Giáo trình Pascal 7.0

Biên tập bởi:
Võ Thanh Ân

Giáo trình Pascal 7.0

Biên tập bởi:

Võ Thanh Ân

Các tác giả:

Võ Thanh Ân

Phiên bản trực tuyến:
http://voer.edu.vn/c/f83d08c4

MỤC LỤC

GIỚI THIỆU NGÔN NGỮ PASCAL VÀ BORLAND PASCAL 7.0

GIỚI THIỆU NGÔN NGỮ PASCAL VÀ BORLAND PASCAL 7.0

GIỚI THIỆU NGÔN NGỮ PASCAL.

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.

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:

- Cập nhật file TURBO.TPL trong thư mục \BP\BIN.
- Sử dụng Free Pascal.
  Gói phần mềm này có thể download miễn phí từ Internet (từ khóa: Free Pascal) hoặc hỏi chép từ Tổ bộ môn CNTT.

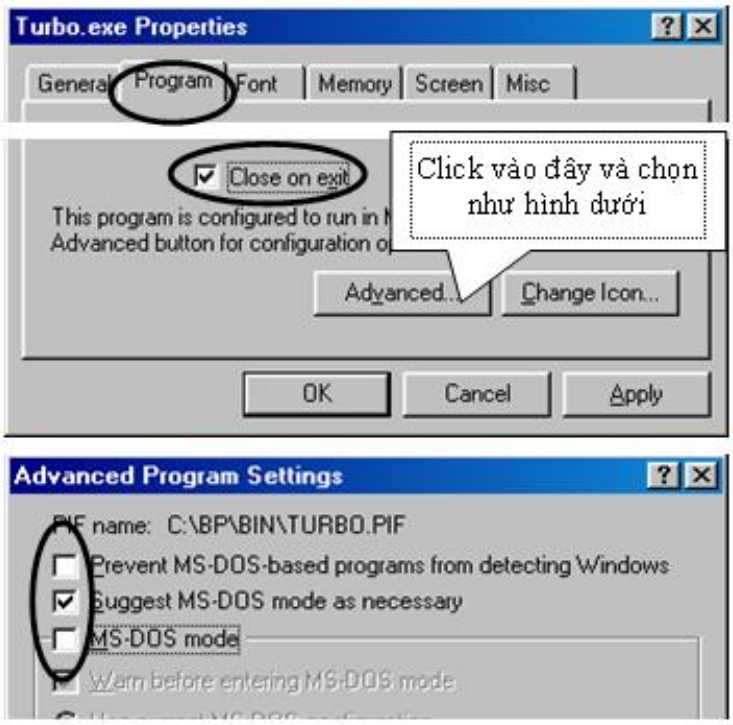
Ngoài ra cũng nên lưu ý là Turbo Pascal chạy ở chế độ thực (real mode) nên khi chạy trên nền Windows XP nó hay khởi động lại máy. Nên chạy Borland Pascal. Khi đó Windows sẽ tạo một môi trường DOS giả lập và chạy ở chế độ đa nhiệm tiện lợi hơn.

SỬ DỤNG PASCAL 7.0

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.

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.
φ	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.
φBack	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.

CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ PASCAL

Bộ Chữ Viết – Từ Khoá – Tên

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).

Từ khoá

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

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 "**Begi**n" 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.

Hằng – Kiểu – Biến

Hằng (Constant)

Hằng là một đại lượng không đổi trong quá trình thực hiện chương trình. Có hai loại hằng là hằng chuẩn và hằng do người dùng định nghĩa.

- Hằng chuẩn là hằng do Pascal định sẵn, ví dụ hằng số PI, hằng số chỉ màu RED=4,… Người sử dụng không cần định nghĩa lại nếu thấy không cần thiết. Các hằng này được Pascal định nghĩa sẵn trong các Unit. Cần tham khảo hướng dẫn (help) đối với mỗi Unit để biết trong Unit có các hằng nào đã được định nghĩa..
- Hằng do người dùng định nghĩa thông qua việc khai báo. Cú pháp:

CONST <Tên hằng> = <Giá trị hằng>; [<Tên hằng> = <Giá trị hằng>;]

Ví dụ: Const A = 50;

Ch = 'K';

D = true;

Kiểu

Một kiểu dữ liệu là một tập hợp các giá trị mà một biến thuộc kiểu đó có thể nhận được *và* một tập hợp các phép toán có thể áp dụng trên các giá trị đó. Có hai loại kiểu là kiểu chuẩn và kiểu do người dùng định nghĩa.

- Kiểu chuẩn là kiểu Pascal định nghĩa sẵn: REAL, INTEGER, CHAR…
- Kiểu do người lập trình định nghĩa thông qua việc khai báo kiểu. Cú pháp:

TYPE <Tên kiểu> = <Kiểu>; [<Tên kiểu> = <Kiểu>;]

Ví dụ: TYPE NguyenDuong = 1..MaxInt;

MaTran = [1..10,1..10] of Integer;

Biến

Biến là một ô nhớ trong bộ nhớ của máy tính, giá trị của biến có thể thay đổi trong quá trình thực hiện chương trình, biến sẽ được giải phóng (thu hồi ô nhớ) khi chương trình kết thúc.

Chương trình quản lý biến thông qua tên biến và mỗi biến tương ứng với một kiểu dữ liệu nhất định.

VAR <Tên biến>[<, Tên biến>]: <Kiểu>; [<Tên biến>[<,Tên biến>]: <Kiểu>;]Biến trước khi sử dụng phải được khai báo. Cú pháp:

Ví dụ: VAR a, b, c: Integer;

X, Y: Real;

I, J: NguyenDuong; {Đã định nghĩa trước}

Biểu Thức – Dấu Chấm Phẩy – Lời Giải Thích

Biểu thức

Là một phần của câu lệnh bao gồm hằng, biến, hàm được liên kết với nhau bằng các phép toán và các dấu ngoặc đơn ( ).

Ví dụ: (-b + sqrt(delta))/(2*a)

Thứ tự thực hiện các phép toán trong một biểu thức như sau:

- Các thành phần trong cặp ngoặc trong cùng được thực hiện trước rồi tới các thành phần trong cặp ngoặc phía ngoài kế tiếp.
- Các phép toán nhân (*) và chia (/) (có cùng mức ưu tiên) và được thực hiện trước so với các phép toán cộng (+) và trừ (-) (có cùng mức ưu tiên). Ví dụ như trong (x*y –z) phép nhân sẽ được thực hiện trước phép trừ.
- Nếu hai phép toán liên tiếp có cùng mức ưu tiên thì thứ tự thực hiện là từ trái qua phải. Ví dụ như trong (x*y/z) phép nhân sẽ được thực hiện trước.
  Lưu ý: Trong lập trình hai biểu thức (x*y/z) và (x/z*y) không phải bao giờ cũng cho cùng kết quả.
- Riêng đối với biểu thức gán thì thứ tự thực hiện là từ phải qua trái.

Dấu chấm phẩy

Dấu chấm phẩy (;) dùng để ngăn cách giữa các câu lệnh. Sau một câu lệnh phải có dấu chấm phẩy (trừ một vài trường hợp đặt biệt).

Ví dụ: Write(‘Nhap so a:’)**;** Readln(a)**;**

Lời giải thích

Trong khi lập trình nhiều lúc cần phải đưa vào lời giải thích, nhằm diễn giải công việc mà đoạn chương trình đó thực hiện, làm cho người đọc chương trình dễ hiểu. Dĩ nhiên, việc thêm lời giải thích này không làm ảnh hưởng đến việc thực thi và kết quả chương trình. Lời giải thích có thể đặt bất cứ vị trí nào trong chương trình, nhưng phải nằm trong cặp dấu { và } hoặc (* và *).

