



BÀI 1: GIỚI THIỆU NGÔN NGỮ PASCAL VÀ CÁC VÍ DỤ ĐƠN GIẢN

I. Xuất xứ ngôn ngữ Pascal:

Pascal là ngôn ngữ lập trình cấp cao do *Niklaus Wirth*, giáo sư điện toán trường đại học kỹ thuật Zurich (*Thụy Sĩ*), đề xuất năm 1970 với tên Pascal để kỷ niệm nhà toán học và triết học nổi tiếng *Blaise Pascal* (người Pháp).

Ngôn ngữ lập trình Pascal có đặc điểm: ngữ pháp, ngữ nghĩa đơn giản và có tính logic; cấu trúc chương trình rõ ràng, dễ hiểu (*thể hiện tư duy lập trình cấu trúc*); dễ sửa chữa, cải tiến.

Trong quá trình phát triển, Pascal đã phát huy được ưu điểm và được dùng để tạo ra nhiều ứng dụng trên nhiều lĩnh vực khác nhau. Các tổ chức và công ty chuyên về máy tính dựa trên Pascal chuẩn đã phát triển thêm và tạo ra các chương trình dịch ngôn ngữ Pascal với nhiều phần bổ sung, giảm thiểu khác nhau. Ví dụ: *TURBO PASCAL* của hãng Borland (Mỹ), *QUICK PASCAL* của hãng Microsoft, *UCSD PASCAL* (*University of California at San Diego*), *ANSI PASCAL* (*American National Standard Institute*), v.v.

So với nhiều sản phẩm Pascal của nhiều tổ chức và công ty khác nhau xuất bản, *TURBO PASCAL* của hãng Borland tỏ ra có nhiều ưu điểm nhất và hiện nay đã trở thành ngôn ngữ lập trình phổ biến nhất trên thế giới sử dụng trong lĩnh vực giảng dạy và lập trình chuyên nghiệp. Chỉ trong vòng vài năm Turbo Pascal được cải tiến qua nhiều phiên bản : *1.0, 2.0, 3.0, 4.0, 5.0, 5.5 (1989), 6.0 (1990), 7.0 (1992)*.

Các tập tin chính của ngôn ngữ Turbo Pascal gồm:

- *Turbo.exe*: chương trình soạn thảo, dịch và liên kết chương trình.
- *Turbo.tpl* (*.tpl* - *Turbo Pascal Library*): tập tin thư viện lưu các đơn vị (*Unit*) chuẩn để chạy với Turbo.exe.

Muốn sử dụng các lệnh đồ họa, phải có các tập tin sau:

- *Graph.tpu*: Đơn vị (*Unit*) chứa các lệnh đồ họa.
- Các tập tin có phần mở rộng *CHR* (*SANS.CHR, TRIP.CHR, GOTH.CHR, v.v.*): Chứa các kiểu chữ trong chế độ đồ họa.
- Các tập tin có phần mở rộng *BGI* (*EGA.VGA.BGI, HERC.BGI, CGA.BGI,...*): để điều khiển các loại màn hình tương ứng khi dùng đồ họa.

II. Khởi động:



— Giáo trình Lập trình Pascal căn bản —

— 2 —

Ta có thể khởi động Pascal từ *Windows* hoặc *MS-DOS*, chuyển đến thư mục *BP* hoặc *TP* và chạy tập tin *BP.EXE* hay *TURBO.EXE*. Hai cách khởi động trên thực hiện như sau:

- *Khởi động từ dấu nhắc của MS-DOS*: Chuyển đến thư mục *BP* hoặc *TP* nơi chứa tập tin *BP.EXE* hoặc *TURBO.EXE*, gõ *BP* hoặc *TURBO* và ấn <Enter>.

- *Khởi động từ Windows*: chọn menu *Start/Program/Borland Pascal*. Nếu chương trình Pascal chưa được cài vào menu *Start*, bạn có thể dùng *Windows Explorer* chuyển đến tập tin *BP.EXE* hoặc *TURBO.EXE* và khởi động Pascal bằng cách chạy tập tin này.

III. Các phím chức năng cần biết của ngôn ngữ Pascal:

- **F2**: Lưu chương trình trong khi soạn thảo.
- **F3**: Tạo một file mới hoặc mở một file cũ.
- **F9**: Dịch thử chương trình để kiểm tra lỗi.
- **Ctrl - F9**: Chạy chương trình.
- **Alt - F5**: Xem kết quả chạy chương trình.
- **Alt - X**: Thoát khỏi màn hình soạn thảo chương trình Pascal.

IV. Cấu trúc một chương trình Pascal:

1. Cấu trúc cơ bản:

Chương trình Pascal đơn giản nhất phải có hai từ khoá *Begin* và *End* như sau:

Begin

End.

Chương trình trên tuy không làm gì khi chạy (ấn *Ctrl - F9*) nhưng là một chương trình hợp lệ do hội đủ điều kiện cần thiết là có hai từ khoá *Begin* và *End*.

Từ khoá *End* có kèm dấu "." phía sau báo hiệu kết thúc chương trình, đây là điều bắt buộc phải có trong một chương trình. Từ khoá *Begin* trên được trình biên dịch hiểu là bắt đầu thực hiện các lệnh sau nó và kết thúc tại từ khoá *End* có dấu chấm ".". Khối lệnh nằm trong cặp từ khoá *Begin* và *End* nếu có dấu chấm theo sau còn gọi là khối chương trình chính. Ngoài ra, nếu sau từ khoá *End* không có dấu hoặc có dấu ";" thì đó có thể là khối chương trình con, khối lệnh của hàm hoặc khối lệnh trong chương trình. Trong chương trình có thể có nhiều khối lệnh, tức có thể có nhiều cặp từ khoá *Begin* và *End*.

2. Phương pháp khai báo và tổ chức cấu trúc một chương trình Pascal:



Việc đặt các phần khai báo và soạn thảo chương trình theo thứ tự như sau:

```
Program ProgName;  
Uses UnitName1, UnitName2, UnitNameN;  
Label LabelName1, LabelName2, LabelNameN;  
Const Const1 = n, Const2 = m, ConstN = k;  
Type Type1 = AnyType;  
Var Var1, Var2, VarN : Type;  
Begin
```

{ Các lệnh của chương trình }

End.

Ồ Giải thích cấu trúc các khai báo trên:

Nếu có phần khai báo nào cần cho chương trình thì phải tuân theo thứ tự trên, ví dụ: phần khai báo thư viện (*USES*) không thể đặt sau phần khai báo hằng số (*CONST*) hoặc sau (*VAR*)... sau mỗi phần khai báo phải có dấu ';'.

- *Program*: Từ khoá này dùng để khai báo tên chương trình, *ProgName* là tên chương trình, tên này khác với tên tập tin. Tên chương trình phải tuân theo quy tắc:

- + không có ký tự trống xen giữa.
- + không đặt số ở ký tự đầu tiên.
- + trong phần tên không chứa các ký tự đặt biệt như: '!', '@', '#', '\$', '%', '^', '&', '*', '(', ')', '-', '+', '/', '|', ':', ';', '.', 'v.v.
- + kết thúc phải có dấu ';'.
- + phần này có thể không có.

4 Ví dụ: một cách khai báo tên chương trình:

```
Program TimUSCLN;  
Begin  
...  
End.
```

- *Uses*: Từ khoá này dùng để khai báo việc sử dụng *Unit* (thư viện) cho chương trình. Thư viện là tập hợp các hàm, thủ tục *do ngôn ngữ Pascal cung cấp kèm theo hoặc cũng có thể do người lập trình tạo ra để sử dụng*. Ta khai báo thư viện thông qua tên của thư viện, và trong chương trình đó ta sẽ có thể sử dụng các thủ tục hoặc



— Giáo trình Lập trình Pascal căn bản —

— 4 —

các hàm có trong thư viện đó. Các thư viện chuẩn của ngôn ngữ Pascal gồm: *CRT*, *DOS*, *GRAPH*, *GRAPH3*, *OVERLAY*, *PRINTER*, *SYSTEM* và *TURBO3*. Trong đó, thư viện *SYSTEM* mặc định được chuyển vào chương trình mà ta không cần phải khai báo. Ví dụ một cách khai báo thư viện:

```
...  
Uses CRT, GRAPH;  
...
```

- *Label*: Dùng để khai báo các nhãn cho chương trình. Nhãn là các tên dùng để đánh dấu trong chương trình để lệnh *GOTO* nhảy đến đúng vị trí đó. Việc sử dụng lệnh *GOTO* được đề cập ở bài 4. Ví dụ một cách khai báo nhãn:

```
...  
Label TH1, N2;  
...
```

- *Const*: Từ khoá này dùng để khai báo các hằng số sử dụng trong chương trình, khi báo hằng số là việc cố định một vài giá trị nào đó trong chương trình thông qua tên hằng, ví dụ cách khai báo hằng:

```
...  
Const k = 5, Max = 500, Ten = 'Nam';  
...
```

- *Type*: từ khoá dùng để khai báo các kiểu hằng dữ liệu sử dụng cho chương trình. Với từ khoá này, ta có thể tự tạo riêng cho mình những kiểu dữ liệu riêng dựa trên các kiểu dữ liệu chuẩn để tiện sử dụng trong việc lập trình. Các khái niệm về dữ liệu chuẩn và phương pháp tạo kiểu dữ liệu tự tạo sẽ được giới thiệu ở các phần sau. Ví dụ một cách để khai báo một kiểu dữ liệu tự tạo:

```
...  
Type Day = Array [1..7] of String[8];  
...
```

- *Var*: Từ khoá dùng để khai báo các biến số được sử dụng trong chương trình. Biến số là các *giá trị có thể thay đổi được* trong suốt quá trình chạy của chương trình. Khái niệm về biến số rất quan trọng trong việc lập trình (*khái niệm này được trình bày kỹ ở bài 3*). Một ví dụ về cách khai báo biến:

```
...
```



```
Var  HoDem, Ten : String;  
      N : Integer;
```

...

Ồ Ghi chú:

- Thứ tự các khai báo trên là điều bắt buộc, ta phải nắm thứ tự này cho dù một số khái niệm ta chưa được biết.
- Trong chương trình Pascal, để tạo lời chú thích, ta sử dụng cặp dấu {...} hoặc (*...*) lồng các câu chú thích vào bên trong nó.
- Trên một dòng có thể viết một hoặc nhiều câu lệnh.

V. Các ví dụ đơn giản làm quen với ngôn ngữ Pascal:

4 Ví dụ 1:

```
Program  GioiThieu;  
Begin  
  Writeln ( ' Trung tam Trung hoc Chuyen nghiep va Day nghe ' );  
  Write  ( '      74 Tran Quoc Toan - Tel: 0511 872664      ' );  
End.
```

F Giải thích chương trình GioiThieu:

- *Begin*: Từ khoá cho biết bắt đầu chương trình.
- *Writeln*: là thủ tục xuất nội dung các thành phần bên trong cặp dấu (...) lên màn hình và chuyển con trỏ xuống dòng. Bên trong cặp dấu (...) có thể có nhiều thành phần gồm *chuỗi ký tự (hằng giá trị chuỗi)*, *biến số* hoặc *hàm*. Giữa các thành phần trong cặp dấu (...) phải cách nhau bằng dấu ',' nếu không cùng loại, tức là chuỗi ký tự phải được cách với biến số hoặc hàm đứng trước nó hay sau nó bằng dấu ','. Chuỗi ký tự muốn hiển thị nguyên văn phải được đặt trong cặp dấu ' '.
- *Write*: là thủ tục xuất nội dung các thành phần bên trong cặp dấu (...) lên màn hình, thủ tục này có chức năng tương tự *Writeln* nhưng không chuyển con trỏ xuống dòng.
- *End*: là từ khoá cho biết kết thúc chương trình.
- Các dòng lệnh nằm giữa *Begin* và *End* là lệnh mà chương trình cần phải thực hiện.
- Để xem chương trình trên, ta chạy bằng *Ctrl - F9* và xem lại bằng *Alt - F5*.



4 Ví dụ 2:

```
Program DonXinPhep;  
Uses CRT;  
Begin  
  ClrScr;  
  Writeln ( ' ***** ' );  
  Writeln ( ' * Cong hoa Xa hoi Chu nghia Viet Nam * ' );  
  Writeln ( ' *      Doc Lap - Tu Do - Hanh Phuc      * ' );  
  Writeln ( ' *      DON XIN PHEP NGHI HOC      * ' );  
  Writeln ( ' ***** ' );  
  Writeln ( '... ' );  
  Readln;  
End.
```

F Giải thích chương trình trên:

- Khai báo: *Uses CRT*; **Đ** khai báo thư viện *CRT*, do có sử dụng lệnh *ClrScr*.
- Lệnh *ClrScr*; **Đ** lau sạch màn hình (*Clear Screen*).
- Các lệnh *Writeln (...)* **Đ** xuất ra màn hình nội dung bên trong dấu (...) và xuống dòng.
- Lệnh *Readln*; **Đ** dừng chương trình, phương pháp này dùng để *hiển thị nội dung sau khi thực hiện các lệnh bên trên và chờ người dùng ấn phím bất kỳ để tiếp tục thực hiện các lệnh kế sau nó*. Trong trường hợp trên, kế tiếp là từ khoá *End* nên chương trình được kết thúc sau khi có một phím bất kỳ được ấn.

4 Ví dụ 3:

```
Program TinhTong;  
Uses CRT;  
Begin  
  ClrScr;  
  Write ( ' 30 + 40 + 15 = ', 30 + 40 + 15 );  
  Readln;  
End.
```

- 1** Kết quả: Máy thực hiện phép tính và hiển thị $30 + 40 + 15 = 85$



— Giáo trình Lập trình Pascal căn bản —

— 7 —

F Trong câu lệnh Write ở trên, có hai thành phần, biểu thức thứ nhất: $30 + 40 + 15$ = ' được hiểu là một chuỗi phải được hiển thị nguyên văn do có cặp dấu ' ' ở hai đầu. Thành phần thứ hai được cách với thành phần thứ nhất bằng dấu ';' và do không có cặp dấu ' ' hai đầu nên nó được tính tổng và trả về giá trị của biểu thức.

_____ o² o _____



BÀI 2 : CÁC KHÁI NIỆM CƠ BẢN CỦA NGÔN NGỮ PASCAL

I. Các từ khoá (*Key word*) trong ngôn ngữ Pascal:

Các từ khoá là các từ dùng để khai báo, đặt tên cho đối tượng trong Pascal, khi ta đặt tên cho đối tượng nào đó, không được đặt trùng tên với các từ khoá.

Bảng từ khoá trong ngôn ngữ Pascal gồm:

and, array, asm, begin, case, const, constructor, destructor, div, do, downto, else, end, file, for, function, goto, if, implementation, in, inline, interface, label, mod, nil, not, object, of, or, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

Turbo Pascal không phân biệt ký tự thường hoặc hoa. Ví dụ, các cách viết sau có ý nghĩa như nhau: *Begin, BEGIN, begin, beGIN, bEGIN,...*

II. Các kiểu dữ liệu cơ bản:

1. Các kiểu dữ liệu dạng số nguyên:

a. Kiểu *Byte*: Kiểu *Byte* thuộc kiểu dữ liệu biểu diễn các giá trị số nguyên từ 0 đến 255. Kiểu *Byte* chiếm 1 byte trên bộ nhớ.

b. Kiểu *Integer*: Kiểu *Integer* là kiểu dữ liệu biểu diễn các giá trị số nguyên từ -32768 đến 32767. Kiểu *Integer* chiếm 2 bytes trên bộ nhớ.

c. Kiểu *Shortint*: Kiểu *Shortint* là kiểu dữ liệu biểu diễn các giá trị số nguyên từ -128 đến 127. Kiểu *Shortint* chiếm 1 byte trên bộ nhớ.

d. Kiểu *Word*: Kiểu *Word* là kiểu dữ liệu biểu diễn các giá trị nguyên từ 0 đến 65535. Kiểu *Word* là kiểu số không biểu diễn được giá trị âm. Kiểu *Word* chiếm 2 bytes trên bộ nhớ.

e. Kiểu *Longint*: Kiểu *Longint* biểu diễn các giá trị số nguyên từ -2.147.483.648 đến 2.147.483.647. Kiểu *Longint* chiếm 4 bytes trên bộ nhớ.

2. Các kiểu dữ liệu dạng số có phần biểu diễn thập phân:

a. Kiểu *Single*: Là tập hợp các số theo kiểu dấu '.' động trong giới hạn từ 1.5E -45 đến 3.4E38 ($1,5 \times 10^{-45}$ đến $3,4 \times 10^{38}$). Kiểu *Single* chiếm 4 bytes trên bộ nhớ.

b. Kiểu *Real*: Là tập hợp các số theo kiểu dấu '.' động trong giới hạn từ 2.9E -39 đến 1.7E 38 ($2,9 \times 10^{-39}$ đến $1,7 \times 10^{38}$). Kiểu *Real* chiếm 6 bytes trên bộ nhớ.



c. Kiểu *Double*: Là tập hợp các số theo kiểu dấu ', ' động trong giới hạn từ $5.0E^{-324}$ đến $1.7E^{308}$ ($5,0 \times 10^{-324}$ đến $1,7 \times 10^{308}$). Kiểu *Double* chiếm 8 bytes trên bộ nhớ.

3. Kiểu Char (ký tự):

Kiểu *Char* dùng để biểu diễn các giá trị là các ký tự thuộc bảng chữ cái: 'A', 'b', 'x',... các con số: 0..9 hoặc các ký tự đặc biệt: '!', '@', '#', '\$', '%', '&', '*',...

Để biểu diễn thông tin, ta cần phải sắp xếp các ký tự theo một chuẩn nào đó và mỗi cách sắp xếp đó gọi là bảng mã, thông dụng nhất là bảng mã *ASCII* (*American Standard Code for Information Interchange*). Bảng mã *ASCII* có 256 ký tự được gán mã số từ 0..255, mỗi ký tự có một mã số nhất định, ví dụ: ký tự 'A' có mã số là 65, 'a' có mã số là 97 trong bảng mã *ASCII*, v.v.

Để hiển thị bảng mã *ASCII*, bạn chạy chương trình sau:

```
Program ASCII_Table;  
Uses CRT;  
Var I : Integer;  
Begin  
  ClrScr;  
  For I := 0 to 255 do  
    Write( I, ' = ', CHR( I ), '  ' );  
  Readln;  
End.
```

4. Kiểu Logic:

Kiểu logic là kiểu biểu diễn hai trạng thái là đúng (*True*) hoặc sai (*False*). Từ khoá để khai báo cho kiểu logic là *BOOLEAN*.

4 Ví dụ:

```
Var Co : Boolean;  
Co := True;
```

5. Kiểu String (chuỗi ký tự):

String là kiểu dữ liệu chứa các giá trị là nhóm các ký tự hoặc chỉ một ký tự, kể cả chuỗi rỗng. Độ dài tối đa của một biến kiểu *String* là 255, tức là nó có thể chứa tối đa một dãy gồm 255 ký tự.

Cú pháp khai báo: (1) *Var Biến_1, Biến_2, Biến_n: String;*



Hoặc (2) *Var Biến_1, Biến_2, Biến_n: String [30];*

Cách khai báo (1) sẽ cho phép biến *HoTen* nhận tối đa 255 ký tự. Cách (2) cho phép biến *HoTen* nhận tối đa 30 ký tự.

Ồ Ghi chú: Cách sử dụng kiểu dữ liệu *String* sẽ được trình bày chi tiết ở bài 8.

