giúp người lập trình tiện tra cứu các thủ tục, hàm và xử lý các lỗi khi Pascal thông báo lỗi trên màn hình

Do phải bám sát đề cương và sự hạn chế về số trang tác giả nên trong giáo trình chưa đưa vào được phần xử lý âm thanh, lập trình hướng đối tượng....

Việc biên soạn lần đầu không thể tránh được thiếu sót, tác giả mong nhận được sự góp ý của bạn đọc và đồng nghiệp để lần xuất bản sau sẽ tốt hơn. Mọi góp ý xin gửi về địa chỉ:

Bộ môn Công nghệ Phần mềm, Khoa Công nghệ Thông tin,

Đại học Nông nghiệp I , Trâu quỳ, Gia lâm, Hà nội.

Xin trân trọng cảm ơn.

Hà nội, tháng 5 năm 2005

Ts. Dương Xuân Thành

Chương I

Chương trình con - Thủ tục và hàm

Khái niệm chương trình con đã được trình bày trong môn học Tin học đại cương, do vậy trong chương này chúng ta nhắc lại sơ qua một số khái niệm cũ và dành thời gian cho việc tìm hiểu sâu về tham số (tham biến và tham trị), lời gọi chương trình con, cách thức bố trí chương trình con trong thân chương trình mẹ. Sau khi học chương này bạn đọc cần nắm được các nội dung chủ yếu sau:

- Thế nào là biến toàn cục, biến địa phương
- Các biến toàn cục và biến địa phương được bố trí ở đâu
- Tầm tác dụng của từng loại biến
- Thứ tự xây dựng các chương trình con có ảnh hưởng thế nào đến toàn bộ chương trình
- Thế nào là tính đệ quy của chương trình con
- Lời gọi chương trình con thế nào là được phép
- Cách khai báo trước để gọi chương trình con không theo thứ tự thiết kế

1. Khái niệm về chương trình con

Chương trình con trong Pascal được hiểu là một chương trình nằm trong lòng một chương trình khác. Chương trình con gồm hai loại: **Thủ tục (Procedure)** và **hàm (Function)**. Các chương trình con được dùng rộng rãi khi xây dựng các chương trình lớn nhằm làm cho chương trình dễ theo dõi, dễ sửa chữa. Một đặc điểm nổi bật của chương trình con là nó có tính đệ quy nhờ thế mà nhiều bài toán sẽ được giải quyết dễ dàng.

Khi một chương trình con được gọi thì các biến được khai báo trong chương trình con (ta gọi là biến cục bộ) sẽ được cấp phát bộ nhớ. Kết thúc chương trình con, các biến cục bộ được giải phóng, điều này sẽ được lặp lại mỗi khi chương trình con được gọi và nó đồng nghĩa với việc thời gian xử lý bài toán sẽ tăng lên.

Bản thân tên gọi của hai loại chương trình con đã nói lên phần nào sự khác nhau giữa chúng. Function (Hàm) là một loại chương trình con cho kết quả là một giá trị vô hướng. Khi gọi tên Function với các tham số hợp lệ ta sẽ nhận được các giá trị, bởi vậy tên hàm có thể đưa vào các biểu thức tính toán như là các toán hạng. Procedure là loại chương trình con khi thực hiện không cho ra kết quả là một giá trị, mỗi Procedure nhằm thực hiện một nhóm công việc nào đó của chương trình mẹ, vì vậy tên của Procedure không thể đưa vào các biểu thức tính toán. Bằng cách xây dựng các chương trình con người lập trình có thể phân mảnh chương trình cho nhiều người cùng làm dưới sự chỉ đạo thống nhất của người chủ trì. Trong Turbo Pascal đã có sẵn một số chương trình con, ví dụ: sin(x), sqrt(x).... là các Function, còn read(), write(), gotoxy (x1,x2)..... là các Procedure.

Trong một chương trình các chương trình con được bố trí ngay sau phần khai báo biến. Cấu trúc tổng quát một chương trình Pascal như sau:

```
PROGRAM tên_chương_trình;
USES tên các UNIT;  (*khai báo các đơn vị chương trình cần thiết*)
LABEL                (*khai báo nhãn*).
CONST                (*Khai báo hằng*)
TYPE                 (*định nghĩa kiểu dữ liệu mới*)
VAR                  (*khai báo biến*)

PROCEDURE Tên_CTC1 (danh sách tham số hình thức);
 Begin
.....                (*thân thủ tục thứ nhất*).
End;

PROCEDURE Tên_CTC2 (danh sách tham số hình thức);
 Begin
.....                (*thân thủ tục thứ hai*)
 End;
```

```
FUNCTION Tên_HAM1(danh sách tham số hình thức):kiểu hàm;
Begin
.....                    (*thân hàm thứ nhất*)
End;
.....

BEGIN                    (*bắt đầu chương trình mẹ*)
.....
END.
```

Ghi chú:

1. Các chương trình con về nguyên tắc cũng bao gồm các phần khai báo báo như đối với một chương trình mẹ, phần nào không cần thiết thì không khai. Điều khác nhau cơ bản là thân chương trình con nằm giữa hai từ khoá Begin và End; (sau End là dấu ";" chứ không phải là dấu "." như trong chương trình mẹ) ngoài ra chương trình con còn có thể thêm phần khai báo các tham số hình thức, các tham số hình thức được đặt trong dấu () và viết ngay sau tên chương trình con.

2. Nếu chương trình con là Function thì cuối chương trình cần có lệnh gán giá trị vào tên chương trình con.

2. Tham số trong chương trình con

Các chương trình con có thể không cần tham số mà chỉ có các biến riêng (biến cục bộ). Trong trường hợp cần nhận các giá trị mà chương trình mẹ truyền cho thì chương trình con cần phải có các tham số (**Parameter**). Tham số được khai báo ngay sau tên chương trình con và được gọi là ***tham số hình thức***.

Những giá trị lưu trữ trong các biến toàn cục của chương trình mẹ, nếu được truyền cho các thủ tục hoặc hàm thông qua lời gọi tên chúng thì được gọi là ***Tham số thực.***

Tham số hình thức bao gồm hai loại:

2.1 Tham biến (*Variabic parameter)*

Tham biến là những giá trị mà chương trình con nhận từ chương trình mẹ, các giá trị này có thể biến đổi trong chương trình con và khi chương trình con kết thúc các giá trị này sẽ được trả về cho tham số thực.

Cách khai báo tham biến:

Tên chương trình con (Var tên tham biến : kiểu dữ liệu);

2.2 Tham trị (*Value parameter*)

Những tham số truyền vào cho chương trình con xử lý nhưng khi quay về chương trình mẹ vẫn phải giữ nguyên giá trị ban đầu thì được gọi là **tham trị.**

Cách khai báo tham trị:

Tên chương trình con (tên tham trị : kiểu dữ liệu);

Dưới đây là một ví dụ khai báo tham số:

PROCEDURE VIDU(x,y,z: integer; lam:boolean; var qq: char);

Câu lệnh khai báo chương trình con trên đây đồng thời khai báo các tham số hình thức trong đó *x, y,z, lam* là các tham trị, với x, y,z có kiểu integer, lam có kiểu boolean, qq là tham biến vì nó được viết sau từ khoá VAR.

Ví dụ 1.1: Lập chương trình tìm số lớn nhất trong n số nguyên được nhập từ bàn phím.

```
Program Tim_cuc_dai;
Uses Crt;
TYPE dayso = array[1..100] of integer;     (* Định nghĩa kiểu dữ liệu  dayso là kiểu mảng
                                              gồm nhiều nhất là 100 phần tử*).
VAR     a: dayso                           (*khai báo biến của chương trình mẹ*)
        n: integer;
PROCEDURE nhapso(m:integer; var x:dayso);
                                   (* Nhập dãy số  cần tìm cực đại vào  mảng một chiều x[i]*)
      Var i : integer;             (*khai báo biến cục bộ của chương trình con*)
Begin
  writeln('Nhap day so kieu integer);
  For i:=1 to m Do                 (* m được truyền từ chương trình mẹ qua tham số thực n*)
Begin
  write('a[', i , '] = '); realln (x[i]);
  End; End;
FUNCTION Max(m: integer; b:dayso); integer;
 (* Hàm MAX dùng để tìm số lớn nhất trong  dãy số đã nhập, kiểu giá  trị của hàm là kiểu integer *)
VAR
  i,t: integer;                                (* Biến riêng của hàm Max *)
Begin
  t:=b[1];                                     (* Gán phần thứ nhất của mảng b[i] cho biến t *)
  For i:=2 to m Do
  if t<b [i] then t:=b[i];
  Max:=t;                                      (* Gán giá trị cho chính hàm Max*)
End;
BEGIN                              (* Thân chương trình mẹ *)
  Write('Ban can nhap bao nhieu so ?   '); Readln(n);
  NHAPSO(N, A);          (* Gọi chương trình con NHAPSO với 2 tham số thực là n và a. Hai tham
                            số này sẽ thay thế cho hai tham số hình thức m, x trong chương trình con *)
  Writeln (' So lon nhat trong day so da nhap = ', MAX(n,a):5);
                         (* Viết ra giá trị của hàm MAX với 2 tham số thực n,a độ dài số là 5 ký tự *)
  Repeat until keypressed;
END.
```

Ví dụ1.1 là một chương trình bao gồm hai chương trình con, chương trình con thứ nhất là một thủ tục (Procedure), chương trình con thứ hai là một hàm (Function).

Chương trình mẹ có lệnh đọc số phần tử n của mảng dayso (tức là số lượng con số sẽ nhập vào). Vì mảng Dayso được khai báo có 100 phần tử nên không thể đọc vào nhiều quá 100 con số.

Sau đó là lệnh gọi chương trình con NHAPSO với 2 tham số thực là n, a, ở đây a là tham biến nghĩa là giá trị của mảng a sẽ được thay đổi trong chương trình con bởi tham số hình thức x[i]. Chương trình con nhập vào tham biến x[i] kiểu mảng n con số thông qua tham trị m (m=n).

Lệnh viết giá trị lớn nhất của dãy số có kèm lời gọi hàm MAX vì hàm MAX thực chất trong trường hợp này chỉ là một con số.

Hàm MAX dùng để tìm số lớn nhất trong các số đã nhập, lời gọi hàm trong chương trình mẹ kèm theo việc truyền hai tham số thực là n và a thay thế cho hai tham số hình thức là m và b. Tên hàm được dùng như là một biến trong bản thân hàm khi ta dùng phép gán giá trị MAX:=t;

Chú ý:

*1. Kiểu dữ liệu trong khai báo tham số hình thức chỉ có thể là: **số nguyên, số thực, ký tự, hoặc Boolean**. Nếu muốn đưa các kiểu dữ liệu có cấu trúc vào trong khai báo tham số thì phải định nghĩa trước kiểu dữ liệu này ở phần khai báo kiểu sau từ khoá Type (xem ví dụ 1.1).*

2. Với kiểu dữ liệu chuỗi, nếu chúng ta khai báo tham số thực trong chương trình mẹ và tham biến trong chương trình con đều là STRING (không quy định độ dài tối đa của chuỗi) thì không cần phải định nghĩa trước kiểu dữ liệu ở phần TYPE. Để thấy rõ vấn đề chúng ta xét ví dụ sau đây:

Ví dụ: 1.2

```
Program Chuong_trinh_me;
Var s:string; m:byte
Procedure  Chuong_trinh_con( Var a:string; n:byte);
```

Cách khai báo trên là được phép trong Pascal .

Nếu chúng ta quy định độ dài chuỗi như một trong ba dạng sau thì sẽ bị báo lỗi:

Dạng thứ nhất

```
Program Chuong_trinh_me;
Var s:string[30]; m:byte
Procedure  Chuong_trinh_con( Var a:string[30]; n:byte);
```

Dạng thứ hai

```
Program Chuong_trinh_me;
Var s:string[30]; m:byte
Procedure  Chuong_trinh_con( Var a:string; n:byte);
```

Dạng thứ ba

```
Program Chuong_trinh_me;
Var s:string; m:byte
Procedure  Chuong_trinh_con( Var a:string[30]; n:byte);
```

Tuy nhiên có một ngoại lệ khi tham số hình thức trong các chương trình con không phải là tham biến mà là tham trị thì có thể khai báo theo dạng thứ hai.

Muốn quy định độ dài chuỗi trong các khai báo tham biến thì phải khai báo kiểu dữ liệu theo mẫu sau:

```
Program Chuong_trinh_me;
Type S1 = string[30];
Var s:s1; m:byte
Procedure  Chuong_trinh_con( Var a:s1; n:byte);
```

3. Truyền tham số cho chương trình con

Trở lại ví dụ 1.1 ta thấy trong mỗi chương trình con có những tham số riêng của mình. Chương trình con nhập số đã sử dụng hai tham số hình thức là m và x. Hai tham số này được chuẩn bị để nhận các giá trị mà chương trình mẹ truyền cho thông qua lời gọi chương trình con với các tham số thực là n và b. Vì m được khai báo kiểu không có từ khoá Var nên nó là tham trị, nghĩa là khi chương trình con kết thúc thì giá trị của tham số thực n vẫn không thay đổi, tham số x là tham biến vì nó được khai báo sau từ khoá Var.

Khi tham số hình thức trong chương trình con là tham biến thì tham số thực trong chương trình mẹ phải là biến chứ không thể là hằng. Trong mọi trường hợp cả hai tham số thực và tham số hình thức đều phải cùng kiểu dữ liệu.

Các tham số thực truyền cho tham biến thì giá trị của nó có thể thay đổi trong chương trình con, khi ra khỏi chương trình con nó vẫn giữ nguyên các giá trị đã thay đổi đó. Trong ví dụ 1.1 tham số thực a là một mảng của n phần tử và tất cả các phần tử đều còn rỗng, khi truyền a vào tham biến x thì ở thời điểm ban đầu các phần tử của x cũng rỗng. Phép gán trong chương trình con NHAPSO sẽ làm thay đổi giá trị các phần tử của x, sau khỏi ra chương trình con nó giữ nguyên các giá trị đã gán tức là các giá trị ta nhập từ bàn phím vào các phần tử của mảng.

Khi tham số hình thức là tham trị thì tham số thực phải là một giá trị. Chương trình con nhận giá trị này như là giá trị ban đầu và có thể thực hiện các phép tính làm biến đổi giá trị đó, quá trình này chỉ tác động trong nội bộ chương trình con, khi ra khỏi chương trình con giá trị của tham số thực không biến đổi. Cụ thể trong ví dụ trên biến n nhận giá trị đọc từ bàn phím trong chương trình mẹ và được truyền cho tham số m trong cả hai chương trình con. Sau lời gọi chương trình con NHAPSO giá trị này có thể bị thay đổi nhưng khi rời NHAPSO đến lời gọi hàm MAX thì n lại vẫn giữ giá trị ban đầu.

Như vậy nếu muốn bảo vệ giá trị một tham số nào đó khi truyền chúng cho chương trình con thì phải qui định chúng là tham trị. Còn nếu muốn nhận lại giá trị mà chương trình con đã sinh ra thay thế cho những giá trị ban đầu có trong chương trình mẹ (hoặc là dùng những giá trị của chương trình con thay thế cho biến chưa được gán giá trị trong chương trình mẹ như vi dụ 1.1) thì tham số phải là tham biến.

Để thấy rõ hơn ý nghĩa của các tham số chúng ta xét ví dụ sau đây:

Người mẹ trao cho con trai một chiếc nhẫn và một túi tiền. Trước khi con đi làm ăn ở phương xa mẹ dặn: "Chiếc nhẫn là tín vật dùng để nhận lại gia đình và họ hàng khi con trở về, còn túi tiền là vốn ban đầu cho con kinh doanh".

Trong qúa trình làm ăn, người con có thể cầm cố chiếc nhẫn nhưng khi trở về nhà nhất thiết phải mang chiếc nhẫn đó về, còn túi tiền khi quay về có thể nhiều lên cũng có thể ít đi, thậm chí không còn đồng nào. Trong ví dụ này chiếc nhẫn đóng vai trò tham trị, còn túi tiền đóng vai trò tham biến.

Vấn đề đặt ra là Pascal làm thế nào để đảm bảo các tính chất của tham trị và tham biến. Điều này sẽ được làm rõ khi nghiên cứu việc bố trí bộ nhớ (mục 5).

Khi lựa chọn tham số cần lưu ý một số điểm sau:

a. Kiểu của tham số trong chương trình con phải là các ***kiểu vô hướng đơn giản*** đã được định nghĩa sẵn trong Pasacl hoặc đã được định nghĩa trong phần đầu của chương trình mẹ. Trong chương trình con không thể định nghĩa kiểu dữ liệu mới.

b. Chương trình con có thực sự cần tham số hay không? Nếu chương trình con chỉ sử dụng các biến toàn cục và biến địa phương cũng đáp ứng được yêu cầu của bài toán thì không nên dùng tham số. Nếu chương trình con thực hiện nhiều công việc trên cùng một loại đối tượng (đối tượng ở đây có thể là hằng, biến, hàm, thủ tục, kiểu), nghĩa là lời gọi chương trình con được lặp lại nhiều lần trên cùng một hoặc một nhóm đối tượng thì cần dùng đến tham số.

c. Nếu không muốn thay đổi giá trị của các tham số thực trong chương trình mẹ khi truyền nó cho chương trình con thì phải dùng tham số hình thức dưới dạng tham trị (trong phần khai báo kiểu không có từ khoá Var). Nếu cần thay đổi giá trị của tham số thực trong chương trình mẹ và nhận lại giá trị mà chương trình con đã xử lý thì tham số trong chương trình con phải là tham biến (tên tham số phải đặt sau từ khoá Var).

4. Biến toàn cục và biến địa phương

4.1 Biến toàn cục

Biến khai báo ở đầu chương trình mẹ được gọi là biến toàn cục. Nó có tác dụng trong toàn bộ chương trình, kể cả các chương trình con. Khi thực hiện chương trình máy dành các ô nhớ ở vùng dữ liệu (Data) để lưu giữ giá trị của biến.

Mở rộng ra tất cả các **đối tượng** trong Pascal (Kiểu dữ liệu, Hằng, Biến, Hàm, Thủ tục) khai báo trong chương trình mẹ được gọi là đối tượng toàn cục. Như vậy một kiểu dữ liệu đã được định nghĩa trong chương trình mẹ thì đương nhiên được phép sử dụng trong các chương trình con của nó.

Trong ví dụ 1.1 biến a và n là biến toàn cục, hai biến này có thể sử dụng trực tiếp trong thủ tục NHAPSO và trong hàm MAX mà không cần khai báo lại.

Pascal dành các ô nhớ ở vùng Data (vùng dữ liệu) cho các đối tượng toàn cục.

4.2 Biến địa phương

Những đối tượng khai báo trong chương trình con chỉ có tác dụng trong nội bộ chương trình con đó, chúng được gọi là đối tượng địa phương. Đối tượng hay được sử dụng nhất là Biến.

Biến địa phương có thể trùng tên với biến toàn cục song hệ thống dành các ô nhớ trong vùng nhớ ngăn xếp (Stack) để lưu giữ các giá trị của biến. Kết thúc chương trình con hệ thống sẽ giải phóng ô nhớ của biến địa phương dùng vào việc khác, giá trị của các biến lưu trữ trong các ô nhớ này sẽ không còn.

Trường hợp trong chương trình con lại có các chương trình con khác thì biến địa phương của chương trình con cấp trên lại được xem là biến toàn cục đối với chương trình con cấp dưới.

Một biến sau khi được khai báo trong một chương trình sẽ chỉ có tầm tác dụng trong bản thân chương trình đó và các chương trình con của nó. Biến này không có tác dụng trong các chương trình cùng cấp khác hoặc trong các chương trình con của chương trình khác. Điều này có nghĩa là các chương trình con và chương trình mẹ có thể có nhiều biến trùng tên, nhưng tầm tác dụng thì khác nhau do đó tính toàn vẹn của dữ liệu luôn được bảo đảm.

Ví dụ 1.3

```
Program Chuong_trinh_con;
Uses crt;
Var i,n:byte;   c1:string[30];
Procedure Bien_dia_phuong;
Var i,n:byte;   c1:string[30];
      Begin
      n:=3;
     C1:='Thu do Ha noi';
     Writeln('Gia tri n trong chuong trinh con: ',n);
      Writeln('Chuoi C1 trong chuong trinh con: ',C1);
     end;
  Begin                  (* thân chương trình mẹ *)
      Clrscr;
      Bien_dia_phuong;
      Writeln;
      n:=0;
      for i:= 1 to 10 do n:= n+i;
      c1:='Happy Birth Day';
      Writeln('Gia tri n trong chuong trinh me: ',n);
      Writeln('Chuoi C1 trong chuong trinh me: ',C1);
      Readln;
  End.
```

Ví dụ 1.3 thiết kế một chương trình mẹ và một chương trình con dưới dạng thủ tục. Phần khai báo biến trong cả hai là như nhau. Phép gán dữ liệu vào biến n và chuỗi C1 là khác nhau.

Sau lời gọi chương trình con Bien_dia_phuong màn hình xuất hiện:

Gia tri n trong chuong trinh con: 3

Chuoi C1 trong chuong trinh con: Thu do Ha noi

Tiếp đó là kết quả các lệnh viết trong chương trình mẹ:

Gia tri n trong chuong trinh me: 55

Chuoi C1 trong chuong trinh me: Happy Birth Day

Lời gọi chương trình con được thực hiện trước, kết quả là biến n mang giá trị 3, còn chuỗi C1 là ' Thu do Ha noi '. Khi trở về chương trình mẹ biến n =55 còn chuỗi C1 = ' Happy Birth Day' . Điều này có nghĩa là biến n và C1 trong chương trình con không ảnh hưởng đến biến n và C1 trong chương trình mẹ.

5. Cách thức bố trí bộ nhớ

Khi một chương trình Pascal dạng EXE được chạy máy sẽ cấp phát một vùng nhớ cơ sở 640 Kb. Vùng nhớ này sẽ bao gồm các ô nhớ nằm liền nhau nghĩa là địa chỉ các ô nhớ tăng liên tục.

Phân loại vùng nhớ	Tên vùng	Dung lượng
Cao	Heap	0 - 655360 Bytes
	Stack Segment	16 - 64 Kb
	Data Segment	64 Kb
	Code Segment	Mỗi đoạn có 64 Kb
Thấp	Program Segment Prefix	256 Bytes

Hình 1.1

Chương trình được bố trí trong bộ nhớ như sau:

* Program Segment Prefix: ghi địa chỉ các hàm, biến, thủ tục

* Code Segment: lưu mã chương trình chính và mã các Unit liên quan đến chương trình, vùng này có thể gồm nhiều đoạn, mỗi đoạn 64 Kb.

* Data Segment: lưu trữ các biến, hằng, kiểu của chương trình chính, vùng này chỉ có 64 Kb nên nếu chương trình chính có quá nhiều hằng, biến thì có thể gặp lỗi: Too many variables

* Stack Segment: Lưu mã chương trình con và biến địa phương

* Heap: vùng nhớ tự do dùng cho việc cấp phát động

Các tham trị và biến cục bộ khai báo trong chương trình con được bố trí vào các ô nhớ của Stack. Khi chương trình mẹ gọi và truyền tham số cho chương trình con thì giá trị của các tham số này sẽ được sao chép vào các ô nhớ đã bố trí ở stack. Mọi biến đổi diễn ra trong stack không ảnh hưởng đến các giá trị của tham số thực trong chương trình mẹ.

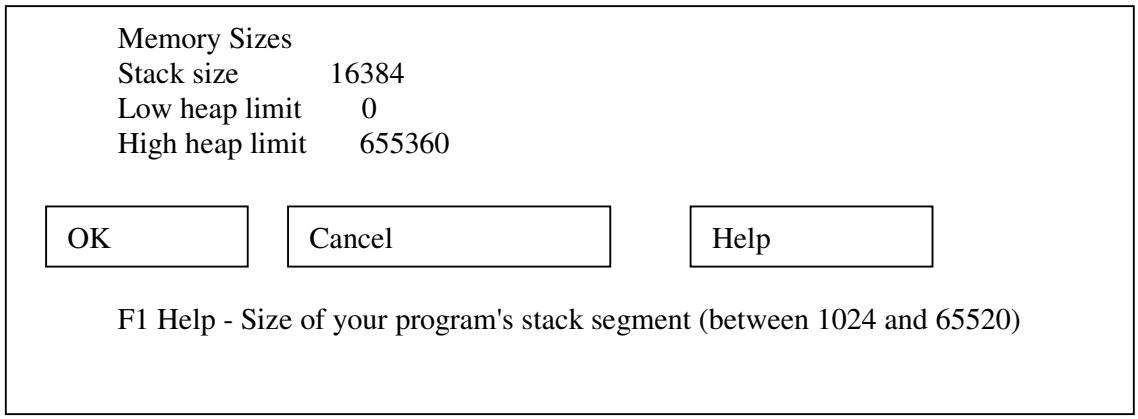
Với các tham biến, Pascal không bố trí ô nhớ riêng mà sử dụng con trỏ trỏ vào địa chỉ của ô nhớ chứa biến toàn cục. Khi chương trình con làm thay đổi giá trị của các tham biến thì

cũng có nghĩa là nó làm thay đổi giá trị của các biến toàn cục trong chương trình mẹ. Kết thúc chương trình con chỉ các biến địa phương là bị giải phóng còn biến toàn cục không bị giải phóng cho nên chúng ta nói chương trình con đã mang các giá trị mới về cho chương trình mẹ.

Cần chú ý rằng Pascal 7.0 chỉ dành 16 Kb cho vùng Stack, dung lượng này đáp ứng đầy đủ các ứng dụng thông thường. Với những ứng dụng sử dụng tính đệ quy mà số lần gọi đệ quy là quá lớn thì sẽ có thể bị lỗi: Stack Overflow (bộ nhớ ngăn xếp bị tràn).

Gặp trường hợp này cần phải mở rộng vùng nhớ Stack bằng cách sau:

Trên thanh thực đơn chọn Options/Memory Size sẽ xuất hiện cửa sổ (hình 1.2)



Hình 1.2

Stack size 16384: dung lượng hiện thời của Stack

Size of your program's stack segment (between 1024 and 65520)

Có thể thay đổi dung lượng Stack trong khoảng 1024 - 65520 Bytes

Muốn thay đổi dung lượng của Stack chúng ta chỉ việc gõ dung lượng mới thay vào vị trí 16384 hiện thời.

Các tham số:

Low heap limit 0

High heap limit 655360 Là vùng nhớ tự do dành cho cấp phát động, không nên nhầm chúng với giá trị tối thiểu và tối đa của Stack.

6. Tính đệ qui của chương trình con

Thông thường lời gọi một chương trình con chỉ được thực hiện khi chương trình con đó đã được thiết kế hoàn chỉnh. Tuy nhiên Pascal cho phép một chương trình con ngay trong quá trình xây dựng lại có thể gọi tới chính nó, không những thế từ một chương trình con còn có thể gọi tới các chương trình con khác cùng cấp hoặc chương trình con cấp cao hơn nó.

Một chương trình con có thể có lời gọi tới chính tên chương trình con đó, tính chất này được gọi là tính "Đệ qui của chương trình con". Đệ quy có thể sử dụng trong cả Procedure và Function. Giống như mảng trong Pascal tương đương với ma trận trong toán, đệ quy trong Pascal tương đương với tính Quy nạp trong toán học. Về điều này chúng ta sẽ đề cập đến trong chương V "Giải thuật Đệ quy".

7. Lời gọi chương trình con

Một chương trình mẹ có thể có nhiều chương trình con trực thuộc, bên trong mỗi chương trình con lại có thể có các chương trình con riêng. Nói cách khác trong Pascal tồn tại một lớp chương trình con ngang cấp nhau, mỗi chương trình con này lại có thể đóng vai trò chương trình mẹ của một lớp chương trình con khác.

Khi thiết kế, mỗi chương trình con phải là một khối riêng biệt không thể lồng nhau hoặc có các lệnh nhảy Goto từ chương trình con này tới chương trình con khác.

7.1 Gọi chương trình con từ trong chương trình mẹ

Lời gọi chương trình con có thể đặt bất kỳ chỗ nào trong chương trình mẹ. Nếu chương trình con là một thủ tục thì lời gọi chương trình con (tức là tên chương trình con) có thể tạo nên một câu lệnh, ví dụ:

```
Readln;     Gotoxy(5,8);
```

Nếu chương trình con là hàm thì tên hàm không thể tạo nên một câu lệnh, nói khác đi tên hàm phải nằm trong một biểu thức hay trong một thủ tục nào đó, ví dụ với hàm khai căn bậc hai SQRT() chúng ta không thể viết

```
sqrt(9);
```

Điều này là dễ hiểu vì hàm cho ta giá trị vô hướng, giá trị này không phải là một lệnh do đó Pascal không biết phải làm gì với giá trị đó.

Cách gọi hàm như sau là hợp lệ:

```
a:= sqrt(9) + 5;
Witeln('Can bac hai cua 9 bang ',sqrt(9));
```

7.2 Gọi chương trình con từ chương trình con khác

Các chương trình con cùng cấp có thể gọi tới nhau và truyền tham số cho nhau.

Nguyên tắc gọi là: những chương trình con xây dựng sau có thể gọi tới các chương trình con đã xây dựng trước nó, đồng thời các chương trình con cấp dưới cũng có thể gọi tới các chương trình con cấp trên nếu chúng cùng một gốc. Điều này có nghĩa là các chương trình con xây dựng trước không thể gọi tới các chương trình con xây dựng sau nếu không có chỉ báo FORWARD (xem mục 8). Xét một số ví dụ sau:

Ví dụ 1.6

```
Program Goi_ctc;
Uses crt;
Type dayso=array[1..60] of byte;     s1=string[30];
Var
```

```
  a:s1; b:dayso;    i,j,n:byte;
Procedure nhapso(m:byte; var c:dayso);
 Begin
    For  i:=1 to m do
  Begin
  Write('c[',i,'] = '); readln(c[i]);
  End;
End;
 Function tinhtong(m:byte;  d:dayso):real;
 var tong:real;
 Begin
    tong:=0;
    for i:= 1 to m do Tong:=tong+d[i];
    tinhtong:=tong;
 End;
Procedure viet(k:byte;  e:dayso);
 Begin
    Write('Tong cac phan tu mang = ',tinhtong(k,e):8:0);
    readln;
 End;

 BEGIN
    clrscr;
    write('Nhap so phan tu n '); readln(n);
    nhapso(n,b);
    viet(n,b);
 END.
```

Ví dụ 1.6 thiết kế ba chương trình con là Nhapso, Tinhtong và Viet. Thủ tục Nhapso dùng để nhập các phần tử vào mảng một chiều. Hàm Tinhtong dùng để tính tổng các phần tử mảng và thủ tục Viet dùng để hiện kết quả tính tổng lên màn hình.

Chương trình mẹ gọi chương trình con Viet và truyền các tham số là số phần tử mảng n và giá trị của các phần tử của mảng (mảng b). Chương trình con Viet lại gọi hàm Tinhtong và truyền các tham số cho hàm này. Đây là trường hợp một chương trình con gọi một chương trình con cùng cấp.

Việc các chương trình con gọi tới nhau phải tuân theo quy định sau đây:

Một chương trình con chỉ có thể gọi tới một chương trình con cùng cấp đã thiết kế trước chương trình con hiện thời.

Trong ví dụ 1.6 nếu chúng ta đưa hàm Tinhtong xuống dưới thủ tục Viet thì khi chạy chương trình sẽ bị báo lỗi:

Unknown Indentifier.

Ví dụ 1.7 dưới đây trình bày cách thức mà một chương trình con cấp dưới gọi tới một chương trình con cấp trên cùng gốc.

Ví dụ 1.7

```
Program goi_ctc;

Uses crt;
Type dayso=array[1..60] of byte;
Var     b:dayso;      i,n:byte;
  {Hai chương trình con Nhapso và Tinhtong cùng cấp với thủ tục Xuly}
  Procedure nhapso(m:byte; var c:dayso);
   Begin
     For  i:=1 to m do
       Begin
       Write('c[',i,'] = '); readln(c[i]);
       End;
   End;

   Function tinhtong(m:byte;  d:dayso):real;
     Var tong:real;
     Begin
     tong:=0;
     for i:= 1 to m do Tong:=tong+d[i];
     tinhtong:=tong;
     End;
  Procedure xuly(j:byte; ds:dayso);
    Procedure viet(k:byte;  e:dayso);
     Var i:byte;
     Begin
      Writeln('Tong cac phan tu mang = ',tinhtong(k,e):8:0);
      Writeln('Day so sap xep giam dan ');
      for i:=1 to k do write(e[i],'  ');
      readln;
     End;                          { Kết thúc thủ tục Viet}

  Procedure sapxep(m:byte;  d:dayso);
 Var p,q:byte; Tg:byte;
  Begin
      For p:= 1 to m-1 do
```

```
        For q:=p+1 to m do
        If d[p]<d[q] then
          Begin
          tg:=d[p]; d[p]:=d[q]; d[q]:=tg;
          End;
         viet(m,d);
      End;   { Kết thúc thủ tục sapxep}

    Begin {than thu tuc Xuly}
          Writeln('Thu tuc xu ly dung de sap xep va viet ket qua');
          sapxep(j,ds);
  end;                              { Kết thúc thủ tục Xuly}

  BEGIN   {Than chuong trinh me}
          clrscr;
          write('Nhap so phan tu n '); readln(n);
          nhapso(n,b);
          xuly(n,b);
END.
```

Ví dụ 1.7 có ba chương trình con cùng cấp là Nhapso, Tinhtong và Xuly. Trong thủ tục Xuly có hai chương trình con là Viet và Sapxep trong đó chương trình con Viet được thiết kế trước, Sapxep được thiết kế sau. Chương trình con Sapxep có lời gọi đến chương trình con Viet cùng cấp với nó, mục đích của lời gọi này là truyền cho chương trình con Viet những dữ liệu mảng đã sắp xếp giảm dần. Chương trình con Viet có lời gọi đến hàm Tinhtong là một chương trình con cấp cao hơn nó, vì Tinhtong đã được thiết kế trước Xuly nên lời gọi là hợp lý. Nếu đảo vị trí của hai chương trình con Viet và Sapxep, nghĩa là đưa chương trình con Viet xuống sau Sapxep thì sẽ bị báo lỗi , về điều này chúng ta sẽ nghiên cứu ở mục 8.

8. Khai báo trước bằng Forward

Như đã nêu trong mục 7 việc các chương trình con gọi tới nhau bị hạn chế bởi thứ tự xây dựng các chương trình con đó. Vì những lý do khác nhau người ta không thể thay đổi thứ tự xây dựng các chương trình con nhưng lại muốn các chương trình con phải gọi được tới nhau không phụ thuộc vào thứ tự xây dựng chúng. Để làm việc này Pascal cho phép sử dụng từ khoá Forward. Nghĩa đen của từ Forward là "phía trước" thường được dùng để báo hiệu một cái gì đó ta sẽ gặp sau này ví dụ: phía trước 200 mét là công trường.

Cú pháp:

Tên chương trình con (danh sách tham số); Forward;

Dòng khai báo trên đây phải được đặt trong chương trình trước khi xây dựng tất cả các chương trình con. Khi tên một chương trình con đã được khai báo với cú pháp nêu trên thì việc nó nằm trước hay sau một chương trình con sẽ gọi tới nó là không quan trọng. Số lượng chương trình con khái báo trước với từ khoá Forward là không hạn chế.

Cần lưu ý rằng nếu có nhiều chương trình con cần khai báo trước thì mỗi tên chương trình con phải đi với một từ khoá Forward, không thể ghép nhiều tên chương trình con với cùng một từ Forward.

Ví dụ 1.8

```
Program Tu_khoa_Forward;
uses crt;
Type dayso=array[1..60] of byte;
var
   a:string; b:dayso;
   i,j,n:byte;

  Function c2(m:byte;  d:dayso):real; forward;
  Procedure c4(p:byte; var q:dayso); forward;

 Procedure c1(m:byte; var c:dayso);
  Begin
  For  i:=1 to m do
   Begin
   Write('c[',i,'] = '); readln(c[i]);
   End;
   End;

 Procedure c3(k:byte;  e:dayso);
   Var i:byte;
  Begin
  c4(k,e);
  writeln('Mang sau khi sap xep');
  for i:= 1 to k do  write(e[i],'  ');
  writeln;
  Write('Tong cac phan tu mang = ',c2(k,e):8:0);
  readln;
  End;

   Function c2(m:byte;  d:dayso):real;
  var tong:real;
```

```
 Begin
 tong:=0;
 for i:= 1 to m do Tong:=tong+d[i];
 c2:=tong;
 End;

Procedure c4(p:byte; var q:dayso);
  Var i,j:byte; tg:integer;
  Begin
  for i:= 1 to (p-1) do
  for j:= i+1 to p do
  if q[i]>q[j] then
  Begin
   tg:=q[i]; q[i]:=q[j]; q[j]:=tg;
  End;
 End;

BEGIN
     clrscr;
     write('Nhap so phan tu n '); readln(n);
     c1(n,b);
     c3(n,b);
END.
```

Ví dụ 1.8 có 4 chương trình con trong đó c1 và c3 thiết kế trước còn c2 và c4 thiết kế sau. Trong c3 có lời gọi đến c2 và c4 do vậy phải khai báo trước c2 và c4 . Nếu không muốn khai báo trước thì cần đưa c2 và c4 lên trên c3.

Bài tập ứng dụng chương 1

1. Lập chương trình tính diện tích toàn phần và thể tích các hình : Trụ tròn, nón.

Yêu cầu: Thiết kế menu theo mẫu sau, Menu có thể dùng con trỏ dịch chuyển để chọn các chức năng: **Hinh Tru Hinh Non Ket thuc**

Việc tính toán diện tích, thể tích mỗi hình ứng với một chương trình con

Tất cả các hình đều chung một chương trình con hiện kết quả. Chức năng Ket thuc dùng để quay về cửa sổ Pascal.

2. Lập một chương trình tạo thực đơn với các chức năng:

Tính giai thừa Tính tổ hợp Trở về

Dùng các chương trình con đã lập để giải bài toán sau: Cho n điểm trên màn hình, qua hai điểm bất kỳ bao giờ cũng nối được bởi một đoạn thẳng. Tính xem có bao nhiêu đoạn thẳng được tạo ra. Tìm đoạn ngắn nhất và dài nhất , chúng nối các điểm nào?

3. Thiết kế thực đơn với các chức năng:

1.giai he pt bac nhat 2. giai pt bac hai 3.Ket thuc

Yêu cầu: Bấm số để chọn chức năng trên thực đơn. Chức năng Ket thuc dùng để quay về cửa sổ Pascal.

Chương trình có 2 chương trình con để giải hệ phương trình bậc nhất 2 ẩn và giải phương trình bậc 2

4. A,B là mảng hai chiều của các số thực, số dòng và cột của từng mảng nhập từ bàn phím, lập chương trình bao gồm các chương trình con: nhập dữ liệu vào mảng, kiểm tra xem có thể nhân hai mảng hay không, nếu có thì chạy chương trình con nhân hai mảng, nếu không thì thông báo không thể nhân hai mảng. Hiện kết quả nhân dưới dạng ma trận.

5. Cho hai chuỗi s1, s2, viết chương trình bao gồm các chương trình con:

NHAP dùng để nhập vào s1, s2 các ký tự của bảng mã gồm cả chữ cái và chữ số,

TACH dùng để tách riêng các chữ số và chữ cái, những chữ số tách ra lưu vào mảng một chiều theo thứ tự của s1 trước, s2 sau.

CONG dùng để cộng các chữ số tách ra từ hai chuỗi

Thông báo kết quả theo mẫu:

Chuỗi s1 sau khi tách:.....

Chuỗi s2 sau khi tách:.....

Tổng các chữ số:.....

6. Lập chương trình với 4 chương trình con dùng để chuyển đổi các số giữa 4 hệ đếm:

Hệ 10 sang hệ 2, 8, 16

Hệ 2 sang hệ 8, 10, 16

Hệ 8 sang hệ 2, 10, 16

Hệ 16 sang hệ 2, 8, 10

Chương 2

Các kiểu dữ liệu có cấu trúc

Trong chương này không trình bày chi tiết các kiểu dữ liệu có cấu trúc đơn giản như kiểu mảng, chuỗi. Nội dung trọng tâm của chương là kiểu bản ghi (Record) có cấu trúc thay đổi, kiểu tệp và kiểu tập hợp. Chương này bạn đọc cần nắm được các vấn đề sau:

- Cách thức định nghĩa một kiểu dữ liệu mới
- Khai báo biến với các kiểu dữ liệu do người lập trình tự định nghĩa
- Cách sử dụng toán tử CASE khi khai báo bản ghi có cấu trúc thay đổi
- Cách thức ghi và đọc dữ liệu cho ba loại tệp: tệp văn bản, tệp có kiểu và tệp không kiểu, chú trọng cách ghi dữ liệu kiểu số vào tệp văn bản và lấy số liệu ra để xử lý
- Xử dụng dữ liệu kiểu tập hợp trong lập trình

1. Dữ liệu kiểu bản ghi (record)

1.1 Khái niệm cơ bản

Kiểu bố trí dữ liệu thông dụng nhất mà con người nghĩ ra là bố trí dưới dạng bảng. Bảng được coi là một đối tượng (để quản lý hoặc nghiên cứu), bảng bao gồm một số cột và một số dòng. Số cột, dòng trong bảng phụ thuộc vào phần mềm quản lý mà chúng ta sử dụng. Trong từng cột dữ liệu có tính chất giống nhau. Các phần mềm quản trị dữ liệu như Excel, Foxpro... đều ứng dụng khái niệm bảng và Pascal cũng không phải là ngoại lệ. Để có được một bảng trước hết Pascal xây dựng nên một dòng gọi là "bản ghi", tập hợp nhiều dòng sẽ cho một bảng, mỗi bảng được ghi vào bộ nhớ dưới dạng một tệp.

Bản ghi (Record) *là một cấu trúc bao gồm một số (cố định hoặc thay đổi) các phần tử có kiểu khác nhau nhưng liên quan với nhau*. Các phần tử này gọi là các trường (Field). Ví dụ bảng điểm của lớp học bao gồm các trường Hoten, Ngaysinh, Gioitinh, Lop, Diachi, Toan, Ly, Hoa,...., dữ liệu điền vào các trường hình thành nên một bản ghi (Record). Có thể có những trường trong một bản ghi lại là một bản ghi, ví dụ trường Ngaysinh ở trên có thể là một bản ghi của ba trường là Ngay, Thang, Nam. Bản ghi không phải là kiểu dữ liệu đã có sẵn trong Pascal mà do người sử dụng tự định nghĩa do đó chúng phải được khai báo ở phần TYPE.

Bản ghi bao gồm hai loại:

* Bản ghi có cấu trúc không đổi : là loại bản ghi mà cấu trúc đã được định nghĩa ngay từ khi khai báo và giữ nguyên trong suốt quá trình xử lý.

* Bản ghi có cấu trúc thay đổi: là loại bản ghi mà cấu trúc của nó (tên trường, số trường, kiểu trường) thay đổi tuỳ thuộc vào những điều kiện cụ thể. Loại bản ghi này khi khai báo thì vẫn khai báo đầy đủ song khi xử lý thì số trường có thể giảm đi (so với cấu trúc đã khai báo) chứ không thể tăng lên.

Điểm mạnh của Bản ghi là cho phép xây dựng những cấu trúc dữ liệu đa dạng phục vụ công việc quản lý, tuy vậy muốn lưu trữ dữ liệu để sử dụng nhiều lần thì phải kết hợp kiểu Bản ghi với kiểu Tệp.

1.2 Khai báo

Kiểu dữ liệu của các trường trong Record có thể hoàn toàn khác nhau và được khai báo sau tên trường, những trường có cùng kiểu dữ liệu có thể khai báo cùng trên một dòng phân cách bởi dấu phảy "," . Cuối mỗi khai báo trường phải có dấu ";" .

Kiểu dữ liệu Record được khai báo như sau:

```
 TYPE
<Tên kiểu > = RECORD
        <Tên trường 1>: Kiểu;
        <Tên trường 2>: Kiểu;
        ......
        <Tên trường n>: Kiểu;
END;
```

Ví dụ 2.1

Khai báo kiểu dữ liệu BANGDIEM bao gồm một số trường nhằm phục vụ việc quản lý điểm.

```
TYPE
BANGDIEM = RECORD
        Hoten: String[25];
        Gioitinh: Char;
        Lop: String[5];
        Diachi: String[30];
        Toan,Ly,Hoa: Real;
END;
```

Với khai báo như trên dung lượng bộ nhớ dành cho các trường (tính bằng Byte) sẽ là:

Hoten 26, Gioitinh 1, Lop 6, Diachi 31, Toan 6, Ly 6, Hoa 6. (Các trường kiểu String bao giờ cũng cần thêm 1 Byte chứa ký tự xác định độ dài chuỗi).

Tổng độ dài của Record bằng 26+1+6+31+18=82 Bytes.

Có thể dùng hàm Sizeof(tên kiểu) để xác định độ dài một kiểu dữ liệu, ví dụ:

Write(sizeof(bangdiem)) sẽ nhận được số 82

Ví dụ 2.2

Xây dựng kiểu dữ liệu quản lý hồ sơ công chức. Chúng ta sẽ tạo ra bốn kiểu dữ liệu mới đặt tên là Diadanh, Donvi, Ngay và Lylich.

```
Type
Diadanh = Record
            Tinh, Huyen, Xa, Thon: String[15];
            End;
Donvi = Record
            Truong: String[30];
            Khoa, Bomon: String[20]
            End;

Ngay = Record
            Ng: 1..31;
            Th: 1..12;
            Nam: Integer;
            End;

Lylich = Record
            Mhs: Word;
```

```
Hoten: String[25];
Ngaysinh: Ngay;
Quequan: Diadanh;
Coquan: Donvi;
End;
```

Trong cách khai báo trên trường Ngaysinh thuộc kiểu Ngay, Quequan thuộc kiểu Diadanh, Coquan thuộc kiểu Donvi, nói cách khác ba trường này lại chính là ba Record.

Để khắc phục cách khai báo nhiều kiểu bản ghi như trên có thể sử dụng các bản ghi lồng nhau. Kiểu bản ghi lồng nhau có thể khai báo trực tiếp, nghĩa là không cần khai báo riêng rẽ các bản ghi con.

Ví dụ 2.3

```
Uses crt;
Type
Lylich=record
Mhs:word;
Hoten:string[25];
Ngaysinh:record
        Ng:1..31;
        Th:1..12;
        Nam:Integer;
        End;
Quequan:record
        Tinh,Huyen,xa,thon:string[15];
        End;
Coquan:record
        Truong:string[30];
        Khoa, Bomon:string[20];
        End;
End;
.................
```

Ngoài cách khai báo kiểu rồi mới khai báo biến, Pascal cho phép khai báo trực tiếp biến kiểu bản ghi theo cú pháp sau:

```
Var
Tên biến:Record
    Tên trường 1:kiểu trường;
    Tên trường 2:kiểu trường;
    .................
```

End;

1.3 Truy nhập vào các trường của bản ghi

Sau khi đã khai báo kiểu dữ liệu ta phải khai báo biến, giả sử cần quản lý danh sách cán bộ một trường đại học chúng ta phải khai báo một biến chứa danh sách viết tắt là DS. Khi đó ta phải khai

VAR

DS: Lylich;

Giống như hai kiểu dữ liệu Mảng và Chuỗi, việc xử lý được thực hiện trên các phần tử của mảng hoặc chuỗi. ở đây mặc dù DS là một biến nhưng chúng ta không thể xử lý chính biến đó mà chỉ có thể xử lý các trường của biến DS. Để truy nhập vào trường cần viết:

<Tên biến>.<Tên trường mẹ>.<tên trường con>….

Ví dụ để nhập dữ liệu cho trường Hoten ta viết các lệnh:

Write(' Ho va ten can bo: '); Readln(DS.hoten);

Lệnh Readln(DS.hoten); cho phép ta gán Họ tên cán bộ vào trường Hoten của bản ghi hiện thời.

Để nhập ngày tháng năm sinh chúng ta phải truy nhập vào các trường con

Readln(Ds.Ngay.Ngays);

Readln(Ds.Ngay.Thang);

Readln(Ds.Ngay.Nam);

Lệnh viết dữ liệu ra màn hình cũng có cú pháp giống như lệnh nhập.

Writeln(DS.Hoten);

Writeln(Ds.Ngay.Ngays);

.....

Chú ý:

Khi khai báo biến DS kiểu LYLICH chúng ta có thể nhập dữ liệu vào biến DS nhưng chỉ nhập được một bản ghi nghĩa là chỉ nhập dữ liệu được cho một người. Nếu muốn có một danh sách gồm nhiều người thì phải có nhiều bản ghi, để thực hiện điều này chúng ta có thể xây dựng một mảng các bản ghi. Trình tự các bước như sau:

* Định nghĩa kiểu dữ liệu bản ghi

* Khai báo biến mảng với số phần tử là số người cần quản lý, kiểu phần tử mảng là kiểu Bản ghi đã định nghĩa. (xem ví dụ 2.4)

Với tất cả các trường khi truy nhập ta luôn phải ghi tên biến rồi đến tên trường mẹ, tên trường con, … điều này không chỉ làm mất thời gian mà còn khiến cho chương trình không đẹp, Pascal khắc phục nhược điểm này bằng cách đưa vào lệnh WITH... DO.

1.4 Lệnh WITH...DO

Cú pháp của lệnh:

WITH <Tên biến kiểu RECORD> DO <Các lệnh>

Khi sử dụng lệnh WITH...DO chuỗi lệnh viết sau DO chỉ cần viết tên trường có liên quan mà không cần viết tên biến. Xét ví dụ nhập điểm cho lớp học với giả thiết lớp có nhiều nhất là 40 học sinh.

Ví dụ 2.4:

```
Program Nhapdiem;
Uses CRT;
Type
BANGDIEM = RECORD
    Hoten: String[25];
    Gioitinh: ('T','G');     (* kiểu liệt kê, 'T' = Trai, 'G' = Gái *)
    Lop: String[5];
    Diachi: String[50];
    Toan,Ly,Hoa: Real;
    End;
Var
    DS_LOP: Array[1..40] of  BANGDIEM      (*danh sách lớp là mảng 40 phần tử*)
    i,j: Integer;
    lam: Char;
BEGIN
    clrscr;
    lam:='C';  i:=1;
    Repeat
    With DS_LOP[i] Do
  Begin
    Write(' Ho va ten hoc sinh: '); Readln(Hoten);
    Write(' Trai hay gai T/G: '); Readln(Gioitinh);
    Write(' Thuoc lop: '); Readln(Lop);
    Write(' Cho o thuong tru: '); Readln(Diachi);
    Write(' Diem toan: '); Readln(Toan);
    Write(' Diem ly: '); Readln(Ly);
    Write(' Diem hoa: '); Readln(Hoa);
  End;
    i:=i+1;
    Write(' NHAP TIEP HAY THOI ? C/K '); Readln(lam);
    Until upcase(lam)='K';
    clrscr;
    For j:=1 to i-1 do
    With DS_LOP[j] DO
    Writeln(Hoten:15,' ',Gioitinh:2,' ',Lop:4,' ',Diachi:10,' Toan:', Toan:4:2,' Ly:',Ly:4:2,' Hoa:',Hoa:4:2);
```

```
    Repeat Until Keypressed;
  END.
```

Ví dụ 2.4 sử dụng mảng DS_LOP gồm 40 phần tử, mỗi phần tử là một Record. Mỗi lần nhập xong dữ liệu cho một phần tử lại hỏi "Nhap tiep hay thoi?", như vậy số phần tử cụ thể của mảng tuỳ thuộc vào câu trả lời C hay là K.

Vấn đề đáng quan tâm ở đây là cách sử dụng lệnh With ... Do, ví dụ 2.4 sử dụng biến mảng DS_LOP vì vậy trước tiên phải truy nhập vào phần tử thứ i của mảng DS_LOP (1<=i<=40), với mỗi phần tử chúng ta tiếp tục truy nhập vào các trường của chúng.

