UBND TỈNH LÂM ĐỒNG
TRƯỜNG CAO ĐẲNG ĐÀ LẠT

GIÁO TRÌNH

MÔN HỌC: LẬP TRÌNH CƠ BẢN

NGÀNH/NGHỀ: CÔNG NGHỆ THÔNG TIN (ỨNG DỤNG PHẦN MỀM)

TRÌNH ĐỘ: CAO ĐẲNG

Ban hành kèm theo Quyết định số: /QĐ-... ngày……tháng….năm ..... …… ..... của……

LƯU HÀNH NỘI BỘ

Đà Lạt, năm 2017

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Đây là tài liệu được xây dựng theo chương trình đào tạo nghề Công nghệ Thông tin (ứng dụng phần mềm).

Trong giai đoạn viết giáo trình chúng tôi cũng đã có những sự điều chỉnh để giáo trình có tính thiết thực và phù hợp hơn với sự phát triển của lĩnh vực công nghệ thông tin.

Lập trình cơ bản là một môn học đào tạo cơ sở nghề Công nghệ Thông tin (ứng dụng phần mềm) trình độ trung cấp, cao đẳng. Ngày nay, chúng ta có thể sản xuất ra các hệ thống phần mềm lớn và hữu ích là nhờ phần lớn vào sự phát triển của các ngôn ngữ lập trình.

Trong phạm vi giáo trình Lập trình cơ bản này, chúng tôi giới thiệu sơ lược về Lập trình cơ bản C++ cho các học sinh sinh viên đã có kiến thức căn bản về công nghệ thông tin. Giới thiệu những vấn đề cơ bản nhất về ngôn ngữ C++. Giới thiệu cho người đọc hiểu được công dụng của Lập trình cơ bản, hiểu cú pháp, công dụng của các câu lệnh dùng trong Lập trình cơ bản. Phân tích được chương trình: xác định nhiệm vụ chương trình (phải làm gì).Vận dụng điều kiện, trợ giúp môi trường của Lập trình cơ bản, chẳng hạn: các thao tác biên tập chương trình, các công cụ, điều khiển, thực đơn lệnh trợ giúp, gỡ rối, bẫy lỗi,v.v.Viết chương trình và thực hiện chương trình trong máy tính.

Trong quá trình biên soạn, mặc dù đã cố gắng tham khảo nhiều tài liệu và giáo trình khác nhưng tác giả không tránh khỏi được những thiếu sót và hạn chế. Tác giả chân thành mong đợi những nhận xét, đánh giá và góp ý để cuốn giáo trình ngày một hoàn thiện hơn. Mọi thông tin đóng góp xin gửi thư điện tử về địa chỉ ngothienhoang@cdndalat.edu.vn

Tài liệu này được thiết kế theo từng mô đun/ môn học thuộc hệ thống mô đun/môn học của một chương trình, để đào tạo hoàn chỉnh nghề Công nghệ Thông tin (ứng dụng phần mềm ở cấp trình độ trung cấp, cao đẳng và được dùng làm Giáo trình cho học sinh sinh viên trong các khóa đào tạo, cũng có thể được sử dụng cho

đào tạo ngắn hạn hoặc cho đào tạo công nhân kỹ thuật, các nhà quản lý và lập trình viên tham khảo.

Đà Lạt, ngày 07 tháng 7 năm 2017

Tham gia biên soạn

1. Chủ biên Ngô Thiên Hoàng
2. Phạm Đình Nam
3. Trương Thị Thanh Thảo
4. Nguyễn Quỳnh Nguyên
5. Phan Ngọc Bảo

.

MỤC LỤC

Trang

GIÁO TRÌNH MÔ ĐUN/MÔN HỌC

Tên môn học: LẬP TRÌNH CƠ BẢN

Mã môn học: MH09

I. Vị trí, tính chất của môn học:

1. Vị trí: Môn học này được bố trí sau môn học: Tin học căn bản.

2. Tính chất: Lập trình cơ bản là môn học cơ sở áp dụng cho trình độ Cao đẳng Công nghệ thông tin (ứng dụng phần mềm).

II. Mục tiêu môn học:

1. Về kiến thức:

- Cung cấp cho học sinh, sinh viên những khái niệm cơ bản về Lập trình cơ bản C/ C++: câu lệnh, từ khóa, cú pháp, cách khai báo và sử dụng biến, hàm và các cấu trúc lệnh trong ngôn ngữ C/ C++;

- Trình bày được ý nghĩa, cách khai báo, cách truy xuất với một số cấu trúc dữ liệu;

- Trình bày được một số thuật toán để xử lý một số yêu cầu đơn giản;

2. Về kỹ năng:

- Cài đặt được một số chương trình ứng dụng đơn giản bằng Lập trình cơ bản C/ C++;

- Viết, chỉnh sửa, thực thi được các chương trình C/ C++ có sẵn;

3. Về năng lực tự chủ và trách nhiệm:

- Có khả năng tự nghiên cứu, tự học, tham khảo tài liệu liên quan đến môn học để vận dụng vào hoạt động hoc tập.

- Vận dụng được các kiến thức tự nghiên cứu, học tập và kiến thức, kỹ năng đã được học để hoàn thiện các kỹ năng liên quan đến môn học một cách khoa học, đúng quy định.