III. Các hàm xử lý dữ liệu cơ bản của ngôn ngữ Pascal:

- *SQR(x)* bình phương của một số nguyên hay thực.
- *ABS(x)* trị tuyệt đối của x.
- *SQRT(x)* căn bậc hai của x.
- *SIN(x)* tính giá trị Sin(x) với x là Radian.
- *COS(x)* tính giá trị Cos(x) với x là Radian.
- *ARCTAN(x)* tính giá trị Arctan(x).
- *LN(x)* hàm logaric cơ số e = 2.718.
- *EXP(x)* hàm e^x .
- *TRUNC(x)* cắt bỏ phần thập phân của x nếu có. Ví dụ: *Trunc(4.86) = 4*, *Trunc(-3.2) = -4*.
- *ROUND(x)* cho số nguyên gần x nhất. Ví dụ: *Round(1.6) = 2*, *Round(-23.68) = -24*, *Round(1.5) = 2*.
- *PRED(x)* cho giá trị đứng trước x, đối số x có thể là kiểu logic, kiểu nguyên hoặc kiểu ký tự. Ví dụ: *Pred('B')*; Ồ cho giá trị 'A', *Pred(2)* cho giá trị 1, *Pred(True)* cho giá trị False. Tuy nhiên, *Pred(False)* lại không cho được giá trị nào do giá trị False đứng trước giá trị True đối với kiểu Boolean.
- *SUCC(x)* cho giá trị đứng sau x, đối số x có thể là kiểu logic, kiểu nguyên hoặc kiểu ký tự. Ví dụ: *Succ('B')*; Ồ cho giá trị 'C', *Succ(2)* cho giá trị 3, *Succ(False)* cho giá trị True.
- *ORD(x)* cho số thứ tự của ký tự x trong bảng mã ASCII. Ví dụ: *Ord('A') = 65*, *Ord('a') = 97*,...
- *CHR(x)* trả về ký tự thứ x trong bảng mã ASCII. Ví dụ: *Chr(65) = 'A'*, *Chr(50) = 2*,...



- **ODD(x)** Trả về giá trị *True* nếu x là số lẻ và trả về giá trị *False* nếu x là số chẵn.

IV. Sử dụng hàm **Random(n)** để lấy một giá trị nguyên ngẫu nhiên:

Hàm *Random(n)* sẽ trả về một giá trị nguyên mà máy lấy ngẫu nhiên có giá trị từ 0 đến n. Trong đó, *n* là một số kiểu *Word* tức là trong khoảng từ 0.. 65535.

Trước khi sử dụng hàm *Random* ta phải gọi thủ tục *Randomize* để khởi tạo bộ tạo số ngẫu nhiên

BÀI 3: HẰNG SỐ, BIẾN SỐ, BIỂU THỨC VÀ CÂU LỆNH ĐƠN GIẢN TRONG NGÔN NGỮ PASCAL

I. Hằng số:

1. Khái niệm:

- Hằng số là các giá trị không thay đổi trong quá trình chạy chương trình.
- Có hai phương pháp sử dụng hằng :
 - + Gán trực tiếp giá trị hằng. Ví dụ: *DT := R * R * 3.14; ChuVi := D * 3.14;*
 - + Đặt cho hằng một tên gọi và trong quá trình soạn chương trình ta dùng tên gọi thay cho việc dùng trực tiếp giá trị đó. Ví dụ: *ChuVi := D * Pi;* trong đó, *Pi* là một hằng số chuẩn của Pascal (tức là ta có thể dùng mà không cần khai báo và gán giá trị).
- Hằng số luôn luôn được khai báo trước phần khai báo biến nếu sử dụng theo phương pháp đặt tên cho hằng.

2. Cú pháp khai báo:

Const a₁ = Trị_số_1, a₂ = Trị_số_2, a_n = Trị_số_n;

Trong đó: *a₁... a_n* là tên các hằng số, các *trị_số_1, 2, ..., n* là các giá trị gán cho các tên hằng *a₁...a_n*.

F Ví dụ một cách khai báo hằng số: *Const Pi = 3.1416, Max = 500;*

4 Ví dụ: chương trình tính chu vi đường tròn có sử dụng hằng số *Pi* do ta định nghĩa:

```
Program TinhCV_DT_HT;
Const Pi = 3.1416;
Var R :Real;
Begin
```



```
Write ( ' Nhập bán kính hình tron : ' );  
Readln (R);  
Writeln ( ' Diện tích hình tron = ', Pi * R * R );  
Writeln ( ' Chu vi hình tron = ', 2 * R * Pi);  
Readln;  
End.
```

Ồ Ghi chú:

- Ta tránh viết: $z := \text{Exp}(1.23) + \text{Sin}(2.34) * \text{Sin}(2.34);$
- Ta sẽ thấy tai hại ngay vì khi muốn tính lại z với giá trị mới của x , ví dụ $x = 1.55$, không lẽ lại đi thay hết 3 vị trí với 2.34 (là giá trị cụ thể của x mà ta đã không sử dụng hằng số) thành 1.55 !!
- Trong chương trình trên, bạn có thể *tối ưu hoá* thêm để chương trình chạy nhanh hơn bằng cách thay hai lần tính $\text{Sin}(x)$ bằng một lần. Cụ thể, ta thực hiện như sau:

```
t := Sin(x);  
z := Exp(a + t * t - x);
```

Tác phong *tối ưu hoá* này sẽ rất có ích cho bạn khi bạn có một chương trình với khối lượng tính toán đồ sộ, có thể chạy vài ngày đêm liên tục nhưng nếu biết tối ưu ngay từ đầu thì sẽ giảm bớt xuống còn một ngày chẳng hạn. Lúc này bạn mới 'thấu hiểu' *tối ưu hoá* để làm gì ?

II. Biến số:

1. Khái niệm:

- Là đại lượng mà *giá trị của nó có thể thay đổi trong quá trình thực hiện chương trình*. Biến được khai báo bằng từ khoá *VAR*.
- Biến là tên của một vùng bộ nhớ lưu trữ dữ liệu.
- Biến được truy xuất trong chương trình thông qua tên biến.
- Biến là một cấu trúc ghi nhớ dữ liệu vì vậy phải được quy định theo một kiểu dữ liệu nào đó, ví dụ kiểu *Integer, Byte, Char,...*

2. Cú pháp khai báo cho các biến:

VAR Tên_biến_1, Tên_biến_2, Tên_biến_n : Kiểu_dữ_liệu_của_biến;

Trong đó: *Tên_biến_1, Tên_biến_2, Tên_biến_n* là *tên* các biến cần khai báo để sử dụng trong chương trình, *Kiểu_dữ_liệu_của_biến* là một trong các kiểu dữ liệu



chuẩn (đã được nêu trong phần II của bài 2) của Pascal hoặc do người dùng định nghĩa.

F Ví dụ một cách khai báo biến:

```
Var a,b : Integer;  
    c   : Real;  
    Ten: String [10];
```

4 Ví dụ: chương trình tính tổng hai số nguyên được nhập từ bàn phím. Trong bài này, ta cần khai báo hai biến *a* và *b* để tính toán.

```
Uses CRT;  
Var a, b : Integer;  
Begin  
    ClrScr;  
    Write( ' Nhập số thứ nhất : ' );  
    Readln(a);  
    Write( ' Nhập số thứ hai : ' );  
    Readln(b);  
    Write( ' Ket qua : ', a, ' + ', b, ' = ', a + b );  
    Readln;  
End.
```

III. Biểu thức:

Một biểu thức được tạo bởi các *toán tử (phép toán)* và các *toán hạng* dùng để thể hiện một công thức toán học. *Toán hạng* có thể là *hằng*, *hàm* hoặc *biến*.

4 Ví dụ: Sau khi khai có báo:

```
Const Max = 120;  
Var x: Integer;
```

ta có thể viết biểu thức sau: $5 + Max * Exp(x)$;

Trong đó: $+$ và $*$ là hai toán tử, các hằng số 5, *Max* và hàm *Exp(x)* là các toán hạng.

Ö Chú ý:

- Một hằng, một biến, một hàm cũng được xem là biểu thức, đó là biểu thức đơn giản.
- Các phép toán trong một biểu thức được sắp xếp theo thứ tự ưu tiên như sau:



- + Các phép toán một ngôi được ưu tiên thứ nhất là: dấu dương (+), dấu âm (-), phép phủ định (*not*).
- + Các phép toán nhân chia: *nhân* (*), *chia* (/), *lấy phần nguyên* (div), *lấy phần dư* (mod), *phép và* (and).
- + Các phép cộng trừ: cộng (+), trừ (-), phép hoặc (or).
- + Các phép so sánh: <, <=, >, >=, =, <>.
- Biểu thức trong cặp dấu ngoặc () được thực hiện trước tiên nếu có.
- Các toán tử cùng thứ tự ưu tiên thì được thực hiện từ trái qua phải.

4 Ví dụ việc sử dụng các toán tử và toán hạng:

$$3 + 5 * 3 = 18$$

$$(3 + 5) * 3 = 24$$

$$5 / 2 * 3 = 7.5$$

$$(5 + 2 > 4) \text{ and } \text{not} (\text{true or } (5 - 3 = 8)) = \text{false}$$

$$(-b + \text{sqrt}(d)) / 2 * a \text{ (có nghĩa: } \frac{-b + \sqrt{d}}{2} a \text{)}$$

IV. Câu lệnh đơn giản:

Sau phần khai báo dữ liệu là phần lệnh của chương trình. Phần này xác định các công việc mà chương trình phải thực hiện xử lý các dữ liệu đã được khai báo. Câu lệnh được chia thành hai loại:

- Câu lệnh đơn giản:
 - + Lệnh gán (:=)
 - + Lệnh Nhập - Xuất (*READ*, *READLN*, *WRITE*, *WRITELN*).
 - + Gọi thủ tục.
 - + Lệnh nhảy (*GOTO*).
- Câu lệnh có cấu trúc:
 - + Lệnh ghép (*BEGIN... END*)
 - + Lệnh lựa chọn (*IF... ELSE*, *CASE... OF*)
 - + Lệnh lặp (*FOR*, *REPEAT... UNTIL*, *WHILE... DO*)
 - + Lệnh *WITH*.

Ồ Ghi chú: Nội dung bài này chỉ đề cập đến các lệnh đơn giản. Các lệnh có cấu trúc được trình bày ở bài 4.

1. Lệnh gán:



Lệnh gán dùng để gán giá trị của một biểu thức (có thể là hàm, biến hoặc giá trị) cho một biến.

Cú pháp:

Biến := *biểu_thức*;

F Đầu tiên, máy tính giá trị của biểu thức ở vế phải, sau đó, giá trị tính được từ vế phải được gán cho vế trái (biến).

Ồ Chú ý:

- Vế trái của lệnh gán chỉ có thể là biến. Ví dụ: viết $x + y = 7$; là sai vì vế trái của câu lệnh này là một biểu thức chứ không phải là một biến.

- Kiểu giá trị của biểu thức (hàm, biến hoặc giá trị) ở vế phải phải trùng với kiểu của biến đã được khai báo, trừ một số trường hợp như biến kiểu thực (Single, Real, Double) có thể nhận giá trị kiểu nguyên (Shortint, Byte, Integer, Word, Longint),... do tập hợp số nguyên là tập con của số thực.

4 Ví dụ: Sau khi đã có khai báo:

```
Var      c1, c2 : Char;  
        i, j      : Integer;  
        x, y      : Real;
```

thì ta có thể thực hiện các phép gán sau:

```
c1 := 'A';  
c2 := Chr(97);  
i  := (23 + 6) * 2 mod 3;  
j  := Round(20 / 3);  
x  := i;  
y  := j;
```

2. Lệnh Xuất:

Lệnh xuất dùng để in lên màn hình các dữ liệu, kết quả hay các thông báo.

Cú pháp (1). *WRITE*(*Biểu_thức_1*, *Biểu_thức_2*,..., *Biểu_thức_n*);
(2). *WRITELN*(*Biểu_thức_1*, *Biểu_thức_2*,..., *Biểu_thức_n*);
(3). *WRITELN*;



Dạng (1): In lên màn hình giá trị các biểu thức tại vị trí hiện hành của con trỏ theo thứ tự viết trong lệnh. Sau khi thực hiện xong lệnh *WRITE(...)*; con trỏ định vị tại sau giá trị *biểu_thức_n* của câu lệnh.

Dạng (2): In lên màn hình giá trị các biểu thức tại vị trí hiện hành của con trỏ theo thứ tự viết trong lệnh. Sau khi thực hiện xong lệnh *WRITELN(...)*; con trỏ định vị tại đầu dòng kế tiếp.

Dạng (3): Dùng để chuyển con trỏ xuống dòng.

4 Ví dụ:

```
Var a, b : Byte;
Begin
  A := 2;
  B := 4;
  Write ( 'Day la ket qua phap nhan A voi B: ', a * b);
  Writeln;
  Writeln( '          * * * *          ');
  Write ( '-----');
End.
```

1 Kết quả sau khi chạy chương trình trên:

```
Day la ket qua phap nhan A voi B: 8
          * * * *
-----
```

Ồ Chú ý: Có hai dạng viết trong thủ tục *Write* và *Writeln* là *viết không quy cách* và *viết có quy cách*. Điều này ta xét qua từng kiểu dữ liệu.

(1). Ví dụ về các dạng viết không có quy cách:

```
Uses CRT;
Var
  I : Integer; R : Real;
  Ch : Char;
  B : Boolean;
Begin
  I := 123; R := 123.456; Ch := 'A'; B := 2<5;
  Writeln( I );           {1}
```




```
Writeln( R);           {2}
Writeln( 3.14 );       {3}
Writeln( 20 * 2.5);    {4}
Writeln;
Writeln( Ch );         {5}
Writeln( B );          {6}
Writeln( #7 );         {7}
```

End.

F *Cách viết không quy cách sẽ canh nội dung theo lề bên trái.*

- Số nguyên được viết ra với số chỗ đúng bằng số chữ số gán vào, kể từ vị trí bên trái. Lệnh {1} in ra: 123

- Số thực được viết ra với trình tự sau: *một dấu cách*, tiếp đến là *một số phần nguyên, dấu chấm, 10 vị trí số thập phân*, tiếp đến là chữ *E*, *dấu của phần mũ (+, -)*, *hai số biểu diễn giá trị phần mũ*:

+ Lệnh {2} in ra: 1.2345600000E+02

+ Lệnh {3} in ra: 3.1400000000E+00

+ Lệnh {4} in ra: 5.0000000000E+01

- Kiểu ký tự in bình thường, một ký tự chiếm một chỗ. Lệnh {5} in ra: A

- Kiểu *Boolean* in ra một trong hai từ *True* hoặc *False*. Lệnh {6} in ra: *True*

- Lệnh {7}: *phát ra một tiếng Beep ở loa.*

(2). Ví dụ về các dạng viết có quy cách:

Var

I : Integer;

R, Z : Real;

Ch : Char;

B : Boolean;

Begin

I := 123; R := 123.456; Ch := 'A'; B := 2<5; Z := 543621.342;

Writeln(I :8); {1}

Writeln(-23564:8); {2}

Writeln(R:12:6); {3}



```
Writeln( 35.123456789:12:6 );      {4}
Writeln( R:12 );                   {5}
Writeln( Ch:5);                    {6}
Writeln('ABC':5);                  {7}
Writeln( B:7 );                    {8}
Writeln( Z:1:2 );                  {9}
```

End.

F Cách viết có quy cách sẽ canh nội dung theo lề bên phải, nếu thừa chỗ thì phần lề bên trái được để trắng.

- Lệnh {1} và {2} dành 8 ký tự trên màn hình để in các số nguyên.
- Lệnh {3} và {4} dành 12 ký tự trên màn hình để in các số thực với 6 số lẻ phần thập phân, kết quả in ra: *123.456000* và *35.123457* (do phần thập phân >6 chỗ nên được làm tròn số).
- Lệnh {5} in giá trị của *R* với 12 chỗ dạng mũ số: *1.23456E+02*
- Lệnh {6},{7} dành 5 chỗ để in chữ *A* và xâu ký tự *ABC*.
- Lệnh {8} dành 7 ký tự để in giá trị *True*.
- Lệnh {9} in số thực *Z* như sau: *Writeln(Z : m : n)*. Nếu $m < n$ thì số thực *Z* được in với *n* số lẻ, còn số chỗ trên màn hình thì tùy vào độ dài của số *Z*. Trong trường hợp $m > n$ và độ dài của số lớn hơn *m* thì số được tự động canh phải. Trường hợp $m > n$ và độ dài của số nhỏ hơn *m* thì số được canh phải dư bao nhiêu ký tự máy để trống bên trái.

Ồ Trường hợp trong câu cần hiển thị dấu ' thì ta phải viết hai dấu ' liên nhau (").

4 Ví dụ: *Write(' Don"t forget me ! ');*

1 Kết quả: Trên màn hình hiển thị:

Don't forget me !

Ồ Ghi chú: Muốn in dữ liệu ra máy in ta dùng lệnh *Write* hoặc *Writeln* với tham số *LST* vào trước. Biến *LST* được khai báo trong *Unit Printer*, vì vậy, để sử dụng lệnh in ta cần phải khai báo thư viện *Printer* trong chương trình.

4 Ví dụ:

Uses Printer;



Begin

Writeln(Lst, ' Welcome to Turbo Pascal Language ! ');

End.

1 Kết quả: Khi chạy máy in ra giấy câu *Welcome to Turbo Pascal Language !*

3. Lệnh Nhập:

Lệnh nhập dùng để đưa dữ liệu từ bàn phím vào các biến.

Cú pháp:

(1) *Readln(Biến_1, biến_2, biến_n);*

(2) *Read(Biến_1, biến_2, biến_n);*

Khi thực hiện lệnh này, máy dừng lại chờ người dùng nhập vào đủ n lần nhập dữ liệu tương ứng với n biến.

Ngoài ra, ta có thể sử dụng thủ tục *Readln* để dừng chương trình và chờ người dùng ấn một phím bất kỳ để tiếp tục, ký tự được ấn không hiển thị lên màn hình.

Ồ Chú ý:

- Các biến trong thủ tục *Readln* phải thuộc kiểu *nguyên*, *thực*, *ký tự* hoặc *xâu ký tự*. Do đó, ta không thể nạp từ bàn phím giá trị *True* hoặc *False* các biến kiểu *Boolean*.

- Dữ liệu nhập vào phải tương ứng với kiểu đã khai báo. Phải ấn phím *Enter* để thực hiện lệnh nhập sau khi gõ xong giá trị cần nhập.

4 Ví dụ 1: Với a, b là hai biến nguyên, x là biến thực. Xét đoạn chương trình sau:

Readln(a, b);

Readln(x);

Nếu ta gõ các phím: *2 24 6.5 14 <Enter>*

1 Kết quả: a nhận giá trị *2*, b nhận giá trị *24*. Các ký tự còn lại bị bỏ qua và không được xét trong thủ tục *Readln(x)* tiếp theo. Như vậy, máy dừng lại ở câu lệnh *Readln(x)* để chờ nhập số liệu cho biến x .

4 Ví dụ 2: Giả sử ta đã khai báo: *Var s1, s2, s3 : String[5];*

Xét câu lệnh: *Readln(s1, s2, s3);*

Nếu ta không nhập ký tự mà chỉ ấn *<Enter>* thì cả 3 biến $s1, s2, s3$ đều là xâu rỗng.



Nếu ta gõ *ABCDE1234567* và ấn phím *< Enter >* thì: *s1 = 'ABCDE', s2 = '12345', s3 = '67'*.

4 Ví dụ 3: Viết chương trình tính diện tích *S* của hình thang với đáy dài *a*, đáy ngắn *b*, chiều cao *h*, tất cả được nhập từ bàn phím.

```
Program DienTichHinhThang;  
Uses CRT;  
Var a, b, h, s : Real;  
Begin  
  ClrScr;  
  Write( 'Nhap gia tri cua a, b, h : ' );  
  Readln(a, b, h);  
  S := (a + b) * h / 2;  
  Write( 'Dien tich S = ', S:1:5);  
  Readln;  
End.
```

1 Kết quả khi chạy chương trình:

Nhap gia tri cua a, b, h : 5 3 4 < Enter >

Dien tich S = 16.00000

Ồ Chú ý: Với cách lấy 3 giá trị bằng một lệnh *Readln(a, b, c);* thì các giá trị ta cần nhập cho mỗi biến phải cách với các giá trị khác ít nhất một ký tự trắng. Ta có thể nhập *a, b, c* bằng 3 lệnh *Readln(a); Readln(b); Readln(c);*

_____ o² o _____

BÀI 4: CÁC LỆNH CÓ CẤU TRÚC TRONG NGÔN NGỮ PASCAL

I. Lệnh ghép:

Lệnh ghép là một nhóm các câu lệnh được đặt giữa hai từ khóa *BEGIN* và *END*. Lệnh ghép được thực hiện bằng cách thực hiện tuần tự các câu lệnh nằm giữa *BEGIN* và *END*.

