MỤC LỤC

Chương 1 **Làm quen ngôn ngữ lập trình Pascal**

1. Giới thiệu các khái niệm cơ bản về lập trình

1.1. Ngôn ngữ Pascal

Vào đầu những năm 1970 do nhu cầu học tập của sinh viên, giáo sư Niklaus Writh - Trường Đại Học Kỹ Thuật Zurich - Thụy Sĩ đã sáng tác một ngôn ngữ lập trình cấp cao cho công tác giảng dạy sinh viên. Ngôn ngữ được đặt tên là PASCAL để tưởng nhớ đến nhà toán học người Pháp Blaise Pascal. Pascal là một ngôn ngữ lập trình có cấu trúc thể hiện trên 3 phương diện.

- Về mặt dữ liệu: Ngoài các kiểu dữ liệu đơn giản còn có các kiểu dữ liệu có cấu trúc. Ta có thể xây dựng các kiểu dữ liệu phức tạp từ các kiểu dữ liệu đã có.
- Về mặt câu lệnh: Từ các câu lệnh đơn giản và lệnh có cấu trúc ta có thể xây dựng các câu lệnh hợp thành.
- Về mặt chương trình: Một chương trình có thể chia làm nhiều chương trình con.

1.2. Turbo Pascal

Khi mới ra đời, Standart Pascal là một ngôn ngữ đơn giản, dùng để giảng dạy và học tập, dần dần các ưu điểm của nó được phát huy và trở thành một ngôn ngữ mạnh. Từ Pascal chuẩn ban đầu, đã được nhiều công ty phần mềm cải tiến với nhiều thêm bớt khác nhau.

TURBO PASCAL là sản phẩm của hãng Borland được dùng rất phổ biến trên thế giới vì những ưu điểm của nó như: tốc độ nhanh, các cải tiến so với Pascal chuẩn phù hợp với yêu cầu người dùng.

TURBO PASCAL 4.0 trở đi có cải tiến rất quan trọng là đưa khái niệm Unit để có thể dịch sẵn các Module trên đĩa, làm cho việc lập trình trở nên ngắn gọn, dễ dàng, chương trình viết dễ hiểu hơn.

Từ phiên bản 5.5 (ra đời năm 1989) trở đi, Turbo Pascal có một kiểu dữ liệu hoàn toàn mới là kiểu Object cho phép đưa các mã lệnh xen kẽ với dữ liệu. Ngoài ra nó còn thư viện đồ hoạ rất phong phú với nhiều tính năng mạnh, ngôn ngữ lập trình cấp cao Delphi cũng sử dụng cú pháp tương tự như Turbo Pascal.

Turbo Pascal 7.0 là phiên bản cuối cùng của Borland. Sau phiên bản này hãng Borland chuyển sang Pascal For Windows trong một thời gian ngắn rồi sản xuất DELPHI.

Turbo Pascal 7.0 hỗ trợ mạnh mẽ lập trình hướng đối tượng nhưng có nhược điểm là bị lỗi "Devide by zero" trên tất cả các máy có xung nhịp lớn hơn 300 MHz. Giải quyết vấn đề này có hai phương án:

a. Cập nhật file TURBO.TPL trong thư mục \BP\BIN.

b. Sử dụng Free Pascal.

Ngoài ra cũng nên lưu ý là Turbo Pascal chạy ở chế độ thực (real mode) nên khi chạy trên nền Windows XP nó hay khởi động lại máy. Nên chạy

Borland Pascal. Khi đó Windows sẽ tạo một môi trường DOS giả lập và chạy ở chế độ đa nhiệm tiện lợi hơn.

2. Làm quen môi trường phát triển phần mềm

2.1. Khởi động Turbo Pascal

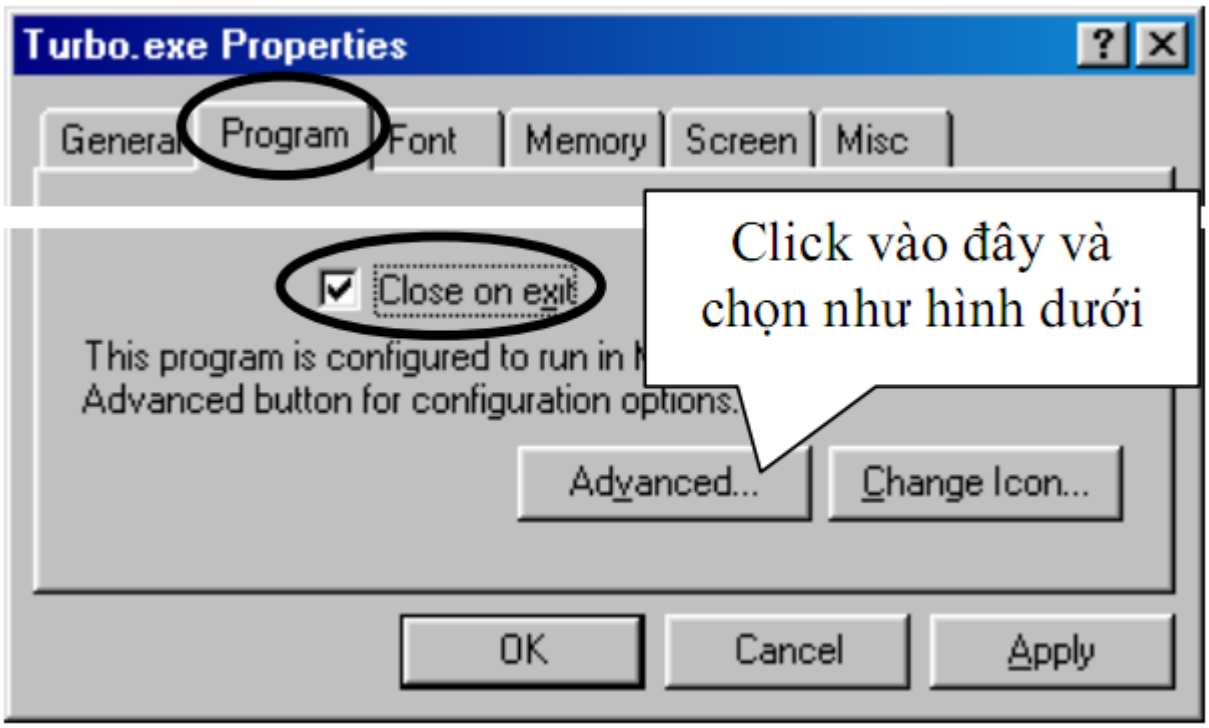
Nếu máy tính chúng ta đã cài đặt Turbo Pascal trên đĩa, ta có thể khởi động chúng như sau (Nếu máy tính chưa có, chúng ta phải cài đặt Turbo Pascal sau đó mới thực thi được)

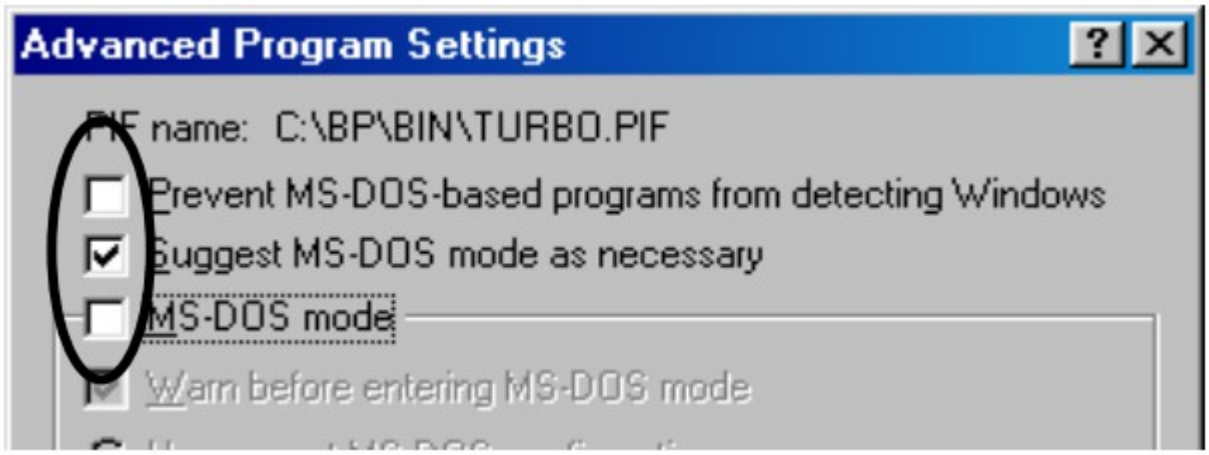
- Từ MS-DOS: Đảm bảo rằng thư mục hiện hành đúng vị trí cài đặt (hoặc dùng lệnh PATH) Turbo Pascal. Ta đánh vào TURBO rồi Enter.

- Từ Windows: Ta nên giả lập MS-DOS Mode cho tập tin TURBO.EXE hoặc Shortcut của nó, nếu không mỗi khi ta thực thi TURBO PASCAL chương trình sẽ thoát khỏi Windows, trở về MS-DOS. Sau khi thoát Turbo Pascal ta phải đánh lệnh EXIT để khởi động lại Windows. Cách giả lập như sau:

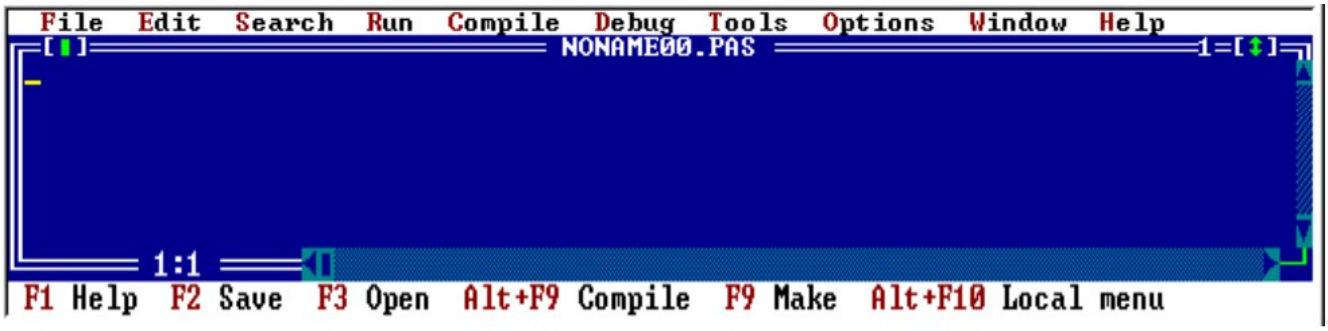
· Nhắp chuột phải lên tập tin TURBO.EXE hoặc Shortcut của nó, chọn Properties.

· Chọn thẻ Program và đánh check như hình sau.





Chọn OK trên các hộp thoại, sau đó khởi động Turbo Pascal, màn hình soạn thảo sau khi khởi động TURBO PASCAL như dưới đây xuất hiện.