Với lệnh

With DS_LOP[i] Do

chúng ta có thể nhập trực tiếp dữ liệu vào các trường Hoten, Gioitinh, ...

```
Write(' Ho va ten hoc sinh: '); Readln(Hoten);
Write(' Trai hay gai T/G: '); Readln(Gioitinh);
.....
```

Để viết dữ liệu ra màn hình chúng ta cũng dùng lệnh With … Do

```
With DS_LOP[j] DO
Writeln(Hoten:15,' ',Gioitinh:2,' ',Lop:4,' ',Diachi:10,' Toan:', Toan:4:2,' Ly:',Ly:4:2,' Hoa:',Hoa:4:2);
```

Đối với các bản ghi lồng nhau như ví dụ 2.3 thì lệnh With.. Do cũng phải lồng nhau, xét ví dụ sau:

Ví dụ 2.5

```
Program Kieu_Record;
Uses crt;
Type Tencb=record
       Mahoso:word;
       Hoten:string[25];
       Ngaysinh: Record
                   ngay:1..31;
                   thang:1..12;
                   nam:integer;
                   End;
             End;
Var
   DS: array[1..50] of tencb;
   i,j:byte; tl:char;
Begin
     Clrscr;
     i:=1;
     Repeat
```

```
  With ds[i] do
 Begin
     Write('Ma ho so: '); Readln(mahoso);
     Write('Ho va ten: '); readln(hoten);
        With ngaysinh do
           Begin
                 Write('Ngay sinh: '); readln(ngay);
                 Write('Thang sinh: '); readln(thang);
                 Write('Nam sinh: '); readln(nam);
           End;
 End;
     Write('Nhap tiep hay thoi? C/K '); readln(tl);
     If upcase(tl)='C' then i:=i+1;
     Until upcase(tl)='K';
     clrscr;
     Writeln('DANH SACH CAN BO CO QUAN');
     For j:= 1 to i do
  Begin
     With ds[j] do
     Begin
     Write(mahoso,' ',hoten,' ');
     With ngaysinh do  Writeln(ngay,' ',thang,' ',nam);
   End;
  End;
  Readln;
END.
```

Trong ví dụ 2.5 để nhập dữ liệu vào bản ghi và viết dữ liệu (tức là danh sách cán bộ cơ quan) ra màn hình, chương trình cần phải sử dụng hai lệnh With .. Do lồng nhau, lệnh thứ nhất với biến DS, còn lệnh thứ hai với biến Ngaysinh. Tuy nhiên nếu có một chút tinh ý thì các khối chương trình nhập và viết dữ liệu ra màn hình có thể thu gọn lại, ví dụ khối viết dữ liệu ra màn hình sẽ như sau:

```
Writeln('DANH SACH CAN BO CO QUAN');
   For j:= 1 to i do
    With ds[j]  do
    Writeln(mahoso,' ',hoten,' ',ngaysinh.ngay,' ',ngaysinh.thang,' ',ngaysinh.nam);
```

1.5 Bản ghi có cấu trúc thay đổi

a. Xây dựng kiểu dữ liệu

Bản ghi có cấu trúc cố định dùng để mô tả một đối tượng mà các cá thể của nó (tức là các xuất hiện của đối tượng) có các thuộc tính (các trường) như nhau. Ví dụ DS là bản ghi mô tả một loại đối tượng là "**học viên** ", mỗi xuất hiện của DS ứng với một học viên cụ thể và tất cả các học viên đều có các thuộc tính như nhau bao gồm (xem ví dụ 2.5)

Mahoso, Hoten, Ngay, Thang, Nam.

Trong thực tế nhiều khi ta gặp những đối tượng mà thuộc tính của chúng lại gồm hai loại:

- Thuộc tính chung cho mọi xuất hiện
- Thuộc tính riêng cho một số xuất hiện đặc biệt

Dưới đây là một số ví dụ minh hoạ:

- Kiểu dữ liệu quản lý vé ngành đường sắt

Trên một tuyến đường sắt có nhiều đoàn tàu chạy trong ngày, có những chuyến tốc hành chỉ dừng lại ở một vài ga dọc đường, có những chuyến tàu thường dừng lại tất cả các ga lẻ. Với tàu tốc hành, hành khách chỉ được mang theo hành lý không quá 20 Kg và sẽ có suất ăn trên tàu. Với tàu thường hành khách phải mua vé hàng hoá nếu có vận chuyển hàng hoá và không có suất ăn trên tàu.

* Thuộc tính chung: tên đoàn tàu (TDT), tuyến đường (TD), giờ đi (GD), loại tàu (LT) (ví dụ: tốc hành - TH, tàu thường - TT)

* Thuộc tính riêng với tàu tốc hành: Số xuất ăn (SXA), số ga lẻ dừng dọc đường (SGD), còn tàu thường có thuộc tính riêng là cước hàng hoá (CHH). Như vậy việc xây dựng các bản ghi dữ liệu sẽ phải chú ý đến các thuộc tính chung cho các loại tàu và các thuộc tính riêng của từng loại (xem ví dụ 2.6).

Ví dụ 2.6:

```
Type QLDS = record
  Ten_doan_tau: string[3];
  Tuyen_duong: string[15];
  Gio_di: real;
  Case Loai_tau : (Toc_hanh, Tau_thuong) of
     Toc_hanh: (So_xuat_an:Word; So_ga_do: Byte);
     Tau_thuong:  (cuoc_hang_hoa:real);
End;
```

Ví dụ 2.6 cho ta một kiểu dữ liệu bản ghi có cấu trúc thay đổi một mức, sự thay đổi ở đây thể hiện qua thuộc tính Loai_tau. Như vậy tổng số trường của mỗi bàn ghi tuỳ thuộc vào đoàn tàu đó thuộc loại gì. Nếu là tàu tốc hành thì mỗi bản ghi có 5 trường, còn tàu thường chỉ có 4 trường. Độ dài của các bản ghi được tính căn cứ vào độ dài của các trường có trong bản ghi đó. Như đã biết độ dài từng trường được tính như sau:

Ten_doan_tau: 4,

Tuyen_duong: 16

Gio_di: 6

So_xuat_an: 2

So_ga_do: 1

Ve_hang_hoa: 6

Bản ghi với tàu tốc hành sẽ là 4+16+6+2+1 = 29 Byte

Bản ghi với tàu thường là: 4+16+6+6 = 32 Byte

- Các bước định nghĩa kiểu bản ghi có cấu trúc thay đổi:

- Để định nghĩa một kiểu bản ghi có cấu trúc thay đổi, chúng ta khai báo các thuộc tính chung trước, tiếp đó tìm trong các thuộc tính chung một thuộc tính dùng để phân loại.

- Thuộc tính phân loại có thể bao gồm một hoặc một số chỉ tiêu phân loại, tất cả các chỉ tiêu của thuộc tính phân loại phải đặt trong cặp dấu mở-đóng ngoặc đơn, ví dụ: Loai_tau : (Toc_hanh, Tau_thuong)

- Sử dụng toán tử Case .. . of để phân loại, ví dụ:

```
Case Loai_tau : (Toc_hanh, Tau_thuong) of
   Toc_hanh: .....
   Tau_thuong:....
```

- Với mỗi chỉ tiêu phân loại, chúng ta có thể khai báo tên một số trường thay đổi hoặc khai báo một bản ghi con với cấu trúc thay đổi, ví dụ:

```
Case Loai_tau : (Toc_hanh, Tau_thuong) of
   Toc_hanh: (So_xuat_an, So_ga_do: Word);
   Tau_thuong: ( Case Cuoc :(cuoc_hanh_ly,cuoc_hang_hoa) of ..... );
```

- Các trường thay đổi nếu có cùng kiểu dữ liệu thì tên trường viết cách nhau bởi dấu phảy.

- Dữ liệu các trường phân loại phải thuộc kiểu đơn giản, cụ thể là:

kiểu nguyên, thực, logic, chuỗi, liệt kê, khoảng con.

Để phân loại chúng ta dùng toán tử Case … Of.

Cần chú ý rằng toán tử Case .. Of ở đây không giống như cấu trúc Case .. Of đã nêu trong phần các cấu trúc lập trình nghĩa là cuối phần khai báo không có từ khoá ***"End;"***

Trong vùng nhớ cấp phát cho chương trình sẽ có hai đoạn dành cho hai loại trường, đoạn thứ nhất dành cho các trường cố định, trong ví dụ 2.6 đoạn này có dung lượng là 26 byte. Đoạn thứ hai dành cho các trường thay đổi, đoạn này sẽ có dung lượng bằng dung lượng của chỉ tiêu phân loại lớn nhất.

Trong ví dụ 2.6 trường phân loại là Loai_tau, chỉ tiêu phân loại là toc_hanh và Tau_thuong. Với chỉ tiêu Toc_hanh, chúng ta khai báo hai trường thay đổi là So_xuat_an và So_ga_do còn với chỉ tiêu Tau_thuong có một trường là Cuoc_hang_hoa. Như vậy dung lượng của trường thay đổi của tàu tốc hành cần 3 byte còn tàu thường cần 6 byte, đoạn nhớ dành cho trường thay đổi sẽ có dung lượng 6 byte.

Chương trình quản lý đường sắt được thiết kế bao gồm một chương trình con lấy tên là NHAP dùng để nhập dữ liệu cho các đoàn tàu, phần thân chương trình chính sẽ yêu cầu nhập số chuyến tàu trên toàn tuyến và cho hiện dữ liệu ra màn hình (xem ví dụ 2.7).

Ví dụ 2.7

```
Program quan_ly_duong_sat;
Uses crt;
Type doan_tau = record
  ten_doan_tau:string[3];
  tuyen_duong:string[15];
  gio_di:real;
  loai:string[10];
  Case loai_tau: (toc_hanh,tau_thuong) of
     toc_hanh:(so_xuat_an:word;so_ga_do:byte);
     tau_thuong:(cuoc_hang_hoa:real);
End;
      dt = array[1..5] of doan_tau;
Var
      dt1:dt;
  n,i,j:byte;tg:doan_tau;
Procedure Nhap(m:byte;var qlds:dt);
Begin
  For i:= 1 to m do
  with qlds[i] do
Begin
  Write('Loai tau: ');readln(loai);
  if loai ='toc hanh' then
     Begin
     write('ten doan tau: '); readln(ten_doan_tau);
       write('tuyen duong: '); readln(tuyen_duong);
       write('gio xuat phat: '); readln(gio_di);
       write('so xuat an: '); readln(so_xuat_an);
       write('so ga do doc duong: '); readln(so_ga_do);
       writeln;
      end
      else
  if loai = 'tau thuong' then
  Begin
     write('ten doan tau: '); readln(ten_doan_tau);
       write('tuyen duong: '); readln(tuyen_duong);
       write('gio xuat phat: '); readln(gio_di);
       write('tien cuoc hang hoa: '); readln(cuoc_hang_hoa);
```

```
                writeln;
               End;
        End;   End;

    Begin
        clrscr;
        write('co bao nhieu doan tau tren toan tuyen: '); readln(n);
        writeln;
         nhap(n,dt1);
        clrscr;
        writeln('danh sach tau chay toan tuyen');
        for i:= 1 to n-1 do       {sap xep du lieu tang dan theo loai tau}
        for j:= i+1 to n do
        with dt1[i] do
        if dt1[i].loai < dt1[i+1].loai then
        begin
            tg:=dt1[i]; dt1[i]:=dt1[j]; dt1[j]:=tg;
        end;
        Writeln(' DANH  MUC CAC DOAN TAU TREN TUYEN');
         for i:= 1 to n do
        with dt1[i] do
        writeln(loai:10,' ',ten_doan_tau:3,' ',tuyen_duong:10,' ',gio_di:4:2,' ',so_xuat_an:3,'
',so_ga_do:3,cuoc_hang_hoa:10:2);
        readln;
    END.
```

- *Kiểu dữ liệu quản lý điểm của sinh viên*

SV là kiểu dữ liệu bản ghi dùng để quản lý điểm của sinh viên. Các trường cố định của SV bao gồm: MHS (mã hồ sơ), HOTEN (họ và tên), NS (ngày sinh), GIOI (nam, nữ), Khoa (Sư phạm - SP, Kinh tế - KT, Cơ điện - CD). Các môn học tuỳ thuộc vào khoa mà sinh viên đang theo học, giả sử chúng ta quy định khoa Sư phạm có các môn: Toán, Lý, Tin cơ bản, lập trình nâng cao, khoa Kinh tế có các môn: Kế toán máy, Marketing, khoa Cơ điện có các môn: Cơ học máy, Sức bền vật liệu, Hình hoạ. Tất cả sinh viên nếu là Nam thì học thêm môn Bơi lội, nếu là Nữ thì học thêm Thể dục nghệ thuật.

Rõ ràng là chúng ta không thể tạo ra kiểu bản ghi cố định cho sinh viên trong toàn trường, bởi lẽ số môn học không giống nhau. Các trường MHS, HOTEN, NS là chung cho mọi sinh viên, trường KHOA và GIOI dùng để phân loại sinh viên từ đó xác định các môn học. Vì rằng mỗi kiểu bản ghi chỉ có thể khai báo *duy nhất một trường phân loại ngang hàng với các trường cố định* nên cùng một lúc chúng ta không thể phân loại theo cả KHOA và GIOI. Giải pháp duy nhất là chọn một trường phân loại với hai chỉ tiêu phân loại đại diện cho Khoa và Gioi, giả sử tên trường phân loại bây giờ lấy tên là MONHOC và hai chỉ tiêu phân loại là PL1 và PL2. PL1 đại diện cho Gioi còn PL2 đại diện cho Khoa.

Các chỉ tiêu phân loại lại có thể trở thành thuộc tính phân loại với một số chỉ tiêu nào đó mà chúng ta gọi là *chỉ tiêu con* chẳng hạn xem PL1 là thuộc tính phân loại với hai chỉ tiêu con là Nam và Nu, PL2 là thuộc tính phân loại với ba chỉ tiêu con là SP,KT,CD. Mỗi chỉ tiêu con bao gồm một số trường cụ thể hoặc nó lại được sử dụng như trường phân loại mới....

Một bản ghi kiểu SV có thể có cấu trúc thuộc một trong các dạng sau:

* Sinh viên khoa Sư phạm

1/ Mhs, Hoten, Ns, Boi_loi, Toan, Ly, Tincoban, Lap_trinh_nang_cao

2/ Mhs, Hoten, Ns, The_duc, Toan, Ly, Tincoban, Lap_trinh_nang_cao

* Sinh viên khoa Cơ điện

3/ Mhs, Hoten, Ns, Boi_loi, Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa

4 / Mhs, Hoten, Ns, The_duc, Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa

* Sinh viên khoa Kinh tế

5 / Mhs, Hoten, Ns, Boi_loi, Ke_toan_may, Marketing

6 / Mhs, Hoten, Ns, The_duc, Ke_toan_may, Marketing

Có thể nhận thấy rằng tên các trường phân loại không có trong cấu trúc bản ghi. Nếu chúng ta muốn trong mỗi bản ghi lại có cả tên khoa và giới tính thì phải đưa thêm vào các trường cố định mới.

Kiểu dữ liệu bản ghi SV được khai báo như sau:

Ví dụ 2.6

```
Type
SV = record
   Mhs: Byte;
   Hoten: String[20];
   NS : Record
   Ngay:1..31;  Thang: 1..12; Nam: Word;
      End;
   Case  monhoc:(pl1,pl2) of
      pl1:( case gioi:(nam,nu) of
         Nam: (Boi_loi:real);
         Nu: (The_duc: real));
         pl2:( case KHOA: (SP,CD,KT) of
            SP: (Toan, Ly, Tincb, Ltnc: Real);
            CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);
            KT: (Ke_toan_may, Marketing:real));
   End;
```

Từ cách khai báo trên chúng ta rút ra một số nhận xét quan trọng sau đây:

Nhận xét 1

Trong một kiểu record các trường cố định được khai báo trước, trường phân loại khai báo sau, như vậy trường phân loại phải là trường khai báo cuối cùng. Các trường thay đổi khai báo bên trong trường phân loại.

Nhận xét 2

Mỗi kiểu dữ liệu Record có cấu trúc thay đổi chỉ được phép có duy nhất một trường phân loại, nghĩa là không thể có hai toán tử case .... of ngang hàng khi khai báo. Nếu chúng ta khai báo kiểu SV như trong ví dụ 2.8 sau đây thì sẽ nhận được thông báo lỗi :

Error in Expression

Ví dụ 2.8

```
Type SV = record
   Mhs: Byte;
   Hoten: String[20]
   NS : Record
         Ngay:1..31;
         Thang: 1..12;
         Nam: Word;
   End;
      Case  GIOI: (Nam, Nu) of
         Nam: Boi_loi:real;
         Nu:The_duc: real;
      Case KHOA: (SP,CD,KT) of
         SP: (Toan, Ly, Tin_cb, Ltnc: Real);
         CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);
         KT: (Ke_toan_may, Marketing:real);
   End;
```

Lỗi xuất hiện do chúng ta đã chọn hai trường phân loại là GIOI và KHOA ngang hàng nhau. Để khắc phục lỗi này chúng ta phải lựa chọn lại cấu trúc của Record. Thay vì có hai trường phân loại cùng cấp chúng ta chọn một trường là MONHOC, chỉ tiêu phân loại là PL1 và PL2. Lúc này tên môn học cụ thể sẽ tuỳ thuộc vào giá trị mà PL1 và PL2 có thể nhận (xem ví dụ 2.9)

Nhận xét 3

Vì mỗi trường lại có thể là một bản ghi cho nên bên trong trường phân loại lại có thể chứa các trường phân loại khác, đây là trường hợp bản ghi thay đổi nhiều mức.

Với kiểu dữ liệu SV đã định nghĩa chúng ta xây dựng chương trình quản lý điểm của sinh viên như ví dụ 2.9 sau đây.

Ví dụ 2.9

```
Program QUAN_LY_DIEM;
Uses crt;
Type
```

```
  SV = record
  Mhs: Byte;
  Hoten: String[20];
  NS : Record
         Ngay:1..31;
         Thang: 1..12;
         Nam: Word;
     End;
  Case  monhoc:(pl1,pl2) of
              pl1:( case gioi:(nam,nu) of
                  Nam: (Boi_loi:real);
                  Nu: (The_duc: real));
              pl2:( case KHOA: (SP,CD,KT) of
                  SP: (Toan, Ly, Tincb, Ltnc: Real);
                  CD: (Co_hoc_may, Suc_ben_vat_lieu, Hinh_hoa:real);
                  KT: (Ke_toan_may, Marketing:real));
              End;
Var
  Ds:Array[1..100] of sv;  tg:sv;
  i,j,k:byte; pl,tk:string[3]; tl,gt:char;
BEGIN
  clrscr;   i:=1;
  writeln(' Nhap diem cho sinh vien ');
  Repeat
  With ds[i] do
     Begin
        Write('Nhap ma ho so '); readln(mhs);
        Write('Nhap ho va ten '); readln(hoten);
         With ns do
            Begin
               Write('Nhap ngay sinh '); readln(ngay);
               Write('Nhap thang sinh '); readln(thang);
               Write('Nhap nam sinh '); readln(nam);
            End;
     Write('Cho biet gioi tinh nam "T" - nu "G" '); Readln(gt);
     if upcase(gt)='T' then ds[i].gioi:=nam else ds[i].gioi:=nu;
   Case ds[i].gioi of
        nam: begin
        Write('Nhap diem mon boi loi '); Readln(boi_loi);
         end;
        nu: begin
```

```
          Write('Nhap diem mon The duc '); readln(the_duc);
          end;    End;  {ket thuc Case ds[i].gioi...}
Write('Nhap ten khoa '); Readln(tk);
   for k:= 1 to length(tk) do tk[k]:=upcase(tk[k]); { chuyển tên khoa thành chữ in}
   if tk='SP' then ds[i].khoa:=sp
   else if tk='CD' then  ds[i].khoa:=cd
   else ds[i].khoa:=kt;
 Case ds[i].khoa of
    sp:Begin
          Write('Nhap diem mon Toan '); Readln(toan);
          Write('Nhap diem mon Ly '); Readln(ly);
          Write('Nhap diem mon Tin Co ban '); Readln(tincb);
          Write('Nhap diem mon Lap trinh nang cao '); Readln(ltnc);
   End;
    cd: Begin
          Write('Nhap diem mon Co hoc '); Readln(co_hoc_may);
          Write('Nhap diem mon Suc ben vat lieu '); Readln(suc_ben_vat_lieu);
          Write('Nhap diem mon Hinh hoa '); Readln(hinh_hoa);
   End;
    kt: Begin
          Write('Nhap diem mon Ke toan may '); Readln(ke_toan_may);
          Write('Nhap diem mon marketing '); Readln(marketing);
   End;
  End; {ket thuc Case..}
                                {Sap xep du lieu tang dan theo ten Khoa}
for j:=1 to i-1 do
for k:=j+1 to i do
  if ds[j].khoa<ds[j+1].khoa then
  Begin
  tg:=ds[j];  ds[j]:=ds[k];  ds[k]:=tg;
  End;

 End; {ket thuc with}
    writeln;
    Write('Nhap tiep hay thoi? C/K '); readln(tl);
    if upcase(tl)='C' then i:=i+1;
    Until upcase(tl)='K';

{Hien du lieu da nhap }
Write('Co xem du lieu khong? C/K '); readln(tl);
```

```
            if upcase(tl)='C' then
          Begin
              Writeln('DU LIEU DA NHAP ');
              For j:= 1 to i do
              with ds[j] do
           Begin
                  write(mhs:3,' ',hoten:20);
                  with ns do write(' ',ngay,'/',thang,'/',nam);
                  case ds[j].khoa of
                  sp:writeln(' Khoa SP ',toan:4:1,ly:4:1,Tincb:4:1,ltnc:4:1);
                  cd:writeln(' Khoa CD ', co_hoc_may:4:1, suc_ben_vat_lieu:4:1,
hinh_hoa:4:1);
                  kt:writeln(' Khoa KT ',ke_toan_may:4:1,marketing:4:1);
                  end;
            End;
            End;
            Readln;
        END.
```

Ví dụ 2.9 tuy đã chạy hoàn chỉnh song có một số nhược điểm sau:

* Tổ chức chương trình chưa hợp lý

* Dữ liệu đưa ra màn hình chưa đẹp

Bạn đọc có thể thiết kế lại bằng cách đưa vào chương trình con Nhap và chương trình con Hien. Chương trình con Hien có thể thiết kế để dữ liệu đưa ra dưới dạng bảng theo mẫu sau đây:

DANH SACH SINH VIEN TRUONG .....

Ma ho so	Ho va Ten	Gioi	Khoa	So mon hoc	Tong diem	Trung binh
.....						

b. Truy nhập

Việc truy nhập vào các trường cố định của bản ghi có cấu trúc thay đổi hoàn toàn giống như bản ghi thường. Còn việc truy nhập và các trường thay đổi cần phải chú ý một số điểm sau:

- Không dùng phép gán hoặc nhập dữ liệu từ bàn phím cho các trường phân loại. Nếu trong ví dụ 2.9 chúng ta đưa vào lệnh

monhoc:='Toan';

thì sẽ nhận được thông báo lỗi: Type Mismatch

còn nếu đưa vào lệnh

Read(monhoc);

thì sẽ nhận được thông báo: Cannot Read or Write Variables of this Type

- Lệnh With ... Do có tác dụng với tất cả các trường kể cả các trường thay đổi bên trong các trường phân loại. Cụ thể trong ví dụ 2.9 với lệnh

With ds do chúng ta có thể đưa vào trực tiếp lệnh

Write('Nhap diem mon boi loi '); Readln(boi_loi);

- Tên các trường phân loại không thể đưa ra màn hình như là một tên trường bình thường nghĩa là cũng giống như trong mục 2.2 không thể viết lệnh Write(monhoc). Trong trường hợp cần thiết chúng ta có thể sử dụng các biến trung gian.

2. Dữ liệu kiểu tệp (file)

2.1 Khái niệm về tệp

Tệp dữ liệu là một dãy các phần tử cùng kiểu được sắp xếp một cách tuần tự. Tệp dữ liệu được cất giữ ở bộ nhớ ngoài (đĩa mềm hoặc đĩa cứng) dưới một tên nào đó, cách đặt tên tuân theo quy định của DOS nghĩa là phần tên tệp dài không quá 8 ký tự và phần đuôi không quá 3 ký tự.

Tệp tập hợp trong nó một số phần tử dữ liệu có cùng cấu trúc giống như mảng (Array) song khác mảng là số phần tử của tệp chưa được xác định.

Trong Pascal có 3 loại tệp được sử dụng là:

a. Tệp có kiểu

Tệp có kiểu là tệp mà các phần tử của nó có cùng độ dài và cùng kiểu dữ liệu. Với những tệp có kiểu chúng ta có thể *cùng một lúc đọc dữ liệu từ tệp ra hoặc nhập dữ liệu vào tệp.*

b. Tệp văn bản (Text)

Tệp văn bản dùng để lưu trữ dữ liệu dưới dạng các ký tự của bảng mã ASCII, các ký tự này được lưu thành từng dòng, độ dài của các dòng có thể khác nhau. Khi ghi một số nguyên, ví dụ số 2003 (kiểu word) vào tệp văn bản, Pascal cần 4 byte cho bốn ký tự chứ không phải là 2 byte cho một số. Việc ghi các số nguyên hoặc thực vào tệp văn bản sẽ phải qua một công đoạn chuyển đổi, điều này sẽ do Pascal tự động thực hiện. Để phân biệt các dòng Pascal dùng hai ký tự điều khiển là CR - về đầu dòng và LF - xuống dòng mới.

Trong bảng mã ASCII ký tự CR = CHR(13) còn LF = CHR(10)

c. Tệp không kiểu

Tệp không kiểu là một loại tệp không cần quan tâm đến kiểu dữ liệu ghi trên tệp. Dữ liệu ghi vào tệp không cần chuyển đổi.

Khi khai báo một biến kiểu tệp thì đồng thời ta cũng phải tạo ra một mối liên hệ giữa biến này và một tệp dữ liệu lưu trên thiết bị nhớ ngoài. Cách khai báo này đã được chuẩn hoá trong Pascal.

Kiểu dữ liệu của các phần tử trong biến kiểu tệp có thể là bất kỳ kiểu nào trừ kiểu của chính nó tức là trừ kiểu tệp. Ví dụ nếu kiểu phần tử là một mảng một chiều của các số nguyên ta có tệp các số nguyên, nếu kiểu phần tử là Record ta có tệp các Record...

Tác dụng lớn nhất của kiểu dữ liệu tệp là ta có thể lưu trữ các dữ liệu nhập vào từ bàn phím cùng các kết quả xử lý trong bộ nhớ RAM ra tệp để dùng nhiều lần. Các kiểu dữ liệu đã học chỉ xử lý trong RAM và in kết quả ra màn hình hoặc ra máy in, khi kết thúc chương trình hoặc mất điện cả dữ liệu nhập vào và kết quả xử lý đều bị mất.

d. Khai báo

Tệp có kiểu được định nghĩa sau từ khoá TYPE còn biến kiểu tệp được khai báo sau từ khoá VAR. Thủ tục khai báo biến kiểu tệp gồm 2 cách:

- Định nghĩa kiểu tệp với từ khoá FILE OF trong phần mô tả kiểu sau từ TYPE, tiếp theo là khai báo biến tệp trong phần khai báo biến

Ví dụ 2.10

```
Type
MSN = Array[1..100] of integer;       (*định nghĩa mảng 100 số nguyên*)
TSN = File of MSN;   (* định nghĩa  dữ liệu kiểu tệp TSN có các phần tử là mảng số nguyên *)
CHUVIET = File of String[80];        (* định nghĩa CHUVIET là tệp các chuỗi có độ dài 80 ký tự *)
BANGDIEM = Record
Hoten: String[25];
Gioitinh: Char;
Lop: String[5];
Diachi: String[50];
Toan,Ly,Hoa: Real;
END;
TBD = File of BANGDIEM;
VAR
Tep1: TSN;          (* biến tep1 có các phần tử là mảng số nguyên *)
Tep2: CHUVIET;        (* biến Tep2 có các phần tử là chuỗi ký tự *)
Tep3: TBD; (* biến Tep3 có các phần tử là record*)
.....
```

- Định nghĩa trực tiếp biến kiểu tệp trong phần khai báo biến

Ví dụ 2.11

```
Var
Tep4: File of Array[1..5] of String[80];
```

Tep5: File of BANGDIEM;

Trong phần khai báo này biến Tep4 là một biến kiểu tệp, tệp này có 5 phần tử, mỗi phần tử là một chuỗi dài tối đa 80 ký tự. Biến Tep5 là một biến tệp mà các phần tử của nó có kiểu Bangdiem.

e. Truy nhập vào tệp

Các phần tử của tệp được lưu giữ tuần tự thành một dãy và việc truy nhập vào từng phần tử phụ thuộc vào thiết bị ghi, đọc của máy vi tính. Turbo Pascal có thể xử lý hai loại tệp là: **Tệp truy nhập tuần tự và tệp truy nhập trực tiếp.**

* **Tệp truy nhập tuần tự**: để truy nhập vào một phần tử nào đó ta bắt buộc phải đi qua các phần tử trước đó. Nếu muốn thêm các phần tử vào tệp thì chỉ có thể thêm vào cuối tệp. Tệp kiểu này dễ hình dung, dễ xử dụng song không linh hoạt, tốn thời gian xử lý. Việc truy nhập tuần tự thường được thực hiện thông qua một vòng lặp. *Tệp văn bản là tệp thuộc kiểu này*.

* **Tệp truy nhập trực tiếp**: là tệp có thể truy nhập vào phần tử bất kỳ trong tệp, trên những thiết bị nhớ ngoài cổ điển như băng từ, băng đục lỗ không thể tạo tệp kiểu này vì không thể đọc ngay vào giữa băng. Chỉ những máy sử dụng đĩa (mềm hoặc cứng) thì mới có thể tạo tệp truy nhập trực tiếp vì có thể điều chỉnh để đầu từ đặt đúng vào một cung từ chứa dữ liệu nào đó.

Muốn truy nhập trực tiếp phải dùng thủ tục Seek(số hiệu phần tử).

f. Mở tệp

Để mở một tệp chuẩn bị lưu trữ dữ liệu Pascal xử dụng hai thủ tục chuẩn sau đây:

ASSIGN(biến tệp, tên tệp); (*liên kết biến tệp với một tên tệp sẽ ghi vào thiết bị nhớ ngoài *)

REWRITE(biến tệp); (* tạo một biến tệp rỗng chuẩn bị nhập dữ liệu vào *)

Trong đó:

* Biến tệp: là tên biến tệp đã khai báo sau từ khoá VAR

* Tên tệp: Là tên do ta chọn để ghi dữ liệu vào đĩa (theo quy định của DOS). Tên tệp có thể bao gồm cả đường dẫn tới thư mục mà chúng ta lựa chọn. Đường dẫn và tên tệp phải đặt trong dấu nháy đơn. Ví dụ:

ASSIGN(f,'a:\baitap.txt');

Sau thủ tục REWRITE ta sẽ có một tệp rỗng vì chưa có phần tử nào được đọc vào. Nếu trên thiết bị nhớ ngoài ta đã có sẵn một tệp trùng tên với tên ghi ở thủ tục ASSIGN thì tệp ngoài sẽ bị xoá đi.

Sau hai thủ tục chuẩn bị trên đây, để tiến hành ghi dữ liệu vào tệp ta lại dùng thủ tục WRITE(....) như đã biết.

Cách viết:

WRITE(biến tệp, các giá trị cần ghi vào tệp);

Bước cuối cùng là phải đóng tệp lại bằng thủ tục

CLOSE(biến tệp);

Sau thủ tục này dữ liệu sẽ được lưu trên đĩa và tệp sẽ bị đóng.

2.2 Tệp văn bản

a. Khai báo tệp văn bản

Tệp văn bản được khai báo trực tiếp trong phần khai báo biến:

```
Var
Bien_tep : Text;
```

Ví dụ

```
Var
Hoso: Text;
T1,T2,T3:Text;
```

b. Truy nhập vào tệp

Truy nhập vào tệp được hiểu là nhập dữ liệu vào tệp, ghi lại dữ liệu trên thiết bị nhớ ngoài, đọc dữ liệu đã có ra màn hình hoặc máy in và xử lý dữ liệu đó.

Đối với tệp văn bản việc ghi dữ liệu vào tệp có thể thực hiện qua hai cặp thủ tục sau:

- Mở tệp mới để ghi

```
Assign(bien_tep, Đường dẫn\ten_tep);
Rewrite(bien_tep);
```

Cặp thủ tục trên dùng để mở một tệp mới chuẩn bị nhận dữ liệu, nếu trong bộ nhớ ngoài đã có tệp trùng tên thì tệp ở ngoài sẽ bị xoá. Nếu bỏ qua *đường dẫn* thì tệp sẽ được lưu vào thư mục hiện hành tức là thư mục mà từ đó Pascal đã được khởi động (C:\TP\BIN). Giả sử chúng ta muốn lưu tệp với tên là BT1.DAT vào thư mục BAITAP trên đĩa A thì phải viết lệnh:

```
Assign(bien_tep, 'A:\BAITAP\BT1.DAT');
```

- Mở tệp đã có để ghi thêm

```
Assign(bien_tep, ten_tep);
Append(Bien_tep);
```

Cặp thủ tục trên mở một tệp đã lưu trong thiết bị nhớ ngoài để nhập thêm dữ liệu, nếu tệp không có thì máy sẽ thông báo lỗi. Dữ liệu nhập thêm sẽ được ghi vào cuối tệp.

- Mở tệp để đọc dữ liệu

Dưới đây là cặp thủ tục dùng để mở tệp đã tồn tại, chuẩn bị cho việc đọc dữ liệu từ tệp ra thiết bị ngoài:

```
Assign(bien_tep, ten_tep);
Reset(bien_tep);
```

c. Ghi dữ liệu vào tệp

Sau khi đã mở tệp chúng ta có thể dùng thủ tục Write hoặc Writeln để ghi dữ liệu vào tệp.

Cú pháp:

Write(Bien_tep, giá trị 1, giá trị 2, ...);

Hoặc

Writeln(Bien_tep, giá trị 1, giá trị 2, ...);

Sự khác nhau giữa Write và Writeln là ở chỗ thủ tục Write sẽ ghi dữ liệu liên tục trên các ô nhớ, không có các ký hiệu về đầu dòng và xuống dòng. Thủ tục Writeln sẽ đưa thêm vào cuối dòng các ký tự điều khiển CR (về đầu dòng) và LF (nhảy xuống dòng dưới).

Giá trị 1, giá trị 2... trong lệnh ghi có thể là chuỗi ký tự viết tường minh, tên biến hoặc biểu thức, các giá trị phải được phân cách ít nhất bằng một khoảng trống.

Thủ tục Writeln(Bien_tep); sẽ tạo nên một dòng rỗng không chứa ký tự nào cả.

Dữ liệu trước khi ghi vào tệp văn bản có thể thuộc các kiểu đơn giản sau đây: ***Số nguyên, số thực, ký tự, chuỗi, logic.*** Dù thuộc kiểu gì thì khi đã ghi vào tệp đều được chuyển đổi thành kiểu ký tự. Điều này cũng đồng nghĩa với quy định rằng các kiểu dữ liệu có cấu trúc như: *Array,* ***Set, Record, File*** không thể lưu trữ trên tệp văn bản.

Khi có thủ tục Close(bien_tep) Pascal sẽ đưa thêm vào dòng cuối cùng ký hiệu kết thúc tệp EOF (End of File).

Ví dụ 2.12

```
Var
T1:text;
Begin
Assign(T1,'DULIEU.DAT');
Rewrite(T1);
Writeln(t1,'Tep van ban');
Write(T1,123);
Write(T1,' ',123.45);
Writeln(T1);
Close(T1);
End.
```

Ví dụ 1.13 tạo ra tệp văn bản DULIEU.DAT tệp này sẽ được lưu trữ tại thư mục hiện hành tức là C:\TP\BIN.

Lệnh Writeln(t1,'Tep van ban'); sẽ ghi vào biến tệp T1 (cũng có nghĩ là ghi vào tệp DULIEU.DAT) dòng chữ "Tep van ban" tiếp đó là các ký tự điều khiển CR, LF.

Lệnh Write(T1,123); sẽ tự động chuyển đổi số 123 thành ký tự và ghi vào tệp không kèm theo CR, LF.

Lệnh Write(T1,' ',123.45); ghi vào tệp một khoảng trống trước dãy 123.45 tiếp đó chuyển đổi và ghi số 123.45 vào tệp theo quy cách ngầm định của Pascal.

Dữ liệu sẽ được ghi vào tệp như sau

Tep van ban

123 1.234500000E+02

(dòng trống)

Từ ví dụ trên có thể nêu một số nhận xét sau:

* Các ký tự nhập vào tệp phải nằm trong cặp dấu nháy đơn

* Các số không cần để trong dấu nháy, Pascal sẽ tự động chuyển đổi chúng khi ghi vào tệp. Nếu chúng ta không chỉ rõ quy cách đọc hoặc ghi số thì các số sẽ được ghi theo quy cách chuẩn, cụ thể là:

- Số nguyên sẽ được ghi chính xác như giá trị của nó trong lệnh Write(...)

- Số thực sẽ được chuyển thành dạng khoa học nghĩa là dạng số với luỹ thừa cơ số 10, ví dụ 123.45 chuyển thành 1.234500000E+02. Trong cách viết khoa học Pascal ngầm định như sau:

- Phần nguyên là 1 ký tự
- Phần lẻ là 10 ký tự kể cả dấu chấm thập phân
- Phần còn lại là 4 ký tự cho luỹ thừa cơ số 10, Ví Dụ:

E+02 = 10^2 , E-03 = 10^{-3}

* Các số có thể chỉ rõ quy cách ghi như sau

Write(T1,123:5);

Ghi vào tệp số nguyên 123 với độ dài 5 ký tự, Pascal sẽ để trống hai vị trí bên trái

Write(T1,' ',123.45:6:2);

Số 123.45 được ghi vào tệp với độ dài 6 ký tự và 2 ký tự chứa số lẻ

d. Đọc dữ liệu từ tệp văn bản

Dữ liệu lưu trữ trong tệp văn bản có thể cho hiện lên màn hình bằng lệnh TYPE của DOS, bằng các phím chức năng F3, F4 của NC hoặc đưa vào Word để xem và sửa chữa nếu cần.

Với Pascal tệp văn bản thuộc loại tệp truy nhập tuần tự, do vậy nếu cố tình dùng lệnh truy nhập ngẫu nhiên Seek(...) thì máy sẽ báo lỗi:

Error 63: Invalid File Type

Sau khi tiến hành mở tệp con trỏ tệp sẽ được đặt tại dòng đầu, Cách thức mà Pascal dùng để đọc dữ liệu là như sau: dùng thủ tục Read, hoặc Readln để đọc dữ liệu từ dòng hiện thời và gán vào biến tương ứng, viết biến đó ra màn hình hoặc máy in.

Read(Bien_tep, Dong); hoặc Readln(Bien_tep, Dong);

Write(Dong); hoặc Writeln(Dong);

Trong hai ví dụ trên Biến trung gian là Dong, biến này phải được khai báo trước và phải thuộc kiểu String.

Để có thể viết toàn bộ dữ liệu từ một tệp văn bản ra các thiết bị ngoài thì các lệnh đọc viết trên phải được lặp đi lặp lại từ dòng 1 đến dòng cuối cùng nghĩa là chương trình phải sử dụng một trong hai vòng lặp:

```
While not eof(Bien_tep) Do
 Begin
   Readln(Bien_tep, Dong);
   Writeln(Dong);
```

```
 End;
Hoặc
 For  i := 1 to Filesize(Bien_tep)   Do
    Begin
       Readln(Bien_tep, Dong);
       Writeln(dong);
    End;
```

Nếu trên một dòng chúng ta đã ghi nhiều giá trị ứng với các biến khác nhau thì không thể đọc riêng từng biến. Cách mà chúng ta ra lệnh cho Pascal ghi dữ liệu vào tệp sẽ ảnh hưởng rất nhiều đến việc chúng ta gọi dữ liệu ra và xử lý dữ liệu đó.

Ví dụ sau đây sẽ giải thích cụ thể điều này.

Ví dụ 2.14

```
  Var
     F:text;
     Hoten:string[25]; heso:real;  socon:byte; t:string;
  Begin
  Hoten:=' Tran Van Tam';    Heso:=4.25;    Socon:= 2; {gán dữ liệu cho các trường}
 Assign(f,'hoso.txt');
 Rewrite(f);
 Witeln(f,hoten,' ',heso:4:2,' ',socon);
  Close(f);
  Reset(f);
  Readln(f,t);     {đọc dữ liệu từ dòng hiện thời và gán vào biến t }
  Writeln(t);
Readln;
End.
```

Chương trình trên đây có nhiều điều cần phải lưu ý:

- Lệnh ghi vào tệp:

Witeln(f,hoten,' ',heso:4:2,' ',socon); sẽ ghi tất cả dữ liệu trên một dòng.

	T	r	a	n		V	a	n		T	a	m		4	.	2	5		2	CR	LF	EOF

Nhìn vào hình vẽ dễ dàng nhận thấy rằng chuỗi Hoten chiếm 13 Byte, Heso chiếm 4 Byte và Socon chiếm 1 Byte. Trong thực tế các số kiểu Real có độ dài là 6 Bytes nhưng vì chúng ta đã yêu cầu Pascal ghi Heso dài 4 ký tự trong đó có hai số lẻ nên khi ghi vào tệp, Heso chỉ chiếm 4 Bytes. Giữa các biến có hai khoảng trống do đó tổng chiều dài của dòng là 20 Bytes.

Lệnh đọc dữ liệu từ tệp Readln(f,t) sẽ đọc cả dòng hiện thời vào biến t và lệnh viết Writeln(t) sẽ đưa ra màn hình cả dòng nghĩa là ta lại nhận được toàn bộ dòng như trên. Cần

phân biệt rằng dữ liệu viết ra là các ký tự của bảng mã chứ không phải các chữ và số như ta đã nhập vào.

Trường hợp biến t không đủ độ dài để chứa hết cả dòng thì số ký tự thừa sẽ bị cắt bỏ.

- Nếu chúng ta viết lệnh đọc:

Readln(f,Hoten,' ',Heso,' ',socon);

thì máy sẽ báo lỗi: Error 106 : Invalid Numeric Format

nghĩa là không thể đọc trực tiếp cùng một lúc từ tệp ra màn hình cả biến kiểu ký tự và biến kiểu số.

- Nếu chúng ta viết lại lệnh ghi dữ liệu:

Witeln(f,hoten);

Writeln(f,heso:5:2,' ',socon);

nghĩa là ghi riêng dữ liệu kiểu "chữ" trên một dòng còn dữ liệu kiểu "số" trên dòng khác, giữa các số có một khoảng cách, sau đó dùng lệnh đọc:

Readln(f,hoten);

Readln(f,heso, Socon);

thì lại nhận được thông báo lỗi ở dòng Readln(f,heso, Socon);

Error 106 : Invalid Numeric Format

nghĩa là các số ghi vào tệp trên cùng một dòng thì cũng không thể đọc chúng như là các biến.

- Chúng ta sửa lại lệnh ghi một lần nữa

Witeln(f,hoten);

Writeln(f,heso:5:2);

Writeln(f,socon);

Rồi dùng lệnh đọc

Readln(f,Hoten);

Readln(f,Heso);

Readln(f,socon);

sau đó là lệnh viết

Writeln(Hoten, ' ', Heso:5:2,' ',socon,' ' ,Heso*290000);

thì chương trình sẽ không báo lỗi .

Đến đây có thể rút ra kết luận là:

- Muốn lấy lại kiểu của dữ liệu nhập vào tệp văn bản thì mỗi biến phải nhập trên một dòng.

- Với các biến kiểu số đã ghi riêng rẽ trên một dòng khi gọi ra Pascal sẽ tự động chuyển đổi từ dạng ký tự thành dạng số và ta có thể đưa các số này vào các biểu thức tính toán bình thường. Trong ví dụ trên chúng ta tính Lương bằng cách lấy Heso*290000.

- Trong bộ nhớ của máy dữ liệu được ghi liên tục trong các ô nhớ, để phân biệt các dòng Pascal dùng cặp ký tự điều khiển CR và LF. Nói cách khác dữ liệu được lưu trữ liên tục chứ không phải dưới dạng bảng, khi lấy dữ liệu ra chúng ta cũng lấy liên tục nhưng lại có thể bố trí trên màn hình sao cho trực quan và dễ theo dõi.

Ví dụ 2.15: xây dựng một chương trình đơn giản để quản lý công chức. Dữ liệu nhập vào bao gồm Họ tên, Hệ số lương và Số con. Dữ liệu xuất ra màn hình bao gồm Họ tên, Hệ số lương, Số con và Lương tháng, Lương tháng ở đây tính theo quy định của nhà nước = heso*290000.

Chương trình đặt ra hai khả năng lựa chọn:

a. Nếu tệp dữ liệu đã tồn tại thì nhập thêm người

b. Nếu tệp chưa có thì mở tệp mới

Trong cả hai trường hợp đều yêu cầu cho biết số người cần nhập.

Dữ liệu in ra dưới dạng bảng.

Ví dụ 2.15

```
Program Quan_ly_can_bo;
Uses crt;
Var f:text; hoten:string[20]; c1,heso:real; c2,i,n,socon:byte;
      ten:string[12];
  Begin
  clrscr;
  Write('Cho biet ten tep '); readln(ten);
  {$I-}
  assign(f,ten);
  reset(f);
  {$I+}
   if Ioresult=0 then
   Append(f)
   else
  rewrite(f);
  write('Nhap bao nhieu nguoi '); readln(n);
   for i:= 1 to n do
     Begin
     Write(' Ho ten ' ); Readln(hoten);
     Write( ' He so '); readln(heso);
     Write(' So con ' ); Readln(socon);
     Writeln(f,hoten);
     Writeln(f,heso:4:2);
     writeln(f,socon);
     End;
     close(f);
```

```
  assign(f,ten);
  reset(f);
  writeln(':----------------------:------:----------:---------------:');
  writeln('|   Ho va ten       | Hs | socon|    Luong   |');
  writeln(':----------------------:-------:---------:----------------:');
  while not eof(f) do
    begin
  readln(f,hoten);
  readln(f,heso);
  readln(f,socon);
 writeln('|',hoten:19,'|',heso:4:2,'| ',socon:4,' | ',heso*290000:10:2,' |');
    end;
    readln;
    End.
```

Ví dụ 2.15 có sử dụng định hướng chương trình dịch {$I+}, {$I-}. Khi định hướng là {$I+} thì chương trình sẽ kiểm tra lỗi vào ra IO (Input, Output) nếu phát hiện thấy lỗi thì dừng chương trình, đây là chế độ ngầm định của Pascal. Nếu định hướng là {$I-} thì việc kiểm tra lỗi vào ra tạm thời không thực hiện, nghĩa là nếu phát hiện thấy lỗi thì tạm treo các thủ tục vào ra và tìm xem hàm IORESULT cho kết quả là gì. Nếu hàm này cho kết quả bằng 0 thì có nghĩa là việc kiểm tra IO không có gì sai sót và chương trình tiếp tục làm việc. Nếu hàm Ioresult cho kết quả khác 0 thì có nghĩa là việc kiểm tra IO phát hiện thấy lỗi và chương trình cần phải được sửa chữa.

Ví dụ 2.16: Tạo tệp văn bản Baitho.txt để lưu trữ một bài thơ có n dòng, dòng cuối cùng ghi " Nam 2003". Khi nhập bài thơ cần hỏi trước bài thơ có bao nhiêu dòng.

Ví dụ 2.16

```
Program tep_van_ban;
Uses crt;
var i,n:integer;  f:text; t:string;
Begin
clrscr;
assign(f,'baitho.txt');rewrite(f);
writeln('Bai tho gom bao nhieu cau?  '); readln(n);
writeln('Hay nhap bai tho cua ban');
for i:= 1 to n do
begin
write('Cau ',i,'  '); readln(t); writeln(f,t);
```

```
end;
writeln(f,'Nam ', 2003);
close(f);
clrscr;
gotoxy(15,5); i:=6;
clrscr;
writeln('Du lieu viet tu tep baitho.txt ');
reset(f);
while not eof(f) do
begin
readln(f,t); gotoxy(15,i);
writeln(t);
i:=i+1;
end;
readln;
End.
```

Ví dụ 2.16 có thể cải tiến theo nhiều cách để có được một chương trình đẹp dùng cho việc lưu trữ văn bản, chẳng hạn chúng ta sẽ đưa vào các chương trình con GHIMOI, GHITHEM và DOC phục vụ việc ghi dữ liệu vào tệp mới, ghi thêm dữ liệu vào tệp đã có hoặc đọc từ tệp ra. Trước khi ghi hoặc đọc cần kiểm tra xem tệp đã tồn tại chưa, khi nhập văn bản vào tệp không hạn chế số dòng, muốn kết thúc việc nhập thì khi bắt đầu một dòng mới chỉ cần bấm dấu "*" v.v... (xem ví dụ 2.17)

Ví dụ 2.17:

```
Program tep_van_ban;
Uses crt;
var i,j,n:integer;  f:text; t:string;
   tl,tl1,tl2:char; ten:string[12];

Procedure Ghimoi(ten:string);
Begin
clrscr;
assign(f,ten);
rewrite(f);
writeln('Hay nhap bai tho cua ban');
 i:=1;
 repeat
 write('Cau ',i,'  '); readln(t);
 if t<>'*' then  writeln(f,t);
```

```
 i:=i+1;
 Until t='*';
 close(f);
End;

Procedure Ghithem (ten:string);
Begin
clrscr;
assign(f,ten);
append(f);
writeln('Hay bo xung bai tho cua ban');
 i:=1;
 repeat
 write('Cau ',i,'  '); readln(t);
 if t<>'*' then  writeln(f,t);
 i:=i+1;
 Until t='*';
close(f);
end;

  Procedure Doc(tep:string);
  Begin
clrscr;
gotoxy(15,5); i:=6;
writeln('Du lieu viet tu tep ',tep);
assign(f,tep);
reset(f);
while not eof(f) do
begin
readln(f,t); gotoxy(15,i);
writeln(t);
i:=i+1;
end;
readln;
end;
  BEGIN   { Thân chương trình mẹ}
  Clrscr;
  Write('cho biet ten tep '); Readln(ten);
  Write('Ban Ghi hay Doc du lieu G/D '); Readln(tl1);
    If upcase(tl1)='G' then
```

```
  Begin
 write('Ghi tep moi hay ghi them vao tep cu M/C ');
 Readln(tl2);
  If upcase(tl2)='M' then Ghimoi(ten)
  else
  Ghithem(ten);
  End
  Else
  Doc(ten);
END.
```

Trong ví dụ 2.17 có sử dụng một biến toàn cục là TEN, kiểu biến là string[12]. Khi lấy Ten làm tham số thực để truyền cho các chương trình con thì cần chú ý định nghĩa kiểu dữ liệu trước. Cụ thể nếu chúng ta viết dòng lệnh:

Procedure Doc(tep:string);

Dưới dạng mới là

Procedure Doc(tep:string[12]); thì máy sẽ báo lỗi.

2.3 Tệp có kiểu

Tệp có kiểu là tệp mà mọi phần tử đều có cùng độ dài và cùng kiểu. Kiểu các phần tử của tệp có thể là số nguyên, thực, ký tự, chuỗi, mảng hoặc bản ghi. Cách thức khai báo biến kiểu tệp đã trình bày trong mục II. Sự khác nhau cơ bản giữa tệp có kiểu và tệp văn bản là tệp có kiểu ***có thể vừa ghi vào vừa đọc ra,*** còn với tệp văn bản chúng ta buộc phải kết thúc ghi bằng lệnh Close(bien_tep) thì mới có thể đọc tệp, còn khi đang đọc tệp chúng ta cũng phải kết thúc đọc và đóng tệp thì mới có thể ghi thêm dữ liệu vào tệp.

a. Đọc và ghi

- Ghi lên tệp

Write(bientep, bien1, bien2, ...)

bien1, bien2, ... là các biến cùng kiểu với biến tệp.

- Đọc tệp

Read(Bientep, bien1, bien2, ...)

Đọc tệp thực chất là đọc các phần tử của tệp và gán cho các bien1, bien2,... cùng kiểu. Việc đọc diễn ra trong bộ nhớ do đó người sử dụng muốn biết được các giá trị cụ thể thì phải viết các biến đã đọc lên màn hình.

Chú ý:

Khác với tệp văn bản, việc ghi và đọc tệp có kiểu không sử dụng các lệnh Writeln hoặc Readln nghĩa là tệp có kiểu không ghi dữ liệu thành các dòng. Các phần tử của tệp có kiểu được ghi liên tục trong các ô nhớ và chỉ có ký hiệu kết thúc tệp EOF.

Khi chúng ta đọc hoặc ghi xong một phần tử thì con trỏ tệp sẽ tự động chuyển đến vị trí kế tiếp.

b. Truy nhập vào tệp

Seek(bientep,i); i=0,1,2,.....

Thủ tục Seek sẽ định vị con trỏ tại phần tử thứ i của tệp, Pascal quy định vị trí các phần tử là 0, 1, 2... như vậy phần tử thứ nhất là phần tử có vị trí 0.

Lệnh Seek(bientep, 5); sẽ định vị con trỏ tệp tại phần tử thứ 6.