Cú pháp:

Begin

<câu lệnh 1>;

<câu lệnh 2>;



...
<câu lệnh n>;

End;

Sau <câu lệnh n> có thể có dấu ';' hoặc không. *Lệnh ghép cũng là một dạng câu lệnh.*

4 Ví dụ:

Begin

temp := x;

x := y;

y := temp;

End;

Ö Chú ý: Sau từ khóa *END* có thể có dấu ';' hay không tùy thuộc vào các lệnh cấu trúc kế tiếp ta được học.

II. Lệnh lựa chọn:

1. Lệnh IF:

Cú pháp:

IF <biểu thức logic> THEN

<lệnh 1>

ELSE

<lệnh 2>;

Lệnh IF có thể không có phần *ELSE* <lệnh 2>.

F Giải thích lệnh: Khi gặp lệnh này máy kiểm tra <biểu thức logic>, nếu biểu thức này có giá trị *TRUE* (tức là đúng như điều kiện đặt ra) thì máy thực hiện <lệnh 1> nếu ngược lại, tức <biểu thức logic> có giá trị *FALSE* thì <lệnh 2> được thực hiện. Trường hợp trong câu lệnh không có phần *ELSE* và <biểu thức logic> có giá trị *FALSE* thì <lệnh 1> không được thực hiện và máy chuyển đến câu lệnh kế sau lệnh IF đó.

Ö Chú ý: câu lệnh trước từ khóa *ELSE* không được có dấu ';'. Trường hợp có câu lệnh ghép được đặt kế trước *ELSE* thì từ khóa *END* trước *ELSE* không được đặt dấu ';'.



4 Ví dụ 1: Chương trình nhập từ bàn phím 2 số nguyên a , b . Kiểm tra và cho biết số nào lớn hơn.

```
Var a, b : Integer;
Begin
  Write( ' Nhập so a: ' );
  Readln(a);
  Write( ' Nhập so b: ' );
  Readln(b);
  If a > b then
    Write( ' So lon hon la ', a) { tại vị trí này không được đặt dấu; }
  Else
    Write( ' So lon hon la ', b);
  Readln; { có thể không có dấu; tại câu lệnh cuối này }
End.
```

4 Ví dụ 2: Viết chương trình kiểm tra trong ba số a , b , c được nhập từ bàn phím, số nào là lớn nhất.

```
Var a, b, c, max : Integer;
Begin
  Write( ' Nhập so a: ' );
  Readln(a);
  Write( ' Nhập so b: ' );
  Readln(b);
  Write( ' Nhập so c: ' );
  Readln(c);
  Max := a;
  If max < b then
    Max := b;
  If max < c then
    Max := c;
  Write( ' So lon hon la ', max);
  Readln;
End.
```

4 Ví dụ 3: Viết chương trình kiểm tra ba số được nhập từ bàn phím có thể là độ dài của ba cạnh trong một tam giác hay không? Nếu đúng là ba cạnh của tam giác thì



tính chu vi và diện tích tam giác, xét tam giác có phải là tam giác *đều*, *cân* hay không.

```
Var  a, b, c, p, s : Real;
Begin
  Write( ' Nhập ba số a, b, c : ' );
  Readln(a, b, c);
  If (a>0) and (b>0) and (c>0) and (a+b>c) and (a+c>b) and (b+c>a) then
    Begin
      Writeln( ' Ba cạnh trên tạo thành một tam giác. ' );
      If (a=b) and (b=c) then write( ' Đây là tam giác đều. ' );
      If (a=b) or (a=c) or (b=c) then write( ' Đây là tam giác cân. ' );
      p := (a + b + c) / 2;
      s := SQRT(p * (p - a) * (p - b) * (p - c));
      Writeln( ' Chu vi: ', 2 * p:0:5, ' . Diện tích: ', s:0:5);
    End
  Else
    Write( 'Ba số này không tạo thành được một tam giác.' );
  Readln;
End.
```

2. Lệnh CASE:

Câu lệnh *IF* ở trên chỉ rẽ vào một trong hai nhánh tương ứng với giá trị của biểu thức logic. Còn lệnh *CASE* (*rẽ nhánh theo giá trị*) cho phép lựa chọn để thực hiện một trong nhiều công việc tùy theo giá trị của biểu thức.

Cú pháp:

```
CASE <biểu thức> OF
  Tập_hằng_1: <lệnh_1>;
  Tập_hằng_2: <lệnh_2>;
  .....
  Tập_hằng_n: <lệnh_n>;
ELSE
  <lệnh_n+1>;
END;
```

Lệnh *CASE* có thể không có phần *ELSE* <lệnh_n+1>;



F *Giải thích lệnh:*

1. Tập_hằng_{*i*} ($i = 1, \dots, n$) có thể bao gồm các hằng và các đoạn hằng, ví dụ:

3 : <lệnh 1>;

5, 10.. 15 : <lệnh 2>;

'A', Chr(152) : <lệnh 3>;

'0'..'9' : <lệnh 4>;

2. Giá trị của <biểu thức> và giá trị trong các Tập_hằng_{*i*} phải có cùng kiểu và phải là kiểu vô hướng đếm được (như nguyên, logic, ký tự, liệt kê).

3. Tập hằng nào có chứa giá trị tương đương với giá trị của <biểu thức> thì lệnh sau dấu ':' của tập hằng đó được thực hiện, sau đó máy thoát khỏi lệnh CASE.

4. Trong trường hợp tất cả các tập hằng không có chứa giá trị tương đương với giá trị của <biểu thức> thì lệnh sau từ khóa ELSE được thực hiện. Trường hợp này nếu không có cả phần ELSE <lệnh $n+1$ >; thì lệnh CASE này được thoát và không có lệnh nào sau dấu ':' được thực hiện.

4 Ví dụ 1: Viết chương trình nhập vào một điểm kiểm tra từ bàn phím và in kết quả xếp loại: loại Yếu (dưới 5 điểm), loại Trung bình (5, 6 điểm), loại Khá (7, 8 điểm), loại Giỏi (9, 10 điểm).

```
Var Diem : Byte;
Begin
  Write(' Nhập diem : ');
  Readln(Diem);
  Case Diem of
    0.. 4 : Write(' Xep loai yeu. ');
    5.. 6 : Write(' Xep loai Trung binh. ');
    7.. 8 : Write(' Xep loai Kha. ');
    9..10: Write(' Xep loai Gioi. ');
  Else
    Write(' Diem nhap sai. ');
  End;
  Readln;
End.
```

4 Ví dụ 2: Viết chương trình cho biết số ngày của một tháng. Thuật toán như sau:



— Giáo trình Lập trình Pascal căn bản —

— 25 —

- Nhập tháng vào biến *Thang*.
- Sau đó, dựa vào biến *Thang* để biết số ngày, số ngày này được đưa vào biến *SoNgay*. Trường hợp:
 - + Tháng 1, 3, 5, 7, 8, 10, 12: *SoNgay* := 31;
 - + Tháng 2:
 - Yêu cầu nhập năm vào biến *Nam*.
 - Ø Trường hợp *Nam* chia hết cho 4: *SoNgay* := 29;
 - Ø Trường hợp *Nam* không chia hết cho 4: *SoNgay* := 28;
 - + Tháng 4, 6, 9, 11: *SoNgay* := 30;
- In nội dung biến *SoNgay*.

Uses CRT;

Var SoNgay, Thang : Byte;

Nam : Integer;

Begin

ClrScr;

Write(' Ban kiểm tra tháng mấy (dạng số): ');

Readln(Thang);

Case Thang of

4, 6, 9, 11 : SoNgay := 30;

2 : Begin

Write(' Tháng này thuộc năm nào (4 chữ số): ');

Readln(Nam);

If Nam mod 4 = 0 then SoNgay := 29;

Else SoNgay := 28;

End

Else

SoNgay := 31;

End;

If Thang = 2 then

Writeln(' Tháng ', thang, ' / ', nam, ' có ', SoNgay, ' ngày. ')

Else Writeln(' Tháng ', thang, ' có ', SoNgay, ' ngày. ');

Readln

End.

III. Các câu lệnh lặp:



Trường hợp để giải quyết bài toán nào đó mà ta cần phải lặp đi lặp lại một công việc nào đó thì ta sẽ cần đến lệnh lặp. Số bước lặp có thể xác định hoặc không xác định. Trong ngôn ngữ Pascal có ba câu lệnh lặp là *FOR*, *REPEAT*, *WHILE*. Nếu số vòng lặp xác định thì ta sử dụng lệnh *FOR* còn vòng lặp không xác định thì ta sử dụng lệnh *REPEAT* hoặc *WHILE*. Tất cả các loại lệnh lặp phải có điểm dừng, cho dù đó là loại xác định hay không xác định.

1. Câu lệnh *FOR*:

Vòng lặp *FOR* có hai dạng là dạng *vòng lặp tiến* và *vòng lặp lùi*.

a. Dạng tiến:

Cú pháp: *FOR Biến := Biểu_thức1 TO Biểu_thức2 DO <Lệnh>*

Biến trong cấu trúc *FOR* gọi là biến điều khiển. Kiểu của biến điều khiển, *Biểu_thức1*, *Biểu_thức2* phải là kiểu vô hướng đếm được (như nguyên, logic, ký tự, liệt kê).

F *Giải thích sự hoạt động lệnh FOR dạng tiến:*

(1). Đầu tiên, *Biến* nhận giá trị của *biểu_thức1*.

(2). Máy kiểm tra *Biến* có nhỏ hơn hoặc bằng *biểu_thức2* hay không tức là xét điều kiện (*Biến* \leq *Biểu_thức2*) ?

(3). Nếu điều kiện trên là sai thì máy thoát khỏi vòng lặp *FOR* để thực hiện các lệnh kế tiếp sau vòng lặp *FOR*. Nếu điều kiện trên là đúng thì *<Lệnh>* được thực hiện, sau đó, *Biến* được tăng một giá trị và quay trở lại bước (2).

<Lệnh> sẽ được thực hiện $((\text{biểu_thức2} - \text{biểu_thức1}) + 1)$ lần.

b. Dạng lùi:

Cú pháp: *FOR Biến := Biểu_thức1 DOWNTO Biểu_thức2 DO <Lệnh>*

F *Giải thích sự hoạt động lệnh FOR dạng lùi:*

(1). Đầu tiên, *Biến* nhận giá trị của *biểu_thức1*.

(2). Máy kiểm tra *Biến* có lớn hơn hoặc bằng *biểu_thức2* hay không tức là xét điều kiện (*Biến* \geq *Biểu_thức2*) ?

(3). Nếu điều kiện trên là sai thì máy thoát khỏi vòng lặp *FOR* để thực hiện các lệnh kế tiếp sau vòng lặp *FOR*. Nếu điều kiện trên là đúng thì *<Lệnh>* được thực hiện, sau đó, *Biến* được giảm một giá trị và quay trở lại bước (2).



Ö Chú ý:

- Không được thay đổi giá trị của biến điều khiển bằng một lệnh bất kỳ trong vòng lặp *FOR*. Điều này có thể làm cho vòng lặp không có lối thoát và *dẫn đến treo máy*.

- Các *Biểu_thức1* và *Biểu_thức2* được ước lượng trước khi vào vòng lặp, do đó số vòng lặp không bị thay đổi. Ta có thể lợi dụng tính tăng hoặc giảm của biến điều khiển để gán giá trị của nó cho bất kỳ biến nào hoặc thực hiện công việc nào đó có tính chất tăng hoặc giảm.

4 Ví dụ 1: Chương trình in lên màn hình 3 câu *Chào các bạn !* có số thứ tự đứng trước mỗi câu.

```
Uses CRT;  
Var I : integer;  
Begin  
  ClrScr;  
  For I := 1 to 5 do  
    Writeln( I , ' => ' , ' Chao cac ban ' );  
  Readln;  
End;
```

4 Ví dụ 2: In lên màn hình 4 dòng chữ cái in thường và IN HOA theo chiều xuôi và chiều ngược.

```
Uses CRT;  
Var kt : Char;  
Begin  
  ClrScr;  
  For kt := 'a' to 'z' do  
    Write(kt : 3);  
  Writeln;  
  For kt := 'z' Downto 'a' do  
    Write(kt : 3);  
  Writeln;  
  For kt := 'A' to 'Z' do  
    Write(kt : 3);  
  Writeln;  
  For kt := 'Z' Downto 'A' do
```



```
Write(kt : 3);  
Readln;  
End.
```

4 Ví dụ 3: Chương trình in lên màn hình 256 ký tự của bảng mã *ASCII*.

```
Var i : Byte;  
Begin  
  For i := 0 to 255 do  
    Begin  
      Writeln(' Ma thu ', i, ' la : ', CHR(i) );  
      If (i+1) mod 22 = 0 then  
        Begin  
          Write(' An phim bat ky de xem tiep ! ');  
          Readln;  
        End;  
    End;  
  Readln;  
End.
```

2. Câu lệnh Repeat:

Cú pháp:

```
REPEAT  
  <Lệnh 1>;  
  <Lệnh 2>;  
  .....  
  <Lệnh n>;  
UNTIL <Biểu thức logic>;
```

F Giải thích sự hoạt động lệnh REPEAT:

Đầu tiên, thực hiện lần lượt các lệnh <Lệnh 1>, <Lệnh 2>, ..., <Lệnh n>, sau đó kiểm tra <Biểu thức logic>. Nếu <Biểu thức logic> nhận giá trị *FALSE* thì lại quay lên đầu vòng lặp thực hiện tiếp <Lệnh 1>, <Lệnh 2>, ..., <Lệnh n>. Nếu <Biểu thức logic> nhận giá trị *TRUE* thì máy thoát khỏi vòng lặp. Như vậy, các lệnh nằm giữa REPEAT... UNTIL được thực hiện ít nhất một lần.

Ö Chú ý:

- Các lệnh nằm giữa REPEAT và UNTIL không có từ khoá Begin và End.



- Trong vòng lặp phải có lệnh nào đó làm thay đổi giá trị một biến trong *<Biểu thức logic>* nhằm làm dừng vòng lặp, nếu không vòng lặp sẽ chạy mãi không ngừng dẫn đến treo máy.

4 Ví dụ 1: Chương trình yêu cầu nhập vào một mật khẩu là 'ttthcn' thì mới thoát khỏi chương trình.

```
Uses CRT;
Var Password : String[6];
Begin
  Repeat
    Write( ' Xin hay nhap mat khau : ' );
    Readln(Password);
  Until Password = 'ttthcn';
  Write( ' Ban da nhap dung mat khau ! ' );
  Delay(1000);
  Readln;
End.
```

F Giải thích lệnh: *Delay(1000)*: Thủ tục *Delay(n)* là thủ tục của *Unit CRT* tức là dừng một khoảng thời gian là *1000 xung nhịp* của máy, vì vậy, tùy theo tốc độ của máy mà có khoảng thời gian thực dừng lại khác nhau.

4 Ví dụ 2: Chương trình để sử dụng bàn phím giả thành phím đàn Piano với quy định: ấn phím *D* phát ra nốt *Do*, phím *R* là nốt *Re*, *M* = *Mi*, *F* = *Fa*, *S* = *Sol*, *L* = *La*, *S* = *Si*.

```
Uses CRT;
Var node : Char;
Begin
  ClrScr;
  Writeln( ' D = Do | R = Re | M = Mi | F = Fa | S = Sol | L = La | X = Si ' );
  Writeln( ' Q = Do cao | W = Re cao | E = Mi cao | K = Ket thuc ' );
  Repeat
    Node := ReadKey;
    Case Node of
      'd' : Begin NoSound; Sound(262); End;
      'r' : Begin NoSound; Sound(294); End;
      'm' : Begin NoSound; Sound(330); End;
```



```
'f' : Begin NoSound; Sound(349); End;
's' : Begin NoSound; Sound(392); End;
'l' : Begin NoSound; Sound(440); End;
'x' : Begin NoSound; Sound(494); End;
'q' : Begin NoSound; Sound(523); End;
'w' : Begin NoSound; Sound(587); End;
'e' : Begin NoSound; Sound(659); End;
End;
Until (Ucase(Node) = 'K');
NoSound;
End.
```

Ồ Ghi chú: Thủ tục *Sound(*n*)* dùng để phát một âm thanh có tần số *n Hertz* cho đến khi gặp hàm *NoSound* (*ngừng phát âm thanh*), hai thủ tục trên thường đi đôi với nhau khi sử dụng. Những chương trình cần sự lặp đi lặp lại theo ý muốn thường sử dụng vòng lặp *Repeat... Until*. Cách thực hiện như sau:

```
Var TiepTuc : Char;
.....
Begin
  Repeat
    <... Các lệnh của chương trình >
    Write( ' Co tiep tục nua không (C/K) ? ' );
    Readln(TiepTuc);
  Until Ucase(TiepTuc) = 'K';
End.
```

3. Câu lệnh While:

Cú pháp:

WHILE < *Biểu thức logic* > *DO*
 < *Lệnh* >;

F Giải thích lệnh: Gặp lệnh này trước tiên máy kiểm tra < *Biểu thức logic* >, nếu nó có giá trị *TRUE* thì thực hiện < *Lệnh* > và sau đó quay lại kiểm tra < *Biểu thức logic* > và quá trình cứ tiếp tục như vậy. Nếu < *Biểu thức logic* > nhận giá trị *FALSE* thì máy lập tức thoát khỏi vòng lặp. Như vậy lệnh *WHILE* dùng để lặp đi lặp lại một công việc trong khi điều kiện còn được thỏa mãn.



Ö Ghi chú: Nếu ngay từ khi mới vào vòng lặp mà thấy điều kiện không được thỏa mãn, máy tự động thoát ngay mà không thực hiện <Lệnh> bên trong vòng lặp.

4 Ví dụ: Chương trình tìm ước số chung lớn nhất của hai số nguyên.

```
Var a, b, r : Integer; tl : Char;
Begin
  Repeat
    Write( ' Nhập hai số a và b : ' );
    Readln(a, b);
    While b <> 0 do
      Begin
        r := a mod b;
        a := b;
        b := r;
      End;
    Writeln( ' Ước số chung lớn nhất là ', a );
    Write( ' Bạn tìm USCLN nữa không (C/K) ? ');
    Readln(tl);
  Until Upcase(tl) = 'K';
End.
```

IV. Các lệnh Goto, Break, Exit và Halt:

1. Lệnh Goto:

Cú pháp:

GOTO Lab;

Trong đó, *Lab* là một nhãn. Nhãn là một tên *như tên biến* hoặc là *một số nguyên từ 0 đến 9999*. Tên nhãn được khai báo theo hướng dẫn ở bài 1 (IV.2).

Khi gặp lệnh *Goto Lab*, máy nhảy không điều kiện đến thực hiện câu lệnh sau nhãn *Lab*.

Lệnh Goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong cùng một thân hàm, thủ tục, cho phép nhảy từ trong một vòng lặp ra ngoài; không cho phép nhảy từ ngoài vào trong một vòng lặp, thủ tục, hàm hoặc khối lệnh.

4 Ví dụ: Chương trình tìm các số nguyên tố nằm giữa hai số nguyên dương *n1* và *n2*, hai số này được nhập từ bàn phím (*khái niệm số nguyên tố: là số nguyên chỉ chia hết cho 1 và chính nó*).



```
Program NguyenToByGoto;
Label L1, L2;
Var i, j, n1, n2 : Integer;
    TL : Char;
Begin
    L1: Write( 'Nhap hai gia tri nguyen : ' );
    Readln(n1, n2);
    For i := n1 to n2 do
        Begin
            For j := 2 to i - 1 do
                If (i mod j = 0) then Goto L2;
            Write( i, ' ');
            L2: ; {; cũng là một lệnh, nhưng là lệnh rỗng, tức là không làm gì cả }
        End;
    Writeln;
    Write( ' Ban muon tiep tục không ? (C/K) ' );
    Readln(TL);
    If (Ucase(TL) = 'C') then Goto L1;
End.
```

2. Lệnh Break:

Trong thân các lệnh lặp *FOR*, *WHILE*, *REPEAT* khi gặp lệnh *Break* thì máy sẽ thoát khỏi chu trình. Nếu có nhiều lệnh lặp lồng nhau thì máy thoát khỏi chu trình trong nhất chứa lệnh *Break*.