Cài đặt và sử dụng Borland Pascal 7.0:

Gói cài đặt Borland Pascal thường được đặt trong thư mục BP70. Mở thư mục này và chạy file cài đặt INSTALL.EXE. Làm theo các hướng dẫn trong quá trình cài đặt. Thông thường sau khi cài đặt xong, chương trình sẽ được đặt trong C:\BP. Hãy vào C:\BP\BIN để cập nhật lại file Turbo.tpl (Chép đè file cùng tên trong thư mục \BP70\Huongdan\ lên file này). Thay vì chạy TURBO PASCAL (File thực thi: BP\BIN\Turbo.exe) hãy tạo Shorcut và chạy BORLAND PASCAL (File thực thi: BP\BIN\BP.exe). Các thao tác sử dụng trên Borland Pascal hoàn toàn giống với các thao tác trên Turbo Pascal nói dưới đây.

2.2. Các thao tác thường sử dụng trên Turbo Pascal

Khi ta muốn tạo mới hoặc mở một tập tin đã có trên đĩa ta dùng phím F3. Sau đó đưa vào tên và vị trí của tập tin. Nếu tập tin đã tồn tại thì Turbo Pascal mở nội dung lên cho ta xem, nếu tên tập tin chưa có thì Turbo Pascal tạo một tập tin mới (với tên mà ta đã chỉ định).

Khi muốn lưu lại tập tin ta dùng phím F2. Trước khi thoát khỏi chương trình, ta nên lưu tập tin lại, nếu chưa lưu chương trình sẽ hỏi ta có lưu tập tin lại hay không. Nếu ta chọn Yes (ấn phím Y) thì chương trình sẽ lưu lại, chọn No (ấn phím N) chương trình sẽ không lưu. Một số phím thông dụng của TURBO PASCAL 7.0

Biểu tượng	Tên phím	Diễn giải
↵	Enter	Đưa con trỏ xuống dòng.
↑	Up	Đưa con trỏ lên 1 dòng.
↓	Down	Đưa con trỏ xuống 1 dòng.
←	Left	Đưa con trỏ qua trái một ký tư.
→	Right	Đưa con trỏ qua phải một ký tự.
Home	Home	Đưa con trỏ về đầu dòng.
End	End	Đưa con trỏ về cuối dòng.
Pg Up	Page Up	Lên một trang màn hình.
Pg Down	Page Down	Xuống một trang màn hình.
Del	Delete	Xoá ký tự tại vị trí con trỏ.
←Back	BackSpace	Xoá ký tự trước con trỏ.
Insert	Insert	Thay đổi chế độ viết xen hay viết chồng.
F1	F1	Gọi chương trình giúp đở.
F2	F2	Lưu tập tin lại.
F3	F3	Tạo mới hoặc mở tập tin.
F4	F4	Thực thi chương trình đến dòng chứa con trỏ.
F5	F5	Phóng lớn cửa sổ.
F6	F6	Chuyển đổi các cửa sổ.
F7	F7	Chạy từng dòng lệnh (hàm xem như một lệnh).
F8	F8	Chạy từng dòng lệnh đơn.
F9	F9	Kiểm tra lỗi chương trình.
Tổ hợp	Alt + F9	Biên dịch chương trình.

Tổ hợp	Ctrl + F9	Chạy chương trình.
Tổ hợp	Ctrl + N	Thêm 1 dòng trước con trỏ.
Tổ hợp	Ctrl + Y	Xoá một dòng tại con trỏ.
Tổ hợp	Ctrl + K + B	Đánh dấu đầu khối.
Tổ hợp	Ctrl + K + K	Đánh dấu cuối khối.
Tổ hợp	Ctrl + K + C	Sao chép khối.
Tổ hợp	Ctrl + K + V	Di chuyển khối.
Tổ hợp	Ctrl + K + Y	Xoá khối.
Trong Borland Pascal các thao tác khối đơn giản và dễ hơn như sau: + Đánh dấu khối: SHIFT + (phím mũi tên) + Copy khối vào clipboard: CTRL+ Ins (phím Insert) + Dán khối (đã copy vào clipboard) vào vị trí mới: SHIFT+ Ins		
Tổ hợp	Ctrl + K + W	Ghi khối lên đĩa thành một tập tin (nội dung của tập tin là khối đã chọn).
Tổ hợp	Ctrl + K + R	Xen nội dung một tập tin (từ đĩa) vào sau vị trí con trỏ.
Tổ hợp	Ctrl + K + H	Tắt/Mở đánh dấu khối.
Tổ hợp	Ctrl + F4	Kiểm tra giá trị biến khi chạy chương trình.
Tổ hợp	Alt + X	Thoát khỏi chương trình.

Chương 2 Các thành phần cơ bản

1. Hệ thống từ khóa và kí hiệu được dùng trong ngôn ngữ lập trình

1.1. Bộ chữ viết

Bộ chữ trong ngôn ngữ Pascal gồm:

- 26 chữ cái la tinh lớn: A, B, C… Z
- 26 chữ cái la tinh nhỏ: a, b, c, … z
- Dấu gạch dưới _ (đánh vào bằng cách kết hợp phím Shift với dấu trừ).
- Bộ chữ số thập phân: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Các ký hiệu toán học: +, -, *, /, =, <, >, (,)
- Các ký hiệu đặc biệt: . , : ; [ ] ? % @ \ | ! # $ { }
- Dấu khoảng cách (khoảng trắng – Space).

1.2. Từ khóa

Các từ khoá là các từ dành riêng (reserved words) của Pascal mà người lập trình có thể sử dụng chúng trong chương trình để thiết kế chương trình. Không được dùng từ khoá để đặt cho các tên riêng như tên biến, tên kiểu, tên hàm… Một số từ khoá của Pascal gồm:

Absolute	External	Mod	Shr
And	File	Nil	String
Array	For	Not	Then
Begin	Forward	Object	To
Case	Function	Of	Type
Const	Goto	Or	Unit

Constructor	If	Packed	Until
Desstructot	Implementation	Procedure	Uses
Div	In	Program	Var
Do	Inline	Record	Virtual
Downto	Interface	Repeat	While
Else	Interrupt	Set	With
End	Label	Shl	Xor

1.3. **Tên**

Tên hay còn gọi là danh biểu (identifier) dùng để đặt cho tên chương trình, hằng, kiểu, biến, chương trình con…tên được chia thành 2 loại.

- Tên chuẩn đã được PASCAL đặt trước, chẳng hạn các hàm số SIN, COS, LN,… hằng số PI, kiểu INTEGER, BYTE, REAL…

- Tên do người dùng tự đặt. Dùng bộ chữ cái, bộ chữ số và dấu gạch dưới để đặt tên, nhưng phải tuân theo qui tắc:

· Bắt đầu bằng chữ cái hoặc "_" sau đó là chữ cái hoặc chữ số.

· Lưu ý:

§ Không có khoảng trống ở giữa tên.

§ Không được trùng với từ khoá.

§ Độ dài tối đa của tên là 127 ký tự, tuy nhiên cần đặt sao cho tên gọn và có ***Ý nghĩa.***

§ Pascal không bắt lỗi việc đặt tên trùng với tên chuẩn, nhưng khi đó ý nghĩa của tên chuẩn không còn giá trị nữa.

§ Pascal không phân biệt chữ hoa và chữ thường (case insensitive) trong từ khóa, tên chuẩn hay tên. Ví dụ "BEGIN" hay "Begin" hay "BeGin" là như nhau. Tuy nhiên sinh viên nên tập thói quen viết một cách thống nhất tên trong toàn bộ chương trình. Điều này giúp các bạn tránh các nhầm lẫn gây

tốn thì giờ khi chuyển sang lập trình bằng các ngôn ngữ có phân biệt chữ hoa chữ thường (case sensitive) như ngôn ngữ C.

2. Các kiểu dữ liệu cơ bản: kiểu số, ký tự, chuỗi, ...

2.1. Kiểu Logic

1. Từ khóa: BOOLEAN
2. Miền giá trị: (True, Fales)
3. Các phép toán: Phép so sánh (=,<,>) và các phép toán logic: And, Or, Not, XOR

Trong Pascal, khi so sánh các giá trị BooLean ta tuân theo quy tắc Fales<True Giả sử A và B là hai giá trị kiểu Boolean. Kết quả của các phép toán được thể hiện qua bảng dưới đây:

A	B	A AND B	A OR B	A XOR B	NOT A
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

2.2. Kiểu số nguyên

2.2.1. Các kiểu số nguyên

Tên kiểu	Phạm vi	Dung lượng
Shortint	-128 → 127	1 byte
Byte	0 → 255	1 byte
Integer	-32768 → 32767	2 byte
Word	0 → 65535	2 byte
LongInt	-2147483648 → 2147483647	4 byte

2.2.2. Các phép toán trên kiểu số nguyên

2.2.2.1. Các phép toán số học

+,-,*,/ (Phép chia cho ra kết quả là số thực)

Phép chia lấy phần nguyên: DIV (ví dụ: 34 DIV 5=6)

Phép chia lấy phần dư: MOD (ví dụ: 34 MOD 5=4)

2.2.2.2. Các phép toán xử lý bit

Trên các kiểu ShortInt, Interger, Byte, Word có các phép toán:

NOT, AND, OR, XOR

A	B	A AND B	A OR B	A XOR B	NOT A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

SHL (phép dịch trái): a SHL n ⇔ a x 2^n

SHR (phép dịch phải): a SHR n ⇔ a DIV 2^n

2.2.3. Kiểu số thực

2.2.3.1. Các kiểu số thực

Tên kiểu	Phạm vi	Dung lượng
Single	$1.5\times10^{-45} \rightarrow 3.4\times10^{+38}$	4 byte
Real	$2.9\times10^{-39} \rightarrow 1.7\times10^{+38}$	6 byte
Double	$5.0\times10^{-324} \rightarrow 1.7\times10^{+308}$	8 byte
Extended	$3.4\times10^{-4932} \rightarrow 1.1\times10^{+4932}$	10 byte

Chú ý: Các kiểu số thực Single, Double và Extended yêu cầu phải sử dụng chung với bộ đồng xử lý số hoặc phải biên dịch chương trình với chỉ thị {$N+} để liên kết bộ giả lập số.