III. Nội dung môn học:

CHƯƠNG 1 TỔNG QUAN VỀ NGÔN NGỮ C/ C++

Mã bài: MH 09.1

Mục tiêu của bài:

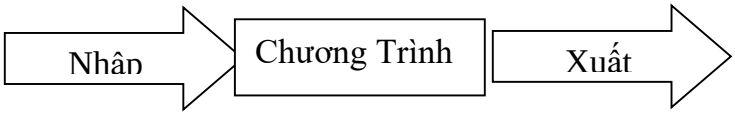
- Trình bày được lịch sử phát triển của ngôn ngữ C/ C++;
- Trình bày được những ứng dụng thực tế của Lập trình cơ bản C/ C++;
- Thao tác thành thạo trên môi trường phát triển C/ C++;
- Sử dụng được hệ thống trợ giúp của phần mềm C/ C++;
- Chủ động tìm hiểu các tính năng của ngôn ngữ C/ C++.

1.1 Giới thiệu ngôn ngữ C++

Lập trình thực chất là điều khiển - bằng một Lập trình cơ bản cụ thể - các xử lý thông tin trên máy tính điện tử theo yêu cầu của bài toán đặt ra.

Kết quả của lập trình là chương trình được hợp thức hóa.

Thông tin gửi đến CT, CT xử lý, kết quả sẽ được gửi đi.



Để lập trình phải biết cách tổ chức dữ liệu (cấu trúc dữ liệu) và cách thức xử lý dữ liệu(thuật toán) để tạo ra chương trình mong muốn.

K. Wirth đã đưa ra công thức :

CHƯƠNG TRÌNH = CẤU TRÚC DỮ LIỆU + THUẬT TOÁN

Có nhiều cách tổ chức dữ liệu cũng như có nhiều thuật toán để giải một bài toán . Đưa ra cách tổ chức dữ liệu tốt nhất và chỉ ra thuật toán tốt nhất là công việc của người lập trình.

Các phương pháp lập trình thường được sử dụng là lập trình có cấu trúc và lập trình theo hướng đối tượng.

Các phương pháp lập trình này phản ánh quan niệm lập trình là một hoạt động khoa học và có phương pháp chứ không phải là một công việc ngẫu hứng.

Đặc trưng của lập trình có cấu trúc là chương trình phải có cấu trúc.

Tính cấu trúc của chương trình thể hiện trên các mặt sau :

- Cấu trúc về mặt dữ liệu :

Từ những dữ liệu đã có , có thể xây dựng những dữ liệu có cấu trúc phức tạp hơn.

- Cấu trúc về mặt lệnh :

Từ những lệnh đơn giản đã có, có thể xây dựng được những lệnh có cấu trúc phức tạp hơn.

- Cấu trúc về mặt chương trình :

Một chương trình lớn có thể phân rã thành nhiều modul (hay các chương trình con) độc lập, mỗi chương trình con lại có thể chia ra thành các chương trình con khác... nên chương trình được tổ chức thành một hệ phân cấp. Nhờ vậy mà một chương trình lớn, phức tạp được phân thành những modul chương trình đơn giản, dễ viết, dễ đọc, dễ sửa...

1.2 Thuật toán

1.2.1 Định nghĩa trực quan thuật toán

Thuật toán là dãy hữu hạn các thao tác, sắp xếp theo trình tự xác định, được đề ra nhằm giải quyết một lớp bài toán nhất định.

Các thao tác sẽ biến đổi trạng thái bài toán trước khi thực hiện tháo tác thành trạng thái kết quả.

Dãy tuần tự các thao tác trong thuật toán biến đổi trạng thái ban đầu của thuật toán thành trạng thái cuối cùng của bài toán.

1.2.2 Các đặc trưng của thuật toán

- Tính xác định.
- Tính dừng (hữu hạn).
- Tính đúng đắn.

1.2.3 Đặc tả thuật toán

Đặc tả thuật toán (hay đặc tả bài toán) là định rõ lớp bài toán mà một thuật toán giải quyết, do đó nó cần chỉ ra các đặc điểm sau :

1. Các đối tượng và phương tiện của Thuật toán cần sử dụng (nhập).
2. Điều kiện ràng buộc (nếu có) trên các đối tượng và phương tiện đó.
3. Các sản phẩm, kết quả (xuất).

4. Các yêu cầu trên sản phẩm kết quả. Thường xuất hiện dưới dạng quan hệ giữa sản phẩm kết quả và các đối tượng,phương tiện sử dụng.

Ta viết :

INPUT : (1) và (2);

OUTPUT : (3) và (4);

1.2.4 Độ phức tạp của Thuật toán

Mỗi Thuật toán đều cần thời gian và các nguồn lực khác để giải quyết một bài toán cụ thể. Các tiêu hao đó đặc trưng độ phức tạp của thuật toán.

Có nhiều thuật toán cùng giải một lớp bài toán, ta mong muốn có được thuật toán hiệu quả hơn, tức tiêu hao ít hơn một trong các loại nguồn lực.

Đối với máy tính, ta quan tâm đến thời gian và kích thước bộ nhớ mà thuật toán sử dụng.