4 Ví dụ: In ra màn hình 4 dãy số từ 1 đến 49.

```
Uses CRT;
Var i, j : Integer;
Begin
    ClrScr;
    For j := 1 to 4 do
        Begin
            Writeln;
            Writeln( 'j = ', j );
            For i := 1 to 300 do { * }
                Begin If i = 50 then
```




```
Break; { Thoát khỏi vòng lặp For * }
Write( i, ' ' );
End;
Readln;
End;
Readln;
End.
```

3. Lệnh Exit:

Nếu lệnh *Exit* thuộc chương trình con thì việc thực hiện *Exit* làm chấm dứt chương trình con, trở về chỗ gọi nó. Nếu lệnh *Exit* thuộc chương trình chính thì việc thực hiện nó sẽ làm chấm dứt chương trình.

4 Ví dụ: Chương trình cứ nhắc lại câu *Welcome to Turbo Pascal Language* sau mỗi lần ấn một phím. Chương trình sẽ thoát khi ấn phím *E* hoặc *e*.

```
Uses CRT;
Label L1;
Var TL : Char;
Begin
  L1: Writeln( ' Welcome to Turbo Pascal Language ! ' );
  TL := Readkey; { Chờ một phím được ấn, giá trị được đặt vào biến TL, đây là
                  hàm của Unit CRT }
  If (Uppcase(TL) = 'E') then
    Exit
  Else
    Goto L1;
End.
```

4. Lệnh Halt:

Lệnh *Halt* dùng để dừng ngay chương trình đang chạy. Lệnh *Halt* thường được dùng khi phải một trường hợp nào đó mà thuật toán không thể tiếp tục được.



BÀI 5. DỮ LIỆU KIỂU VÔ HƯỚNG LIỆT KÊ VÀ KIỂU ĐOẠN CON

I. Kiểu liệt kê:

Kiểu liệt kê được định nghĩa bằng cách liệt kê tất cả các giá trị của kiểu thông qua các tên do người lập trình đặt ra và danh sách các giá trị trên được đặt trong cặp ngoặc đơn ().

4 Ví dụ:

```
Type Days = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);  
Viec = (DiHoc, LamBai, ThiNghiem, Nghi);
```

Khi đó, ta có thể khai báo biến như sau:

```
Var HomQua, HomNay : Days;  
Lam : Viec;
```

Hoặc ta có thể khai báo trực tiếp với mô tả kiểu dữ liệu như sau:

```
Var GioiTinh : (Nam, Nu);  
Color : (Red, Blue, Green, White, Black);
```

Ồ Chú ý:

(1). Có thể thực hiện phép gán trên các trị kiểu liệt kê, ví dụ:

```
Lam := Nghi;  
Color := Blue;
```

(2). Các giá trị của các kiểu liệt kê có thể so sánh với nhau theo quy định: Giá trị đứng trước nhỏ hơn giá trị đứng sau. Ta chỉ sử dụng toán tử so sánh cho kiểu liệt kê và cũng là toán tử duy nhất dùng cho kiểu này.

4 Ví dụ: Theo như khai báo trên, nếu so sánh $Thu < Fri$ cho kết quả *True*, hoặc $Red \geq Blue$ cho kết quả *False*.

(3). Các hàm chuẩn áp cho kiểu liệt kê:

- Hàm *ORD*: Cho thứ tự trị của đối số trong kiểu liệt kê.

4 Ví dụ: theo như khai báo trên, $ORD(Sun) = 0$, $ORD(Mon) = 1$.

- Hàm *PRED*: Cho trị đứng trước của đối số trong kiểu liệt kê.



4 Ví dụ: theo như khai báo trên, $PRED(Sat) = Fri$, $PRED(LamBai) = DiHoc$. $PRED(Sun)$ ⚡ lỗi chương trình.

- Hàm *SUCC*: Cho trị đi sau đối số trong kiểu liệt kê.

4 Ví dụ: theo như khai báo trên, $SUCC(Fri) = Sat$. $SUCC(Sat)$ ⚡ lỗi chương trình.

(4). Không thể *nhập, xuất đối với dữ liệu kiểu liệt kê*. Giá trị thuộc kiểu liệt kê thường được dùng để làm chỉ số cho vòng lặp *FOR*, các trường hợp lựa chọn trong lệnh *CASE*, chỉ số cho các mảng (*Array*).

4 Ví dụ: Chương trình đổi thứ trong tuần ra số. *Chủ nhật ứng với số 0, Thứ hai ứng với số 1,...*

Type

Thu = (ChuNhat, ThuHai, ThuBa, ThuTu, ThuNam, ThuSau, ThuBay);

Var

Ngay : Thu;

Begin

Writeln(' Chuong trinh doi thu ra so ');

For Ngay := ChuNhat to ThuBay do

Write(Ord(Ngay));

Readln;

End.

II. Kiểu đoạn con:

Kiểu đoạn con được định nghĩa do người dùng dựa trên cơ sở các kiểu vô hướng đếm được (*Nguyên, Logic, Ký tự, Liệt kê*) theo dạng:

$Tên_kiểu_đoạn_con = Hằng_dưới..Hằng_trên;$

Trong đó: *Hằng_dưới, Hằng_trên* là các giá trị hằng có cùng kiểu giá trị và thỏa mãn điều kiện: $Hằng_dưới < Hằng_trên$. Khi đó, các giá trị của kiểu đoạn con sẽ xác định trong khoảng từ *Hằng_dưới* đến *Hằng_trên*.

4 Ví dụ:

Type

Ky_so = '0'..'9'; { Kiểu gồm các ký tự số từ '0' đến '9' }

Ngay = (Hai, Ba, Tu, Nam, Sau, Bay, ChuNhat);

Ngay_Lam_Viec = Hai.. Bay; {Kiểu Ngay_Lam_Viec là khoản con của kiểu Ngay }



ChiSo = 1.. 50; { Kiểu ChiSo gồm các số nguyên từ 1 đến 50 }

Tuoi_Lam_Viec = 18.. 50;

Kiểu miền con giúp cho chương trình dễ đọc, dễ kiểm tra và tiết kiệm bộ nhớ.

BÀI 6. KIỂU TẬP HỢP VÀ KIỂU MẢNG

I. Kiểu tập hợp:

1. Định nghĩa:

Dữ liệu kiểu tập hợp là một tập hợp của những dữ liệu cùng thuộc một kiểu vô hướng đếm được. Một kiểu tập hợp được khai báo theo dạng sau:

SET OF Kiểu_cơ_số;

4 Ví dụ:

Type

Chu_so = Set of 0.. 9;

Chu_hoa = Set of 'A'.. 'Z';

Var

So : Chu_so;

Chu : Chu_hoa;

Mau : Set of (Xanh, Vang, Tim);

Ồ Chú ý:

- Các giá trị được đưa vào tập hợp cần có số thứ tự trong khoảng từ 0 đến 255.
- Như vậy, với khai báo:

Type

Tap_so = Set of 10.. 256;

1 Kết quả khi dịch máy sẽ thông báo lỗi: *Set base type out of range.*

- Một dữ liệu kiểu tập hợp có dạng các phần tử nằm trong hai dấu ngoặc *[]*. Ví dụ: *['A', 'D', 'E'], [3, 5.. 9];*

- Tập hợp rỗng ký hiệu là *[]*.
- Biến tập hợp cho phép có từ 0 đến 256 phần tử.
- Có thể thực hiện phép gán trên kiểu tập hợp. Ví dụ:

So := [0, 4, 9];

Chu := []; {Tập hợp rỗng}



Mau := [Vang, Tim];

2. Các phép toán trên tập hợp:

a. Phép toán quan hệ:

Phép toán = **Đ** cho giá trị *True* nếu hai tập hợp bằng nhau.

Phép toán < > **Đ** cho giá trị *True* nếu hai tập hợp khác nhau.

Phép toán <= **Đ** $A \leq B$ cho giá trị *True* nếu A là tập con của B .

Phép toán >= **Đ** $A \geq B$ cho giá trị *True* nếu B là tập con của A .

Ồ Chú ý: Không có *phép toán* < và > cho kiểu tập hợp. Để kiểm tra tập hợp A có thật sự nằm trong B hay không ta dùng câu lệnh:

If (A < > B) and (A <= B) then Write('A là tập con thật sự của B ');

b. Phép toán IN:

Phép toán *IN* dùng để xem xét một phần tử nào đó có nằm trong tập hợp không ? Nếu phần tử đó có trong tập hợp thì phép toán sẽ trả về giá trị *True*, ngược lại cho giá trị *False*. Ví dụ:

'C' In ['A', 'C', 'D'] cho kết quả *True*.

'E' In ['A', 'C', 'D'] cho kết quả *False*.

c. Phép toán hợp, giao, hiệu:

Gọi A, B là hai tập hợp cùng kiểu dữ liệu.

$A + B$ là hợp của A và B : tập hợp các phần tử thuộc A hoặc thuộc B .

$A * B$ là giao của A và B : tập hợp các phần tử thuộc A và thuộc B .

$A - B$ là hiệu của A và B : tập hợp các phần tử thuộc A và không thuộc B .

4 Ví dụ:

A := [1, 3, 9];

B := [9, 2, 5];

Vậy:

$A * B$ có giá trị là $[9]$.

$A - B$ có giá trị là $[1, 3]$.

4 Ví dụ: Viết chương trình nhập vào một chữ cái. Xét xem chữ cái đó là nguyên âm hay phụ âm.

Var

ChuCai, NguyenAm : Set of Char;

Ch : char;



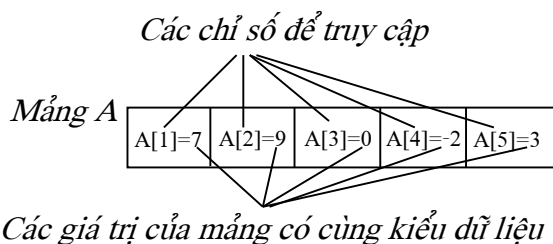
```
Begin
  ChuCai := ['A'..'Z', 'a'..'z'];
  NguyenAm := ['A', 'E', 'I', 'O', 'U'];
  Repeat
    Write( ' Nhập một chu cai để kiểm tra: ' );
    Readln(Ch);
  Until Ch IN ChuCai;
  If Upcase(Ch) IN NguyenAm then
    Writeln(Ch, ' là nguyên âm. ' )
  Else
    Writeln(Ch, ' là phụ âm. ');
  Readln;
End.
```

II. Kiểu mảng:

1. Khái niệm:

Mảng (*Array*) là một kiểu dữ liệu có cấu trúc bao gồm một số cố định các thành phần có cùng kiểu, có cùng một tên chung. Các thành phần của mảng được truy xuất thông qua các chỉ số.

4 Ví dụ: Mảng A gồm năm phần tử: $A[1]=7$, $A[2]=9$, $A[3]=0$, $A[4]=-2$, $A[5]=3$:



Công dụng của mảng là dùng để lưu trữ một dãy số liệu có cùng một tính chất nào đó. Ví dụ: các điểm kiểm tra một môn học nào đó của một học sinh, các giá trị của một dãy số được nhập từ bàn phím.

2. Khai báo mảng một chiều:

Type

Tên_kiểu_mảng = ARRAY [Chỉ_số] OF Kiểu_phần_tử;

Var

Tên_biến_mảng : Tên_kiểu_mảng;



Trong đó:

- *Kiểu_phần_tử* là kiểu dữ liệu của mỗi phần tử trong mảng (*là kiểu bất kỳ*).
- *Chỉ_số* là danh sách các chỉ số để truy cập đến các thành phần của mảng.

Các chỉ số có thể là:

- + Một đoạn con, ví dụ:

Type

Ho_Ten = Array[1..100] of String[30];

He_so_luong = Array[1..100] of Real;

- + Một danh sách liệt kê, ví dụ:

Type

Toc_do = Array[(Oto, Tai, Buyt, GanMay)] of Integer;

- + Một kiểu dữ liệu, ví dụ:

Type

ASCIIType = Array[Byte] of Char;

Xe = (Oto, Tai, Buyt, GanMay);

Toc_do = Array[Xe] of Integer;

Với các kiểu mảng trên, ta có thể khai báo các biến mảng sau:

Var

HeSo : He_so_luong;

HT : Ho_Ten;

Speed : Toc_do;

Ngoài cách định nghĩa *Tên_kiểu_mảng* như ở trên ta cũng có thể khai báo một biến mảng trực tiếp sau lệnh *VAR*:

Var ch : Array[0.. 25] of Char;

Th : Array[-2.. 4] of Real;

3. Truy cập các phần tử của mảng:

Việc truy nhập vào một phần tử nào đó của biến mảng được thực hiện qua tên biến mảng, theo sau là giá trị chỉ số đặt trong dấu *[]*. Ví dụ:

Ch[2] := 'B';

Th[1] := 12.5;

HT[1] := 'Vu Duc Dung';



4 Ví dụ 1: Nhập n số thực từ bàn phím vào một mảng, tính trung bình cộng của các số này.

```
Uses CRT;
Var i,n : Integer;
    s : Real;
    a : Array[1.. 100] of Real;
Begin
  ClrScr;
  Write( ' Ban muon nhap bao nhieu PT cho mang : ' );
  Readln(n);
  For i := 1 to n do
    Begin
      Write( ' PT A[ ', i , ' ]= ' );
      Readln(a[i]);
    End;
  s := 0;
  For i := 1 to n do
    s := s + a[i];
  Write( ' Trung binh cong cua day so = ', s / n : 0 : 4 );
  Readln;
End.
```

4 Ví dụ 2: Nhập từ bàn phím n phần tử thực của một mảng, sắp xếp dãy theo thứ tự tăng dần, xuất giá trị của mảng lên màn hình.

```
Var a : array[1..10] of Real;
    b : array[1..10] of Real;
    temp : Real;
    i, j, n : integer;
Begin
  n:=10;
  For i := 1 to n do
    Begin
      Write( ' PT thu ', i , ':' );
      Readln( a[i] );
    End;
```




```
For i := 1 to n - 1 do
  For j := n downto i do
    If a[i] > a[j] then
      Begin
        temp := a[i];
        a[i] := a[j];
        a[j] := temp;
      End;
  For i := 1 to n do
    Write( a[i] : 0 : 3 , ' ');
  Readln;
End.
```

4. Mảng nhiều chiều:

Phần này chủ yếu trình bày các mảng hai chiều. Các mảng nhiều hơn hai chiều được suy diễn một cách tự nhiên.

Việc khai báo mảng hai chiều cũng giống như mảng một chiều, chỉ có điều khác là nó có hai tập chỉ số được viết cách nhau bởi dấu ','.

4 Ví dụ:

Type

Mang1 = Array[1.. 30, 1.. 50] of Integer;

Mang2 = Array[1.. 3, 0.. 2] of Real;

Var

A : Mang1;

B : Mang2;

Trong đó, số phần tử của mảng số thực B là $3 \times 3 = 9$ (phần tử), sắp đặt trong bộ nhớ theo thứ tự như sau:

$B[1, 0] \ B[1, 1] \ B[1, 2]$

$B[2, 0] \ B[2, 1] \ B[2, 2]$

$B[3, 0] \ B[3, 1] \ B[3, 2]$

⚠ **Chú ý:** Mảng hai chiều còn gọi là ma trận. Trong ví dụ trên, B là ma trận cấp 3×3 . Trong mảng hai chiều, chỉ số sau truy cập nhanh hơn chỉ số trước. Để truy cập đến phần tử *hàng thứ i, cột thứ j* của mảng hai chiều B ta dùng cách viết:

$B[i][j]$



hoặc

$B[i, j]$

4 Ví dụ: Nhập một ma trận m hàng, n cột từ bàn phím. Tính và in ra màn hình tổng của mỗi cột và tổng của mỗi hàng.

```
Const mMax = 30, nMax = 30;
Type
  Mang = Array[1.. mMax, 1.. nMax] of Real;
Var
  n, m, i, j : Integer;
  sum : Real;
  a : Mang;
Begin
  Write( ' Ban muon nhap ma tran bao nhieu hang va cot ? ' );
  Readln( m, n );
  For i := 1 to m do
    For j := 1 to n do
      Begin
        Write( ' PT thu [ ', i, ', ', j, ' ] = ' );
        Readln( a[ i, j ] );
      End;
    For j := 1 to n do
      Begin
        sum := 0;
        For i := 1 to m do
          Sum := sum + a[ i, j ];
        Write( ' Tong cot ', j, ' = ', sum : 0 : 5 );
      End;
    For i := 1 to m do
      Begin
        sum := 0;
        For j := 1 to n do
          Sum := sum + a[ i, j ];
        Write( ' Tong hang ', i, ' = ', sum : 0 : 5 );
      End;
    Readln;
```



End.

_____ o² o _____

BÀI 7. CHƯƠNG TRÌNH CON: HÀM VÀ THỦ TỤC

Khi lập trình, có những đoạn chương trình cần dùng nhiều lần. Để tránh việc viết lại đoạn này, ta nên chuyển đoạn chương trình này thành một chương trình con và mỗi lần cần thực hiện công việc đó thì ta gọi nó thông qua tên.

Chương trình con còn để mẫu hoá một chương trình làm công việc nào đó. Người khác dùng chương trình con *chỉ cần biết truyền số liệu vào và lấy kết quả ra như thế nào mà không cần phải quan tâm đến thuật toán trong chương trình con như thế nào*.

Khi viết những chương trình lớn, để dễ dàng quản lý, gọn rối và hiệu chỉnh chương trình, ta nên phân chương trình thành nhiều *công việc độc lập*, mỗi công việc là một chương trình con. Chương trình con gồm có hai loại là *HÀM (Function)* và *THỦ TỤC (Procedure)*.

I. Hàm và thủ tục:

Cấu trúc của hàm có dạng:

```
FUNCTION Tên_Hàm(ThamSố1: Kiểu; TS2: Kiểu;... ) : Kiểu;  
  Var Các_biến_cục_bộ;  
  Begin  
    Các_lệnh_tính_toán;  
    ...;  
    Tên_Hàm := Giá_trị;  
  End;
```

Phương pháp gọi hàm: ta gọi hàm thông qua tên kèm theo tham số của hàm như sau:

Tên_hàm(Danh sách các tham số thực sự);

Cấu trúc của thủ tục có dạng:

```
PROCEDURE Tên_Thủ_tục(TS1: Kiểu; TS2: Kiểu;...; Var TS3: Kiểu; Var TS4:  
Kiểu;... );
```



Var các biến cục bộ;

Begin

Các lệnh;

...;

End;

Phương pháp gọi thủ tục:

Tên_hàm(Danh sách các tham số thực sự);

Sự khác nhau cơ bản giữa hàm và thủ tục là hàm trả về một giá trị thông qua tên hàm, hàm có thể tham gia vào các biểu thức tính toán còn thủ tục không cho giá trị nào cả. Khi tạo hàm, trong thân hàm bao giờ cũng có giá trị gán cho tên hàm để hàm trả về giá trị này khi được gọi.

Các tham số khác sau tên hàm và tên thủ tục gọi là các tham số hình thức (hay còn gọi là đối). Trong thủ tục, các tham số hình thức có hai loại: các tham số được khai báo sau từ khoá *Var* gọi là tham số biến, các số khai báo không có từ khoá *Var* ở trước gọi là tham số giá trị. Trong hàm chỉ có tham số giá trị, tức khai báo mà không có từ khoá *Var*.

Tham số thực sự là các tham số dùng trong lời gọi hàm hay thủ tục. Danh sách các tham số thực sự trong lời gọi hàm phải tương ứng với danh sách các tham số hình thức trong phần khai báo chương trình con và chúng phải tương ứng về kiểu.

Trong thủ tục, các tham số giá trị thường là các biến để chứa dữ liệu đưa vào thủ tục; các tham số biến là các biến mà kết quả tính toán của thủ tục sẽ chứa vào đó khi ra khỏi thủ tục, ta có thể dùng chúng để tính toán tiếp.