Ví dụ: {Day la phan giai thich}

Cấu Trúc Của Một Chương Trình Pascal

Một chương trình Pascal gồm 2 phần chính: Phần khai báo và phần thân chương trình. Khi thực thi, chương trình Pascal sẽ thực thi tuần tự từng lệnh một theo như thứ tự đã được viết, trừ khi gặp các cấu trúc điều khiển rẻ nhánh hoặc lặp, bắt đầu từ thân chương trình chính.

Phần khai báo

Phần khai báo có thể có các mục sau:

- Tên chương trình PROGRAM <Tên chương trình>;
- Khai báo sử dụng unit USES <Tên Unit>[,<Tên Unit>];

- Khai báo nhãn LABEL <Tên nhãn>[,<Tên nhãn>];
- Khai báo hằng CONST
- Khai báo kiểu TYPE
- Khai báo biến VAR
- Khai báo chương trình con (sẽ trình bày phần sau).

Phần thân chương trình

Bắt đầu bằng từ khoá **BEGIN** và kết thúc bằng từ khoá **END.** (end và dấu chấm). Giữa BEGIN và END. là các câu lệnh.

Ví dụ:

PROGRAM MyFirstProg;

VAR i: Integer;

BEGIN

{Các câu lệnh viết ở đây}

END.

CÁC KIỂU VÔ HƯỚNG CHUẨN VÀ CÁC CÂU LỆNH ĐƠN

CÁC KIỂU VÔ HƯỚNG CHUẨN VÀ CÁC CÂU LỆNH ĐƠN

CÁC KIỂU VÔ HƯỚNG CHUẨN

Các Kiểu Vô Hướng Chuẩn (Standard scalar types)

Kiểu vô hướng (scalar type) là kiểu dữ liệu gồm một tập các giá trị của nó sắp xếp theo một thứ tự tuyến tính. Kiểu vô hướng chuẩn (Standard scalar type) là kiểu vô hướng do Pascal định nghĩa sẵn. Dưới đây là danh sách các kiểu vô hướng chuẩn cùng với miền giá trị và kích thước mà mỗi kiểu chiếm trong bộ nhớ.

Stt	Kiểu	Kích thước	Miền xác định
	Boolean	1 byte	FALSE..TRUE
	Char	1 byte	256 ký tự của bảng mã ASCII.
	Shortint	1 byte	-128..127
	Byte	1 byte	0..255
	Integer	2 byte	-32768..32767
	Word	2 byte	0..65535
	Longint	4 byte	-2147483648..2147483647
	Real	6 byte	2.9E-39..1.7E+38
	Single	4 byte	1.5E-45..3.4E+38
	Double	8 byte	5.0E-324..1.7E+308
	Extended	10 byte	3.4E-4932..1.1E+4932
	Comp	8 byte	-9.2E-18..9.2E+18

Trong đó 7 kiểu đầu gọi là kiểu đếm được (ordinal type), còn các kiểu sau là không đếm được.

Một Số Phép Toán Trên Các Kiểu

Các phép toán trên kiểu số

Các phép toán này rất gần gũi với chúng ta, do chúng ta sử dụng chúng hằng ngày trong đời sống.

Phép toán	Ý nghĩa	Kiểu đối số	Kiểu trả về	Ví dụ
?	Lấy đối số	Số nguyên, số thực	Giống đối số	Đối số của 2 là -2
+	Cộng	Số nguyên, số thực	Giống đối số	10 + 9 ? 19
?	Trừ	Số nguyên, số thực	Giống đối số	10 - 9 ? 1
*	Nhân	Số nguyên, số thực	Giống đối số	10*9 ? 90
/	Chia	Số nguyên, số thực	Số thực	10 / 4 ? 2.5
Div	Chia lấy phần nguyên	Số nguyên	Số nguyên	10 div 3 ? 3
Mod	Chia lấy phần dư	Số nguyên	Số nguyên	10 mod 3 ? 1

Một Số Hàm Số

Dưới đây là một số hàm được Pascal thiết kế sẵn. Người sử dụng có thể gọi và sử dụng chúng mà không cần phải khai báo unit qua câu khai báo USES.

Thật ra chúng thuộc về Unit SYSTEM.TPU

Hàm	Ý nghĩa	Kiểu đối số	Kiểu trả về	Ví dụ
ABS(x)	Trị tuyệt đối x	Số nguyên, số thực	Giống đối số	Abs(-2) ? 2

SQR(x)	Bình phương x	Số nguyên, số thực	Giống đối số	Sqr(2) ? 4
SQRT(x)	Căn bậc hai x	Số nguyên, số thực	Số thực	Sqrt(9) ? 3
EXP(x)	Hàm e^x	Số nguyên, số thực	Số thực	Exp(3) ? ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***
LN(x)	Hàm ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***	Số nguyên, số thực	Số thực	Ln(2) ? ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***
SIN(x)	Hàm lượng giác	Số nguyên, số thực	Số thực	Sin(PI) ? 0
COS(x)	Hàm lượng giác	Số nguyên, số thực	Số thực	Cos(PI) ? 1
ARCTAN(x)	Hàm lượng giác	Số nguyên, số thực	Số thực	Arctan(1) ? $\frac{\pi}{4}$
SUCC(x)	Succ(x) ? x + 1	Số nguyên	Số nguyên	
PRED(x)	Pred(x) ? x ?1	Số nguyên	Số nguyên	
ROUND(x)	Làm tròn	Số thực	Số nguyên	Round(8.6) ? 9
TRUNC(x)	Làm tròn	Số thực	Số nguyên	Trunc(8.6) ? 8
ORD(x)	Lấy mã ASCII	Ký tự	Số nguyên	Ord(‘a’) ? 97
CHR(x)	ký tự ? mã ASCII	Số nguyên	Ký tự	Chr(65) ? ‘A’
ODD(x)	Kiểm chẳn lẽ	Số nguyên	Logic	Odd(5) ? True

Các phép toán logic

Các phép toán logic, toán hạng của nó phải là một kiểu Boolean. Toán hạng cũng như các kết quả của phép toán chỉ nhận 1 trong 2 giá trị: hoặc là TRUE hoặc là FALSE (không có giá trị khác).

Các toán tử logic tác động lên kiểu Boolean, cho kết quả là kiểu Boolean AND (và), OR (hoặc), XOR, NOT (phủ định). Sau đây là bảng chân trị của các toán tử này.

Toán hạng X	**Toán hạng Y**	**X OR Y**	X AND Y	**X XOR Y**	**NOT X**
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE

Mở rộng:

Các phép toán lôgic còn áp dụng được cho kiểu số nguyên, trên cơ sở biểu diễn nhị phân của số nguyên đó. Ví dụ xét hai số nguyên X và Y lần lượt bằng 10 và 22, thuộc kiểu byte. Biểu diễn nhị phân của X là 0000 1010 và của Y là 0001 0110. Khi đó phép toán được thực hiện theo thứ tự từng bit như sau:

X 0 0 0 0 1 0 1 0

Y 0 0 0 1 0 1 1 0

X AND Y 0 0 0 0 0 0 1 0

Vậy (10 AND 22) cho kết quả là 2

X 0 0 0 0 1 0 1 0

Y 0 0 0 1 0 1 1 0

X OR Y 0 0 0 1 1 1 1 0

Vậy (10 OR 22) cho kết quả là 30

X 0 0 0 0 1 0 1 0

Y 0 0 0 1 0 1 1 0

X XOR Y 0 0 0 1 1 1 0 0

Vậy (10 XOR 22) cho kết quả là 28

Còn có hai phép toán bit nữa là SHIFT LEFT và SHIFT RIGHT, lần lượt được kí hiệu là SHL và SHR. Phép toán SHL làm đẩy các bit lên một số vị trí về bên trái và thêm các giá trị 0 vào các bit tận cùng bên phải. Cú pháp:

<Biểu thức nguyên> SHL <sốbit>

Ví dụ:

X 0 0 0 0 1 0 1 *0*

X SHL 1 0 0 0 1 0 1 *0 0* **{Đẩy về bên trái 1 bit}**

X SHL 2 0 0 1 0 1*0 0 0* **{Đẩy về bên trái 2 bit}**

Vậy (10 SHL 1) cho kết quả 20

(10 SHL 2) cho kết quả 40

Thực hiện tương tự đối với phép toán SHR

CÂU LỆNH

Khái Niệm Về Một Câu Lệnh

- Một câu lệnh đơn xác định một công việc mà chương trình phải thực hiện để xử lý các dữ liệu đã được mô tả và khai báo. Các câu lệnh được phân cách bởi dấu ; (chấm phẩy). Dấu ; có tác dụng ngăn cách giữa các câu lệnh, nó không thuộc vào câu lệnh.