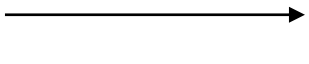
1.3 Diễn đạt thuật toán

1.3.1. Ngôn ngữ tự nhiên

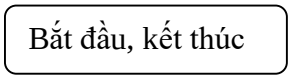
Mô tả các bước thực hiện của thuật toán dưới dạng văn bản bằng ngôn ngữ tự nhiên như tiếng Việt, Anh,. . .

1.3.2. Lưu đồ

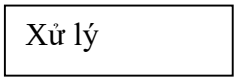
Sơ đồ toàn cảnh có cấu trúc biểu diễn các bước thực hiện của thuật toán, trong đó sử dụng các hình vẽ có quy ước sau:



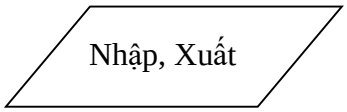
Mũi tên chỉ luồng lưu chuyển của dữ liệu.



Điểm bắt đầu, kết thúc

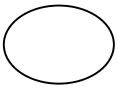


Hình chữ nhật biểu diễn các bước xử lý



Hình bình hành biểu thị nhập xuất

Hình thoi biểu thị thời điểm ra quyết định



Hình tròn biểu thị điểm ghép nối

1.3.3. Mã giả

Dựa vào cú pháp và ngữ nghĩa của một Lập trình cơ bản nào đó (chẳng hạn C, Pascal, ...). Cho nên mã giả cũng dựa trên các cơ sở sau đây:

- Ký tự
- Các từ
- Các xâu ký tự (Chuỗi)
- Hằng
- Biến
- Kiểu dữ liệu
- Lệnh gán
- Khối lệnh
- Các cấu trúc điều khiên
- Câu lệnh trả về
- . . .

1.3.4. Lập trình cơ bản

1.4 Chương trình

1.4.1. Khái quát về chương trình.

Thuật toán phải được diễn đạt sao cho máy tính có thể hiểu và thi hành được. Lập trình cơ bản (programming language) được sử dụng vào mục đích này.

Lập trình cơ bản là tập hợp các qui tắc chặt chẻ về cú pháp (syntax), về ngữ nghĩa (semantic) cho phép tạo ra các văn bản để diễn đạt thuật toán **.**

Một văn bản như vậy gọi là chương trình (program).

Chương trình viết bằng Lập trình cơ bản cấp cao gọi là chương trình nguồn (Source program)

Chương trình viết bằng ngôn ngữ máy gọi là chương trình đích (Target program).

Máy chỉ có thể thi hành chương trình dưới dạng ngôn ngữ máy, vì vậy chương trình nguồn muốn được khai thác phải được chuyển đổi thành chương trình đích tương đương. Chương trình có nhiệm vụ chuyển đổi chương trình nguồn thành chương trình đích tương đương được gọi là chương trình dịch.

Có 2 loại dịch khác nhau : biên dịch (Complier) và thông dịch (Interpreter).

1.4.2. Mã và dữ liệu

Một chương trình bất kỳ bao gồm hai phần : mã và dữ liệu.

Khi nạp một chương trình đã được dịch sang mã máy vào RAM, phần RAM chứa đoạn mã máy là các lệnh thể hiện thao tác được gọi là code segment. Trong code segment chứa phần mã của chương trình.

Phần RAM chứa các dữ liệu là đối tượng của các thao tác được gọi là data segment. Trong data segment chứa phần dữ liệu của chương trình.

Trong các lần chạy khác nhau, chỉ có phần dữ liệu của chương trình là thay đổi, còn phần mã là không đổi.

Data Segment →	**Biến toàn cục, dữ liệu chương trình**
Code Segment →	**Mã chương trình**

1.5 Giới thiệu C/C++

Ý tưởng quan trọng nhất của C xuất phát từ ngôn ngữ BCPL do Martin Richards thiết kế. Anh hưởng của BCPL lên C gián tiếp thông qua ngôn ngữ B do Ken Thompson viết năm 1970 cho hệ thống UNIX đầu tiên trên máy PDP-7.

Từ ngôn ngữ B, Dennish Ritchie và Brian Kernighan phát triển thành ngôn ngữ C vào những năm 1970 tại phòng thí nghiệm của hãng AT & T để phát triển cốt lõi của hệ điều hành UNIX.

Trong nhiều năm, chuẩn cho C trên thực tế là một phiên bản được cung cấp cùng với hệ điều hành Unix version 5. Nó được mô tả lần đầu tiên trong cuốn :

” The C programming language “

của Dennish Ritchie và Brian Kernighan.

Năm 1983 một hội đồng được thành lập để tạo ra một chuẩn cho C, gọi là chuẩn ANSI (American National Standards Institute : Viện định chuẩn Quốc gia Mỹ). Sau 6 năm, chuẩn cuối cùng ANSI C được đề nghị vào tháng 12/1989, và bản đầu tiên được dùng vào năm 1990. Ngày nay các trình biên dịch C chính đều giữ đúng theo ANSI chuẩn.

C là một ngôn ngữ có khả năng tổ hợp những thành phần tốt nhất của ngôn ngữ bậc cao và sự điều khiển linh hoạt của ngôn ngữ assembly .

Năm 1987 hãng Borland đã đưa ngôn ngữ C vào thị trường của IBM-PC thông qua Turbo C.

Vào những năm 1980, Bjarne Stroustrup đã cho ra đời ngôn ngữ C++ bằng cách cài vào ngôn ngữ C khái niệm lập trình hướng đối tượng.