4 Ví dụ cách sử dụng tham số giá trị và tham số biến:

Var a, b, c, d : Integer;

Procedure Chuyen(x, y: Integer; Var u, v: Integer);

Begin { Từ khoá bắt đầu thủ tục Chuyen }

*x := 2 * x;*

*y := 3 * y;*

*u := 4 * u;*

*v := 5 * v;*

End;

Begin { Từ khoá bắt đầu chương trình chính }

a := 10;



```
b := 10;  
c := 10;  
d := 10;  
Chuyen(a, b, c, d);  
Write('a = ', a, ' b = ', b, ' c = ', c, ' d = ', d);  
Readln;  
End.
```

1 Kết quả khi chạy chương trình: $a = 10$. $b = 10$. $c = 40$. $d = 50$

II. Biến toàn cục, biến cục bộ và việc truyền dữ liệu:

Biến toàn cục là biến khai báo ở đầu chương trình chính, tồn tại trong suốt thời gian làm việc của chương trình. Ta có thể sử dụng và làm thay đổi giá trị của biến toàn cục nhờ các câu lệnh trong chương trình chính cũng như trong tất cả các chương trình con.

Biến cục bộ là biến là biến khai báo ở đầu chương trình con. Chúng được cấp phát bộ nhớ khi chương trình con được gọi đến và bị xóa khi máy thoát khỏi chương trình con đó. Biến cục bộ có giá trị trong chương trình con và tất cả các chương trình con khác nằm trong chương trình con này.

Nếu tên biến cục bộ của một chương trình con trùng với một tên biến toàn cục thì máy không bị nhầm lẫn, máy sẽ dùng hai ô nhớ khác nhau để lưu trữ hai biến, khi ra khỏi chương trình con, biến cục bộ tự động được xóa.

Khi gặp một lời gọi đến chương trình con, máy sẽ thực hiện các bước sau:

- Cấp phát bộ nhớ cho các đối, các biến cục bộ.
- Truyền giá trị của các tham số thực sự cho các tham số giá trị tương ứng, truyền địa chỉ các tham số thực sự ứng với tham số biến cho các tham số biến của thủ tục.
- Thực hiện các lệnh trong chương trình con, trong khi thực hiện chương trình con, các biến cục bộ và các tham số giá trị có thể bị biến đổi nhưng không ảnh hưởng đến các biến bên ngoài. Trái lại, *mọi thay đổi của tham số biến trong chương trình con sẽ kéo theo sự thay đổi của tham số thực sự tương ứng (vì có sự truyền theo địa chỉ)*. Do đó, *khi thoát khỏi chương trình con, các tham số thực sự ứng với tham số biến vẫn giữ được giá trị mới nhất do chương trình con tạo ra*.
- Thực hiện xong các lệnh của chương trình con, máy xóa tất cả các đối và các biến cục bộ và trở về lệnh kế sau nơi gọi nó.



Việc lấy kết quả thực hiện chương trình con như sau: nếu là *hàm* thì lấy kết quả thông qua tên hàm, nếu là *thủ tục* thì kết quả ở tham số thực sự ứng với tham số biến. Khi cần lấy duy nhất một giá trị từ chương trình con thì ta lập một *FUNCTION*, khi cần lấy từ hai giá trị trở lên từ chương trình con hoặc không lấy giá trị nào thì ta phải lập *PROCEDURE*.

4 Ví dụ 1: Lập hàm tính diện tích hình thang. Nhập dữ liệu của hai thửa ruộng hình thang và tính tổng diện tích hai thửa ruộng.

```
Var a1, b1, h1, a2, b2, h2, s : Real;
(***** Bat dau Function *****)
Function DTHinhThang(a, b, h) : Real;
Begin
    DTHinhThang := (a + b) * h / 2;
End;
(***** Bat dau chương trình chính *****)
Begin
    Write( ' Canh dai, ngan va cao cua thua ruong thu nhât: ' );
    Readln(a1, b1, h1);
    Write( ' Canh dai, ngan va cao cua thua ruong thu hai: ' );
    Readln(a2, b2, h2);
    s := DTHinhThang(a1, b1, h1) + DTHinhThang(a2, b2, h2);
    Writeln( ' Tong dien tich hai thua ruong = ', s : 0 : 3);
    Readln;
End.
```

4 Ví dụ 2: Lập hàm tính ước số chung lớn nhất (*USCLN*). Sau đó, dùng hàm này để tính *USCLN* và bội số chung nhỏ nhất (*BSCNN*) của hai số được nhập từ bàn phím.

```
Var m, n, usc, bsc: Integer;
(***** Function USCLN *****)
Function USCLN(a, b : Integer): Integer;
Var r : Integer;
Begin
    While b <> 0 do
        Begin
            r := a mod b;
```



```
    a := b;
    b := r;
  End; { a hiện tại là USCLN của a và b ban đầu }
  USCLN := a;
End;
(***** bắt đầu chương trình chính *****)
Begin
  Write( ' Nhập số thứ nhất : ' );
  Readln(m);
  Write( ' Nhập số thứ hai: ' );
  Readln(n);
  usc := USCLN(m, n);
  bsc := m * n div USCLN(m, n);
  Writeln( ' Ước số chung lớn nhất của ', m, ' và ', n, ' là : ', usc );
  Writeln( ' Bội số chung nhỏ nhất của ', m, ' và ', n, ' là : ', bsc );
  Readln;
End.
```

4 Ví dụ 3: Lập một thủ tục để tính đồng thời diện tích và thể tích hình cầu.

```
Var r, s, v : Real;
Reply : Char;
(***** Function *****)
Procedure SVHinhCau( r : Real; Var s, v :Real);
Begin
  s := 4 * pi * r * r;
  v := 4 * pi * r * r * r / 3;
End;
(***** bắt đầu chương trình chính *****)
Begin
  Repeat
    Write( ' Nhập bán kính hình cầu : ' );
    Readln(r);
    SVHinhCau(r, s, v);
    Writeln( ' Diện tích = ', s : 0 : 4, ' . Thể tích = ', v : 0 : 4 );
    Write( ' Bạn có tiếp tục không?(C/K) ' );
    Readln(Reply);
```



```
Until Upcase(Reply) = 'K';  
End.
```

III. Các hàm và thủ tục thường dùng của Unit CRT:

Unit CRT có nhiều hàm, thủ tục dùng để điều khiển màn hình, bàn phím và âm thanh. Nó cho phép mở các cửa sổ với các màu sắc khác nhau, thay đổi màu của các dòng chữ trên màn hình, giúp cho việc trình bày màn hình đẹp và hấp dẫn hơn, tổ chức hội thoại giữa người và máy thuận tiện. Khi dùng các hàm và thủ tục này, ở đầu chương trình chính cần phải có khai báo *USES CRT*; Các thủ tục của *Unit CRT* gồm:

1. Thủ tục ClrScr:

Xoá màn hình và đưa con trỏ về vị trí (1,1) trên màn hình. Màn hình mặc định được chia thành 25 dòng và 80 cột. Cột đầu tiên đánh số 1, dòng đầu tiên đánh số 1.

2. Thủ tục ClrEOL:

Xoá từ vị trí con trỏ đến cuối dòng hiện hành. Sau khi thực hiện xong, con trỏ đứng ngay vị trí trước khi gọi thực hiện thủ tục.

3. Thủ tục DelLine:

Xoá dòng con trỏ đang đứng, các dòng sau sẽ được chuyển lên trên một dòng.

4. Thủ tục InsLine:

Chèn dòng trống vào vị trí hiện hành của con trỏ trên màn hình.

5. Thủ tục GotoXY(x, y: Byte):

Đưa con trỏ đến, cột thứ x, dòng thứ y.

6. Hàm WhereX: Byte

Cho giá trị kiểu byte cho biết con trỏ đang ở cột nào.

7. Hàm WhereY: Byte

Cho giá trị kiểu byte cho biết con trỏ đang ở dòng nào.

8. Thủ tục Sound(Hz : Word):

Phát âm thanh có tần số Hz cho đến khi gặp thủ tục NoSound thì dừng lại.

9. Thủ tục NoSound:

Tắt loa phát âm thanh ở máy.

10. Thủ tục TextBackGround(Color : Byte):



Chọn màu nền trong chế độ văn bản (*Chế độ mặc định khi chạy Pascal*). Color có giá trị từ 0 đến 7.

11. Thủ tục TextColor(Color : Byte):

Chọn màu của ký tự trình bày trên màn hình. Color có giá trị từ 0 đến 15 ứng với 16 màu. Các hằng xác định màu nền và chữ cho biến Color như sau:

Black (<i>đen</i>)	= 0	DarkGray (<i>xám</i>)	= 8
Blue (<i>xanh dương</i>)	= 1	LightBlue (<i>xanh dương nhạt</i>)	= 9
Green (<i>xanh lục</i>)	= 2	LightGreen (<i>xanh lục nhạt</i>)	= 10
Cyan (<i>lam</i>)	= 3	LightCyan (<i>lam nhạt</i>)	= 11
Red (<i>đỏ</i>)	= 4	LightRed (<i>đỏ nhạt</i>)	= 12
Magenta (<i>tím</i>)	= 5	LightMagenta (<i>tím nhạt</i>)	= 13
Brown (<i>nâu</i>)	= 6	Yellow (<i>vàng</i>)	= 14
LightGray (<i>xám nhạt</i>)	= 7	White (<i>trắng</i>)	= 15

Ö Ghi chú: Ta có thể dùng các hằng giá trị trên bằng chữ hoặc số đều được. Ví dụ: *TextColor(4)* hoặc *TextColor(Red)* đều có ý nghĩa là chọn chữ màu đỏ. Chọn chữ màu xanh và chữ nhấp nháy: *TextColor(Green + Blink)*.

12. Hàm KeyPressed: Boolean

Hàm kiểm tra xem có phím nào được ấn trên bàn phím hay không. Nếu có hàm trả về giá trị *True*, nếu không hàm cho giá trị *False*.

13. Hàm ReadKey: Char

Hàm này chờ đọc một ký tự từ bàn phím (*ký tự được nhập không được hiển thị trên màn hình*). Các phím trên bàn phím như *A, B, C, ... 1, 2, 3, 4, ... v.v.* chỉ tạo một mã khi được ấn, còn các phím chức năng như *F1, F2, ..., Home, End, Alt, Ctrl, Ctrl - Home, ...* tạo hai mã khi được ấn, trong đó mã thứ nhất có giá trị 0. Để nhận biết một hay một tổ hợp phím bất kỳ được ấn, ta phải dùng một biến kiểu *Char* với hai lần thực hiện hàm *ReadKey* như sau:

Ch := ReadKey;

If Ch = #0 then Ch := Readkey;

Sau đây là một số phím đặc biệt và tổ hợp phím hay dùng:

<i>Esc</i>	<i>27</i>	<i>â</i>	<i>0/80</i>
<i>Tab</i>	<i>9</i>	<i>ß</i>	<i>0/75</i>
<i>Enter</i>	<i>13</i>	<i>à</i>	<i>0/77</i>
<i>Home</i>	<i>0/71</i>	<i>F1</i>	<i>0/59</i>
<i>End</i>	<i>0/79</i>	<i>F2</i>	<i>0/60</i>



<i>PageUp</i>	<i>0/73</i>	<i>F10</i>	<i>0/68</i>
<i>PageDown</i>	<i>0/81</i>	<i>Ctrl - F1</i>	<i>0/94</i>
<i>á</i>	<i>0/72</i>	<i>Ctrl - F2</i>	<i>0/95</i>

4 Ví dụ 1: Dịch chuyển con trỏ và in một số dòng chữ trên màn hình.

```
Uses CRT;
Var x, y : Integer;
Begin
  ClrScr;
  x := 20;
  y := 3;
  GotoXY(x + 2, y);
  Write(' PASCAL '); { In tu cot 22 dong 3 }
  GotoXY(x - 2, y + 2);
  Write(' BAN HAY DEN VOI '); { In tu cot 18 dong 5 }
  GotoXY(x, y + 3);
  Write(' TURBO PASCAL '); { In tu cot 20 dong 6 }
  GotoXY(WhereX + 2, WhereY);
  Write(' 7.0 '); { sau TURBO PASCAL in số 7.0 }
  Readln;
End.
```

4 Ví dụ 2: Nhận biết phím nào được ấn.

```
Uses CRT;
Var Ch : Char;
Begin
  Write(' Ban hay an mot phim bat ky : ');a
  Ch := ReadKey;
  If Ch := #0 then
    Begin
      Ch := Readkey;
      Writeln(' Ban vua an mot phim dac biet co ma = ', Ord(Ch));
    End
  Else
    Writeln(' Ban vua an mot phim co ma ASCII = ', Ord(Ch));
  Readln;
```



End.

4 Ví dụ 3: Viết chương trình hiển thị 16 dòng với nội dung bất kỳ, tại đầu mỗi dòng hiển thị số thứ tự của dòng đó đồng thời hiển thị màu của dòng đó theo số thứ tự (*theo bảng màu*).

```
Uses CRT;
Var i : Integer;
Begin
  For i := 0 to 15 do
    Begin
      TextColor( i );
      Writeln( i, ' là mã số màu của dòng này. ' );
    End;
  Readln;
End.
```

_____ o² o _____



BÀI 8. KIỂU XÂU KÝ TỰ

I. Khai báo và các phép toán:

Xâu (*String*) là kiểu dữ liệu có cấu trúc dùng để xử lý các chuỗi ký tự. Kiểu *String* có nhiều điểm tương tự như kiểu mảng (*Array*) nhưng cũng có điểm khác nhau là: số ký tự trong một biến kiểu chuỗi có thể thay đổi còn số phần tử của kiểu mảng luôn cố định.

1. Khai báo kiểu chuỗi:

VAR

Tên_Biến : String[n];

Trong đó: *n* là số ký tự tối đa có thể có của chuỗi. Chiều dài tối đa của một chuỗi là 255. Nếu trong phần khai báo không ghi *[n]* thì chuỗi có độ dài mặc định là 255.

4 Ví dụ:

Var

HoTen : String[30]; { HoTen có thể chứa tối đa 30 ký tự }

St : String; { St có thể chứa tối đa 255 ký tự }

Với *St* là một chuỗi, để chỉ ra các ký tự thứ *i* của *St* ta viết *St[i]*. Các *St[i]* đều có kiểu *Char*. Ví dụ: *St := 'ABCD';* thì lệnh *Write(St[3])* sẽ in ra ký tự 'C'.

Cấu trúc của *String* như sau: Trong bộ nhớ nó chiếm số *Byte* bằng số ký tự tối đa, cộng với một byte đầu tiên (tại vị trí *s[0]*) chứa ký tự mà mã thập phân *ASCII* của ký tự này sẽ cho biết chuỗi đó có độ dài bao nhiêu.

Chẳng hạn biến *HoTen* bên trên được gán giá trị:

HoTen := 'Ly Dong Giang';

Khi đó, độ dài chuỗi chỉ là 13, mặc dù độ dài cực đại cho phép là 30 như đã khai báo. Sau đây cấu trúc chuỗi *HoTen*:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17									26	27	28	29	30
Chr(13)	L	y		D	o	n	g		G	i	a	n	g	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Ồ Ghi chú: Ký tự * biểu diễn ký tự không xác định.

2. Nhập và in chuỗi ký tự:

Muốn in một chuỗi ký tự ta dùng lệnh *Write(St)* hoặc *Writeln(St)*.



Lệnh *Readln(St)* sẽ đọc các ký tự cho chuỗi *St* với độ dài thực sự là số ký tự gõ vào từ bàn phím. Nếu ta gõ *< Enter >* luôn mà không nhập cho nó ký tự nào thì *St* là chuỗi rỗng.

4 Ví dụ:

```
Var YourName, st1, st2 : String[40];
Begin
  Write( ' Please enter your name: ' );
  Readln(YourName);
  Writeln( ' Hello ', YourName + ' ! ' );
  st1 := ' Turbo Pascal ';
  st2 := ' Borland"s product is ' + st1;
  Writeln(st2);
  Readln;
End.
```

3. Các phép toán trên chuỗi ký tự:

a. Phép gán:

Biến := Biểu_thức;

Đại lượng bên phải của lệnh phải được đặt giữa hai dấu nháy đơn nếu đó là chuỗi ở dạng hằng. Ta có thể sử dụng dấu cộng (+) để ghép các chuỗi khi gán. Ví dụ: *HoTen := 'Huynh Ngoc' + ' Nhan';*

b. Phép nối String:

Ký hiệu bằng dấu +.

4 Ví dụ: 'Turbo' + ' Pascal' = 'Turbo Pascal'

c. Các phép toán so sánh:

Khi so sánh hai chuỗi, các ký tự của hai chuỗi được so sánh từng cặp một từ trái qua phải theo giá trị trong bảng mã *ASCII*.

4 Ví dụ:

Nếu so sánh:

'ABC' = 'ABC' có giá trị *True*.

'ABC' = 'AB' có giá trị là *False*.

'ABCD' < 'ABED' có giá trị là *True*.

'ABC' > 'AD' có giá trị là *False*.



II. Các thủ tục và hàm xử lý xâu ký tự:

1. Các thủ tục:

a. Delete(*St* , *Pos*, *Num*):

- Trong đó:
- *St* (*String*): Biến kiểu *String*.
 - *Pos* (*Position*): Biến kiểu nguyên.
 - *Num* (*Number*): Biến kiểu nguyên.

Công dụng: Thủ tục này dùng để xóa khỏi xâu *St* một số *Num* ký tự bắt đầu từ vị trí thứ *Pos*.

4 Ví dụ: Nếu *St* = 'ABCDEFGH'; thì:

- Delete*(*St*, 2, 4); ➤ làm cho *St* = 'AFH'.
Delete(*St*, 2, 10); ➤ làm cho *St* = 'A'.
Delete(*St*, 9, 3); ➤ làm cho *St* = 'ABCDEFGH'.

b. Insert(*St2*, *St1*, *Pos*):

- Trong đó:
- *St2* và *St1*: Biến kiểu *String*.
 - *Pos*: Biến kiểu nguyên.

Công dụng: Thủ tục này dùng để chèn xâu *St2* vào xâu *St1* ở vị trí *Pos*. Ví dụ: Nếu *St* := 'ABCD' thì sau lệnh *Insert*('TFG', *St*, 3) ta nhận được *St* := 'ABTFGCD'.

Trường hợp *Pos* vượt quá chiều dài của *St1* thì *St2* sẽ được nối đuôi vào *St1*. Ví dụ: *St* = 'ABCD', vậy lệnh *Insert*('TFG', *ST*, 9); sẽ làm cho *St* = 'ABCDTFG'.

c. Str(*Value*, *St*):

- Trong đó:
- *Value*: Là một biểu thức nguyên hay thực có ghi dạng in ra.
 - *St*: Biến kiểu *String*.

Công dụng: Thủ tục này dùng để đổi giá trị số *Value* thành kiểu xâu rồi gán cho *St*.

4 Ví dụ:

- i* := 1234;
Str(*i*:5, *St*); { ta được *St* = ' 1234' có 5 ký tự }
x := 123.5678901;
Str(*x*:10:5, *St*); { ta được *St* = ' 123.56789' }

d. Val(*St*, *Var*, *Code*):

- Trong đó:
- *St*: Biểu thức kiểu *String*.
 - *Var*: Là biến kiểu nguyên hay thực.
 - *Code*: Biến kiểu nguyên.



Công dụng: Thủ tục này đổi xâu chữ *St* (biểu diễn ở dạng số nguyên hay thực) thành số và gán cho biến *Var*. *Code* là biến nguyên dùng để phát hiện lỗi: nếu phép biến đổi đúng thì *Code* có giá trị 0, nếu sai do *St* không biểu diễn đúng số nguyên hay thực thì *Code* sẽ có giá trị bằng vị trí của ký tự sai trong xâu *St*. Ví dụ:

Giả sử: $St := '234'$, *i* và *e* là hai biến nguyên.

$Val(St, i, e)$; { cho ta $i = 234$ và $e = 0$ }

Nếu $St := '21x'$ thì $Val(St, i, e)$ { cho ta *i* không xác định và $e = 3$, tức là ký tự thứ ba gây ra lỗi }

4 Ví dụ về một ứng dụng có sử dụng thủ tục *Val* để đọc số nguyên từ bàn phím. Bình thường ta dùng thủ tục *Readln(i)* để đọc số nguyên *i*. Song nếu trong lúc nhập số, ta chẳng may gõ nhầm chữ cái vào thì máy dừng lại, có thể gây lãng phí thời gian. Thủ tục dưới đây có thể báo lỗi nếu ta nhập một số có chữ trong số đó.