Ví dụ:

CLRSCR; {Xóa màn hình}

Writeln('Nhap vao day mot so nguyen:'); {Thông báo nhập liệu}

Readln(SoNguyen); {Chờ nhập liệu}

Writeln('Binh phuong cua no la: ',SoNguyen*SoNguyen); {Kết xuất}

- Câu lệnh hợp thành: Nếu trong chương trình có nhiều câu lệnh liên tiếp cần được xử lí và xem như một câu lệnh đơn chúng ta cần bao nó giữa hai từ khóa BEGIN và END;
- Câu lệnh có cấu trúc: Bao gồm cấu trúc rẽ nhánh, cấu trúc điều kiện chọn lựa, cấu trúc lặp. *Mỗi câu lệnh có cấu trúc xác định một câu lệnh tương đương một câu lệnh đơn.* Trong câu lệnh có cấu trúc có thể chứa nhiều câu lệnh hợp thành.

Ví dụ:

….	
Writeln(‘Cho biet so tuoi:’);	Câu lệnh đơn.
Readln(Tuoi);	Câu lệnh đơn
IF (Tuoi<4) THEN	
Writeln(‘Ban con be qua. Chua phuc vu duoc’)	
ELSE	
Begin	
Write(‘ Ban chon mon an nao:’);	
Readln(MonAn);	
End;	
Câu lệnh hợp thành từ hai câu lệnh đơn	
Câu lệnh có cấu trúc, xem như một câu lệnh đơn.	
Writeln(‘Xin cho doi it phut!’);	Câu lệnh đơn.
…..	

Một Số Lệnh Đơn

Lệnh gán

Lệnh gán dùng để gán giá trị của một biểu thức vào một biến. Giá trị biểu thức khi tính xong sẽ được gán vào biến. Phép gán được thực hiện theo thứ tự từ phải qua trái. Dưới đây là cú pháp và ví dụ về lệnh gán.

Cú pháp: **<Tên biến> := <Biểu thức>**

Program LenhGan;

Tại vị trí này biến x có giá trị là 1. Biến y có giá trị là 2 trước khi thực hiện phép gán, và có giá trị 3 sau khi thực hiện phép gán.Var x, y, z: Integer;

Begin

x := 1;

y := 2;

Z có giá trị là 4 sau khi thực hiện phép gán y:=y+x;

z := x + y;

End.

Chú ý

- Khi một giá trị gán cho biến, nó sẽ thay thế giá trị cũ mà biến đã lưu giữ trước đó (biến sẽ nhận giá trị mới).
- Trong lệnh gán, biểu thức ở bên phải và biểu thức ở bên trái phép gán phải cùng kiểu dữ liệu. Nếu không sẽ có thông báo lỗi "Type Mismatch" khi biên dịch chương trình.
  Thực ra không nhất thiết như thế. Một số trường hợp gọi là **type casting** có thể xảy ra. Trong trường hợp trên nếu biến z kiểu Real thì biểu thức gán z:=x+y; vẫn chấp nhận được.

Lệnh viết dữ liệu ra màn hình

Để xuất dữ liệu ra thiết bị (mặc định là viết dữ liệu ra màn hình) Pascal có 3 mẫu viết sau:

- Write(Mục1, Mục2,…, MụcN);
- Writeln(Mục1, Mục2,…, MụcN);
- Writeln;

Trong đó Mục1, Mục2,…,Mục**N** là các mục cần viết (cần đưa ra màn hình). Có thể là một trong các loại dưới đây.

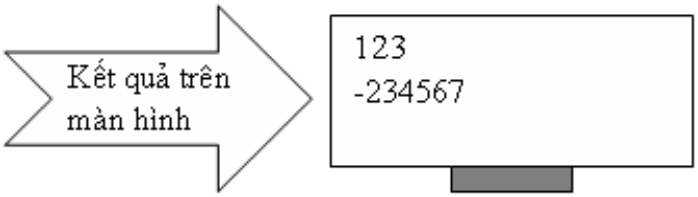
- Biến Write(i, j);
- Biểu thức Write(-c / (2*a));
- Hằng Write(PI);
- Giá trị kiểu vô hướng chuẩn Write(19, 29, True, 'a');
- Một chuỗi văn bản Write('How are you?');

Thủ tục Writeln; dùng để xuống dòng. Lệnh Writeln(Mục1, Mục2,…,Mục**n**); làm việc đơn giản là đặt con trỏ xuống đầu dòng tiếp theo. Do đó lệnh này tương đương với lệnh hợp thành: Begin Write(Mục1, Mục2,…,Mục**n**); Writeln; End;

- Viết kiểu số nguyên
- Viết không qui cách: Các số nguyên sẽ được viết ra với số chỗ đúng bằng với số chữ số mà nó cần có.

Ví dụ:

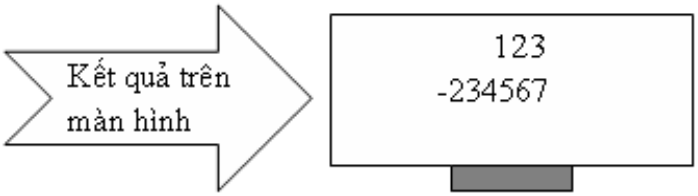
```
Var i: Integer;

Begin

i := 123;

Writeln( i );
```




```
Write(-234567);

End.
```

- Viết có qui cách: Ta bố trí số chỗ cố định để viết số nguyên, bằng cách thêm dấu hai chấm (**:**) theo sau là số chỗ để viết. Máy sẽ bố trí viết số nguyên từ phải sang trái (canh phải), nếu thừa sẽ để trống bên trái.

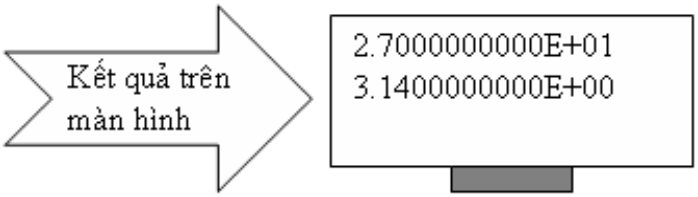
Ví dụ:




```
Var i: Integer;

Begin

i := 123;
```

```
Writeln( i :10);

Write(-234567:10);

End.
```

- Viết kiểu số thực



- Viết không qui cách: Số viết ra sẽ biểu diễn theo dạng dấu chấm động. Trong ví dụ dưới đây 2.7000000000E+01 chính là $2.7 * 10^{+01}$

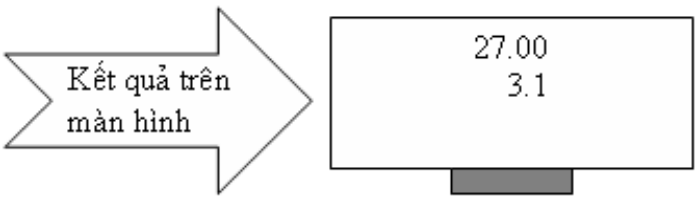
Ví dụ:

```
Var i: Real;

Begin

i := 27;

Writeln( i );

Write(3.14);

End.
```

- Viết có qui cách: Ta bố trí số **:** số chỗ cố định để viết số **:** số chỗ cho phần lẽ (thập phân). Máy sẽ bố trí viết số nguyên từ phải sang trái (canh phải), nếu thừa sẽ để trống bên trái.