Năm 1988, hãng Zortech giới thiệu một trình biên dịch C++ cho các máy tính MS-DOS.

Cho tới nay đã xuất hiện nhiều phiên bản C++ trong môi trường Microsoft Windows: Borland C++, Turbo C++ của hãng Borland; Visual C++ của Microsoft.

Hiện nay có nhiều chương trình lớn được viết hay được viết lại bằng C/C++:

- Hệ điều hành UNIX.
- Hệ điều hành Windows.
- Hệ quản trị cơ sở dữ liệu Dbase.
- Các chương trình soạn thảo văn bản.
- Các bảng tính điện tử.
- . . .

1.6 Khởi động và thoát khỏi C++

Sử dụng Turbo C++ 3.0 để thực hiện việc cài đặt.

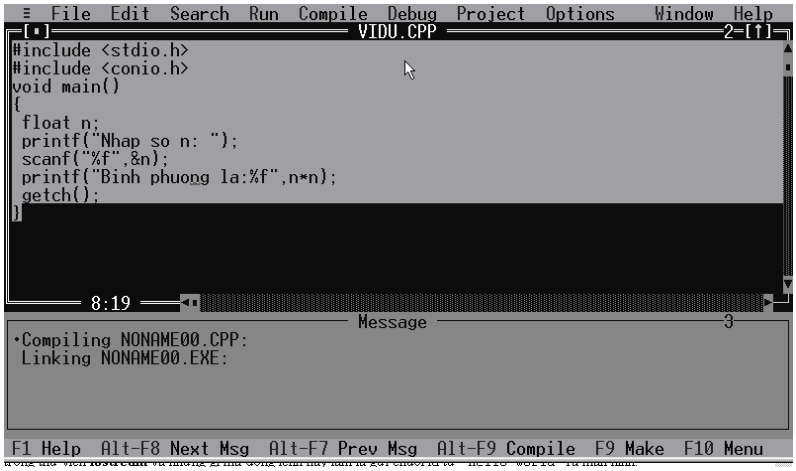
Ta thực hiện việc chuyển thư mục đến thư mục chứa tập tin cài đặt. Chẳng hạn E:\SOFT\TC30. Và gõ lệnh Install

Ví dụ: CD E:\SOFT\TC30

Install

Sau khi cài đặt TC xong sẽ nằm trong thư mục C:\TC

Cửa sổ soạn thảo của TC như hình sau



Hình 1: Cửa sổ soạn thảo của TC

*Menu

- Thanh đầu tiên là thanh Menu. Để mở thanh menu ta nhấn phím F10 sau đó dùng phím mũi tên qua lại để đưa đến mục chọn thích hợp nhấn phím Enter;

- Dùng phím mũi tên lên xuống để chọn mục chọn thích hợp, xong nhấn Enter.

* Cửa sổ soạn thảo

Cửa sổ soạn thảo là chứa tiêu đề là tên tập tin như ở hình vẽ là VIDU.CPP. Chúng ta sẽ soạn thảo chương trình vào trong cửa sổ này và nhớ rằng mỗi một chương trình hay mođun chứa trong một tập tin, nghĩa là tập tin không được chứa 2 chương trình mà chỉ được chứa một chương trình.

Để phóng to cửa sổ , ta nhấn phím F5.

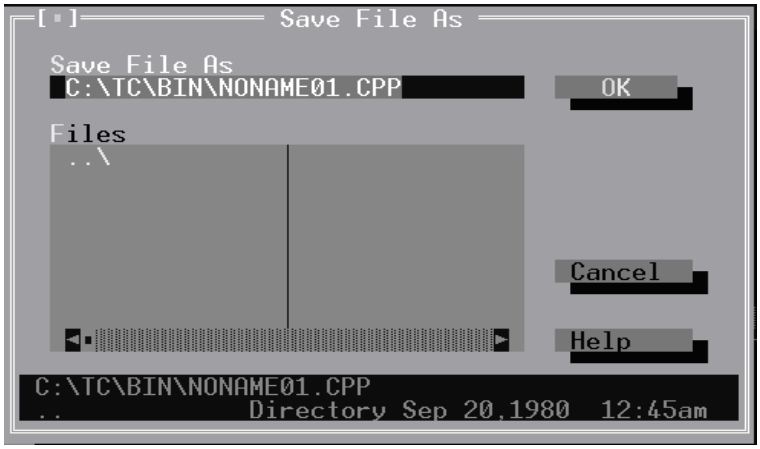
Để chuyển qua lại các cửa sổ, ta gõ F6.

Để đóng cửa sổ, ta gõ Alt+F3.