2.2.3.2. Các phép toán trên kiểu số thực: +,-,*,/

Chú ý: Trên kiểu số thực không tồn tại phép toán DIV và MOD

2.2.3.3. Các hàm số học sử dụng cho kiểu số nguyên và số thực:

SQR(x) Trả về x^2

SQRT(x) Trả về căn bậc hai của x (x≥0)

ABS(x) Trả về |x|

SIN(x) Trả về SIN(x) theo radian

COS(x) Trả về COS(x) theo radian

ARCTAN(x) Trả về ARCTANG(x) theo radian

LN(x)	Trả về Ln(x)
EXP(x)	Trả về e^x
TRUNC(x)	Trả về số nguyên gần với x nhất nhưng bé hơn x
INT(x)	Trả về số nguyên của x
FRAC(x)	Trả về phần thập phân của x
ROUND(x)	Làm tròn số nguyên x
PRED(n)	Trả về giá trị đứng trước n
SUCC(n)	Trả về giá trị đứng sau n
ODD(n)	Cho giá trị True nếu n là số lẻ
INC(n)	Tăng n thêm1 đơn vị (n=n+1
DEC(n)	Giảm n đi 1 đơn vị (n=n-1)

2.2.3.4. **Kiểu ký tự**

Từ khóa: CHAR

Kích thước: 1 Byte

Để biểu diễn 1 ký tự , ta có thể sử dụng một trong số các cách sau đây

Đặt ký tự trong cặp dấu nháy đơn. Ví dụ ‘A’,’0’

Dùng hàm CHR(n) (trong đó n là mã ASCII của ký tự cần biểu diễn). Ví dụ CHR(65) biểu diễn ký tự ‘A’

Dùng ký hiệu #n(Trong đó n là mã ASCII của ký tự cần biểu diễn). Ví dụ # 65

Các phép toán =,>,<,>=,<=,<>

Các hàm trên kiểu ký tự

UPCASE(ch): Trả về ký tự in hoa tương ứng với ký tự ch. Ví dụ: UPCASE(‘a’)=’A’

ORD(ch): Trả về số thứ tự trong bảng mã ASCII của ký tự ch. Ví dụ: ORD('A')=65

CHR(n): Trả về ký tự tương ứng trong bảng mã ASCII có số thứ tự là n. Ví dụ: CHR(65)='A'

PRED(ch): Cho ký tự

SUCC(ch): Cho ký tự đứng sau ký tự ch.Ví dụ: SUCC(*A*)='B'

3. Hằng, biến, hàm, các phép toán và biểu thức

3.1. Khai báo hằng

Hằng là một đại lượng có giá trị không đổi trong suốt chương trình

Cú pháp:

CONST<Tên hằng>=<Giá trị>;

Hoặc

CONST<Tên hằng>:=<giá trị>;

Ví dụ:

```
CONST Max=100;
Name='Tran Van Hung';
Continue = False;
Logic = ODD(5); {Logic=True}
```

Chú ý: Chỉ các hàm chuẩn dưới đây mới cho phép sử dụng trogn một biểu thức hằng:

ABS	**CHR**	**HI**	**LO**	**LENGTH**
ODD				
ORD	**PTR**	**ROUND**	**PRED**	**SUCC**
	SIZEOF			
SWAP	**TRUNC**			

3.2. **Khai báo biến**

Biến là 1 đại lượng mà giá trị của nó có thể thay đổi trong quá trình thực hiện chương trình

Cú pháp:

VAR<Tên biến>|,<Tên biến 2,...>|: <Kiểu dữ liệu>;

Ví dụ:

```
VAR  a,b: Integer; {khai báo 2 biến a,b có kiểu integer}
     x,y:real; {khai báo 2 biến x,y có kieery real}
```

Chú ý: ta có thể vừa khai báo vừa gán biến, vừa gán giá trị khởi đầu cho biến bằng cách sử dụng cú pháp như sau:

CONST<Tên biến>:<Kiểu>=<Giá trị>;

Ví dụ:

```
CONST x:integer=5;
```

Với khai báo biến x như trên, trong chương trình giá trị của biến x có thể thay đổi. (Điều này không đúng nếu chúng ta khai báo x là hằng

3.3. **Định nghĩa kiểu:**

Ngoài các kiểu dữ liệu của Turbo Pascal cung cấp ta có thể định nghĩa các kiểu dữ liệu mới dựa trên các kiểu dữ liệu đã có.

Cú pháp:

TYPE <Tên kiểu>=<Mô tả kiểu>;

VAR<Tên biến>:<Tên kiểu>;

Ví dụ:

```
TYPE
    Sothuc=Real;
    Tuoi=1..100;
```

```
        ThuNgay=(Hai,Ba,Tu,Nam,Sau,Bay,CN)
    VAR
        x:Sothuc;
        tt:tuoi;
        Day:ThuNgay;
```

3.4. Biểu thức

Biểu thức (Expression) là công thức tính toán mà trong đí bao gồm các phép toán, các hằng, các biến, các hàm và các dấu ngoặc đơn.

Ví dụ: (x+sin(y))/(5-2*x) Biểu thức số học

(x+4)*2=(8+y) Biểu thức Logic

Trong một biểu thức, thứ tự ưu tiên của các phép toán được liệt kê theo thứ tự sau:

Lời gọi hàm

Dấu ngoặc ()

Phép toán 1 ngôi (NOT,-)

Phép toán *,/,DIV,MOD,AND

Phép toán +,-,OR,XOR

Phép toán so sánh =,<,>,<=,>=,<>,IN

4. Các lệnh, khối lệnh

4.1.Câu lệnh đơn giản

Câu lệnh gán (:=): <Tên biến>:=<Biểu thức>;

Lời gọi hàm, thủ tục

4.2. Câu lệnh có cấu trúc

Câu lệnh ghép: **BEGIN...END;**

Các cấu trúc điều khiển: IF..., Case ..., FOR..., REPEAT...,WHILE...

4.3. Các lệnh nhập xuất dữ liệu

4.3.1. Lệnh xuất dữ liệu

Để xuất dữ liệu ra màn hình , ta sử dụng ba dạng sau:

(1) WRITE(<Tham số 1>[<Tham số 2>,...]);

(2) WRITELN(<Tham số 1>[<Tham số 2>,...]);

(3) WRITELN;

Các thủ tục trên có chức năng sau:

(1) Sau khi xuất giá trị của các tham số ra màn hình thì con trỏ không xuống dòng

(2) Sau khi xuất giá trị của các tham số ra màn hình thì con trỏ xuống đầu dòng tiếp theo

(3) Xuất ra màn hình 1 dòng trống

Các tham số có thể là các hằng, biến, biểu thức. Nếu có nhiều tham số trong câu lệnh thì các tham số phải được phân cách nhau bởi dấu phẩy.

Khi sử dụng lệnh WRITE/WRITELN, ta có 2 cách viết: Không quy cách và có quy cách:

Viết không quy cách: dữ liệu xuất ra sẽ canh lề bên trái. Nếu dữ liệu là số thực thì sẽ được in ra dưới dạng biểu biễn khoa học

Ví dụ:

```
WRITELN(x); WRITE(SIN(3*x));
```

Viết có quy cách: Dữ liệu xuất ra sẽ được canh lề ở phía bên phải

Ví dụ:

```
WRITELN(x:5); WRITE(SIN(13*x):5:2);
```

Câu lệnh	Kết quả trên màn hình
Writeln('Hello');	Hello
Writeln('Hello':10);	Hello
Writeln(500);	500
Writeln(500:5);	500
Writeln(123.457)	1.2345700000E+02
Writeln(123.45:8:2)	123.46

4.3.2. Nhập dữ liệu

Để nhập dữ liệu từ bàn phím vào các biến có kiểu dữ liệu chuẩn (trừ các biến kiểu BOOLEAN), ta sử dụng cú pháp sau đây:

READLN(<Biến 1>,<Biến 2>,...,<Biến n>);

Chú ý: Khi gặp câu lệnh READLN; (không có tham số), chương trình sẽ dừng lại chờ người sử dung nhấn phím ENTER mới chạy tiếp.

4.3.3. Các hàm và thủ tục thường dùng trong nhập xuất dữ liệu

Hàm KEYPRESSED: hàm trả về giá trị TRUE nếu như có một phím bất kỳ được nhấn, nếu không hàm cho giá trị là FALSE

Hàm READKEY: Hàm có chức năng đọc một ký tự tưg bộ đệm bàn phím

Thủ tục GOTOXY(X,Y:Integer): Di chuyển con trỏ đến cột X dòng Y.

Thủ tục CLSCR: Xóa màn hình đưa con trỏ về góc trên bên trái màn hình

Thủ tục CLREOL: Xóa các ký tự từ vị trí con trỏ đến hết dòng

Thủ tục DELLINE: Xóa các dòng tại vị trí con trỏ dồn các dòng ở phía dưới lên

Thủ tục TEXTCOLOR(CoLor:Byte): Thiết lập màu cho các ký tự. Trong đó Color €[0,15].

Thủ tục TEXTBACKGROUND(Color:Byte): Thiết lập mầu nền cho màn hình

5. Thực thi chương trình, nhập dữ liệu, nhận kết quả

5.1.Các bước cơ bản khi lập một chương trình Pascal

Bước 1: Soạn thảo chương trình

Bước 2; Dịch chương trình (nhấn phím **F9**), nếu có lỗi phải sửa lỗi

Bước 3: Chạy chương trình (nhấn phím **Ctrl+F9**)

5.2. Cấu trúc chung của một chương trình Pascal

Một chương trình trong Pascal gồm các phần khai báo và sau đó là thân của chương trình.

- Khai báo Program
- Khai báo Uses
- Khai báo Label
- Khai báo Const
- Khai báo Type
- Khai báo Var
- Khai báo các chương trình con (thủ tục hay hàm)
- Thân chương trình

Thân của chương trình được bắt đầu bằng từ khoá Begin và kết thúc bằng từ khoá End và dấu chấm ".". Giữa Begin và End. là các phát biểu.

Ví dụ:

```
Program Chuongtrinhmau;
   Uses
```

……

Label

……

Const

…...

Type

……

Var

….. (Khai báo tên và kiểu của các biến)

Function …

End;

Procedure …

End;

Begin

……

……

End.

Thông thường trong một chương trình Pascal, các khai báo Uses, Label, const, type, Function, Procedure có thể có hoặc không tuỳ theo bài, nếu không dùng biến thì cũng không cần khai báo Var (như ví dụ ở bài 1), tuy nhiên hầu hết các chương trình đều dùng khai báo Program, var các biến và thân chương trình.

Chương 3 Các cấu trúc điều khiển

1. Các lệnh cấu trúc lựa chọn

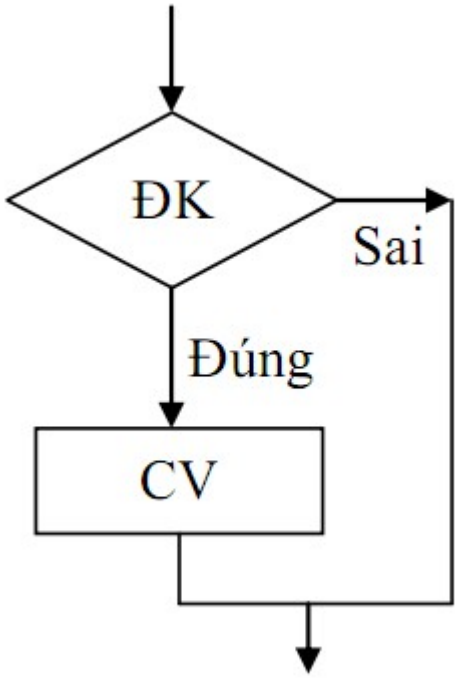
1.1. Lệnh cấu trúc rẽ nhánh

1.1.1. Dạng không đầy đủ

Cú pháp: IF <Điều kiện> THEN <Công việc>;

Nếu điều kiện đúng thì thực hiện công việc (ngược lại là điều kiện sai thì không thực thi công việc)

Lưu đồ giải thuật:



Ví dụ:

```
Var a,b: Interger;
Begin
    Write('Nhập a:'); Readln(a);
    Write('Nhập b:'); Readln(b);
If b<>0 then
    Write('Thương hai số vừa nhập:',a/b:5:2);
    Readln;
End.
```

1.1.2. Dạng đầy đủ

Cú pháp: IF <Điều kiện> THEN <Công việc 1>
ELSE <Công việc 2>;

Nếu điều kiện là đúng thì thực hiện công việc 1 ngược lại là điều kiện sai thì thực thi công việc 2. Chú ý trước ELSE không có dấu ; (chấm phẩy).