Các tệp sau khi mở theo ngầm định con trỏ tệp luôn định vị tại phần tử đầu tiên nghĩa là phần tử có số thứ tự là 0. Trong quá trình đọc nếu chúng ta dùng một cấu trúc lặp để đọc thì khi đọc xong một phần tử con trỏ tệp sẽ tự động chuyển đến phần tử kế tiếp.

```
while not eof(BT) do   {Kiểm tra xem con trỏ tệp đã nằm ở phần tử cuối chưa}
begin
Read(bt,i); write(i:5);
end;
```

c. Các hàm xử lý tệp

*. **Filesize(bientep)** cho biết số phần tử có trong tệp

*. **FilePos(bientep)** cho biết vị trí hiện thời của con trỏ tệp

*. **Eof(Bientep)** cho giá trị True nếu con trỏ tệp ở vị trí cuối tệp (vị trí cuối tệp được hiểu là vị trí sau phần tử cuối cùng), nếu con trỏ tệp định vị tại bất kỳ phần tử nào của tệp thì hàm eof() cũng cho giá trị False.

Giả sử tệp DS có 8 phần tử khi đó hàm

Filesize(DS) sẽ cho kết quả là số 8,

còn thủ tục

Seek(DS,Filesize(DS))

sẽ định vị con trỏ tại phần tử thứ 8 (số thứ tự của phần tử này là 7) nghĩa là cuối tệp (bởi vì phần tử đầu tiên có số thứ tự là 0).

Ví dụ 2.18

Tạo một tệp lấy tên là TEPCK.DAT để vừa ghi vừa sửa dữ liệu

```
Program Tep_co_kieu;
Uses crt;
Var bt: file of byte; i:byte; n:real;
 Begin
 clrscr;
 assign(bt,'TEPCK.DAT');
 rewrite(bt);
 for i:= 0 to 5 do write(bt,i); {ghi vào tệp năm số nguyên}
 reset(bt);
 writeln('  Du lieu luu tru trong tep TEPCK.DAT ');
 while not eof(BT) do   {viết dữ liệu từ tệp ra màn hình}
 begin
 Read(bt,i); write(i:5);
```

```
 end;
 writeln;
 seek(bt,3);  {định vị con trỏ tại phần tử thứ 4}
 textcolor(magenta);
 read(bt,i);   {đọc phần tử thức 4 vào biến i}
 writeln('So cu trong tep o vi tri 3:   ',i);
 i:=33;
 seek(bt,3);
 write(bt,i);  {sửa phần tử thứ 4, giá trị mới là 33}
seek(bt,3); read(bt,i)
 Writeln('So moi trong tep o vi tri 3: ', i);
 writeln('Vi tri hien thoi cua con tro: ',filepos(bt));
 readln;
close(bt);
End.
```

Nhận xét:

Ví dụ 2.18 cho thấy khi ghi hoặc đọc dữ liệu vào tệp có kiểu thì chúng ta luôn luôn phải dùng đến các biến có cùng kiểu với biến tệp.

Đoạn chương trình

```
i:=33;   seek(bt,3);   write(bt,i);
```

dùng để ghi vào phần tử thứ 4 giá trị mới bằng 33. Nếu chúng ta viết lại đoạn lệnh này với ý tưởng ghi trực tiếp giá trị 33 vào phần tử thứ 4

```
seek(bt,3);   write(bt,33);
```

thì Pascal sẽ báo lỗi "Variable Indentifier Expertied" nghĩa là Pascal không hiểu số 33 thuộc kiểu dữ liệu gì.

Ví dụ 2.19

Lập chương trình để nhập điểm cho học sinh và ghi kết quả vào tệp có tên là DIEM.DAT. Chương trình định nghĩa một kiểu dữ liệu mới là Bangdiem (Kiểu Record) với các trường: Stt, Hoten, Diachi, Gioitinh, Lop, Toan, Ly, Hoa. Biến tệp Ds thuộc loại tệp có kiểu. Ví dụ 2.19 chỉ mới làm công việc là nhập dữ liệu vào tệp mà chưa đọc dữ liệu ra.

```
Program Kieutep;
Uses Crt;
Type
Bangdiem = Record
   Stt: Integer;
   Hoten, Diachi: String[25];
   Gioitinh, Lop: string[5];
   Toan,Ly,Hoa: Real;
```

```
End;
Tepdiem = File of Bangdiem;
Var
DS: Tepdiem;
NGUOI: Bangdiem;
 i,j: Integer;  lam,TL: Char;

Begin
Assign(DS,'diem.dat');
Rewrite(DS);
Nguoi.stt:=1;
textbackground(white);
textcolor(5);
lam:='L'; TL:='C'; While lam='L' do
Begin
clrscr;
i:=10; j:=5;
gotoxy(i,j); write(' So thu tu: ');
gotoxy(i,j+1); write(' Ho va ten: ');
gotoxy(i,j+2); write(' Nam hay nu: ');
gotoxy(i,j+3); write(' Thuoc lop: ');
gotoxy(i,j+4); write(' Dia chi: ');
gotoxy(i,j+5); write(' Diem Toan: ');
gotoxy(i,j+6); write(' Diem Ly: ');
gotoxy(i,j+7); write(' Diem Hoa: ');
i:=i+15;
With Nguoi do
Begin
Gotoxy(i,j);     Readln(STT);
Gotoxy(i,j+1); Readln(Hoten);
Gotoxy(i,j+2); Readln(Gioitinh);
Gotoxy(i,j+3); Readln(lop);
Gotoxy(i,j+4); Readln(Diachi);
Gotoxy(i,j+5); Readln(Toan);
Gotoxy(i,j+6); Readln(Ly);
Gotoxy(i,j+7); Readln(Hoa);
End;
Gotoxy(i,j+9); Write('Co ghi vao danh sach khong? C/K ');
Readln(TL);
If upcase(TL) ='C' then
```

```
Begin
Write(DS,NGUOI);
NGUOI.STT:= NGUOI.STT+1;
End;
textbackground(white);
textcolor(blue);
Gotoxy(28,22);
Write(' NHAP TIEP HAY THOI ? L/T ');
Readln(lam);
lam:=upcase(lam);
textbackground(white);
textcolor(5);
End;
Close(DS);
End.
```

Ví dụ 2.20 Nhập dữ liệu quản lý điểm tuyển sinh trung cấp vào tệp, tính tổng điểm và kết quả cho từng người sau đó đọc dữ liệu ra màn hình

Ví dụ 2.20

```
Program Diem_trung_cap;
Uses crt;
 Type dong=record
     mhs:byte;
     hoten:string[12];
     toan,ly,tong:real;
     ketqua:string[7];
     end;
 ds = file of dong;
 Var nguoi:dong; dsl:ds; i,n:byte;
  Begin
  clrscr;
  assign(dsl,'dsl.txt');
  rewrite(dsl);
  write('Nhap bao nhieu nguoi ? '); readln(n);
  for i:= 1 to n do
  Begin
  With nguoi do
    Begin
    write('Ma ho so  ', i); mhs:=i;writeln;
```

```
    write('Ho va ten '); readln(hoten);
    write('Diem toan '); readln(toan);
    write('Diem ly   '); readln(ly);
    tong:=toan+ly;
    if tong>10 then ketqua:='Do' else ketqua:='Truot';
    end;
  write(dsl,nguoi);
  writeln;
  End;
  close(dsl);
  clrscr;
  reset(dsl);
  writeln('       Du lieu luu tru trong tep DSL.TXT ');
  while not eof(dsl) do
  with nguoi do
  begin
  read(dsl,nguoi);
writeln(mhs:3,hoten:15,toan:5:2,ly:5:2,tong:7:2,ketqua:6);
  end;
  readln;
 End.
```

2.4 Tệp không kiểu

Như đã biết tệp văn bản chứa đựng trong nó chỉ các ký tự của bảng mã, tất cả các kiểu dữ liệu khác đều phải chuyển về kiểu này. Tệp có kiểu đòi hỏi khắt khe về kiểu dữ liệu của biến và của tệp. Tệp không kiểu là loại tệp không quan tâm đến kiểu dữ liệu. Sự khác nhau giữa tệp có kiểu và tệp không kiểu là ở chỗ sau khi đã khai báo biến tệp việc mở tệp để ghi hoặc để đọc cần chỉ rõ độ lớn (tính bằng Byte) của từng phần tử tệp. Điều này có nghĩa là giữa các phần tử không có ký hiệu phân cách như thường thấy (tệp văn bản là CR và LF, tệp có kiểu là khoảng cách).

a. Khai báo biến tệp

Để khai báo một biến tệp không kiểu chúng ta dùng mẫu:

Var Bientep: File;

Ví dụ:

Var BT:file;

b. Mở tệp để ghi - đọc

Sau khi đã khai báo biến tệp, muốn mở tệp không kiểu để ghi chúng ta dùng cặp thủ tục:

Assign(bientep, tentep);

Rewrite(Bientep, n);

Còn mở tệp để đọc là các thủ tục

Assign(bientep, tentep);

Reset(Bientep, n);

ở đây n là độ lớn tính theo Byte do người sử dụng quy định để ghi (hoặc đọc) dữ liệu của từng phần tử (Record).

Nếu chọn n=1 nghĩa là mỗi phần tử có độ dài 1 byte thì dữ liệu ghi vào tệp chỉ có thể thuộc kiểu Byte, char, shortint, còn nếu chọn n=2 thì dữ liệu có thể là Integer, word. Với số thực Real phải chọn n=6.

Giả sử mỗi phần tử ghi vào tệp là một chuỗi dài tối đa là 10 ký tự thì phải chọn n=11 bởi vì mỗi chuỗi khi ghi vào bộ nhớ cần thêm một byte để chứa ký tự cho biết độ dài thực của chuỗi.

Nếu bỏ qua tham số n thì Pascal sẽ dùng giá trị ngầm định là 128.

c. Đọc và ghi tệp không định kiểu

**** Đọc tệp không định kiểu***

BlockRead(bientep, biennho, i, j);

Bản thân từ khoá BlockRead cho thấy việc đọc tệp không kiểu là đọc từng khối (block). Khối ở đây được hiểu là một vùng nhớ lưu trữ dữ liệu. Khối có thể là một biến, một chuỗi, một Record hay một mảng.

- bientep: là biến tệp liên kết với tên tệp đã chỉ ra trong thủ tục assign(...)

- biennho: là một biến đã được khai báo cùng kiểu với các phần tử của tệp, biến nhớ đóng vai trò vùng nhớ đệm để lưu trữ dữ liệu đọc từ phần tử của tệp ra. Từ biến nhớ này chúng ta có thể cho dữ liệu hiện trên màn hình hay in ra máy in.

- i là số phần tử quy định cho mỗi lần đọc

- j là một biến kiểu Word dùng để ghi lại số phần tử thực sự đã được đọc. Tham số j có thể bỏ nếu không cần thiết.

- Ghi tệp không định kiểu

*** BlockWrite(bientep, biennho, i);**

Thủ tục BlockWrite chỉ có 3 tham số, ý nghĩa của các tham số này cũng giống như với thủ tục BlockRead

d. Truy nhập tệp không kiểu

Tệp không kiểu cũng được truy nhập như tệp có kiểu nghĩa là cũng dùng thủ tục Seek(bientep,n) để truy nhập vào phần tử thứ n+1 của tệp.

Điều cần đặc biệt lưu ý là với tệp không kiểu là mỗi lần con trỏ tệp dịch chuyển nó sẽ dịch chuyển một số byte đúng bằng số byte đã quy định trong lệnh Rewrite() hoặc Reset()

Ví dụ 2.21

```
Program tep_khong_kieu1;
Uses crt;
```

```
Var bt:file; a:array[1..15] of byte;
   i,j,k:byte; n:word;

   Begin
   Clrscr;
   assign(bt,'tep0kieu.dat');
   rewrite(bt,1); {độ dài từng phần tử là 1 byte}
   For i:= 1 to 15 do
   begin
   a[i] := i;
   blockwrite(bt,a[i],1); {ghi vào tệp 15 số, mỗi lần ghi 1 số}
   end;
   j:=a[2]*a[3];
   for i:= 0 to 14 do
   begin
   seek(bt,i);
   Blockread(bt,k,1);  {đọc từng phần tử của tệp vào biến K}
   textcolor(red);
   write(k,' ');
   end;
   writeln;
   seek(bt,filesize(bt)-1); {chuyển con trỏ đến phần tử cuối}
   blockwrite(bt,j,1);  {sửa giá trị phần tử cuối}
   seek(bt,filesize(bt)-1);
   blockread(bt,k,1);  {đọc lại phần tử cuối}
   write(k);   { viết phần tử cuối lên màn hình}
   close(bt);
   readln;
   end.
```

Ví dụ 2.21 tạo một biến tệp không kiểu là BT, biến tệp này kết nối với một tệp lưu trữ dữ liệu Tep0kieu.dat.

Đầu tiên chương trình ghi lên tệp 15 số nguyên (1 byte), sau đó đọc dữ liệu từ tệp và viết chúng ra màn hình (phần tử cuối là 15). Tiếp đó sửa dữ liệu ở phần tử cuối và lại viết phần tử đó ra màn hình (bây giờ là 6).

Ví dụ 2.22

Chương trình dưới đây tạo tệp không kiểu BT, các phần tử tệp là chuỗi có độ dài bằng 3, lệnh mở tệp để ghi phải quy định độ dài phần tử bằng 4 vì trong bộ nhớ string[3] sẽ chiếm 4 bytes.

```
Program tep_khong_kieu2;
Uses crt;
Var bt:file; a:array[1..30] of string[3];
   i,j:byte; k:string[3];

   Begin
   Clrscr;
   assign(bt,'tep0kieu.dat');
   rewrite(bt,4); { độ lớn của mỗi phần tử là 4 bytes}
   For i:= 1 to 24 do
   begin
   a[i] := chr(i+64)+chr(i+65)+chr(i+66);
   blockwrite(bt,a[i],1);
   end;
   reset(bt,4);
   for i:= 0 to 23 do
   begin
   seek(bt,i);
   Blockread(bt,k,1);
   textcolor(green);
   write(k,' ');
   end;
   close(bt);
   readln;
   end.
```

Ví dụ 2.23

Nhập vào tệp các phần tử là Record và sau đó viết chúng ra màn hình. Trong phần khai báo Record chúng ta chọn Hoten là string[15] và Diem thuộc kiểu Real như vậy độ dài của mỗi phần tử phải là 15+1+6=22.

```
Program tep_khong_kieu4;
Uses crt;
Type hs = record
        hoten:string[15];
        diem:real;
        end;
Var
   bt:file; k,nguoi:hs;  i,j:byte;
```

```
Begin
Clrscr;
assign(bt,'tep0kieu.dat');
rewrite(bt,22);
write('Nhap bao nhieu nguoi? ');
readln(n);
  For i:= 1 to n  do
 with nguoi do
 Begin
 write('Ho va ten : '); readln(hoten);
 Write('Diem tong : '); readln(diem);
 blockwrite(bt,nguoi,1);
end;
for i:= 0 to n-1 do
begin
seek(bt,i);
Blockread(bt,k,1);
textcolor(red);
with k do
writeln(hoten,'  ',diem:5:2);
end;
close(bt);
readln;
end.
```

Để tìm hiểu thêm về tệp không kiểu chúng ta có thể mở Help của Pascal và xem trong đó ví dụ về việc dùng tệp không kiểu để sao chép dữ liệu từ tệp này sang tệp khác.

3. Dữ liệu kiểu tập hợp

3.1 Khái niệm tập hợp

a. Khái niệm tập hợp

Một tập hợp bao gồm n phần tử cùng kiểu dữ liệu (0 <= n <= 255), kiểu của các phần tử phải là kiểu vô hướng đếm được (nguyên, ký tự, logic, liệt kê, đoạn con). Số thứ tự các phần tử trong tập hợp luôn nằm trong khoảng 0 - 255.

b. Khai báo kiểu và gán dữ liệu vào tập hợp

Để mô tả kiểu tập hợp, Pascal dùng từ khoá **Set of** tiếp đó là ***kiểu dữ liệu*** cơ bản của các phần tử tạo nên tập hợp. Cách viết kiểu dữ liệu này phải tuân theo những quy định mà Pascal đã thiết kế:

* Với Pascal 7.0 số phần tử không quá 256 và số thứ tự của các phần tử phải tăng dần

* Kiểu dữ liệu cơ bản của các phần tử của tập hợp có thể viết tường minh hoặc thông qua các giá trị cụ thể. Ví dụ :

```
Type
  a = set of Char;
  b =  set of Byte;
  c = set of 1..100;
  d = set of  'a' .. 'z';
  Maytinh =  set of  (Compact, Mitac, IBM, CMS);
```

Trong ví dụ trên kiểu dữ liệu của các phần tử của tập hợp a, b được khai báo tường minh, a là tập các ký tự, b là tập các số nguyên. Kiểu dữ liệu các phần tử của tập c và d được xác lập qua các giá trị viết sau SET OF, tập c sẽ là tập các số nguyên trong khoảng từ 1 đến 100, tập d là tập các chứa các chữ cái thường còn Maytinh là tập hợp của 4 phần tử đã liệt kê.

Để thấy rõ hơn chúng ta xét ví dụ sau:

Ví dụ 2.24

```
      Program Taphop1;
Uses crt;
Type
  mau= set of (xanh, hong, tim, vang, den, nau);
  sn1 = set of 0..100;
  sn2 = 100..200;
  chucai1 = set of 'a'..'z';
  chucai2 = set of 'A'..'Z';
  Kytu = set of Char;
Var
  a,b:mau; b1:sn1;
  b2 : set of sn2;  c :chucai1;   d:chucai2;   e:kytu;
  i,j,k:word;
  tl : set of boolean;
BEGIN
  clrscr;
  i:=300; j:=160; k:=i+j;
  a:=[xanh.. vang];
  b:=[tim,nau];
  b1:=[15,255,i+j,i*j];
  b2:=[1, 200, 155, 123, 145];
  c:=['z','à','k'];
```

```
  d:=[‘2’, ‘3’,'4'];
  tl:=[true,false];
  readln;
END.
```

Ví dụ 2.24 khai báo năm kiểu tập hợp là Mau, Sn1, Chucai1, Chucai2, Kytu còn Sn2 là kiểu dữ liệu đoạn con. Biến a thuộc kiểu tập hợp MAU, biến B1 và B2 thuộc kiểu tập hợp số nguyên, biến c và d thuộc kiểu tập hợp chữ cái, biến e thuộc kiểu ký tự còn biến Tl thuộc kiểu boolean. Các phép gán trong ví dụ sẽ xác định một số tập hợp, chẳng hạn câu lệnh

a:=[xanh, hong, den, tim];

xác định tập hợp a gồm các phần tử xanh, hong, den, tim.

e:=['0'..'9', 'a'..'z'];

xác định tập hợp e gồm 10 ký tự 0, 1,...9 và 26 ký tự 'a', 'b', ... 'z'

Khi khai báo kiểu tập hợp cần chú ý một số điều sau đây:

*** *Thứ nhất***: Thứ tự các phần tử trong tập hợp luôn theo chiều tăng dần, nếu chúng ta khai báo:

sn1 = set of 200..100;

hoặc

chucai = set of 'z' .. 'a';

thì Pascal sẽ thông báo lỗi: Lower bound greater than upper bound.

*** *Thứ hai***: Kiểu dữ liệu của các phần tử mặc dù có thể là số nguyên song khi khai báo lại chỉ có thể là kiểu Byte.

Lệnh SN0 = set of Byte; là đúng

Các lệnh khai báo sau là sai:

SN1 = set of Word;

SN2 = set of Integer;

SN3 = set of Shortint;

Lỗi mà Pascal thông báo là:

Error 23: Set base type out of Range

*** *Thứ ba:*** Những gía trị mà chúng ta khai báo sau từ khoá SET OF sẽ dẫn tới một trong hai khả năng:

- Nếu là khai báo kiểu liệt kê, kiểu Boolean thì đây chính là các giá trị mà tập hợp có thể nhận. Trở lại ví dụ 2.24 các phép gán

a:=[xanh, tim, luc];

tl:=[tru, fal];

sẽ bị báo lỗi vì phần tử "luc" không có trong danh sách liệt kê của màu, còn Tru và Fal không có trong kiểu boolean.

- Với những kiểu còn lại (nguyên, ký tự, đoạn con) những khai báo sau từ khoá Set of không phải là *giá trị của các phần tử* của tập hợp mà là *thứ tự* của các phần tử đó, *Kiểu của các dữ liệu đó* sẽ cho biết các phần tử của tập hợp sẽ phải thuộc kiểu gì

Ví dụ: 2.25

```
Type
sn1 = set of  0..100;   {các phần tử  tập hợp phải là số nguyên}
kytu = set of char;    {các phần tử tập hợp phải thuộc kiểu ký tự}
Var
  a : sn1;   b : kytu;
.....
```

Với khai báo như trong ví dụ 2.25 chúng ta không thể dùng phép gán:

```
b := ['a', 12, 'c'];  ( các phần tử của tập b phải là số )
c:= [1, 2, 100];   ( các phần tử của tập c phải là ký tự )
```

**** Thứ tư***: Không được gán trực tiếp các số nguyên có giá trị lớn hơn 255, ví dụ phép gán

```
b2:=[100, 200, 256]; sẽ bị báo lỗi  Error 76: Constant out of Range
```

* Thứ tư: Các phần tử trong tập hợp có thể là hằng, biến hay biểu thức. Nếu là tập số nguyên thì giá trị các biến hay biểu thức trong phép gán có thể lớn hơn 255

Trong ví dụ 2.25 phép gán sau không bị báo lỗi

```
i:=300; j:=160; k:=i+j;
 b1:=[15,255,i+j,i*j];
```

3.2 Phân loại tập hợp

a. Tập hợp cùng kiểu

Các tập hợp được coi là cùng kiểu nếu chúng được xây dựng trên cùng một kiểu cơ bản.

Giả sử chúng ta khai báo các tập hợp như v í d ụ sau:

Ví dụ 2.26

```
Type
  mau= set of (xanh, hong, tim, vang, den, nau);
  mau_xe = set of (xanh..vang);
  sn1 = set of 0..100;
  sn2 = 100..200;
  sn3 = set of Byte;
  chucai1 = set of 'a'..'z';
  chucai2 = set of 'A'..'Z';
```

```
  Kytu = set of Char;
Var
  a:sn1; b:sn2; c:sn3; mau_oto:mau; chu:kytu;
  .....
```

thì các nhóm tập hợp sau đ ây là cùng kiểu

mau, mau_xe

sn1, sn2, sn3

chucai1, kytu

chucai2, kytu

b. Hình thành một tập hợp

Một tập hợp được hình thành khi chúng ta gán cho *biến tập hợp* các giá trị cụ thể, các giá trị này được đặt trong hai dấu ngoặc vuông.

Ví dụ 2.27

```
c := [2, 15, 30, 100..200];
mau_oto := [xanh, den, vang, tim];
a := [];
chu := ['A'..'Z'];
```

Nếu một tập hợp không chứa một phần tử nào cả thì nó được gọi là **tập rỗng** và được ký hiệu là []. Trong ví dụ 2.28, a là một biến tập hợp rỗng.

Tập hợp rỗng được xem là cùng kiểu với mọi kiểu tập hợp.

3.3 Các phép tính trên tập hợp

Các phép tính giới thiệu sau đây chỉ thực hiện được trên các tập hợp cùng kiểu, nghĩa là các phần tử của mọi tập hợp phải cùng thuộc một kiểu cơ bản.

a. Phép gán

Phép gán được dùng để hình thành nên một tập hợp. Có thể gán một tập hợp cho một biến tập hợp cùng kiểu hoặc gán một biến tập hợp cho một biến tập khác cùng kiểu. (xem ví dụ 2.28)

b. Phép hợp

Phép hợp được ký hiệu bởi toán tử " + ". Hợp của hai tập hợp A và B khi đó có thể viết là A + B

Hợp của hai tập A và B là một tập hợp gồm các phần tử thuộc A hoặc thuộc B.

Giả sử các tập A, B, C được hình thành như sau:

```
A := [1..8,10..15];
```

B := [5..9, 20,30];

C := A + B;

Tập C được hình thành bởi phép hợp A và B nên sẽ gồm các phần tử [1..15,20,30]

c. Phép giao

Giao của hai tập A và B ký hiệu là A*B là một tập hợp gồm các phần tử đồng thời thuộc cả A và B.

C := A*B; tập C sẽ có các phần tử [5, 6, 7, 8]

d. Phép trừ

Phép trừ hai tập hợp A và B ký hiệu là A - B cho ta một tập gồm các phần tử thuộc A nhưng không thuộc B

C := A - B; tập C sẽ có các phần tử [1..4, 10..15]

còn [1..5] - [1..10] = [ ]

e. Phép thuộc về

Phép thuộc về ký hiệu là IN là một phép thử xem một giá trị hay một biến (không phải là biến tập) có thuộc về một tập khác hay không? Nếu có giá trị phép thử là True, ngược lại phép thử cho kết quả False.

Ví dụ 2.28

T1:= ['a'..'z'];

T2:= [true,false];

T3 := 'c';

Write(T3 in T1); cho kết quả True

Write(false in T2); cho kết quả True

Wite('1' in T1); cho kết quả False

3.4. Các phép so sánh trên tập

Các phép so sánh nói trong mục này chỉ được thực hiện trên các tập cùng kiểu. Kết quả của phép so sánh thuộc kiểu Boolean nghĩa là cho ra một trong hai giá trị True hoặc False.

a. Phép bằng, ký hiệu là =

Hai tập A và B gọi là bằng nhau nếu mọi phần tử của A đều thuộc B và mọi phần tử của B đều thuộc A không phụ thuộc vào thứ tự các phần tử.

Ví dụ phép so sánh

[1..6] = [1,2,4,5,3,6] cho kết quả True

['a', 'b', 'c'] = ['A', 'B', 'C'] cho kết quả False

b. Phép không bằng, ký hiệu là <>

Hai tập A và B gọi là không bằng nhau nếu tập A có ít nhất một phần tử không thuộc tập B hoặc tập B có ít nhất một phần tử không thuộc tập A.

[1..6] <> [1,2,4,5,3,6] cho kết quả False

['a', 'b', 'c'] <> ['A', 'B', 'C'] cho kết quả True

c. Phép nhỏ hơn hoặc bằng, ký hiệu là <=

Tập A gọi là nhỏ hơn hoặc bằng tập B nếu mọi phần tử của tập A đều thuộc tập B.

[1..6] <= [1,2,4,5,3,6,8,9] cho kết quả True

[1..6] <= [1,2,4,5,3] cho kết quả False

d. Phép lớn hơn hoặc bằng, ký hiệu là >=

Tập A gọi là lớn hơn hoặc bằng tập B nếu mọi phần tử của tập B đều thuộc tập A.

[1..6] >= [1,2,4,5] cho kết quả True

[1,5] >= [1,2,4,5] cho kết quả False

Chú ý: ***Trong Pascal không tồn tại các phép so sánh nhỏ hơn < và lớn hơn >***

Các ví dụ về dữ liệu kiểu tập hợp

Ví dụ 2.29 : Tìm số nguyên tố

Ta đã biết số nguyên tố là các số chỉ chia hết cho 1 và chính nó (ngoại trừ số 1) để tìm các số nguyên tố trong khoảng từ 1 đến n chúng ta sử dụng nguyên lý sàng Eratosthène. Nguyên lý này có thể phát biểu như sau: nếu a là một số nguyên tố trong tập [1..n] thì không thể tồn tại một số b thuộc tập [1..n] mà b là bội số của a. Nói cách khác nếu ta đã tìm được a là một số nguyên tố thì ta phải loại trừ khỏi tập [1..n] tất cả các số là bội số của a.

```
Program Tim_so_nguyen_to;
Uses crt;
Const n=255;
Type Songuyen = 1..n;     {định nghĩa kiểu dữ liệu Songuyen gồm n số}
Var  snt,sang: set of songuyen;
   so:songuyen; i:integer;
Begin
   clrscr;
   Writeln('Cac so nguyen to tu 2 den ',n);
   snt:=[];{Tập snt ban đầu là tập rỗng}
   sang:=[2..n]; {Xác lập tập Sang gồm số nguyên tố đầu tiên (2) và các số khác}
   so:=2; {Gán vào biến So  số nguyên tố đầu tiên = 2}
Repeat
   While not (so in sang) do so:=so+1;
   snt:=snt+[so]; {Bổ xung số nguyên tố vừa tìm được vào tập Snt}
```

```
    i:=so;
     Write(so,'  '); {Viết số nguyên tố vừa tìm được ra màn hình}
    While i<=n do
      Begin
      sang:=sang-[i]; i:=i+so;  {Xoá các số là bội số của số nguyên tố vừa tìm được}
      End;
Until sang = [];
Readln;
End.
```

Ví dụ 2.30

Trò chơi gieo xúc xắc: nguyên tắc chơi như sau: người chơi chọn một số trong khoảng từ 1 đến 6, máy sẽ gieo 3 con xúc xắc để được 3 số. Nếu số của người chơi nằm trong tập số mà máy đã gieo thì người chơi thắng, ngược lại người chơi thua. Nếu ba lần chơi mà người chơi đều thắng thì tuyên bố người chơi trúng số độc đắc.

```
Program xuc_xac;
Uses crt;
Var xucxac: set of 1..6;
   So,i,dem,a,b,c:1..6;tl:char;
Begin
Clrscr;
Repeat
clrscr;
For i:= 1 to 3 do
 Begin
   Textcolor(5);
   Write('Moi ban chon mot so tu 1 den 6 '); readln(so);
   a:=random(6); b:=random(6); c:=random(6);
   xucxac:=[a,b,c];
   Writeln('So ma ban da chon ',so);
   Writeln('So ma may da gieo ',a,' ',b,' ',c);
   If so in xucxac then
     Begin
     textcolor(14);
     writeln('Ban thang van thu ',i);
```

```
      dem:=dem+1;
      End
      else
      Writeln('Ban thua van thu ',i);
    If dem = 3 then Writeln('Ban trung doc dac - Xin chuc mung');
   End;
Writeln;
Write('Choi tiep hay thoi? C/K ');
Readln(tl);
Until upcase(tl)='K';
End.
```

Bài tập ứng dụng chương 2

1. Nhập vào thư viện n đầu sách (n<=50), các thông số cần đưa vào là: Tên sách, Tên tác giả, Năm xuất bản, Tên nhà xuất bản, Số trang, Giá tiền. Hiện danh mục sách đã nhập lên màn hình.

2. Dữ liệu tuyển sinh gồm các trường: SBD, HOTEN, TOAN, LY, HOA, TONG, KETQUA.

Lập chương trình nhập dữ liệu cho một phòng thi không quá 40 thí sinh, số liệu trong trường Tong = Toan+Ly+Hoa, trường Ketqua điền vào chữ "DO" nếu tổng >=20, còn lại là "TRUOT".

Hiện lên màn hình những người DO

3. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số chứng từ.

Lập chương trình nhập hàng vào kho, sau mỗi mặt hàng nhập lại hỏi: "Nhap tiep hay thoi? C/K ". Nếu bấm "K" thì kết thúc nhập và hiện danh mục hàng đã nhập lên màn hình.

4. TOADO là kiểu bản ghi chứa toạ độ (cột,dòng) của ba điểm trên màn hình. Lập chương trình kiểm tra xem ba điểm này có thẳng hàng không, nếu không thẳng hàng thì tạo nên tam giác gì (Vuông, Cân, ...)

5. Dữ liệu quản lý vé xe lửa gồm: Số toa, số ghế, giá vé, ga đi, ga đến, Đã bán vé chưa. Mỗi toa tầu có 50 ghế đánh số thứ tự từ 1 đến 50.

Lập chương trình nhập vào những ghế đã bán vé, thông báo còn bao nhiêu chỗ chưa bán vé.

6. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số chứng từ. Ngày nhập bao gồm Ngay, Thang, Nam.

Lập chương trình nhập hàng vào kho, kết thúc nhập khi mã hàng bằng dấu "*" .

Cho một tháng nào đó, hiện những hàng đã nhập trong tháng đó.

7. Dữ liệu quản lý đất đai bao gồm hai trường Chủ đất và Đất. Chủ đất bao gồm: Mã hồ sơ, Họ tên, Mã đất, Địa chỉ. Đất bao gồm: Mã đất, Diện tích, Loại đất, Mục đích sử dụng.

Nếu mục đích sử dụng là trồng trọt thì có thuế nông nghiệp

Nếu mục đích sử dụng là công ích (trường học, bệnh viện, sân vận động) thì không phải nộp thuế.

Nếu mục đích sử dụng là kinh doanh bất động sản thì có thuế đất và thuế môn bài.

Nhập dữ liệu cho n người (n<=30) sau đó hiện lên màn hình 10 người sử dụng nhiều đất nhất.

8. Viết chương trình nhập vào tệp văn bản F một số dòng văn bản, đếm xem trong tệp các ký tự a,b,....z mỗi ký tự xuất hiên bao nhiêu lần?

9. Thông tin về thí sinh thi đại học bao gồm Hoten, namsinh, diemthi, ketqua. Thông tin trên được ghi trong một tệp văn bản, mỗi trường trên một dòng. Lập chương trình in ra màn hình mỗi thí sinh trên một dòng

Ví dụ:

Trong tệp	In ra màn hình
Tran Tam 1980 8 Kha	Tran Tam 1980 8 Kha

10. Dữ liệu quản lý hàng hoá bao gồm: Mã hàng, Tên hàng, Số lượng, Đơn giá, Ngày nhập, Số hoá đơn. Lập chương trình nhập dữ liệu từ bàn phím và ghi vào tệp có kiểu. Cho biết một số hoá đơn, hiện lên màn hình số liệu ứng với hoá đơn đó.

11. T1 và T2 là các tệp văn bản, nội dung T1, T2 nhập từ bàn phím, hãy nối T1 với T2 thành T3, hiện T3 lên màn hình.

12. T1 là tệp văn bản chứa 15 ký tự dạng số Integer. Lập chương trình chuyển T1 thành tệp số nguyên T2, tính tổng các số và viết kết quả lên màn hình.

13. Giả sử thư mục hiện hành là C:\TP\BIN. Cho biết một tên tệp từ bàn phím, kiểm tra xem trong thư mục hiện hành có tệp đó chưa, nếu có thì hiện nội dung tệp, nếu chưa thì mở tệp mới để ghi dữ liệu.

14. Hai tập A và B gọi là liên thông nếu chúng có ít nhất một phần tử chung. Nếu các cặp tập hợp sau là liên thông (A,B), (B,C), (C,D), (D,E) thì ta nói có một đường nối từ A đến E.

Lập chương trình tìm đường nối giữa n điểm A_1, A_2, ... An trên màn hình biết rằng toạ độ giữa các các điểm lần lượt là (x_1, y_1), (x_2, y_2), ... (xn, yn). In ra tất cả đường nối theo mẫu:

A_1 -> A_2 ->... -> An

Nếu giữa các cặp điểm Ai, Aj không có đường nối thì trả lời không có.

Ví dụ:

$A_1(1,2)$, $A_2(2,3)$,$A_3(3,4)$, $A_4(4,1)$, $A_5(5,6)$

A_1 -> A_5 không có đường nối

A_1 -> A_2 có hai đường nối

A_1 -> A_2

A_1 -> A_4 -> A_3 ->A_2

Chương 3

Đơn vị chương và trình Thư viện chuẩn

Khái niệm Đơn vị chương trình giống như một cuốn sách và Thư viện chuẩn giống như một tủ sách. Thư viện chuẩn của Pascal không chứa hết mọi đơn vị chương trình, lý do là vì tiết kiệm bộ nhớ. Chương này bạn đọc nên dành nhiều thời gian cho việc khai thác các Unit đã thiết kế trong Pascal, tuy nhiên đã là một người lập trình thì không thể không biết tự tạo ra các Unit. Những phần người đọc cần nắm được là:

- Các thủ tục hoặc hàm có trong các Unit System,Graph, Crt, Dos..
- Phương pháp xây dựng các Unit
- Những nguyên tắc cần tuân thủ khi tham chiếu đến các Unit
- Biên dịch Unit từ chương trình nguồn sang dạng mã máy (đuôi tệp là TPU)

1. Khái niệm đơn vị chương trình (Unit)

Thuật ngữ Unit trong ngôn ngữ lập trình Pascal được dịch là "Đơn vị chương trình". Mỗi Unit được xem như một modul nhỏ chứa đựng một số công cụ cần thiết giúp cho người lập trình có thể dễ dàng thiết kế chương trình. Các Unit có thể gộp chung lại thành thư viện chương trình hoặc để phân tán trong một thư mục quy ước của Pascal .

Lệnh tham chiếu đến Unit được đặt ở đầu chương trình với cú pháp:

USES tênUnit;

Ví dụ: USES CRT, GRAPH;

Các Unit được tổ chức trong Pascal dưới hai dạng:

* Các file độc lập với phần mở rộng là TPU (Turbo Pascal Unit), ví dụ GRAPH.TPU

* File thư viện chuẩn với phần mở rộng TPL (Turbo Pascal Library), ví dụ TURBO.TPL

Thư viện chuẩn của Pascal chứa đựng một số Unit cơ bản, hay được dùng đến, chúng được đóng gói lại và được để cùng chỗ với tệp khởi động TURBO.EXE. Thông thường chương trình Pascal được cài đặt trên ổ đĩa C, với Pascal 7.0 đường dẫn đến nơi lưu trữ các tệp TURBO.EXE và TURBO.TPL là:

C:\TP\BIN

Khi tệp TURBO được gọi, nghĩa là chương trình Pascal được khởi động thì tệp TURBO.TPL cũng tự động được tải vào bộ nhớ. Lúc này các Unit chứa trong thư viện chuẩn sẽ sẵn sàng được tham chiếu đến. Việc truy cập đến các Unit trong thư viện chuẩn nhanh hơn so với truy cập vào các Unit độc lập vì chúng đã thường trú trong bộ nhớ. Khi có lời gọi một Unit nào đó bao giờ Pascal cũng ưu tiên tìm kiếm chúng trong thư viện chuẩn, nếu không tìm thấy thì tiếp tục tìm kiếm ở bên ngoài.

2. Thư viện chuẩn

Thư viện chuẩn của Pascal có tên là TURBO.TPL, thư viện này ngầm định ban đầu chứa năm Unit là:

2.1 Crt

CRT bao gồm các thủ tục quản lý màn hình, bàn phím và âm thanh, nhờ có Unit này mà người lập trình có thể thiết kế giao diện chương trình tương đối đẹp

2.2 Dos

Unit này chứa các chức năng quản lý tệp, đĩa, ngày tháng. Ngoài ra cũng có thể dùng Unit này để gọi trực tiếp các lệnh của hệ điều hành DOS hoặc các ngắt hệ thống.

2.3 Overlay

Unit này được sử dụng khi chương trình nguồn có dung lượng lớn. Mặc dù đã được biên dịch sang dạng mã máy song đôi khi do bộ nhớ trong (RAM) không đủ nên không thể tải cùng một lúc toàn bộ Code của chương trình nguồn. Sử dụng OVERLAY có thể tải từng phần

của chương trình nguồn vào bộ nhớ trong để chạy. Với những máy PC có Ram 128MB, hoặc 256 MB thì không mấy khi phải sử dụng Unit này.

2.4 Printer

Unit này định nghĩa tên máy in là LST. Việc kết xuất thông tin bằng lệnh Write khi tham chiếu đến LST sẽ cho phép ta in ra máy in các kết quả bài toán.

2.5 System

Đây là Unit cơ bản và quan trọng nhất của Pascal, nó chứa các thủ tục vào ra như Read, Write…, các hàm sơ cấp thông dụng như Ln, Sqrt, Sin, Cos… Khi Pascal được khởi động và thư viện chuẩn được nạp vào bộ nhớ thì Unit này cũng tự động liên kết với mọi chương trình vì thế ở đầu chương trình không cần đến lời gọi USES SYSTEM.

Ngoài những Unit có sẵn trong thư viện chuẩn, Pascal còn cho phép người lập trình thiết kế thêm các Unit mới, các Unit này sau đó sẽ được lưu vào tệp Turbo.tpl và được xem như một bộ phận cấu thành của ngôn ngữ. Việc chuyển các Unit của người lập trình vào thư viện chuẩn được thực hiện thông qua trình tiện ích TPUMOVER sẽ đề cập đến ở mục sau.

3. Các Unit khác

Pascal có khoảng 20 Unit độc lập lưu trữ trong thư mục TP\UNIT. Các Unit này được phân chia thành 4 nhóm gồm:

* Interface Unit: nhóm Unit chung cho mọi ứng dụng

* Objects Unit: danh mục (theo vần a,b,c..) của các Unit thuộc kiểu Turbo vision

* System Unit: Unit hệ thống

* Turbo vision Unit: Unit phục vụ cho lập trình hướng đối tượng

Muốn tìm hiểu về một Unit nào đó ta có thể đọc phần giới thiệu Unit trong các tệp cùng tên với phần mở rộng là INT, các tệp này lưu trữ trong thư mục TP\DOC. Các tệp đuôi INT này là tệp văn bản nên có thể đọc chúng bằng bất kỳ phần mềm soạn thảo văn bản nào ví dụ Word, NC… Cũng xin lưu ý thêm là một số Unit mặc dù có tên trong thư mục TP\UNIT như GRAPH3.TPU, TURBO3.TPU nhưng do đã quá lỗi thời nên không được giới thiệu trong thư mục TP\DOC. Dưới đây là một vài Unit thông dụng

3.1 Graph

Unit GRAPH được lưu trữ trong một File riêng có tên là GRAPH.TPU. Unit này chứa các thủ tục và hàm cho phép lựa chọn loại màn hình, số dòng, cột và các thủ tục vẽ các hình cơ bản, tô màu nét vẽ hoặc màu nền các hình khép kín…

3.2 Memory

Unit Memory chứa các phục vụ quản lý vùng nhớ cấp phát cho chương trình, các chương trình với phần mở rộng EXE thì vùng nhớ cơ sở được cấp phát là 640 Kb. Đoạn thấp nhất của vùng nhớ này (256 byte) dành cho việc đánh địa chỉ các biến, hàm, thủ tục trong chương trình.

3.3 Menus

Unit Menus chứa các phục vụ thiết kế thực đơn như chọn phím nóng (Hot key), tạo dải thực đơn (Menu Bar), khung thực đơn (menu box)…

3.4 Views

Unit Views chứa các thủ tục và hàm liên quan đến việc quan sát các khung hoặc cửa sổ như thanh cuốn (Scroll Bar), con trỏ (Cursor), chuyển trang (Nextview, Prevview)…

4. Xây dựng các Unit

Các Unit cũng được viết và lưu vào thư mục ngầm định TP\BIN như các chương trình Pascal khác. Tệp lệnh nguồn của Unit sẽ có phần mở rộng là PAS. Để soạn thảo một Unit chúng ta có thể dùng một phần mềm soạn thảo văn bản bất kỳ nhưng nói chung người ta vẫn hay dùng công cụ soạn thảo (Text Editor) của Pascal. Mặc dù được lưu dưới dạng một tệp đuôi PAS nhưng các Unit không thể chạy bằng tổ hợp lệnh Ctrl – F9 như các chương trình bình thường.

Các chương trình Pascal khi biên dịch sang dạng mã máy sẽ có phần mở rộng là EXE (Execute), còn các Unit khi biên dịch sẽ có phần mở rộng là TPU (Turbo Pascal Unit).

Cấu trúc một Unit bao gồm bốn phần cơ bản sau đây:

4.1 Phần tiêu đề

Phần này chỉ gồm một dòng mở đầu là từ khoá UNIT sau đó là tên Unit, kết thúc dòng bằng dấu chấm phảy.

Unit tenUnit;

Tên Unit phải viết cách từ khoá Unit ít nhất một khoảng trống, các ký tự của tên phải viết liền nhau theo các quy định giống như tên chương trình.

Cần phải đặc biệt lưu ý rằng Pascal quy định bắt buộc tên Unit sau này sẽ được dùng làm tên tệp để lưu trữ tệp nguồn của Unit (đuôi Pas) đồng thời cũng là tên của đơn vị chương trình (đuôi Tpu) do vậy khi đặt tên không nên chọn tên quá dài, nên chọn các tên gợi nhớ đến nội dung của Unit để tiện sử dụng về sau.

4.2 Phần khai báo chung

Phần này bắt đầu bằng từ khoá INTERFACE

Bản thân từ khoá Interface có nghĩa là dùng chung, điều này quy ước rằng ở đây chúng ta phải khai báo các kiểu dữ liệu mới, các biến, hằng, hàm, thủ tục mà sau này các chương trình tham chiếu đến Unit sẽ sử dụng. Có thể hình dung rằng những gì viết trong phần này sẽ là những tài nguyên mà Unit có thể cung cấp cho các chương trình ngoài, nó giống như bản thực đơn mà nhà hàng trình bày cho thực khách.

Ví dụ chúng ta xây dựng một Unit lấy tên là HHP (hình học phẳng) trong đó có các hàm tính diện tích , chu vi các hình chữ nhật, tam giác, tròn là dtcn, dttg, dttr, cvcn, cvtg, cvtr. Khi đó phần khai báo chung sẽ là:

```
INTERFACE
Function dtcn( a,b: real) : real;
Function dttg( a,b,c: real) : real;
Function dttr( a: real) : real;
Function cvcn( a,b: real) : real;
Function cvtg( a,b,c: real) : real;
Function cvtr( a: real) : real;
```

Ví dụ trên khai báo ba hàm tính diên tích và ba hàm tính chu vi, sau này khi các chương trình tính toán tham chiếu đến Unit hhp thông qua lệnh USES HHP sẽ có thể dùng trực tiếp các hàm này. Các tham số khai báo trong cả sáu hàm đều là các tham trị nghĩa là giá trị của chúng sẽ không biến đổi khi quay về làm việc với chương trình chính. Việc sử dụng tham số là tham trị hay tham biến sẽ tuỳ thuộc vào các bài toán cụ thể.

4.3 Phần nội dung

Phần nội dung bắt đầu bằng từ khoá IMPLEMENTATION. Tại đây chúng ta sẽ xây dựng nên các hàm, thủ tục mà tên của chúng đã được giới thiệu ở phần Interface. Việc xây dựng các chương trình con đã được giới thiệu ở phần lập trình cơ bản nên chúng ta không đề cập đến ở đây.

Điều cần nhấn mạnh là những hàm và thủ tục chúng ta đã giới thiệu ở Interface thì bắt buộc phải được xây dựng ở đây bởi lẽ chúng chính là những gì mà các chương trình tham chiếu đến Unit cần sử dụng. Ngược lại trong phần này chúng ta có thể xây dụng các hàm và thủ tục khác hoặc khai báo các biến, hằng mới không có trong phần Interface, chúng được xem là phần riêng của Unit mà các chương trình tham chiếu đến Unit không được phép sử dụng. Ví dụ:

```
IMPLEMENTATION
Function dtcn( a,b: real) : real;
Var s : real;
Begin
S:=a*b;  Dtcn:=s;
End;
...
Function cvcn( a,b: real) : real;
Var s : real;
Begin
S:=a*b;  Dtcn:=s;
End;
```

4.4 Phần khởi động

Phần khởi động đặt giữa hai từ khoá BEGIN và END, sau End là dấu chấm giống như các chương trình Pascal bình thường. Trong phần này chúng ta có thể đưa vào các lệnh gán giá trị ban đầu cho biến hoặc lời gọi các thủ tục riêng nào đó cần cho quá trình tạo hàm và thủ tục của phần IMPLEMENTATION .

Phần khởi động là không bắt buộc phải có, trong trường hợp không có phần này thì chúng ta bỏ đi từ khoá BEGIN song vẫn phải có từ khoá END. để báo hiệu kết thúc Unit. Dưới đây là cấu trúc tổng thể của một Unit.

```
Unit  Tên_Unit ;
Interface .....
Uses  Tên_Unit; (tên các Unit  sẽ dùng trong các chương trình con của Unit này)
Const .... (khai báo hằng)
Type ...   (khai báo kiểu dữ liệu)
Var ...     (khai báo biến cho các chương trình con trong Unit )
Tên các Procedure và Function của Unit

Implementation
(nội dung của từng chương trình con)

Begin
Phần khởi tạo giá trị ban đầu (tuỳ chọn)
End.
```

Về bản chất Unit cũng là một chương trình của Pascal, nó được xây dựng trên cơ sở các từ khoá và từ vựng mà Pascal đã thiết kế do vậy từ bên trong Unit chúng ta lại có thể tham chiếu đến các Unit khác không phân biệt là Unit chuẩn của Pascal hay Unit do người sử dụng tạo ra.

Ví dụ 3.1 Xây dựng Unit HHP chứa hàm tính diện tích hình chữ nhật.

```
Unit HHP;
Interface
Function dtcn( a,b: real) : real;
Implementation
Function dtcn( a,b: real) : real;
Uses crt, dos;
Var s : real;
Begin
     S:=a*b;
     Dtcn:=s;
End;
```

Sau khi đã soạn thảo song Unit chúng ta cần ghi Unit thành tệp với ***tên tệp trùng với tên đã chọn ở phần tiêu đề***. Các Unit lúc này mới chỉ ở dạng tệp văn bản nên khi ghi vào đĩa Pascal sẽ gán phần đuôi là .Pas. Các chương trình Pascal có thể tham chiếu đến các Unit này nhưng Pascal sẽ phải làm công việc biên dịch trước do đó sẽ mất một thời gian. Tốt nhất để có thể sử dụng các Unit chúng ta phải dịch chúng sang dạng mã máy thông qua chức năng Compile trên thực đơn chính của Pascal . Trước khi dịch cần chú ý rằng tuỳ chọn Destination của thực đơn Compile phải là Disk chứ không phải là Memory. Có thể dùng một trong ba cách dịch đã biết trong chức năng này là Make, Build, hoặc Run tuy nhiên cách hay được dùng là từ màn hình soạn thảo Unit, bấm tổ hợp phím Alt + F9, nếu không gặp một lỗi nào Pascal sẽ dịch Unit sang dạng mã máy và tự động lưu tệp đã biên dịch vào thư mục TP\BIN. Tệp này sẽ có tên giống như tên tệp nguồn nhưng phần mở rộng sẽ là TPU.

Các TPU có thể lưu vào thư mục khác do người sử dụng tự chọn hoặc thư mục UNITS của Pascal. Nếu lưu và thư mục khác thì người sử dụng cần khai báo đường dẫn đến nơi lưu trữ thông qua lựa chọn Options – Directories – Unit Directories.

Trong trường hợp việc dịch gặp lỗi Pascal sẽ thông báo lỗi như khi dịch một chương trình thông thường.

Chú ý:

Nếu tên tệp nguồn của Unit trong thư mục TP\BIN khác với tên ghi sau từ khoá Unit thì việc dịch không thông báo lỗi, song Pascal không tạo ra được TPU.

Lệnh tham chiếu Uses TênUnit trong các chương trình trước hết sẽ tham chiếu đến các TPU, nghĩa là đến các Unit đã biên dịch, nếu không tìm thấy các TPU, Pascal sẽ tìm các chương trình nguồn của Unit và tự động biên dịch chúng trước khi thực hiện chương trình.

Ví dụ 3.2: xây dựng Unit

Chúng ta sẽ xây dựng một Unit lấy tên là HHP (hình học phẳng). Unit này sẽ tạo nên một số hàm dùng để tính diện tích , chu vi các hình tròn, chữ nhật và tam giác. Đối với tam giác chỉ xét trường hợp biết kích thước ba cạnh, bạn đọc có thể thêm vào đây các hàm mới để tính cho hai trường hợp còn lại của tam giác (cạnh – góc – cạnh và góc – cạnh – góc) hoặc tính thể tích, diện tích cho các hình không gian...

```
Unit HHP;
Interface
    Function Dtcn( a,b:real): real;
    Function Dttr( a:real): real;
    Function Dttg( a,b,c:real): real;
    Function Cvcn( a,b:real): real;
    Function Cvtr( a:real): real;
    Function Cvtg( a,b,c:real): real;
Implementation
    Function Dtcn( a,b:real): real;
    Var dt:real;
```

```
    Begin
        dt:=a*b;
        dtcn:=dt;
    End;

    Function Dttr( a:real): real;
    Var dt:real;
    Begin
        dt:=pi*a*a;
        dttr:=dt;
    End;

    Function Dttg( a,b,c:real): real;
    Var p,dt:real;
    Begin
        p:=(a+b+c)/2;
        dt:=sqrt(p*(p-a)*(p-b)*(p-c));
        dttg:=dt;
    End;

    Function Cvcn( a,b:real): real;
    Var cv:real;
    Begin
        cv:=(a+b)*2;
        cvcn:=cv;
    End;

    Function cvtr( a:real): real;
    Var cv:real;
    Begin
        cv:=2*pi*a;
        cvtr:=cv;
    End;
    Function cvtg( a,b,c:real): real;
    Var cv:real;
    Begin
        cv:=a+b+c;
        cvtg:=cv;
    End;
End.
```

4.5 Phần hướng dẫn

Mỗi Unit khi xây dựng xong cần có phần hướng dẫn để người sử dụng không gặp phải các lỗi khi chương trình tham chiếu đến Unit, chẳng hạn trong Unit HHP khi tính diện tích chu vi tam giác sẽ có thể xảy ra trường hợp số đo các cạnh a, b, c không tạo thành tam giác. Trong trường hợp này người thiết kế Unit phải hướng dẫn người sử dụng kiểm tra điều kiện tạo nên tam giác trước khi gọi các Function của HHP.