Ví dụ:

```
Var i: Real;

Begin
```



```
i := 27;

Writeln( i :10:2);

Write(3.14:10:1);

End.
```

Lệnh đọc dữ liệu từ bàn phím

Là lệnh gán giá trị cho biến, giá trị này được nhập từ bàn phím khi chạy chương trình. Có 3 dạng như sau:

- Read(Biến1, Biến2,…, BiếnN);
- Readln(Biến1, Biến2,…, BiếnN);
- Readln;

Các cụm dữ liệu gõ từ bàn phím cho các biến được phân biệt với nhau bằng cách gõ phím khoảng trắng (Space Bar) ít nhất một lần (hoặc Enter). Kết thúc việc gán bởi phím Enter.

Read

Nên hiểu việc nhập liệu từ bàn phím như sau: Mỗi khi nhập dữ liệu từ bàn phím. Phải kết thúc việc nhập liệu bằng phím ENTER. Như vậy dữ liệu sẽ được đưa vào máy tính trước tiên đến bộ đệm (buffer bàn phím). Vậy luôn luôn trong bộ đệm có tới hai thành phần: Dữ liệu và phím ENTER. READLN(Bien) xử lí dữ liệu *và* phím ENTER để đưa con trỏ xuống đầu dòng kế tiếp. READ(Bien) xử lí dữ liệu *mà không* xử lí phím ENTER. Vậy sau lệnh READ(Bien) trong buffer vẫn còn phím ENTER. Điều này gây ra "sự cố" khi ngay các câu lệnh sau đó có lệnh READLN hoặc lệnh chờ gõ một phím (READKEY), chương trình sẽ "chạy luôn" mà không dừng lại.
và Readln khác nhau ở chỗ là đối với Readln sau khi gõ Enter thì con trỏ xuống dòng tiếp theo, còn Read thì không. Nên dùng Readln đọc dữ liệu để dễ phân biệt trên màn hình.

Readln; là lệnh không đọc gì cả, chỉ chờ ta gõ phím Enter. Người dùng thường dùng Readln cuối chương trình trước End. để khi chương trình chạy xong, màn hình dừng lại cho ta xem, gõ Enter để về chế độ soạn thảo. Nói chung là khi gặp lệnh Readln; thì chương trình ngừng lại, đợi ta gõ Enter thì chương trình thực thi tiếp. Ta thường phải kết hợp giữa lệnh Write và Readln để việc nhập liệu rõ ràng.

Ta có thể nhập liệu mà không cần qua bàn phím. Tuy nhiên điều này sẽ được nói tới khi sinh viên học qua dữ liệu kiểu File.

LỆNH CÓ CẤU TRÚC

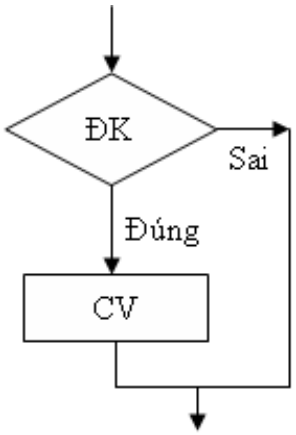
LỆNH CÓ CẤU TRÚC

LỆNH CẤU TRÚC RẼ NHÁNH

Dạng Không Đầy Đủ

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

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



Lưu đồ cú pháp (hình bên)

Ví dụ:

```
Var a,b: Integer;
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;
```

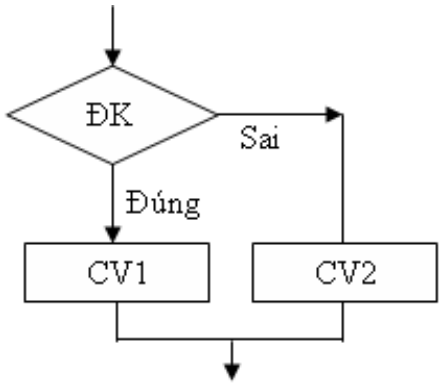
End.

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: Integer;

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);

Else

Write( ‘Không thể chia cho 0’ );

Readln;

End.
```

LỆNH CẤU TRÚC LỰA CHỌN

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 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: Integer;

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 (nhuần 29) ngày.');

End;

Readln;

End.
```

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: Integer;

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 (năm nhuần 29) ngày.' );

ELSE

Write( 'Tháng sai. Phải nhập số từ 1 đến 12.' );

End;

Readln;

End.
```

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

CÁC LỆNH VÒNG LẶP

Lệnh Lặp Với Số Lần Xác Định

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. Nếu 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 khoá FOR phải là biến đếm được và giá trị đầu phải <= giá trị cuối. Trong các lệnh của công việc không nên 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 từ 1, 2, 3, …, n ta có thể làm như sau:

```
Var i, n: Integer;

Begin

Write( ‘Nhập vào một số: ’); Readln(n);

Wrtieln( ‘Dưới đây là dãy số từ 1 đến số bạn vừa nhập’ );

For i := 1 To n Do

Write(‘ ’ , i);

Readln;

End.
```

Dạng 2

Ý nghĩa tương tự như 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: Integer;

Begin

Write( ‘Nhập vào một số: ’); Readln(n);

Wrtieln( ‘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 ngữ 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;

USES CRT, DOS;

Var Bien:byte; CtrlBreak: Boolean;

BEGIN

GetCBreak(CtrlBreak);

IF (CtrlBreak=FALSE) THEN CtrlBreak:=not CtrlBreak;

SetCBreak(CtrlBreak);
```

```
Writeln(‘ Phai go CTRL-Break moi cham dut dược!’);

For bien:=240 to 250 do

Begin

IF (bien=245) THEN bien:=252;

Writeln(‘Gia tri hien nay cua bien la: ‘, bien,#7);

Delay(100);

End;

END.
```

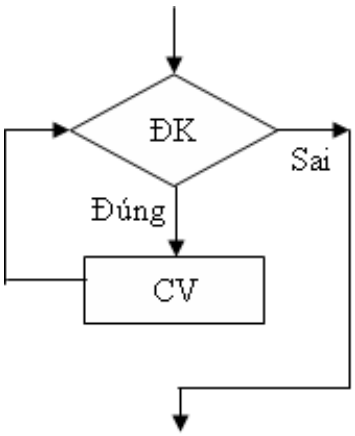
Giải thích:

- 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) kiểm tra 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 tình huống.
- #7 (Kí tự số 7) là mã ASCII làm xuất ra tiếng Beep của loa bên trong máy.
- Khi **bien** (điều khiển 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 **bien** đã duyệt qua hết phạm vi của kiểu dữ liệu (tức giá trị 255) thì **bien** quay lai giá trị 0 … và mọi thứ lại tiếp tục …trừ khi gõ Ctrl - Break.

Lệnh Lặp Với Số Lần Lặp Không Xác Định

Dạng 1

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



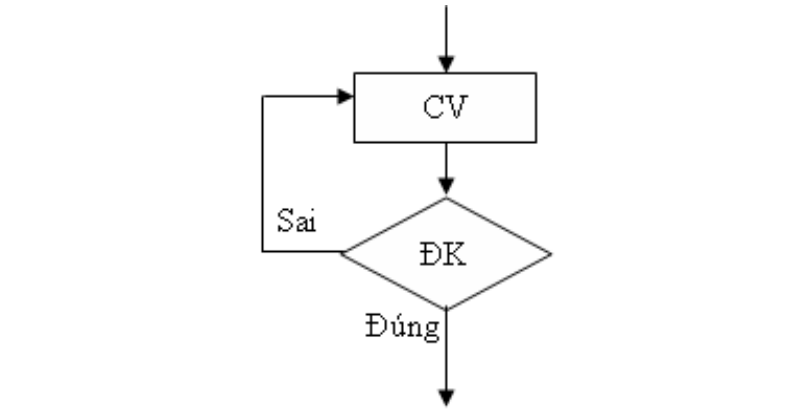
Ý nghĩa: Vào lệnh sẽ kiểm tra **điều kiện**, nếu **điều kiện** đúng thì thực thi **công việc**, sau đó quay lại kiểm tra **điều kiện**. Cứ tiếp tục như thế cho tới khi nào **điều kiện** sai thì kết thúc.