Ví dụ:

```
Var a.b: interger;
Begin
    Write('Nhập a:');Readln(a);
    Write('Nhập a:');Readln(a);
    If b<>0 then
        Write('Thương hai số vừa nhập:',a/b:5:2);
Else
        Write('Không thể chia cho 0');
        Readln;
    End.
```

1.2. Lệnh cấu trúc lựa chọn

1.2.1. Dạng không đầy đủ

Cú pháp: CASE <biến> OF
Hằng 1a, 1b,..., 1x: <Công việc 1>;
Hằng 2a, 2b,..., 2x: <Công việc 2>;
.....
Hằng na, nb,..., nx: <Công việc n>;
END;

Ý nghĩa:

Trước hết kiểm tra giá trị của biến có bằng một trong các hằng 1a, 1b,…1x hay không. Nếu đúng thì thực hiện công việc công việc 1, rồi kết thúc lệnh (thực hiện tiếp các lệnh sau END; nếu có). Nếu không, thì kiểm tra giá trị của biến có bằng một trong các hằng 2a,2b,…,2x hay không. Nếu đúng thì thực hiện công việc 2, rồi kết thúc lệnh (thực hiện tiếp các lệnh sau END). Nếu không thì cứ tiếp tục kiểm tra như vậy. Nếu giá trị của biến không bằng bất cứ hằng nào từ 1a đến nx thì câu lệnh CASE kết thúc mà không làm gì cả.

Ví dụ: Viết chương trình nhập vào một tháng, sau đó in lên màn hình tháng đó có bao nhiêu ngày.

```
Var T: interger;
Begin
    Write('Nhập vào một tháng: '); Readln(T);
CASE T OF
    1,3,5,7,8,10,12:    Write('Tháng có 31 ngày.');
    4,6,9,11:           Write('Tháng có 30 ngày.');
    2:                  Write('Tháng có 28 (nhuận 29) ngày.');
End;
Readln;
End.
```

1.2.2. **Dạng đầy đủ**

***Cú pháp*:** **CASE <biến> OF**
Hằng 1a, 1b,…, 1x: <Công việc 1>;
Hằng 2a, 2b,…, 2x: <Công việc 2>;
……
Hằng na, nb,…, nx: <Công việc n>;
ELSE
<Công việc N+1>
END;

Ý nghĩa:

Khác dạng không đầy đủ ở chỗ nếu giá trị của biến không bằng bất cứ hằng nào từ 1a đến nx thì câu lệnh CASE sẽ thực thi công việc N+1.

Ví dụ: Viết chương trình nhập vào một tháng, sau đó in lên màn hình tháng đó có bao nhiêu ngày.

```
Var T: interger;
Begin
        Write('Nhập vào một tháng: '); Readln(T);
    CASE T OF
        1,3,5,7,8,10,12:    Write('Tháng có 31 ngày.');
        4,6,9,11:           Write('Tháng có 30 ngày.');
        2:                  Write('Tháng có 28 (nhuận 29) ngày.');
    ELSE
        Write('Tháng sai. Phải nhập số từ 1 đến 12.')
    End;
    Readln;
```

End.

Chú ý: Biến sau từ khóa CASE phải là biến đếm được.

1.3. Các lệnh vòng lặp

1.3.1. Lặp với số lần xác định

a. Dạng 1

Cú pháp: **FOR <biến>:=<đầu> TO <cuối> DO <Công việc>**

Ý nghĩa: Các bước thực hiện như sau:

Bước 1: Kiểm tra giá trị đầu có <= (nhỏ hơn hoặc bằng) giá trị cuối hay không. Nếu đúng thì gán giá trị đầu cho biến và thực thi công việc

Bước 2: Kiểm tra giá trị biến <> (khác) giá trị cuối hay không. Nêu đúng thì tăng thêm biến một đơn vị (biến:= SUCC (biến)) rồi thực hiện công việc.

Lập lại bước 2. Cho đến khi giá trị biến bằng giá trị cuối thì kết thúc câu lệnh.

Chú ý:

Biến sau từ khóa FOR phải là biến đếm được và giá trị đầu phải <= giá trị cuối. Trong các lệnh của công việc khoongneen có các lệnh làm thay đổi giá trị của biến đếm. Vòng lặp kết thúc, giá trị biến là giá trị cuối.

Ví dụ: Để in lên màn hình dãy số 1,2,3…,n ta có thể làm như sau:

```
Var I,n: Interger;
Begin
        Write('Nhập vào một số:');Readln(n);
        Write('Dưới đây là dãy số từ 1 đến số bạn vừa nhập:');
        For i:=1 to n do
                Write('     ',i);
```

Readln;

End.

b. **Dạng 2**

Ý nghĩa: Tương tự dạng 1, nhưng sau mỗi lần lặp thì biến giảm đi một đơn vị (biến:=PRED (biến)).

Cú pháp: **FOR <biến>:=<đầu> DOWNTO <cuối> DO <Công việc>**

Ví dụ: Liệt kê các số nguyên dương là ước số của một số cho trước

```
Var I,n:Interger;
Begin
    Write('Nhập vào một số:'); Readln(n);
    Writeln('Dưới đây liệt kê các ước số của số bạn vừa nhập');
For i:=n Downto 1 Do
            If n Mod i=0 Then
                Write('  ',i);
    Readln;
End.
```

Mở rộng vấn đề:

Không giống với các ngôn ngưc khác, Pascal không kiểm tra (biến >cuối) trong câu lệnh FOR … To … Do để kết thúc vòng lặp mà là kiểm tra (biến = cuối) để thực hiện lần lặp cuối cùng. Vì lẽ đó việc can thiệp vào biến đếm có thể gây ra sự cố "Vòng lặp vô tận". Ví dụ sau đây cho thấy rõ điều đó.

```
Program Lapvotan;
```

```
Use Crt, Dos;
Var Bien:byte; CtrlBreak:Boolean;
BEGIN
GetCBreak(CtrlBreak);
IF (CtrlBreak=FALSE) THEN CtrlBreak:=Not CtrlBreak;
SetCBreak(CtrlBreak);
Writeln('Phải gõ CTRL-BREAK mới chaasmd]ts được!');
For bien:=240 to 250 do
    Begin
        If (bien=245) Then bien:=252;
        Writeln('Giá trị hiện nay của biến là:',Bien,#7);
        Delay(100);
    End;
End.
```

Giải thích:

Thủ tục GetCBreak(Bien:Boolean) và thủ tục SetCBreak(Bien:Boolean) thuộc Unit Dos và thủ tục Delay(Num:Word) thuộc Unit CRT nên phải khai báo USES DOS, CRT;

Thủ tục GetCBreak(CtrlBreak) Kieemrtra tình trạng cài đặt Ctrl + Break hiện tại và trả về tình trạng đó trong biến CtrlBreak. Thủ tục SetCBreak(True); kích hoạt việc cho phép gõ Ctrl + Break để ngưng chương trình trong mọi tinh huống.

#7 (Kí tự số 7) là mã ASCII làm xuất ra tiếng Beep của loa bên trong máy

Khi biến (điều kiện vòng lặp) đạt giá trị 245 thì bị gán lại thành 252 nên không khi nào bien bằng 250 để Pascal chấm dứt vòng lặp. Ngay cả khi biên

đã duyệt qua hết phạm vi của kiểu dữ liệu (tức giá trị 255) thì bien quay lại giá trị 0… và mọi thứ lại tiếp tục … trừ khi gõ Ctrl – Break.

2. Lệnh lặp với số lần lặp không xác định

2.1. Dạng 1:

Cú pháp: **WHILE <điều kiện> DO** **<Công việc>**

Ý nghĩa:

Vào lệnh sẽ kiểm tra điều kiện, nếu điều kiện đúng thì thực thi công việc, sau đso quay lại kiểm tra điều kiện. Cứ tiếp tục như thế cho tới khi nào điều kiện sai kết thúc.

Ví dụ:

Tính tiền gửi ngân hàng. Lãi xuất hàng tháng là 1.7% người đó gửi vào ngân hàng vốn ban đầu là 1000000(1 triệu), cứ sau mỗi tháng sau. Hỏi sau bao lâu người đó được 1 tỷ đồng?

```
Var LS, Vn, Mm, tam: Real;
    Sothang,i:interger;
Begin
    Writeln('CHƯƠNG TRÌNH TÍNH TIỀN GỬI NGÂN HÀNG');
    Ls:=1.7/100;{Lãi xuất 1.7%}
    Vn:=1000000;{Số vốn ban đâu là 1 triệu}
    Mm:=1000000000; {Số tiền mong muốn 1 tỷ}
    Sothang:=0;
    Tam:=Vn;
    While (Tam<mm) do
```

```
        Begin
            Tam:=Tam+Ls*tam;
            Sothang:=Sothang +1;
        End;
        Writeln('Số tháng =',Sothang);
        Writeln('Tiền vốn cộng lại là:',tam:12:2);
    Readln;
    End.
```

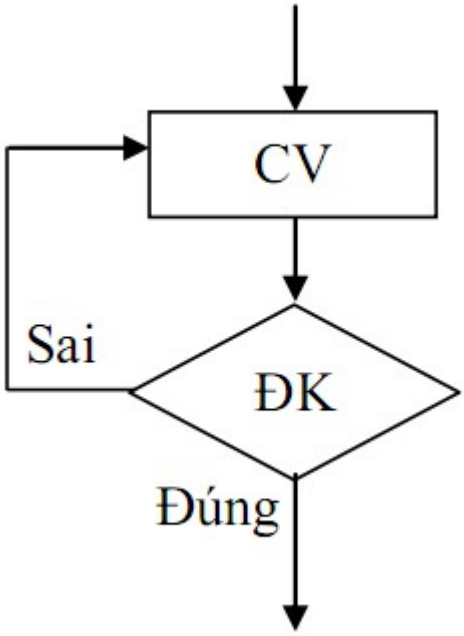
2.2. Dạng 2:

Cú pháp: **REPEAT** **<Công việc>** **UNTIL <Điều kiện>**

Ý nghĩa:

Vào lệnh sẽ thực thi công việc, sau đó kiểm tra điều kiện, nếu điều kiện sai thì tiếp tục thực hiện công việc sau đó kiểm tra điều kiện.Cứ tiếp tục như thế cho tới khi nào điều kiện đúng thì kết thúc.

Lưu đồ thuật toán



Ví dụ: Viết chương trình nhập vào bán kính, tính chu vi diện tích của hình tròn. Sau khi in ra chu vi, diện tích thì hỏi người dùng có tiếp tục nữa không? (C/K). Khi nào người dùng ấn phím 'K' thì thoát, ngược lại cho người dùng tiếp tục nhập vào bán kính và in ra chu vi và diện tích mới.

```
USES crt;
Var C,S,R:Real;
    Traloi:Char;
Begin
    Clrscr;
    Repeat
        Write('Nhập vào bán kính:');Readln(R);
        C:=2*R*PI;{Chu vi hình tròn}
        S:=PI*R*R;{Diện tích hình tròn}
        Writeln('Chu vi:',C:0:2);
        Writeln('Diện tích:',S:0:2);
```

```
        Writeln;
        Write('Tiếp tục (C/K)?');Readln(Traloi);
    Until UpCase(Traloi)='K'; {Lưu ý: 'K'in hoa}
End.
```

Sự khác nhau giữa While … do và Repeat …Until và For…To…Do

Vòng lặp FOR là vòng lặp xác định trước số lần lặp. Trừ khi cần thiết, nói chung không can thiệp vào biến đếm vòng lặp.