Thông thường phần hướng dẫn của Pascal đặt trong thư mục DOC dưới dạng các tệp văn bản (phần mở rộng là TXT). Do bộ mã dùng trong Pascal không có các ký tự tiếng Việt nên phần hướng dẫn chỉ có thể viết bằng tiếng Việt không dấu.

Mặc dù có phần hướng dẫn song nhiều khi người lập trình không đọc kỹ hoặc đọc mà vẫn quên các điều kiện của bài toán do vậy cần cố gắng hạn chế những ràng buộc mà người sử dụng phải tuân theo. Để làm được điều này cần phải chú ý đến các lớp bài toán có thể tham khảo đến Unit, đồng thời phải lường trước những sai sót mà người sử dụng có thể mắc phải. Trong thực tế ngay cả người lập trình chuyên nghiệp cũng không thể nào tránh được sai lầm do vậy cần chú ý đến các thông báo lỗi khi người sử dụng đưa ra các lệnh không phù hợp.

Ví dụ 3.3 sau đây nêu cách sử dụng Unit HHP tính diện tích chu vi tam giác

```
Program tinh_dtcv;
Uses crt,hhp;
Var m,n,q:real;
  Begin
    clrscr;
    Write('Cho biet ba canh '); readln(m,n,q);
    if (m+n>q) and (n+q>m) and (q+m>n) then
    Begin
    Writeln('Dien tich tam giac la ',dttg(m,n,q):5:2);
    Writeln('Chu vi tam giac la  ',cvtg(m,n,q):5:2)
     End
     Else
    Writeln('So lieu da cho khong tao thanh tam giac');
     readln;
  End.
```

Trong ví dụ trên nếu người chạy chương trình nhập vào giá trị của ba cạnh là 1, 3, 8 và trong chương trình không kiểm tra điều kiện tạo nên tam giác thông qua câu lệnh

if (m+n>q) and (n+q>m) and (q+m>n) then ...

thì sẽ dẫn tới diện tích là căn bậc hai của một số âm. Chúng ta có thể thiết kế lại Unit sao cho khi các tham số truyền cho chương trình con không phù hợp thì sẽ xuất hiện thông báo lỗi và yêu cầu nhập vào các giá trị khác. Muốn thế chúng ta phải thay thế các Function tính diện tích và chu vi tam giác thành các Procedure.

Thông thường các Unit được thiết kế chưa thể bao quát ngay mọi yêu cầu mà các bài toán đặt ra vì vậy việc chỉnh sửa, nâng cấp là điều khó tránh khỏi. Để làm việc này người thiết kế Unit nên lưu trữ các cấu trúc dữ liệu, các gợi ý và cách thức tổ chức Unit để khi cần có thể

tra cứu dễ dàng. Điều này có ý nghĩa khi chúng ta nên cung cấp cho người dùng không chỉ các TPU mà cả tệp nguồn với mục đích người sử dụng có thể tự nâng cấp khi cần.

5. Tham chiếu đến các Unit

Một chương trình có thể tham chiếu đến nhiều Unit cùng một lúc, các Unit này có thể là Unit chuẩn của Pascal hay là Unit tự tạo. Trong các Unit tự tạo người thiết kế cũng có thể đưa vào các lệnh tham chiếu đến các Unit khác. Pascal có một nguyên tắc đặt ra là *trong một chương trình các Unit không được tham chiếu vòng tròn.*

Giả sử chúng ta đã thiết kế ba Unit là U1, U2, U3 với phần tiêu đề và Interface như sau:

```
Unit  U1 ;
Interface
Uses U2;
....
End.

Unit  U2;
Interface
Uses U3;
....
End.

Unit  U3;
Interface
Uses U1;
....
End.
```

Nếu trong một chương trình chúng ta có lệnh

Uses U1, U2, U3;

Thì sẽ dẫn đến một tình trạng là U1 tham chiếu đến U2, U2 tham chiếu đến U3 và U3 tham chiếu lại U1, trong trường hợp này Pascal sẽ thông báo lỗi: Circular Unit Reference (Unit tham chiếu vòng tròn). Đây có thể là một lỗi khó hiểu đối với người lập trình nếu như người đó không biết được bản thân các Unit mà mình sử dụng đang tham chiếu đến những Unit nào.

Điều này có thể giải thích là do Unit U1 cần phải tham chiếu đến U3 thông qua U2, nhưng U3 lại phải tham chiếu đến U1 trong khi bản thân U1 chưa hoàn thiện do đó sẽ gây lỗi.

Một vấn đề khác cũng cần phải chú ý là thứ tự tham chiếu đến các Unit, ví dụ lệnh:

Uses CRT, GRAPH; quy ước rằng CRT được tham chiếu trước, GRAPH tham chiếu sau.

Trong trường hợp các TPU và chương trình tham chiếu đến chúng có một số biến , hàm trùng tên và có một số đại lượng được khởi tạo giá trị ban đầu thì Pascal sẽ ưu tiên sử dụng các giá trị được khởi tạo cuối cùng. Chúng ta sẽ giải thích điều này qua ví dụ sau;

Ví dụ 3.4

```
Unit u1;
Interface
Var traloi:char;
Procedure P11;
Implementation
      Procedure P11;
      Begin  Writeln('Day la Unit U1');
      End;
  Begin
    traloi:='K';
  End.

Unit u2;
Interface
Var traloi:char;
Procedure P11;
Implementation
  Procedure P11;
  Begin
  Writeln('Day la Unit U2');
  End;

Begin
traloi:='C';
End.

 Program dung_unit;
 Uses crt,u1,u2;
 Begin
       Clrscr;
      if upcase(traloi)='C' then
      Begin
      p11;
      Writeln('Bien Traloi mang gia tri : ',traloi);
      Writeln('Thu tu tham chieu la U1, U2 ')
```

```
        End
        Else
        Begin
        p11;
        Writeln('Bien tra loi mang gia tri : ',traloi);
        Writeln('Thu tu tham chieu la U2, U1 ')
    End;
        Readln;
End.
```

Ví dụ 3.4 xây dựng hai Unit U1 và U2, cả hai đều có biến Traloi, và thủ tục P11. Thủ tục P11 chỉ làm một công việc là viết lên màn hình dòng chữ "Day la unit ...". Trong U1 biến Traloi được gán giá trị là 'K' còn trong U2 biến này mang giá trị 'C'.

Chương trình Dung_Unit có lệnh tham chiếu

uses crt, u1, u2;

Do Unit U2 được tham chiếu sau U1 nên giá trị của biến Traloi được khởi tạo cuối cùng sẽ thuộc về U2 nghĩa là Traloi = "C". Lúc này biểu thức điều kiện *if upcase(traloi)='C'* trong chương trình Dung_unit mang giá trị True do đó trên màn hình chúng ta nhận được kết quả

Day la Unit U2

Bien Traloi mang gia tri : C

Thu tu tham chieu la U1, U2

Nếu chúng ta đảo ngược lại thứ tự tham chiếu

Uses crt, u2, u1;

thì sẽ nhận được kết quả

Day la Unit U1

Bien Traloi mang gia tri : K

Thu tu tham chieu la U2, U1

Ví dụ trên cũng cũng chỉ ra rằng nếu trong các TPU chúng ta thiết kế có các hàm hoặc thủ tục trùng tên với nhau và trùng tên với hàm, thủ tục có trong chương trình chính, đồng thời các biến (hàm) được gán giá trị ban đầu ngay trong phần Implemetation thì khi chương trình tham chiếu đến TPU nó sẽ lấy giá trị của biến (hàm) được gán cuối cùng. Nếu trong bản thân chương trình chính cũng có lệnh gán giá trị cho biến (hàm) (trùng tên với các biến (hàm) có trong TPU) thì Pascal coi đây là giá trị được gán cuối cùng và giá trị này sẽ được sử dụng trong các ứng dụng.

Nói tóm lại việc thiết kế Unit đúng chưa cho chúng ta kết quả đúng nếu chúng ta không không nắm được nguyên lý sử dụng chúng. Cách tốt nhất là tránh dùng các hàm và biến trùng tên nhau. Trong trường hợp các Unit được thiết kế phục vụ một lớp bài toán nào đó và bắt buộc phải sử dụng một số thủ tục hoặc hàm trùng tên thì cần chỉ rõ cách thức sử dụng chúng trong phần hướng dẫn sử dụng.

6. Trình tiện ích TPUMOVER

Thuật ngữ TPUMOVER được ghép nối bởi hai thành tố TPU (Unit của Pascal) và MOVER (chuyển động, dịch chuyển) vì thế có thể hiểu TPUMOVER nghĩa là dịch chuyển các Unit. Trình tiện ích TPUMOVER được lưu trữ dưới dạng một File chương trình với phần mở rộng là EXE. Nhờ TPUMOVER người sử dụng có thể tự tổ chức các Unit do mình tạo ra thành một thư viện dưới dạng File với tên File theo quy định của hệ điều hành DOS và phần mở rộng là TPL (ngầm định của Pascal). Không những thế người sử dụng còn có thể thêm, bớt các Unit của mình vào chính thư viện chuẩn TURBO.TPL của Pascal hoặc trích một Unit nào đó trong thư viện để trở thành một TPU độc lập.

Tổ chức các Unit tự tạo thành thư viện riêng của người lập trình sẽ tránh được việc các TPU nằm rải rác trong thư mục nhưng lại gặp một rắc rối là mỗi lần muốn dùng Unit ta lại phải tách chúng ra ngoài. Sở dĩ như vậy là vì Pascal chỉ tự động nạp thư viện chuẩn TURBO.TPL khi khởi động máy. Cách tốt nhất là những Unit hay được dùng đến chúng ta gộp chúng vào thư viện chuẩn TURBO.TPL. Việc đưa các Unit tự tạo vào TURBO.TPL sẽ cho phép tăng tốc độ xử lý bởi vì các Unit trong thư viện chuẩn đã được tự động nạp vào bộ nhớ Ram khi Pascal khởi động, việc tìm kiếm ở Ram bao giờ cũng nhanh hơn tìm kiếm ở bộ nhớ ngoài.

Muốn biết cú pháp của Tpumover chúng ta có thể đọc trong tệp Until.doc mục số 4, tệp này nằm trong thư mục TP\DOC hoặc từ màn hình soạn thảo chương trình của Pascal chúng ta gõ Tpumover rồi bấm tổ hợp phím Ctr – F1. Pascal đưa ra cú pháp của Tpumover như sau:

Tpumover Filename Operations

- Filename là tên thư viện chuẩn (Turbo) hoặc tên thư viện mà chúng ta muốn tạo ra
- Operations là danh sách tuỳ chọn của Tpumover, bao gồm

+Unitname: tên Tpu muốn đưa vào thư viện (dấu cộng ở đầu)

-Unitname: tên Tpu muốn xoá khỏi thư viện (dấu trừ ở đầu)

*Unitname: tên Tpu muốn tách thành Unit độc lập (dấu sao ở đầu)

Nếu chúng ta bỏ qua Operations thì Tpumover sẽ hiện lên danh sách các Unit đang có trong thư viện Filename.

Ví dụ 3.5:

Giả sử chúng ta muốn đưa thêm hai Unit U1, U2 đã thiết kế vào thư viện chuẩn Turbo. Để chạy Tpumover chúng ta có hai cách:

a. Từ màn hình nền của Window:

Chọn Start – Run

Trong cửa sổ Run gõ vào

C:\TP\BIN\Tpumover.exe Turbo +u2 +u1

Chọn OK

b. Từ dấu nhắc hệ thống C:\> của Dos:

Gõ các lệnh vào thư mục Bin

C:\>CD TP ↵

C:\TP>CD BIN ↵

Tiếp đó gõ lệnh

C:\TP\BIN>Tpumover Turbo +U1 +U2 ↵

Hai Unit U1, U2 lúc này đã được đưa vào trong thư viện chuẩn Turbo.tpl.

Để xoá Unit U2 khỏi Turbo chúng ta gõ lệnh:

C:\TP\BIN>Tpumover Turbo -U2 ↵

Muốn đưa Unit U1 thành một Unit độc lập ta gõ lệnh:

C:\TP\BIN>Tpumover Turbo *U1 ↵

Unit U1 sau khi tách ra sẽ trở thành tệp U1.TPU trong thư mục BIN nhưng bên trong thư viện chuẩn Turbo.tpl vẫn tồn tại Unit U1. Khi có lệnh tham chiếu

Uses U1;

Pascal sẽ ưu tiên tìm U1 trong Turbo, nếu không thấy thì mới tìm ở ngoài.

7. Công dụng của Unit

7.1 Làm gọn nhẹ chương trình

Vì các hàm và thủ tục cần thiết cho chương trình đều đã cất trong Unit nên kích thước chương trình sẽ giảm đáng kể.

7.2 Tăng tốc độ thực hiện chương trình

Như đã biết trong phần lập trình cơ bản, mỗi chương trình trước khi chạy đều cần phải biên dịch. Vì các Unit mà chương trình tham chiếu đến đã được biên dịch trước cho nên thời gian dịch chương trình có Unit sẽ giảm và hệ quả là tốc độ thực hiện chương trình tăng.

7.3 Cho phép viết được các chương trình lớn

Những chiếc PC đầu tiên sử dụng bộ xử lý Intel 8086, bộ xử lý này chỉ xử lý 16 bit dữ liệu. Số dương 16 bits lớn nhất khi không xét dấu là $1111\ 1111\ 1111\ 1111_2 = 65535_{10}$. Nói cách khác Intel 8086 chỉ có thể quản lý được 65535 địa chỉ (tức là 64 KB). Với những bộ nhớ có dung lương lớn hơn 64 KB người ta phải chia chúng thành các đoạn nhỏ mỗi đoạn có kích thước 64 KB và dùng hai tham số cho một địa chỉ, tham số thứ nhất Segment là địa chỉ của đoạn trong bộ nhớ và tham số thứ hai Offset là địa chỉ của ô nhớ trong mỗi đoạn. Các máy thế hệ sau này dù xử lý 32 bit nhưng kích thước mỗi đoạn vẫn chỉ là 64 KB. Đây chính là lý do khiến cho các chương trình Pascal không thể vượt quá 64 KB. Nhờ các Unit chúng ta có thể chuyển những hàm, thủ tục, thường trình vào Unit. Trong bộ nhớ các Unit này sẽ được nạp vào các đoạn khác với đoạn chứa chương trình. Điều đó có nghĩa là kích thước chương trình có thể vượt quá 64 KB.

7.4 Bảo đảm an toàn

Sau khi đã biên dịch các TPU được lưu trữ dưới dạng mã máy, việc tham chiếu đến TPU là tham chiếu đến phần Interface, như vậy chúng ta không sợ bị thay đổi những thiết kế trong phần Implementation. Đây chính là cách mà chúng ta bảo vệ các cấu trúc dữ liệu cũng

như cấu trúc các thủ tục, hàm đã thiết kế. Việc nâng cấp thay đổi chỉ có thể thực hiện trên tệp nguồn và sau đó lại phải biên dịch trước khi có thể sử dụng.

Cũng cần phải lưu ý rằng Unit không phải chỉ hoàn toàn có lợi, tất cả các Unit trong thư viện chuẩn Turbo đều được nạp vào bộ nhớ khi Pascal khởi động và thường trú ở đó cho nên chúng chiếm dụng một không gian nhớ nhất định. Việc đưa quá nhiều Unit vào thư viện chuẩn sẽ làm giảm đi kích thước bộ nhớ mà chương trình được phép sử dụng.

8. Một số Unit chuẩn

Pascal cung cấp cho người lập trình khoảng 20 Unit chuẩn do đó việc giới thiệu toàn bộ các Unit trong cuốn sách này là điều không thể. Bạn đọc có thể tham khảo danh mục các thủ tục và hàm của từng Unit trong phụ lục ở cuối sách. Trong mục này chúng ta sẽ tìm hiểu một số Unit thông dụng nhất.

8.1 System

System là đơn vị cơ bản của Turbo Pascal trong đó cài đặt toàn bộ các thủ tục và hàm mà chúng ta đã sử dụng từ đầu chương trình, system đã được thiết kế để chạy tự động khi có một chương trình được thực hiện do đó không cần lệnh gọi USES ở đầu chương trình.

Trong UNIT SYSTEM có một số thủ tục hay dùng khi lập trình như sau:

a. Thủ tục EXIT:

Khi gặp EXIT trong một chương trình thì kết thúc chương trình và ra khỏi chương trình đó, nếu đặt exit trong chương trình chính thì dừng toàn bộ công việc.

b. Thủ tục CHDIR (tên thư mục):

Thủ tục CHDIR (change Directory) cho phép chuyển thư mục có tên viết trong dấu ngoặc đơn thành thư mục hiện thời.

c. Thủ tục RMDIR (tên thư mục):

Thủ tục RMDIR (remove Directory) cho phép xoá một thư mục rỗng trên ổ đĩa hiện thời.

d. Thủ tục Rename (<tên cũ>; <Tên mới>):

Thủ tục này đổi tên tệp ngoại vi từ tên cũ thành tên mới.

8.2 Dos

DOS là UNIT chứa các thủ tục liên quan đến hệ điều hành và quản lý tệp:

a. SETDATE(năm, tháng, ngày):

Thủ tục này sẽ đặt lại ngày tháng năm cho đồng hồ của máy, ví dụ: SETDATE (1998, 11, 25);

Lưu ý rằng chỉ có thể đặt lại cho các năm trong phạm vị từ năm 1980 - 2099 nếu đưa vào giá trị khác may sẽ báo lỗi.

b. SETTIME (giờ, phút, giay, phần trăm giây):

Đặt lại giờ cho đồng hồ của máy.

c. GETTIME (giờ, phút, giây, phần trăm giây):

Cho biết giờ hệ thống, muốn viết giờ hệ thống ra màn hình chúng ta có thể viết một chương trình như sau:

```
Program  Xemgio;
Var x1, x2, x3, x4: wrod;
begin
Gettime (x1, x2, x3, x4);
Write ('Bay gio là ',x1:2,' gio ', x2:2,' phut ', x3:2,' giay');
end.
```

Chương trình trên không viết phần trăm giây, xin mời các bạn sửa chương trình để có thể thông báo toàn bộ các thông số , mỗi số trên một dòng.

8.3 Crt

CRT là Unit liên quan đến các thủ tục trình bày màn hình. Crt không nằm trong thư viện chuẩn nên không được nạp tự động. Trong CRT có các thủ tục sau:

a. Gotoxy(m, n):

Chuyển con trỏ tới toạ độ cột m và dòng n. Cần lưu ý rằng 1<=m<=80; 1<=n<=25.

b. CLRSCR: xoá sạch màn hình đưa con trỏ về toạ độ (1,1)

c. CLREOL: xoá các ký tự từ vị trí con trỏ hiện thời đến hết dòng

d. SOUND(n): Tạo ra một âm thanh với tần số n Hz.

Để hiểu rõ hơn về Thủ tục này xin xem chương Âm thanh

e. DELAY (n): Làm chậm (hoặc kéo dài) lệnh phía trước n mili giây (1 giây = 1000 mili giây). Giá trị lớn nhất cho phép là 65535 ms (tức là giá trị số nguyên kiểu Word lớn nhất mà máy có thể xử lý)

f. NOSOUND: Tắt tiếng kêu do SOUND (n) tạo ra

Ba thủ tục sound, delay, nosound được sử dụng để tạo ra các bản nhạc, cần lưu ý rằng sau thủ tục sound và delay âm thanh được tạo ra sẽ kéo dài liên tục nếu không có nosound mặc dù trong delay ta đã chỉ định khoảng thời gian kéo dài là n ms, chỉ khi có thủ tục nosound thì âm thanh mới kéo dài đúng n miligiây

g. LOWVIDEO :làm tối màn hình.

h. HIGHVIDEO : làm sáng thêmmàn hình.

i. NORMVIDEO: trả về trạng thái sáng bình thường.

j. Thuộc tính màu

Có hai thủ tục gán thuộc tính màu cho ký tự viết ra màn hình là:

TEXTCOLOR(Mã màu) : màu chữ

TEXTBACKGROUND(Mã màu): màu nền

Cả hai thủ tục phải viết trước các lệnh Write hoặc Writeln và chúng sẽ có tác dụng cho đến khi chúng ta đặt lại mã màu mới.

Tham số "mã màu" trong hai thủ tục định màu ở trên có thể tham khảo trong bảng 3.1.

Pascal sử dụng một Byte trong bộ nhớ để lưu trữ thuộc tính màu, thuộc tính màu ở đây được hiểu bao gồm: màu chữ, màu nền và khả năng nhấp nháy của chữ.

Trong byte thuộc tính bốn Bit thấp được dùng lưu trữ màu chữ, ba Bit tiếp theo lưu trữ màu nền, Bit cao nhất lưu trữ thuộc tính nhấp nháy. Như đã biết trong hệ đếm cơ số 2 sử dụng khuôn 4 Bit chúng ta có thể biểu diễn các số từ 0 đến 15, còn khuôn 3 Bit chỉ biểu diễn được các số từ 0 đến 7. Điều này có nghĩa là màu chữ có thể chọn một trong 16 giá trị (từ 0 đến 15)

còn màu nền chỉ có 8 giá trị (từ 0 đến 7). Nếu Bit thứ 8 được bật, nghĩa là cộng thêm vào thuộc tính màu giá trị 128 ($1000\ 0000_2 = 128$) thì chữ sẽ nhấp nháy.

Bảng 3.1

Mã màu	Tên hằng	Màu tiếng Việt	Mã màu	Tên hằng	Màu tiếng Việt
0	Black	Đen	4	Red	Đỏ
1	Blue	Xanh lam	5	Magenta	Tím
2	Green	Xanh lá mạ	6	Brown	Nâu
3	Cyan	Xanh lơ	7	LightGray	Xám nhạt

Mã màu	Tên màu	Màu tiếng Việt	Mã màu	Tên hằng	Màu tiếng Việt
8	DackGray	Xám sẫm	12	LightRed	Đỏ sáng
9	LightBlue	Xanh lam sáng	13	LightMagenta	Tím sáng
10	LightGreen	Xanh lá mạ sáng	14	Yellow	Vàng
11	Lightcyan	Xanh lơ sáng	15	White	Trắng

Giá trị thuộc tính màu được tính theo công thức

Thuộc tính màu = Mã màu nền*16 + Mã màu chữ

Ví dụ: chọn chữ màu trắng mã màu là 15, nền màu xám sáng mã màu nền là 7 thì thuộc tính màu sẽ bằng

7 * 16 + 15 = 127

Trong thực tế không cần phải cộng thêm vào mã màu chữ số 128 mà chỉ cần cho mã màu chữ lớn hơn 15 thì chữ đã nhấp nháy rồi. Pascal có sẵn một biến là TextAttr kiểu Byte để lưu trữ thuộc tính màu, nếu thuộc tính màu nhỏ hơn 128 thì

Mã màu chữ = TextAttr Mod 16

Mã màu nền = TextAttr Div 16

Nếu thuộc tính màu lớn hơn 128 như cách chọn màu sau đây:

```
Textcolor(16);
TextbackGround(7);
```

Thì thuộc tính màu là 240 và mã màu chữ thực tế sẽ là 0, còn mã màu nền là 7. Sở dĩ như vậy là vì Pascal đã tự động cộng thêm 128 vào mã màu, hay nói cách khác

Mã màu chữ = (240 - 128) Mod 16 = 0

Mã màu nền = (240 - 128) div 16 = 7

Ví dụ 3.6 viết lên màn hình một chuỗi ký tự "*", chương trình có sử dụng 3 biến : biến Ttm lưu trữ thuộc tính màu, biến Mc là mã màu chữ, Mn là mã màu nền. Người sử dụng có thể chạy chương trình với các tham số màu khác nhau và chương trình sẽ thông báo thuộc tính màu và mã màu thực tế .

Ví dụ 3.6

```
Program Ma_mau;
Uses crt;
Var Ttm,Mc,Mn:byte;
Begin
  clrscr;
  textcolor(26);
  textbackground(7);
  writeln('**********');
  ttm:=textattr;
  mc:= textattr mod 16;
  writeln('Thuoc tinh mau ',ttm);
  Writeln('Mau chu : ', mc);
  if ttm>127 then
   mn:= (ttm-128) div 16
  else
  mn:= ttm div 16;
  Writeln('Mau nen : ', mn);
  repeat until keypressed;
End.
```

j. Hàm KEYPRESSED:

Hàm Keypressed kiểm tra xem trong quá trình làm việc có phím nào được bấm hay không, nếu có thì hàm mang giá trị True, còn nếu không thì mang giá trị False.

Hàm Keypressed thường được dùng trong các câu lệnh để kiểm tra điều kiện, ví dụ:

If keypressed then.... nếu có phím được bấm thì làm công việc...

Hoặc: Repeat .... Until Keypressed;

lặp lại công việc cho đến khi có phím được bấm

Cũng có thể dùng câu lệnh Repeat until keypressed để dừng chương trình, khi muốn tiếp tục thì chỉ cần bấm một phím bất kỳ. (Cần phân biệt với lệnh Readln, khi muốn tiếp tục thì phải bấm Enter).

k. Hàm READKEY:

Hàm Readkey nhận diện phím được bấm, giá trị mà nó nhận về là một ký tự. Có hai khả năng xảy ra đối với hàm Readkey:

*. Nếu vùng đệm bàn phím đang rỗng, nghĩa là các ký tự đưa vào đã được xử lý hết thì Readkey sẽ dừng chương trình chờ cho đến khi một phím được bấm, phím được bấm một mặt được đưa vào xử lý, mặt khác được lưu trong hàm Readkey.

*. Nếu vùng đệm bàn phím không rỗng, nghĩa là đã có các ký tự được bấm đang nằm chờ thì Readkey sẽ lấy ra ký tự đầu tiên trong vùng đệm và vùng đệm bàn phím sẽ giảm đi một ký tự.

l. Hàm GETKEY:

Hàm Getkey hoạt động tương tự như hàm Readkey nhưng giá trị mà nó nhận là một số dương đối với các phím thông thường, nếu là các phím có mã quét mở thì thì giá trị Getkey nhận gồm hai số, số đầu là số 0 còn số thứ hai là mã quét mở của phím, ví dụ:

Nếu bấm chữ J thì Getkey cho số 75, còn nếu bấm phím dịch chuyển con trỏ (mũi tên) sang trái thì Getkey cho 0 - 75.

Chúng ta cần có sự phân biệt rõ ràng phím trên bàn phím và ký tự trong bảng mã. Bàn phím kiểu Mỹ chỉ có 101 phím, còn bảng mã có 256 ký tự. Chỉ có một phần nhỏ ký tự trong bảng mã nhìn thấy được trên bàn phím (bao gồm các chữ cái, chữ số, toán tử số học, toán tử so sánh và một vài ký tự khác), các ký tự còn lại muốn cho hiện trên màn hình phải dùng tổ hợp 2 phím Alt - Số thứ tự.

Dưới đây là bảng đối chiếu một số phím và mã phím thường dùng

Bảng 3.2

Phím	**Mã**
BackSpace	8
Tab	9
Enter	13
Esc	27
Shift - Tab	0-15
↑	0-72
←	0-75
→	0-77
↓	0-80
Insert	0-82
Delete	0-83

Phím	**Mã**
Home	0-71
End	0-79
PageUp	0-73
PageDown	0-81
Ctrl - End	0-117
Ctrl - Home	0-119
Ctrl - ←	0-115
Ctrl - →	0-116
Ctrl - PageUp	0-132
Ctrl - PageDown	0-118

Để có thể nhận diện các phím có mã quét mở rộng chúng ta có thể dùng hai biến kiểu Char, một biến chứa ký tự có mã bằng 0, còn biến kia chứa ký tự của phần mã quét mở, nói cách khác cần phải gọi hàm Readkey hai lần mới bắt được phím có mã quét mở.

Ví dụ 3.7:

```
Var  C1,C2:char;
Begin
C1:=Readkey;  If C1=chr(0) then C2:=Readkey;
End.
```

Chúng ta cũng có thể dùng biến kiểu Byte để bắt các phím có mã quét mở. Trong thực tế vì đã biết số thứ nhất trong mã quét mở bằng không nên chỉ cần tìm số thứ hai của mã quét như ví dụ sau:

```
Var mq:byte;
Begin
Mq=readkey;
If  ord(mq) = 0 then mq := ord(readkey);
End.
```

Nếu biến mq có giá trị 75 thì phím được bấm không phải là chữ J mà là phím ← bởi vì mã quét ở đây là 0 - 75.

Chú ý:

Khi xây dựng chương trình có việc đọc mã quét, cần đặc biệt chú ý trong vùng đệm bàn phím đã có mã phím nằm chờ hay không, nếu có thì phải làm sạch vùng đệm bàn phím trước khi đọc mã. Việc làm sạch vùng đệm bàn phím có thể thực hiện bằng câu lệnh lặp sau:

```
While  Keypressed do mq:= readkey;
```

Biến mq phải được khai báo trước và thuộc kiểu Char.

Ví dụ 3.8:

Viết chương trình hiện mã quét của các phím, nếu là phím có mã quét mở rộng thì hiện cả hai số dưới dạng 0_00.

```
Program Batphim;
 Uses crt;
 Var k,k1,k2:byte;
Function bp(Var mq1,mq2:byte):byte;
Begin
    mq1:=ord(readkey);
    if  mq1=0 then
     begin
       mq2 := ord(readkey);
       bp := mq2;
     end
     else
      bp:=mq1;
end;

  Begin
    Clrscr;
    Writeln('Xin moi bam 1 phim, bam phim END de ket thuc ');
    Repeat
    K := bp(k1,k2);
    if  k1<>0 then
    Writeln('Ma phim ban vua bam = ',k1)
```

```
  else
  Writeln('Ma phim ban vua bam = ',k1,'_',k2);
  Until k=79;
End.
```

Chương trình đặt tên là BATPHIM có một chương trình con là hàm BP, hàm BP có hai tham biến dùng để lưu các mã quét mq1, mq2.

Lời gọi trong chương trình mẹ:

K:=bp(k1,k2); sẽ truyền cho hàm BP hai tham số thực k1,k2, các tham số này sẽ chứa mã quét mq1, mq2 khi quay về chương trình mẹ. Biến K dùng để lưu mã quét mq2 của phím END khi muốn kết thúc chương trình .

Chạy chương trình trên, khi xuất hiện thông báo:

" Xin moi bam 1 phim, bam phim END de ket thuc",

chúng ta có thể bấm một phím nào đó trên bàn phím và sẽ nhận được thông báo mã phím vừa bấm.

Chú ý: Các phím chức năng phụ trên bàn phím không thấy được mã quét bao gồm:

CAPSLOCK, SHIFT, CTRL, ALT, INS, DEL, NUMLOCK, PRINT SCREEN, SCROLL LOCK, PAUSE, F11, F12

Việc bắt phím thường được sử dụng khi thiết kế các Menu (thực đơn) trong trường hợp bài toán cần có một vài phương án lựa chọn công việc. Dưới đây là ví dụ thiết kế thực đơn với bốn tuỳ chọn

Tamgiac Chunhat Tron Ketthuc

Để chuyển con trỏ đến một chức năng nào đó ta bấm các phím ← hoặc →, để chọn chức năng bấm phím Enter, kết thúc công việc bấm End. Chương trình có sử dụng Unit HHP đã thiết kế.

Ví dụ 3.9

```
 Program Menu;
 Uses  crt,hhp;
 label  h1,h2,h3;
 Var   c1,c2,c3,c4:string[20];
       l1,l2,l3,l4,mq1,mq2:byte;      chon:char;
       Procedure tr;          { thu tuc tinh hinh tron }
          Var a,dt,cv:real;
      Begin
       clrscr;
       write('Hay cho biet ban kinh hinh tron '); Readln(a);
       dt:=pi*a*a; cv:=2*pi*a;
       writeln('Dien tich hinh la : ',dt:12:4);
       writeln('Chu vi hinh la    : ',cv:12:4);
```

```
    writeln('Bam Enter de quay ve thuc don ');
    repeat until keypressed;
  End;

  Procedure cn; { thu tuc tinh hinh chu nhat }
  Var a,b,dt,cv:real;
  Begin
      clrscr;
      write('Hay cho biet hai canh a, b '); Readln(a,b);
      dt:=a*b; cv:=(a+b)*2;
      writeln('Dien tich hinh la : ',dt:12:4);
      writeln('Chu vi hinh la    : ',cv:12:4);
      writeln('Bam Enter de quay ve thuc don ');
      repeat until keypressed;
  End;

  Procedure tg; { thu tuc tinh hinh tam giac }
  Var a,b,c,p,dt,cv:real;
  Begin
      clrscr;
      write('Hay cho biet ba canh a,b,c '); Readln(a,b,c);
      if (a+b>c) and (b+c>a) and (c+a>b) then
      Begin
          p:=(a+b+c)/2;
          dt:=sqrt(p*(p-a)*(p-b)*(p-c));
          cv:=a+b+c;
          writeln('Dien tich hinh la : ',dt:12:4);
          writeln('Chu vi hinh la    : ',cv:12:4);
      end
      else
      writeln('Cac kich thuoc ',a:6:2,' ',b:6:2,' ',c:6:2,' khong tao thanh tam giac');
      writeln('Bam Enter de quay ve thuc don ');
    repeat until keypressed;
    End;

Procedure w1;    { thiet ke chuc nang thu nhat }
Begin
  textbackground(5);
  textcolor(10);
  window(1,2,l1+2,2);  write(c1);
```

```
    gotoxy(1,1);
 End;

Procedure w2; { thiet ke chuc nang thu hai }
Begin
    textbackground(5);
    textcolor(10);
    window(l1+2,2,l1+l2+3,2);  write(c2);
    gotoxy(1,1);
End;

Procedure w3; { thiet ke chuc nang thu ba }
Begin
     textbackground(5);
     textcolor(10);
     window(l1+l2+3,2,l1+l2+l3+3,2);  write(c3);
    gotoxy(1,1);
 End;

Procedure w4; { thiet ke chuc nang ket thuc }
Begin
    textbackground(5);
     textcolor(10);
    window(l1+l2+l3+4,2,l1+l2+l3+l4+8,2);  write(c4);
    gotoxy(1,1);
End;

BEGIN        {Than chuong trinh chinh}
    Clrscr;
    c1:='Tamgiac'; l1:=length(c1);
    c2:='Chunhat'; l2:=length(c2);
    c3:='Tron';    l3:=length(c3);
    c4:='Ketthuc'; l4:=length(c4);
    clrscr;
    textcolor(red); textbackground(green);
    write('Bam ->, <- chuyen thuc don | Bam Enter de chon | Bam End ket thuc ');
h1:  w1;w2;w3;w4;w1;
    while keypressed do chon:=readkey;       {lam sach vung dem ban phim}
    chon:=readkey;              {tinh tam giac}
    if ord(chon)=13  then
```

```
    begin
       textcolor(blue); textbackground(14);
       window(1,4,80,25);
        tg;
       clrscr;
       goto h1
    end
    else
    begin
       if ord(chon)=0 then chon:=readkey;
        if ord(chon)=79 then halt; {bam END de ket thuc}
        if ord(chon)=77 then w2;
    end;

h2: w1;w2;w3;w4; w2;
   while keypressed do
   chon:=readkey;      {lam sach vung dem ban phim}
   chon:=readkey;                {tinh chu nhat}
   if ord(chon)=13 then
   begin
       textcolor(blue); textbackground(14);
       window(1,4,80,25);
       cn;
       clrscr;
       goto h2;
   end
   else
   Begin
       if ord(chon)=0 then
       begin
           chon:=readkey;
               if ord(chon)=79 then halt; {bam END de ket thuc}
               if ord(chon)=77 then w3; {chuyen sang tinh hinh tron}
                if ord(chon)=75 then  goto h1; {quay ve tinh tam giac}
       end;
       end;

h3: w1;w2;w3;w4;w3;
 while keypressed do chon:=readkey;       {lam sach vung dem ban phim}
   chon:=readkey;          {tinh hinh tron}
```

```
if ord(chon)=13 then
   begin
      textcolor(blue); textbackground(14);
      window(1,4,80,25);
      tr;
      clrscr;
      goto h3;
   end
   else
   Begin
   if ord(chon)=0 then
      begin
         chon:=readkey;
         if ord(chon)=79 then halt;      {bam END de ket thuc}
         if ord(chon)=77 then w4;        {sang chuc nang ket thuc }
         if ord(chon)=75 then  goto h2; {quay ve tinh chu nhat}
      end;
   end;
   while keypressed do chon:=readkey;        {lam sach vung dem ban phim}
   chon:=readkey;
   if ord(chon)=13 then halt
    else
       if ord(chon)=0 then chon:=readkey;
      if ord(chon)=79 then halt;
      if ord(chon)=75 then goto h3;
END.
```

8.4 Printer

Đơn vị chuẩn Printer có một phần khai báo máy in phục vụ việc in ấn dữ liệu ra giấy, máy in được cài đặt tên là LST.

Khi cần in ấn ta phải có lời gọi USES PRINTER; sau đó trong các lệnh Write hoặc writeln phải đưa tên máy in LST vào trước các tên biến cần in, ví dụ:

Writeln (lst, 'Ket qua tinh toan:', x1);

Dòng lệnh trên sẽ in ra giấy dòng chữ: "Ket qua tinh toan:" và giá trị biến của x1.

Bài tập ứng dụng chương 3

1. Lập chương trình tạo thực đơn hai mức, mức 1 theo chiều ngang , mức 2 theo chiều dọc theo mẫu dưới đây:

HINH PHANG	HINH KHONG GIAN
Tam giac	Hinh non
Tron	Hinh chop
Ket thuc	Ket thuc

Yêu cầu: khi con trỏ ở một trong hai chức năng HINH PHANG, HINH KHONG GIAN nếu bấm Enter thì xuất hiện thực đơn dọc phía dưới, còn nếu bấm phím dịch chuyển con trỏ thì sẽ chuyển con trỏ sang phải hoặc trái. Khi thực đơn dọc xuất hiện sẽ có con trỏ để dịch chuyển lên xuống, bấm Enter để thực hiện một trong các việc tính diện tích, chu vi (đối với hình phẳng) hoặc diện tích toàn phần và thể tích (đối với hình không gian).

2. Cho hai ma trận chữ nhật A(m,n), B(k,q) xây dựng chương trình kiểm tra xem có thể nhân hai ma trận hay không? Xây dựng một Unit tính tích hai ma trận.

3. Xây dựng một Unit với các thủ tục SXT (sắp xếp tăng) và SXG (sắp xếp giảm) để sắp xếp một dãy số.

4. Xây dựng một Unit lấy tên là HAMSOHOC trong đó có các hàm tính căn bậc n, luỹ thừa bậc n của một số (n>2).

5. Xây dựng một Unit nhằm đổi tên màu tiếng Anh sang tiếng Việt ví dụ RED - DO, BLUE - XANH DA TROI...

6. Xây dựng chương trình cho một dòng chữ chạy trên màn hình. Khi dòng chữ đang chạy có thể đổi hướng chạy bằng cách bấm các phím dịch chuyển con trỏ

7. Phát triển bài tập 6 bằng cách bổ xung tính năng tăng hoặc giảm tốc độ chạy khi bấm các phím PageUp hoặc PageDown

Chương IV

Con trỏ và cấu trúc động

Chương này đòi hỏi các kiến thức của môn Cấu trúc dữ liệu và giải thuật, đặc biệt là kiến thức về đữ liệu kiểu Cây. Do cách thức bố trí trong kế hoạch đào tạo môn này lại học song song với môn Lập trình nâng cao nên sẽ có một vài khó khăn khi trình bày cũng như khi nghe giảng. Trong chương này bạn đọc cần chú ý các vấn đề sau:

- Thế nào là kiểu dữ liệu con trỏ
- Sự khác nhau giữa kiểu dữ liệu con trỏ và biến con trỏ
- Sự phân vùng bộ nhớ cho biến con trỏ
- Cách thức mà hệ thống cấp phát bộ nhớ khi chương trình đang làm việc
- Thu hồi bộ nhớ dành cho từng biến và thu hồi hàng loạt
- Cây và cây nhị phân
- Bộ nhớ kiểu LIFO và FIFO và ứng dụng trong thiết kế cây nhị phân
- Con trỏ mảng và mảng con trỏ

1. Khái niệm

Khi khai báo một biến, dù là biến đơn hay biến thuộc kiểu dữ liệu có cấu trúc mặc nhiên chúng ta đã quy định độ lớn vùng nhớ dành cho biến.

Ví dụ

a: Real; biến a cần 6 byte

b: aray[1..100] of Integer; biến mảng b cần 200 byte.

Việc khai báo như trên thường là phỏng đoán dung lượng cần thiết chứ không thật chính xác. Để tránh lỗi chúng ta thường khai báo dôi ra gây nên lãng phí bộ nhớ. Việc xác định địa chỉ lưu trữ biến và cấp phát bộ nhớ được thực hiện khi biên dịch, nghĩa là các địa chỉ này cũng như dung lượng bộ nhớ cần cấp phát đã được cố định trước khi thực hiện các thao tác khác. Các đại lượng này không thay đổi trong suốt quá trình thực hiện chương trình, nói cách khác đây là các đại lượng tĩnh.

Để tiết kiệm bộ nhớ, ngay khi chương trình đang làm việc người lập trình có thể yêu cầu cấp phát bộ nhớ cho các biến, điều này được gọi là *cấp phát bộ nhớ động*. Cấp phát bộ nhớ động được thực hiện thông qua *biến con trỏ*. Muốn có biến con trỏ chúng ta phải định nghĩa *kiểu con trỏ* trước.

2. Kiểu dữ liệu con trỏ - biến con trỏ

2.1 Con trỏ có định kiểu

Kiểu con trỏ là một kiểu dữ liệu đặc biệt dùng để biểu diễn các địa chỉ. Kiểu con trỏ do người lập trình định nghĩa theo cú pháp sau:

```
Type
    Tên kiểu con trỏ = ^Kiểu dữ liệu;
```

Tên kiểu con trỏ tuân theo quy định đặt tên của Pascal, *Kiểu dữ liệu* của kiểu con trỏ là các kiểu dữ liệu đã định nghĩa trước trong pascal. Để một kiểu con trỏ có thể đại diện cho một biến nào đó thì *Kiểu dữ liệu* viết sau ký tự ^ sẽ phải giống như kiểu dữ liệu của biến đó, nói cách khác hai kiểu dữ liệu phải tương thích.

Ví dụ 4.1 khai báo kiểu con trỏ:

```
Type
  Chu = string[20];  CT1 = ^Byte;  CT2 = ^chu;  CT3 = ^Nguoi;
  Nguoi = record
     Hoten:string[20];
     Namsinh: 1900..2100;
     End;
```

Ví dụ 4.1 định nghĩa ba kiểu con trỏ, riêng kiểu CT3 cách định nghĩa có vẻ hơi ngược là định nghĩa kiểu con trỏ trước, định nghĩa kiểu dữ liệu sau. Thật ra chúng ta cứ nên theo thói quen là định nghĩa kiểu dữ liệu trước rồi định nghĩa kiểu con trỏ sau, cách định nghĩa CT3 chẳng qua là muốn giới thiệu để chúng ta biết rằng Pascal cho phép làm ngược lại, tuy nhiên cần nhớ rằng nếu định nghĩa kiểu con trỏ trước thì ngay trong phần Type phải định nghĩa kiểu dữ liệu (không nhất thiết phải liền ngay sau định nghĩa kiểu con trỏ).

Cần chú ý rằng *Pascal chỉ cho phép đưa trực tiếp vào định nghĩa kiểu con trỏ các kiểu dữ liệu đơn giản sau: **số nguyên, số thực, ký tự***. Các kiểu dữ liệu có cấu trúc muốn đưa vào con trỏ thì phải thông qua một tên kiểu khai báo trong phần Type

Cách định nghĩa hai kiểu con trỏ Hoten và Ds sau là sai:

```
Type
  Hoten = ^String[20];
  Ds = ^Array[1..10] of Byte;
```

Muốn sử dụng kiểu chuỗi và mảng cho kiểu con trỏ chúng ta phải định nghĩa như sau:

```
Type
  S1 = string[20];
  Hoten = ^S1;
  a = array[1..10] of byte;
  Ds = ^a;
```

2.2 Biến con trỏ

Biến con trỏ cũng như biến mảng, biến kiểu bản ghi hay kiểu tập hợp có thể khai báo thông qua kiểu con trỏ hoặc khai báo trực tiếp. Biến con trỏ có định kiểu sẽ trỏ đến một kiểu dữ liệu cụ thể.

Để thuận tiện từ nay chúng ta dùng thuật ngữ "**Con trỏ**" thay cho thuật ngữ " **Biến con trỏ**"

Ví dụ 4.2

```
Var
  So: ^Integer;
  Sinhvien: Ct3;
  Hoten: Ct2;
  Thutu, Mahoso: ^Word;
```

Trong ví dụ 4.2 chúng ta đã khai báo ba con trỏ So, Thutu, Mahoso theo kiểu trực tiếp, hai con trỏ Sinhvien và Hoten khai báo thông qua kiểu đã định nghĩa trong ví dụ 4.1. Con trỏ So trỏ tới kiểu dữ liệu số nguyên, con trỏ Sinhvien trỏ tới kiểu dữ liệu bản ghi còn con trỏ Hoten trỏ tới kiểu dữ liệu chuỗi.

Địa chỉ của các biến động và biến tĩnh sẽ được Pascal lưu trữ vào biến con trỏ điều này có nghĩa là *biến con trỏ không dùng để lưu trữ các giá trị của biến mà là địa chỉ của biến*. Dù kích thước vùng dữ liệu mà các biến con trỏ trỏ tới khác nhau thế nào thì kích thước của biến con trỏ cũng vẫn là 4 byte.

Các hàm và thủ tục xử lý biến con trỏ được Pascal lưu trữ trong Unit System.

Quy ước: Các biến con trỏ gọi là tương thích nếu chúng trỏ tới cùng một kiểu dữ liệu

2.3 Con trỏ không định kiểu

Con trỏ không định kiểu là kiểu con trỏ không quan tâm đến kiểu dữ liệu mà nó trỏ tới. Pascal dùng tên chuẩn Pointer để khai báo kiểu này.

Var

Tên biến:Pointer;

Con trỏ không định kiểu được coi là tương thích với mọi kiểu con trỏ.

Chú ý:

** Về bản chất tất cả con trỏ đều chứa địa chỉ nên chúng không có gì khác nhau song để tránh nhầm lẫn trong các quá trình xử lý Pascal chỉ coi các con trỏ cùng trỏ tới một kiểu dữ liệu là tương thích với nhau.*

2.4 Địa chỉ của một đối tượng

Đối tượng mà chúng ta đề cập trong mục này có thể là biến, hàm hay thủ tục. Khi biên dịch chương trình mỗi đối tượng được cấp phát một vùng nhớ, vùng nhớ này bao gồm một số ô nhớ liền kề nhau.

Địa chỉ một đối tượng trong bộ nhớ được xác định bởi địa chỉ của ô nhớ đầu tiên mà hệ thống dành cho đối tượng đó.

Bộ nhớ của các máy PC hiện nay là rất lớn và chúng được chia thành nhiều đoạn, mỗi đoạn có 65536 ô nhớ (2^{16} ô) . Ô đầu tiên của mỗi đoạn có địa chỉ là 0 do đó ô cuối cùng có địa chỉ là 65535. Như vậy, để biết địa chỉ một ô nhớ cần biết ô nhớ đó thuộc đoạn nào và đó là ô nhớ số bao nhiêu trong đoạn đó.

Địa chỉ đoạn gọi là Segment và địa chỉ tương đối của ô nhớ trong đoạn gọi là Offset, mỗi giá trị này Pascal dùng 2 byte để lưu trữ nên một địa chỉ cần 4 byte, 2 byte thấp cho Offset và 2 byte cao cho segment.

Nếu ghi địa chỉ bằng các số nhị phân thì chúng ta phải dùng 32 chữ số 0 và 1 điều này khá là phiền phức do vậy người ta dùng hệ đếm cơ số 16. Cách ghi địa chỉ ô nhớ được quy ước như sau: địa chỉ đoạn viết trước, vị trí của ô trong đoạn viết sau, ký hiệu $ được thêm vào trước các giá trị số để thể hiện rằng các số viết trong hệ 16.

Ví dụ: $0101:$FFFF

Ví dụ trên cho ta địa chỉ ô nhớ cuối cùng (ô thứ $ffff_{16} = 65535_{10}$) thuộc đoạn 257.

3. Các thủ tục và hàm tác động trên con trỏ

3.1 Gán giá trị ban đầu

Giả sử ct là một biến con trỏ đã được định nghĩa, để đảm bảo rằng ct chưa trỏ đến bất kỳ một đối tượng nào, nghĩa là ct là một con trỏ rỗng chúng ta phải gán cho ct giá trị NIL

ct := Nil;

3.2 Gán địa chỉ của một đối tượng cho con trỏ

Giả sử ct là một con trỏ và x là một đối tượng (biến, hàm, thủ tục), có ba cách gán địa chỉ của đối tượng x cho con trỏ ct:

a. ct := @x;

Trong phép gán trên toán tử @ tác động trên đối tượng x sẽ gán vào con trỏ ct địa chỉ kiểu Pointer của đối tượng đó.

b. **ct := Addr(x);**

Hàm Addr() cho địa chỉ của đối tượng x, địa chỉ này thuộc kiểu Pointer

c. **ct := Ptr(segment,offset) ;**

Hàm Ptr trong phép gán trên đòi hỏi các tham số segment và offset phải là giá trị kiểu Word viết trong hệ 16, ví dụ:

ct := Ptr($B800, $0000); đưa con trỏ trỏ tới ô nhớ của vùng Video Ram

Nhận xét:

Hai phép gán @ và Addr() cùng trả về địa chỉ kiểu pointer nên chúng là tương đương.

3.3 Phép gán giữa hai con trỏ

Hai con trỏ tương thích (cùng kiểu) có thể gán giá trị cho nhau, khi đó chúng cùng trỏ tới một địa chỉ.

Ví dụ 4.3

```
Var
ct1: ^Float;
ct2: ^Byte;
ct3: Pointer;
x: string;
```

Ví dụ trên khai báo ba con trỏ thuộc ba kiểu khác nhau, ct1 là con trỏ thực, ct2 là con trỏ nguyên và ct3 là con trỏ không định kiểu, x là biến chuỗi. Khi đó các phép gán:

```
ct3:=@x;
ct2 := ct3;
```

là hợp lệ vì ct2 và ct3 là tương thích, chúng cùng trỏ đến địa chỉ của biến x.

Còn phép gán

```
ct1 := ct2;
```

là không hợp lệ vì hai con trỏ không tương thích.

3.4 Phép so sánh hai con trỏ

Chỉ tồn tại phép so sánh = (bằng nhau) và <> (khác nhau) giữa hai con trỏ nếu chúng tương thích. Kết quả so sánh là một giá trị Boolean nghĩa là True hoặc False.

*Hai con trỏ **tương thích** gọi là bằng nhau nếu chúng cùng trỏ tới một đối tượng, ngược lại gọi là khác nhau.*

Ví dụ 4.4

```
        Program contro;
Uses crt;
Var
        x,y:real; z:string;
        ct1: ^integer;  ct2: ^byte;  ct3:pointer;  ct4,ct5: ^real;
```

```
Begin
clrscr;
z:='Ha noi';  x:=5;  y:=-123.45;
ct1:=@x; ct2:=@z;  ct3:=@z;  ct4:=@z;  ct5:=@y;
Writeln(ct1=ct2);       { không tương thích, máy sẽ báo lỗi}
  Writeln(ct1=ct3);   {false}
  Writeln(ct2=ct3);   {true}
  writeln(ct4=ct5);   {false}
  readln;
  end.
```

Ví dụ 4.4 khai báo năm con trỏ, ct1 và ct2 trỏ tới các kiểu nguyên khác nhau, ct3 là con trỏ không kiểu tức là tương thích với mọi con trỏ khác, hai con trỏ ct4 và ct5 là cùng kiểu. Các phép so sánh trong thân chương trình cho thấy một số điều cần chú ý :

a. Phép so sánh ct1=ct2 là không hợp lệ vì hai con trỏ không tương thích

b. Phép so sánh ct1 = ct3 cho kết quả False vì hai con trỏ tương thích nhưng trỏ tới các địa chỉ khác nhau

c. Phép so sánh ct2 = ct3 cho kết quả True vì hai con trỏ là tương thích và cùng trỏ tới một địa chỉ.

d. Phép so sánh ct4 = ct5 cho kết quả False vì hai con trỏ cùng kiểu nhưng trỏ tới các địa chỉ khác nhau.