* Làm việc với tập tin

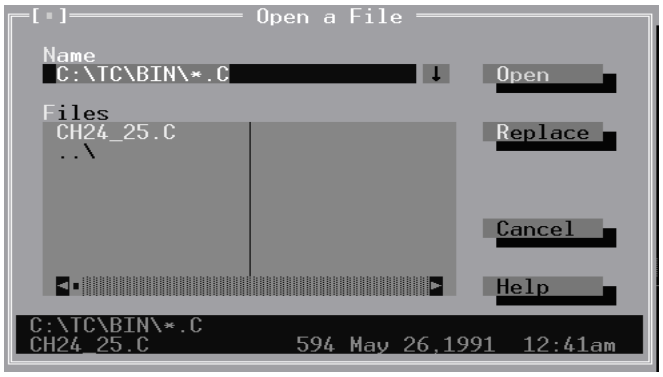
- Lưu tập tin đang soạn thảo:

Khi lưu trữ tập tin lần đầu thì hộp thoại như trên sẽ hiện ra. Gõ vào tên tập tin cần lưu trữ. Mặc định TC C++ 3.0 sẽ tự động thêm vào phần mở rộng .CPP. Sau khi lưu trữ tên tập tin sẽ được đưa và tiêu đề cửa sổ soạn thảo của chương trình. Hãy chú ý đường dẫn cuối hộp thoại, chúng ta có thể chọn lại đường dẫn bằng cách nhấn phím Tab cho thanh sáng ở trong danh sách, dùng mũi tên lên xuống để di chuyển đến thư mục cần chọn. Thư mục có tên ..\ là thư mục cha, ta có thể di chuyển ra ngoài thư mục cha bằng cách chọn mục này và nhấn phím Enter.



Hình 2: Hộp thoại khi lưu trữ tập tin lần đầu

Mở tập tin đã lưu trữ: Nhấn phím F3



Hình 3: Mở tập tin đã lưu trữ

Nhấn phím Tab để điểm sáng nằm bên danh sách và dùng phím mũi tên để chọn tập tin chương trình cần mở. Nhấn Enter.

Trong quá trình mở tập tin, chúng ta chú ý đường dẫn ở dưới cùng của hộp thoại.

- Có thể tạo thư mục cần lưu trữ tập tin vào thư mục do chúng ta tạo ra. Bằng những thao tác sau:

Trong Turbo C++, chọn Menu File/DOS Shell, màn hình DOS sẽ hiện ra, ta tiến hành tạo thư mục. Tạo thư mục xong gõ lệnh Exit để vào lại Turbo C++.

Vào Menu File/Change Dir để thay đổi mặc định lưu trữ tập tin và thư mục. Và hộp thoại như sau hiện ra.

Nhấn phím Tab để di chuyển điểm sáng sang cây thư mục. Chọn thư mục cần chuyển, nhấn Enter. Nhấn phím Tab để di chuyển điểm sáng đến nút OK và nhấn phím Enter.

Công việc này nên được làm lần đầu tiên vào trong Turbo C++ để việc lưu tập tin và mở tập tin cho chính xác.

* Các thao tác soạn thảo

Copy nội dung:

Di chuyển nội dung:

* Các tài liệu sẵn có:

- Để tra cứu hướng dẫn, nhấn F1.

- Để tra cứu hướng dẫn dành cho từ con trỏ đang đứng, ta gõ Ctrl+F1.

* Các thao tác biên dịch và chạy chương trình

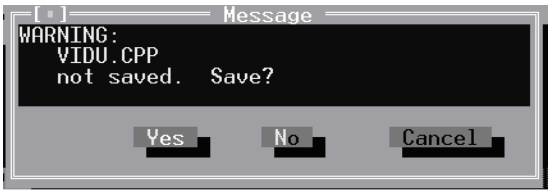
- Biên dịch chương trình ta nhấn: F9.

Để chạy chương trình ta nhấn: Ctrl + F9.

- Để xem kết quả chương trình ta nhấn Alt + F5.

- Nhấn Alt+X hoặc File/Exit để thoát khỏi chương trình.

Những tập tin nào chưa được lưu trong quá trình soạn thảo,chương trình sẽ hỏi có lưu tập tin đó không? Nếu muốn lưu trước khi thoát thì chọn Yes, Không thì chọn No. Muốn huỷ việc thoát khỏi TC thì chọn Cancel. Thao tác chọn được thực hiện bằng phím Tab.



Hình 4: Hộp thoại hỏi có lưu tập tin hay không?

Bài tập

1. Sử dụng sơ đồ khối để thiết kế thuật toán cho bài toán giải phương trình bậc 2: $ax^2 + bx + c = 0$.
2. Sử dụng Turbo C++ 3.0 để thực hiện việc cài đặt và cấu hình

CHƯƠNG 2 CÁC THÀNH PHẦN CƠ BẢN

Mã bài : MH 09.2

Mục tiêu của bài:

- Trình bày được các khái niệm: tập kí tự, từ khóa, kiểu dữ liệu, biến, hằng, biểu thức, câu lệnh, khối lệnh trong ngôn ngữ C/ C++;
- Trình bày được cấu trúc, phương thức thực thi của một chương trình C/ C++;
- Viết được một số đoạn chương trình cụ thể để thực hiện các yêu cầu đơn giản;
- Nghiêm túc, tỉ mỉ trong việc tiếp nhận các kiến thức và áp dụng vào thực hành.

2.1 Hệ thống ký hiệu và từ khóa

2.1.1 Ký hiệu cơ sở

Ngôn ngữ C++ được xây dựng từ bộ ký hiệu cơ sở sau:

- Bộ 26 chữ cái La-Tinh viết thường (nhỏ): a,b,...,z.
- Bộ 26 chữ cái La-Tinh viết hoa (lớn): A,B,...,Z.
- Bộ 10 chữ số hệ thập phân : 0,1,...,9.
- Bộ dấu các toán tử số học : + - * /
- Bộ dấu các toán tử so sánh: < > =
- Ký tự gạch nối: _ (Khác dấu trừ -).
- Các ký hiệu khác: ' " ; ,.: [ ] # $ & { } % ! . . .