Cả hai vòng lặp WHILE và REPEAT đểu là vòng lặp không xác định trước số lần lặp. Cần phải có câu lệnh thay đổi giá trị biến điều khiển vòng lặp để có thể thoát ra khỏi vòng lặp

Trong vòng lệnh WHILE… DO thì điều kiện sẽ được kiểm tra trước, nếu điều kiện đúng thì thực hiện công việc. Còn trong lệnh REPEAT… UNTIL thì ngược lại, công việc được làm trước rồi mới kiểm tra điều kiện, nếu điều kiện đúng thì vòng lặp kết thúc. Như vậy đối với vòng lặp REPEAT bao giờ thân vòng lặp cũng được thực hiện ít nhất 1 lần, trong khi thân vòng lặp WHILE có thể không được thực hiện lần nào. Tùy những hoàn cảnh khác nhau mà ta lựa chọn vòng lặp cho thích hợp. Nếu dùng 2 lệnh này để giải cùng 1 bài toán, cùng một giải thuật như sau thì điều kiện sau WHILE và điều kiện sau UNTILL là phủ định nhau.

Chương 4 Hàm và thủ tục

1. Khái niệm chương trình con

Trong chương trình, có những đoạn cần phải lặp đi lặp lại nhiều lần ở những chỗ khác nhau. Để tránh phải viết lại các đoạn đó người ta thường phân chương trình ra thành nhiều Module, Mỗi Module giải quyết một công việc nào đó, các Module như vậy là những chương trình con (SubProgram)

Một tiện lợi khác của việc sử dụng Module là ta có thể dễ dàng kiểm tra tính đúng đắn của nó trước khi ráp nối vào chương trình chính. Do đó việc xác định sai sót và tiến hành điều chỉnh trong chương trình sẽ thuận lợi hơn.

Trong Pascal chương trình con được viết dưới dạng hàm (Function) hoặc thủ tục (Procedure) Hàm và thủ tục đều là những chươgn trình con nhưng hàm khác thủ tục ở chỗ hàm trả về một giá trị cho lệnh gọi thông qua tên hàm còn thủ tục thì không. Do đó ta chỉ dùng hàm khi thỏa mãn các yêu cầu sau:

Ta muốn nhận 1 kết quả và chỉ 1 mà thôi

Ta cần dùng tên chương trình con (chứa kết quả đó) để viết trong các biểu thức

Nếu không thỏa mãn 2 yêu cầu trên thì ta dùng thủ tục.

BorLand Pascal thiết kế và cài đặt sẵn trong các Unit đi kèm theo gói phần mềm nhiều thủ tục và hàm rất tiện dụng. Muốn sử dụng các thủ tục hoặc hàm trong Unit nào ta chỉ cần khai báo tên Unit đó trong câu lệnh USES. Tuy nhiên phần lớn các thủ tục và hàm dùng trong chương trình là do người dùng phải tự viết.

Ví dụ:

```
{$X+}
Progam TestExtenSyntax;
Uses crt;
Var I,j:byte;
{----------------------}
Function DoiViTri(I,j:byte):byte;
Var Tam:byte;
Begin
        Tam:=I;i:=j;j:=tam;
        Gotoxy(I,j);write(‘*’)
End;
Begin
I:=5;j:=20;
 Gotoxy(i,j); write('*');
  Doivitri(i,j);
  readln;
END.
```

2. Các hàm và thủ tục trong ngôn ngữ lập trình

2.1. Hàm (Function)

Hàm là một chương trình con tính toán trả về cho ta một giá trị kiểu vô hướng.

Cấu trúc hàm như sau:

FUNCTION <Tên hàm>[(<Th.số>:<Kiểu>[;<Th.số>: <Kiểu>])]: <KiểuKQ>;	(Header)
[VAR <Biến>:<Kiểu>[;<Biến>: <Kiểu>]]	Khai báo các biến cục bộ nếu có.

BEGIN
<các câu lệnh>
END;
	Thân hàm

. Tên hàm là một danh biểu, phải tuân thủ theo quy tắc đặt danh biểu.

. Một hàm có thể không có hoặc có một hoặc nhiểu tham số. Trong trường hợp có nhiều tham số có cùng môt kiểu dữ liệu thì ta có thể viết chúng cách nhau bởi dấu ,(phẩy). Ngược lại, các tham số hình thức khác kiểu nhau thì phải cách nhau dâu ;(chấm phẩy).

. KiểuKQ là kiểu vô hướng, nó phản ánh kiểu của giá trị mà hàm trả lại về sau khi chạy xong. Ví dụ, ta khai báo hàm như sau:

FUNCTION TEST(x,y:Interger;z:Real):Real;

Đây là một tên hàm có tên là TEST, với 3 tham số, x và y thuộc kiểu interger, z thuộc kiểu Real

. Trong hàm, ta có thể sử dụng các hằng, kiểu, biến dùng riêng trong nội bộ hàm.

. Thông thường mục đích sử dụng hàm là để lấy trị trả về do đó cần lưu ý gán

kết quả cho tên hàm trong thân hàm.

Ví dụ 1: Ta xây dựng hàm DT truyền tham số vào là bán kính của hình tròn, hàm này sẽ trả về diện tích của hình tròn đó.

```
Program TinhDienTich;
Uses Crt;
VAR   BanKinh: real; Ch: Char;
{-----------------------------------------}
Function DT(Radius:Real):Real;
Begin
  DT := PI * Radius* Radius;
End;
{-----------------------------------------}
                                        Begin
  Clrscr;
  Repeat
    Write('Nhập bán kính: '); Readln(BanKinh);
    Writeln('Diện tích hinh tron tuong ung: ' ,DT(Bankinh):0:2);
    Writeln;
         Write('Tiếp tục (C/K)? ');
     Repeat
       ch:=readkey;
     Until Upcase(ch) in ['C','K'];
  Until UpCase(Ch) = 'K';                {Lưu ý: 'K' in hoa}
End.
```

ể trả về giá hàm.

Ví dụ 2:

```
Program TinhGiaithua;
                                        USES CRT;
Var Num:longint; Ch:char; X,Y:byte;
```

```
{------------------------------------------}
Function GiaiThua(m: longint): longint;
Var Tam, Dem:Longint;
BEGIN
IF (M<0) THEN
 Begin
    Write('Khong tinh duoc'); HALT(1);
  End
ELSE
  Begin
      Tam:=1;
      For Dem:=1 to m do Tam:=Tam*Dem;
      GiaiThua:=Tam;
  End;
  END;
                {---------- Chương trình chính ----------}
BEGIN
                    Writeln('CHUONG TRINH TINH GIAI THUA.');
REPEAT
   Write('Cho so nguyen muon tinh giai thua. M= ');
   X:=WhereX; Y:=WhereY;
   REPEAT
Gotoxy(X,Y); CLREOL; Readln(Num);
  UNTIL (Num>=0);
   Writeln(M,'! = ',GiaiThua(Num));
```

```
    REPEAT
       Write('Tinh nua khong ? (C/K) :'); CH:=READKEY;
    UNTIL Upcase(Ch) in ['C','K'];
          Writeln(Ch);
UNTIL Upcase(Ch)='K';
Readln
END.
```

2.2. Thủ tục (Procedure)

Cấu trúc của một thủ tục như sau:

PROCEDURE <Tên>(<Th.số>:<Kiểu>[;<Th.số>: <Kiểu>]): <Kiểu>;	(Header)
[VAR <Biến>:<Kiểu>[;<Biến>: <Kiểu>]	Khai báo các biến cục bộ nếu có.

BEGIN
<các câu lệnh>
END;
	Thân thủ tục.

Như vậy cấu trúc của một thủ tục cũng tương tự nhq cấu trúc của một hàm. Chỉ có hai điều khác:

- Header bắt đầu bằng từ khóa Procedure thay vì Function.
- Không có câu lệnh gán <Tenham:=GiaTri;> trong thân Procedure.

Ví dụ:

Thủ tục INSO sau sẽ in các số từ 1 đến giá trị biến truyền vào. Với n là tham số

thực tế, So là tham số hình thức.

```
Program TEST;
Var n: Integer;
{---------------------------------------}
Procedure INSO(So: Integer);
Var  i: Integer;
Begin
  For i := 1 to So do
    Write( i:10 );
End;
{------------ Chương trình chính ----------------------}
Begin
  Write('Nhập một số bất kỳ lớn hơn không: '); Readln(n);
  INSO( n );
  Readln;
End.
```

3. Tham trị và tham biến

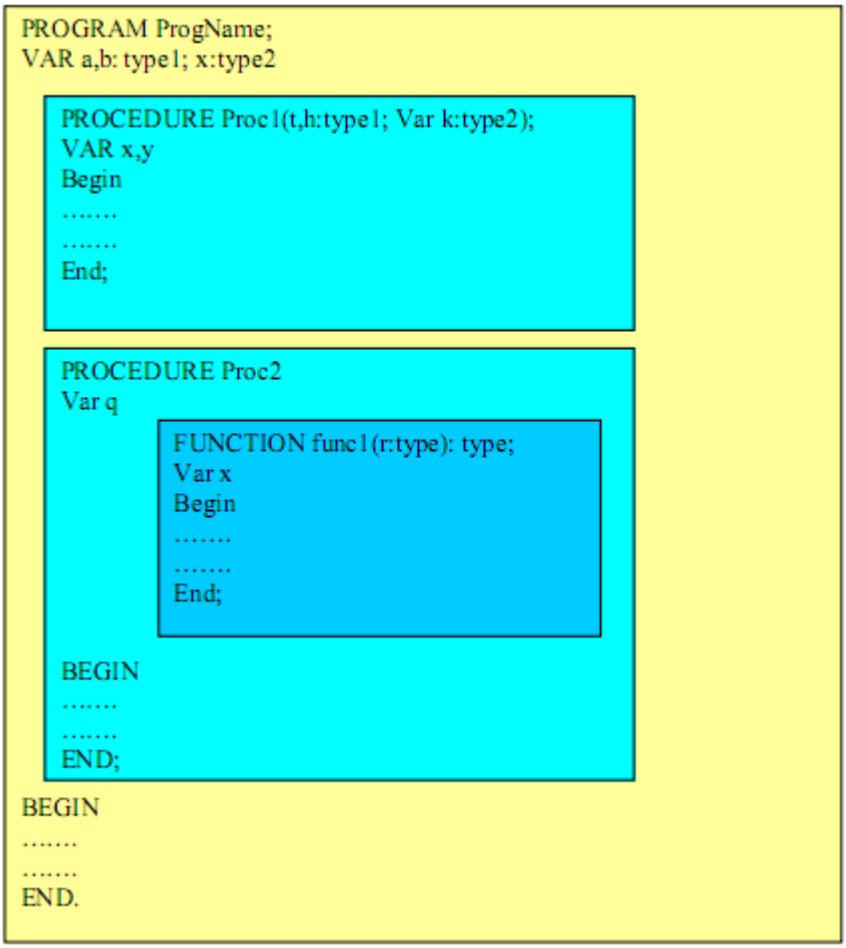
Một chương trình có thể gồm một chương trình chính và nhiều chương trình con. Kèm theo đó là các biến, các tham số khai báo ở các vị trí khác nhau trong chương trình. Khả năng từ một vị trí nào đó trong chương trình "nhìn thấy" một chương trình con, một biến đã được khai báo là rất quan trọng. Mặt khác khi làm việc theo nhóm, các chương trình con, các modune khác nhau của chương trình có thể do nhiều người, nhiều nhóm lập trình khác nhau thực hiện.