Ví dụ: Tính tiền gởi ngân hàng. Lãi suất hàng tháng là 1.7%, người đó gởi vào ngân hàng vốn ban đầu là 1000000 (1 triệu), cứ sau mỗi tháng tiền lãi được gộp vào vốn và trở thành vốn mới để tính cho tháng sau. Hỏi sau bao lâu người đó được 1 tỷ đồng?

```
var Ls, Vn, Mm, tam: real;

sothang, i: integer;

Begin

Writeln('CHUONG TINH TINH TIEN GOI NGAN HANG');

Ls := 1.7/100; {Lãi suất 1.7%}

Vn := 1000000; {Số vốn ban đầu - 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('So thang = ' ,sothang);

Writeln('Tien von cong lai la: ',tam:12:2);

readln;

End.
```

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.

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

```
Uses Crt;

Var C, S, R: Real;

Traloi: Char;

Begin
```

```
Clrscr;

Repeat

Write('Nhập 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 REPEATE … UNTIL và FOR ..TO .. DO

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

Cả hai vòng lặp While và Repeat đều 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 một lần, trong khi thân vòng lặp WHILE có thể không được thực hiện lần nào. Tuỳ những hoàn cảnh khác nhau mà ta lựa chọn loại vòng lặp cho thích hợp. Nếu dùng 2 lệnh này để giải cùng một bài toán, cùng một giải thuật như nhau thì điều kiện sau WHILE và điều kiện sau UNTIL là phủ định nhau.

CHƯƠNG TRÌNH CON

Chương IV: CHƯƠNG TRÌNH CON

1. KHÁI NIỆM VỀ CHƯƠNG TRÌNH CON

Trong chương trình, có những đoạn cần phải lập đi, lập 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 chương 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

Đối với Borland Pascal 7.0 điều này không còn bắt buộc vì ta có thể gọi hàm như gọi một thủ tục. Không nhất thiết phải lấy giá trị trả về. Để thực hiện được điều này trong menu Options >Compiler cần khai báo cú pháp mở rộng (eXtended syntax), hoặc trong chương trình cần có dẫn hướng biên dịch {$ X+}. Nếu không, khi biên dịch (gõ F9) Pascal sẽ thông báo lỗi "Error 122: Invalid variable reference". Tuy vậy, dù không có dẫn hướng biên dịch {$ X+}, khi gõ CTRL+F9 chương trình vẫn chạy như thường! Ví dụ:{$X+}Program TestExtendSyntax;uses crt;var i,j:byte;{----------}Function **DoiViTri**(i,j: byte):byte;Var Tam:byte;BEGINTam:=i; i:=j; j:=tam;Gotoxy(i,j); write('*')END;{----------}BEGINi:=5; j:=20;Gotoxy(i,j); write('*');**Doivitri**(i,j);readln;END.
khi thoả mãn các yêu cầu sau.

- Ta muốn nhận một kết quả và chỉ một mà thôi.
- Ta cần dùng tên chương trình con (chứa kết quả đó) để viết trong các biểu thức.

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

Borland Pascal thiết kế và cài đặt sẵn trong các Unit đi gèm theo gói phần mềm nhiều thủ tục và hàm rất tiện dùng. 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.

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 qui tắc đặt danh biểu đã đề cập ở chương I.
- Một hàm có thể không có hoặc có một hoặc nhiều tham số. Trong trường hợp có nhiều tham số có cùng một 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 dấu **;** (chấm phẩy).
- KiểuKQ là một kiểu vô hướng, nó phản ảnh kiểu của giá trị mà hàm trả về lại sau khi chạy xong. Ví dụ, ta khai báo hàm như sau:

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

Đây là một hàm có tên là TEST, với 3 tham số, x và y thuộc kiểu Integer, z thuộc kiểu real, hàm trả về một kết quả 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; Phép gán để trả về giá trị cho tên hàm. 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.
```

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.

1. 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 như 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.

1. LỜI GỌI CHƯƠNG TRÌNH CON VÀ VẤN ĐỀ TRUYỀN THAM SỐ.

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:

PROGRAM ProgName;VAR a,b: type1; x:type2BEGIN…….……END.PROCEDURE Proc1(t,h:type1; Var k:type2);VAR x,yBegin…….……End;PROCEDURE Proc2Var qBEGIN…….……END;FUNCTION func1(r:type): type;Var xBegin…….……End;

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.

Qui định này về tầm vực là qui định của riêng từng ngôn ngữ. Mỗi khi học một ngôn ngữ mới sinh viên cần tham khảo qui định vê tầm vực của riêng ngôn ngữ 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.
- 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 được 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).
- 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.
  Có một điều khó chịu là Pascal cho phép "quá tải" các tham số trong các thủ tục của "bản thân" nó như trong các thủ tục Write, Writeln. Chúng ta gọi Writeln('Mot tham so') hay Writeln('Tham so thu nhat','Tham so thu hai') đều được trong khi điều đó lại không cho phép đối với các chương trình con được viết bởi người dùng!
- - 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

<u>* Trong khi thực hiện thủ tục</u>:

- 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

- *Mấy vấn 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.

- *Và 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 lưu ý để phòng ngừa hiệu ứng lề ngoài mong muốn.

1. TÍNH ĐỆ QUI CỦA CHƯƠNG TRÌNH CON

Như đã nói trên một chương trình con trong Pascal có thể gọi về chính nó. Một lời gọi như thế gọi là một lời gọi đệ qui (recursion). Gọi đệ qui là một kỹ thuật lập trình rất quan

trọng vì nó thường ngắn gọn và thường … phù hợp với suy nghĩ tự nhiên về nhiều cách giải bài toán. Thậm chí nhiều bài toán hầu như chỉ có thể dùng đệ qui. Tuy nhiên xét về tốc độ giải thuật cũng như tối ưu không gian bộ nhớ thì đệ qui thường không phải là một giải pháp tốt. Người ta thường cố gắng khắc phục đệ qui bằng cách dùng vòng lặp và sử dụng stack nhưng đó là công việc không mấy dễ dàng.

Ví dụ 1:

Định nghĩa giai thừa của một số nguyên không âm m như sau:

$$m! = \begin{cases} 1 & \text{if } (m=0) \text{ or } (m=1) \\ m*(m-1)! & \text{if } (m \geq 2) \end{cases}$$

Lập trình để tính giai thừa của một số nguyên không âm nhập từ bàn phím.

Cách 1: Dùng đệ qui.

```
Function GT(m: Integer): Longint;

Begin

If ( m = 0 ) or ( m = 1 ) then

GT := 1

Else

GT := m * GT( m-1 );

End;
```

Rõ ràng cách viết đệ qui là "phù hợp một cách tự nhiên" với định nghĩa của giai thừa.

Việc thực thi một lời gọi đệ qui diễn ra tương tự như sau:

Ví dụ ta truyền vào giá trị m = 4, tức gọi GT(4).

GT(4) m = 4 → Tính 4 * GT(4-1) → gọi GT(3)

GT(3) m = 3 → Tính 3 * GT(3-1) → gọi GT(2)

GT(2) m = 2 → Tính 2 * GT(2-1) → gọi GT(1)

GT(1) m = 1 → Gán GT(1):=1

Cuối cùng một quá trình "tính ngược" sẽ cho phép trả về giá trị của GT(4):

GT(4) ? 4 * (3 * (2 * GT(1))).