Đặc biệt có khoảng trắng dùng để ngăn cách các từ (phím Space). Các ký hiệu cơ sở đều có trên bàn phím.

2.1.2 Các từ

Từ trong C++ được xây dựng bởi các ký hiệu cơ sở trên. Có 2 loại từ:Từ khóa và tên.

a. Từ khóa (Key Word)

Là những từ có ý nghĩa hoàn toàn xác định, chúng thường được dùng để khai báo các kiểu dữ liệu, để viết các toán tử, và các câu lệnh. Sau đây là các từ khóa trong Borland C++ (Turbo C++):

asm	auto	break	case	catch	char
class	const	continue	default	delete	do
double	else	enum	extern	float	for
friend	goto	if	Inline	int	long
new	operator	private	Protected	public	register

return	short	signed	Sizeof	static	struct
switch	template	this	throw	try	typedef
union	unsigned	virtual	void	volatile	while
cdecl	_cs	_ds	_es	_export	far
huge	interrupt	_loadds	Near	pascal	_regparam
_saveregs	_seg	_ss			

b. Tên hoặc danh hiệu (identifier):

Là từ do người sử dụng tự đặt để giải quyết bài toán của mình. Từ tự đặt dùng để đặt tên cho hằng, biến, hàm, tên kiểu dữ liệu mới,...

Tên được đặt theo quy tắc: phải bắt đầu bằng một chữ cái hoặc dấu gạch nối,sau đó là các chữ cái, chữ số hoặc dấu gạch nối, và không được trùng với từ khóa.

Tên có thể viết bằng chữ thường hoặc chữ hoa. Trong C++ có phân biệt chữ thường và chữ hoa.

2.2 Các Kiểu Dữ Liệu Cơ Bản Trong C++

Trong C++ có 5 kiểu dữ liệu cơ bản là:

- Các kiểu Ký tự.
- Các kiểu nguyên.
- Kiểu Số thực dấu chấm động độ chính xác đơn
- Kiểu Số thực dấu chấm động độ chính xác kép
- Kiểu void.

Các kiểu dữ liệu khác đều dựa vào các kiểu dữ liệu trên.

2.2.1 Ký tự

a. Ký tự 8 bit

Một giá trị ký tự có kiểu dữ liệu khai báo bằng từ khóa char, được lưu trử trong 8 bit và biểu diễn thông qua bảng mã ASCII .

Chẳng hạn :

Ký tự	Mã ASCII (hệ 10)
0	48
1	49
A	65

a 97

. . . .

Có các kiểu ký tự 8 bit tương ứng với các từ khóa :

signed char (như char, có dấu)

unsigned char (Không dấu).

Sau đây là bảng kích thước, phạm vi biểu diễn của các kiểu ký tự :

KIỂU	Phạm vi biểu diễn	Kích thước	Số ký tự
char	-128 → 127	1 byte	256
signed char	-128 → 127	1 byte	256
unsigned char	0 → 255	1 byte	256

b. Ký tự UNICODE

Các ký tự Unicode trong C/C++ được định nghĩa bởi : wchar_t

Mỗi ký tự Unicode rộng 16 bit.

c. Kiểu TCHAR

Dùng chung cho char và wchar_t (cho ký tự 8 bit hay unicode 16 bit).

2.2.2 Kiểu nguyên

Trong C++ cho phép sử dụng các kiểu số nguyên được khai báo bởi từ khóa int, hoặc đi kèm theo int với các từ khóa long, short, unsigned.

Số lượng bit được dùng để lưu trử một giá trị int phụ thuộc vào kích thước từ (word) của máy. Thường thì máy 16-bit sẽ dùng 16 bit để lưu trử một giá trị int , trong khi đó máy 32-bit sẽ dùng 32 bit .

Kích thước và phạm vi biểu diễn của chúng được cho trong bảng sau :

Kiểu	Phạm vi biểu diễn	Kích thước
int	Chiếm 1 từ của máy	
short int , short	-32768 → 32767 ($-2^{15} \rightarrow 2^{15} - 1$)	16 bit

unsigned short int	0 → 65535 (0 → 2^{16} -1)	16 bit
long int , long	-2147483648 → 2147483647 (-2^{31} → 2^{31} -1)	32 bit
unsigned long int	0 → 4294967295 (0 → 2^{32} -1)	32 bit
unsigned int	Số nguyên không âm , chiếm 1 từ của máy.	

2.2.3 Kiểu số thực

C++ cho phép sử dụng 3 kích thước giá trị thực, tương ứng với 3 từ khóa :

- float
- double
- long double

Kích thước và phạm vi biểu diễn của chúng được cho trong bảng sau :

Kiểu	Ý nghĩa	Phạm vi biểu diễn	Kích thước	Độ chính xác
float	Số thực chính xác đơn	-3.4E+38 → 3.4E+38	32 bit	6 số thập phân
double	Số thực chính xác kép	-1.7E+308 → 1.7E+308,0	64 bit	10 số thập phân
long double		Kích thước 96 bit hoặc 128 bit		

2.3 Biến, hằng, biểu thức

2.3.1. Định nghĩa

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán. Ta thường dùng các ký tự hoa để biểu diễn các hằng ký hiệu.