Khi đó khả năng xảy ra các nhóm khác nhau dùng cùng một tên biến, tên hàm, tên thủ tục cho các mục đích khác nhau là rất lớn. Vì vậy ngoài khả

năng "nhìn thấy", chương trình cần có một cơ chế cấu trúc sao cho có thể "che khuất" các biến khi cần thiết. Phần sau đây, nhằm mục đích đó, nghiên cứu các khái niệm liên quan đến "tầm vực "của biến và của chương trình (con) cũng như các hiệu ứng lề (side effect) có thể xảy ra.

KHỐI (block): Một khối bắt đầu từ Header (PROGRAM | FUNCTION | PROCEDURE) của khối đó cho đến từ khóa END (END. hoặc END;) của thân chương trình/chương trình con tương ứng.

Minh họa:



Trong minh họa trên ta có các khối ứng với chương trình chính, các khối ứng với các Procedure Proc1, Procedure Proc2, Function func1, trong đó Proc1 và Proc2 là hai khối con cùng cấp, func1 là khối con của khối Proc2.

TẦM VỰC: Tầm vực của một biến hay một chương trình con là phạm vi mà biến đó hoặc chương trình con đó được nhìn thấy trong chương trình (ie: có thể gọi được biến đó hoặc chương trình con đó). Tầm vực của một biến hay một chương trình con bắt đầu từ chỗ nó được khai báo trong khối cho đến hết khối mà nó được khai báo trong đó, kể cả trong các khối con trừ khi trong khối con có khai báo lại biến hoặc chương trình con đó.

Theo qui định trên, Và áp dụng cho hình minh họa trước ta thấy:

- Các biến a,b là các biến toàn cục có thể gọi được ở bất cứ nới đâu trong chương trình.
- Biến x của chương trình chính có thể gọi được ở bất cứ đâu trong chương trình trừ

trong PROCEDURE Proc1 và trong FUNCTION func1vì trong procedure/function

này có khai báo lại biến x. Trong thân procedure/function đó khi gọi x là ta gọi đến

biến x cục bộ của nó chứ không phải biến x toàn cục.

- Các biến t,h,k và y chỉ có thể gọi được trong Proc1 mà thôi.
- Biến x nếu gọi trong Proc1 là biến cục bộ của riêng nó mà thôi.
- Biến q có thể gọi được trong Proc2 và trong func1 mà thôi. Biến r chỉ có thể gọi được

trong Func1 mà thôi. Biến x nếu gọi trong func1 là biến cục bộ của riêng func1, không

liên quan gì đến biến x khai báo trong chương trình chính và trong Proc1.

- Procedure Proc1 có thể gọi được trong Proc2, Func1 và trong chương trình chính. Trong Procedure Proc1 dĩ nhiên, theo qui định này, cũng có thể gọi chính nó (Đây là trường hợp gọi đệ qui mà ta sẽ nghiên cứu sau)
- Proc2 có thể gọi được trong chương trình chính, trong Func1 và trong chính nó. Proc1

không thể gọi được Proc2.

- Func1 chỉ có thể gọi được bởi Proc2.
- Proc1 và chương trình chính không thể gọi được Func1.
- Có một ngoại lệ: Chương trình chính không thể gọi chính nó.

4. Sự hoạt động của chương trình con khi được gọi và sự bố trí biến

- Khi chương trình hoặc chương trình con được gọi thì các biến, các "tên" chương trình con được bố trí trong một vùng nhớ gọi là STACK. Khi chương trình chính được gọi

thì các biến toàn cục đqợc bố trí vào stack và tồn tại ở đó cho đến lúc chấm dứt chương trình. Khi các chương trình con được gọi thì các biến trong khai báo tham số hoặc sau từ khóa VAR (của nó) được bố trí vào stack và sẽ được giải phóng khi chương trình con này chấm dứt. Điều này rất có lợi vì nó cho phép ta sử dụng vùng

nhớ hợp lí hơn. Người ta càng dùng ít biến toàn cục càng tốt để tránh lỗi (trong thời

gian chạy) làm tràn stack (Stack overflow error).

5. VẤN ĐỀ TRUYỀN THAM SỐ KHI GỌI CHƯƠNG TRÌNH CON.

- Khi gọi một chương trình con (thủ tục hay hàm) ta phải theo các qui định sau đây:

· - Nếu chương trình con có qui định các tham số thì phải truyền giá trị hoặc biến cho các tham số đó.

· - Phải truyền đủ số tham số.

· - Phải truyền đúng kiểu dữ liệu theo thứ tự các tham số đã khai báo.

Để hiểu rõ cách Pascal xử lí việc truyền tham số chúng ta cần xem qua ví dụ sau đây:

```
Program  ParameterPassing;
Var a,b:byte; c:integer;
{-----------------------------------------------------}
Procedure TestVar (x,y,z: byte; Var t: integer);
Var d: byte;
Begin
D:=4;              {1}
X:=X+D; B:=B+X; T:=T+D;      {2}
Writeln(,,Ben trong thu tuc:");
Writeln(,,A=",a, ,,B=",b,"C=",c,"D=",d,"X=",x,"Y=",y,"Z=",z,"T=",t);
End;
{-----------------------------------------------------}
BEGIN
        A:=3; B:=5; C:=8;
        Writeln(,,Truoc khi goi thu tuc:");
        Writeln(,,A=",a, ,, B=",b," C=",c);

TestVar(a,5,c,c);
```

Writeln(‘Sau khi goi thu tuc:’);

Writeln(‘A=’,a, ‘ B=’,b,’C=’,c);

Readln;

END.

- Quá trình chạy chương trình trên và diễn biến trong bộ nhớ như sau:
- * Trước khi gọi thủ tục:
- Cấp vùng nhớ cho các biến toàn cục a,b,c.

STACK	A=3	B=5	C=8		

Kết xuất của chương trình:

Truoc khi goi thu tuc:

A=3 B=5 C=8

- * Trong khi thực hiện thủ tục:
- · Cấp vùng nhớ cho các biến cục bộ x,y,z,t,d.
- · Chuyển giao tham số: TestVar(a,5,c,c);

Các tham số x,y,z gọi là các tham trị. Việc chuyển giao giá trị cho các tham số này có thể được thực hiện bằng trị hoặc bằng biến, giá trị được chuyển giao sẽ được COPY vào ô nhớ tương ứng của các biến đó. Các ô nhớ ứng với x,y,z lần lượt có giá trị là 3,5,8.

Tham số T được khai báo sau từ khóa VAR được gọi là tham biến. Việc chuyển giao tham số chỉ có thể được thực hiện bằng biến. Ở đây ta đã chuyển giao biến C cho vị trí tham số T.

Pascal không copy giá trị của biến C vào ô nhớ ứng với T mà tạo một "con trỏ" để trỏ về C, mọi thao tác đối với T sẽ được thực hiện ở ô nhớ của C. Biến D sẽ được khởi tạo (lần đầu) bằng 0.

STACK	A=3	B=5	C=8	x=3
	y=5	z=8	T= (Trỏ về C)	d=0

Sau dòng lệnh {1} và {2} của thủ tục trong bộ nhớ sẽ là:

STACK	A=3	B=5+(3+4)	C=8+4	x=3+4
	Y=5	z=8	T= (Trỏ về C)	d=4

Kết xuất của chương trình khi chạy đến câu lệnh cuối của thủ tục là:

Truoc khi goi thu tuc:

A=3 B=5 C=8

Ben trong thu tuc:

A=3 B=12 C=12 D=4 X=7 Y=5 Z=8 T=12

- * Sau khi thực hiện thủ tục:
- Thu hồi các vùng nhớ đã được cấp cho thủ tục:

STACK	A=3	B=5+(3+4)	C=8+4	

Kết xuất của chương trình khi chạy đến câu lệnh cuối là:

Truoc khi goi thu tuc:

A=3 B=5 C=8

Ben trong thu tuc:

A=3 B=12 C=12 D=4 X=7 Y=5 Z=8 T=12

Sau khi goi thu tuc:

A=3 B=12 C=12

'M ộ v ấ đề cần nhớ:

Đối với tham trị có thể chuyển giao bằng trị hoặc bằng biến. Giá trị được chuyển giao được COPY vào nội dung ô nhớ của biến tham trị.

Đối với tham biến chỉ có thể chuyển giao bằng biến. Một con trỏ sẽ trỏ về biến chuyển giao, mọi thao tác sẽ được thực hiện trên biến chuyển giao.

'V à kết luận quan trọng:

Sự thay đổi của tham biến bên trong thủ tục sẽ làm thay đổi giá trị của biến chuyển giao (Trường hợp của biến C). Điều này không xảy ra đối với tham trị (Trường hợp của biến A, sự thay đổi của biến X không ảnh hưởng đến nội dung của ô nhớ A).

Sự thay đổi của biến chuyển giao trong trường hợp tham biến được gọi là hiệu ứng lề (Side effect). Người lập trình phải hết sức lqu ý để phòng ngừa hiệu ứng lề ngoài mong muốn.

Chương 5 Dữ liệu kiểu tập hợp, mảng và bản ghi

1. Kiểu tập hợp, các phép toán trên tập hợp

1.1. Định nghĩa và khai báo

Một tập hợp bao gồm một số phần tử có cùng kiểu gọi là kiểu phần tử. Số các phần tử tối đa trong tập hợp là 256, kiểu phần tử có thể là kiểu vô

hướng liệt kê, kiểu miền con hoặc kiểu char. Khái niệm tập hợp trong ngôn ngữ Pascal gắn liền với tập hợp trong toán học.

Ta có hai cách khai báo kiểu tập hợp.