4. Truy nhập dữ liệu

Khi con trỏ ct đang trỏ tới một vùng dữ liệu nào đó Pascal cho phép dùng ký hiệu ct^ như là một biến để truy nhập vào vùng dữ liệu đó. Biến ct^ mang trong nó dữ liệu của vùng mà con trỏ ct đang trỏ tới.

Như vậy chúng ta có thể truy nhập tới một biến, hàm hay thủ tục mà không cần biết tên các đối tượng này miễn là biết con trỏ đang trỏ vào chúng.

Ví dụ 4.5

```
Program contro1;
Uses crt;
Type z1=string[3];
 Var
    z:string;  ct:^z1; i:byte;

Begin
clrscr;
z:='Ha noi';  ct:=@z;
writeln(ct^);
```

```
  for i := 1 to length(z) do write(upcase(ct^[i]));
  readln;
  end.
```

Chạy chương trình ta nhận được kết quả:

Ha noi

HA NOI

Điều này cho thấy rằng mọi xử lý trên biến z đều có thể xử lý trên biến ct^ bởi vì biến con trỏ ct đang trỏ vào z.

Đến đây cần có sự phân biệt chính xác về biến con trỏ CT và biến CT^. Biến con *trỏ CT mang trong nó địa chỉ của đối tượng mà nó trỏ tới, còn biến CT^ lại chứa đựng dữ liệu trong vùng nhớ mà con trỏ CT đang trỏ tới.*

Với con trỏ có kiểu tất cả các thao tác trên biến, hàm hay thủ tục mà con trỏ đang trỏ tới có thể thay thế bởi thao tác trên biến ct^. Kiểu của biến ct^ chính là kiểu đã khai báo cho con trỏ ct chứ không phải là kiểu của đối tượng mà biến ct^ đại diện.

Về điều này cần có một số giải thích cụ thể qua ví dụ sau:

Ví dụ 4.6

```
        Program contro;
uses crt;
Type z1=string[3];
Var x,y:real; z:string;
   ct1:^byte; ct2:^integer; ct3:pointer; i:char;
   ct4:^real; ct5:^word; ct6:^z1;  ct7:^longint;
Begin
  clrscr;
  z:='H';
  ct1:=@z; ct2:=@z; ct5:=@z; ct6:=@z; ct7:=@z;
  writeln(ct1^);  writeln(ct6^);  writeln(ct2^);  writeln(ct5^);  writeln(ct7^);
  readln;
End.
```

Chạy chương trình trên chúng ta nhận được kết quả:

1

H

18433

18433

18433

Thay lệnh gán z:='H'; bằng lệnh gán z:='Ha noi Viet nam': thì kết quả chạy chương trình sẽ là

15

Ha noi Viet nam
18447
18447
543246351

Trong ví dụ 4.6 tất cả con trỏ đều được gán địa chỉ của biến z, nhưng vì các con trỏ đại diện cho các kiểu dữ liệu khác nhau nên kết quả mà biến con trỏ trả về cũng khác nhau.

Biến Ct1^ thuộc kiểu Byte nên nó cho ta dữ liệu trong Byte đầu tiên của vùng nhớ chứa z, đó chính là ô nhớ chứa độ dài của chuỗi z.

Biến Ct6^ thuộc kiểu z1 tức là kiểu chuỗi nên kết quả mà nó trả về chính là chuỗi z (không phụ thuộc vào CT6^ khai báo dài bao nhiêu).

Các biến còn lại thuộc kiểu số nên kết quả trả về cũng là số, với trường hợp ký tự H các biến con trỏ ct2^, ct5^, ct7^ cho ta giá trị 18433. Nếu ký tự gán cho z là A thì giá trị trả về là 16641, là B thì giá trị này tăng thêm 256...

Với chuỗi "Ha noi Viet nam" sau H là 14 ký tự nữa cho nên giá trị trả về là

18433 + 14 = 18447.

Đến đây chúng ta có thể rút ra một số kết luận:

*. Địa chỉ của một đối tượng có thể gán cho bất kỳ con trỏ nào.

*. Kết quả mà biến ct^ trả về thuộc kiểu dữ liệu của con trỏ chứ không thuộc kiểu dữ liệu của đối tượng.

*. Muốn sử dụng biến ct^ như một biến thông thường thay thế cho đối tượng thì biến con trỏ và đối tượng phải tương thích về kiểu (cùng một kiểu dữ liệu).

*. Với con trỏ không định kiểu (Pointer) chúng ta không thể coi chúng là tương đương với các biến định kiểu thông thường, điều này có nghĩa là không thể sử dụng các thủ tục Write, Read hoặc phép gán cho biến ct^ nếu ct là Pointer.

Ví dụ: trở lại ví dụ 4.5 các cặp thao tác mà chúng ta thực hiện sau đây là tương đương:

Thao tác trên biến	Thao tác trên con trỏ
z:='Ha noi';	ct2^ := 'Ha noi';
x:=5;	ct1^ := 5;
y:=-123.45;	ct5^ := -123.45;

5. Mảng con trỏ và con trỏ kiểu mảng

Con trỏ là một kiểu dữ liệu cho nên biến con trỏ có thể là các thành phần của mảng, ngược lại mảng là một kiểu dữ liệu có cấu trúc nên con trỏ cũng có thể trỏ tới các biến mảng.

5.1 Con trỏ kiểu mảng

Khai báo:

```
Type m = array[1..5] of byte;
Var
ct1: ^m;
```

Với cách khai báo trên đây ct1 là biến con trỏ kiểu mảng, khi đó biến ct1^ sẽ gồm 5 phần tử, mỗi phần tử là một số kiểu Byte. Việc truy nhập vào biến ct1^ thực chất là truy nhập vào từng phần tử, ví dụ:

Read(ct1^[i]); hoặc Write(ct1^[i]); với 1<=i<=5.

5.2 Mảng các con trỏ

Khai báo:

```
        Var
ct: array[1..10] of  ^string;
s1,s2: String;
   Begin
  s1:='Ha noi Viet nam';
  s2:='Hppy New Year';
.....
```

Cách khai báo trên cho ta ct là mảng của 10 con trỏ, tất cả mười con trỏ này đều trỏ đến kiểu dữ liệu String. Mỗi con trỏ có thể trỏ đến một đối tượng khác nhau. Trong trường hợp này cách truy nhập dữ liệu cần phải thận trọng.

Nếu chúng ta chưa gán địa chỉ của bất kỳ đối tượng nào cho biến con trỏ mà chỉ thực hiện phép gán:

ct[i]^ := s1; với 1 <= i <= 10.

thì tất cả mười con trỏ đều trỏ tới biến s1.

Khi đó các lệnh

Write(ct[1]^); Write(ct[2]^); .... Write(ct[10]^); cho kết quả như nhau.

Trong trường hợp chúng ta gán dữ liệu từ một đối tượng cho nhiều biến con trỏ thì tất cả các con trỏ đều trỏ tới đối tượng được gán cuối cùng.

Nếu thực hiện phép gán

ct[1] := @s2;

nghĩa là gán địa chỉ của biến s2 vào con trỏ thứ nhất trong mảng thì chỉ có con trỏ ct[1] là trỏ tới biến s2, các con trỏ còn lại chưa trỏ vào đâu cả.

Xét ví dụ sau:

Ví dụ 4.7

```
Uses crt;
Type
   m = array[1..5] of byte;
Var
     i:byte; s1,s2:string;
   mct: array[1..10] of  ^string; ctm: ^m;
Begin
  clrscr;
  for i:=1 to 5 do
  begin
```

```
ctm^[i]:=i;
write(ctm^[i], ' ');
end;
writeln;
writeln(ctm^[3]);
s1:='Ha noi Viet nam';
s2:='Happy New Year';
mct[10]^:='aaaaa';
mct[4]^:='bbbbb';
mct[1]:=@s1;
mct[2]:=@s2;
writeln(mct[1]^);
writeln(mct[2]^);
writeln(mct[3]^);
writeln(mct[5]^);
readln;
end.
```

Chạy chương trình chúng ta nhận được kết quả:

```
1  2  3  4  5
3
Ha noi Viet nam
Happy New Year
bbbbb
bbbbb
```

Ví dụ 4.7 khai báo ctm là con trỏ kiểu mảng của các số nguyên, còn mct là mảng của các con trỏ kiểu chuỗi.

Với con trỏ kiểu mảng ctm chúng ta chỉ có một con trỏ, con trỏ này trỏ tới kiểu dữ liệu mảng m đã khai báo nên nó có 5 thành phần và chúng ta có thể truy nhập đến từng thành phần thông qua biến ctm^[i].

Biến mct cho ta một biến mảng mỗi thành phần mảng là một con trỏ và tất cả các con trỏ này đều trỏ tới cùng một kiểu dữ liệu.

Phép gán:

```
mct[10]^:='aaaaa';  mct[4]^:='bbbbb';
```

là gán dữ liệu trực tiếp vào hai biến mct[10]^ và mct[4]^, sau hai phép gán này cả mười con trỏ đều trỏ tới chuỗi 'bbbbb' chính vì vậy các lệnh:

```
writeln(mct[3]^);
writeln(mct[5]^);
```

đều cho kết quả là bbbbb.

Hai lệnh gán:

```
mct[1]:=@s1;
mct[2]:=@s2;
```

đã định lại hướng của con trỏ, con trỏ mct[1] trỏ tới biến s1, còn con trỏ mct[2] trỏ tới biến s2 vì vậy các lệnh:

```
writeln(mct[1]^);
writeln(mct[2]^);
```

cho kết quả là:

```
Ha noi Viet nam
Happy New Year
```

6. Cấp phát động

6.1 Quản lý vùng nhớ Heap

Với một biến con trỏ ct chúng ta có một biến ct^ tương ứng, đây không phải là biến tĩnh vì không được khai báo ở phần Var, nhưng cũng chưa phải là biến động. Muốn ct là một biến động thực sự thì phải dùng kỹ thuật cấp phát bộ nhớ động.

Cấp phát bộ nhớ động là việc cấp phát bộ nhớ được thực hiện bởi câu lệnh trong thân chương trình chứ không phải bằng cách khai báo biến hoặc tham số. Khi một biến được cấp phát bộ nhớ động thì nó trở thành biến động (Dynamic Variable). Vùng nhớ dành cho cấp phát động bao giờ cũng là vùng nhớ tự do Heap.

Theo mặc định khi một biến động được hình thành thì địa chỉ của nó được lưu trong biến con trỏ tương ứng.

Trước khi nghiên cứu cách thức cấp phát động chúng ta cần biết hệ thống quản lý vùng nhớ tự do như thế nào.

Vùng nhớ thấp được dành cho hệ điều hành, tiếp đó là vùng lưu mã chương trình (Code) vùng lưu các biến toàn cục (Data), vùng lưu các biến cục bộ (Stack). Pascal dùng toàn bộ vùng nhớ còn lại của máy PC cho vùng nhớ tự do (Heap).

Vùng Stack được thiết kế phát triển theo chiều đi xuống, nghĩa là các địa chỉ cao được sử dụng trước, địa chỉ thấp được sử dụng sau

Vùng Heap lại phát triển theo chiều đi lên, nghĩa là địa chỉ thấp được sử dụng trước, địa chỉ cao sử dụng sau.

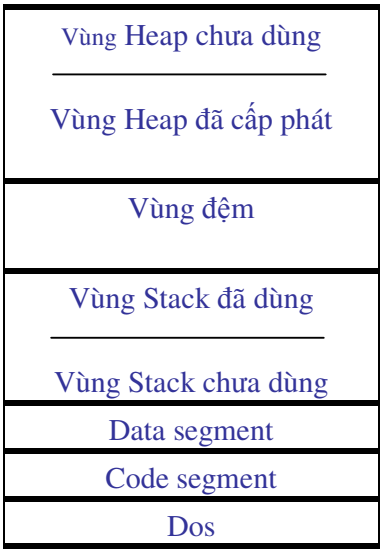
Toàn bộ vùng nhớ tự do Heap trong quá trình sử dụng bị chia thành nhiều khối với kích thước khác nhau. Do việc thu hồi vùng nhớ diễn ra thường xuyên trong chương trình nên các khối nhớ tự do còn lại có thể không nằm kề nhau.

Để quản lý Heap, Pascal có hai hàm là:

Hàm **MemAvail** cho biết tổng dung lượng còn được phép sử dụng trên Heap

Hàm **MaxAvail** cho biết dung lượng của khối lớn nhất còn tự do trên Heap.

Khi xin cấp phát động, điều quan trọng là cần biết kích thước khối nhớ lớn nhất còn tự do chứ không phải là tổng dung lượng tự do bởi vì mỗi đối tượng phải được lưu trữ trong một vùng nhớ liên tục.



Hình 4.1

Kích thước đối tượng dt được xác đinh bởi hàm Sizeof(dt), do vậy nếu:

MaxAvail < Sizeof(dt)

thì không thể xin cấp phát vùng nhớ trên Heap cho dt được, mặc dù tổng dung lượng trên Heap có thể vẫn còn lớn hơn kích thước dt nhiều lần.

Để biết địa chỉ của Heap bắt đầu và kết thúc từ đâu có thể sử dụng các biến không kiểu đã thiết kế trong Pascal :

Biến **HeapOgr**: cho địa chỉ điểm bắt đầu của Heap dùng cho cấp phát động.

Lệnh Write(Seg(heaporg^),' : ', ofs(heaporg^)); sẽ hiện địa chỉ này lên màn hình dưới dạng Segment:Ofset.

Biến **HeapEnd**: Cho địa chỉ điểm cuối của Heap được sử dụng cho đối tượng.

Khi chương trình bắt đầu được tải vào bộ nhớ thì các giá trị trên cũng được hệ thống khởi gán và giữ nguyên không thay đổi

Biến **HeapPtr**: địa chỉ đỉnh Heap, tức là địa chỉ đáy của vùng nhớ tự do còn được phép sử dụng.

Lúc đầu giá trị HeapPtr = Heaporg, sau đó giá trị HeapPtr sẽ phát triển theo chiều hướng lên.

6.2 Thủ tục cấp phát bộ nhớ cho con trỏ định kiểu

Với một con trỏ định kiểu ct thủ tục cấp phát bộ nhớ động sẽ là :

New(ct);

Thủ tục New(ct) cùng một lúc thực hiện các công việc sau:

* Cấp phát một vùng nhớ trên Heap với kích thước bằng kích thước kiểu dữ liệu mà con trỏ trỏ tới.

* Tạo một biến động định kiểu ct^ để có thể truy nhập vào vùng dữ liệu của đối tượng.

* Lưu địa chỉ của đối tượng vào con trỏ ct

Khi một biến động ct^ đã hết giá trị sử dụng thì cần thu hồi vùng nhớ đã cấp phát cho nó để dùng vào việc khác. Thủ tục thu hồi là

Dispose(ct);

Sau khi thu hồi vùng nhớ mặc dù biến ct^ vẫn còn tồn tại nhưng dữ liệu trong vùng nhớ đã dành cho ct sẽ không được bảo vệ, điều này có nghĩa là hệ thống có thể dùng vùng nhớ này vào việc khác. Ví dụ sau đây sẽ cho ta thấy rõ điều này.

Ví dụ 4.8

```
        Program cp_dong;
Uses crt;
Type s=string[30];
  Var
    ct:^s;
  Begin
    clrscr;
    new(ct);
    ct^:='Ha noi Viet nam';
    Writeln('Bien ct^ sau khi cap phat dong: ', ct^);
    Dispose(ct);
    Writeln('Bien ct^ sau thu tuc Dispose(ct) :  ', ct^);
    readln;
  End.
```

Chạy chương trình chúng ta nhận được kết quả:

Bien ct^ sau khi cap phat dong: Ha noi Viet nam

Bien ct^ sau thu tuc Dispose(ct) : Ha Viet nam

Có thể thấy rằng việc giải phóng vùng nhớ dành cho biến đã làm cho dữ liệu thay đổi mặc dù chúng ta chưa hề ra lệnh thay đổi nội dung vùng nhớ này.

6.3 Thủ tục cấp phát bộ nhớ cho con trỏ không định kiểu

Pascal có hai thủ tục cấp phát và thu hồi vùng nhớ cho con trỏ không định kiểu ct là:

Getmem(ct, n); cấp cho ct n Byte.

Freemem(ct, n); thu hồi n Byte vùng nhớ đã cấp cho ct.

Việc cấp phát không định kiểu thường được dùng trong trường hợp chưa biết trước kích thước của đối tượng, ví dụ để lưu một ảnh nằm trong khung chữ nhật toạ độ góc trên trái là x1, y1 và toạ độ góc dưới phải là x2, y2. Gọi n là kích thước ảnh tính bằng Byte, chúng ta xác định kích thước ảnh bằng hàm:

n := Imagesize(x1,y1,x2,y2);

Thủ tục Getmem(ct, n) sẽ cấp cho con trỏ không kiểu ct một vùng nhớ với kích thước đúng bằng kích thước ảnh, sau đó có thể lưu ảnh vào biến con trỏ thông qua thủ tục:

GetImage(x1,y1,x2,y2, ct^);

6.4 Thu hồi nhiều vùng nhớ

Để thu hồi một lúc nhiều vùng nhớ hoặc toàn bộ vùng Heap, chúng ta thực hiện các bước sau đây:

* Khai báo một biến con trỏ ct không kiểu để lưu trữ địa chỉ ô nhớ bắt đầu thu hồi (tức là đánh dấu vị trí để sau này thu hồi các vùng nhớ từ vị trí này).

* Đánh dấu vị trí sẽ bắt đầu thu hồi bởi thủ tục:

Mark(ct)

* Thu hồi vùng nhớ (tức là huỷ bỏ toàn bộ các biến động đã được cấp phát trong vùng nhớ kể từ khi đánh dấu)

Release(ct);

6.5 Các ví dụ về cấp phát động

Ví dụ 4.9

Ví dụ 4.9 thiết kế chương trình con Diachi nhằm thông báo kích thước vùng nhớ Heap mà hệ thống cấp cho Pascal, kích thước của khối nhớ lớn nhất có trong Heap và địa chỉ đỉnh Heap trước và sau khi cấp phát động.

Con trỏ không kiểu bd (bắt đầu) dùng để đánh dấu vị trị bắt đầu cấp phát vùng nhớ cho các biến động ct1 và ct2. Sau này khi thu hồi vùng nhớ chương trình sẽ bắt đầu thu hồi từ đây.

```
Program Quan_ly_Heap;
Uses crt;
Var bd:pointer;
    ct1,ct2:^string;
 Procedure diachi;
 Begin
     writeln('Kich thuoc toan bo vung nho Heap:       ',memavail);
     writeln('Kich thuoc khoi  lon nhat tren Heap: ',maxavail);
     writeln('Dia chi bat dau vung Heap:  ','$',seg(heaporg^),' : $',ofs(heaporg^));
     writeln('Dia chi ket thuc vung Heap:  ','$',seg(heapend^),': $',ofs(heapend^));
     writeln('Dia chi dinh Heap:  ','$',seg(heapptr^),' : $',ofs(heapptr^));
     writeln;writeln;
  end;
 Begin
     clrscr;
     writeln('Trang thai ban dau cua Heap');
```

```
  diachi;
  Mark(bd);
  new(ct1);
  ct1^:='aaaaaaaaaaaaa';
  writeln;
  writeln('Sau khi cap phat cho bien dong ct1^ ');
  diachi;
  New(ct2);
  ct2^:='qqq';
  writeln('Sau khi cap phat cho bien dong ct2^ ');
  diachi;
  release(bd);
  writeln('Sau khi thu hoi toan bo bien dong ');
  diachi;
  readln;
  End.
```

Ví dụ 4.10

Ví dụ 4.10 minh hoạ việc dùng con trỏ mảng để quản lý danh sách học sinh. Kiểu dữ liệu Nguoi bao gồm các trường Stt (số thứ tự), Hoten (họ và tên), Gioi (giới tính), Tong (tổng điểm), Xeploai (xếp loại). Con trỏ Dslop trỏ đến kiểu dữ liệu ds, vì ds là mảng của 100 phần tử kiểu Nguoi nên dslop là con trỏ mảng. Trong ví dụ chúng ta chỉ hình thành nên **một con trỏ** và con trỏ này có thể trỏ tới 100 phần tử của mảng. Thủ tục xin cấp phát động New(dslop) sẽ tạo nên một biến động kiểu mảng, phần thử thứ i của biến động này sẽ là dslop^[i] và chúng ta có thể hình dung dslop^[i] chính là bản ghi thứ i trong mảng ds. Ví dụ cũng nêu lên cách thức sử dụng toán tử With ... Do trong các bài toán có liên quan đến con trỏ.

Vì toán tử With ... Do chỉ có tác động đối với các biến kiểu Record nên chúng ta không thể đưa vào trong With ... Do các con trỏ mà chỉ có thể là các biến động.

```
  Program con_tro_mang;
  Uses crt;
  Type nguoi=record
     stt:byte;
     Hoten:string[25];
     Gioi:char;
     tong:real;
     xeploai:string[5];
  end;
     ds=array[1..100] of nguoi;
Var
     dslop:^ds; i,n:byte;
```

```
Begin
    clrscr;
    write('cho biet so hoc sinh can nhap: ');
    Readln(n);
    new(dslop);
    For i:=1 to n do
    with dslop^[i] do
    Begin
        writeln('So thu tu: ',i); stt:=i;
        write('Nhap Ho ten: '); readln(hoten);
        write('Nhap gioi T/G : '); readln(gioi);
        write('Nhap tong diem: '); readln(tong);
        write('Nhap xep loai: '); readln(xeploai);
        End;
        writeln('          DANH SACH HOC SINH');
        writeln('So TT : Ho va ten   :  Gioi : Tong diem : Xep loai ');
        For i:= 1 to n do
        With dslop^[i] do
            writeln(stt:3, hoten:15,'   ' ,gioi:4,'     ', Tong:5:2,'       ', xeploai);
            Readln;
        End.
```

Thay vì sử dụng con trỏ mảng, chúng ta sử dụng mảng các con trỏ. Trong ví dụ 4.11 Dslop là mảng của 100 con trỏ, các con trỏ này trỏ tới kiểu dữ liệu Nguoi. Biến không kiểu BD dùng để đánh dấu vị trí bắt đầu cấp phát động.

Thủ tục New(dslop[i]) trong vòng lặp For i:= 1 to n cho phép tạo nên n biến động dslop[i]^, các biến động này được cấp phát vùng nhớ trên Heap.

Ví dụ 4.11

```
Program mang_con_tro;
uses crt;
Type nguoi=record
        stt:byte;
        Hoten:string[25];
        Gioi:char;
        tong:real;
        xeploai:string[5];
        end;
Var
     bd:pointer;
```

```
    dslop:array[1..100] of ^nguoi; i,n:byte;
Begin
    clrscr;
    write('cho biet so hoc sinh can nhap: ');
     Readln(n);
     mark(bd);
     For i:= 1 to n do
      begin
         new(dslop[i]);
         with dslop[i]^ do
         begin
         writeln('So thu tu: ',i);stt:=i;
         write('Nhap Ho ten: '); readln(hoten);
         write('Nhap gioi T/G : '); readln(gioi);
         write('Nhap tong diem: '); readln(tong);
         write('Nhap xep loai: '); readln(xeploai);
         end;   { with}
       end;    {For}
writeln('So TT : Ho va ten   :  Gioi : Tong diem : Xep loai ');
For i:= 1 to n do
with dslop[i]^ do
writeln(stt:3, hoten:15,'   ', gioi:4,'  ', Tong:5:2,'    ', xeploai);
Readln;
END.
```

7. Danh sách liên kết và hàng đợi

Với con trỏ kiểu mảng đã nêu việc lập trình tạo danh sách cũng như duyệt danh sách khá đơn giản, hạn chế của kiểu mảng là phải khai báo trước kích thước mảng do đó nhiều khi dẫn tới không tiết kiệm bộ nhớ. Sử dụng biến động chúng ta có thể khắc phục được nhược điểm này bằng cách cần đến đâu thì tạo biến động đến đó. Số biến động tạo ra chỉ bị hạn chế bởi bộ nhớ trong (Ram) của máy PC mà cụ thể là phụ thuộc vào kích thước vùng Heap.

Danh sách được hiểu là một tập hợp hữu hạn các phần tử liên kết với nhau, trường hợp tổng quát nhất mỗi phần tử là một bản ghi. Điều đặc biệt của mỗi bản ghi trong danh sách là ngoài các trường dữ liệu, còn một trường dùng để liên kết và trường này lại là một con trỏ. Con trỏ này có nhiệm vụ trỏ vào địa chỉ của bản ghi kế tiếp. Nếu bản ghi hiện thời là bản ghi cuối cùng thì con trỏ sẽ trỏ vào Nil.

Như vậy xuất phát từ bản ghi đầu, lần theo địa chỉ lưu ở trường con trỏ chúng ta có thể truy nhập vào bản ghi tiếp theo, quá trình sẽ tiếp diễn cho đến bản ghi cuối cùng.

Một danh sách chưa có phần tử nào được gọi là danh sách rỗng. Việc thêm một phần tử vào danh sách (nghĩa là tạo nên danh sách) có thể rơi vào một trong ba khả năng:

a. Phần tử mới được thêm vào đầu danh sách

b. Phần tử mới được nối vào cuối danh sách

c. Phần tử mới được chèn vào một vị trí xác định.

Trường hợp a chúng ta có danh sách liên kết ngược (LIFO), còn trường hợp b chúng ta có danh sách liên kết thuận (FIFO) hay còn gọi là hàng đợi QUEUE.

7.1 Danh sách liên kết ngược

Là loại danh sách mà trường liên kết của phần tử tạo ra sau luôn trỏ vào phần tử tạo ra trước đó. Trường liên kết của phần tử tạo ra đầu tiên trỏ vào Nil. Điều này dẫn tới việc khi kết xuất thông tin ra chúng ta phải bắt đầu từ phần tử tạo ra cuối cùng vì chỉ có như vậy chúng ta mới biết địa chỉ của phần tử tạo ra trước đó. Nếu cố tình đi từ phần tử tạo ra dầu tiên thì chúng ta không thể biết phần tử tiếp theo là phần tử nào. Liên kết kiểu này gọi là kiểu LIFO (Last In - Firt Out) hay còn gọi là kiểu xếp chồng. Phần tử nhập vào cuối cùng sẽ được lấy ra đầu tiên. Danh sách tạo ra theo kiểu này được gọi là danh sách liên kết ngược.

Giả thiết rằng chúng ta cần xây dựng một danh sách học sinh với các trường dữ liệu là Mhs (mã hồ sơ), Hoten (Họ và tên), Diem (điểm tổng kết) , trường liên kết lấy tên là Tiep (tiếp tục). Kiểu dữ liệu bản ghi với các trường nêu trên lấy tên là Nguoi. Để tạo ra danh sách học sinh chúng ta cần tạo ra một kiểu con trỏ DS trỏ vào kiểu dữ liệu Nguoi và trường liên kết Tiep trong bản ghi Nguoi sẽ trỏ vào kiểu dữ liệu Ds.

Hình 4.2

```
Type
   Ds = ^nguoi;
   Nguoi = Record
   Mhs:byte;
  Hoten: string[20];
   Diem:real;
   Tiep:  Ds;
  End;
```

Chú ý:

Với cách thức nói trên nếu chúng ta khai báo kiểu bản ghi Nguoi trước khi khai báo kiểu con trỏ Ds thì sẽ bị lỗi vì khi đó con trỏ Tiep sẽ trỏ vào một kiểu dữ liệu chưa được định nghĩa.

Sau khi khai báo kiểu dữ liệu cần khai báo biến con trỏ Dslop để lưu trữ dữ liệu nhập vào và biến Ctcuoi (con trỏ cuối) để trỏ vào phần tử cuối cùng.

Vấn đề là làm thế nào để con trỏ liên kết Tiep luôn trỏ vào phần tử tạo ra trước đó. Để làm việc này chúng ta tạo ra biến động Dslop để lưu trữ dữ liệu. Cứ mỗi bản ghi cần nhập vào thì tạo ra một biến động Dslop mới. Địa chỉ của biến động Dslop luôn được gán cho con trỏ cuối Ctcuoi. Dưới đây là đoạn mô phỏng chương trình tạo danh sách liên kết ngược:

```
Ctcuoi:=nil;  {khởi tạo danh sách}
Bắt đầu lặp
      New(dslop);   {tạo biến động lưu trữ dữ liệu nhập vào}
      Nhập dữ liệu;  {nhập dữ liệu cho phần tử thứ i}
      Tiep:=ctcuoi;   {trường liên kết của phần tử thứ i trỏ vào địa chỉ
                        của con trỏ cuối ctcuoi }
      Ctcuoi :=dslop;  {hướng con trỏ cuối vào bản ghi hiện thời}
Kết thúc lặp
```

Nhận xét:

*** Vòng lặp thứ 1:**

- Tạo biến động Dslop sau đó nhập dữ liệu vào phần tử đầu tiên.

- Câu lệnh Tiep:= ctcuoi; sẽ hướng con trỏ liên kết Tiep của bản ghi này trỏ vào Nil vì lúc này con trỏ cuối đang là Nil.

- Câu lệnh ctcuoi:=dslop; sẽ hướng con trỏ cuối đến phần tử thứ nhất.

*** Vòng lặp thứ 2:**

-Tạo biến động Dslop lần thứ hai, nhập dữ liệu cho phần tử thứ hai. Câu lệnh Tiep:= ctcuoi; sẽ hướng trường liên kết của phần tử thứ hai đến phần phần tử thứ nhất vì lúc này ctcuoi đang mang địa chỉ của phần tử thứ nhất.

- Câu lệnh ctcuoi:=dslop; sẽ chuyển hướng con trỏ cuối sang phần tử thứ hai.

- Có thể dễ dàng xuy ra rằng nếu vòng lặp thực hiện lần thứ ba thì con trỏ liên kết của phần tử thứ ba sẽ trỏ vào phần tử thứ hai, còn con trỏ cuối sẽ trỏ vào phần tử thứ ba...

Quá trình sẽ lặp lại cho đến khi kết thúc.

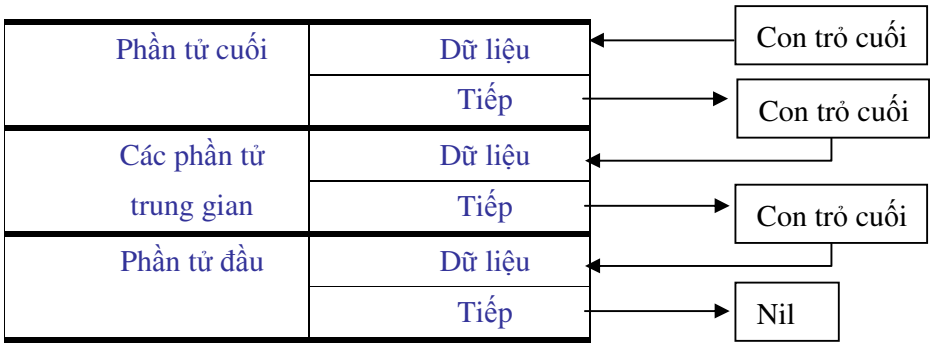
Dưới đây là ví dụ xây dựng danh sách, chương trình con Hien_LIFO cho hiện dữ liệu lên màn hình theo chiều ngược, phần tử nhập sau hiện trước.

Ví dụ 4.12

```
Program danh_sach_lien_ket_nguoc;
Uses crt;
Type   ds= ^nguoi;
       nguoi=record
       Mhs:byte;  Hoten:string[25];    Diem:real;    Tiep:ds;     End;
```

```
Var
        dslop, ctcuoi:ds; i,j:byte;
        lam:char;
  Procedure Hien_LIFO; {Thu tuc hien du lieu tu cuoi ve dau}
  var ct1:ds;
  Begin
     clrscr;
     writeln('Du lieu da nhap - Hien tu cuoi ve dau');
     writeln;
     ct1:=ctcuoi;
     while ct1<>nil do
     with ct1^ do
     Begin
        Write(Mhs,'  ', hoten);
        for j:=1 to (20-length(hoten)) do write(' ');
        writeln(diem:5:2);
        ct1:=tiep;
     end;
  End;

  Begin  {Than chuong trinh chinh}
     clrscr;
     ctcuoi:=nil; i:=0;
     Repeat
     New(dslop);  i:=i+1;
     With dslop^ do
     Begin
        writeln('Ma ho so so: ',i); mhs:=i;
        write('Ho va ten: '); readln(hoten);
        write('Diem      : '); Readln(diem);
        tiep:=ctcuoi;
        ctcuoi:=dslop;
        writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
        writeln;
     end;
     Until lam in ['k','K'];
     Hien_lifo;
     Readln;
END.
```

Trong ví dụ trên thay vì trường Tiep của phần tử hiện thời trỏ trực tiếp vào phần tử đứng trước chúng ta cho nó trỏ vào "con trỏ cuối" rồi "con trỏ cuối" sẽ trỏ vào phần tử kế tiếp. Mô hình mô tả quá trình nhập dữ liệu như sau:



Hình 4.3

7.2 Hàng đợi Queue - Danh sách liên kết thuận

Với loại danh sách mà phần tử nào nhập trước thì được lấy ra trước chúng ta gọi là danh sách liên kết thuận, nó cũng giống như hàng đợi người nào đến trước thì được gọi trước.

Để tạo ra hàng đợi ngoài ctcuoi chúng ta phải thêm vào con trỏ đầu (ctdau). Con trỏ cuối ctcuoi luôn trỏ vào phần tử cuối cùng, còn con trỏ đầu lại luôn trỏ vào phần tử đầu của danh sách. Dưới đây mô phỏng thuật toán chương trình xây dựng Queue.

```
Ctdau:=nil;  {khởi tạo danh sách}
Bắt đầu lặp
    New(dslop);   {tạo biến động lưu trữ dữ liệu }
    Nhập dữ liệu; {nhập dữ liệu cho phần thử thứ i, i=1,2... }
    Nếu Ctdau = Nil thì  ctdau :=dslop;
        Còn nếu Ctdau <> Nil  thì ctcuoi^.tiep := dslop;
    Ctcuoi := dslop;
    Ctcuoi^.tiep := Nil;
     Kết thúc lặp
```

Chương trình mô phỏng trên cho ta kết quả sau:

*** Vòng lặp thứ nhất:**

- Nhập dữ liệu cho phần tử thứ nhất
- Ctdau trỏ vào phần tử thứ nhất
- Ctcuoi trỏ vào phần tử thứ nhất
- Trường liên kết Tiep trỏ vào cuối danh sách (Nil), vì Ctcuoi đang trỏ vào phần tử thứ nhất nên trường Tiep của Ctcuoi cũng chính là trường Tiep của phần tử thứ nhất.

*** Vòng lặp thứ hai**

- Nhập dữ liệu cho phần tử thứ hai

- Vì ctdau đang trỏ vào phần thử thứ nhất (ctdau<>Nil) nên chuyển hướng trường Tiep của ctcuoi đến phần tử thứ hai. Vì Ctcuoi đang trỏ vào phần tử thứ nhất nên điều này cũng có nghĩa là trường Tiep của phần tử thứ nhất trỏ vào phần tử thứ hai (không trỏ vào Nil nữa).

- Xác nhận lại rằng ctcuoi trỏ vào phần tử thứ hai (ctdau vẫn trỏ vào phần tử thứ nhất).

- Trường liên kết Tiep trỏ vào cuối danh sách (Nil), vì Ctcuoi đang trỏ vào phần tử thứ hai nên trường Tiep của Ctcuoi cũng chính là trường Tiep của phần tử thứ hai.

* Vòng lặp tiếp theo sẽ tương tự như vòng lặp thứ hai

Dưới đây là chương trình xây dựng hàng đợi, chương trình con Hien_FIFO cho hiện dữ liệu lên màn hình theo đúng thứ tự nhập vào.

Ví dụ 4.13

```
Program danh_sach_lien_ket_thuan;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
  Mhs: Byte;  Hoten:string[25];    Diem:real;    tiep:ds;
  End;
Var
dslop, ctdau, ctcuoi:ds; bd:pointer;
lam:char; i,j:byte;

  Procedure Hien_FIFO; {Hien du lieu tu dau xuong cuoi}
   Var ct1:ds;
  Begin
   Clrscr;
   writeln('Du lieu da nhap - Hien tu dau xuong cuoi');
   writeln;
   ct1:=ctdau;
   while ct1<>nil do
        with ct1^ do
        Begin
            Write(Mhs,'  ',hoten);
             for j:=1 to (20-length(hoten)) do write(' ');
            writeln(diem:5:2);
            ct1:=tiep;
        End;
  End;
```

```
Begin  {Than chuong trinh chinh}
clrscr;
mark(bd);
ctdau:=nil;  i:=0;
 Repeat
   New(dslop);  {tạo biến động Dslop}
   i:=i+1;
   With dslop^ do
    Begin
        Witeln('Ma ho so so: ',i); mhs:=i;
        Write('Ho va ten: '); readln(hoten);  {nhập dữ liệu}
        Write('Diem      : '); Readln(diem);
         if ctdau=nil then
            ctdau:=dslop   {ctdau trỏ vào phần tử thứ 1 }
        else
            ctcuoi^.tiep:=dslop; {Lưu trữ địa chỉ của phần tử hiện thời - kể từ phần tử
                                  thứ hai vào trường liên kết Tiep của Ctcuoi }
            ctcuoi:=dslop;          {ghi nhận lại con trỏ cuối, nghĩa là Tiep đang trỏ
                                  vào phần tử hiện thời}
            ctcuoi^.tiep:=nil;      { trường liên kết Tiep của con trỏ cuối trỏ vào Nil
                                  - ket thuc danh sach}
Writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
writeln;
End;
Until lam in ['k','K'];
Hien_FiFO;
Release(bd);
End.
```

7.3 Chèn thêm phần tử vào danh sách

Việc đầu tiên khi muốn chèn thêm phần tử vào danh sách là phải xác định được chính xác vị trí cần chèn, muốn vậy danh sách phải có một trường khoá, mỗi phần tử trong danh sách sẽ ứng với một và chỉ một giá trị của trường khoá. Ví dụ có thể lấy trường số thứ tự (STT) hoặc mã hồ sơ (MHS) làm trường khoá, không thể dùng trường Hoten để làm khoá vì trong danh sách có thể có nhiều người trùng Hoten.

Quá trình chèn một phần tử vào danh sách sẽ qua các bước:

* Xác định vị trí chèn

* Tạo một biến động (xem như một con trỏ nháp) và xin cấp phát vùng nhớ cho biến động để lưu dữ liệu sẽ chèn vào danh sách.

* Chuyển trường Tiep của phần tử hiện thời đến phần tử bổ xung

* Chuyển trường Tiep của phần tử bổ xung đến phần tử trước phần tử hiện thời.

```
New(ct1);     {tạo phần tử nháp để tạm lưu dữ liệu}
With ct1^ do   Nhập dữ liệu cho phần phần tử bổ xung
dslop:=ctcuoi;  {hướng con trỏ đến phần tử cuối cùng trong danh sách}
While (dslop<>nil) and (dslop^.mhs <> n) do
 dslop:=dslop^.tiep;  {hướng con trỏ đến vị trí cần chèn}
ct1^.tiep:=dslop^.tiep;  {gán trường liên kết  Tiep của phần tử hiện thời cho Tiep của
ct1 - nghĩa là chèn ct1 vào trước phần tử hiện thời}
dslop^.tiep:=ct1;  {trường liên kết Tiep của bản ghi hiện thời trỏ vào ct1 - nghĩa là lùi
phần tử hiện thời ra sau ct1 }
```

7.4 Xoá một phần tử khỏi danh sách

Quá trình xoá một phần tử trong danh sách không phải là quá trình làm rỗng ô nhớ chứa phần tử mà đơn giản chỉ là chuyển hướng trường liên kết không trỏ vào phần tử đó nữa. Nếu chúng ta xoá nhiều phần tử thì bộ nhớ sẽ bị phân mảnh ra nhiều đoạn ngắt quãng, trong trường hợp này cần bố trí lại các ô nhớ để một đối tượng được bố trí trên một số ô nhớ liên tục (xem ví dụ 4.14).

Ví dụ 4.14

```
Program danh_sach_lien_ket_nguoc;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
  Mhs:byte;
  Hoten:string[25];
  Diem:real;
  tiep:ds;
  End;
Var
dslop, ctcuoi:ds; i,j,n:byte;
lam:char;

  Procedure Hien_LIFO; {Thu tuc hien du lieu tu cuoi ve dau}
  var ct1:ds;
  Begin
  clrscr;
  writeln('Du lieu da nhap - Hien tu cuoi ve dau');
  writeln;
  ct1:=ctcuoi;
  while ct1<>nil do
    with ct1^ do
```

```
  Begin
      Write(mhs,'  ',hoten);
      for j:=1 to (20-length(hoten)) do write(' ');
      writeln(diem:5:2);
      ct1:=tiep;
  end;
 Readln;
 End;
{***************}
Procedure Chen;
Var n:byte; ct1:ds;
Begin
Clrscr;
writeln('Chen truoc ma ho so nao? '); Readln(n);
New(ct1);
With ct1^ do
Begin
   Writeln('Ma ho so so: ',n-1);
   Write('Ho va ten: '); readln(hoten);
   Write('Diem      : '); Readln(diem);
   End;
   dslop:=ctcuoi;
   While (dslop<>nil) and (dslop^.mhs <> n) do dslop:=dslop^.tiep;
   ct1^.tiep:=dslop^.tiep;
   dslop^.tiep:=ct1;
End;
{***************}
Procedure Xoa;
Var n:byte; ct1:ds;
Begin
Write('Cho biet ma ho so can xoa '); readln(n);
dslop:=ctcuoi;
While (dslop<>nil) and (dslop^.mhs<>n) do
   Begin
      ct1:=dslop;      dslop:=dslop^.tiep;
   End;
   If dslop=ctcuoi then ctcuoi:=dslop^.tiep
   Else
      ct1^.tiep:=dslop^.tiep;
End;
```

```
  Procedure Nhap;
    Var lam:char;
  Begin
  clrscr;
  ctcuoi:=nil;
  Repeat
   New(dslop); i:=i+1;
   With dslop^ do
   Begin
   Writeln('Ma ho so so: ',i); Mhs:=i;
   Write('Ho va ten: '); readln(hoten);
   Write('Diem      : '); Readln(diem);
   tiep:=ctcuoi;
   ctcuoi:=dslop;
   Writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
   writeln;
   End;
   Until lam in ['k','K'];
  End;

  Begin  {Than chuong trinh chinh}
  Repeat
  clrscr;
  Writeln('1: Nhap du lieu');
  Writeln('2: Hien_lifo   ');
  Writeln('3: Chen them phan tu');
  Writeln('4: Xoa phan tu ');
  Write('Xin moi chon cong viec! Bam 99 de ket thuc '); readln(n);
  Case n of
  1:Nhap;
  2:Hien_lifo;
  3:Chen;
  4:Xoa;
  End;
  Until n=99;
End.
```

Ví dụ 4.15

Danh sách liên kết thuận

```
Program danh_sach_lien_ket_thuan;
Uses crt;
Type
ds= ^nguoi;
nguoi=record
mhs:byte;
  Hoten:string[25];
  Diem:real;
  tiep:ds;
  End;
Var
dslop, ctdau, ctcuoi:ds;
lam:char; i,j,n:byte;

  Procedure Hien_FIFO; {Hien du lieu tu dau xuong cuoi}
  var ct1:ds;
  Begin
    clrscr;
    ct1:=ctdau;
    writeln('Du lieu da nhap - Hien tu dau xuong cuoi');
    writeln;
    ct1:=ctdau;
    while ct1<>nil do
    with ct1^ do
    Begin
          Write(mhs,'  ',hoten);
           for i:=1 to (20-length(hoten)) do write(' ');
          writeln(diem:5:2);
          ct1:=tiep;
    end;
    readln;
    End;

  {*****************}
```

```
 Procedure Nhap;
 Begin
   clrscr;
 ctdau:=nil; j:=0;
 Repeat
  New(dslop);    j:=j+1;
  With dslop^ do
   Begin
      writeln('Ma ho so: ',j); mhs:=j;
      write('Ho va ten: '); readln(hoten);
      write('Diem      : '); Readln(diem);
      if ctdau=nil then
         ctdau:=dslop   {ctdau tro vao phan tu 1 }
         else
         ctcuoi^.tiep:=dslop; {truong lien ket Tiep tro vao phan tu tiep theo }
         ctcuoi:=dslop;  {ghi nhan lai con tro cuoi nghia la Tiep dang tro
                          vao phan tu tiep theo}
         ctcuoi^.tiep:=nil;      {truong lien ket tro vao Nil - ket thuc danh sach}
         writeln('Nhap tiep hay thoi? C/K '); lam:=readkey;
         writeln;
      End;
      Until lam in ['k','K'];
End;

  {***************}

 Procedure Chen;
 Var n:byte; ct1:ds;
 Begin
    Clrscr;
    writeln('Chen truoc ma ho so nao? '); Readln(n);
    New(ct1);
    with ct1^ do
       Begin
          writeln('Ma ho so so: ',n-1);
          write('Ho va ten: '); readln(hoten);
          write('Diem      : '); Readln(diem);
       End;
    dslop:=ctdau;
While (dslop<>nil) and (dslop^.mhs <> n) do dslop:=dslop^.tiep;
```

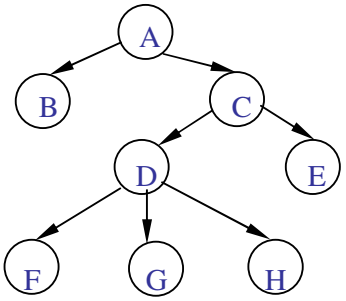
```
ct1^.tiep:=dslop^.tiep;
dslop^.tiep:=ct1;
End;

 {***************}

    Procedure Xoa;
Var n:byte; ct1:ds;
Begin
    write('Cho biet ma ho so can xoa '); readln(n);
    dslop:=ctdau;
    while (dslop<>nil) and (dslop^.mhs<>n) do
        Begin
            ct1:=dslop;
            dslop:=dslop^.tiep;
        End;
        If dslop=ctdau then ctdau:=dslop^.tiep
        Else
        ct1^.tiep:=dslop^.tiep;
        End;

{***************}

 Begin   {Thân chương trình}
 Repeat
    clrscr;
    writeln('1: Nhap du lieu');
    writeln('2: Hien_Fifo   ');
    writeln('3: Chen them phan tu');
    writeln('4: Xoa phan tu ');
    write('Xin moi chon cong viec! Bam 99 de ket thuc '); readln(n);
    Case n of
    1:Nhap;
    2:Hien_Fifo;
    3:Chen;
    4:Xoa;
    End;
    Until n=99;
END.
```

8. Cây nhị phân

Chúng ta đã làm quen với nhiều kiểu dữ liệu có cấu trúc như mảng, tệp, bản ghi, … Dữ liệu kiểu cây cũng là một kiểu dữ liệu có cấu trúc được xây dựng trong Pascal.

8.1 Khái niệm Cây (Tree)

Cây gồm một tập hợp hữu hạn các phần tử gọi là nút (hay là đỉnh) và một tập hợp hữu hạn các cạnh có hướng nối từng cặp nút với nhau. Trong cây có một nút đặc biệt gọi là Gốc, giữa các nút có một quan hệ phân cấp gọi là quan hệ cha con.



Hình 4.4

* Nút được gọi là Gốc khi từ nút đó chỉ có các cạnh có hướng đi khỏi nút.

* Số nút con của một nút nào đó gọi là bậc tự do (Degree) hay còn gọi là cấp của nút đó. Trong hình vẽ trên nút A và C có bậc tự do bằng 2, còn nút D có bậc tự do bằng 3.

* Nút có cấp bằng 0 (tức là không có nút con) gọi là nút lá, nút phân bổ giữa gốc và lá là nút nhánh.

* Mức của một nút là một số xác định vị trí của nút trong cây. Gốc của một cây được coi là ở mức 1, mức của một nút con nào đó bằng mức của nút cha cộng với 1.

* Chiều cao của cây bằng số mức lớn nhất của nút có trên cây.

Hình vẽ bên thể hiện một cây, nút A là gốc cây, các nút B, F, G, H, E là nút lá, còn các nút C, D là nút nhánh, chiều cao của cây là bằng 4 vì các nút F, G, H có mức là 4.

Cây nhị phân là trường hợp đặc biệt của cây mà bậc tự do của mỗi nút lớn nhất là bằng hai, nói cách khác mỗi nút cha chỉ có nhiều nhất là hai nút con.

Nếu chúng ta xây dựng các cây mà mỗi nút cha có 10 hoặc 16 nút con thì ta sẽ có cây thập phân hoặc cây thập lục phân.

2. Cây nhị phân

Như đã nêu, cây nhị phân là loại cây đặc biệt mà mỗi nút cha chỉ có nhiều nhất là 2 nút con, các nút con này được gọi là con bên trái (CBT) và con bên phải (CBP). Mô hình cây được vẽ từ trên xuống, gốc ở trên cùng, lá ở dưới. Dữ liệu trong mỗi nút có thể là số hoặc ký

tự và việc lưu trữ dữ liệu được quy định như sau: Nếu dữ liệu trong nút con lớn hơn dữ liệu trong nút cha thì lưu vào con bên phải, ngược lại lưu vào con bên trái.

Tuỳ thuộc vào dữ liệu lưu trữ tại các nút chúng ta có thể có một trong ba dạng cây sau đây:

a. Cây tự nhiên

Cây tự nhiên là loại cây mà cấp của các nút nằm trong miền (0..2)

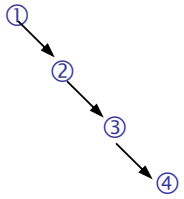
b. Cây đối xứng

Cây đối xứng là loại cây mà cấp của tất cả các nút đều bằng 2.

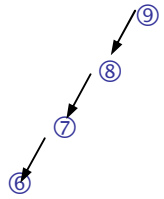
c. Cây một phía hay còn gọi là cây mọc lệch

Cây một phía là loại cây mà cấp của tất cả các nút đều bằng 1.

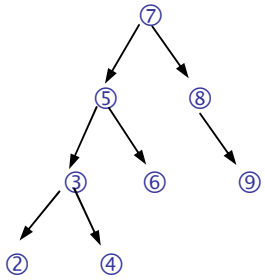
Hình 4.5 dưới đây mô tả các loại cây đã nêu:



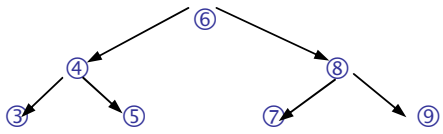
a. Cây lệch phải



b. Cây lệch trái



d. Cây tự nhiên



e. Cây đối xứng

Hình 4.5

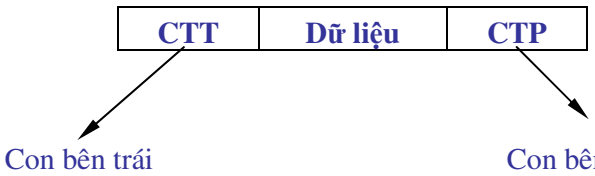
Cây được vẽ theo chiều từ trên xuống dưới với nút gốc ở trên và nút lá ở dưới. Một nút mà từ đó có thể truy nhập đến các nút khác thì được gọi là nút cha của các nút đó, còn các nút được truy nhập gọi là nút con.

Cây nhị phân có thể cài đặt bằng cách dùng các cấu trúc liên kết đã nêu trên. Dữ liệu trên một nút được lưu trữ bằng bản ghi, tại mỗi nút có hai con trỏ, CTT và CTP. CTT dùng để trỏ đến con bên trái và CTP dùng để chỉ đến con bên phải. Ví dụ dưới đây nêu cách cài đặt cây nhị phân, dữ liệu tại mỗi nút là một ký tự của bảng mã ASCII. Cần chú ý rằng theo quy ước thì các ký tự có thứ tự là:

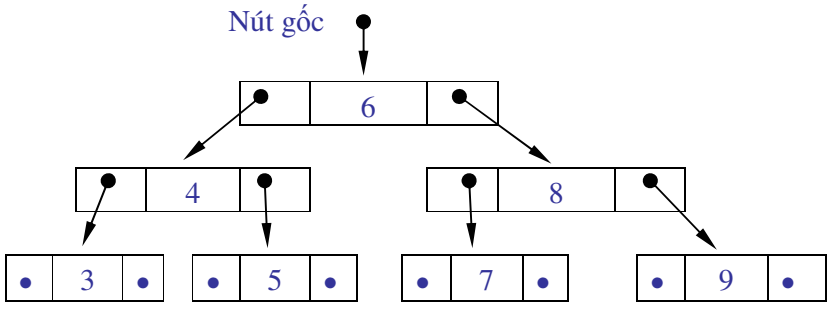
a < b < c < ..... < z và **A < B < C < ..... < Z**

```
Type
ct=^Kytu;
Kytu = record
    Nut : char;
    Ctt,Ctp : ct;
    end;
Var  chucai : char;  tim : boolean;
     goc, ct1 : ct;   batdau : pointer;
```

Ví dụ trên đã khai báo kiểu dữ liệu con trỏ đặt tên là **Kytu,** đây là kiểu bản ghi với mục đích lưu trữ một ký tự của bảng mã, con trỏ CT dùng để trỏ vào kiểu dữ liệu Kytu. Bên trong kiểu Kytu có các con trỏ **Ctt** và **Ctp** dùng để trỏ vào các nút biểu diễn con bên trái và con bên phải của nút cha.