Procedure InputInteger(Var i : Integer);

Var

St : String[6];

e : Integer;

Begin

Repeat

Readln(St); { Nhập vào xâu số nguyên }

Val(St, i, e); { Biến đổi và phát hiện lỗi }

If e <> 0 then

Writeln(#7, ' Loi nhap lieu ! ');

Until e = 0;

End;

2. Các hàm:

a. Length(St): cho ta độ dài của biểu thức xâu ký tự *St*. Ví dụ: với $St = 'ABCDEFGH'$ thì *Length(St)* sẽ trả về giá trị 7.

b. Copy(St, Pos, Num):

Trong đó: - *St*: Biểu thức kiểu xâu ký tự.

- *Pos, Num*: Biểu thức kiểu nguyên.

Hàm này trả về cho ta một xâu mới từ xâu *St*, hàm bắt đầu chép từ vị trí *Pos* và chép *Num* ký tự. Ví dụ: $St = 'ABCDEFGH'$ thì lệnh $Copy(St, 3, 2) = 'CD'$ và $Copy(St, 4, 10)$ cho ta $'DEF'$.



Ồ Ghi chú:

- Nếu $Pos + Num > Length(St)$ thì hàm sẽ trả về các ký tự trong xâu St .
- Nếu $Pos > Length(St)$ thì hàm $Copy$ sẽ trả về cho ta một xâu rỗng.

c. Concat(St_1, St_2, \dots, St_n): Hàm này dùng để ghép tất cả các xâu ký tự St_1, St_2, \dots, St_n thành một xâu theo thứ tự các đối số cung cấp cho hàm.

Ồ Ghi chú:

- Số lượng đối của hàm $Concat$ phải ≥ 2 .
- Nếu tổng số chiều dài các xâu > 255 thì máy sẽ báo lỗi.
- Có thể dùng phép cộng (+) để ghép xâu ký tự. Ví dụ: $St := Concat(St1, St2 + 'N');$

d. Pos($St1, St2$):

Trong đó: $St1, St2$ là biểu thức xâu ký tự.

Hàm này trả về số nguyên biểu diễn vị trí đầu tiên của $St1$ gặp trong xâu $St2$. Nếu không tìm thấy thì $Pos = 0$.

4 Ví dụ: nếu $St := 'ABCDEFGBCD'$ thì $Pos('DE', St) = 4$, $Pos('BCD', St) = 2$, $Pos('XY', St) = 0$.

4 Ví dụ 1: Viết chương trình nhập vào từ bàn phím một xâu ký tự và in ra màn hình xâu ký tự ngược tương ứng. Ví dụ: nhập *TRUNG TAM CONG NGHE AVNET* máy in ra *TENVA EHGN GNOC MAT GNURT*.

```
Program DaoChuo;
Uses CRT;
Var
  Cau : String[80];
  i : Byte;
Begin
  Wite('Nhap vao mot cau : ');
  Readln(Cau);
  For i := Length(Cau) DownTo 1 do
    Write(Cau[i]);
  Readln;
End.
```




4 Ví dụ 2: Hiển thị chuỗi con trong chuỗi mẹ được nhập từ bàn phím, vị trí và số ký tự hiển thị cũng được nhập từ bàn phím.

```
Program SubString;  
Uses CRT;  
Var  
    St : String;  
    Pos, Len : Byte;  
Begin  
    Writeln('Nhập vào một chuỗi : ');  
    Readln(St);  
    Writeln('Muốn hiển thị xâu từ vị trí nào : ');  
    Readln(Pos);  
    Writeln('Độ dài xâu ký tự con : ');  
    Readln(Len);  
    Write('Xâu ký tự con là : ', Copy(St, Pos, Len));  
    Readln;  
End.
```

4 Ví dụ 3: Viết các hàm chuyển đổi xâu ký tự thành chữ hoa và chữ thường.

```
Function ToUpper(s : String) : String;  
Var i : Byte;  
Begin  
    For i := Length(s) do  
        s[i] := Upcase(s[i]);  
    ToUpper := s;  
End;  
(*****)  
Function ToLower(s : String) : String;  
Var i : Byte;  
Begin  
    For i := Length(s) do  
        If s[i] In ['A'..'Z'] then  
            s[i] := Chr(Ord(s[i]) + 32);  
    ToLower := s;  
End;
```



BÀI 9. DỮ LIỆU KIỂU BẢN GHI VÀ KIỂU TẬP

I. Kiểu bản ghi:

1. Khái niệm và định nghĩa:

Các kiểu cấu trúc dữ liệu như kiểu mảng, tập hợp đều được tạo ra bằng một tập hợp các phần tử có cùng kiểu.

Để tạo ra một kiểu cấu trúc dữ liệu mới với các phần tử dữ liệu có kiểu khác nhau, người ta định nghĩa ra bản ghi (*Record*). *RECORD* là một cấu trúc bao gồm nhiều thành phần. Các thành phần có thể thuộc các kiểu dữ liệu khác nhau và được gọi là các trường (*Field*), mỗi trường đều được đặt tên.

Để mô tả một kiểu *T* có cấu trúc *Record* với danh sách các trường có tên là *S1*, *S2*, ..., *Sn* và có các mô tả kiểu tương ứng là trường có tên là *T1*, *T2*, ... *Tn* ta dùng cách viết như sau:

```
Type  
  T = Record  
    S1 : T1;  
    S2 : T2;  
    ...  
    Sn : Tn;  
  End;
```

Ví dụ: Mô tả thời gian *DATE* có ba trường: Ngày, Tháng, Năm

```
Type  
  Date = Record  
    Ngày: 1..31;  
    Tháng: 1..12;  
    Năm: Word;  
  End;
```

4 Ví dụ: Để mô tả Nhân sự của phòng tổ chức, ta dùng các trường: *HoDem*, *Ten*, *NgaySinh*, *Luong*,... ở đây ta lấy ví dụ có 5 trường:

```
Type  
  NhanSu = Record  
    HoDem: String[20];  
    Ten: String[7];  
    NgaySinh: Date;
```



```
    Luong: Real;
    CoGiaDinh: Boolean;
End;
Var
    NV, NV1: NhanSu;
    DS: Array[1..100] of NhanSu;
    {Danh sach tren la kieu mang mo ta nhan su cua mot co quan co duoi 100
    nhan vien}
```

Ồ Ghi chú: Ta có thể viết trực tiếp mô tả trường NgaySinh nếu như chưa có kiểu Date như sau:

```
Type
    NhanSu = Record
        HoDem: String[20];
        Ten: String[7];
        NgaySinh: Record
            Ngay: 1..31;
            Thang: 1..12;
            Nam: Word;
        End;
        Luong: Real;
        CoGiaDinh: Boolean;
    End;
```

2. Sử dụng Record:

Muốn truy cập một biến kiểu Record, ta phải truy cập theo thành phần của chúng. Cú pháp để truy cập đến một thành phần nào đó là:

<Tên biến Record>. <Tên trường>

4 Ví dụ:

```
NV.HoLot := 'Huynh Dinh';
NV.Ten := 'Can';
NV.NgaySinh.Ngay := 4;
NV.NgaySinh.Thang := 2;
NV.NgaySinh.Nam := 1982;
NV.Luong := 500000;
NV.CoGiaDinh := False;
```



4 Ví dụ 1: Nhập lý lịch nhân viên của một cơ quan.

```
Uses CRT;
Type
  Date = Record
    Ngay: 1..31;
    Thang: 1..12;
    Nam: Word;
  End;
  NhanSu = Record
    HoDem: String[20];
    Ten: String[7];
    NgaySinh: Date;
    Luong: Real;
    CoGiaDinh: Boolean;
  End;
Var
  DS: Array[1..100] of NhanSu;
  i, SoNV: Byte;
  GD: Char;
Begin
  ClrScr;
  Writeln('      NHAP HO SO NHAN VIEN      ');
  Write(' So nhan vien tai co quan: ');
  Readln(SoNV);
  For i:=1 to SoNV do
    Begin
      ClrScr;
      Write(' Ho dem: ');          Readln(DS[i].HoDem);
      Write(' Ho dem: ');          Readln(DS[i].Ten);
      Write(' Ngay sinh: / ');
      GotoXY(14,3);                Readln(DS[i].NgaySinh.Ngay);
      GotoXY(17,3);                Readln(DS[i].NgaySinh.Thang);
      GotoXY(20,3);                Readln(DS[i].NgaySinh.Nam);
      Write(' Luong: ');            Readln(DS[i].Luong);
      Write(' Co gia dinh (Y/N)?: '); Readln(GD);
```



```
If Upcase(GD) = 'Y' then
    DS[i].CoGiaDinh := True
Else
    DS[i].CoGiaDinh := False;
```

```
End;
```

```
Readln;
```

```
End.
```

Ồ Ghi chú:

- Các biến *Record* cùng kiểu có thể gán cho nhau. Ví dụ: *NV := NV1*; thay vì ta phải thực hiện:

```
NV.HoDem := NV1.HoDem;
```

```
NV.Ten := NV1.Ten;
```

.....

- Có thể dùng phép so sánh:

```
If NV = NV1 then Write('Cung mot nhan vien !');
```

Hoặc:

```
If (NV.HoDem = NV1.HoDem) and (NV.Ten = NV1.Ten) then
```

```
Write('Hai nhan vien cung ho ten !.');
```

- Không được dùng các thao tác sau:

+ Các thủ tục đọc và ghi (*Read*, *Readln*, *Write*, *Writeln*) cho cả một biến kiểu *Record* như: *Readln(NV)*, *Writeln(NV)*;

+ Sử dụng các phép toán quan hệ như: *<*, *>*, *<=*, *>=*. Nhưng có thể sử dụng phép toán *<>* và *=* cho hai biến *Record* có cùng kiểu.

+ Tất cả các phép toán số học và logic.

3. Câu lệnh With:

Khi cần truy cập nhiều thành phần của một biến kiểu *Record*, ta có thể dùng câu lệnh *With* để chương trình được gọn hơn.

Cú pháp:

```
WITH <Biến kiểu Record> DO <Câu lệnh>
```

4 Ví dụ 1: Theo như ví dụ 1, ta có thể viết ngắn gọn hơn như sau:

```
Uses CRT;
```

```
Type
```

```
    Date = Record
```

```
        Ngay: 1..31;
```



```
Thang: 1..12;
Nam: Word;
End;
NhanSu = Record
  HoDem: String[20];
  Ten: String[7];
  NgaySinh: Date;
  Luong: Real;
  CoGiaDinh: Boolean;
End;
Var
  DS: Array[1..100] of NhanSu;
  i, SoNV: Byte;
  GD: Char;
Begin
  ClrScr;
  Writeln('      NHAP HO SO NHAN VIEN      ');
  Write(' So nhan vien tai co quan: ');
  Readln(SoNV);
  For i:=1 to SoNV do
    With DS[i] do
      Begin
        ClrScr;
        Write(' Ho dem: ');      Readln(HoDem);
        Write(' Ho dem: ');      Readln(Ten);
        Write(' Ngay sinh: / ');
        With NgaySinh do
          Begin
            GotoXY(14,3);      Readln(Ngay);
            GotoXY(17,3);      Readln(Thang);
            GotoXY(20,3);      Readln(Nam);
          End;
        Write(' Luong: ');      Readln(Luong);
        Write(' Co gia dinh (Y/N) ? : '); Readln(GD);
        If Upcase(GD) = 'Y' then
```



```
        CoGiaDinh := True
    Else
        CoGiaDinh := False;
    End;
Readln;
End.
```

Ồ Ghi chú: Như vậy chúng ta có thể lồng các chỉ thị With ... Do ... vào với nhau để truy nhập vào các trường ở sâu trong Record phức tạp như biến Ds[i]. Cú pháp như sau:

```
With A do
    With B do
        .....
```

Với A, B đều được mô tả là Record song B là một trường của A thì ta có thể có cách viết như sau:

<pre>With A do With B do Begin End;</pre>	→	<pre>With A, B do Begin End;</pre>
---	---	--

4 Ví dụ 2: Đoạn chương trình ở ví dụ 1 có thể viết lại:

```
.....
For i:=1 to SoNV do
    With DS[i], NgaySinh do
        Begin
            ClrScr;
            Write(' Ho dem: ');      Readln(HoDem);
            Write(' Ho dem: ');      Readln(Ten);
            Write(' Ngay sinh: / ');
            GotoXY(14,3);             Readln(Ngay);
            GotoXY(17,3);             Readln(Thang);
            GotoXY(20,3);             Readln(Nam);
            Write(' Luong: ');        Readln(Luong);
            Write(' Co gia dinh (Y/N)?: '); Readln(GD);
            If Upcase(GD) = 'Y' then
```



```
CoGiaDinh := True
Else
CoGiaDinh := False;
End;
.....
```

4. Record có cấu trúc thay đổi:

Các kiểu Record trình bày trên là kiểu Record cố định vì số thành phần cũng như cấu trúc của Record là đã cố định. Bên cạnh đó Pascal còn *cho phép lập các Record có một phần cấu trúc thay đổi được*.

Trước hết, ta xét thí dụ sau: trong mục *NhanSu*, nếu ta xét thêm trường *NgheNghiep* thì sẽ có nhiều trường hợp xảy ra, chẳng hạn:

- *Công nhân* : Cần ghi rõ ngành gì ? Bậc thợ mấy ?
- *Kỹ sư* : Ngành gì ? Trình độ thực tế ?
- *Bác sĩ* : Chuyên khoa gì ?
- *Cá biệt* : Không ghi gì thêm ?

Tuy ta có thể lập một Record gồm đầy đủ các trường kể trên nhưng rất cồng kềnh (trong khi đó có thể một người ở một thời điểm nào đó chỉ có một ngành nghề) và chiếm nhiều ô nhớ.

Tiếp theo ta có thể lập ra bốn kiểu Record giống nhau phần đầu (*HoDem, Ten, NgaySinh, Luong, CoGiaDinh*) nhưng chỉ khác nhau phần cuối là nghề nghiệp (*NgheNghiep*), tức là sẽ có các trường tương ứng với bốn nghề khác nhau. Cách này cũng làm cồng kềnh chương trình vì ta phải dùng đến bốn kiểu Record.

Ngôn ngữ Pascal cho phép lập Record có dạng sau để tiết kiệm ô nhớ và cho phép linh hoạt sử dụng:

Type

```
Nghe = (CongNhan, KySu, BacSi, CaBiet);
Nganh = (KhaiThac, CoKhi, CheBien, Nuoi, KinhTe);
Khoa = (Noi, Ngoai, Nhi, Phu);
NhanSu = Record
HoDem: String[20];
Ten: String[7];
NgaySinh: Date;
Luong: Real;
```




```
CoGiaDinh: Boolean;
CASE NgheNghiep: Nghe Of
    CongNhan: (NganhCN: Nganh; BacTho: Byte);
    KySu: (NganhKS: Nganh; TrinhDoTT: (Kem, TB, kha, Gioi));
    BacSi: (ChuyenKhoa: Khoa);
    CaBiet: ();
END; { Of Record }
Var NV, NV1: NhanSu;
Begin
    ...
    With NV do
        Begin
            HoDem := 'Vo Thanh';
            Ten := 'Chau';
            NgheNghiep := CongNhan;
            NganhCN := CoKhi;
            BacTho := 3;
        End;
    ...
    With NV1 do
        Begin
            HoDem := 'Huynh Dinh';
            Ten := 'Can';
            NgheNghiep := KySu;
            NganhKS := KinhTe;
            TrinhDoTT := Kha;
        End;
    ...
END.
```

F Giải thích minh hoạ trên:

- *HoDem, Ten, NgaySinh, CoGiaDinh* là các thành phần cố định của *Record NhanSu*.
- *NganhCN, NganhKS, BacTho, TrinhDoTT, ChuyenKhoa* là các thành phần thay đổi của *Record NhanSu*.



- Trong khai báo một kiểu Record, nếu có thành phần thay đổi thì *phải được đặt sau các thành phần cố định và chỉ được phép có một trường thay đổi*.

- Phần thay đổi nằm sau cùng trong danh sách và được bắt đầu bằng câu lệnh *CASE*. (Phần thay đổi này lại có thể chứa Record khác có kiểu cấu trúc thay đổi).

Ồ Ghi chú:

- Phần thay đổi là một trường gọi là trường đánh dấu (*Tag Field*) và được đặt trong câu lệnh *CASE* (Ví dụ trên là *NgheNghiep*). Ứng với mỗi giá trị của trường đánh dấu, ta có các biến dạng của Record với danh sách các trường tương ứng được đặt sau các nhãn của lệnh *CASE* và toàn bộ danh sách này phải được đặt trong hai dấu ngoặc đơn *()* ngay cả khi nó rỗng như trường hợp *CaBiet* ở ví dụ trên.

- Trường mô tả phải là các kiểu đơn giản (*Byte, Integer, Word, LongInt, Real, Double, Char, Boolean*).

- Tất cả các tên biến trong phần thay đổi đều bắt buộc phải khác nhau. Theo ví dụ trên, *Nganh* trong hai trường hợp của *NgheNghiep* là *CongNhan* và *KySu* được ký hiệu bằng hai tên khác nhau là: *NganhCN* và *NganhKS*.



BÀI 10. DỮ LIỆU KIỂU TẬP

I. Khái niệm:

Khi giải các bài toán có nhiều và cần sử dụng nhiều lần về sau thì ta phải tổ chức dữ liệu lưu trữ trên đĩa (*dữ liệu kiểu tệp*). Khi kết thúc chương trình hoặc tắt máy thì dữ liệu kiểu tệp vẫn tồn tại trên đĩa.

Định nghĩa một kiểu tệp *T* với các phần tử có kiểu *KPT* (*Kiểu phần tử*) được viết trong phần mô tả kiểu với từ khóa *File Of* như sau:

TYPE

T = FILE OF KPT;

4 Ví dụ:

Type

FileReal = File of Real;

Date = record

Ngày: 1..31;

Thang: 1..12;

Nam: Word;

End;

NhanSu = Record

MaSo: Word;

HoDem: String[20];

Ten: String[7];

NgàySinh: Date;

Luong: Real;

End;

FnhanSu = File Of NhanSu;

Var

F1: FileReal;

F2: FNhanSu;

Ồ Ghi chú:

- Kiểu phần tử của tệp có thể là bất kỳ kiểu dữ liệu nào ngoại trừ kiểu tệp.
- Biến tệp được khai báo bằng cách sử dụng một kiểu tệp đã được định nghĩa trước đó hoặc khai báo trực tiếp với mô tả kiểu. Ví dụ:

Var



F3: File Of Char;

F4: File Of Array[1..5] Of Integer;

- Biến tệp là một biến thuộc kiểu dữ liệu tệp. Một biến kiểu tệp đại diện cho một tệp. Việc truy cập dữ liệu ở một tệp được thể hiện qua các thao tác với thông số là biến tệp đại diện.

II. Cấu trúc và phân loại tệp:

Các phần tử của một *Array (Mảng)* hoặc *Record* có thể truy cập được tùy ý (*Random Access*) thông qua tên biến, chỉ số hoặc tên trường. Các phần tử của tệp không có tên và việc truy cập không thể tùy tiện được. Các phần tử của tệp được sắp xếp thành một dãy và ở mỗi thời điểm chương trình chỉ có thể truy nhập vào một phần tử của tệp thông qua giá trị của biến đệm (*Tampon Variable*). Biến đệm dùng để đánh dấu vị trí truy nhập hay còn gọi là cửa sổ của tệp. Ta có thể hình dung một tệp như là một cuộn phim chụp ảnh. Mỗi một ảnh là một phần tử và ống kính là cửa sổ để nhìn vào nên tại mỗi thời điểm chỉ nhìn thấy một ảnh. Sau mỗi lần chụp, cửa sổ sẽ nhìn vào ảnh ở vị trí kế tiếp.

Ta có thể dùng lệnh làm dịch chuyển cửa sổ sang vị trí tiếp theo hoặc về vị trí đầu tệp. Mỗi tệp đều được kết thúc bằng dấu hiệu đặc biệt để báo hiệu hết tệp, hay gọi là *EOF(F) (End Of File F)*. Pascal có một hàm chuẩn *EOF* trả về giá trị kiểu Boolean với tham số là biến tệp để xem cửa sổ đã đặt vào vị trí kết thúc tệp đó chưa. Nếu chưa đến cuối tệp thì hàm *EOF* trả về giá trị *False*.