Khai báo gián tiếp

```
Type CHUCAI = SET OF CHAR; {Tập các chữ cái}
CHUSO = SET OF 0..9;
NGAYTT = (CHUNHAT, HAI, BA, TU, NAM, SAU, BAY);
NGAY = SET OF NGAYTT;
Var cc: CHUCAI;
cs: CHUSO;
ng: NGAY;
```

Khai báo trực tiếp

```
Var cc: SET OF CHAR;
cs: SET OF 0..9;
ng: SET OF (CHUNHAT, HAI, BA, TU, NAM, SAU, BAY);
```

1.2. Mô tả một tập hợp và các phép toán trên tập hợp

1.2.1. Mô tả tập hợp

Một tập hợp được mô tả bằng cách liệt kê các phần tử của tập hợp, chúng cách nhau dấu , (phẩy) và được đặt trong cặp dấu [] (ngoặc vuông). Các phần tử có thể là hằng, biến, biểu thức. Cụ thể, ta xét các tập hợp dưới đây.

```
[]                      {Tập hợp rỗng}
[3..8]                  {Tập hợp chữ số từ 3 đến 8}
[0..50, 60, 70, 80, 90]
['A'..'G']              {Tập hợp chữ cái từ A đến G}
[i, i+j*2, 10, 12]      {i,j phải là biến số nguyên}
```

1.2.2. Các phép toán trên tập hợp

a. Phép gán

Ta có thể gán giá trị của tập hợp đã được mô tả vào các biến tập hợp cùng kiểu. Tập rỗng có thể gán cho mọi biến kiểu tập hợp.Với cách khai báo như trình bày ở trên, ta có thể thực hiện phép gán như dưới đây.

```
cc := [„a‟ .. „n‟];
cs := [1..5, 7, 9];
ngay := [];
```

b. Phép hợp

Hợp của hai tập hợp A và B là một tập hợp ký hiệu A + B có các phần tử bao gồm tất cả các phần tử của 2 tập hợp. Ta hãy xem ví dụ dưới đây.

```
A := [2 .. 9];
B := [7 .. 15];
```

Khi đó A + B = [2 .. 15];

Lưu ý: A + B = B + A

c. Phép giao

Giao của hai tập hợp A và B là một tập hợp ký hiệu A * B có các phần tử vừa của tập hợp A vừa của tập hợp B. Ta hãy xem ví dụ dưới đây.

```
A := [2 .. 9];
B := [7 .. 15];
```

Khi đó A * B = [7 .. 9];

Lưu ý: A * B = B * A

d. **Phép trừ**

Hiệu của hai tập hợp A và B là một tập hợp ký hiệu A - B có các phần tử của tập hợp A mà không có trong tập hợp B. Ta hãy xem ví dụ dưới đây.

A := [2 .. 9];

B := [7 .. 15];

Khi đó A - B = [2 .. 6];

e. **Phép trừ**

Phép thử IN cho phép ta xem một giá trị nào đó có thuộc tập hợp hay không. Nếu có cho kết quả là TRUE, ngược lại cho kết quả là FALSE. Ta hãy xem ví dụ dưới đây.

A := [2 .. 9];

i := 3;

i IN A {cho kết quả là True}

f. **Các phép toán so sánh =,<>,<=,>=**

Hai tập hợp muốn so sánh với nhau thì chúng phải có cùng kiểu phần tử. Kết quả của phép so sánh trả về kiểu Boolean (đúng – sai).

- Phép so sánh = (bằng). Hai tập hợp A và B bằng nhau (A = B cho kết quả True) khi chúng có các phần tử bằng nhau từng đôi một (không kể thứ tự các phần tử trong tập hợp).

- Ngược lại với phép so sánh bằng là so sánh khác, tức là các phần tử A và B không bằng nhau từng đôi một. Nếu A = B cho kết quả TRUE thì A <> B cho kết quả FALSE và ngược lại.

- Phép so sánh <= (nhỏ hơn hoặc bằng). A <= B cho kết quả là True khi mọi phần tử của A đều có trong B.

- Phép so sánh >= (lớn hơn hoặc bằng). A >= B cho kết quả là True khi mọi phần tử của B đều có trong A.

1.3. Viết và đọc dữ liệu trên tập hợp

Đối với dữ liệu kiểu tập hợp ta không thể viết ra hoặc đọc vào bằng các lệnh Read, Readln, Write, Writeln. Tuy nhiên ta có thể lập trình thực hiện các thao tác này.

Ví dụ sau sẽ nhập vào một tập hợp kiểu Char và in lên màn hình tập hợp vừa nhập.

```
Type CHUCAI = SET OF CHAR;
Var cc: CHUCAI;
        i, n: Integer;
        ch: Char;
Begin
        Write(‘ Tập hợp có bao nhiêu phần tử? ‘); Readln(n);
        cc := [];
         For i := 1 to n do Begin
                Write(‘ Phần tử thứ ‘ ,i, ’ là: ’);
                Readln(ch);
                cc:= cc + [ch];
                End;
         Writeln(‘Các phần tử trong tập hợp bạn vừa nhập’);
For ch := ‘A’ to ‘z’ do
                If ch IN cc then
```

```
        Write(ch: 5);
        Readln;
    End.
```

2. Khái niệm mảng, khai báo mảng, gán giá trị

2.1. Khái niệm

Mảng (Array) là một tập hợp các phần tử cố định có cùng kiểu gọi là kiểu phần tử. Kiểu phần tử có thể là kiểu vô hướng, kiểu String, kiểu tập hợp, kiểu Record. Đôi khi ta cũng dùng mảng để làm kiểu phần tử cho mảng, trường hợp này gọi là mảng của mảng.

2.2. Mảng một chiều

2.2.1. Khai báo:

***Cú pháp*: ARRAY[<Tập chỉ số>] OF <Kiểu phần tử>;**

Tập chỉ số phải là một kiểu miền con, kiểu vô hướng liệt kê, kiểu char hoặc kiểu boolean. Tuy nhiên. người ta thường dùng kiểu miền con các số nguyên là dễ hình dung nhất vì nó gần giống với khái niệm chỉ số trong toán học.

Có hai cách khai báo là khai báo gián tiếp và khai báo trực tiếp.

```
Type
        Vector = Array[1..10] of Integer;
Var
        vt: Vector ;
```

Hoặc ta khai báo như sau:

```
Var
```

vt: Array[1..10] of Integer;

Ta thấy cách cách khai báo trực tiếp thì ngắn hơn, nhưng trong một số trường hợp lại bất tiện, thậm chí không sử dụng được như khi truyền tham số cho các chương trình con.

Ta lấy ví dụ sau:

Procedure THUTUC(A: Array[1..20] of Integer);

Thủ tục này sai vì ta không thể truyền tham số cho nó. Ta phải viết:

Type MANG = Array[1..20] of Integer;

Procedure THUTUC(A: MANG);

2.2.2. **Truy xuất các phần tử trong mảng**

Mỗi phần tử của mảng được truy xuất thông qua tên biến mảng cùng với chỉ số

của mảng trong cặp dấu []. Ta hãy xét ví dụ dưới đây.

Khai báo và hình ảnh diễn giải dưới đây, giúp chúng ta dễ hiểu hơn.

Type MANG = Array[1..10] of Integer;

Var A: MANG;

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

Ví dụ: Viết chương trình nhập vào một mảng và in ra mảng đó sau khi xắp xếp các phần tử của mảng tăng dần.

```
Uses Crt;
Type MANG = Array[1..50] of Integer;
    Var A: MANG;
     i, j, n, tam: Integer;
Begin
```

```
        Write('Bạn nhập bao nhiêu phần tử: '); Readln(n);
        {Nhập n phần tử}
        For i := 1 to n do Begin
            Write('Phần tử ' , i , ' là: ');
            Readln(A[i]);
                End;
        {Sắp xếp tăng dần}
        For i := 1 to n-1 do
            For j := i + 1 to n do
If A[i] > A[j] then Begin
                    tam := A[i];
                    A[i] := A[j];
                    A[j] := tam;
                        End;
{In các phần tử của mảng ra}
        For i:=1 to n do
            Write(A[i]:10);
            Readln;
    End.
```

Chú ý: Hai mảng A và B có cùng số phần tử và cùng kiểu phần tử, ta có thể thay toàn bộ phần tử A bởi các phần tử tương ứng của B bằng một phép gán A := B.

2.3. Mảng nhiều chiều

2.3.1. Khai báo

ARRAY[<Tập chỉ số 1> , <Tập chỉ số 1>] OF <Kiểu phần tử>;

Ví dụ ta có thể khai báo:

```
Type MANG = Array[1..20,1..20] of Integer;
Var A: MANG;
```

Hoặc khai báo:

```
Var A: Array[1..20,1..20] of Integer;
```

Mảng hai chiều có thể khai báo như là mảng một chiều của mảng một chiều, ta có thể khai báo nhưu sau:

```
Type KieuPhantu = Array[1..20] of Integer;
Var A: Array[1..20] of KieuPhantu;
```

2.3.2. **Truy xuất các phần tử của mảng**

Mảng hai chiều tổ chức như một ma trận, các phần tử của ma trận cũng tương tự như các phần tử của mảng hai chiều. Ta truy xuất các phần tử của mảng hai chiều thông qua tên biến, theo sau là cặp chỉ số cách nhau bởi dấu , (phẩy) hoặc hai cặp dấu [][]. Ví dụ: A[3,2] hoặc A[2][3].

Ta có thể hình dung mảng A: Array[1..4, 1..5] như sau.

A[1,1]	A[1,2]	A[1,3]	A[1,4]	A[1,5]
A[2,1]	A[2,2]	A[2,3]	A[2,4]	A[2,5]
A[3,1]	A[3,2]	A[3,3]	A[3,4]	A[3,5]
A[4,1]	A[4,2]	A[4,3]	A[4,4]	A[4,5]

Ví dụ: Nhập vào một ma trận số nguyên rồi in ma trận đó theo dạng toán học.

```
Type MANG = Array[1..20,1..20] of Integer;
Var A: MANG;
 i, j, n, m: Integer;
Begin
```

```
Write('Ma trận có bao nhiêu dòng: '); Readln(n);
Write('Ma trận có bao nhiêu cột: '); Readln(m);
    {Nhập vào mảng hai chiều}
    For i := 1 to n do
For j := 1 to m do Begin
                Write('Phần tử A[ ' ,i, ' , ' ,j, '] là: ');
                Readln(A[ i , j ]);
            End;
{In các phần tử ra như một ma trận}
For i := 1 to n do Begin
    For j := 1 to m do
        Write(A[i,j]:10);
        Writeln;
    End;
Readln;
End.
```

3. Kiểu bản ghi

3.1. Khái niệm và khai báo

Chúng ta đã học các cấu trúc dữ liệu như mảng (Array), tập hợp (Set). Các kiểu dữ liệu này là một tập hợp có cùng kiểu do đó kả năng sử dụng các kiểu dữ liệu này còn hạn chế. Vấn đề đặt ra là có một kiểu cấu trúc dữ liệu có nhiều phần tử khác kiểu nhau nhưng lại liên quan đến nhau, người ta đã định nghĩa kiểu mẩu tin (RECORD) để đáp ứng vấn đề đó.

Cấu trúc dữ liệu kiểu RECORD được gắn liền với cấu trúc dữ liệu kiểu FILE (được trình bày chương sau) để lưu trữ dữ liệu. Dĩ nhiên Pascal cũng cho phép sử dụng RECORD độc lập với FILE.

Khai báo dữ liệu kiểu RECORD bắt đầu là chữ RECORD tiếp theo là danh sách các phần tử của Record gọi là các trường (Fields), mỗi trường có tên trường, kiểu trường. Kết thúc khai báo Record là từ khoá END; (End và chấm phẩy). Ta xem cú pháp khai báo dưới đây.