Như vậy cây nhị phân trong hình 4.5e có thể được biểu diễn như là cây liên kết của các bản ghi dưới đây

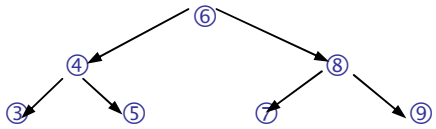


Hình 4.6

Trong hình 4.6 các nút chứa các giá trị 3, 5, 7, 9 là nút lá, chúng không có con bên trái hoặc con bên phải. Các con trỏ trái CTT và con trỏ phải CTP của chúng đều trỏ vào Nil

Một nút cha có thể có một hoặc hai cây con và bài toán xử lý cây nhị phân là thăm một nút, xử lý dữ liệu tại nút đó rồi chuyển sang thăm nút khác. Vấn đề phức tạp là ở chỗ mỗi nút đều có hai con trỏ do vậy khi di chuyển sang nút khác chúng ta chọn hướng đi nào, không những thế khi đến một nút chúng ta xử lý dữ liệu trước rồi mới đi thăm các nút con hay thăm nút con trước rồi xử lý dữ liệu tại nút cha sau? Tuỳ thuộc vào cách thức thăm và xử lý chúng ta sẽ có các kết quả khác nhau.

Dưới đây là một ví dụ minh hoạ.



Hình 4.7

Chúng ta gọi thao tác quét một cây nhị phân là việc thăm mỗi nút đúng một lần, không sót một nút nào và thông tin tại mỗi nút được xử lý cũng chỉ một lần.

Giả sử với cây nhị phân trên hình 4.7, chúng ta thực hiện quá trình quét như sau:

1. Thăm nút gốc và xử lý dữ liệu, cụ thể là hiện giá trị có tại nút
2. Quét cây con bên trái
3. Quét cây con bên phải

Với trình tự này kết quả nhận được sẽ là 6 4 3 5 8 7 9

Để tổng quát chúng ta ký hiệu các bước của tiến trình quét cây như sau:

G: xử lý dữ liệu tại nút (tức là gốc của các cây mẹ hoặc cây con)

T: quét cây con bên trái

P: quét cây con bên phải

Như vậy có tất cả 6 cách quét khác nhau:

TGP; GTP; TPG; GPT; PGT; PTG

Có thể nhận thấy rằng đối với việc quét một cây điều quan trọng không phải là xử lý dữ liệu tại các nút mà là việc xử lý ấy được thực hiện khi nào. Trên cùng một cây với mỗi trình tự thực hiện chúng ta nhận được một kết quả và các kết quả này là không giống nhau.

Trong 6 cách trên, ba cách đầu (cây con bên trái được quét trước cây con bên phải) là ba cách được quan tâm nhiều hơn cả. Căn cứ vào thời điểm xử lý dữ liệu tại nút (bước G), chúng ta đặt cho ba cách quét này các tên như sau:

TGP: quét trung tâm (Inorder)

GTP: quét trước (Preorder)

TPG: quét sau (Postorder)

Với cây hình 4.7 nếu xử dụng cách quét TGP chúng ta có kết quả như sau:

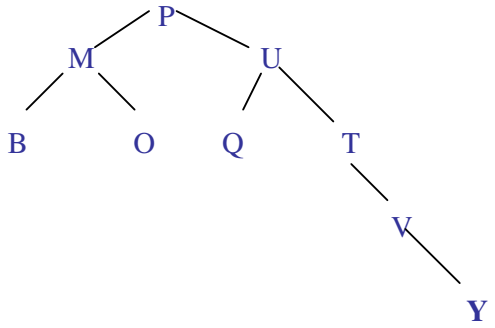
3 4 5 6 7 8 9

Sở dĩ có kết quả như trên là vì chúng ta đã quét theo nguyên tắc: quét cây con bên trái, xử lý dữ liệu ở gốc rồi sau đó quét cây con bên phải. Vì số 6 nằm ở gốc nên nó chưa được xử lý mà phải chuyển sang cây bên trái. Với cây con bên trái, số 4 nằm ở gốc nên lại phải theo nguyên tắc TGP nghĩa là sang bên trái tiếp. Số 3 nằm ở nút lá nên không còn đi đâu nữa kết quả là đầu ra ta có số 3. Quay về G ta có số 4 sau đó sang phải ta có số 5. Xong cây con bên trái ta quay về G và có số 6, tiếp tục với cây con bên phải ta có kết quả đã nêu.

Cách quét TGP sẽ cho ta kết quả là một dãy sắp xếp theo chiều tăng dần, điều này không những đúng với số mà cũng đúng với cây nhị phân chứa các ký tự tại mỗi nút, chúng ta sẽ nghiên cứu kỹ điều này qua ví dụ 4.16.

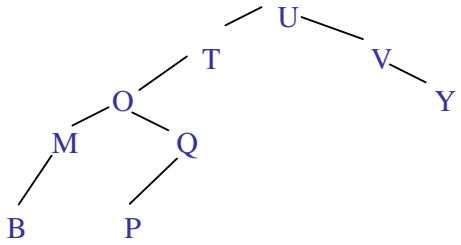
Khi xây dựng cây nhị phân, dữ liệu tại nút gốc mang một ý nghĩa quan trọng, nó sẽ quyết định dữ liệu nhập tiếp theo sẽ nằm ở cây con bên trái hay cây con bên phải.

Giả sử với một tập các ký tự : P, M, U, B, O, Q, T, V, Y nếu chúng ta tạo cây bằng cách đặt ký tự P vào nút gốc và nhập theo thú tự trên thì cây sẽ có dạng hình 4. 8



Hình 4.8

Còn nếu chúng ta đặt U vào gốc và nhập dữ liệu theo thứ tự U, T, V, O, M, Q, Y, B, P thì cây sẽ có dạng như trong hình 4.9



Hình 4.9

Ví dụ 4.16

```
Program  Caynhiphan;
Uses crt;
Type
ct=^kytu;
kytu = record
       Nut:char;
       ctt,ctp:ct;
       end;
Var   chucai:char;   goc:ct; i,j:byte;
        A:array[1..100] of char;
    ch:char;

Procedure Taocay (Var tg: ct);
{Chuong trinh con kiem tra xem ky tu dua vao da co trong cay chua
neu chua thi chen vao}
Begin
If tg=nil then
 Begin
  New(tg);
  With tg^ do
   Begin
    Nut:= chucai;
    ctt:=nil; ctp:=nil;
   End;
 End
 Else
  With tg^ do
  Begin
   if chucai<nut then taocay(ctt)
   else
   if chucai>nut then taocay(ctp)
   else
    Begin
     textcolor(red);
     Writeln('Ky tu ',chucai,' da co trong cay-Hay nhap tiep');
    end;
  End;
End;
```

```
Procedure quet_trung_tam(contro:ct);
Begin
if contro<>nil then
  Begin
  quet_trung_tam(contro^.ctt);
  write(contro^.nut,'  ');
  quet_trung_tam(contro^.ctp);
  end;
End;

Procedure quet_truoc(contro:ct);
Begin
if contro<>nil then
  Begin
  write(contro^.nut,'  ');
  quet_truoc(contro^.ctt);
  quet_truoc(contro^.ctp);
  end;
End;

Procedure quet_sau(contro:ct);
Begin
if contro<>nil then
  Begin
  quet_sau(contro^.ctt);
  quet_sau(contro^.ctp);
  write(contro^.nut,'  ');
  end;
End;

BEGIN
   Clrscr;
   writeln('Xay dung cay nhi phan cac ky tu');
   goc:=nil; i:=1; taocay(goc);
Repeat
   textcolor(14);
   write('Nhap ky tu vao nut - Bam so 9 de dung ');
   readln(chucai); chucai:=upcase(chucai);
   if (ord(chucai)>=65) and (ord(chucai)<=90) then
   Begin
```

```
    taocay(goc);
    a[i]:=chucai;
    i:=i+1;
    end;
  Until (Ord(chucai)<65) or (ord(chucai)>90);
  writeln;
  textcolor(14);
  writeln('Du lieu nhap ban dau');
  for j:= 1 to i do write(a[j],'  ');
  writeln;
  writeln('Du lieu hien theo cach quet Trung tam TGP');
  quet_trung_tam(goc);
  writeln;
  writeln('Du lieu hien theo cach quet Truoc GTP');
  quet_truoc(goc);
  writeln;
  writeln('Du lieu hien theo cach quet Sau TPG');
  quet_sau(goc);
  readln;
 END.
```

Chương trình con Taocay trong ví dụ 4.6 tạo nên một cây nhị phân mà dữ liệu tại các nút là các ký tự của bảng mã ASCII, khi bấm một phím bất kỳ trên bàn phím không phải là một trong các chữ cái chương trình tạo cây sẽ kết thúc. Các chương trình con Quet_trung_tam, Quet_truoc, Quet_sau thực hiện việc quét cây theo các trình tự TGP, GTP, TPG.

Nếu chúng ta nhập vào dãy ký tự U, T, V, O, M, Q, Y, V, U thì chương trình sẽ cho kết quả:

```
Du lieu nhap ban dau:
U  T  V  O  M  Q  Y  V  U

Du lieu hien theo cach quet Trung tam TGP
B  M  O  P  Q  T  U  V  Y

Du lieu hien theo cach quet Truoc GTP
U  T  O  M  B  Q  P  V  Y

Du lieu hien theo cach quet Sau TPG
B  M  P  Q  O  T  Y  V  U
```

Bài tập ứng dụng chương 4

1. Lập chương trình nhập vào mảng A n số thực, n nhập từ bàn phím. Nhập vào chuỗi S tối đa 30 ký tự. Sử dụng phương pháp cấp phát động cho các đối tượng A, S. Cho hiện lên màn hình cách thức bố trí bộ nhớ mà hệ thống dành cho mảng A và chuỗi S.

2. Sử dụng con trỏ mảng và mảng con trỏ để quản lý sách trong thư viện, giả thiết rằng mỗi cuốn sách gồm các thông tin: Tên sách, tên tác giả, số trang, giá tiền. Với cùng một số sách, bộ nhớ mà hệ thống sử dụng cho con trỏ mảng và mảng con trỏ có bằng nhau không?

3. Số liệu tuyển sinh bao gồm:

SBD, Họ và tên, phòng thi, điểm thi.

Sử dụng biến động nhập vào một số thí sinh, cho hiện danh sách thí sinh theo hai cách: hiện theo thứ tự nhập vào và hiện từ cuối lên đầu.

4. Tên người Việt là một chuỗi dài nhất là 7 ký tự. Nhập vào bộ nhớ n tên, sử dụng một trong các cách quét cây để hiện tên theo thứ tự tăng dần.

5. Tạo cây nhị phân mà mỗi nút chứa một số thực. Quét cây theo trình tự GTP, TGP, TPG. Tại mỗi nút yêu cầu xử lý dữ liệu là : hiện giá trị tổng các số trong nút cha và nút con.

6. Dãy số $a_1, a_2, ... , a_n$ là các số thực. Lập chương trình đưa dãy số trên vào cây nhị phân. Thiết kế 6 chương trình con tương ứng với 6 cách quét cây, yêu cầu xử lý tại mỗi nút là hiện trị tuyệt đối của số trong nút.

Chương 5

Giải thuật đệ quy

Nội dung của chương này đề cập đến những bài toán có tính đệ quy. Không phải bài toán nào cũng có tính đệ quy và không phải các bài toán có tính đệ quy thì đều phải giải bằng giải thuật đệ quy. Các vấn đề cần quan tâm trong chương này:

- Bài toán có tính đệ quy không
- Có cần dùng giải thuật đệ quy không
- Đệ quy có mang lại hiệu quả hơn các phương pháp thông thường hay không

1. Khái niệm đệ quy

Trong thân một chương trình con có thể đưa lời gọi tới chính chương trình con đó, tính chất này được gọi là tính "Đệ qui của chương trình con".

Trong toán học khái niệm giai thừa được định nghĩa như sau:

0! = 1

Nếu n > 0 thì n! = 1*2*3*...*n

Từ định nghĩa trên dễ dàng thấy rằng

n! = n*(n-1)!

(n-1)! = (n-1)*(n-2)!

.....

1! = 1*0! =1

Cách thức lập luận như trên đưa chúng ta tới một nhận xét tổng quát là: lời giải của bài toán A có thể được thực hiện bởi lời giải của bài toán A' có dạng giống như A. Thật vậy, việc tính n! có thể được thực hiện bởi việc tính (n-1)!.

Điều quan trọng để bài toán có lời giải là tuy A' giống như A song thực chất A' phải nhỏ hơn A và quá trình "thu nhỏ" phải có điểm dừng. Trong bài toán tính giai thừa từ chỗ cần tính giai thừa của n chúng ta đi tính giai thừa của (n-1), để tính giai thừa của (n-1) chúng ta đi tính giai thừa của (n-2)... kết thúc là giai thừa của 0.

Một đối tượng được gọi là đệ quy nếu nó được định nghĩa dưới dạng của chính nó.

Giải thuật của bài toán A được gọi là đệ quy nếu nó dẫn tới việc giải bài toán A' giống như A nhưng nhỏ hơn A và quá trình phải có điểm dừng.

Xét bài toán tính giai thừa với n = 5:

5! = 5*4!

4! = 4*3!

3! = 3*2!

2! = 2*1!

1! = 1

Ô nhớ cuối	1! = 1
	2! = 2*1!
	3! = 3*2!
	4! = 4*3!
Ô nhớ đầu	5! = 5*4!

Hình 5.1

Như vậy khi biết 1! thì tính được 2!, biết 2! Thì tính được 3!,...

Bằng cách sử dụng bộ nhớ ngăn xếp theo nguyên tắc LIFO (Last In - First Out) (Hình 1.3) những gì gửi vào cuối cùng thì được lấy ra trước tiên. Bóc ô nhớ trên đỉnh ta có 1! = 1 và lộ ra ô tiếp theo 2! = 2*1!. Vì 1! đã biết nên tính được 2! = 2, bóc tiếp ô nhớ phía dưới ta có 3! = 3*2!, vì 2! đã biết nên 3! = 3*2 = 6 quá trình tiếp tục cho đến khi bóc ô nhớ dưới cùng và chúng ta nhận được 5! = 120. Dưới đây là chương trình tính giai thừa

Ví dụ 5.1:

```
Program Giaithua;
Uses crt;
Var  n: integer;
Function GT(m: integr): integer;            (* Hàm GT tính giai thừa của n*)
Begin
if m=0 then GT:=1 else GT:= m*GT(m-1); (*Gọi đệ qui của GT *)
End;
Begin
Clrscr;
write ('Tinh giai thua cua n = '); Readln (n) (*Đọc giá trị n*)
write('Gia thua cua ',n, ' = ',GT(n));   (* Viết giá trị hàm GT *)
Repeat until keypressed;
End.
```

Ví dụ 5.1 có một số điều đáng lưu ý sau đây:

1. Hàm GT được xây dựng để tính giai thừa với tham số hình thức m, kiểu của m là kiểu Integer. Giá trị m sau này sẽ được thay thế bằng tham số thực n qua lời gọi GT(n) trong chương trình mẹ.

2. Khi định nghĩa kiểu của GT là Integer thì giá trị của n chỉ được chọn nhỏ hơn 8 vì 8! = 40320 vượt quá giá trị cực đại của Integer (32767). Để có thể tính giai thừa với n>=8 ta phải định nghĩa function GT có kiểu Longint hoặc Real. Với kiểu Real lệnh viết giá trị của giai thừa phải là viết số thực với phần lẻ bằng 0, ví dụ:

```
Write (GT(n):12:0);
```

3. Sử dụng thuật toán đệ quy đồng nghĩa với việc phải xây dựng chương trình con, vì vậy nếu sử dụng các vòng lặp có thể giải được bài toán thì nên dùng vòng lặp. Trừ trường hợp bắt buộc phải giải bài toán không có tính lặp hoặc bài toán có khả năng truy hồi.

Với bài toán tính giai thừa có thể dùng vòng lặp và chúng ta thấy chương trình sẽ đơn giản hơn nhiều so với cách dùng tính đệ quy:

Ví dụ 5.2

```
Program Tinh_GT;
Uses crt;
Var i,n:byte; gt:longint;
Begin
 Clrscr;
 Write(' Nhap gia tri n '); Readln(n);
 if (n = 0) or  (n=1) then gt:=1;
 if  n > 1 then  gt:=1;
 For i:= 1 to n do gt:=gt*i;
 Writeln('Giai thua cua ',n,' = ',gt);
```

```
Readln;
End.
```

Ví dụ 5.3: Lập chương trình tìm ước số chung lớn nhất của hai số nguyên n, m.
Ước số chung lớn nhất của hai số n và m tính theo công thức :

$$USC(n,m) = \begin{cases} n \text{ nếu } m = 0 \\ USC(m, n \bmod m) \text{ nếu } m <> 0 \end{cases}$$

Ví dụ n= 4, m=8
USC(4,8) = USC(8, 4) = USC(4,0) = 4
Chương trình được xây dựng như sau:

```
Program timUSC ;
Uses crt;
Var    n,m : word;     Lam: Char;
  FUNCTION USC(a,b:word): word;
  Begin
  If  b=0 then USC := a Else USC := USC(b,a mod b);
  End;
BEGIN
  Repeat
  Clrscr;
Write('  Hay cho hai so "n" va "m" can tim uoc so chung ');
Readln(n,m);
Writeln('  Uoc so chung lon nhat cua  ',n,' va ',m, ' la ',USC(n,m):5);
   Writeln;
   Write('Tim tiep hay thoi ? C/K '); read(lam);
  Until Upcase(lam)='K';
 END.
```

Bài toán kinh điển ứng dụng giải thuật đệ quy là bài toán Tháp Hanoi. Nội dung bài toán như sau:

Có n chiếc đĩa tròn đường kính khác nhau, các đĩa có lỗ ở giữa. Người ta xếp lồng các đĩa này vào một cọc theo thứ tự trên nhỏ dưới to. Yêu cầu phải xếp các đĩa này sang cọc mới theo nguyên tắc:

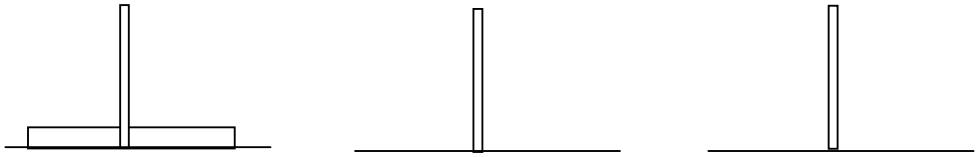
1. Mỗi lần chỉ được chuyển 1 đĩa
2. Không được để xảy ra tình trạng đĩa to ở trên, đĩa nhỏ ở dưới dù là tạm thời
3. Được phép dùng một cọc trung gian

Giả sử cọc đầu tiên là cọc A, cọc trung gian là B và cọc cần xếp đĩa sang là cọc C. Chúng ta sẽ xét một số trường hợp đơn giản để tìm quy luật (Hình 5.2)

* n =1: cọc A chỉ có một đĩa

Chuyển đĩa từ cọc A sang cọc C

Kết thúc



Hình 5.2

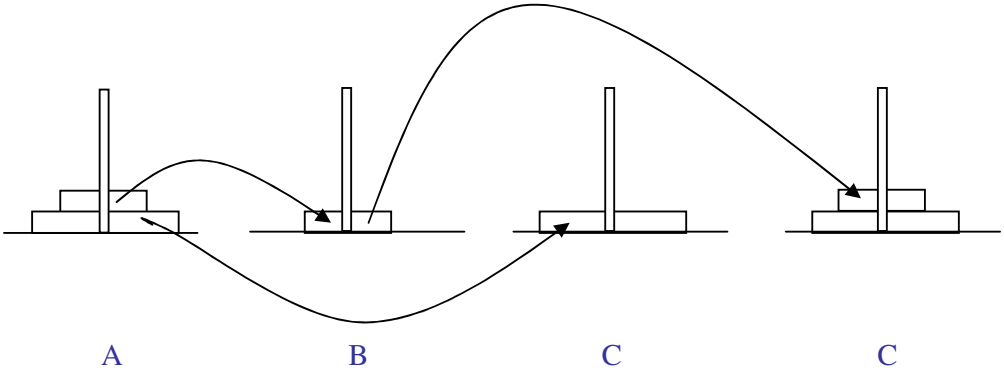
* n = 2: cọc A có 2 đĩa (Hình 5.3)

Chuyển đĩa trên sang B

Chuyển đĩa dưới sang C

Chuyển đĩa từ B sang C

Kết thúc



Hình 5.3

* n = 3: cọc A có 3 đĩa

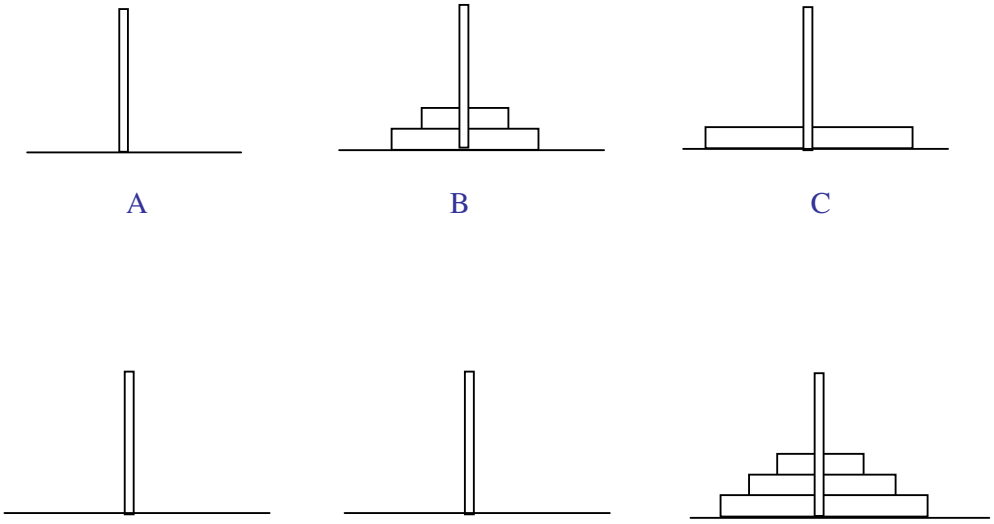
Với trường hợp 3 đĩa nếu chúng ta coi 2 đĩa trên là một đĩa thì bài toán quy về trường hợp n = 2, khi đó lời giải sẽ là: (Hình 5.4)

Chuyển 2 đĩa trên sang B

Chuyển đĩa dưới cùng sang C

Chuyển 2 đĩa từ B sang C

Kết thúc



Hình 5.4

Việc chuyển 2 đĩa từ A sang B chúng ta đã có lời giải (trường hợp n = 2)
Tóm lại với bài toán n đĩa chúng ta có lời giải tổng quát là:

- Chuyển n-1 đĩa trên cùng từ A sang B
- Chuyển đĩa dưới cùng từ A sang C
- Chuyển n-1 đĩa từ B sang C
- Kết thúc

Việc chuyển n-1 đĩa từ A sang B lại dẫn tới bài toán giống như chuyển n đĩa, song số lượng đĩa đã giảm đi 1, nghĩa là:

- Chuyển n-2 đĩa từ A sang B
- Chuyển đĩa thứ n-1 từ A sang C
- Chuyển n-2 đĩa từ B sang C
- Kết thúc

Quá trình sẽ dẫn tới lúc trên cọc A chỉ còn đĩa thứ n trên cọc B có n-1 đĩa và trên cọc C không có đĩa nào, đến đây ta chuyển đĩa thứ n từ A sang C. Bài toán sẽ đươc thu nhỏ lại với việc chuyển n-2 đĩa từ B sang A.

Số lần thực hiện các bước chuyển với n = 3 được cho trong bảng 5.1, ở đây chúng ta đã quy ước đĩa trên cùng là đĩa số 1, đĩa giữa là đĩa số 2 và đĩa dưới cùng là đĩa số 3. Dãy số trong các cọc luôn luôn sắp xếp theo chiều tăng dần chứng tỏ rằng đĩa nhỏ luôn ở trên, đĩa to luôn ở dưới.

Bảng 5.1

Bước	ý nghĩa	Cọc A	Cọc B	Cọc C
0		1,2,3		
1	Chuyển hai đĩa trên cùng từ A sang B	2,3		1
2		3	2	1
3		3	1,2	
4	Chuyển đĩa số 3 từ A sang C		1,2	3
5	Chuyển 2 đĩa từ B sang C	1	2	3
6		1		2,3
7				1,2,3

Các ví dụ trên cho thấy số lần chuyển tăng rất nhanh khi số đĩa tăng tuyến tính, với n= 1 số lần chuyển là 1, với n = 2 số lần chuyển là 3, với n = 3 số lần chuyển là 7.

Bảng 5.2

Bước	ý nghĩa	Cọc A	Cọc B	Cọc C
0		1,2,3,4		
1	Chuyển 3 đĩa trên cùng từ cọc A sang cọc trung gian B	2,3,4	1	
2		3,4	1	2
3		3,4		1,2
4		4	3	1,2
5		1,4	3	2
6		1,4	2,3	
7		4	1,2,3	
8	Chuyển đĩa 4 từ A sang C		1,2,3	4
9	Chuyển 2 đĩa trên cùng từ B sang A		2,3	1,4
10		2	3	1,4
11		1,2	3	4
12	Chuyển đĩa 3 từ B sang C	1,2		3,4
13	Chuyển 2 đĩa 1 và 2 từ A sang C	2	1	3,4
14			1	2,3,4
15				1,2,3,4

Có thể dễ dàng tìm ra công thức truy hồi để tính số lần chuyển đĩa

Nếu số đĩa là n thì số lần chuyển là $2^n - 1$

Với n = 4 số lần chuyển sẽ là 15 (bảng 5.2).

Nếu trên cọc có 32 đĩa thì số lần chuyển đã là hơn 4 tỷ, còn nếu số đĩa là 64 theo như bài toán cổ về tháp Hanoi thì số lần chuyển sẽ là: 36893488147000000000 lần

Giả sử chúng ta cho rằng mỗi lần chuyển một đĩa hết 1 giây thì để chuyển hết số đĩa nói trên từ cọc A sang cọc C chúng ta cần 1 169 884 834 700 năm. Đây là khoảng thời gian mà bản thân hệ mặt trời và dải ngân hà cũng khó có thể tồn tại.

2. Thiết kế giải thuật đệ quy - Khử đệ quy

Với một bài toán toán bất kỳ, để xem nó có tính đệ quy hay không chúng ta phải trả lời được các câu hỏi sau đây:

2.1 Bài toán đã cho có thể được định nghĩa dưới một dạng giống hệt như nó nhưng có kích thước nhỏ hơn hay không?

2.2 Quá trình thu nhỏ bài toán có dẫn tới một điểm dừng nào đó hay không? Nói cách khác bài toán thu nhỏ có dẫn tới một trường hợp đặc biệt mà ta gọi là trường hợp xuy biến nghiã là biết lời giải hay không?

Với bài toán giai thừa trường hợp xuy biến là 0! = 1

Với bài toán Tháp Hanoi trường hợp xuy biến là khi chỉ có 1 đĩa trên cọc A.

2.3 Khi xử dụng tính đệ quy mỗi lần thu nhỏ bài toán kích thước bài toán giảm đi như thế nào?

Các ví dụ về đệ quy có thể xem trong nhiều tài liệu về lập trình, vấn đề chúng ta quan tâm ở đây là khi một bài toán có thể giải quyết bằng cả đệ quy và cách thông thường thì nên dùng cách nào?

Với bài toán tính giai thừa sử dụng vòng lặp For... việc lập trình rất đơn giản, còn dùng đệ quy sẽ phức tạp hơn. Với bài toán tìm ước số chung lớn nhất rõ ràng là cách dùng đệ quy sẽ đơn giản hơn các cách khác. Với bài toán Tháp Hanoi nếu không dùng đệ quy chúng ta khó mà tìm ra một thuật giải nào trong sáng hơn. Nói như vậy cũng có nghĩa là không có một chuẩn mực nào chung cho mọi bài toán. Kinh nghiệm của các nhà lập trình đã chỉ ra rằng nếu một bài toán vừa có thể giải bằng đệ quy, vừa có thể giải bằng phương pháp lặp thông thường thì nên tránh dùng đệ quy.

Một bài toán có thể thay thế giải thuật đệ quy bằng các giải thuật không tự gọi đến chúng thì gọi là khử đệ quy.

Ví dụ với bài toán tính giai thừa chúng ta hoàn toàn có thể giải bằng cách dùng vòng lặp For . Bài toán trong trường hợp này đơn giản hơn cả về thuật toán lẫn kỹ thuật lập trình.

3. Hiệu lực của bài toán đệ quy

Như đã nêu đệ quy là một trong các công cụ để lập trình giải các bài toán. Cần khẳng định rằng đệ quy chỉ được dùng cho một số bài toán mà chúng ta tìm được giải thuật đệ quy. Trở lại ví dụ 4.16 về xây dựng cây nhị phân. Nếu cây mà mỗi nút cha có nhiều nhất hai nút con nhưng không quy định rõ con bên trái và con bên phải bố trí thế nào thì chúng ta không thể dùng được tính đệ quy. Trong trường hợp quy định rõ giá trị ở con bên trái luôn nhỏ hơn giá trị ở nút cha còn ở con bên phải là lớn hơn nút cha thì chúng ta thấy việc khảo sát một cây nhị phân tổng quát có thể xuy về việc khảo sát riêng từng cây con bên trái và cây con bên phải. Chương trình con Taocay trong ví dụ 4.16 đã sử dụng tính đệ quy để bổ xung dữ liệu vào các nút, với cách viết này chương trình Taocay không đơn giản hơn cách viết dùng vòng lặp. Tuy nhiên khi duyệt cây với các chương trình con Quet_trung_tam, Quet_truoc, Quet_sau

thì tính đệ quy đã làm cho chương trình trở nên trong sáng và rất dễ hình dung là cây được quét như thế nao.

Tính đệ quy về một khía cạnh nào đó có thể xem như tính quy nạp toán học, tuy nhiên cần phân biệt rằng đệ quy được thực hiện là nhờ các bộ nhớ kiểu LIFO hoặc FIFO còn quy nạp toán học thuần tuý chỉ là lý thuyết. Việc sử dụng quy nạp toán học để chứng minh tính đúng đắn của giải thuật đệ quy đã được một vài tác giả thực hiện, điều này chỉ chứng minh tính đúng đắn của giải thuật đệ quy chứ không cho biết giải thuật ấy hiệu quả như thế nào. Vấn đề là người lập trình phải tự xác định xem thuật giải nào tiêu tốn ít công sức của người lập trình và tài nguyên trong máy nhất.

Bài tập chương 5

1. Dãy số Fibonaci a_1, a_2, ... a_n được định nghĩa như sau:
$a_1 = a_2 = 1$
$a_3 = a_2 + a_1$
.....
$a_n = a_{n-1} + a_{n-2}$

Lập chương trình với việc áp dụng tính đệ quy để xây dựng dãy số Fibonaci, giá trị n nhập từ bàn phím.

2. Lập chương trình giải bài toán Tháp Hanoi

3. Hàm F(x,y) được định nghĩa như sau:

$$F(x,y) = \begin{cases} y+1 & \text{nếu x = 0} \\ F(x-1,1) & \text{nếu y = 0} \\ F(x-1,F(x,y-1)) & \text{với các trường hợp còn lại} \end{cases}$$

Lập chương trình có dùng đệ quy để tính hàm F với giá trị x,y nhập từ bàn phím.

4. S là một chuỗi ký tự có độ dài tối đa là 255. Lập chương trình nhập S từ bàn phím sau đó in ra chuỗi ngược của chuỗi ban đầu. Chương trình có dùng đệ quy
Ví dụ: S = 'Chuc mung ngay 8/3'
In ra '3/8 yagn gnum cuhC'

5. Cho dãy số a_1, a_2, ... , a_n . Gọi số hoán vị của dãy số là k.
Lập một chương trình hiện lên số hoán vị của dãy số và giá trị cụ thể của từng hoán vị
Ví dụ: dãy số 1 3 2
In ra: Số hoán vị = 6
Giá trị cụ thể 1 2 3; 1 3 2; 2 1 3; 2 3 1; 3 1 2; 3 2 1

Chương 6

Đồ hoạ

Đồ hoạ trong Pascal không phải là công cụ chuyên dùng để thiết kế hình ảnh. Tuy nhiên nếu biết tận dụng các công cụ sẵn có trong Unit Graph chúng ta có thể làm được nhiều việc, đặc biệt là vẽ đồ thị các hàm số. Trong chương này bạn đọc tiếp cận các khái niệm sau:

- Các thủ tục vẽ hình đơn giản
- Các thủ tục viết chữ trong chế độ đồ hoạ
- Các thủ tục tô màu
- Các phương pháp xử lý ảnh Bitmap
- Phương pháp vẽ đồ thị hàm số

1. Khái niệm chung

Màn hình máy vi tính có thể dùng ở một trong hai chế độ: chế độ TEXT - hiển thị văn bản và chế độ GRAPHIC - hiển thị đồ hoạ.

Trong chế độ TEXT màn hình thường được chia thành 25 dòng và 80 cột, nếu viết kín màn hình ta có thể viết được 2000 ký tự. Chúng ta có thể thay đổi độ phân giải để viết ra 25 dòng x 40 cột, 50 dòng x 80 cột hoặc 132 dòng x 43 cột.

Muốn vẽ hình, tô màu các hình ta phải chuyển sang chế độ đồ hoạ, trong chế độ này màn hình được xem là một ma trận điểm, tuỳ thuộc độ phân giải ta có thể có ma trận 640x480 điểm hoặc 1024x720 điểm.... Mỗi điểm trên màn hình được gọi là 1 Pixel tức là một phần tử ảnh (Picture Element), ta có thể hoàn toàn chủ động trong việc thay đổi màu sắc của từng điểm để tạo ra một bức tranh theo ý muốn. Vị trí của mỗi điểm trên màn hình được biểu diễn bởi hai toạ độ: Hoành độ và Tung độ. Gốc toạ độ (0,0) là điểm ở góc trên bên trái màn hình.

Như đã nêu trong chương 1 phần cài đặt Pascal, muốn chuyển sang làm việc ở chế độ đồ hoạ, trong thư mục hiện hành (thư mục chứa chương trình Pascal) phải có các tệp GRAPH.TPU, *.BGI và *.CHR. Lời gọi đơn vị chương trình đồ hoạ phải đặt ở đầu chương trình ngay sau từ khoá PROGRAM như ví dụ 6.1.

Ví dụ 6.1

```
Program Ve_hinh;
Uses GRAPH;
.....
```

Trong phần thân chương trình cần phải đưa vào các thông báo về kiểu màn hình, chế độ đồ hoạ (MODE) tương ứng. Những người làm tin học ứng dụng thường không quan tâm lắm đến các thông số này do vậy dễ lúng túng khi cần khai báo. Để khắc phục nhược điểm đó trong Pascal đã thiết kế sẵn một thủ tục khởi tạo chế độ đồ hoạ là:

Initgraph(var GD,GM: Integer, DP:string[n]);

Khi gọi thủ tục này với các tham số hợp lệ Initgraph sẽ tự xác định kiểu màn hình và Mode đồ hoạ tối ưu .

Để gọi thủ tục Initgraph cần phải khai báo trước các tham số GD, GM thuộc kiểu Integer (Trong đó GD: Graph Driver - là một số nguyên xác định kiểu màn hình; GM: Graph Mode - cũng là một số nguyên xác định Mode đồ hoạ).

Nếu ngay sau từ khoá Begin của phần thân chương trình chúng ta khai báo

GD:= Detect;

thì Initgraph hiểu là nó phải tự đi xác định kiểu màn hình và Mode đồ hoạ sao cho đạt kết quả tối ưu. Nói chung trừ những trường hợp đặc biệt, chúng ta không nên tự xác định những thông số này làm gì.

Tham số DP (Driver Path) là đường dẫn tới thư mục chứa các tệp điều khiển kiểu màn hình đồ hoạ, thông thường Pascal được cài đặt trong trong đĩa cứng nên DP sẽ là 'C:\tp\bgi' nghĩa là ổ đĩa C, thư mục con BGI trong thư mục TP.

Ví dụ 6.2 là chương trình vẽ một đường tròn có tâm tại chính giữa màn hình và bán kính là 50 Pixel.

Ví dụ 6.2

```
Program Ve_hinh_tron;
Uses graph;
Var  GD,DM: Integer;
Begin
GD:= detect;
Initgraph(GD,GM,'c:\tp\bgi');
If graphresult <> grok then halt(1);
Circle(320,240,50);
Readln; CloseGraph;
End.
```

Dòng thứ 8 trong ví dụ 6.2

If graphresult <> grok then halt(1);

là câu lệnh kiểm tra giá trị của hàm graphresult (kết quả kiểm tra đồ hoạ), nếu hàm này nhận giá trị 0 thì đồ hoạ không có lỗi, chương trình tiếp tục làm việc, còn nếu giá trị của graphresult khác 0 chứng tỏ việc kiểm tra phát hiện ra lỗi, chương trình phải dừng lại.

Bảng 6.1 cho mã lỗi mà hàm graphresult trả về.

Thông thường lỗi xảy ra chủ yếu là do người sử dụng khai báo không đúng tham số GD, nghĩa là không chỉ ra cho Pascal biết các tệp điều khiển đồ hoạ nằm ở đâu, muốn sửa chữa chỉ cần khai báo lại tham số này theo đúng vị trí mà nó được cài đặt.

2. Một số thủ tục cơ bản để vẽ hình

2.1. MOVETO(x,y) : Di chuyển con trỏ đến toạ độ x,y (x là hoành độ, y là tung độ)

x,y là các giá trị kiểu Integer, với màn hình VGA thì 0<= x <=639, 0<= y <= 479

2.2. LINETO(x,y) : Vẽ một đường thẳng từ vị trí con trỏ hiện thời tới toạ độ x,y kết thúc quá trình vẽ con trỏ nằm tại toạ độ mới.

Pascal có sẵn hai hàm để xác định toạ độ góc dưới bên phải màn hình đó là Getmaxx và Getmaxy. Để vẽ một đường chéo của màn hình từ góc trên bên tráixuống góc dưới bên phải ta có thể viết các lệnh

```
MOVETO(0,0);
LINETO(Getmaxx,Getmaxy);
```

2.3. LINE(x1,y1,x2,y2) : Thủ tục này vẽ một đường thẳng từ toạ độ x1,y1 đến toạ độ x2,y2

2.4. LINEREL(dX,dY) : Vẽ đường thẳng từ vị trí hiện thời (toạ độ x,y) tới toạ độ x+dx, y+dy.

Bảng 6.1

Mã lỗi	Tên lỗi	Nghĩa của lỗi
0	Grok	Tốt, không có lỗi
1	GrnoInitgraph	Không tìm thấy đơn vị đồ hoạ
2	Grnotdetected	Không có phần cứng đồ hoạ
3	GrFilenotfound	Không tìm thấy các tệp điều khiển đồ hoạ
4	GrInvalidDriver	File điều khiển đồ hoạ (BGI) bị hỏng
5	GrNoloadMem	Không đủ bộ nhớ để nạp trình điều khiển đồ hoạ
6	GrNoscanMem	Không đủ bộ nhớ khi duyệt (kiểm tra)
7	grNoFloodMem	Không đủ bộ nhớ khi kết xuất (đưa ra)
8	grFontNotFound	Không tìm thấy Font (các tệp đuôi CHR)
9	grNoFontMem	Không đủ bộ nhớ để nạp Font
10	grInvalidMode	Sai mod đồ hoạ khi lựa chọn các điều khiển
11	grError	Lỗi đồ hoạ (lỗi tổng quát)
12	grIOerror	Có lỗi vào/ra đồ hoạ
13	grInvalidFont	Lỗi các tệp chứa Font
14	grInvalidFontNum	Lỗi số hiệu Font

2.5. CIRCLE(x,y,r) : vẽ đường tròn tâm tại toạ độ x,y bán kính bằng r Pixel

2.6. PUTPIXEL(x,y, n) : Thủ tục này sẽ vẽ một điểm sáng tại toạ độ x,y với màu là n. Giá trị n lấy trong khoảng 0-15 hoặc viết trực tiếp tên màu theo tiếng Anh.

2.7. RECTANGLE(x1,y1,x2,y2) : Vẽ hình chữ nhật toạ độ góc trên bên trái là x1,y1 , toạ độ góc dưới bên phải là x2,y2.

2.8. BAR(x1,y1,x2,y2) : Vẽ một hình chữ nhật góc trên bên trái có toạ độ x1,y1 góc dưới bên phải có toạ độ x2,y2. Khi dùng kết hợp với thủ tục số 9 sẽ đồng thời cho phép kẻ các vân hoa trên nền và tô màu cho nền.

2.9. SETFILLSTYLE(n1,n2) : Thủ tục định vân hoa và màu nền cho thủ tục BAR.

n1 là một giá trị nguyên với 0<= n1 <= 11 : định kiểu vân hoa

n2 là số hiệu mã màu đã giới thiệu 0<= n2 <= 15

Trong chương trình khi muốn vẽ hình chữ nhật với màu và vân hoa thì cần đưa thủ tục SETFILLSTYLE(n1,n2) vào trước thủ tục BAR. Giá trị của màu và kiểu vân hoa sẽ được giữ cho đến khi ta định nghĩa lại.

3. Thiết lập màu đồ hoạ

Các thủ tục trình bày trong mục II không có tham số định màu kèm theo. Nếu muốn định màu cho nét vẽ chúng ta có thể dùng hai thủ tục sau đây :

3.1. SETCOLOR(n) : Định màu cho các nét vẽ (n là số hiệu màu)

3.2. SETBKCOLOR(n) : Định màu nền đằng sau nét vẽ

Pascal sử dụng một Byte để lưu trữ các thuộc tính màu, trong đó 4 bit thấp là màu chữ, 3 bit cao là màu nền, bit cao nhất dành cho thuộc tính nhấp nháy. Mã nhị phân 4 bit có thể biểu diễn các số từ 0 (số 0000) đến 15 (số 1111) do đó thuộc tính màu của chữ có 16 giá trị ứng với 16 màu, còn thuộc tính màu nền chỉ có 8 giá trị.

Chú ý:

* Giá trị n trong thủ tục 10 có thể chọn từ 0 đến 15, còn trong thủ tục 11 chỉ được chọn từ 0 đến 7.

* Thay vì chọn số hiệu màu n, Pascal cho phép viết tên hằng màu, như vậy hai lệnh sau đây là tương đương:

Textcolor(4);

Textcolor(Red);

Bảng giá trị n và màu tương ứng xem trong mục 8.3 chương III (Bảng 3.1)

Để tô màu cho một điểm chúng ta đưa ngay tham số màu vào thủ tục PutPixel, còn tô màu cho một hình khép kín thì dùng thủ tục SetFillstyle.

3.3. GetBkcolor: cho biết mã màu nền hiện dùng

4. Viết chữ trong chế độ đồ hoạ

Khi đã chuyển sang làm việc ở chế độ đồ hoạ ta không thể viết chữ bình thường như trong chế độ văn bản. Muốn viết chữ trên màn hình đồ hoạ chúng ta sử dụng một số thủ tục sau đây:

4.1. OUTTEXT(chuỗi) : Thủ tục này sẽ cho hiện chuỗi ký tự tại vị trí con trỏ hiện thời. Chuỗi có thể viết trực tiếp hoặc thông qua biến chuỗi như trong ví dụ 6.3 sau đây:

Ví dụ 6.3

```
Uses Graph;
Var
 chuviet : string[30]
Begin
outtext('Cong hoa xa hoi chu nghia Viet nam');
chuviet:='Viet nam dan chu cong hoa';
outtext(chuviet);
End.
```

4.2. OUTTEXTXY(x,y,chuoi) : thủ tục này sẽ viết ra chuỗi ký tự tại toạ độ x,y.

4.3. SETTEXTSTYLE(Kiểu chữ, Chiều viết, Kích thước); Xác định kiểu chữ trong Mode đồ hoạ

Kiểu chữ là một tham số nguyên nhận giá trị trong khoảng 0-10

Chiều viết chỉ nhận 1 trong hai giá trị :

0: chữ nằm ngang; 1: chữ thẳng đứng

Kích thước Là hệ số phóng to chữ có thể chọn từ 0-10

Chú ý:

Các phiên bản Pascal cũ chỉ có 5 kiểu chữ đánh số từ 0 đến 4, phiên bản 7 bổ xung thêm các kiểu từ 5 đến 10 (xem bảng 6.2)

4.4. CLOSEGRAPH; *chấm dứt chế độ đồ hoạ trở về chế độ văn bản.*

Sau đó muốn quay lại chế độ đồ hoạ ta lại phải gọi lại INITGRAPH.

Trong một số trường hợp để chuyển nhanh giữa chế độ đồ hoạ và văn bản chúng ta có thể dùng hai thủ tục sau đây:

RESTORECRTMODE; ngừng chế độ đồ hoạ chuyển sang chế độ văn bản.

SETGRAPHMODE(n) ; Ngắt chế độ văn bản đã tạo ra bởi Restorecrtmode thiết lập trở lại chế độ đồ hoạ. Tham số n có thể lựa chọn trong khoảng 0-2. Ví dụ 6.4 dưới đây trình bày cách sử dụng các thủ tục này.

Giá trị	Tên Font	Tệp lưu trữ (đuôi CHR)
0	DefaultFont	Không có
1	TriplexFont	TRIP
2	SmallFont	LITT
3	SansSerifFont	SANS
4	GothicFont	GOTH
5	ScriptFont	SCRI
6	SimplexFont	SIMP
7	TriplexScriptFont	TSCR
8	ComplexFont	LCOM
9	EuropeanFont	EURO
10	BoldFont	BOLD

Bảng 6.2

Ví dụ 6.4

```
Program dohoa_text;
uses crt,graph;
var
gd,gm:integer;
begin
gd:=detect;
initgraph(gd,gm,'c:\tp\bgi');
if graphresult<>grok then halt(1);
moveto(0,0); setcolor(5);
```

```
lineto(300,300); delay(2500);
circle(400,300,100); delay(1500);
restorecrtmode;                (* Chuyển về chế độ văn bản *)
gotoxy(20,20);textcolor(9);
write('AAAAAAAA');
readln;
setgraphmode(2);  (* Trở về chế độ đồ hoạ với n=2 cho màn hình VEGA*)
setcolor(blue);
circle(100,100,50);
delay(2000);
restorecrtmode;          (* Chuyển sang chế độ văn bản lần thứ hai*)
textcolor(3);
gotoxy(20,0);  write('NNNNNNNNNNNNNNN');
readln;
closegraph;                (* Kết thúc chế độ đồ hoạ*)
End.
```

5. Các ví dụ

Việc sử dụng các thủ tục đồ hoạ không có gì phức tạp, với một chút cố gắng bạn có thể vẽ được những hình rất đẹp theo mong muốn. Trong phần này chúng ta sẽ sử dụng các thủ tục đã trình bày để vẽ một số hình.

5.1 Vẽ đồ thị hình Sin

Dưới đây là một chương trình vẽ đồ thị hình sin. Chạy chương trình ta sẽ thấy ba đường hình sin với các biên độ và màu sắc khác nhau.

Ví dụ 6.5

```
Program Do_thi_hinh_sin;
uses graph,crt;
const  m=0.1;
Var  t3,t4,t1,n,t2,gd,gm:integer;    t,x,y,z:real;
  Begin
  gd:=detect;
  Initgraph(gd,gm,'C:\tp\bgi');
  if graphresult<>grok then Halt(1);
  x:=0;     t3:=100;     n:=0;     t2:=10;
   while t2<=600 do
   Begin
  setcolor(green);
```

```
y:=sin(x);
t1:=round(y*50);
t3:=round(y*70);
t4:=round(y*100);
t1:=200-t1;
t3:=200-t3;
 t4:=200+t4;
moveto(10,200);
lineto(620,200);
line(10,80,10,300);
settextstyle(3,0,3);
outtextxy(610,205,'x');
settextstyle(3,0,3);
outtextxy(15,75,'y');
settextstyle(4,0,3);
setcolor(5);
outtextxy(200,300,'do thi ham sin(x)');
setcolor(12);
moveto(10,200);
putpixel(t2,t1,11);
putpixel(t2,t3,14);
setcolor(red);
putpixel(t2,t4,random(14));
setcolor(12);
delay(5);
 x:=x+0.07;
 t2:=t2+1;
 end;
 repeat until keypressed;
 t1:=1;
 t2:=200;
 while t1<=220 do
 begin
 line(340,240,round(sqrt(440*440-t1*t1)),t1);
 t1:=t1+1;
 delay(15);
 end;
 repeat until keypressed;
 closegraph;
 End.
```

5.2 Vẽ các hình không gian đơn giản

Chương trình sau đây sẽ vẽ một khung chữ nhật bao quanh toàn bộ màn hình, sau đó vẽ hai đoạn thẳng chia màn hình thành 4 phần. Trong góc phần tư thứ nhất vẽ một hình chóp, các nét nhìn thấy vẽ liền, nét khuất vẽ bằng nét đứt. Trong góc phần tư thứ hai vẽ một hình trụ. Xin mời các bạn nhập chương trình và sau đó vẽ thêm vào các góc phần tư còn lại các hình không gian khác, chẳng hạn hình hộp chữ nhật, hình nón...

Ví dụ 6.6

```
Program vehinh;
Uses crt,graph;
var i,j:word; gd,gm:integer;
Begin
clrscr;
gd:=detect;
initgraph(gd,gm,'C:\tp\bgi');
if graphresult<>grok then halt;
rectangle(0,0,639,479);
moveto(320,0); lineto(320,480);
moveto(0,240); lineto(639,240);
outtextxy(30,90,'Hinh chop');
setcolor(4);
setlinestyle(0,1, normwidth);
moveto(100,200);
Linerel(120,0);
Linerel(50,-40);
setcolor(2);
setlinestyle(3,1, normwidth);
Linerel(-120,0);
Linerel(-50,40);
setcolor(4);
setlinestyle(0,1, normwidth);
Lineto(185,50);
Lineto(220,200);
moveto(270,160);
Lineto(185,50);
setcolor(2);
setlinestyle(3,2, normwidth);
Lineto(150,160);
moveto(185,50);
setcolor(14);
```

```
lineto(185,180);
circle(185,179,2);
moveto(100,200);
lineto(270,160);
moveto(150,160);
lineto(220,200);
ellipse(480,50,0,360,60,25);
ellipse(480,170,180,360,60,25);
setlinestyle(0,15, normwidth);
moveto(420,50); lineto(420,170);
moveto(540,50); lineto(540,170);
setlinestyle(2,15, normwidth);
setcolor(5);
ellipse(480,170,0,180,60,25);
setcolor(9);
outtextxy(330,90,'Hinh tru');
repeat until keypressed;
closegraph;
end.
```

5.3 Viết chữ trên màn hình

Sử dụng các thủ tục đã nêu chúng ta có thể viết chữ trên màn hình đồ hoạ, ví dụ sau đây sẽ viết lên màn hình dòng chữ "Chuc mung nam moi" và cho dòng chữ này chạy từ đáy lên đỉnh màn hình. Việc cho một dòng chữ chạy được thực hiện bằng cách viết chữ tại một vị trí nào đó sau đó xoá chữ đi và viết lại chữ tại toạ độ khác. Tuy nhiên trong chế độ đồ hoạ chúng ta không dùng được lệnh xoá CLRSCR do đó có thể dùng cách viết lại dòng chữ với màu trùng màu nền.