Việc phân loại tệp dựa trên việc bố trí các phần tử của tệp trong bộ nhớ ngoài và cách truy cập vào tệp: Tệp truy nhập tuần tự (*Sequential Access*) hoặc tệp truy nhập trực tiếp (*Direct Access*).

Đối với tệp truy nhập tuần tự việc đọc một phần tử bất kỳ của tệp phải đi qua các phần tử trước đó; muốn thêm một phần tử vào tệp, phải đặt cửa sổ vào vị trí cuối tệp. Bộ nhớ ngoài tương ứng với cấu trúc này là băng từ. Tệp truy nhập tuần tự đơn giản trong việc tạo lập hay xử lý nhưng kém tính linh hoạt.

Đối với tệp truy nhập trực tiếp, ta có thể đặt cửa sổ vào một vị trí bất kỳ của tệp. Bộ nhớ ngoài điển hình là đĩa từ (*do đầu từ khi đọc có thể được điều khiển đặt vào một chỗ bất kỳ trên đĩa tại mọi thời điểm*).

Tệp truy nhập trực tiếp chỉ được định nghĩa ở Turbo Pascal, Pascal chuẩn không có. Khi không nói rõ là tệp loại gì thì đó được mặc định là tệp truy nhập tuần tự.

III. Các thao tác trên tệp:



1. Mở tệp mới để cất dữ liệu:

Chương trình chỉ có thể lưu lại dữ liệu vào một tệp sau khi ta làm thủ tục mở tệp. Việc mở tệp được tiến hành với hai thủ tục đi liền nhau theo thứ tự:

Assign(FileVar, FileName)

ReWrite(FileVar);

Trong đó:

- FileVar:

- FileName: tên của tệp đặt trong thiết bị nhớ ngoài được đưa vào dạng một *String* (quy tắc đặt tên tương tự hệ điều hành). Ta nên đặt tên sao cho tên đó phản ánh được ý nghĩa hay bản chất, nội dung của tệp.

4 Ví dụ:

Assign(F1, 'HoSo.txt'); {Gán tên là HoSo.txt cho biến F1}

ReWrite(F1); {Mở tệp HoSo.txt, tệp chưa có phần tử nào}

Sau khi mở tệp xong, tệp sẽ rỗng vì chưa có phần tử nào, cửa sổ của tệp sẽ không có giá trị xác định vì nó trở vào cuối tệp (EOF).

Ồ Ghi chú: Khi mở tệp, nếu trên bộ nhớ ngoài (cùng đường dẫn) đã có sẵn tệp có tên trùng với tên tệp được mở thì nội dung cũ sẽ bị xóa.

2. Ghi các giá trị vào tệp với thủ tục Write:

Thủ tục Write sẽ đặt các giá trị mới vào tệp.

Cú pháp:

Write(FileVar, Item1, Item2, ..., ItemN);

Trong đó: *Item1, Item2, ..., ItemN*: là các giá trị cần ghi vào tệp.

4 Ví dụ: Ta cần ghi vào tệp *ChuCai.txt* các giá trị 'a'..'z', thực hiện như sau:

...

Assign(F1, 'ChuCai.txt');

ReWrite(F1);

For ch:= 'a' to 'z' do

Write(F1, ch);

...

4 Ví dụ 1: Tạo một tệp chứa các số nguyên từ 1 đến 100 với tên tệp trên đĩa là *"Nguyen.txt"*.



```
Program TaoTepSoNguyen;  
  Var i: Integer;  
      F: File of Integer;  
Begin  
  Assign(F,'Nguyen.txt');  
  ReWrite(F);  
  For i:= 1 to 100 do  
    Write(F,i);  
  Close(F);  
End.
```

ⓘ **Ghi chú:** Một tệp có thể được dùng làm tham số của chương trình con với lời khai báo bắt buộc phải sau chữ *Var* tức là tệp được dùng làm tham số biến.

3. Đọc dữ liệu từ một tệp đã có:

Đối với tệp tuần tự, ta không thể vừa ghi vừa đọc được cùng một lúc. Sau khi ghi dữ liệu vào tệp và đóng lại, ta có thể đọc lại các giá trị dữ liệu trong tệp.

Một chương trình muốn sử dụng các dữ liệu đã được chứa trong một tệp, đầu tiên phải mở tệp đó ra để đọc, thủ tục sau nhằm mở một tệp để đọc:

Cú pháp:

```
Assign(FileVar, FileName);  
Reset(FileVar);
```

Sau lệnh *Reset*, nếu tệp không rỗng thì của sổ tệp bao giờ cũng trở vào phần tử đầu tiên của tệp và chương trình sẽ sao chép phần tử của tệp được trở sang biến đệm của sổ. Nội dung tệp này không bị xóa. Nếu ta mở một tệp chưa tồn tại trên đĩa thì sẽ có lỗi.

Để đọc dữ liệu từ tệp, ta dùng thủ tục *READ* dạng sau:

```
Read(FileVar, Var1, Var2,..., VarN);
```

Trong đó: *Var1, Var2,..., VarN* là các biến có cùng kiểu thành phần của *FileVar*. Gặp lệnh này máy sẽ đọc các giá trị tại vị trí của sổ đang trở (*nếu có*) gán sang biến tương ứng cùng kiểu. Sau đó, của sổ dịch chuyển sang vị trí tiếp theo và đọc giá trị cho biến khác, cứ thế đọc cho đến biến *VarN*. *READ* chỉ có thể đọc giá trị của tệp để gán giá trị cho các biến.



Việc đọc một phần tử của tệp cần thỏa mãn điều kiện: phần tử đó không phải là phần tử cuối tệp tức là *EOF*. Do đó, trước khi muốn đọc tệp và gán cho biến *X*, cần phải thử xem tệp đó đã kết thúc chưa bằng câu lệnh:

```
If Not EOF(FileVar) Then Read(FileVar, X);
```

Hoặc nếu muốn đọc tất cả các phần tử của tệp:

```
While Not EOF(FileVar) Do
```

```
Begin
```

```
Read(FileVar, X);
```

```
Xử lý biến x nếu cần;
```

```
...
```

```
End;
```

Thực hiện xong ta phải đóng tệp với thủ tục sau:

```
Close(FileVar);
```

4 Ví dụ1: Giả sử đã tồn tại một tệp có tên là *Nguyen.txt* chứa các số kiểu Byte và có ít nhất ba phần tử. Thực hiện đọc ra giá trị thứ nhất và thứ ba của tệp và gán cho hai biến *A*, *B* tương ứng.

```
Program DocSo;
```

```
Var A, B: Byte;
```

```
F: File Of Byte;
```

```
Begin
```

```
Assign(F, 'Nguyen.txt');
```

```
Reset(F);
```

```
Read(F, A); {đọc một phần tử thứ nhất của tệp ra biến A}
```

```
Read(F, B); {đọc một phần tử thứ hai của tệp ra biến B}
```

```
Read(F, B); {đọc một phần tử thứ hai của tệp ra biến B}  
{lúc này B không giữ giá trị thứ hai nữa}
```

```
Close(F);
```

```
End.
```

Vì đây là tệp có cấu trúc tuần tự nên muốn đọc phần tử thứ ba ta buộc phải đọc qua phần tử thứ hai.

Ba lần *Read(F,...)* ở trên có thể thay thế bằng một lệnh đọc duy nhất:

```
Read(F, A, B, B);
```



4 Ví dụ 2: Đọc tất cả các phần tử của một tệp chứa các số có Integer nào đó và ghi ra màn hình giá trị các số đó và cuối cùng ghi ra số phần tử của tệp.

```
Program DocTepSo;
Uses CRT;
Var i, SoPT: Integer;
    F: File Of Byte;
    FileName: String;
Begin
  ClrScr;
  Write('Tep can doc la gi ? (Tep so nguyen):');
  Readln(FileName);
  Assign(F, FileName);
  Reset(F);
  SoPT:= 0;
  While Not EOF(F) Do
    Begin
      Read(F,i); {doc mot phan tu cua tep ra bien i}
      Write(i, ' ');
      Inc(SoPT); {dem so phan tu}
    End;
  Close(F);
  Writeln;
  Write('So phan tu cua tep ',FileName,' la ',SoPT);
  Readln
End.
```

4. Tệp truy nhập trực tiếp:

Pascal chuẩn chỉ định nghĩa một kiểu tệp truy nhập tuần tự. Tuy nhiên các bộ nhớ ngoài như đĩa từ,... có thể cho phép tính toán tọa độ của một phần tử bất kỳ trong tệp (vì độ dài của các phần tử là như nhau), do đó có thể truy nhập trực tiếp vào một phần tử của tệp mặc dù cấu tạo logic của tệp vẫn là dạng tuần tự. Trong *Turbo Pascal*, để truy nhập trực tiếp vào phần tử của tệp, sử dụng thủ tục ***SEEK***.

Cú pháp:

Seek(FileVar, No);



Trong đó, *No* là số thứ tự của phần tử trong tệp (*phần tử đầu tiên của tệp được đánh số 0*). Gặp thủ tục này, chương trình sẽ đặt cửa sổ của tệp vào phần tử thứ *No*. Tiếp theo muốn đọc phần tử đó ra thì dùng *Read*, nếu muốn đặt giá trị mới vào dùng *Write*.

4 Ví dụ: Giả sử tệp *Nguyen.txt* trên đĩa ở thư mục hiện hành đã chứa 100 số nguyên từ 1 đến 100. Ta kiểm tra xem phần tử thứ hai (*đếm từ 0*) của tệp có giá trị bằng 3 không, nếu không thì sửa lại bằng một giá trị nhập từ bàn phím.

```
Var
  i: Byte;
  F: File Of Byte;
  Answer: Char;
Begin
  Assign(F,'Nguyen.txt');
  Reset(F);
  Seek(F,2); { Dat cua so tep vao vi tri thu 3 }
  Read(F,i);
  Writeln('i = ',i);
  Write('Ban muon sua lai khong?(C/K):');
  Readln(Answer);
  If Answer In['c','C'] Then
    Begin
      Seek(F,2);
      Write(' Ban muon sua lai bang bao nhieu?');
      Readln(i);
      Write(F,i); { Thay doi gia tri cua phan tu hien tai }
    End;
  Close(F);
  Readln
End.
```

5. Các thủ tục và hàm xử lý tệp của Turbo Pascal:

a. Hàm *FileSize(FileVar)*: Hàm cho giá trị biểu thị số phần tử của tệp *FileVar*. Hàm nhận giá trị 0 khi tệp rỗng.

b. Hàm *FilePos(FileVar)*: cho biết vị trí hiện tại của con trỏ (*cửa sổ*) tệp *FileVar*.



Một tệp đã tồn tại chỉ có thể lớn thêm bằng cách ghi thêm các phần tử mới vào vị trí cuối cùng của tệp. Muốn đưa con trỏ đến vị trí cuối cùng của tệp ta thực hiện lệnh sau:

Seek(FileVar, FileSize(FileVar));

c. Thủ tục Erase(FileVar): Dùng để xóa tệp trên đĩa có tên ấn định với FileVar.

⦿ **Chú ý:** Không được xóa tệp đang mở.

4 Ví dụ:

```
...  
Write('Cho biet ten tep can xoa:');  
Readln(FileName);  
Assign(F, FileName);  
Erase(F);  
...
```

d. Thủ tục Rename(FileVar, Str): Dùng để thay đổi tên tệp với tên mới bằng biến *Str* kiểu *String*.

⦿ **Ghi chú:** - Tên mới phải không trùng tên tệp nào có sẵn trên đĩa đang làm việc
- Không được đổi tên tệp đang mở.

4 Ví dụ: Muốn đổi tên tệp File1.dat thành File2.dat, thực hiện như sau:

```
Assign(F, 'File1.dat');  
Rename(F, 'File2.dat');
```

6. Tệp văn bản (Text Files):

Trong Pascal có một kiểu tệp đã được định nghĩa trước, đó là tệp văn bản được định nghĩa với tên chuẩn Text.

Cú pháp khai báo:

F1, F2 :Text;

Thành phần cơ sở của tệp kiểu Text là ký tự. Tuy nhiên, văn bản có thể được cấu trúc thành các dòng, *mỗi dòng được kết thúc bởi dấu hiệu EOLN (End Of Line)*. Như vậy, muốn đọc và in ra từng dòng của tệp văn bản thì sử dụng dạng Text.

Tệp văn bản được kết thúc bởi dấu End Of File, cụ thể với Turbo Pascal là *Ctrl-Z* (^Z) có mã ASCII = 26.



a. Hàm EOF(Var F: Text): Boolean. Hàm trả về giá trị False khi cửa sổ tệp chưa đến cuối tệp, ngược lại, cho giá trị True. Hàm này thường sử dụng để kiểm tra xem đã đọc hết tệp văn bản chưa. Ví dụ:

While not EOF(F) Do...

b. Hàm EOLN(Var F: Text): Boolean. Hàm trả về giá trị False khi cửa sổ tệp chưa đến điểm cuối dòng hoặc cuối tệp, ngược lại, cho giá trị True. Hàm này thường sử dụng để kiểm tra xem đã đọc đến cuối dòng chưa. Ví dụ:

While not EOLN(F) Do..

c. Ghi vào một tệp văn bản: Ta có thể ghi các giá trị kiểu *Integer*, *Real*, *Boolean*, *String* vào tệp văn bản bằng lệnh *Write* hoặc *Writeln*. Có ba dạng viết:

Write(FileVar, Item1, Item2,...,ItemN); (1)

Writeln(FileVar, Item1, Item2,...,ItemN); (2)

Write(FileVar); (3)

F Lệnh (1): Viết các giá trị *Item1, Item2,...,ItemN* là các hằng, biểu thức hay biến có kiểu đơn giản như: *Nguyên*, *Thực*, *Ký tự*, *Chuỗi*, *Logic* vào biến tệp *FileVar*.

F Lệnh (2): Tương tự như (1) nhưng có thêm dấu hiệu hết dòng vào tệp sau khi đã viết hết các giá trị *Item1, Item2,...,ItemN*.

F Lệnh (3): chỉ thực hiện việc đưa thêm dấu hiệu hết dòng vào tệp.

Ồ Ghi chú: Từ câu lệnh (2) ta có thể chuyển sang viết như sau:

Begin

Write(FileVar, Item1);

...

Write(FileVar, Item2);

Writeln(FileVar);

End;

4 Ví dụ: Thực hiện ghi vào một tệp các thông tin sau:

Chao cac ban den voi ngon ngu lap trinh Pascal

Trung tam Cong nghe Avnet

Var F: Text;

Begin



```
Assign(F,'VanBan.txt');
Rewrite(F);
Writeln(F,'Chao cac ban den voi ngon ngu lap trinh Pascal');
Writeln(F,'          Trung tam Cong nghe Avnet          ');
Writeln(F,'          -----          ');
Writeln(F);
Close(F);
End.
```

Ồ Ghi chú: Trong lệnh Writeln, Write ta có thể hiển thị có quy cách như đã trình bày trước đây.

d. Đọc dữ liệu từ tệp văn bản:

Ta có thể đọc không những các ký tự từ tệp văn bản mà còn có thể đọc lại các số nguyên, thực, logic từ tệp văn bản thông qua các thủ tục:

Read(FileVar, Var1, Var2,..., VarN); (1)

Readln(FileVar, Var1, Var2,..., VarN); (2)

Readln(FileVar); (3)

Trong đó, *Var1, Var2,..., VarN* là các biến thuộc kiểu ký tự, nguyên, thực, logic, chuỗi. *Lệnh (1)* sẽ đọc nội dung một hay nhiều phần tử mà không chuyển của sổ tệp xuống dòng. *Lệnh (2)* đọc như lệnh (1) nhưng sẽ di chuyển của sổ tệp sang đầu dòng tiếp theo sau khi đã lần lượt đọc các biến tương ứng. *Lệnh (3)* đưa của sổ tệp sang đầu dòng tiếp theo mà không đọc gì cả.

4 Ví dụ: Chương trình sau đọc và in ra nội dung tệp văn bản *VanBan.txt* đã được tạo ra từ chương trình trên.

```
Program Doc_File_Text;
Uses CRT;
Var
  F: Text;
  Line:String[80];
Begin
  ClrScr;
  Assign(F,'VanBan.txt');
  Reset(F);
  While Not EOF(F) Do
    Begin
```



```
    Readln(F, Line);
    Writeln(Line);
End;
Close(F);
Readln;
End.
```

e. Thủ tục thêm dòng:

Cú pháp:

Append(Var F: Text);

Lệnh Append mở tệp văn bản để ghi bổ sung các dòng, định vị của sổ tệp vào cuối tệp. Lần sử dụng kế tiếp với thủ tục Write hay Writeln sẽ thêm văn bản vào cuối tệp.

4 Ví dụ: Chương trình sau đây thêm hai dòng vào cuối tệp *VanBan.txt*.

```
Var F: Text;
Begin
    Assign(F, 'Vanban.txt');
    Append(F);
    Writeln(F, 'Day la dong thu nhat them vao.');
```

```
    Writeln(F, 'Day la dong thu hai them vao.');
```

```
    Close(F);
```

```
End.
```

_____ o² o _____



PHẦN BÀI TẬP THỰC HÀNH

: 1. **Luyện tập căn bản:** Khởi động chương trình Pascal và thực hiện:

1.1. Viết chương trình hiển thị lên màn hình nội dung sau :

```
* * * * *
*           Trung tam Cong nghe AVnet           *
*           74 - Tran Quoc Toan                   *
* * * * *
```

1.2. Viết chương trình hiển thị lên màn hình tam giác sau :

```
  *
 * * *
* * * * *
* * * * * * *
```

1.3. Viết chương trình hiển thị lên màn hình các biểu thức sau :

- a. $5000 + 100 + 200$
- b. $645 + 350 - 345$
- c. $45 + 45 - 32$

1.4. Viết chương trình để tính kết quả các biểu thức sau :

- a. $5000 + 100 + 200$
- b. $645 + 350 - 345$
- c. $45 + 45 - 32$

1.5. Chạy thử chương trình sau để tự rút ra nhận xét :

```
Program BieuThuc;
Begin
  Write ( ' 45 + 756 + 16 = ' );
  Writeln (45 + 756 + 16 );
  Write ( ' 36 - 56 + 3 = ' );
  Writeln ( 36 - 56 + 3 );
  Readln;
End.
```

: 2. Bài tập đơn giản làm quen với các kiểu dữ liệu và một số hàm chuẩn của Pascal



2.1. Tìm chỗ sai trong chương trình sau:

```
Var i, n : Integer;  
b : Byte;  
Begin  
  n := 3;  
  b := 278;  
  i := b + n;  
  Writeln(i);  
End.
```

2.2. Viết chương trình nhập giá trị cho các biến từ bàn phím với kiểu của các biến là các kiểu dữ liệu đã được học, sau đó hiển thị mỗi giá trị của mỗi biến trên một dòng.

2.3. Viết chương trình đọc ký tự từ bàn phím, sau đó cho biết mã số của ký tự vừa nhập trong bảng mã *ASCII*.

2.4. Viết chương trình tính $\sin(x)$, $\cos(x)$. Trong đó, góc x được nhập từ bàn phím và được đo theo đơn vị Radian. (ta có thể chuyển đổi bằng cách: $\cos(x * \pi / 180)$)

2.5. Viết chương trình có sử dụng các hàm chuẩn của Turbo Pascal để tính giá trị:

- bình phương
- trị tuyệt đối
- căn bậc hai
- logarit cơ số e ($e = 2.718$)
- hàm e mũ x (e^x)
- sau khi cắt bỏ phần thập phân
- làm tròn số

của x . Trong đó, x là một giá trị kiểu thực được nhập từ bàn phím.

2.6. Viết chương trình cho biết giá trị đứng trước, đứng sau của một giá trị x kiểu ký tự (*Char*) và y kiểu logic (*Boolean*), trong đó, x và y được nhập từ bàn phím.

: 3. Áp dụng các lệnh đơn giản

3.1. Chương trình sau cho kết quả gì?

```
Begin  
  Writeln( False > True : 60 );
```



```
Writeln( '1' > '2' );  
Readln;  
End.
```

Thử lại trên máy để kiểm tra.