Cách 2: Dùng vòng lặp.

```
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;
```

Lưu ý: Một chương trình con đệ qui bao giờ cũng có ít nhất hai phần:

- Phần gọi đệ qui. Trong ví dụ trên là GT:=m*GT(m-1).
- Phần "neo". Trong ví dụ trên là IF (m=0) or (m=1) THEN GT:=1. Phần này rất quan trọng vì nó đảm bảo quá trình đệ qui phải dừng sau một số hữu hạn bước. Quên phần này sẽ xảy ra lỗi làm tràn bộ nhớ stack (stack overflow) khi xảy ra quá trình đệ qui.

Ví dụ 2:

Số Fibonacci được định nghĩa như sau:

Fibo(*n*)=

Chúng ta thấy bản thân định nghĩa số Fibonacci đã chứa một biểu thức truy hồi, tức về mặt lập trình đã dẫn tới một gợi ý lời gọi đệ qui. Chúng ta có thể xây dựng một hàm tính số Fibonacci như sau:

Cách 1: (Dùng đệ qui)

```
FUNCTION FIBO(n: word): word;

BEGIN

IF (n=1) or (n=2) THEN

FIBO:=1

ELSE

FIBO := FIBO(n-1)+FIBO(n-2);

END;
```

Trong cách này việc xây dựng hàm tính số Fibonacci tương đối dễ dàng vì cách viết hoàn toàn đồng nhất với định nghĩa toán học. Ví dụ thứ hai này phức tạp hơn ví dụ thứ nhất vì lời gọi đệ qui chia làm hai nhánh.

Cách 2: (Dùng chỗ lưu trữ tạm)

```
FUNCTION FIBO(n:word):word;

Var Counter,F1,F2:word;

BEGIN

F1:=1; F2:=1; Fibo:=1;

FOR Counter:=3 TO n DO

Begin

Fibo:=F1+F2;
```

```
F1:=F2;

F2:=Fibo;

End;

END;
```

Trong cách 2 này việc khử đệ qui không còn dễ dàng nữa vì cách đó không chứa rõ ràng một qui tắc tổng quát cho phép xử lí.

Ví dụ 3:

Bài toán tháp Hà Nội:

Có 3 cái cọc, đánh dấu A, B, C, và N cái đĩa. Mỗi đĩa đều có một lỗ chính giữa để đặt xuyên qua cọc, các đĩa đều có kích thước khác nhau. Ban đầu tất cả đĩa đều được đặt ở cọc thứ nhất theo thứ tự đĩa nhỏ hơn ở trên.

Yêu cầu: Chuyển tất cả các đĩa từ cọc A qua cọc C với ba ràng buộc như sau:

1. Mỗi lần chỉ chuyển được một đĩa.
2. Trong quá trình chuyển đĩa có thể dùng cọc còn lại để làm cọc trung gian.
3. Chỉ cho phép đặt đĩa có bán kính nhỏ hơn lên đĩa có bán kính lớn hơn.

Trong bài toán trên hình dung một lời giải tổng quát cho trường hợp tổng quát N đĩa là không dễ dàng. Hãy bắt đầu với các trường hợp đơn giản.

N = 1. Lời giải trở thành tầm thường (nhưng không kém phần quan trọng đâu!). Đơn giản là chuyển đĩa này từ cọc A qua cọc C là xong.

N = 2. Để đảm bảo ràng buộc thứ hai ta bắt buộc chuyển đĩa trên cùng từ cọc A qua cọc B. Chuyển tiếp đĩa còn lại từ cọc A qua cọc C. Chuyển tiếp đĩa đang ở cọc B sang cọc C.

N=3. Ta phải thực hiện 7 bước như sau:

Trạng thái ban dầu	----------		
Bước 1: Chuyển một đĩa từ A qua C.	----------		--
Bước 2: Chuyển một đĩa từ A qua B.	------	----	--
Bước 3: Chuyển một đĩa từ C qua B.	------	------	

Bước 4: Chuyển một đĩa từ A qua C.		------	------
Bước 5: Chuyển một đĩa từ B qua A.	--	----	------
Bước 6: Chuyển một đĩa từ B qua C.	--		----------
Bước 7: Chuyển một đĩa từ A qua C.			----------

Hãy quan sát kết quả ở bước thứ ba. Đây là một kết quả quan trọng vì nó cho ta thấy từ trường hợp N=3 bài toán đã được phân chia thành hai bài toán với kích thước nhỏ hơn: đó là bài toán chuyển 1 đĩa từ cọc A qua cọc C lấy cọc B làm trung gian và bài toán chuyển 2 đĩa (dời) từ cọc B sang cọc C lấy cọc A làm trung gian. Hai bài toán con này đã biết cách giải (trường hợp N=1 và trường hợp N=2).

Nhận xét đó cho ta gợi ý trong trường hợp tổng quát:

- Bước 1: Dời (N-1) đĩa trên cùng từ cọc A sang cọc B lấy cọc C làm trung gian.
- Bước 2: Chuyển 1 đĩa dưới cùng từ cọc A sang cọc C.
- Bước 3: Dời (N-1) đĩa đang ở cọc B sang cọc C lấy cọc A làm trung gian.

Bài toán đối với N đĩa như vậy được "đệ qui" về hai bài toán (N-1) đĩa và bài toán 1 đĩa. Quá trình đệ qui sẽ dừng lại khi N=0 (không còn đĩa để dời hoặc chuyển).

Chương trình sẽ như sau:

```
PROGRAM ThapHanoi;

Uses crt;

TYPE

Cot = Char;

{-----------------------------------------------------}

Procedure Chuyen(X,Y:Cot);

BEGIN

Writeln(X,' -> ',Y);

END;

{-----------------------------------------------------}
```

```
Procedure Doi(N:byte; A,B,C:Cot);
{Dời N đĩa từ cọc A sang cọc C lấy cọc B làm trung gian}
BEGIN
IF (N>0) THEN
Begin
Doi(N-1,A,C,B); {Dời N-1 đĩa từ cọc A sang cọc B lấy cọc C làm trung gian}
Chuyen(A,C);
Doi(N-1,B,A,C); {Dời N-1 đĩa từ cọc B sang cọc C lấy cọc A làm trung gian}
End;
END;
{-----------------------------------------------------}
BEGIN
Clrscr;
Write('Cho biet so dia :'); Readln(Sodia); Writeln('Cac buoc thuc hien:');
Doi(Sodia,'A','B','C');
Writeln; Writeln('Thuc hien xong!'); READLN;
END.
```

Nếu áp dụng chương trình này cho trường hợp N=3 ta có quá trình gọi đệ qui như sau:

Doi(1,A,B,C)
Doi(0,A,C,B)
Chuyen(A,C)

Doi(0,B,A,C)		
Doi(3,A,B,C)		
Doi(2,A,C,B)	Chuyen(A,B)	
	Doi(1,C,A,B)	
Doi(0,C,B,A)		
Chuyen(C,B)		
Doi(0,A,C,B)		
Chuyen(A,C)		
Doi(1,B,C,A)		
Doi(0,B,A,C)		
Chuyen(B,A)		
Doi(0,C,B,A)		
Doi(2,B,A,C)	Chuyen(B,C)	
		Doi(1,A,B,C)
Doi(0,A,C,B)		
Chuyen(A,C)		
Doi(0,B,A,C)		

Ví dụ này cho thấy việc kết xuất ở các câu lệnh Chuyen(X,Y) chỉ xảy ra khi toàn bộ các lời gọi đệ qui đã được thực hiện và cũng cho thấy thứ tự các lời gọi đệ qui lần cuối cùng. Nhận xét này rất quan trọng khi bạn viết thủ tục đệ qui vì lẽ bạn cần phải hình dung trước thứ tự các kết xuất nhất là khi lời gọi đệ qui có rất nhiều nhánh.