Ví dụ 6.7

```
Program chucchay;
uses crt,graph;
var i,j,gd,gm:integer;
Begin
clrscr;
gd:=detect;
Initgraph(gd,gm,'C:\tp\bgi');
if graphresult<>grok then halt;
settextstyle(0,0,2);
For i:=439 downto 0 do
Begin
setbkcolor(3);
```

```
setcolor(magenta);
outtextxy(180,i,'Chuc mung nam moi');
delay(24);
setcolor(3);
outtextxy(180,i,'Chuc mung nam moi');
end;
closegraph;
end.
```

Chương trình dưới đây thiết kế một đồng hồ ba kim , tốc độ chạy của kim giây tuỳ thuộc vào lệnh DELAY(n) , nếu chọn DELAY(1000) thì cứ 1 giây kim giây chuyển một vị trí. Khi nhập chương trình vào máy cần lưu ý khai báo lại đường dẫn đến thư mục chứa các tệp *.BGI

Ví dụ 6.8

```
Program Ve_dong_ho;
uses crt,graph;
 var
x,y, maxx,maxy, gd,gm,color,i,j,t:integer;
 N:real;
 LAM,TT:CHAR;
 begin
   gd:=detect;
   initgraph(gd,gm,'c:\tp\BGI');
   setcolor(5);
   rectangle(30,20,610,450);
   rectangle(31,21,609,449);
   rectangle(32,22,608,448);
   setfillstyle(9,2);
   bar(33,23,607,447);
   setcolor(red);
   setbkcolor(red);
   for i:=1 to 10 do circle(320,240,i);
   setcolor(11);
   setbkcolor(white);
   for i:=11 to 80 do circle(320,240,i);
    setcolor(14);
    setbkcolor(white);
   for i:=80 to 160 do circle(320,240,i);
    setcolor(white);
```

```
for i:=160 to 200 do circle(320,240,i);
setcolor(11);
circle(320,240,79);
circle(320,240,80);
setcolor(4);
circle(320,240,159);
circle(320,240,160);
settextstyle(3,0,4);
outtextxy(310,40,'XII');
outtextxy(405,60,'I');
outtextxy(470,120,'II');
outtextxy(490,200,'III');
outtextxy(480,290,'IV');
outtextxy(410,370,'V');
outtextxy(310,400,'VI');
outtextxy(210,370,'VII');
outtextxy(135,290,'VIII');
outtextxy(130,210,'IX');
outtextxy(155,130,'X');
outtextxy(220,60,'XI');
setcolor(blue);
Settextstyle(4,0,5);
outtextxy(230,100,'DIAMON');
setcolor(random(14));
for i:=1 to 20 do
circle(320,360,i );
settextstyle(1,0,2);
setcolor(5);
outtextxy(200,450,'Copyright by Dr. Duong Xuan Thanh');
    for i:=1 to 20 do
  begin
  setcolor(random(14));
  circle(320,360,i );
  end;
for i:=1 to 20 do
begin
setcolor(random(14));
circle(320,360,i );

end;
```

```
for t:=0 to 12 do   {-------- Kim gio --------}
begin
setcolor(12);
moveto(320,240);
setlinestyle(0,0,3);
SetWriteMode(xorput);
linerel(round(110*cos((t*30-89)*pi/180)),round(110*sin((t*30-89)*pi/180)));
 moveto(320,240);
linerel(round(110*cos((t*30-90)*pi/180)),round(110*sin((t*30-90)*pi/180)));
 moveto(320,240);
linerel(round(110*cos((t*30-91)*pi/180)),round(110*sin((t*30-91)*pi/180)));
 moveto(320,240);
linerel(round(110*cos((t*30-92)*pi/180)),round(110*sin((t*30-92)*pi/180)));
  for i:=0 to 60 do  { ---------Kim phut -------}
  begin
  setcolor(12);
  moveto(320,240);
  setlinestyle(0,0,3);
  SetWriteMode(xorput);
  linerel(round(130*cos((i*6-89)*pi/180)),round(130*sin((i*6-89)*pi/180)));
  moveto(320,240);
  linerel(round(130*cos((i*6-90)*pi/180)),round(130*sin((i*6-90)*pi/180)));
  moveto(320,240);
  linerel(round(130*cos((i*6-91)*pi/180)),round(130*sin((i*6-91)*pi/180)));
              (*--------Kim giay--------*)
  for j:=0 to 360 do
  begin
  moveto(320,240);
  setlinestyle(0,0,3);
  SetWriteMode(XORPut);
  setcolor(12);
  linerel(round(150*cos((j-90)*pi/180)),round(150*sin((j-90)*pi/180)));
  moveto(320,240);
  linerel(round(150*cos((j-91)*pi/180)),round(150*sin((j-91)*pi/180)));
  delay(1000);
  moveto(320,240);
  linerel(round(150*cos((j-90)*pi/180)),round(150*sin((j-90)*pi/180)));
  moveto(320,240);
  linerel(round(150*cos((j-91)*pi/180)),round(150*sin((j-91)*pi/180)));
  end;
```

```
    moveto(320,240);
    linerel(round(130*cos((i*6-89)*pi/180)),round(130*sin((i*6-89)*pi/180)));
    moveto(320,240);
    linerel(round(130*cos((i*6-90)*pi/180)),round(130*sin((i*6-90)*pi/180)));
    moveto(320,240);
    linerel(round(130*cos((i*6-91)*pi/180)),round(130*sin((i*6-91)*pi/180)));
  end;
   moveto(320,240);
   linerel(round(110*cos((t*30-89)*pi/180)),round(110*sin((t*30-89)*pi/180)));
   moveto(320,240);
   linerel(round(110*cos((t*30-90)*pi/180)),round(110*sin((t*30-90)*pi/180)));
   moveto(320,240);
   linerel(round(110*cos((t*30-91)*pi/180)),round(110*sin((t*30-91)*pi/180)));
   moveto(320,240);
   linerel(round(110*cos((t*30-92)*pi/180)),round(110*sin((t*30-92)*pi/180)));
   end;
   repeat  until keypressed;
END.
```

6. Xử lý ảnh Bitmap

Mỗi hình vẽ trên một khu vực màn hình được coi như một ảnh và được gọi là Bitmap, các Bitmap được lưu trữ trong bộ nhớ bởi tập hợp các bit. Mỗi ảnh lại bao gồm nhiều phần tử ảnh (Pixel), mỗi Pixel được lưu trữ bởi 4 bit và việc xử lý ảnh thực chất là xử lý từng bit.

Pascal đã thiết kế sẵn một số thủ tục để lưu trữ ảnh hoặc để đưa ảnh ra màn hình:

6.1 Hàm xác định kích thước (dung lượng) ảnh

ImageSize(x1,y1,x2,y2);

Hàm này cho ta kích thước tính bằng Byte của một ảnh trong miền chữ nhật, góc trên bên trái x1,y1 và góc dưới bên phải là x2,y2.

6.2 Thủ tục gán kích thước vào biến

Getmem(biến, Imagesize(x1,y1,x2,y2));

Gán vào biến đã lựa chọn kích thước ảnh trong vùng chữ nhật góc trên trái x1,y1 góc dưới phải x2,y2, kích thước được tính bằng Byte

6.3 Thủ tục lưu trữ ảnh

GetImage(x1,y1,x2,y2, biến);

Cất ảnh trong miền chữ nhật góc trên trái x1,y1 góc dưới phải x2,y2 vào một biến, đây là biến không định kiểu và phải có kích thước đủ lớn để lưu ảnh, trong nhiều ứng dụng đồ hoạ biến này thuộc kiểu con trỏ và được xác định kích thước thông qua thủ tục Getmem.

6.4 Thủ tục cho hiện ảnh

PutImage(x, y, biến, phương thức);

Cho hiện ảnh lưu trữ trong "biến" lên màn hình bắt đầu tại toạ độ x,y, "Phương thức" là một số giá trị mà Pascal đã thiết kế nhằm đưa ảnh Bitmap ra màn hình. Dưới đây là bảng các giá trị của "Phương thức"

Tên phương thức	Phép toán tác động
CopyPut hoặc NormalPut	MOV (màu ảnh đè lên trên màu nền)
XorPut	XOR
OrPut	OR
AndPut	AND
NotPut	NOT

Để hiểu rõ hơn về khái niệm "Phương thức" chúng ta cần lưu ý đến các phép toán mà mỗi phương thức xử dụng.

Chẳng hạn khi phương thức là XorPut thì có thể hiểu về ngữ nghĩa Put là đưa ảnh ra, Xor là phép toán trên Bit. Còn về phương diện kỹ thuật số thì XorPut được hiểu như sau:

Nếu điểm ảnh trong Bitmap có màu trắng (số hiệu màu là 15) và nền có màu xanh da trời (số hiệu màu là 1) thì khi đưa ra màn hình điểm ảnh này sẽ có màu vàng bởi vì $15_{10} = 1111_2$, $1_{10} = 0001_2$. Phép toán Xor được thực hiện trên từng bit

1111_2 xor $0001_2 = 1110_2$

Số $1110_2 = 14_{10}$ ứng với màu vàng.

Tương tự như vậy nếu chọn phương thức đưa ảnh ra màn hình là AndPut với màu ảnh là xanh da trời (0001) , còn nền có màu đỏ (0100) thì:

0001_2 and $0100_2 = 0000_2$

nghĩa là ảnh sẽ có màu đen.

Thông thường chúng ta nên dùng "phương thức" là CopyPut nghĩa là màu ảnh sẽ hiện đè lên màu nền. Cách hiện màu này cho ta đúng màu sắc ảnh đã được lưu trữ trong Bitmap. Các phương thức còn lại mặc dù đưa được ảnh ra màn hình nhưng màu sắc không trung thực, không đúng với màu ảnh đã được lưu trữ trong Bitmap.

Xử dụng phương thức XorPut chúng ta có thể cho hiện ảnh và ẩn ảnh một cách dễ dàng. Để thấy rõ hơn chúng ta xét ví dụ sau:

Giả sử ảnh Bitmap được lưu trong một biến con trỏ P, ảnh có màu trắng và nền màn hình có màu đỏ. Thủ tục đưa ảnh ra màn hình bắt đầu từ vị trí tâm màn hình theo phương thức XorPut:

PutImage(320,240,P^,XorPut);

Với thủ tục trên ảnh đưa ra màn hình sẽ có màu xanh ngọc (số hiệu màu là 11) vì

1111_2 xor $0100_2 = 1011_2 = 11_{10}$

Nếu bây giờ chúng ta lại đưa tiếp ảnh đó ra màn hình tại cùng vị trí với ảnh đã có thì ảnh trên màn hình sẽ bị xoá, nói đúng hơn là ảnh đưa ra lần thứ hai sẽ có màu trùng với màu nền.

Trở lại ví dụ trên, ảnh trên màn hình đang có màu là 1011 còn ảnh trong Bitmap vẫn là màu 1111, khi đó màu ảnh trên màn hình sẽ là

1111_2 xor $1011_2 = 0100_2$ (nghĩa là màu đỏ trùng màu nền)

Từ đây chúng ta có thuật toán vẽ hình chuyển động như sau:

1- Vẽ hình tại toạ độ x,y theo phương thức XorPut

2- Lưu hình trên màn hình n miligiây

3- Xoá hình (bằng cách lặp lại bước 1)

4- Thay đổi toạ độ x := x+dx; y:=y+dy;

5- Quay lại bước 1

Chú ý: *khi vẽ hình chuyển động cần phải tạo các các điểm dừng để khi không muốn chuyển động có thể ngắt ngay chương trình.*

Dưới đây là chương trình vẽ một bầu trời sao, số ngôi sao trên bầu trời là 100 trong đó có 40 ngôi sao nhấp nháy, 20 ngôi sao nhấp nháy nhanh và 20 sao nhấp nháy chậm, các ngôi sao được phân bổ ngẫu nhiên. Trong chương trình còn thiết kế một đĩa bay, đĩa bay có 25 kích thước khác nhau, mỗi hình vẽ đĩa bay được cất vào một biến con trỏ P, các biến con trỏ này nằm trong một mảng con trỏ. Các đĩa bay xuất hiện ngẫu nhiên trên bầu trời, chuyển động nhỏ dần và biến mất, sau đó lại hiện lên tại một toạ độ nào đó. Chương trình dừng lại khi bấm một phím bất kỳ.

Với các máy tính có dung lượng bộ nhớ lớn (128 hoặc 256 Mb) chúng ta có thể tăng số hình đĩa bay lên 50 hình khi đó đĩa bay sẽ chuyển động mịn màng hơn.

Ví dụ 6.9

```
Program Bautroi_diabay;
Uses graph,crt;
Var
  i,j,gd,gm:integer;  a,x,y,xx,yy:word;
  p: array[1..30]  of pointer;
  m,n :array[1..40] of word;

  Procedure bautroi(a:word);
  var i,j:word; m,n :array[1..40] of word; ch:char;

  Begin
  for j:= 1 to 5 do
   Begin
    for i:= 1 to a do
    begin
```

```
      x:=random(639);  y:=random(479);
      if   i<=40 then  (*chon 40 ngoi sao nhap nhay*)
      begin
      m[i]:=x; n[i]:=y; (*ghi toa do sao nhap nhay vao mang*)
      end;
      putpixel(x,y,random(15));(*hien sao *)
     end;
  end;
end;

  Procedure nhapnhay;
  var i,j:word;  k:byte;
  Begin
    for i:= 1 to 40 do (*chon 40 ngoi sao nhap nhay*)
    begin
     m[i]:=random(639);
     n[i]:=random(479); (*ghi toa do sao nhap nhay vao mang*)
    end;
     setcolor(random(15));
     for j:=0 to 3 do
     For i:=1 to 20 do (*20 sao nhap nhay nhanh*)
     begin
     circle(m[i+j],n[i+j],1);
     delay(5);
     end;
     for j:=0 to 3 do
     For i:=21 to 40 do (*20 sao nhap nhay cham*)
     begin
     setcolor(random(4));
     circle(m[i+j],n[i+j],1);
     delay(10);
   end;
   End;

 procedure   vedia;
 Var xx,yy,x,y,x1,y1,x2,y2:integer; p: array[1..30]  of pointer;
    r,i,r1,r2,r3,r4,r5:word;
 Begin
 setcolor(14);setbkcolor(0);
          i:=1; x:= 50;  y:= 50;  {diem dau tien dia bay xuat hien}
```

```
for r:= 25 downto 1 do {chọn 25  đĩa với các kích thước giảm dần}
 Begin
 r1:=round(r/2);
 r2:=round(r/3);
 r3:=round(r/5);
 r4:=2*r2;
 r5:=round(r/10);
 Ellipse(x,y,0,360,r,r1);
 Ellipse(x,y-r3,194,346,r,r2);
 Line(x+r2,y-r2,x+r4,y-r4);
  circle(x+r4,y-r4,r5);
  line(x-r2,y-r2,x-r4,y-r4);
 circle(x-r4,y-r4,r5);
 x1:=x-r;
 y1:=y-r4-r5;
 x2:=x+r;
 y2:=y+r1;
 Getmem(p[i], imagesize(x1,y1,x2,y2));(*xac dinh kich thuoc anh*)
 getimage(x1,y1,x2,y2,p[i]^); (*cat anh vao bien con tro*)
 putimage(x1,y1,p[i]^,xorput); (*xoa anh*)
 i:=i+1;
 End;
 i:=1;
 While not keypressed do
 Begin     (*hien dia bay voi cac kich thuoc khac nhau*)
 putimage(x,y,p[i]^,xorput);
 delay(100);
 xx:=x; yy:=y;
 x:=x+random(25);
  y:=y+random(25);
 putimage(xx,yy,p[i]^,xorput);        (*xoa dia bay*)
  for j:= 1 to 40 do            (*tao su nhap nhay cua ngoi sao*)
   begin
     setcolor(random(15));
     circle(m[j],n[j],1);
   end;
 i:=i+1;
 if i>25 then    {da hien het cac hinh dia bay}
 begin
   x:=random(500);
```

```
   y:=random(400);
   i:=1;
  end;
end;
end;

BEGIN   {than chuong trinh chinh}
  gd:=detect; a:=100; {a la so ngoi sao tren bau troi}
  initgraph(gd,gm,'C:\tp\bgi');
  if graphresult<>grok then halt;
  bautroi(a);         {hien bau troi sao}
  while not keypressed do
  begin
  nhapnhay;   {thiet ke cac ngoi sao nhap nhay}
  vedia;   {goi dia bay}
  end;
  closegraph;
END.
```

7. Đồ thị hàm số

Giả thiết hàm y = f(x) xác định và liên tục trên đoạn [a,b], cần vẽ đồ thị hàm trong một khu vực nào đó trên màn hình. Hệ toạ độ quen thuộc mà chúng ta sử dụng là hệ xoy với gốc toạ độ nằm ở góc dưới bên trái, trục x hướng sang phải, trục y hướng lên trên.

Màn hình máy vi tính tuỳ theo kích thước và chế độ làm việc có thể là ma trận điểm từ 640x480 đến 1024x720. Gốc toạ độ của màn hình lại ở góc trên bên trái và trục x hướng theo chiều từ trên xuống dưới. Để thuận tiện cho việc vẽ và quan sát đồ thị chúng ta cần phải chuyển đổi từ toạ thực (như trong toán học) sang toạ độ màn hình.

Ký hiệu toạ độ thực của miền vẽ đồ thị điểm góc dưới trái là xmin, ymin và góc trên phải là xmax, ymax. Toạ độ màn hình góc trên trái là cmin , dmin, góc dưới phải là cmax, dmax.

Một điểm bất kỳ trên đồ thị có toạ độ thực là x, y và toạ độ màn hình là c, d. Có thể chứng minh rằng:

c = m*(x - xmin) + cmin

d = n*(y - ymin) + dmin

xmax, ymax

xmin, ymin

Trong đó:

p = (cmax - cmin)/(xmax - xmin)

q = (dmin - dmax)/(ymax- ymin)

Hiệu số (dmin - dmax) mang dấu âm cho biết chiều biến thiên của trục thẳng đứng giữa toạ độ màn hình và toạ độ thực là ngược nhau.

Khi vẽ đồ thị trên màn hình chúng ta sẽ dùng các công thức trên để chuyển đổi toạ độ tính được từ thực tế sang toạ độ màn hình.

Phương pháp vẽ đồ thị là xác định toạ độ từng điểm sau đó nối các điểm với nhau bằng các đoạn thẳng. Để đồ thị trơn tru thì số điểm phải thật nhiều, tuy nhiên chúng ta không thể tăng vô hạn số điểm. Giả sử chia đoạn [a,b] thành n đoạn nhỏ, chúng ta sẽ có n+1 điểm đánh số từ 0 đến n. Khi đó chiều rộng mỗi đoạn sẽ là h = (b - a)/n.

Hoành độ của một điểm bất kỳ trong toạ độ thực:

xi = a + h*i,

và tung độ tương ứng sẽ là f(xi).

Điều cần chú ý khi chọn các toạ độ thực và toạ độ màn hình là miền trên màn hình phải bao kín miền toạ độ thực. Muốn vậy chúng ta chọn:

xmin = a, xmax = b, ymin = min(yi), ymax = max(yi).

Việc khảo sát hàm số f(x) để tìm max và min theo kiểu toán học là không thực tế trong Pascal, do vậy có thể thay thế việc tìm min(yi) và max(yi) bằng việc lưu các giá trị f(xi) vào một mảng, sau đó tìm cực trị trong mảng.

Để chương trình có thể ứng dụng cho tất cả các hàm số chúng ta sẽ xây dựng một chương trình con lấy tên là HAM khai báo hàm và tính giá trị của hàm tại các toạ độ xi. Một chương trình con lấy tên là TOADO để tính các hệ số m, n, và tung độ các điểm chia (tức là giá trị của hàm f(x) tại các toạ độ xi).

Ví dụ 6.10

```
Program vedothi;
Uses graph;
  Const n=200;
  cmin=3; dmin=3;cmax=200; dmax=140;{ve o goc phan tu thu nhat}
  xmin=-5; xmax=5;
Var gd,gm:integer;
  ymin,ymax,h,m,p,q:real;
  y:Array[0..n] of real;


Function F(x:real) : real;
Begin
  {f:=x*x;}
   F:=x*x*x-2*x;
   End;

Procedure Toado;
Var i:integer; xi,yi:real;
Begin
  h:=(xmax-xmin)/n;
  For i:= 0 to n do
Begin
```

```
  xi:=xmin+i*h;
  yi:=f(xi);
  y[i]:=f(xi);
End;
  ymin:=y[0]; ymax:=-ymin;
   For i:=1 to n do
Begin
  If ymin>y[i] then ymin:=y[i];
  If ymax<y[i] then ymax:=y[i];
End;
  p:= (cmax-cmin)/(xmax-xmin);
  q:= (dmin-dmax)/(ymax-ymin);
End;

Procedure Ve;
Var i,c,d:integer; xi:real;
Begin
   c:=cmin;
  d:= round(q*(y[0]-ymin)+dmax);
  Moveto(c,d);
  For i:=1 to n do
Begin
  xi:=xmin+i*h;
  c:=Round(p*(xi-xmin)+cmin);
  d:=Round(q*(y[i]-ymin)+dmax);
  setcolor(red);
  Lineto(c,d)     {noi cac diem do thi}
End;
  setcolor(14);
  moveto(cmin,dmin); Lineto(cmax,dmin);{ve khung do thi}
  moveto(cmax,dmin); Lineto(cmax,dmax);
  moveto(cmax,dmax); Lineto(cmin,dmax);
  moveto(cmin,dmax); Lineto(cmin,dmin);
  d:=Round(q*(-ymin)+dmax);
  if (d>=dmin) and (d<=dmax) then
Begin
  line(cmin,d,cmax,d);{truc hoanh}
  line(cmax,d,cmax-6,d-3); {dau mui ten}
  line(cmax,d,cmax-6,d+3);
end;
  c:=Round(p*(-xmin)+cmin);
  if (c>=cmin) and (c<=cmax) then
begin
  Line(c,dmin,c,dmax); {truc tung}
  Line(c,dmin,c-3,dmin+6); {dau mui ten}
  Line(c,dmin,c+3,dmin+6);
end;        End;
```

```
BEGIN
  gd:=detect;
  Initgraph(gd,gm,'c:\tp70\bgi');
  if graphresult<>grok then halt;
  toado;
  ve;
   readln;
  closegraph;
END.
```

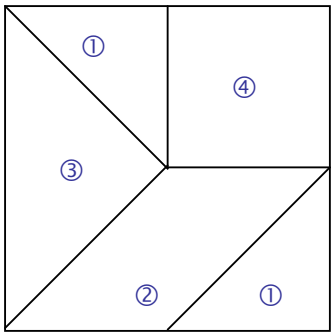
Với các hàm số cho theo tham số x = x(t), y = y(t), hoặc cho trong toạ độ cực r = f(φ) phương pháp vẽ cũng tương tự, chúng tôi xin dành cho bạn đọc như là những bài tập ứng dụng.

Bài tập ứng dụng chương 6

1. Nhập và cho chạy thử chương trình trong ví dụ 6.9 sau đó thiết kế thêm một tên lửa. Cho tên lửa phóng từ góc dưới bên phải màn hình lên góc trên bên trái. Luồng khói phía đuôi dài bằng 3 lần chiều dài tên lửa.

2. Cho các ký tự của dòng chữ "Happy Birth Day" hiện ngẫu nhiên và nhấp nháy trên màn hình, tiếp đó cho chúng chuyển động đan xen và thu dần về tâm (320,240). Cuối cùng các ký tự lại từ tâm chạy ra trên một quỹ đạo tròn và đã sắp xếp thành dòng chữ hoàn chỉnh.

3. Chọn kích thước và thiết kế các hình đơn giản 1, 2, 3, 4 (xem hình vẽ), mỗi hình cất vào một biến con trỏ. Cho hiện các hình tại các vị trí tuỳ ý sao cho hình vừa chuyển động vừa tự quay, cuối cùng các hình tự ghép với nhau thành một hình vuông ở giữa màn hình



4. Lập chương trình vè đồ thị hàm số cho dưới dạng tham số

x = x(t), y =y(t)

5. Lập chương trình vè đồ thị hàm số cho dưới dạng toạ độ cực

r = f(φ)

6. Cho hàm y = f(x) xác định và liên tục trong khoảng (a,b). Lập chương trình nhập dạng một hàm cụ thể. Vẽ đồ thị hàm sau đó tính tích phân xác định

$$\int_a^b f(x)dx$$

Phụ lục 1

Bảng mã ASCII

Ngôn ngữ Pascal sử dụng các ký tự trong bảng mã ASCII để xây dựng nên các từ khoá, tên chuẩn, các hàm, thủ tục hoặc các lệnh có cấu trúc. Bảng mã ASCII gồm 256 ký tự được đánh số từ 0 đến 255 và được phân thành các nhóm ký tự sau:

1. Các ký tự điều khiển:

32 ký tự đầu tiên(từ 0 đến 31) là các ký tự điều khiển, các ký tự này không in được ra màn hình hoặc máy in, ví dụ: ký tự Enter(mã 13), ký tự ESC(mã 27).

2. Các ký tự đặc biệt:

Các ký tự có mã ASCII trong khoảng 32-47, 58-64, 91-96, và 123-127 là các ký tự đặc biệt như dấu chấm, dấu phẩy, ký tự @, ký tự Del,...

3. Nhóm chữ số:

Các ký tự có mã ASCII trong khoảng 48-57 là 10 chữ số thập phân 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

4. Nhóm chữ cái:

+ Các ký tự có mã ASCII trong khoảng 65-90 là các chữ cái in từ A đến Z

+ Các ký tự có mã ASCII trong khoảng 97-122 là các chữ cái nhỏ từ a đến z

Các chữ cái thường có mã ASCII lớn hơn 32 so với chữ in tương ứng. Ví dụ mã ASCII của a là 97 còn của A là 65.

5. Nhóm ký tự đồ hoạ

Các ký tự có mã ASCII trong khoảng 128-255 là các ký tự đồ hoạ.

Để hiện toàn bộ các ký tự của bảng mã lên màn hình (trừ 32 ký tự đầu) chúng ta có thể cho chạy chương trình sau:

```
Program Ma_ascii;
  Var n : Byte;
  Begin
     For n:=32 to 255 do   Write(chr(n),'  ');
   End.
```

Dưới đây là bảng mã ASCII và hình dạng của 128 ký tự đầu tiên.

Bảng mã ASCII

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
0	NUI	33	!	66	B
1	SOH	34	“	67	C
2	STX	35	#	68	D
3	ETX	36	$	69	E
4	EOT	37	%	70	F
5	ENQ	38	&	71	G
6	ACK	39	’	72	H
7	BEL	40	(	73	I
8	BS	41	)	74	J
9	HT	42	*	75	K
10	LF	43	+	76	L
11	VT	44	,	77	M
12	FF	45	-	78	N
13	CR	46	.	79	O
14	SO	47	/	80	P
15	SI	48	0	81	Q
16	DLE	49	1	82	R
17	DC1	50	2	83	S
18	DC2	51	3	84	T
19	DC3	52	4	85	U
20	DC4	53	5	86	V
21	NAK	54	6	87	W
22	SYN	55	7	88	X
23	ETB	56	8	89	Y
24	CAN	57	9	90	Z
25	EM	58	:	91	[
26	SUB	59	;	92	\
27	ESC	60	<	93	]
28	FS	61	=	94	^
29	GS	62	>	95	_
30	RS	63	?	96	‘
31	US	64	@	97	a
32	SPACE	65	A	98	b

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
99	c	109	m	119	w
100	d	110	n	120	x
101	e	111	o	121	y
102	f	112	p	122	z
103	g	113	q	123	{
104	h	114	r	124	|
105	i	115	s	125	}
106	j	116	t	126	~
107	k	117	u	127	DEL
108	l	118	v		

Phụ lục 2

Tóm tắt các thủ tục và hàm của Turbo Pascal 7.0

Toàn bộ các thủ tục, hàm và định hướng biên dịch của Pascal có thể tìm hiểu trong phần Help trên thực đơn chính. Tổng cộng có 65 trang màn hình, mỗi trang được chia đôi và có 21 dòng tức là có khoảng 2500 mục cần nghiên cứu. Số lượng này ngay cả với những người chuyên nghiệp cũng không thể nắm bắt hết được. Trong phụ lục này chúng tôi chỉ chọn ra một số nhỏ những hàm và thủ tục thông dụng, khi cần biết ý nghĩa và cách dùng của một hàm hay thủ tục nào đó bạn đọc chỉ cần viết tên hàm hay thủ tục đó lên màn hình soạn thảo Pascal sau đó bấm tổ hợp phím Ctrl - F1, Pascal sẽ hiện lên các chỉ dẫn và ví dụ minh hoạ.

1. Hàm ABS

Cú pháp: Abs(r: Real) : Real; Abs(i: Integer) : Integer;

Công dụng: Cho giá trị tuyệt đối của đối số.

2. Hàm ADDR

Cú pháp: Addr(Var Variable) : Pointer;

Công dụng: Cho địa chỉ của biến.

3. Thủ tục APPEND

Cú pháp: Append(VAR F : Text);

Công dụng: Mở tệp văn bản để ghi bổ sung và định vị con trỏ tại cuối tệp.

4. Thủ tục ARC(Graph Unit)

Cú pháp: Arc(x,y:integer;gd,gc,R:word);

Công dụng: Vẽ cung tròn tâm(x,y) bán kính R, từ góc gd đến góc gc.

5. Hàm ARCTAN

Cú pháp: Arctan(r: Real) : Real;

Công dụng: Cho Arctangent của đối r.

6. Thủ tục ASSIGN

Cú pháp: Assign(var F : file; Name String);

Công dụng: Liên kết biến tệp F với tệp có tên chỉ định trong Name.

7. Thủ tục ASSIGNCRT(CRT unit)

Cú pháp: AssignCrt(Var F : text);

Công dụng: Thay việc gửi kết quả ra màn hình bằng việc lại đưa vào tệp F.

8. Thủ tục BAR(Graph unit)

Cú pháp: Bar(x1,y1,x2,y2:integer);

Công dụng: Vẽ vùng chữ nhật góc trên trái (x1,y1) góc dưới phải (x2,y2).

9. Thủ tục BARD3(Graph Unit)

Cú pháp: Bar3d(x1,y1,x2,y2:integer; Depth:word; Top:Boolean);

Công dụng: Vẽ hình hộp 3 chiều có tô mầu. Chiều sâu là Depth Pixel (điểm ảnh). Nếu Top là True thì hộp có nắp, nếu Top là False thì hộp không nắp.

10. Thủ tục BLOCKREAD

Cú pháp: Blockread(Var F:file; Var B:type; N:Integer; <KQ:Word>);

Công dụng: Đọc N mẫu tin từ tệp không kiểu F vào bộ nhớ (qua biến B). KQ cho biết số mẫu tin thực sự đọc được.

11. Thủ tục BLOCKWRITE

Cú pháp: Blockwrite(Var F:file; Var B:type; N:Integer);

Công dụng: Ghi N mẫu tin từ vùng đệm B ra tệp không kiểu F.

12. Thủ tục CHDIR

Cú pháp: ChDir(S:String);

Công dụng: Chuyển đổi thư mục hiện thời sang thư mục có đường dẫn S.

13. Hàm CHR

Cú pháp: Chr(I: Integer);

Công dụng: Cho ký tự có mã ASCII bằng I.

14. Thủ tục CIRCLE (Graph Unit)

Cú pháp: Circle(x,y:Integer; R:word);

Công dụng: Vẽ đường tròn tâm(x,y) bán kính R.

15. Thủ tục CLEARDEVICE(Graph Unit)

Cú pháp: ClearDevice;

Công dụng: Xoá màn hình đồ hoạ.

16. Thủ tục CLEARVIEWPORT(Graph Unit)

Cú pháp: Clearviewport;

Công dụng: Xóa Viewport hiện hành.

17. Thủ tục CLOSE

Cú pháp: Close(Var F:File);

Công dụng: Đóng tệp.

18. Thủ tục CLOSEGRAPH(Graph Unit)

Cú pháp: CloseGraph;

Công dụng: Ngừng chế độ đồ hoạ, chuyển về chế độ Text. Thủ tục này cũng giải phóng vùng nhớ do hệ đồ hoạ sử dụng.

19. Thủ tục CLREOL(CRT Unit)

Cú pháp: Clreol;

Công dụng: Xóa ký tự từ vị trí con trỏ (cursor) đến cuối dòng hiện hành.

20. Thủ tục CLRSCR(CRT Unit)

Cú pháp: Clrscr;

Công dụng: Xoá màn hình trong chế độ văn bản và đưa con trỏ về vị trí(1,1).

21. Hàm CONCAT

Cú pháp: Concat(S1,S2,…,Sn:String);

Công dụng: Tạo chuỗi mới bằng cách nối các chuỗi S1,…, Sn. Nếu chiều dài của chuỗi kết quả lớn hơn 255 ký tự, Pascal sẽ cắt bỏ các ký tự từ 256 trở đi .

22. Hàm COPY

Cú pháp: Copy(S:String; P,L:Integer):String;

Công dụng: Trích từ chuỗi S một chuỗi con dài l ký tự bắt đầu từ ký tự thứ P.

23. Hàm COS

Cú pháp: Cos(X:Real): Real;

Công dụng: Cho cosine của góc X.

24. Hàm CSEG

Cú pháp: Cseg: word;

Công dụng: Cho địa chỉ của đoạn mã chương trình.

25. Thủ tục DEC

Cú pháp: Dec(Var X: SCalar; n:longInt);

Công dụng: Giảm biến X đi n. Nếu không có n thì X sẽ bị giảm đi 1.

26. Thủ tục DELAY(CRT Unit)

Cú pháp: Delay(s:word);

Công dụng: Tạm dừng chương trình trong s miligiây , s<=65535

27. Thủ tục DELETE

Cú pháp: Delet(S:String; P,L:integer);

Công dụng: Xoá L ký tự từ vị trí P ra khỏi chuỗi S

28. Thủ tục DELLINE(CRT Unit)

Cú pháp: Delline;

Công dụng: Xoá dòng chứa con trỏ. Các dòng phía dưới sẽ dồn lên phía trên.

29. Thủ tục DETECTGRAPH(CRT Unit)

Cú pháp: DetectGraph(Var GD,GM:Integer);

Công dụng: Phát hiện kiểu màn hình, mode đồ hoạ và gán cho các biến GD,GM.

30. Hàm DISKFREE(DOS Unit)

Cú pháp: DiskFree(Drive:word):longint;

Công dụng: Cho biết số byte còn trống trên ổ đĩa chỉ định trong Drive. Drive = 0 ứng với ổ ngầm định ,Drive = 1 cho ổ A, 2 cho ổ B.

31. Hàm DISKSIZE(DOS Unit)

Cú pháp: Disksize(Drive:word):longint;

Công dụng: Cho biết dung lượng đĩa tính theo byte. Tham số Drive xác định ổ đĩa (Drive = 1 cho ổ A, 2 cho ổ B, 0 cho ổ ngầm định).

32. Thủ tục DISPOSE

Cú pháp: Dispose(P:Pointer);

Công dụng: Giải toả vùng heap cấp phát cho biến con trỏ P. Dispose dùng kèm với New.

33. Hàm DOSEXITCODE(DOS Unit)

Cú pháp: Dosexitcode:word;

Công dụng: Cho mã lỗi của tiến trình con (0 = chấm dứt bình thường, 1 = chấm dứt do CTRL – C, 2 = chấm dứt do lỗi thiết bị, 3 = chấm dứt do thủ tục KEEP).

34. Hàm DOSVERSION(DOS Unit)

Cú pháp: Dosversion : word;

Công dụng: Cho số hiệu phiên bản (Version) của hệ điều hành. Phần chính nằm ở byte cao, phần phụ nằm ở byte thấp.

35. Thủ tục DRAWPOLY(Graph Unit)

Cú pháp: Drawpoly(n:word; Var Polypoints);

Công dụng: Vẽ đa giác n đỉnh. Mảng Polypoints chứa toạ độ các đỉnh của đa giác.

36. Hàm ENVCOUNT(DOS Unit)

Cú pháp: EnvCount:integer;

Công dụng: Cho số lượng chuỗi được định nghĩa trong môi trường DOS

37. Hàm EOF

Cú pháp: Eof(F:File): Boolean;

Công dụng: Cho giá trị TRUE nếu con trỏ định vị ở cuối tệp , các trường hợp khác cho kết quả FLASE.

38. Hàm EOLN

Cú pháp: Eoln(F:File): Boolean;

Công dụng: Cho giá trị TRUE nếu con trỏ nằm cuối dòng(CR/LF) hoặc ở cuối tệp, cho FLASE trong trường hợp trái lại.

39. Thủ tục ERASE

Cú pháp: Erase(F:File);

Công dụng: Xoá tệp và các thông tin về tệp F khỏi đĩa .

40. Thủ tục EXEC

Cú pháp: Exec(Path,Cmdline:String);

Công dụng: Thực hiện tệp có tên nằm trong path với các tham biến dòng lệnh được định nghĩa trong Cmdline.

41. Thủ tục EXIT

Cú pháp: Exit;

Công dụng: Thoát khỏi khối hiện thời (khối có thể là chương trình con, cấu trúc lặp) nếu exit nằm trong chương trình chính thì thoát khỏi chương trình).

42. Hàm EXP

Cú pháp: EXP(X:Real):Real;

Công dụng: Luỹ thừa e của đối số X.

43. Hàm FEXPAND(DOS Unit)

Cú pháp: Fexpand(P:PathStr):PathStr;

Công dụng: Cho đường dẫn của tệp P . Ví dụ tệp A1.pas hiện lưu trong thư mục c:\tp\bin. Trong phần khai báo chương trình có lệnh: Uses Dos; Khi đó lệnh

Writeln(Fexpand('a1.pas') cho kết quả là C:\TP\BIN\A1.PAS

44. Hàm FILEPOS

Cú pháp: FilePos(F:File):integer;

Công dụng: Cho số hiệu bản ghi của tệp F mà con trỏ đang trỏ tới.

45. Hàm FILESIZE

Cú pháp: FileSize(F:file):integer;

Công dụng: Cho số lượng bản ghi đang có trong tệp F.

46. Thủ tục FILLELIPES(Graph Unit)

Cú pháp: Fillelipse(x,y:integer; a,b:word);

Công dụng: Vẽ elipse có tâm(x,y) với hai bán trục là a, b.

47. Thủ tục FILLPOLY(Graph Unit)

Cú pháp: Fillpoly(n:word; Var Polypoints);

Công dụng: Vẽ đa giác n đỉnh. Mảng Polypoints chứa toạ độ các đỉnh của đa giác.

48. Thủ tục FLOODFILL(Graph Unit)

Cú pháp: FloodFill(x,y:integer; border:word);

Công dụng: Tô màu một vùng khép kín trên màn hình đồ hoạ nếu toạ độ(x,y) nằm bên trong vùng. Nếu x,y nằm ngoài vùng thì miền bên ngoài được tô.

49. Thủ tục FLUSH

Cú pháp: Flush(Var F:Text);

Công dụng: Đẩy ra đĩa tất cả kết quả đang nằm trong vùng đệm cho tệp F.

50. Hàm FRAC

Cú pháp: Frac(x:Real):Real;

Công dụng: Cho phần lẻ thập phân của đối số x.

51. Thủ tục FREEMEM

Cú pháp: Freemem(Var F:pointer; I:Integer);

Công dụng: Giải phóng I byte của vùng heap đã cấp phát cho biến con trỏ F bằng thủ tục GetMem.

52. Thủ tục GETARCCOORDS(Graph Unit)

Cú pháp: Getarrccoords(Var ArcCoords : ArcCooType);

Công dụng: Cho biết các toạ độ đã được lệnh Arc hay Ellipse trước đó sử dụng. Cấu trúc của ArcCoordsType như sau:

```
TYPE
  ArcCoordstype = Recode
        x, y, XStard, YStart, Xend, Yend:Integer;
        End;
```

53. Thủ tục GETASPECTRATIO(Graph Unit)

Cú pháp:

```
GetAspectratio(Var Xasp, Yasp:word);
```

Công dụng: Cho độ phân giải của màn hình đồ hoạ (chứa trong các biến Xasp và Yasp).

54. Hàm GETBKCOLOR(Graph Unit)

Cú pháp: GetBKColor : word;

Công dụng: chọn mầu nền , có thể thay thế số hiệu màu bởi tên hằng màu.

55. Hàm GETCOLOR(Graph Unit)

Cú pháp: GetColor:word;

Công dụng: Chọn màu nét vẽ, có thể thay thế số hiệu màu bởi tên hằng màu.

56. Thủ tục GETDATE(DOS Unit)

Cú pháp: GetDate(Var year, Month, Day, Dayofweek:word);

Công dụng: Cho ngày tháng hiện hành của hệ thống.

57. Thủ tục GETDEFAULTPALETTE(Graph Unit)

Cú pháp: Getdefaultpalette(Var Pal:PaletteType);

Công dụng: Cho trong Pal bảng mầu Ngầm định cho trình điều khiển đồ hoạ hiện hành. Cấu trúc của kiểu PaletteType như sau:

```
TYPE
  PaletteType = Record
        Size:Byte;
        Colors: Array[0..Maxcolor] of Shortint;
       End;
```

58. Thủ tục GETDIR

Cú pháp: GetDir(d: Byte; Var S: String);

Công dụng: Cho biết thư mục hiện hành trên ổ đĩa được chỉ định trong d. Tên thư mục được cho trong S (nếu d = 0 thì GetDir sẽ tìm trong ổ đĩa ngầm định, còn nếu d = 1, 2, 3 ứng với các ổ đĩa A, B, C).

59. Hàm GETDRIVERNAME(Graph Unit)

Cú pháp: GetDriverName : String;

Công dụng: Cho tên của trình điều khiển đồ hoạ hiện hành.

60. Thủ tục GETFATTR(DOS Unit)

Cú pháp: GetFAttr(Var F; Var Attr: word);

Công dụng: Cho biết thuộc tính của tệp F. Trước khi gọi thủ tục này tệp phải được gán thuộc tính nhưng không mở.

61. Thủ tục GETFILLPATTERN(Graph Unit)

Cú pháp: GetFillPattern(Var FP : FillpatternType);

Công dụng: Cho trong FP số hiệu của mẫu tô hiện hành (màu nền chỉ có 8 giá trị). Cấu trúc của FillPartternType là:

```
TYPE
   FillPartternType = Array[1..8] of Byte
```

62. Thủ tục GETFILLSETTINGS(Graph Unit)

Cú pháp: GetFillSettings(Var FS : FillSettingsType);

Công dụng: Cho trong FS mầu và mẫu tô hiện hành. Cấu trúc của FillSettingsType là:

```
TYPE
   FillSettingsType = Record
         Pattern : word;   Color : word;
        End;
```

63. Thủ tục GETFTIME(DOS Unit)

Cú pháp: GetFTime(Var F; Var Time : Longint);

Công dụng: cho biết ngày tháng và giờ mà tệp F được ghi.

64. Hàm GETGRAPHMODE(Graph Unit)

Cú pháp: GetGraphMode : Integer;

Công dụng: cho biết kiểu đồ hoạ hiện hành.

65. Thủ tục GETIMAGE(Graph Unit)

Cú pháp: Getimage(x1,y1,x2,y2 : integer; Var Bitmap);

Công dụng: lưu trữ phần hình chữ nhật (x1,y1,x2,y2) trên màn hình đồ hoạ vào BitMap.

66. Thủ tục GETLINESETTINGS(Graph Unit)

Cú pháp: GetlineSettings(Var LST : LineSettingsType);

Công dụng: Cho trong LST giá trị hiện hành của dạng đường , mẫu, bề dày. Cấu trúc của LineSettingsType như sau:

```
TYPE
   LineSettingsType = Record
         LineStyle , Pattern , Thickness : word;
        End;
```

67. Hàm GetMaxcolor(Graph Unit)

Cú pháp: GetMaxcolor : word;

Công dụng: cho số hiệu cao nhất của mã mầu trong bảng mầu hiện hành.

68. Hàm GETMAXMODE(Graph Unit)

Cú pháp: GetMaxMode : word;

Công dụng: Hàm cho giá trị là một số kiểu word, số này ứng với kiểu đồ hoạ có độ phận giải cao nhất của màn hình đang dùng. Hầu hết các màn hình có kiểu đồ hoạ từ 0 đến Getmaxmode.

69. Hàm GETMAXX(Graph Unit)

Cú pháp: GetMaxX : integer;

Công dụng: cho hoành độ lớn nhất của màn hình đồ hoạ hiện hành.

70. Hàm GETMAXY(Graph Unit)

Cú pháp: GetMaxY : Integer;

Công dụng: cho tung độ lớn nhất của kiểu màn hình đồ hoạ hiện hành.

71. Thủ tục GETMEM

Cú pháp: GetMem(Var P : Pointer; I: integer);

Công dụng: Cấp phát cho biến con trỏ P một vùng nhớ có dung lượng I byte.

72. Hàm GETMODENAME(Graph Unit)

Cú pháp: GetModeName(ModeNumber : word) : String;

Công dụng: cho một chuỗi mô tả kiểu đồ hoạ ứng với ModeNumber.

73. Thủ tục GETMODERANGE(Graph Unit)

Cú pháp: GetModeRange(GraphDriver:Integer; Var Lomode, Himode:Integer);

Công dụng: Cho kiểu phân giải thấp nhất(LoMode) và cao nhất (HiMode) của trình điều khiển đồ hoạ ứng với GraphDriver.

74. Thủ tục GETPALETTE(Graph Unit)

Cú pháp: GetPalette(Var P : PaletteType);

Công dụng: trả về giá trị là một biến kiểu bản ghi gồm 2 trường trường size cho biết số lượng màu trong Palette, trường color cho biết mã màu hiện hành. PaletteType là kiểu Record được định nghĩa như sau:

```
TYPE
  PaleteType = Record
      Size : Byte;
      Colors : Array[0..15] of Shortint;
    End;
```

75. Hàm GETPALETTESIZE(Graph Unit)

Cú pháp: GetPaletteSize : word;

Công dụng: Cho số lượng sơ đồ màu lớn nhất mà kiểu màn hình đồ hoạ hiện hành có thể cung cấp. Ví dụ màn hình EGA hàm này cho kết quả là 16.

76. Hàm GETPIXEL(Graph Unit)

Cú pháp: GetPixel(x,y:Integer);

Công dụng: cho mầu của điểm ảnh tại toạ độ(x,y).

77. Thủ tục GETTEXTSETTINGS(Graph Unit)

Cú pháp: GetTextSettings(Var TS : GettextsettingsType);

Công dụng: Cho biết các giá trị định dạng văn bản hiện hành. Các giá trị này lưu vào biến TS kiểu bản ghi. Cấu trúc của bản ghi là:

```
TYPE
  GetTextSettingsType = Record;
        Font, Direction, Charsize, Horiz, Vert:word;
            End;
```

78. Thủ tục GETTIME(DOS Unit)

Cú pháp: GetTime(Var Hour,Minute,Second,Sec100:word);

Công dụng: Cho giờ hệ thống.

79. Hàm GETX(Graph Unit)

Cú pháp: Getx : Integer;

Công dụng: Cho toạ hoành độ cuả con trỏ hiện hành.

80. Hàm GETY(Graph Unit)

Cú pháp: Gety : Integer;

Công dụng: Cho tung độ cuả con trỏ hiện hành.

81. Thủ tục GOTOXY(CRT Unit)

Cú pháp: Gotoxy(x,y: Integer);

Công dụng: Đưa Curson (con trỏ) đến toạ độ(x,y);

82. Hàm GRAPHERRORMSG(Graph Unit)

Cú pháp: GraphErrorMsg(Code : Integer):String;

Công dụng: Cho thông báo lỗi đồ hoạ ứng với mã lỗi trong Code.

83. Thủ tục GRAPHDEFAULTS(Graph Unit)

Cú pháp: GraphDefaults;

Công dụng: Trả về các giá trị ngầm định cho định dạng đồ hoạ.

84. Thủ tục GRAPHRESULT(Graph Unit)

Cú pháp: Graphresult : Integer;

Công dụng: Cho mã lỗi của thủ tục đồ hoạ sau cùng.

85. Thủ tục HALT

Cú pháp: Halt;

Công dụng: Dừng ngay chương trình.

86. Hàm HI

Cú pháp: Hi(I : Integer): Byte;

Công dụng: Cho Byte cao trong số nguyên I.

87. Thủ tục HIGHVIDEO(CRT Unit)

Cú pháp: HighVideo;

Công dụng: tăng độ sáng màn hình .

88. Hàm IMAGESIZE(Graph Unit)

Cú pháp: ImageSize(x1,y1,x2,y2 : Integer);

Công dụng: Cho dung lượng tính theo byte để lưu trữ một miền ảnh Bitmap trong vùng chữ nhật x1,y1,x2,y2.

89. Thủ tục INC

Cú pháp: Inc(Var x [;n : longint]);

Công dụng: Tăng giá trị x thêm n (x phải thuộc kiểu nguyên). Nếu bỏ qua n thì x sẽ tăng lên 1. Ví dụ:

```
Var x:integer;
Begin
X:=15;    Inc(x);  {x sẽ có giá trị là 16}
Inc(x,5);          {x sẽ có giá trị là 20}
End;
```

90. Thủ tục INITGRAPH(Graph Unit)

Cú pháp: InitGraph(Var GD, GM:Integer; Driverpath:string);

Công dụng: Khởi tạo môi trường đồ hoạ và đưa phần cứng vào mode đồ hoạ. GD là trình điều khiển đồ hoạ, GM là mode đồ hoạ. Nếu GD bằng Detect, thủ tục tự động kiểm tra phần cứng và chọn các thông số tối ưu. Thủ tục sẽ tìm các tệp BGI trong đường dẫn DriverPath.

91. Hàm INSERT

Cú pháp: Insert(Source:String; Var Target:String; Index:Integer);

Công dụng: Chèn chuỗi Source vào chuỗi Target ở vị trí Index.

92. Thủ tục InsLine(CRT Unit)

Cú pháp: InsLine;

Công dụng: chèn một dòng trống vào vị trí hiện hành của con trỏ trên màn hình.

93. Hàm INT

Cú pháp: Int(x:Real): Real;

Công dụng: Cho phần nguyên của x

94. Hàm INTR(DOS Unit)

Cú pháp: Intr(Fune:Byte; Var Regs:Registers);

Công dụng: Gọi ngắt có số hiệu Fune của Bios với các thanh ghi được định nghĩa qua Regs.

95. Hàm IORESULT(CRT Unit)

Cú pháp: IOResult : word;

Công dụng: Cho mã lỗi khi thực hiện thao tác vào/ra. Nếu IOResult khác không thì có nghĩa là đã xảy ra lỗi.

96. Thủ tục KEEP(DOS Unit)

Cú pháp: Keep(Exitcode:word);

Công dụng: Kết thúc chương trình nhưng vẫn giữ nó thường trú.

97. Hàm KEYPRESSED(CRT Unit)

Cú pháp: Keypressed : Boolean;

Công dụng: Cho giá trị TRUE khi có một phím được bấm.

98. Hàm LENGTH

Cú pháp: Leng(S:String):Integer;

Công dụng: Cho chiều dài của chuỗi S

99. Thủ tục LINE(Graph Unit)

Cú pháp: Line(x1,y1,x2,y2:Integer);

Công dụng: Vẽ đường thẳng từ (x1,y1) đến (x2,y2).

100. Thủ tục LINEREL(Graph Unit)

Cú pháp: Linerel(Dx,Dy:Integer);

Công dụng: Vẽ đường thẳng từ vị trí hiện tại của con trỏ (toạ độ x,y) đến điểm (x+Dx,y+Dy).

101. LINETO(Graph Unit)

Cú pháp: Lineto(x,y:Integer);

Công dụng: Vẽ đường thẳng từ vị trí hiện tại của con trỏ đến điểm(x,y).

102. Hàm LN

Cú pháp: Ln(Var x:Real):Real;

Công dụng: Cho logarit tự nhiên của x.

103. Thủ tục LOWVIDEO(CRT Unit)

Cú pháp: LowVideo;

Công dụng: giảm độ sáng màn hình.

104. Thủ tục MARK

Cú pháp: Mark(P:pointer);

Công dụng: đánh dấu địa chỉ đỉnh heap trong con trỏ P

105. Hàm MAXAVAIL

Cú pháp: MaxAvail : LongInt;

Công dụng: Cho kích thước khối lớn nhất của vùng nhớ chưa cấp phát của Heap.

106. Hàm MEMAVAIL

Cú pháp: MemAvail : ;

Công dụng: cho tổng dung lương của các khối nhớ tự do trong Heap.

107. Thủ tục MKDIR

Cú pháp: MKDir(S:String);

Công dụng: Tạo thư mục mới với tên chỉ định trong S.

108. Thủ tục MOVE

Cú pháp: Move(Var V1,V2,n:Integer);

Công dụng: chép n byte từ biến V1 đến biến V2.

109. Thủ tục MOVEREL(Graph Unit)

Cú pháp: MoveRel(Dx,Dy:Integer);

Công dụng: chuyển con trỏ từ toạ độ hiện thời (x,y) đến toạ độ x+dx, y+dy.

110. Thủ tục MOVETO

Cú pháp: MoveTo(x,y:Integer);

Công dụng: định vị con trỏ tại toạ độ(x,y).