2.3.2 Các loại hằng

Trong C++ có các loại hằng sau đây:

1. Hằng số thực

Giá trị được lấy là float và double. Viết theo 2 cách :

a. Dạng thập phân (dấu chấm cố định) :

Bao gồm : Phần nguyên, dấu chấm thập phân, phẩn phân.

Phần nguyên	•	Phần phân

b. Dạng khoa học hay dạng mũ (dấu chấm động)

Có 3 thành phần : Phần định trị , ký hiệu E hoặc e , và phần bậc.

Phần định trị là một số nguyên hoặc số thực dạng thập phân. Phần bậc là một số nguyên. Hai phần này cách nhau bởi ký tự E hoặc e.

Phần định trị	E hoặc e	Phần bậc

Ví dụ :

12.234E-3 // biểu diễn giá trị 0.012234

0.35E4 // biểu diễn giá trị 3500.0

-12.22e-3 // biểu diễn giá trị -0.01222

1e6 // biểu diễn giá trị 1 000 000

2. Hằng nguyên

a. Hằng int

Là số nguyên có kiểu int .

b. Hằng long

Biểu diễn : thêm L hoặc l vào sau hằng int.

c. Hằng unsigned

- Biểu diễn : thêm u vào sau hằng int.
- Có giá trị từ 0 đến 65535.

d. Hằng int hệ 8

- Hằng int hệ 8 luôn nhận giá trị dương.
- Dạng biểu diễn : $0c_1c_2c_3$ …

  Với : c_i là một số nguyên trong khoảng từ 0 đến 7.

Ví dụ :

- Hằng int hệ 8 : 0345
- Giá trị của nó trong hệ 10 : 3*8*8 + 4*8 + 5 = 229.

e. Hằng int hệ 16 :

Trong hệ này sử dụng 16 ký tự :

0,1,2,3,4,5,6,7,8,9,

a hoặc A

b hoặc B

c hoặc C

d hoặc D

e hoặc E

f hoặc F

Dạng biểu diễn : 0x $c_1c_2c_3$ … hoặc 0X $c_1c_2c_3$ …

Trong đó c_i là một chữ số hệ 16.

Ví dụ :

Các hằng nguyên hệ 16 : 0xa3 ; 0Xa3 ; 0xA3 ; 0XA3 là như nhau.

Giá trị của nó trong hệ 10 là : 10*16 + 3 = 163.

2.3.3 Hằng ký tự

Là một ký tự được viết trong 2 dấu nháy đơn.

Chẳng hạn 'a' , 'A', '3', '+' . . . Trình biên dịch của C++ sẽ lưu trữ các hằng này bằng cách dùng các mã số ASCII (hệ 10) của nó, tương ứng là 97,65, 43, 51 . . .

- Ký tự có thể biểu diễn bởi hệ 8:

  Cách viết : '\$c_1c_2c_3$' , trong đó c_i là các ký hiệu trong hệ 8.

- Ký tự có thể biểu diễn bởi hệ 16:

  Cách viết : '\x$c_1c_2c_3$' hoặc '\X$c_1c_2c_3$' trong đó c_i là các ký hiệu trong hệ 16.

Ký tự	**Hệ 8**		**Hệ 16**		**Mã ASCII hệ 10**
	Mã ASCII	**Biểu diễn**	**Mã ASCII**	**Biểu diễn**	
'a'	**141**	**'\141'**	**61**	**'\x61'**	**97**
'A'	**101**	**'\101'**	**41**	**'\x41'**	**65**

- Đối với một số hằng ký tự đặc biệt, ta sử dụng cách viết sau (thêm dấu \) :

Cách viết	Ký tự
'\' '	'
'\" '	"
'\\'	\

‘\n’	\n (chuyển
‘\0’	\0 (NULL)
‘\t’	Tab
‘\b’	Backspace
‘\r’	CR (Về đầu
‘\f’	LF (sang trang)

Ghi chú :

- Hằng ký tự có thể tham gia vào các biểu thức như mọi số nguyên khác.

Ví dụ : ‘9’ -’0’ = 57 -48 = 9

- Cần phân biệt :

‘0’ : là ký số 0 có mã ASCII hệ 10 là 48.

‘\0’ : là ký tự NULL có mã ASCII hệ 10 là 0.

2.3.4 Hằng xâu ký tự (Chuỗi)

Là một dãy ký tự được bao trong 2 dấu nháy kép.

Ví dụ :

“Da Lat”

“” // Xâu rỗng

Ghi chú :

Xâu ký tự được lưu trử trong máy dưới dạng một mảng các ký tự. Trình biên dịch tự động thêm ký tự NULL ‘\0’ (được xem là dấu hiệu kết thúc xâu) vào cuối mỗi xâu.

2.3.5 Biểu thức hằng

Biểu thức hằng chỉ bao gồm các hằng. Các biểu thức như vậy được xác định vào lúc biên dịch.

2.3.6 Định nghĩa một hằng

Dùng chỉ thị #define (Có thể định nghĩa lại giá trị hằng) :

Cách viết : #define TÊN-HẰNG GIÁ_TRỊ_HẰNG

Tác dụng : TÊN-HẰNG sẽ được thay thế bởi GIÁ_TRỊ_HẰNG cho phần còn lại của văn bản chương trình.