***Cú pháp*: <Tên kiểu> = RECORD**
<Tên trường> [, <Tên trường>]:<Kiểu trường>
<Tên trường> [, <Tên trường>]:<Kiểu trường>
.....
END;

Ta hãy xét ví dụ dưới đây. Khai báo Record là các thuộc tính của học sinh.

```
Type  HOCSINH = RECORD
Holot: String[30];
Ten: String[10];
Lop: Byte;
Diachi: String;
                    END;
Var  hsa, hsb, hsc: HOCSINH;
Lop10A: Array[1..50] of HOCSINH;
```

3.2. Truy xuất một bản ghi

Để truy xuất một trường của Record, ta cần dùng tên biến kiểu Record sau đó là dấu . (chấm) rồi đến trường. Ví dụ ta sẽ nhập vào và in ra các thuộc tính của học sinh a (hsa) đã khai báo trên.

```
Type  HOCSINH = RECORD
            Holot: String[30];
            Ten: String[10];
            Lop: Byte;
            Diachi: String;
  END;
        Var  hsa: HOCSINH;
Begin
  Write(‘ Nhập vào họ và các chữ lót: ‘); Readln(hsa.Holot);
  Write(‘ Nhập vào tên: ‘); Readln(hsa.Ten);
  Write(‘ Nhập vào lớp: ‘); Readln(hsa.Lop);
  Write(‘ Nhập vào địa chỉ: ‘); Readln(hsa.Diachi);
  Writeln(‘ Thông tin về học sinh bạn vừa nhập ’);
  Writeln(‘ Họ và tên: ’ , hsa. Holot,’  ‘, hsa.Ten);
  Writeln(‘ Lớp: ’ , hsa.Lop);
  Writeln(‘ Địa chỉ: ’ , hsa.Diachi);
  Readln;
End.
```

Ta lưu ý lệnh hsb := hsa là sao chép toàn bộ hsa vào hsb. Đây là việc truy xuất vào toàn bộ biến kiểu Record chứ không riêng lẻ một trường nào cả.

3.3. Câu lệnh With … do

Như trên ta thấy việc truy xuất một trường biến kiểu Record phải thông qua tên và dấu chấm, làm phức tạp thêm chương trình, giải quyết bớt phần nào sự hức tạp này, Pascal đưa ra câu lệnh With … do ta hãy viết chương trình nhập học sinh của lớp 10A đã khai báo ở trên, có dùng lệnh With … do.

```
Var  Lop10A: Array[1..50] of HOCSINH;
i, n: Integer;
Begin
        Write(‘ Lớp có bao nhiêu học sinh? ‘); Readln(n);
        For i:=1 to n do
                With Lop10A[i] do Begin
                        Write(‘ Nhập họ và các chữ lót: ‘); Readln(Holot);
                        Write(‘ Nhập vào tên: ‘); Readln(Ten);
                        Write(‘ Nhập vào lớp: ‘); Readln(Lop);
                        Write(‘ Nhập vào địa chỉ: ‘); Readln(Diachi);
End;
Writeln(‘ Thông tin về các học sinh bạn vừa nhập ’);
        For i:=1 to n do
        With Lop10A[i] do Begin
                Writeln(‘ Họ và tên: ’ , Holot,’  ‘, Ten);
                Writeln(‘ Lớp: ’ , Lop);
                Writeln(‘ Địa chỉ: ’ , Diachi);
                End;
Readln;
End.
```

Chương 6 Dữ liệu kiểu chuỗi

1. Khai báo

Khai báo kiểu String (chuỗi) có hai cách là khai báo gián tiếp (khai báo kiểu rồi mới khai báo biến) và khai báo trực tiếp. Độ dài tối đa của một biến kiểu String là 255 ký tự, tuy nhiên ta có thể giảm độ dài tối đa của chuỗi khi khai báo biến để tiết kiệm ô nhớ. Việc khai báo thực hiện như dưới đây.

Khai báo gián tiếp

```
Type  KIEU = String[20];          {Độ dài tối đa là 20}
Var    st: KIEU;
```

Khai báo trực tiếp

```
Var    st: String[20];          {Độ dài tối đa là 20}
       maxst: String;           {Độ dài tối đa là 255}
```

2. Các thao tác trên chuỗi

2.1. Các phép toán và hàm trên kiểu chuỗi

a. Phép gán:

Phép gán được tiến hành bình thường như một phép gán trong các kiểu vô hướng khác. Tuy nhiên cần lưu ý là hằng String nằm trong cặp dấu ‘ ’ (nháy đơn).

Ví dụ: Hoten := ‘Nguyễn Văn Thành’;

b. Phép cộng

Là phép ghép chuỗi thứ nhất với chuỗi thứ hai.

Ví dụ: Hoten := ‘Nguyễn Văn ’ + ‘Thành’;

c. So sánh chuỗi

Khi so sánh hai chuỗi ký tự thì các ký tự được so sánh từng cặp một trái qua phải theo bảng mã ASCII. Các khả năng có thể xảy ra như sau:

- Hai chuỗi hoàn toàn giống nhau thì bằng nhau (‘ABC’=’ABC’).

- Tính từ trái qua phải, chuỗi nào có ký tự đầu tiên khác nhau nhỏ hơn thì nhỏ
hơn (‘ABCDEF’<’ABCFGH’).

- Một chuỗi có độ dài bé hơn chuỗi kia mà nó hoàn toàn giống đầu của chuỗi kia
thì nó nhỏ hơn (‘ABC’<’ABCDEF’).

d. Thủ tục Read và Readln

Hai thủ tục này có tác dụng đối với chuỗi cũng tương tự như đối với các kiểu vô hướng chuẩn khác. Tuy nhiên cần có một số lưu ý:

- Nếu đọc một lúc nhiều biến kiểu Read(biến 1, biến 2,…, biến n) thì dễ bị nhầm

lẫn. Cụ thể là nếu giá trị nhập vượt qua độ dài tối đa của biến một thì phần vượt qua đó mới gán cho biến 2, ngược lại máy sẽ lấy tất cả các ký tự (kể cả khoảng trắng) để gán cho biến một, khi đủ độ dài của biến một thì mới gán cho biến hai. Do đó, biến kiểu String tốt nhất là mỗi lần nhập chỉ một biến.

- Mặc dù chiều dài tối đa của chuỗi là 255 ký tự song việc nhập một chuỗi từ bàn phím theo lệnh Read hoặc Readln chie cho phép đọc tối đa 127 ký tự.

- Mặc dù ta có khai báo độ dài chuỗi nhưng độ dài thực tế là độ dài nhập từ bàn phím. Nếu khi nhập chuỗi ta chỉ khỏ phím Enter mà không khỏ bất kỳ ký tự nào thì chuỗi sẽ rỗng (st=’’).

e. Thủ tục Write và Writeln

Hai thủ tục này có tác dụng đối với chuỗi cũng tương tự như đối với các kiểu vô hướng chuẩn khác. Tuy nhiên cần có một số lưu ý:

- Cách viết không qui cách Write(st) hoặc Writeln(st) thì mỗi ký tự sẽ chiếm một

vị trí.

- Cách viết có qui cách Write(st:n) hoặc Writeln(st:n) thì máy sẽ dành n vị trí để viết chuỗi st, vvậy chuỗi sẽ được viết canh trái nếu n<0, canh phải nếu n>0.

- Nếu viết thẳng một hằng chuỗi ký tự mà trong đó có dấu ‘ (nháy đơn), chẳn hạn

câu tiếng anh: I’m a student thì ta phải dùng 2 nháy đơn liên tiếp ‘’ tại chổ đó.

Lưu ý: Là 2 nháy đơn chứ không phải nháy kép. Vậy khi lập trình ta phải viết câu

đó là: Write(‘I’’m a student’).

f. **Thủ tục Delete(St,Pos,n)**

Xóa khỏi chuỗi st n ký tự bắt đầu từ vị trí pos tính từ bên trái sang.

```
Var    St: String[20];
Begin
        St := ‘CHUOI CHUA BI CAT’;
        St := Delete(St,6,5);
       Write(St);
End.
```

Kết quả: CHUOI BI CAT

g. **Thủ tục Insert(Obj, St, Pos)**

Thêm chuỗi obj vào chuỗi st tại vị trí pos.

```
Var St, Obj: String[20];
Begin
        St := ‘CHUOI THEM’;
        Obj := ‘DA ’;
        Insert(obj,St,7);
        Write(St);
End.
```

Kết quả: CHUOI DA THEM

h. **Hàm Str(S[:n[:m]], St)**

Đổi giá trị S thành chuỗi rrồi gán cho st, số n, m nếu có sẽ là vị trí số chữ số
phần nguyên và thập phân của S.

```
Var St: String[20];
 S: Real;
Begin
S := 987987987;
 Str(S:9:0,St);
 Write(St);
End.
```

Kết quả: 987987987

i. **Thủ tục Val(St, S, Code)**

Đổi chuỗi St thành số và gán cho S, Code là một biến kiểu Integer. Nếu đổi đúng thì Code nhận giá trị 0, nếu sai so St không biểu diễn dạng số nguyên hay số thực thì Code nhận giá trị bằng vị trí của ký tự sai trong chuỗi St.

```
Var St: String[20];
 X: Real;
Code: Integer;
Begin
 St := '789.789';
 Val(St, X, Code);
 Writeln('X = ' ,X, ' ; Code = ' , Code);
```

```
{Kết quả: X=798.798 - Code = 0}
St := '789A789';
Val(St, X, Code);
Writeln('X = ' ,X, ' ; Code = ' , Code);
{Kết quả: X = 0 - Code = 4}
End.
```

j. **Hàm Length(st)**

Cho kết quả là một số nguyên chỉ độ dài của chuỗi (số ký tự của chuỗi).

Ví dụ để viết một dòng ở giữa màn hình ta làm như sau:

```
GotoXY((80-Length(st)) div2,12); Write(st);
```

k. **Hàm Copy (St, Pos, n)**

Kết quả trả về của hàm là một chuỗi, trích từ chuỗi St, chéptừ ví trí Pos và chép n ký tự.

```
Var St, Obj: String[20];
Begin
    St := 'TURBO PASCAL 7.0';
    Obj := Copy(st,7,6);
    Write(Obj);
End.
    Kết quả: PASCAL
```

l. **Hàm Concat(St1,St2,St3…Stn)**

Cho kết quả là một chuỗi mới được ghép từ các chuỗi St1, St2, St3,…, Stn theo thứ tự truyền vào hàm. Kết quả này giống như phép cộng chuỗi.

m. **Hàm Pos(Obj,St)**

Cho kết quả là một vị trí đầu tiên của Obj trong chuỗi St. Nếu không tìm thấy thì hàm trả về kết quả là 0.

```
Var St, Obj: String[20];
Begin
    St := 'TURBO PASCAL 7.0';
    Obj := 'PASCAL';
    Write(Pos(Obj, St));
End.
```

Kết quả: 7

2.2. Truy xuất từng ký tự trong chuỗi

Ta có thể truy xuất từng ký tự trong chuỗi thông qua tên biến, nó tương tự như việc truy xuất một mảng, dĩ nhiên kiểu của từng ký tự trong chuỗi là Char. Giả sử ta có biến St là biến kiểu string thì St[i] (i là một số nguyên thoả 1 <= i <= length(St)) là ký tự thứ i của chuỗi.

Dưới đây là chương trình cho nhập vào một số nhị phân, in ra kết quả là số thập phân tương ứng, sử dụng việc truy xuất các ký tự trong chuỗi nhị phân Bin. HamMu là hàm mũ a^n (xin xem cách thiết kế ở chương unit).

```
Var Bin: String[20];
Dec, i: Integer;
Begin
    Write(' Nhập một số nhị phân: '); Readln(Bin);
Dec := 0;
For i := 1 to length(Bin) do
    If Bin[i] = '1' Then
```

```
        Dec := Dec + HamMu( 2, length(Bin)-i );
        Write('Số nhị phân vừa nhập có giá trị là: ' , Dec);
        Readln;
End.
```