3.2. Cho biết kết quả và kiểu dữ liệu của các biểu thức sau:

- a) $5 + 3.0$
- b) $6/3 + 2 \text{ div } 3$
- c) $(10 \leq 3) \text{ And } (\text{Not True And } (12 \text{ div } 3 \leq 1))$
- d) $(10 * ((45 \bmod 3) + 1)) / 6$



Sau đó, viết chương trình thực hiện các phép tính trên (*hiển thị kết quả ở dạng có định quy cách*).

3.3. Viết chương trình tạo ra một thiệp mời dự sinh nhật. Trong đó, các giá trị lấy từ bàn phím gồm: *Họ tên người được mời, Ngày tổ chức tiệc, Địa điểm, Họ tên người mời*.

3.4. Viết chương trình tính tổng các chữ số của một số có 2 chữ số (*Hướng dẫn: sử dụng phép chia Div và Mod*).

3.5. Áp dụng phương pháp trên, viết chương trình tính tổng các chữ số của một số có 3, 4 chữ số.

3.6. Viết chương trình đổi một số nguyên được lấy từ bàn phím biểu diễn số giây thành giờ, phút, giây và hiển thị ở dạng *giờ : phút : giây*

3.7. Trong môi trường Turbo Pascal, để tạo ký tự  chỉ cần ấn *Alt - 219* (các số 2, 1, 9 và gõ tại khu vực phím số). Viết chương trình in lên màn hình từ **DA NANG** bằng ký tự 

3.8. Viết chương trình tính giá trị biểu thức sau:

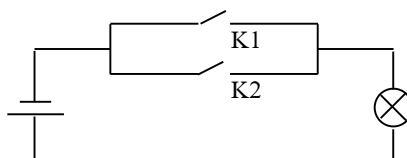
$$\frac{x^3 + \sin(b) - e^{0.0002345}}{5 + e^b + c(0.256 + x) - \sqrt{x + b}}$$

: 4. Bài tập cho các loại lệnh có cấu trúc

4.1. Bài tập cho cấu trúc lệnh If:

a. Viết chương trình để giải phương trình bậc hai $ax^2 + bx + c = 0$.

b. Viết chương trình mô tả sự hoạt động của mạch điện (*hình dưới*) khi có hai công tắc mắc song song với nhau, tức là cho biết *trạng thái sáng hay tối* của bóng đèn khi hai công tắc đóng hoặc ngắt. (*Hướng dẫn: Sử dụng các biến logic với phép toán OR*).



c. Nhập 3 số a, b, c tương ứng với 3 cạnh của một tam giác. Tính diện tích hình tam giác theo công thức:

$$s := \sqrt{p(p-a)(p-b)(p-c)}$$

d. Tính tiền thực lĩnh cho mỗi nhân viên trong xí nghiệp x theo công thức sau:

$$\text{Thực lĩnh} = \frac{(\text{Lương chính} * \text{Số ngày công})}{26} + (\text{Phụ cấp} - \text{Tạm ứng})$$

Với quy định: nghỉ quá 5 ngày sẽ bị trừ 20% tổng thực lĩnh, làm thêm quá 3 ngày được tăng 10% tổng thực lĩnh.

4.2. Bài tập cho cấu trúc lệnh Case:

a. Viết chương trình nhập một ký tự từ bàn phím, kiểm tra nó và:

- hiển thị *la so* nếu nó là số.
- hiển thị *la chu hoa* nếu nó là chữ hoa.
- hiển thị *la chu thuong* nếu nó là chữ thường.
- ngoài ra, hiển thị *Khong phai la so hoac chu cai*.

b. Viết chương trình đổi năm dương lịch (*dạng số*) thành năm âm lịch (*dạng chữ*).

Ví dụ: nhập năm 2000 dương lịch máy cho biết năm âm lịch là *Canh Thìn*. (**Hướng dẫn:** sử dụng phép MOD giữa *năm* với 10 để lấy phần *Địa Can* và MOD giữa *năm* với 12 để lấy phần *Địa Chi*, số dư tương ứng sẽ được kết quả theo bảng sau:

Số dư (MOD 10)	Địa Can	Số dư (MOD 12)	Địa Chi
0.....	Canh	0.....	Thân
1.....	Tân	1.....	Dậu
2.....	Nhâm	2.....	Tuất
3.....	Quý	3.....	Hợi



4..... *Giáp*
 5..... *Ất*
 6..... *Bính*
 7..... *Đinh*
 8..... *Mậu*
 9..... *Kỷ*

4..... *Tý*
 5..... *Sửu*
 6..... *Dần*
 7..... *Mẹo*
 8..... *Thìn*
 9..... *Ty*
 10..... *Ngọ*
 11..... *Mùi*

c. Giải phương trình bậc hai $ax^2 + bx + c = 0$.

4.3. Bài tập cho cấu trúc vòng lặp For:

a. Viết chương trình nhập một số tự nhiên N từ bàn phím và tính:

$$e = 1 + 1/1! + 1/2! + \dots + 1/N!$$

b. Giải bài toán dân gian sau:

Vừa gà vừa chó.

Bó lại cho tròn.

Đếm đủ 100 chân.

Hỏi có mấy gà, mấy chó ?

c. Viết chương trình kiểm tra công thức sau đúng hay sai với mọi N dương được nhập từ bàn phím:

$$1 + 2 + 3 + \dots + N = N(N+1)/2$$

d. Viết chương trình nhập vào chiều dài, chiều rộng của hình chữ nhật và in hình chữ nhật đó ra màn hình bằng các dấu *. Ví dụ: nhập *dài = 7, rộng = 3*, hình chữ nhật sẽ có dạng sau:

```
*****
*       *
*****
```

e. Viết chương trình tính $n!$ trong đó, n là một số nguyên được nhập từ bàn phím (Hướng dẫn: ta nên khai báo biến để chứa kết quả là một biến kiểu **LongInt**).

4.4. Bài tập cho cấu trúc vòng lặp Repeat:

a. Viết chương trình làm các công việc sau: Tính diện tích hình chữ nhật, diện tích hình tam giác, hình tròn. Dùng lệnh **Repeat... Until** để lập một menu lựa chọn công việc theo mẫu:



TÍNH DIỆN TÍCH CÁC HÌNH

- 1. Hình chu nhật.
- 2. Hình tam giác.
- 3. Hình tròn..
- 4. Kết thúc.

Lựa chọn một mục của menu bằng cách ấn số tương ứng, ấn phím số 4 máy dừng chương trình.

b. Viết chương trình nhập vào từ bàn phím lần lượt các số nguyên, dấu hiệu chấm dứt là số 0. Tính tổng và trung bình cộng của các số đã nhập.

c. Viết chương trình in ra bảng tính căn bậc hai của một trăm số nguyên dương đầu tiên.

4.5. Bài tập cho cấu trúc vòng lặp While:

a. Viết chương trình nhập vào từ bàn phím lần lượt các số nguyên, dấu hiệu chấm dứt là số 0. Tính tổng và trung bình cộng của các số đã nhập.

b. Viết chương trình tìm và hiển thị các số nguyên tố nhỏ hơn một số n được nhập từ bàn phím (*Số nguyên tố là số chỉ chia hết cho 1 và chính nó*).

c. Viết chương trình giả làm trò chơi xổ số như sau: Người chơi nhập 5 lần, mỗi lần một số nguyên tùy ý, máy kiểm tra nếu trong các số người chơi nhập vào có 3 số trở lên trùng với các số máy lấy ngẫu nhiên thì người đó thắng và ngược lại là thua. Nếu thua thì máy báo *Bạn đã thua !* ngược lại máy báo *Bạn đã thắng !*

d. Viết chương trình nhập vào một ký tự *ch* bất kỳ, nếu nó là chữ số thì báo *ch là chu so*, nếu nó là chữ cái thì báo *ch là chu cai*, ngoài ra, báo *ch không phải là so hoac chu cai* và thoát khỏi chương trình.

: 5. Bài tập cho dữ liệu kiểu đoạn con, liệt kê và kiểu mảng (Bài 5 và 6)

5.1. Viết chương trình nhập vào một dãy n số $a[1], a[2], \dots, a[n]$ và in ra màn hình các thông tin sau:

- Tổng các phần tử của dãy.
- Số lượng số dương và tổng của các số dương của dãy.
- Số lượng số âm và tổng của các số âm của dãy.
- Trung bình cộng của dãy.



5.2. Viết chương trình nhập vào một dãy n số $a[1], a[2], \dots, a[n]$ và in ra màn hình các thông tin sau:

- Số hạng dương lớn nhất của dãy và chỉ số (*vị trí*) của nó.
- Số hạng dương nhỏ nhất của dãy và chỉ số (*vị trí*) của nó.
- Số hạng âm lớn nhất của dãy và chỉ số (*vị trí*) của nó.
- Số hạng âm nhỏ nhất của dãy và chỉ số (*vị trí*) của nó.

5.3. Viết chương trình nhập vào mảng a gồm 10 phần tử nguyên, sau đó, nhập một giá trị x . Tìm trong mảng a nếu có phần tử nào *có giá trị bằng với x* thì hiển thị lên màn hình *vị trí* của nó trong mảng a .

5.4. Viết chương trình nhập vào một ma trận vuông, xuất màn ra màn hình ma trận đó và cho biết tổng các phần tử trên đường chéo chính.

: 7. Bài tập tạo thủ tục và hàm (Bài 7):

7.1. Viết một thủ tục dùng để vẽ hình vuông bằng dấu *. Chiều dài của cạnh hình vuông được nhập từ bàn phím. Gọi thực hiện thủ tục bởi chương trình chính.

7.2. Lập ba *thủ tục* tính diện tích hình tam giác, hình chữ nhật và hình tròn.

7.3. Lập ba *hàm* tính diện tích hình tam giác, hình chữ nhật và hình tròn.

7.4. Lập một hàm để kiểm tra một số có phải là số nguyên tố hay không. Sau đó, cho chương trình chạy liên tục và hỏi người dùng: *Bạn có tiếp tục không ?* cho đến khi người dùng nhập ký tự k hoặc K thì dừng lại.

7.5. Viết hàm để tính giá trị a^n . Trong đó, a và n là hai giá trị kiểu thực. (*Hướng dẫn: $a^n = e^{n * \ln(a)}$*).

7.6. Viết một hàm để tính giá trị $n!$.

: 8. Bài tập cho phần xử lý chuỗi (Bài 8):

8.1. Viết chương trình nhập vào một chuỗi và đếm trong chuỗi đó có bao nhiêu ký tự 'a', 'b' và 'c' (*kể cả 'A', 'B', 'C'*).

8.2. Viết chương trình đếm trong một chuỗi được nhập từ bàn phím có bao nhiêu từ, giả sử mỗi từ cách nhau bằng một ký tự trắng (*tạm chấp nhận giữa hai từ không được nhập quá 1 ký tự trắng*).

8.3. Viết chương trình nhập vào một chuỗi s , sau đó, nhập vào một từ bất kỳ và kiểm tra trong chuỗi s nếu có từ đó thì xóa đi (*tại vị trí đầu tiên*), nếu không tìm thấy từ đó trong s thì báo *Khong co tu nay trong chuoai vua nhap !*



8.4. Tương tự câu trên (7.3) nhưng nếu tìm thấy trong chuỗi s có bao nhiêu từ đó thì xoá hết.

8.5. Viết chương trình nhập vào từ bàn phím Họ và tên Việt Nam, sau đó in phần tên ra màn hình. Ví dụ: nhập *Phan Van Anh Tuan* thì in ra *Tuan*.

: BÀI TẬP TỔNG QUÁT

1. Tìm tất cả các số có 3 chữ số a, b, c sao cho tổng các lập phương của các chữ số bằng chính số đó.

$$abc = 100a + 10b + c = a^3 + b^3 + c^3$$

2. Tìm và in ra các số nguyên tố nhỏ hơn một số cho trước n .

3. Viết chương trình đếm số lần xuất hiện của từng loại ký tự từ 'A' đến 'Z' chứa trong một chuỗi được nhập từ bàn phím.

4. Nhập mảng hai chiều A gồm m hàng và n cột.

- Tìm giá trị lớn nhất và nhỏ nhất trên mỗi hàng, mỗi cột cùng với vị trí (dòng, cột) của giá trị này.

- Tìm phần tử có giá trị lớn nhất và nhỏ nhất của mảng A cùng với vị trí (dòng, cột) của hai phần tử này.

- Trong mảng A có bao nhiêu phần tử bằng với phần tử lớn nhất của mảng.

5. Viết chương trình nhập vào từ bàn phím một ma trận vuông và in ra màn hình tổng các phần tử trên đường chéo chính.

6*. Viết một chương trình dùng để giải các bài toán bằng cách tổ chức mỗi thủ tục để giải một bài toán và tạo menu để gọi thực hiện các thủ tục đó theo yêu cầu sau:

1. Giải phương trình bậc hai ($ax^2 + bx + c = 0$).

2. Tính $\sin(x)$.

3. Tính $\cos(x)$.

4. Tính x^3 .

Ồ Ghi chú: Ngoài ra, học viên tự tìm thêm bài tập để thực hành.



MỤC LỤC

BÀI I. Giới thiệu ngôn ngữ pascal và các ví dụ đơn giản	1
I. Xuất xứ ngôn ngữ Pascal	1
II. Khởi động	1
III. Các phím chức năng cần biết của ngôn ngữ Pascal.....	2
IV. Cấu trúc một chương trình Pascal	2
1. Cấu trúc cơ bản.....	2
2. Phương pháp khai báo và tổ chức cấu trúc một chương trình Pascal.....	2
V. Các ví dụ đơn giản làm quen với ngôn ngữ Pascal.....	5
BÀI 2. Các khái niệm cơ bản của ngôn ngữ pascal.....	7
I. Các từ khoá (Key word) trong ngôn ngữ Pascal	7
II. Các kiểu dữ liệu cơ bản	7
1. Các kiểu dữ liệu dạng số nguyên	7
a. Kiểu Byte	7
b. Kiểu Integer	7
c. Kiểu Shortint	7
d. Kiểu Word	7
e. Kiểu Longint	7
2. Các kiểu dữ liệu dạng số có phần biểu diễn thập phân	7
a. Kiểu Single	7
b. Kiểu Real	7
c. Kiểu Double	7
3. Kiểu Char (ký tự)	8
4. Kiểu Logic	8
5. Kiểu String (chuỗi ký tự).....	8
III. Các hàm xử lý dữ liệu cơ bản của ngôn ngữ Pascal	8
IV. Sử dụng hàm Random(n) để lấy một giá trị nguyên ngẫu nhiên	9
BÀI 3. Hằng số, biến số, biểu thức và câu lệnh đơn giản trong ngôn ngữ pascal ..	10
I. Hằng số	10
1. Khái niệm.....	10
2. Cú pháp khai báo.....	10
II. Biến số.....	11
1. Khái niệm.....	11
2. Cú pháp khai báo cho các biến	11



III. Biểu thức	12
IV. Câu lệnh đơn giản.....	12
1. Lệnh gán	13
2. Lệnh Xuất	14
3. Lệnh Nhập.....	17
BÀI 4. Các lệnh có cấu trúc trong ngôn ngữ pascal.....	18
I. Lệnh ghép	18
II. Lệnh lựa chọn	19
1. Lệnh IF.....	19
2. Lệnh CASE	21
III. Các câu lệnh lặp.....	23
1. Câu lệnh FOR.....	23
a. Dạng tiến.....	23
b. Dạng lùi	24
2. Câu lệnh Repeat	25
3. Câu lệnh While.....	27
IV. Các lệnh Goto, Break, Exit và Halt.....	28
1. Lệnh Goto	28
2. Lệnh Break.....	29
3. Lệnh Exit.....	30
4. Lệnh Halt	30
Bài 5. Dữ liệu kiểu vô hướng liệt kê và kiểu đoạn con	31
I. Kiểu liệt kê.....	31
II. Kiểu đoạn con.....	32
Bài 6. Kiểu tập hợp và kiểu mảng	33
I. Kiểu tập hợp:.....	33
1. Định nghĩa.....	33
2. Các phép toán trên tập hợp	33
a. Phép toán quan hệ	33
b. Phép toán IN	34
c. Phép toán hợp, giao, hiệu.....	34
II. Kiểu mảng	35
1. Khái niệm.....	35
2. Khai báo mảng một chiều.....	35
3. Truy cập các phần tử của mảng.....	36



4. Mảng nhiều chiều	37
Bài 7. Chương trình con: Hàm và Thủ tục	40
I. Hàm và thủ tục	40
II. Biến toàn cục, biến cục bộ và việc truyền dữ liệu	42
III. Các hàm và thủ tục thường dùng của Unit CRT	44
1. Thủ tục ClrScr	44
2. Thủ tục ClrEOL	44
3. Thủ tục DelLine	45
4. Thủ tục InsLine	45
5. Thủ tục GotoXY(x, y: Byte)	45
6. Hàm WhereX: Byte	45
7. Hàm WhereY: Byte	45
8. Thủ tục Sound(Hz : Word)	45
9. Thủ tục NoSound	45
10. Thủ tục TextBackGround(Color : Byte)	45
11. Thủ tục TextColor(Color : Byte)	45
12. Hàm KeyPressed: Boolean	45
13. Hàm ReadKey: Char	45
Bài 8. Kiểu xâu ký tự	48
I. Khai báo và các phép toán	48
1. Khai báo kiểu xâu	48
2. Nhập và in xâu ký tự	48
3. Các phép toán trên xâu ký tự	49
a. Phép gán	49
b. Phép nối String	49
c. Các phép toán so sánh	49
II. Các thủ tục và hàm xử lý xâu ký tự	49
1. Các thủ tục	49
a. Delete(St, Pos, Num)	49
b. Insert(St2, St1, Pos)	50
c. Str(Value, St)	50
d. Val(St, Var, Code)	50
2. Các hàm	51
a. Length(St)	51



b. Copy(St, Pos, Num)	51
d. Pos(St1, St2)	52
Bài 9. Dữ liệu kiểu bản ghi và kiểu tệp	54
I. Kiểu bản ghi.....	54
1. Khái niệm và định nghĩa.....	54
2. Sử dụng Record	55
3. Câu lệnh With	57
4. Record có cấu trúc thay đổi	59
Bài 10. Dữ liệu kiểu tệp	62
I. Khái niệm.....	62
II. Cấu trúc và phân loại tệp.....	63
III. Các thao tác trên tệp	63
1. Mở tệp mới để cất dữ liệu	63
2. Ghi các giá trị vào tệp với thủ tục Write	64
3. Đọc dữ liệu từ một tệp đã có	65
4. Tệp truy nhập trực tiếp	67
5. Các thủ tục và hàm xử lý tệp của Turbo Pascal.....	68
a. Hàm FileSize(FileVar)	68
b. Hàm FilePos(FileVar)	68
c. Thủ tục Erase(FileVar).....	68
d. Thủ tục Rename(FileVar, Str)	68
6. Tệp văn bản (Text Files).....	69
a. Hàm EOF(Var F: Text): Boolean.....	69
b. Hàm EOLN(Var F: Text): Boolean	69
c. Ghi vào một tệp văn bản.....	69
d. Đọc dữ liệu từ tệp văn bản.....	70
e. Thủ tục thêm dòng.....	71
PHẦN BÀI TẬP THỰC HÀNH	72
: 1. Luyện tập căn bản	72
: 2. Bài tập đơn giản làm quen với các kiểu dữ liệu và một số hàm chuẩn của Pascal.....	72
: 3. Áp dụng các lệnh đơn giản	73
: 4. Bài tập cho các loại lệnh có cấu trúc	74
4.1. Bài tập cho cấu trúc lệnh If.....	74



— Giáo trình Lập trình Pascal căn bản —

— 90 —

4.2. Bài tập cho cấu trúc lệnh Case	7 5
4.3. Bài tập cho cấu trúc vòng lặp For	7 5
4.4. Bài tập cho cấu trúc vòng lặp Repeat	7 6
4.5. Bài tập cho cấu trúc vòng lặp While	7 6
: 5. Bài tập cho dữ liệu kiểu đoạn con, liệt kê và kiểu mảng	77
: 7. Bài tập tạo thủ tục và hàm	77
: 8. Bài tập cho phần xử lý chuỗi	7 8