111. Thủ tục MSDOS(DOS Unit)

Cú pháp: MSDOS(Var Regs:Registers);

Công dụng: gọi phục vụ DOS với tập giá trị của các thanh ghi trong Regs.

112. Thủ tục NEW

Cú pháp: New(Var P:Pointer);

Công dụng: cấp phát vùng heap cho con trỏ P.

113. Thủ tục NORMVIDEO(CRT Unit)

Cú pháp: NormVideo;

Công dụng: khôi phục thuộc tính màn hình ngầm định.

114. Thủ tục NOSOUND(CRT Unit)

Cú pháp: NoSound;

Công dụng: tắt âm thanh phát ra loa.

115. Hàm ODD

Cú pháp: Odd(I:Integer): Boolean;

Công dụng: Cho True khi I là số lẻ và False khi I là số chẵn

116. Hàm OFS

Cú pháp: Ofs(<Variable, Procedure, or Function>):Integer;

Công dụng: Cho Offset địa chỉ vùng nhớ của biến, thủ tục hay hàm.

117. Hàm ORD

Cú pháp: Ord(S:Scalar):Integer;

Công dụng: Cho số thứ tự của S (có kiểu vô hướng đếm được).

118. Thủ tục OUTTEXT(Graph Unit)

Cú pháp: Outtext(Textstring):String;

Công dụng: viết chuỗi Textstring trong chế độ đồ hoạ. Cần chọn trước kiểu chữ, kích thước, màu sắc và hướng viết.

119. Thủ tục OUTTEXTXY(Graph Unit)

Cú pháp: OuttextXY(x,y:Integer; TextString:String);

Công dụng: Hiển thị chuỗi TextString ở vị trí(x,y). Cần chọn trước kiểu chữ, kích thước, màu sắc và hướng viết.

120. Thủ tục OVRCLEARBUF(Overlay Unit)

Cú pháp: OvrClearBuf;

Công dụng: xoá sạch vùng đệm Overlay.

121. Hàm OVRGETBUF(Overlay Unit)

Cú pháp: OvrGetBuf : LongInt;

Công dụng: Cho kích thước của vùng đệm Overlay.

122. Thủ tục OVRINIT(Overlay Unit)

Cú pháp: OVRInit(FileName:String);

Công dụng: Khởi tạo quá trình quản lý Overlay và mở tệp Overlay được chỉ định trong FileName. Thủ tục này cần được gọi tới trước bất cứ thủ tục Overlay nào.

123. Thủ tục PACKTIME(DOS Unit)

Cú pháp: PackTime(Var DT: DateTime; Var Time:Longint);

Công dụng: Chuyển hoá thông tin ngày, tháng và giờ trong DT (kiểu bản ghi) thành số 4 Byte và gửi vào biến Time .

124. Thủ tục OVRINITEMS(Overlay Unit)

Cú pháp: OvrInitEMS(fileName:String);

Công dụng: Nạp tệp Overlay vào vùng nhớ mở rộng (EMS) nếu còn đủ .

125. Thủ tục OVRSETBUF(Overlay Unit)

Cú pháp: OvrsetBuf(BufSize:Integer);

Công dụng: Cấp phát Bufsize byte cho vùng đệm Overlay. BufSize phải nhỏ hơn vùng đệm Overlay ngầm định.

126. Hàm PARAMSTR

Cú pháp: ParamStr(I:word):String;

Công dụng: Cho tham số dòng lệnh thứ I. Paramstr(0) cho đường dẫn tên tệp chương trình hiện hành (c:\tp\bin\turbo.exe)

127. Hàm PI

Cú pháp: Pi:Real;

Công dụng: Cho hằng số Pi

128. Thủ tục PIESLICE(Graph Unit)

Cú pháp: Pieslice(x,y:Integer; a1,a2,R:word);

Công dụng: Vẽ hình quạt tâm(x,y) bán kính R, từ góc a1 đến góc a2.

129. Hàm POS

Cú pháp: Pos(SubS,S:String):Integer;

Công dụng: Cho vị trí bắt đầu của chuỗi con SubS trong S. Nếu SubS không nằm trong S thì hàm POS cho giá trị 0.

130. Hàm PRED

Cú pháp: Pred(S:Scalar):Integer;

Công dụng: Cho phần tử phía trước phần tử S (S phải thuộc kiểu vô hướng đếm được). **Pred(5) = 4.**

131 Thủ tục PUTIMAGE(Graph Unit)

Cú pháp: PutImage(x,y:Integer; Var BitMap; BitBlt:word);

Công dụng: Hiện ảnh chứa trong BitMap bắt đầu tại điểm (x,y). BitBlt chỉ định phương thức hiện ảnh và có thể nhận giá trị kiểu số hoặc hằng ký tự như sau:

COPYPUT = 0; XORPUT = 1; ORPUT = 2; ANDPUT = 3; NOTPUT = 4;

132. Thủ tục PUTPIXEL(Graph Unit)

Cú pháp: PutPixel(x,y:Integer; Color:word);

Công dụng: Vẽ một điểm ảnh (Pixel) tại toạ độ (x,y) theo mầu chỉ định bởi color.

133. Hàm RANDOM

Cú pháp: Random(I:word):word;

Công dụng: Cho một số ngẫu nhiên k với 0<=k<I.

134. Hàm RAMDOMIZE

Cú pháp: Randomize;

Công dụng: Khởi tạo giá trị ban đầu của bộ tạo số ngẫu nhiên trong đồng hồ hệ thống. Giá trị này được sử dụng khi có lời gọi hàm Random. Nếu không có thủ tục Randomize thì giá trị mà hàm Random tạo ra sẽ giống nhau mỗi khi chương trình được chạy.

135. Thủ tục READ và READLN

Cú pháp: Read([Var F : File,] Parameters);

Readln([Var F : File,] Parameters);

Công dụng: Nhập số liệu từ thiết bị chuẩn (thường là bàn phím) vào biến Parameters hay từ tệp được chỉ định trong F. Readln chỉ dùng với tệp văn bản, sau khi nhập số liệu Readln di chuyển con trỏ xuống dòng kế tiếp.

136. Hàm READKEY(CRT Unit)

Cú pháp: Readkey : Char;

Công dụng: Đọc một mã phím vào một biến. Với các phím mã phím gồm hai số, số đầu là 0 thì cần gọi lại hàm Readkey một lần nữa để lấy giá trị thứ hai của mã phím.

137. Thủ tục RECTANGLE

Cú pháp: Rectangle(x1,y1,x2,y2:Integer);

Công dụng: Vẽ hình chữ nhật có góc ở trên bên trái ở toạ độ(x1,y1) và có góc dưới bên phải ở toạ độ(x2,y2).

138. Thủ tục RELEASE

Cú pháp: Release(Var P:Pointer);

Công dụng: Giải toả vùng nhớ heap đã cấp phát kể từ lần ghi (Mark) gần nhất trước đó. P cất giữ địa chỉ của đỉnh heap.

139. Thủ tục RENAME

Cú pháp: Rename(Var F:File; S:String);

Công dụng: Đổi tên tệp F thành S.

140. Thủ tục RESET

Cú pháp: Reset(Var F:File [;I:Integer]);

Công dụng: Mở tệp F để đọc. Nếu tệp đó không kiểu, ta có thể chỉ định kích thước mẫu tin trong I.

141. Thủ tục RESTORECRTMODE(Graph Unit)

Cú pháp: RestoreCRTmode;

Công dụng: thoát khỏi kiểu màn hình đồ hoạ trở về kiểu văn bản.

142. Thủ tục REWRITE

Cú pháp: Rewrite(Var F:File [;I:Integer]);

Công dụng: Chuẩn bị tệp để ghi. Nếu tệp chưa tồn tại Pascal sẽ tạo ra tệp mới với tên tệp ghi trong F . Nếu đã có một tệp trên đĩa thì tệp cũ sẽ bị xoá và một tệp rỗng được thế chỗ. Với tệp không kiểu ta có thể định chiều dài của recor (bản ghi) trong I.

143. Thủ tục RMDIR(DOS Unit)

Cú pháp: RmDir(S:String);

Công dụng: xoá thư mục được có tên chỉ định trong S

144. Hàm ROUND

Cú pháp: Round(x;Real):LongInt;

Công dụng: Làm tròn x.

145. Thủ tục RUNERROR

Cú pháp: RunError [(ErrorCode:word)];

Công dụng: Dừng chương trình, phát sinh lỗi Run-Time với số hiệu lỗi Errorcode.

146. Thủ tục SECTOR(Graph Unit)

Cú pháp: Sector(x,y:Integer;g1,g2,a,b:word);

Công dụng: Vẽ và tô đầy cung elip có tâm (x,y), bán kính ngang là a, bán kính dọc là b, bắt đầu từ góc g1 đến góc g2.

147. Thủ tục SEEK

Cú pháp: Seek(Var F:Flie; P:Integer);

Công dụng: Di chuyển con trỏ tệp đến đầu của bàn ghi thứ P trong tệp tin F.

148. Thủ tục SEEKEOF

Cú pháp: SeekEof(Var F:File):Boolean;

Công dụng: kiểm tra tình trạng kết thúc tệp (chỉ dùng cho tệp văn bản), nếu kết thúc tệp kết quả nhận được là True, ngược lại là False (xem thêm eof).

149. Hàm SEEKEOLN

Cú pháp: SeekEoln(Var F:File):Boolean;

Công dụng: kiểm tra tình trạng kết thúc dòng, SeekEoln tương tự như Eoln chỉ có khác biệt là nó nhảy qua khoảng trống, tab trước khi kiểm tra dấu kết thúc hết dòng.

150. Hàm SEG

Cú pháp: Seg(Var Variable):word;

Seg(<Procedure hoặc Function>):word;

Công dụng: Cho địa chỉ đoạn của biến, hàm hay thủ tục.

151. Thủ tục SETALLPALETTE(Graph Unit)

Cú pháp: SetAllPalette(Var Palette);

Công dụng: Thay đổi bảng mầu theo thiết kế cho trong Palette.

152. Thủ tục SETBKCOLOR(Graph Unit)

Cú pháp: SetBKcolor(Color:word);

Công dụng: chọn màu nền ngầm định cho màn hình đồ hoạ.

153. Thủ tục SETCBREAK(DOS Unit)

Cú pháp: SetCBreak(Break: Boolean);

Công dụng: thiết lập trạng thái cho tổ hợp CTRL – BREAK. Khi Break = True, hệ thống sẽ ngừng chương trình mỗi khi Ctrl-Break được bấm. Nếu Break=False việc kiểm tra chỉ tiến hành khi vào ra (I/O) với bàn phím hoặc máy in.

154. Thủ tục SETCOLOR(Graph Unit)

Cú pháp: Setcolor(color:word);

Công dụng: chọn mầu cho nét vẽ theo số hiệu màu color .

155. Thủ tục SETDATE(DOS Unit)

Cú pháp: Setdate(Year, Month, Day:word);

Công dụng: Cập nhật đồng hồ hệ thống theo ngày tháng năm truyền qua tham biến.

156. Thủ tục SETFILLPATTERN(Graph Unit)

Cú pháp: SetFillPattern(Patter:FillPatterType;color:word);

Công dụng: Thiết lập mẫu tô hình vẽ (do người dử dụng định nghĩa tự chọn)

157. Thủ tục SETFILLSTYLE(Graph Unit)

Cú pháp: SetFillStyle(Pattern:word;color:word);

Công dụng: ấn định mẫu tô và mầu tô.

158. Thủ tục SETGRAPHMODE(Graph Unit)

Cú pháp: SetGraphMode(Mode:Integer);

Công dụng: ấn định kiểu đồ hoạ hiện hành.

159. Thủ tục SETLINESTYPE(Graph Unit)

Cú pháp: SetLineStyte(lineStyle:word; Pattern:word: thickness:word);

Công dụng: ấn định dạng, mẫu và bề dầy của đường vẽ.

160. Thủ tục SETPALETTE(Graph Unit)

Cú pháp: SetPalette(m1:word; m2:shortInt);

Công dụng: đổi mầu thứ m1 trong bảng mầu thành màu m2. (0<=m1<=15)

Setpalette(0,red); đổi màu thứ nhất trong bảng màu thành màu đỏ

161. Thủ tục SETRGBPALETTE(Graph Unit)

Cú pháp: SetRGBPalette(m1,RedValue, Greenvalue,BlueValue:Integer);

Công dụng: thay đổi các thành phần tổ hợp màu, mầu m1 trong bảng mầu được thay bằng tổ hợp mầu: đỏ,xanh,lục. Thủ tục này chỉ dùng với các thiết bị IBM 8514 và VGA

162. Thủ tục SETTEXTJUSTIFY(Graph Unit)

Cú pháp: Settextjustify(Horiz, Vert:word);

Công dụng: Xác định vị trí hiển thị văn bản trong màn hình đồ hoạ.

163. Thủ tục SETTEXTSTYLE(Graph Unit)

Cú pháp: SetTextStyle(Font:word; direction:word; Charsize:word);

Công dụng: Thiết lập Font chữ, chiều hiển thị và kích cỡ chữ.

164. Thủ tục SETTIME(DOS Unit)

Cú pháp: SetTime(H, M, S, S100:word);

Công dụng: đặt lại giờ hệ thống bao gồm giờ (H), phút (M), giây (S), phần trăm giây (S100).

165. Thủ tục SETVIEWPORT(Graph Unit)

Cú pháp: SetviewPort(x1,y1,x2,y2:Integer; Clip: Boolean;)

Công dụng: Chọn hình chữ nhật(x1,y1,x2,y2) làm cửa sổ làm việc. Khi Clip = TRUE cho phép vẽ ra ngoài của cửa sổ.

166. Thủ tục SETWRITEMODE(Graph Unit)

Cú pháp: SetWriteMode(WriteMode;Integer);

Công dụng: Chọn một trong 2 kiểu đường: Copyput(0) hay Xorput (1).

167. Hàm SIN

Cú pháp: Sin(x:Real):Real;

Công dụng: Cho Sin(x)

168. Hàm SIZEOF

Cú pháp: Sizeof(Var Variable):word;

Công dụng: Cho kích thước(số byte) của biến hay kiểu dữ liệu.

169. Thủ tục SOUND(Graph Unit)

Cú pháp: Sound(Freq:word);

Công dụng: Phát âm thanh có tần số Freg.

170. Hàm SQR

Cú pháp: Sqr(x:Real):Real; Sqr(x:Integer):Integer;

Công dụng: Cho bình phương của x.

171. Hàm SQRT

Cú pháp: Sqrt(x:Real):Real;

Công dụng: Cho căn bậc hai của x

172. Thủ tục STR

Cú pháp: Str(I:Integer; [:length;] Var S:String);

Str(x:Real; [:length; [:Decimals;]] Var S:String);

Công dụng: Đổi số nguyên hay thực ra chuỗi và cất trong S; length là độ dài của chuỗi(số ký tự), Decimals là chữ số sau dấu chấm thập phân.

173. Hàm SUCC

Cú pháp: Suuc(S:Scalar):Integer;

Công dụng: Cho giá trị tiếp sau S.

174. Thủ tục TEXTBACKGROUND(CRT Unit)

Cú pháp: TextbackGround(color:Byte);

Công dụng: Thay đổi mầu nền chữ viết bằng mầu chỉ ra trong color.

175. Thủ tục TEXTCOLOR(CRT Unit)

Cú pháp: TextColor(Color: Byte)

Công dụng: Thay đổi mầu ký tự bằng mầu chỉ ra trong color.

176. Hàm TEXTHEIGHT(Graph Unit)

Cú pháp: TextHeight(TextString:String):word;

Công dụng: Cho chiều cao(số pixel) của TextString.

177. Thủ tục TEXTMODE(CRT Unit)

Cú pháp: TextMode(Mode:Word);

Công dụng: Chọn Mode văn bản.

178. Hàm TEXTWIDTH(Graph Unit)

Cú pháp: TextWidth(TextString: String):Word;

Công dụng: Cho chiều rộng(số pixel) của chuỗi TexString.

179. Hàm TRUNC

Cú pháp: Trunc(x:Real):integer;

Công dụng: Cho phần nguyên của x.

180. Thủ tục TRUNCATE

Cú pháp: Truncate(Var F);

Công dụng: Cắt dữ liệu khỏi tệp kể từ vị trí hiện thời của con trỏ.

181. Thủ tục UNPACKTIME(DOS Unit)

Cú pháp: UnPackTime(Time:Longint; Var DT:DateTime);

Công dụng: Giải mã biến Time (là một số kiểu Longint và chứa trong nó các đại lượng ngày, giờ), trả kết quả về DT. Cấu trúc của DateTime là:

```
TYPE
   DateTime = Record
        Year,Month,Day,Hour,Min,Sec:word;
    End;
```

182. Hàm UPCASE

Cú pháp: Upcase(S:Char):Char;

Công dụng: Chuyển các ký tự trong S thành chữ in.

183. Thủ tục VAL

Cú pháp: Val(S:String;Var R:Real;Var Code:Integer);

Val(S:String;Var I:Real;Var Code:Integer);

Công dụng: Đổi chuỗi S thành số (R hoặc I). Nếu đổi chuỗi thành công biến Code nhận giá trị 0. Nếu không đổi được, code chứa số nguyên cho biết vị trí của ký tự trong chuỗi S đã gây lỗi.

184. Hàm WHEREX(CRT Unit)

Cú pháp: WhereX:Byte;

Công dụng: Cho biết cột đang chứa Con trỏ trong cưả sổ hiện hành.

185. Hàm WHEREY(CRT Unit)

Cú pháp: WhereY:Byte;

Công dụng: Cho biết dòng đang chứa Con trot trong cửa sổ hiện hành.

186. Thủ tục WINDOW(CRT Unit)

Cú pháp: Window(x1,y1,x2,y2:Byte);

Công dụng: định nghĩa một cửa sổ góc trái trên (x1,y1) và góc phải dưới là (x2,y2).

187. Thủ tục WRITE và WRITELN

Cú pháp: Write(Var F:File, Parameters); Writeln(Var F:File, Parameters);

Công dụng: Write sẽ ghi giá trị các tham số lên màn hình hoặc lên tệp. Writeln làm việc giống Write nhưng ghi thêm dấu hiệu hết dòng(CR/LF) vào cuối .

Phụ lục 3

Định hướng biên dịch (Compiler Directive)

Định hướng biên dịch trong ngôn ngữ Pascal bao gồm 52 kiểu và được quy định viết như sau:

a. Vị trí

Các định hướng biên dịch được đặt tại những vị trí thích hợp trong chương trình tuỳ thuộc vào các yêu cầu cụ thể của bài toán.

b. Tất cả định hướng biên dịch đều đặt trong cặp ký tự " { " và " } "

Mỗi định hướng đều có gía trị ngầm định (by default) và trình biên dịch sẽ sử dụng giá trị này nếu trong chương trình không có những định hướng biên dịch khác. Để tìm hiểu nội dung của toàn bộ các định hướng biên dịch trong Pascal xin mở Help từ trang màn hình thứ 1 đến trang màn hình thứ 2. Dưới đây là một số định hướng biên dịch thông dụng:

1. Ghép tệp

Cú pháp:{$I filename}

Định hướng cho trình biên dịch gộp tên tệp chỉ định bởi FileName vào lúc dịch. Kết quả là tệp được chèn vào văn bản cần dịch ngay phía sau dẫn hướng này. Phần mở rộng ngầm định cho filename là .PAS. Nếu trong filename không chỉ định thư mục, thì trình biên dịch sẽ tìm kiếm tệp đó trong thư mục hiện hành.

2. Kiểm tra vào ra (Input/Output)

Cú pháp:{$I+} hoặc {$I-}

Ngầm định: {$I-}

Không tự động sinh mã kiểm tra kết quả lệnh khi có lệnh gọi đến các thủ tục vào ra. Khi dùng {$I-}, chương trình sẽ không dừng ngay cả khi có lỗi vào ra (mã lỗi khác 0). Muốn kiểm tra lỗi cần dùng hàm IOResult . Nếu thủ tục vào ra có lỗi và định hướng là {$I+} thì chương trình sẽ chấm dứt và đưa ra thông báo lỗi.

3. Link Object File

Cú pháp: {$L filename}

Yêu cầu trình biên dịch liên kết tệp chỉ định bởi Filename với chương trình hay unit cần dịch. Phần mở rộng ngầm định của filename là .OBJ. Nếu trong filename không chỉ rõ đường dẫn thì trình biên dịch sẽ tìm tệp đó trong thư mục hiện hành.

4. Xử lý số(Numeric Processing)

Cú pháp:{$N+} hoặc {$N-}

Ngầm định: {$N-}

Định hướng việc phát sinh mã dấu phảy động với số thực trong Pascal. Với {$N-} việc sinh mã để thực hiện tính toán dấu phẩy động do các chương trình trong thư viện Run-Time đảm nhận. Với {$N+} sẽ dùng bộ xử lý số 8087 để sinh mã.

5. Overlay code generation.

Cú pháp: {$O+} hoặc {$O-}

Ngầm định: {$O-}

Khi chọn {$O+} sẽ cho phép unit là Overlay. Trình tạo mã các xử lý đặc biệt để truyền tham số cho các thủ tục hay hàm có Overlay.

6. Kiểm tra phạm vi (Range)

Cú pháp: {$R+} hoặc {$R-}

Ngầm định: {$R-} Định hướng này cho không cho phát sinh mã kiểm tra phạm vi. Với {$R+} tất cả biểu thức chỉ số mảng và chuỗi đều được kiểm tra xem có nằm trong phạm vi đã định nghĩa hay không. Tất cả các phép gán vào biến kiểu vô hướng và đoạn con cũng đều được kiểm tra phạm vi. Nếu có lỗi, chương trình kết thúc và hiện thông báo lỗi Run-time. Định hướng này sẽ làm cho chương trình có kích thước lớn và chạy chậm, chỉ nên dùng khi chạy thử chương trình . Sau khi chương trình đã chạy tốt thì nên bỏ.

7. Kiểm tra tràn Stack

Cú pháp: {$S+} hoặc {$S-}

Ngầm định: {$S-}

Định hướng này khống chế việc sinh mã kiểm tra tràn Stack.

Với {$S+} trình biên dịch sinh mã ở phần đầu của từng thủ tục hay hàm để kiểm tra có đủ chiều dài Stack cho các biến cục bộ và vùng nhớ khác. Khi không đủ chiều dài Stack thì lệnh gọi đến thủ tục hay hàm được biên dịch với {$S+} sẽ chấm dứt chương trình và xuất hiện thông báo lỗi Run-time.

Với {$S-} lệnh gọi như thế sẽ đưa đến ngừng hệ thống.

8. {$U-} hoặc {$U+}

Ngầm định {$U-}

Định hướng này không cho phép dùng tổ hợp phím Ctrl - C để ngắt chương trình. Với {$U+} người sử dụng có thể dùng chương trình bất kỳ lúc nào bằng cách bấm tổ hợp Ctrl - C.

Phụ lục 4

Thông báo lỗi

Để tham khảo các thông báo lỗi, từ Text Editor (màn hình soạn thảo chương trình) hãy gõ từ Error rồi bấm Ctrl-F1, chúng ta sẽ truy nhập vào mục Error - Message. Trong mục này Pascal trình bày ba nhóm lỗi:

Compile error messages gồm các lỗi từ số 1 đến số 99 (liên tục)
Compile error messages gồm các lỗi từ số 100 đến số 170 (liên tục)
Run - Time error messages gồm các lỗi từ số 1 đến số 216 (không liên tục)

Các lỗi Run-time và Compile có một số lỗi cùng số hiệu nhưng ý nghĩa khác nhau. Do số lượng lỗi khá nhiều nên chúng tôi bỏ bớt một số lỗi ít gặp.

1. Lỗi biên dịch

Số hiệu lỗi	Thông báo	Giải thích ý nghĩa
1	**Out of memory**	Bộ nhớ không đủ khi biên dịch
2	**Identifler expected**	Một tên được dùng không phù hợp. Có thể bạn đang khai báo lại một danh từ riêng, từ khoá của Turbo Pascal.
3	**Unknown identifier**	Tên (hoặc biến) chưa được khai báo, hoặc không được biết đến ở trong khối hiện thời. Có thể đó là một tên thiếu hoặc nhầm một vài kí tự so với tên đã được khai báo.
4	**Duplicate identifier**	Sử dụng hai tên trùng nhau trong cùng một khối. Có thể là tên chương trình trùng tên biến.
5	**Syntax error**	Lỗi chính tả trong văn bản nguồn. Có thể là do bỏ quên dấu nháy trên 'abcde'.
6	**Error in real constant**	Lỗi khi biểu diễn các số thực, cách viết sai có thể là: .98 (không có chữ số 0 ở đầu).
7	**Error in integer constant**	Lỗi biểu diễn số nguyên
8	**String constant**	Thường là do quên đánh dấu kết thúc hoặc mở đầu một xâu kí tự, hoặc do vị trí dấu phảy và dấu nháy không đúng.
10	**Unexpected end of file**	Kết thúc file không hợp lệ * File nguồn kết thúc trước lệnh **end** của phần lệnh chính, hoặc lệnh **begin** và **end** không đi cặp với nhau. * Phần chú thích đóng mở không đúng
11	**Line too long**	Dòng lệnh dài quá 127 kí tự
12	**Type identifier expected**	Biến (tên) không phù hợp về kiểu mà đáng ra nó phải có.
13	**Too many open files**	Mở quá nhiều file hoặc trong file CONFIG.SYS không có khai báo FILES=n (n là số lượng File sẽ mở) hoặc giá trị n quá nhỏ, hãy tăng giá trị này lên khoảng từ 15 đến 20.

14	**Invalid file name**	Tên file không đúng hoặc chỉ định một đường dẫn không tồn tại.
15	**File not found**	Không tìm thấy file trong thư mục hiện hành hoặc trong các đường dẫn đã chỉ định để tìm kiếm từng loại file.
16	**Disk full**	Đĩa đã đầy, không còn chỗ. Hãy xoá bỏ một vài file, nếu là đĩa mềm hay dùng đĩa khác.
17	**Invalid compiler directive**	Ký tự định hướng dịch không đúng hoặc chưa được biết, một trong các tham số dẫn hướng có thể lỗi.
18	**Too many files**	Quá nhiều file (của chương trình hay unit) được đưa vào dịch. Cần giảm số File đi.
19	**Undefined type in pointer definition**	Một kiểu được xem như là con trỏ trỏ đến kiểu đã được khai báo trước, song nó chưa hề được khai báo.
20	**Variable identifier expecvted**	Biến (tên) không thể hiện một biến mà đáng ra nó phải có.
21	**Error in type**	Lỗi về kiểu dữ liệu
22	**Structure too large**	Cỡ lớn nhất cho phép của một kiểu cấu trúc là 65535 byte.
23	**Set base type out of range**	Kiểu cơ bản của tập hợp phải là một tập con 0..255 hoặc là kiểu liệt kê có không nhiều hơn 256 giá trị.
24	**File components may not be files or object**	Cấu trúc file of file hoặc file of object là không cho phép. Kiểu File và kiểu cấu trúc không thể chứa một kiểu object hoặc kiểu file.
25	**Invalid String leng**	Độ dài của một chuỗi phải trong khoảng 1 - 255
26	**Type mismatch**	Lỗi này xảy ra trong các trường hợp sau: * Giá trị gán cho biến không tương thích về kiểu. * Tham số thực sự và tham số hình thức không tương thích về kiểu * Kiểu của biểu thức không tương thích với kiểu chỉ số khi làm chỉ số mảng. * Không tương thích giữa các kiểu của các toán hạng trong một biểu thức.
27	**Invalid subrange base type**	Kiểu cơ sở của kiểu khoảng con không hợp lệ.
28	**Lower bound greater than upper bound**	Khai báo kiểu miền con cận dưới cao hơn cận trên, hoặc giá trị đầu và cuối trong vòng For không phù hợp
29	**Ordinal type expected**	Kiểu thực, kiểu chuỗi, kiểu cấu trúc, kiểu con trỏ không được cho phép ở đây. Các kiểu này không phải là các kiểu đếm được.
30	**Integer constant expected**	Không viết đúng hằng số nguyên.
31	**Constant expected**	Hằng viết không đúng.
32	**Integer or real constant expected**	Viết không đúng hằng thực hoặc hằng nguyên.

33	**Pointer type identifier expected**	Biến (tên) không thể hiện một kiểu con trỏ mà đáng ra nó có.
34	**Invalid function result type**	Kiểu kết quả của hàm không hợp lệ. Các kiểu kết quả của hàm trả về hợp lệ là các kiểu đơn giản, kiểu chuỗi và kiểu con trỏ.
35	**Label identifier expected**	Tên nhãn không đúng.
36	**BEGIN expected**	Thiếu từ khoá Begin hoặc
37	**END expected**	Thiếu từ khoá End
38	**Integer expression expected**	Biểu thức kiểu nguyên không đúng.
39	**Ordinal expression expected**	Biểu thức phía trước phải là biểu thức đếm được.
40	**Boolean expression expected**	Biểu thức phía trước phải là kiểu logic.
41	**Operand types do not match operator**	Toán tử sử dụng trong các biểu thức không phù hợp với toán hạng, ví dụ A mod 2
42	**Error in expression**	Lỗi trong một thao tác. Ví dụ quên viết một toán tử giữa hai toán hạng.
43	**Legal assignment**	* Không thể gán giá trị cho file và biến không kiểu. * Tên hàm chỉ có thể gán giá trị trong thân của hàm.
44	**File identifier expected**	Biến (tên) không thể hiện là một trong bản ghi tương ứng hoặc trong biến object.
45	**Object file to large**	Pascal không thể liên kết với các file .OBJ lớn hơn 64k.
46	**Undefined external**	Thủ tục hoặc hàm external không đặt định nghĩa PUBLIC trong file object. Có thể bạn đã chỉ ra tất cả các file object trong dẫn hướng biên dich {$L filename}, cần kiểm tra chính tả trong tên thủ tục hay hàm trong file ASM.
47	**Invallid object file record**	File .OBJ chứa một bản ghi không hợp lệ. Cần đảm bảo rằng file này thật sự phải là một file .OBJ.
48	**Code segment too large**	Bộ nhớ dành cho chương trình hay unit là 65520 byte (64KB). Nếu chương trình lớn hãy chuyển vài thủ tục hay hàm vào trong một unit. Nếu bạn đang biên dịch unit hãy chia nhỏ nó thành một hay nhiều unit .
49	**Data segment too large**	Kích thước to nhất của một đoạn dữ liệu chương trình là 65520 byte (64KB), kể cả dữ liệu được khai báo trong các unit được dùng.
50	**Do expected**	Thiếu từ khoá DO hoặc để không đúng chỗ.

51	**Invalid PUBLIC defination**	Định nghĩa PUBLIC không đúng. * Hai hay nhiều chỉ dẫn PUBLIC trong ngôn ngữ assemebly định nghĩa cùng một tên. * File .OBJ định nghĩa ký hiệu PUBLIC không thuộc về đoạn CODE
52	**Invalid EXTRN definition**	Định nghĩa EXTRN không đúng. * Biến (tên) được chuyển tới hướng dẫn EXTRN trong ngôn ngữ asembly. Nhưng nó không được khai báo trong chương trình Pascal hay unit, không có cả phần giao diện của bất cứ unit được dùng nào.
53	**Too many EXTRN definitions**	Turbo Pascal không thể quản lý file .OBJ với nhiều hơn 256 khai báo EXTRN.
54	**OF expected**	Thiếu từ khoá OF hoặc từ OF không xuất hiện nơi nó cần có.
55	**INTERFACE expected**	Thiếu danh từ riêng Interface.
56	**Invalid relocatable reference**	Chỉ dẫn tái định vị bị sai
57	**Then expected**	Thiếu từ THEN hoặc nó được đặt không đúng chỗ
58	**To or Downto expected**	Từ khoá To hoặc Downto đặt không đúng vị trí.
59	**Undefined forward**	* Thủ tục hoặc hàm được khai báo trong phần interface của một unit, nhưng chưa định nghĩa trong phần implementation. * Thủ tục hoặc hàm được khai báo với từ khoá forwad nhưng định nghĩa của nó không có.
61	**Invalid typecast**	Kích thước của biến và kiểu nhận khác nhau, hoặc một biểu thức được đưa vào nơi chỉ cho phép biến có mặt.
62	**Division by zero**	Phép chia cho số 0.
63	**Invalid file type**	Kiểu file không đúng. Ví dụ: readln không thể dùng với file có kiểu hoặc seek với file text.
64	**Cannot Read and Write variables of this type**	Kiểu biến không thể áp dụng trong các lệnh Read hoặc Write
65	**Pointer variable expected**	Chưa khai báo biến kiểu con trỏ.
66	**String variable expected**	Chưa khai báo biến kiểu chuỗi.
67	**String expression expected**	Biểu thức phải thuộc kiểu chuỗi.
68	**Circular unit reference**	Lỗi vì các Unit tham chiếu vòng tròn (a gọi tới b, b gọi tới c, c gọi tới a)
69	**Unit name mismatch**	Tên của unit tìm thấy trong file .TPU không phù hợp với tên chỉ ra trong mệnh đề uses.

70	**Unit version mismatch**	Một hoặc nhiều unit được dùng bởi unit này đã được thay đổi từ khi unit được dịch. Dùng Compliler/Make or Compile/Built trong IDE và lựa chọn/M hoặc /B trong trình biên dịch theo dòng lệnh để tự động dịch lại các unit cần thiết.
71	**Internal stack overflow**	Tràn stack , dung lượng hiện có của Stack không đủ để nạp những gì chương trình yêu cầu
72	**Unit file format error**	File đơn vị chương trình .TPU bị lỗi. Có thể file được tạo nên bởi một version cũ của pascal, cần tạo lại file mới bằng cách dịch lại file nguồn.
73	**IMPLEMENTATION expected**	Từ khoá impementation không xuất hiện nơi cần. Có thể là do gộp phần imlementation của một thủ tục, một hàm hoặc một hành vi vào trong phần interface của unit.
74	**Constan and case types do not match**	Kiểu của biến chọn trong cấu trúc case không tương thích với giá trị của các nhãn
75	**Record or object variable expected**	Biến phải là kiểu dữ liệu bản ghi hoặc kiểu object.
76	**Constant out of range**	Lỗi có thể là: * Đánh chỉ số mảng với một giá trị vượt ra ngoài phạm vi. * Gán một giá trị vượt phạm vi cho một biến. * Cho một hằng vượt phạm vi làm tham số cho một biến hoặc hàm.
77	**File variable expected**	Biến phía trước phải thuộc kiểu file.
78	**Pointer expression expected**	Biểu thức phía trước phải thuộc kiểu con trỏ.
79	**Integer or real expected**	Biểu thức phải là kiểu nguyên hay kiểu thực
80	**Label not within current block**	Nhãn không liên kết với Block hiện tại. Lệnh goto nhảy tới một nhãn nằm ngoài khối hiện tại.
81	**Label already defined**	Nhãn đã được sử dụng
82	**Undefined label in preceding statement part**	Nhãn được khai báo và chỉ dẫn trong phần lệnh trước nhưng chưa được định nghĩa
83	**Invalid @ argument**	Đối số (Argument) hợp lệ cho toán tử @ là một biến, tên thủ tục hoặc hàm.
84	**UNIT expected**	Thiếu từ khoá unit
85	**“;” expected**	Thiếu dấu ;
86	**“:” expected**	Thiếu dấu :
87	**“,” expected**	Thiếu dấu ,
88	**“(” expected**	Thiếu dấu ngoặc đơn mở
89	**“)” expected**	Thiếu dấu ngoặc đơn đóng
90	**“=” expected**	Thiếu dấu bằng
91	**“:=” expected**	Thiếu toán tử gán

92	**"[" or "(." expected**	Thiếu dấu móc trái
93	**"]" or ".)" expected**	Thiếu dấu móc phải
94	**"." Expected**	Thiếu dấu chấm .
95	**".." expected**	Thiếu dấu miền con .
96	**Too many variables**	Quá nhiều biến. (tổng cộng các biến toàn cục khai báo trong chương trình hay unit không được vượt hơn 64k. Tổng cộng các biến địa phương khai báo trong chương trình con hay unit cũng không được vượt hơn 64k).
97	**Invalid FOR control variable**	Biến điều khiển trong câu lệnh FOR phải thuộc kiểu đơn giản đếm được và phải được khai báo trước.
98	**Integer variable expected**	Biến phía trước phải là kiểu nguyên
99	**File types are not allowed here.**	Kiểu file không cho phép ở đây (hằng định kiểu không thể là kiểu file.)
100	**String length mismatch**	Độ dài của một chuỗi kí tự không phù hợp với số thành phần trong mảng ký tự.
101	**Invalid ordering of fields**	Trường của hằng bản ghi hay hằng object phải viết trong trật tự của sự khai báo.
102	**String constant expected**	Chuỗi hằng không xuất hiện ở nơi cần có.
103	**Integer or real variable expected**	Biến phải thuộc kiểu nguyên hay kiểu thực.
104	**Ordinal variable expected**	Biến phía trước phải là kiểu đếm được.
105	**INLINE error**	Toán tử < không cho phép cùng các tham chiếu tái định vị tới biến , các tham chiếu như vậy luôn luôn có kích thước word.
106	**Character expression expected**	Biểu thức phía trước phải là kiểu ký tự.
107	**Too many relocation items**	Kích thước của bảng tái định vị file kiểu .EXE vượt hơn 64K, lỗi này xuất hiện là do chương trình quá to nên chương trình linker của Turbo Pascal không quản lý được. Nó cũng có thể quá to để DOS thi hành.
108	**Overflow in arithmetic operation**	Kết quả của các phép toán số học không nằm trong phạm vi số nguyên dài (-2147483648 .. 21474783647). Hiệu chỉnh phép toán hoặc dùng giá trị kiểu số thực thay thế giá trị kiểu nguyên.
109	**No enclosing FOR, WHILE, or REPEAT statement**	Thủ tục chuẩn Break và continue không thể dùng ngoài lệnh for, while hoặc repeat.
112	**CASE constant out of range**	Trong cấu trúc CASE các hằng là số nguyên phải trong phạm vi -32768..32767.
113	**Error in statement**	Ký hiệu này không thể bắt đầu một câu lệnh.

114	**Cannot call an interrupt procedure**	Bạn không thể trực tiếp gọi một thủ tục ngắt.
117	**Target address not fount**	Lệnh Searess Find Error trong IDE hoặc lựa chọn /F trong phiên bản dịch dòng lệnh không phát hiện được vị trí câu lệnh tương ứng với địa chỉ chỉ ra.
118	**Include files are not allowed here**	Mọi phần lệnh phải hoàn toàn nằm trong một file.
121	**Invalid qualifer**	Lỗi thuộc một trong cáctrường hợp sau: * Đánh chỉ số một biến không phải là phần tử mảng. * Chỉ ra một trường trong một biến không phải là bản ghi. * Lấy địa chỉ trả về của một biến không phải là con trỏ.
122	**Invalid variable reference**	Chương trình tham chiếu tới biến, nhưng biến lại không phải là một vị trí bộ nhớ (vì biến phải được lưu trong những ô nhớ có địa chỉ). Thường là chương trình đang cố gắng thay đổi một tham số **const** hoặc đang gọi một hàm con trỏ nhưng quên lấy địa chỉ trả về của kết quả.
123	**Too many symbols**	Chương trình hoặc unit khai báo vượt quá 64K ký hiệu. Nếu định hướng dịch là {$D+}, hãy thử đặt nó off, nghĩa là {$D-}, (điều này sẽ ngăn trở bạn tìm lỗi run – time trong khối này). Hãy thử chuyển một số khai báo (hằng, biến, kiểu) tới một unit riêng rẽ.
124	**Statement part too large**	Pascal giới hạn dung lượng của phần lệnh khoảng 24K. Gặp lỗi này, hãy cắt các đoạn và đưa vào một hay nhiều thủ tục hơn.
126	**Files must be var parameters**	Kiểu file không thể đưa làm tham trị trong chương trình con, kiểu File chỉ có thể là tham biến nghĩa là phải khai báo với từ khoá **Var**.
127	**Too many conditional symbols**	Không đủ chỗ để khai báo ký hiệu điều kiện thêm nữa. Cố gắng bỏ đi vài ký hiệu hoặc rút ngắn vài tên tượng trưng.
129	**ENDIF directive missing**	Lỗi khi biên dịch các cấu trúc có điều kiện. Số lượng của các từ khoá mở {$Ifxxx} và các từ khoá đóng {$ENDIF}trong file nguồn phải bằng nhau.
130	**Error in initial conditional defines**	Các ký hiệu điều kiện ban đầu chỉ định trong Option/Compiler/Condition Defines (trong môi trường khép kín IDE) hoặc trong dịch hướng biên dịch /D (với trình biên dịch dòng) không hợp lệ.
131	**Header does not match previous definition**	Các khai báo ở phần đầu (chương trình) không phù hợp với khai báo thủ tục hoặc hàm chỉ ra trong phần **interface** hoặc trong phần khai báo **forward**
133	**Cannot evaluate this expression**	Không thể ước lượng (định giá) biểu thức. Có thể đây là một phép tính không hợp lệ. Ví dụ bạn đang cố gắng dùng hàm Sin trong khai báo const.
134	**Expression incorrectly terminated**	Biểu thức kết thúc không hợp lệ. Ví dụ cuối biểu thức phải là toán hạng chứ không phải là toán tử.
137	**Structured variables are not allowed here**	Cấu trúc biến không cho phép thực hiện các thao tác mà chương trình đòi hỏi. Ví dụ, bạn đang cố gắng làm phép nhân 2 bản ghi.

138	**Cannot evaluate without system unit**	Thư viện TURBO.TPL phải bao gồm System Unit để chương trình gỡ rối có thể tính toán các biểu thức.
140	**Invalid floating – point operation**	Một thao tác trên hai gía trị thực đã tạo nên sự tràn hoặc một phép chia cho số 0.
141	**Cannot compile overlays to memory**	Chương trình dùng overlays phải ghi kết quả dịch trên đĩa chứ không thể gửi vào bộ nhớ
142	**Pointer or procedural variable expected**	Hàm chuẩn Assigned yêu cầu đối số là một biến con trỏ hoặc kiểu thủ tục.
143	**Invalid procedure or function reference**	Lỗi dùng tên một thủ tục trong một biểu thức tính toán. (Nếu bạn đang chuẩn bị gán một thủ tục hoặc hàm cho một biến thủ tục, nó phải được dịch trong trạng thái {$F+}và không thể khai báo với inline hay interrupt.)
144	**Cannot overlay this unit**	Bạn đang cố gắng overlay một unit mà nó không được dịch với trạng thái {$0+}.
146	**File access denied**	Không thể mở file hay tạo ra file mới. Lỗi thường xảy ra khi trình biên dịch đang cố gắng viết vào một file chỉ đọc (Read only file).
147	**Object type expected**	Biến (tên) không thể hiện một kiểu object.
148	**Local object types are not allowed**	Kiểu object chỉ có thể định nghĩa ở tầm xa nhất của một chương trình hay unit. Định nghĩa kiểu object trong thủ tục hay hàm là không được phép.
149	**VIRTUAL expected**	Từ dành riêng virtual bị thiếu.
150	**Method identifier expected**	Biến (tên) không thể hiện một hành vi.
151	**Virtual constructors are allowed**	Hành vi constructor phải là tĩnh.
152	**Constructor identifier expected**	Biến (tên) không thể hiện một constructor.
153	**Destructor identifier expedted**	Biến (tên) không thể hiện một destructor.
154	**Fail only allowed within constructors**	Thủ tục chuẩn Fail chỉ được dùng bên trong một constructor.
155	**Invalid combination of opcode and operands**	Mã assembler không chấp nhận liên kết các toán hạng. Các nguyên nhân có thể: * Quá ít hay quá nhiều các toán hạng đối với lệnh assembler này, ví dụ INC AX, BX hay **Mov AX**. * Số lượng các toán hạng là phù hợp, nhưng kiểu hoặc thứ tự không phù hợp với mã; ví dụ **DEC1, MOV AX, CL** hoặc **MOV 1, AX**.

161	**Code generation error**	Phần lệnh phía trước chứa một lệnh **LOOPNE, LOOPE, LOOP,** hoặc **JCXZ** song các lệnh nhảy này không thể với tới nhãn đích.
162	**ASM expected**	Bạn đang cố gắng biên dịch một hàm hoặc thủ tục assembler có sẵn bên trong và chúng chứa một câu lệnh **begin...end** thay vì dùng **asm...end**.

2. Lỗi liên quan đến hệ điều hành DOS (Dos errors)

1	**Invalid function number**	Số hiệu của một hàm (DOS) không tồn tại
2	**File Not Found**	Không tìm thấy file (có thể là do các thủ tục Reset, *Append, Rename, Rewrite, Erase* báo lại khi tên File không hợp lệ hoặc tên gán cho biến file không xác định một file tồn tại.)
3	**Path not found**	Đường dẫn không tìm thấy, lỗi có thể là do: Các thủ tục *Reset, Append, Rename, Rewrite, ChDir, Mkdir RmDir* hoặc *Erase* gặp tên gán cho tên biến file không hợp lệ hoặc đường dẫn tới một thư mục con không tồn tại.
4	**Too many open files**	Do các thủ tục Reset, *Rewrite* hoặc *Append* báo lại nếu chương trình có quá nhiều file được mở. DOS không cho phép mở quá 15 file cho một ứng dụng. Nếu gặp lỗi này khi mở ít hơn 15 file thì có thể là trong file CONGIG.SYS không chứa khai báo FILES=n hoặc giá trị n qúa nhỏ hãy tăng n lên
5	**File access denied**	Từ chối truy cập fille, lỗi có thể xảy ra khi các thủ tục: * làm việc với file nhưng lại bị chỉ định tới tên thư mục * làm việc với các file đã gán thuộc tính chỉ đọc. * Làm việc với các file chưa được mở
6	**Invalid file handle**	Lỗi này được báo lại nếu một thẻ file không hợp lệ được chuyển tới một lời gọi hệ thống DOS, nguyên nhân có thể là do file bị lỗi.
12	**Invalid file access code**	Thủ tục Reset hoặc Append gặp phải một file có kiểu hoặc không có kiểu mà giá trị FileMode không hợp lệ.
15	**Invalid drive number**	Do các thủ tục GetDir hoặc ChDir báo lại nếu mã số ổ đĩa không hợp lệ.
16	**Cannot remove current directory**	Không thể chuyển thư mục đang làm việc (bởi thủ tục RmDir)
17	**Cannot rename across drives**	Không đổi được tên vì hai tên không có cùng đường dẫn
18	**No more files**	Do biến DosError trong unit Dos và WinDos báo lại khi một lời gọi tới FindFirst hoặc FindNext không tìm thấy file thích hợp với tên file và tập các thuộc tính chỉ ra.

3. Lỗi vào ra (Input/Output error)

Lỗi vào ra thường xuất hiện khi biên dịch chương trình, nếu định hướng dịch là {$I+} (ngầm định) thì chương trình sẽ dừng ngay khi gặp lỗi. Nếu ở đầu chương trình chúng ta quy định {$I-} thì chương trình tiếp tục thi hành, và lỗi được thông báo bởi hàm IOResult. Dưới đây là những lỗi hay gặp:

100	**Disk read error**	Lỗi đọc dữ liệu, thường là do thủ tục Read thông báo khi trên một file có kiểu nếu bạn đang cố gắng đọc quá vị trí cuối file.
101	**Disk write error**	Lỗi khi thực hiên các thủ tục Close, Write, Writeln hoặc Flush khi đĩa đã đầy
102	**File not assigned**	File không thể tạo ra, lỗi này xuất hiện khi thực hiện các thủ tục Reset, Rewrite, Append, Rename và Erase nhưng biến file chưa được gán tên qua thủ tục Assign.
103	**File not open**	File chưa được mở khi thực hiện các thủ tục Close, Read, Write, Seek, Eof, FilePos, FileSize, Flush, BlockRead hoặc BlockWrite .
104	**File not open for input**	File (văn bản) chưa được mở để đọc khi thực hiện các thủ tục Read, Readln, Eof, Eoln, SeekEof hoặc SeekEoln .
105	**File not open for output**	File chưa được mở để ghi khi thực hiện các thủ tục Write hoặc Writeln .
106	**Invalid numeric format**	Trị số đọc từ file văn bản không đúng với dạng số hợp lệ Khi thực hiện các thủ tục Read hoặc Readln .

4. Các lỗi liên quan đến phần cứng

Các lỗi liên quan đến phần cứng có thể tham khảo thêm trong các tài liệu về hệ điều hành MS-DOS

150	**Disk is write protected**	Đĩa (mềm) bị khoá chống ghi. Hãy bỏ các lẫy chống ghi trên đĩa ra.
151	**Unknown Unit**	Tên đơn vị chương trình chưa được biết đến
152	**Drive not ready**	ổ đĩa không sẵn sàng. Có thể do chưa đưa đĩa vào, có thể mất nguồn điện nuôi.
153	**Unknown command**	Không có lệnh này. Chắc bị gõ nhầm một vài ký tự.
154	**CRC error in data**	Lỗi vùng dữ liệu
155	**Bad drive request structure length**	
156	**Disk seek error**	Lỗi khi truy nhập đĩa
157	**Unknown media type**	Kiểu thiết bị không rõ
158	**Sector not found**	Cung từ không tìm thấy. Có lẽ bị hỏng do nhiều nguyên nhân: mốc, xước hoặc bị bụi bẩn...
159	**Printer out of paper**	Máy in không có giấy
160	**Device write fault**	Việc ghi vào thiết bị không thực hiện được
161	**Device read fault**	Việc đọc bị lỗi
162	**Hardware failure**	Phần cứng bị hỏng

5. Lỗi trình biên dịch (run-time error)

Khi trình biên dịch gặp các lỗi này chương trình sẽ bị dừng ngay lập tức.

200	**Division by zero**	Chương trình cố gắng chia một số cho 0 trong phép toán **mod** hoặc **div**.
201	**Range check error**	Nếu định hướng biên dịch là {$R+} thì lỗi sẽ được thông báo khi một trong các tình huống sau xảy ra: * Chỉ số của một mảng vượt ra ngoài phạm vi đã khai báo * Gán một giá trị vượt ra ngoài phạm vi cho phép của một biến. * Giá trị gán cho tham số của một hàm hay một biến vượt ra ngoài phạm vi cho phép
202	**Stack overflow error**	Stack bị tràn. Nếu định hướng biên dịch là {$S+} thì lỗi sẽ được thông báo khi không có đủ chỗ trống trên stack để bố trí bộ nhớ cục bộ của các chương trình con. Có thể thay đổi Stack bằng cách dùng định hướng biên dịch $M. * Lỗi này cũng có thể do vòng lặp vô tận gây ra, hoặc do một thủ tục viết trong ngôn ngữ assembler không duy trì tình trạng stack.
203	**Heap overflow error**	Lỗi này do các thủ tục New hoặc GetMem báo lại nếu không còn đủ chỗ trống trong heap để cấp phát một vùng nhớ theo yêu cầu.
204	**Invalid pointer operation**	Lỗi này do các thủ tục Dispose hoặc FreeMem gây ra nếu như con trỏ là Nil hoặc trỏ tới một vị trí nằm ngoài heap.
205	**Floating point overflow**	Phép tính với dấu chấm động tạo ra kết quả là một số quá lớn .
207	**Invalid floating point operation**	Một tham số thực trong các hàm Trunc hoặc Round không thể chuyển đổi thành một số nguyên trong phạm vi số nguyên dài LongInt (-2147483648 - 2147483647). * Tham số của hàm Sqrt mang giá trị âm. * Tham số của hàm Ln là zero hoặc âm. * Sự tràn 8087 đã xảy ra. (xem trong chương 14. “Using the 80x87” trong cuốn Language gude.)
208	**Overlay manager not installed**	Trình quản lý overlay chưa được cài đặt. Lỗi có thể là do không gọi OverInit hoặc lời gọi OverInit không chạy.
209	**Overlay file read error**	Lỗi xảy ra khi trình quản lý overlay cố gắng đọc một overlay từ file overlay.

Mục lục

Tài liệu tham khảo

1. Dương Xuân Thành, Tin học đại cương, Nxb Thống kê 2003
2. Dương Xuân Thành, Giáo trình ngôn ngữ lập trình Pascal, Nxb Thống kê 2004
3. Đỗ xuân Lôi, Cấu trúc dữ liệu và giải thuật, Nxb Khoa học và Kỹ thuật - 1998
4. Nguyễn Tô Thành, Lập trình nâng cao trên ngôn ngữ Pascal, Nxb Đại học quốc gia 2001
5. Larry Nyhoff - Sanford Leeostima, Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu, Nxb Đà nẵng 1998
6. Quách Tuấn Ngọc, Bài tập ngôn ngữ lập trình Pascal, Nxb Thống kê 2001