UNIT

Chương 5: UNIT

KHÁI NIỆM VỀ UNIT

Khái Niệm Về Unit

Việc tạo ra các chương trình con trong một chương trình đã làm cho việc lập trình đỡ vất vả hơn rất nhiều. Tuy nhiên, các chương trình con này chỉ có tác dụng trong chương trình chứa chúng mà thôi, trong một chương trình khác muốn sử dụng chương trình con này bắt buộc phải viết lại chúng, như vậy rất mất thời gian. Để khắc phục, người ta gom các chương trình con thường sử dụng thành một module độc lập và biên dịch sẵn trên đĩa. Sau đó, bất kỳ chương trình nào cũng có thể sử dụng lại các chương trình con này mà không cần phải viết lại chúng. Các module như vậy được gọi là UNIT. Khái niệm Unit được tạo ra từ version 4.0 của Turbo Pascal.

Có hai loại UNIT là Unit chuẩn của Pascal tạo ra và Unit do người lập trình tự tạo để phục vụ riêng cho mình.

Các Unit Chuẩn

Giới thiệu một số Unit chuẩn

- Unit CRT: Gồm các hằng, kiểu, biến, hàm, thủ tục liên quan đến chế độ màn hình văn bản (TEXT mode).
- Unit PRINTER: Gồm các hằng, kiểu, biến, hàm, thủ tục liên quan đến chế độ in ấn qua cổng LPT1 (Connector DB25).
- Unit GRAPH: Gồm các hằng, kiểu, biến, hàm, thủ tục liên quan đến chế độ đồ họa.
- Unit DOS: Gồm các hằng, kiểu, biến, hàm, thủ tục liên quan đến việc xử lí trực tiếp các thanh ghi, các ngắt và lời gọi đến các hàm chức năng của hệ điều hành MS-DOS.
- Unit OVERLAY: Gồm các hằng, kiểu, biến, hàm, thủ tục liên quan đến việc bố trí các đoạn mã thực thi được truy xuất trên đĩa (nạp/ nhã) thay vì đặt hết một lúc vào bộ nhớ khi chạy chương trình.
- Các Unit khác: SYSTEM, TURBO3, GRAPH 3… là các Unit phiên bản 3.0 sử dụng.

Cú pháp: **USES <Tên Unit> [, <Tên Unit>];**Khi muốn sử dụng một Unit nào thì ta phải khai báo tên Unit đó ở đầu chương trình (trừ các unit mặc định của Pascal như unit SYSTEM) với cú pháp như dưới đây.

Một số hàm và thủ tục hay dùng trong Unit CRT

- **ClrScr**: Thủ tục xoá màn hình.
- **GotoXY**(*x*, *y*: Byte): Dời con trỏ tới vị trí cột *x*, dòng *y* trên màn hình. Thông thường, màn hình trong TextMode(Co80) có 25 dòng (từ dòng 1 đến dòng 25) và 80 cột (cột 1 đến cột 80). Vậy toạ độ góc trên trái của màn hình là (1, 1), toạ độ góc dưới phải là (80, 25)
  Hiện nay đối với các màn hình TextMode giả lập của Windows khi chạy Borland Pascal có thể được thiết lập mặc định tới 80 cột và 50 dòng. Sinh viên phải thử cụ thể trên màn hình. Một số màn hình LCD wide screen cũng có thể cho số cột lớn hơn! Hầu như các projector hiện nay hỗ trợ kém chế độ văn bản. Cần cẩn thận khi lập trình để hiển thị (cuối cùng) trên projector.
  .
- **Delay**(*ms*: Word): Thủ tục trì hoãn chương trình trong *ms* mili-giây.
- **Sound**(*hz*: Word): Thủ tục phát ra âm thanh qua loa bên trong (internal speaker) với tần số *hz*.
- **Nosound**: Thủ tục ngừng phát ra âm thanh.
- **Keypressed**: Hàm cho kết quả là TRUE nếu có một phím được ấn.
- **Readkey**: Hàm cho kết quả là mã ASCII của ký tự khi phím được ấn.
- **TextBackGround**(color: Byte): Thủ tục chọn màu nền. Ta có thể đặt màu nền cho toàn màn hình bằng cách đặt lệnh này vừa trước lệnh ClrScr.
- **TextColor**(color: Byte): Thủ tục chọn màu cho chữ.

Dưới đây là danh sách các hằng màu mà Pascal định sẵn.

- Black = 0 Đen.
- Blue = 1 Xanh dương.
- Green = 2 Xanh lục.
- Cyan = 3 Xanh trứng sáo.
- Red = 4 Đỏ.
- Magenta = 5 Tím cánh sen.
- Brown = 6 Nâu.
- LightGray = 7 Xám sáng.
- DarkGray = 8 Xám tối.
- LightBlue = 9 Xanh dương sáng.
- LightGreen = 10 Xanh lục sáng.
- LightCyan = 11 Xanh trứng sáo sáng.
- LightRed = 12 Đỏ sáng.
- LightMagenta = 13 Tím cánh sen sáng.

- Yellow = 14 Vàng.
- White = 15 Trắng.

(8 hằng trị đầu tiên từ Black đến LightGray áp dụng cho cả màu chữ lẫn màu nền. Các hằng trị còn lại chỉ áp dụng cho màu chữ).

Unit CRT cũng thiết lập biến hệ thống TextAttr để xác định chế độ màu của màn hình văn bản hiện tại. Ví dụ để thiết lập màn hình có màu chữ xanh lục sáng trên nền xanh da trời ta thiết lập câu lệnh gán:

TextAttr:=LightGreen+16*Blue;

THIẾT LẬP UNIT

Các Bước Tạo Một Unit

Bước 1

Tạo ra một tập tin Pascal có đuôi **.**PAS và có cấu trúc như trình bày dưới đây, lưu ý là tên của unit phải trùng với tên tập tin.

UNIT <Tên Unit>; {Tên unit bắt buộc phải trùng với tên tập tin}

INTERFACE {Không có dấu ; ở đây}

{Đây là phần giao diện của Unit. Trong phần này chúng ta sẽ khai báo các unit đã có mà các unit này sử dụng, khai báo các hằng, kiểu, biến mà các chương trình khác sẽ sử dụng. Khai báo các hàm, thủ tục mà chương trình khác sẽ gọi tới, chỉ khai báo tên chương trình con, các tham số, kiểu kết quả. Những hàm, thủ tục thiết lập ở phần sau mà không khai báo trong phần này thì các chương trình khác không gọi tới được.}

IMPLEMENTATION {Không có dấu ; ở đây}

{Đây là phần hiện thực các hàm, thủ tục đã khai báo trong phần Interface. Trong phần này nếu có các chương trình con được dùng riêng bên trong Unit mà không khai báo trong phần Interface, các chương trình con này sẽ không thể truy cập được bởi người dùng Unit.}

BEGIN

{Phần chứa các câu lệnh sẽ được thực thi ngay trước khi câu lệnh đầu tiên của chương trình gọi Unit này được thực hiện. Phần này không bắt buộc phải có, tuy nhiên trong trường hợp đó vẫn phải giữ lại từ khóa “END.” dưới đây.}

END.

Bước 2

Unit không được thiết kế để chạy mà để biên dịch đặt lên đĩa nên ta không thể nhấn CTRL+F9 mà làm theo trình tự sau:

? Chọn menu **Compile** (Alt + C).

? Tiếp tục chọn **Destination** để chuyển thành **Disk**. *Lưu ý*: Destination Disk là tạo unit lên đĩa, Memory là tạo unit lên bộ nhớ RAM.

? Chọn lại menu **Complie** và chọn tiếp chức năng **Complie** (Alt + F9).

Lúc này trên đĩa xuất hiện tập tin là tên của unit ta tạo với phần mở rộng là TPU.

Kể từ đây, ta có thể sử dụng unit này bằng cách gọi nó trong câu lệnh USES như đã nói trên.

Ví dụ ứng dụng

Dưới đây là chương trình tạo ra một unit đơn giản có 3 hàm là *HamMu* để tính *a* mũ *n* (*an*), *GiaiThua* để tính *n* giai thừa (*n!*) và *USCLN* để tính ước số chung lớn nhất của hai số nguyên không âm..

Unit MyUnit; {Trùng tên với tập tin MyUnit.pas}

INTERFACE

Function HamMu(a: Real; n: Integer): Real;

Function GiaiThua(n: Integer): Longint;

Function USCLN(X,Y:Word):word;

IMPLEMENTATION

Function HamMu(a: Real; n: Integer): Real;

Var tam: Real;

i: Integer;

Begin

```
tam := 1;
For i:=1 to n do
tam := tam * a;
HamMu := tam;
End;
Function GiaiThua(n: Integer): Longint;
Var tam: Longint;
i: Integer;
Begin
tam := 1;
For i:=1 to n do
tam := tam * i;
GiaiThua := tam;
End;
Procedure HoanChuyen(var x,y:word);
VAR Tam:word;
BEGIN
Tam:=x; x:=y; y:=Tam;
END;
Function USCLN(x,y:Word):word;
BEGIN
While (y<>0) DO
```

```
Begin

IF (x<y) THEN HoanChuyen(x,y)

ELSE x:=x-y;

End;

USCLN:=x;

END;

END.
```

Sau khi biên dịch ta sẽ có tập tin unit là MyUnit.TPU. Khi sử dụng unit này người dùng có thể gọi các hàm đã khai báo trong phần INTERFACE nhưng không thể gọi tới Procedure HoanChuyen được.

TẬP TIN TURBO.TPL

File \BP\BIN\TURBO.TPL (Turbo Pascal Library) là tập tin thư viện gom các Unit thường dùng nhất vào một tập tin duy nhất và được nạp vào bộ nhớ ngay lúc khởi động Pascal để ta có thể dùng các Unit chứa sẵn trong tập tin thư viện này mà không cần đọc đĩa. Mặc định, sau khi cài đặt, TURBO.TPL chứa các Unit SYSTEM, DOS, OVERLAY, PRINTER, CRT. Riêng đối với Unit System.tpu ta không cần phải khai báo "USES SYSTEM;" để sử dụng các thủ tục writeln hay readln .v.v.

Pascal cũng cho phép ta gỡ bỏ khỏi TURBO.TPL các Unit không cần thiết hoặc thêm vào đó các Unit khác bằng cách chạy file \BP\BIN\TPUMOVER.EXE.

TPUMOVER.EXE chạy trong môi trường DOS. Cú pháp sử dụng như sau:

- *Hỏi cú pháp sử dụng*:

TPUMOVER.EXE ?

- *Xem một tập tin thư viện đang chứa các Unit nào*:

TPUMOVER.EXE <Tên tập tin thư viện> ?

- *Thêm/ bớt/trích một Unit khỏi tập tin thư viện*:

TPUMOVER.EXE <Tên tập tin thư viện> <Tác vụ> ?

Trong đó <tên tập tin thư viện> là tập tin có đuôi file mặc định là .TPL

Và tác vụ là một trong 3 trường hợp sau đây:

<+UnitName> : Để thêm Unit UnitName này vào tập tin thư viện.

<-UnitName> : Để loại Unit UnitName này khỏi tập tin thư viện.

<*UnitName> : Để trích Unit UnitName này khỏi tập tin thư viện.

Bạn cần cẩn thận khi loại một Unit ra khỏi tập tin thư viện. Để an toàn, tốt hơn hết nên trích xuất nó ra đã … trước khi làm thao tác loại bỏ.

Ví dụ:

Thêm MyUnit.TPU vào tập tin thư viện TURBO.TPL :

TPUMOVER.EXE TURBO.TPL +MyUnit.TPU ?

Xem coi tập tin thư viên TURBO.TPL đang chứa các Unit nào:

TPUMOVER.EXE TURBO.TPL ?

Gỡ Unit OVERLAY.TPU khỏi TURBO.TPL:

TPUMOVER.EXE TURBO.TPL -OVERLAY.TPU ?

Tham gia đóng góp

Tài liệu: Giáo trình Pascal 7.0

Biên tập bởi: Võ Thanh Ân

URL: http://voer.edu.vn/c/f83d08c4

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: GIỚI THIỆU NGÔN NGỮ PASCAL VÀ BORLAND PASCAL 7.0

Các tác giả: Võ Thanh Ân

URL: http://www.voer.edu.vn/m/89295181

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: CÁC KIỂU VÔ HƯỚNG CHUẨN VÀ CÁC CÂU LỆNH ĐƠN

Các tác giả: Võ Thanh Ân

URL: http://www.voer.edu.vn/m/4200cb86

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: LỆNH CÓ CẤU TRÚC

Các tác giả: Võ Thanh Ân

URL: http://www.voer.edu.vn/m/8cfd408d

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: CHƯƠNG TRÌNH CON

Các tác giả: Võ Thanh Ân

URL: http://www.voer.edu.vn/m/a1f9cbe7

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: UNIT

Các tác giả: Võ Thanh Ân

URL: http://www.voer.edu.vn/m/6f8e74b2

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Chương trình Thư viện Học liệu Mở Việt Nam

Chương trình Thư viện Học liệu Mở Việt Nam (Vietnam Open Educational Resources – VOER) được hỗ trợ bởi Quỹ Việt Nam. Mục tiêu của chương trình là xây dựng kho Tài nguyên giáo dục Mở miễn phí của người Việt và cho người Việt, có nội dung phong phú. Các nội dung đều tuân thủ Giấy phép Creative Commons Attribution (CC-by) 4.0 do đó các nội dung đều có thể được sử dụng, tái sử dụng và truy nhập miễn phí trước hết trong trong môi trường giảng dạy, học tập và nghiên cứu sau đó cho toàn xã hội.

Với sự hỗ trợ của Quỹ Việt Nam, Thư viện Học liệu Mở Việt Nam (VOER) đã trở thành một cổng thông tin chính cho các sinh viên và giảng viên trong và ngoài Việt Nam. Mỗi ngày có hàng chục nghìn lượt truy cập VOER (www.voer.edu.vn) để nghiên cứu, học tập và tải tài liệu giảng dạy về. Với hàng chục nghìn module kiến thức từ hàng nghìn tác giả khác nhau đóng góp, Thư Viện Học liệu Mở Việt Nam là một kho tàng tài liệu khổng lồ, nội dung phong phú phục vụ cho tất cả các nhu cầu học tập, nghiên cứu của độc giả.

Nguồn tài liệu mở phong phú có trên VOER có được là do sự chia sẻ tự nguyện của các tác giả trong và ngoài nước. Quá trình chia sẻ tài liệu trên VOER trở lên dễ dàng như đếm 1, 2, 3 nhờ vào sức mạnh của nền tảng Hanoi Spring.

Hanoi Spring là một nền tảng công nghệ tiên tiến được thiết kế cho phép công chúng dễ dàng chia sẻ tài liệu giảng dạy, học tập cũng như chủ động phát triển chương trình giảng dạy dựa trên khái niệm về học liệu mở (OCW) và tài nguyên giáo dục mở (OER) . Khái niệm chia sẻ tri thức có tính cách mạng đã được khởi xướng và phát triển tiên phong bởi Đại học MIT và Đại học Rice Hoa Kỳ trong vòng một thập kỷ qua. Kể từ đó, phong trào Tài nguyên Giáo dục Mở đã phát triển nhanh chóng, được UNESCO hỗ trợ và được chấp nhận như một chương trình chính thức ở nhiều nước trên thế giới.