Ví dụ :

#define MAX 100 // Thay thế MAX bằng 100

- Dùng từ khóa const (Không định nghĩa lại được giá trị hằng)

- Cú pháp : Const kiểu TÊN-HẰNG = GIÁ_TRỊ_HẰNG;
- Tác dụng : Cho phép định nghĩa một hằng ký hiệu có tên là TÊN-HẰNG biểu thị một gia trị là GIÁ_TRỊ_HẰNG và sau này không thể sửa đổi GIÁ_TRỊ_HẰNG của TÊN-HẰNG được .

Ví dụ :

Const double PI = 3.1416;

2.3.7 Biến

2.3.7.1 Định nghĩa

Biến là một phần bộ nhớ được đặt tên, được dùng, để giữ một giá trị mà có thể thay đổi trong chương trình.

Vậy biến gắn với tên và kiểu dữ liệu, có giá trị thay đổi trong quá trình tính toán.

2.3.7.2 Khai báo biến

Mỗi biến phải được khai báo trước khi sử dụng. Cú pháp khai báo như sau:

- Khai báo một biến:

  Kdl Bien;

- Khai báo nhiều biến cùng một kiểu:

  Kdl Bien1, Bien2, Bien3;

Trong đó:

- Kdl là kiểu dữ liệu nào đó như char, int, double,...
- Bien, Bien1,... là tên chỉ tên của biến.

Giữa Kdl và Tên biến phải cách nhau ít nhất 1 khoảng trắng. Trong phần tên biến, nếu có nhiều biến thì giữa 2 biến phải tách ra bởi dấu phảy (,).

2.3.7.3 Khởi đầu cho các biến

Nếu trong khai báo, ngay sau tên biến ta đặt dấu = (phép gán) và một giá trị dữ liệu tương ứng thì đó chính là cách vừa khai báo vừa khởi đầu cho 1 biến.

2.3.7.4 Lấy địa chỉ cho biến:

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến.

Để nhận địa chỉ biến ta dùng toán tử & với cú pháp: &Bien.

2.4 Các phép toán

Trong C++ có các loại toán tử: phép gán, các toán tử số học, các toán tử quan hệ và logic, các toán tử thao tác trên Bit...

2.4.1 Các toán tử số học:

C++ có 8 toán tử số học:

Toán tử	Ý nghĩa	Ví dụ	Ghi chú
-	Lấy đối	-a ; -(a+b)	Toán tử 1 ngôi
--	Tự giảm dần	--x	Toán tử 1 ngôi
++	Tự tăng dần	++x	Toán tử 1 ngôi
+	Cộng	a + b	Toán tử 2 ngôi
-	Trừ	a –b	Toán tử 2 ngôi
*	Nhân	a * b	Toán tử 2 ngôi
/	Chia	a /b	Toán tử 2 ngôi
%	Chia lấy phần dư	a%b	Toán tử 2 ngôi

Thứ tự ưu tiên các toán tử số học:

Ưu tiên của các toán tử số học được cho trong bảng sau đây theo thứ tự từ trên xuống dưới. Các toán tử cùng độ ưu tiên sẽ được thực hiện từ trên trái sang phải.

Ưu tiên	Toán tử
1	++ -- -
2	* / %

3	+ -

2.4.2 Toán tử quan hệ và logic

Các toán tử quan hệ và logic thường được sử dụng chung với nhau, được dùng để tạo ra các kết quả đúng, sai. Trong C++, mọi số khác 0 đều được coi là giá trị đúng (true), giá trị duy nhất sai (false) mang hình thức số 0.

2.4.2.1 Các toán tử quan hệ

Các toán tử quan hệ được dùng để so sánh 2 giá trị với nhau.

Sau đây là bảng các toán tử quan hệ và ý nghĩa của chúng:

Toán tử	Ý nghĩa	Ví dụ	
>	Lớn hơn	a > b	3>7 có giá trị 0
>=	Lớn hơn hay bằng	a >= b	3 >= 7 có gía trị 0
<	Nhỏ hơn	a < b	3<7 có giá trị 1
<=	Nhỏ hơn hay bằng	a <= b	3 <= 7 có gía trị 1
==	Bằng nhau	a == b	3 == 7 có giá trị 0
!=	Khác nhau	a != b	3 != 7 có giá trị 1

2.4.2.2 Các toán tử logic

Các toán tử logic được dùng để nối kết hai giá trị, hoặc trong trường hợp phủ định sẽ tạo một giá trị đảo ngược. Các giá trị có thể nguyên hay thực.

Trong C++ có 3 toán tử logic:

- Phép phủ định 1 ngôi: !
- Phép và (AND): &&
- - Phép hoặc (OR): ||

Ý nghĩa của các toán tử được cho trong bảng sau:

A	B	!a	a&&b	a||b
Khác không (1)	Khác không (1)	0	1	1
Khác không (1)	Bằng không	0	0	1
Bằng không	Khác không (1)	1	0	1

Bằng không	Bằng không	1	0	0

2.4.2.2.3 Thứ tự ưu tiên của các toán tử quan hệ và logic

Ưu tiên	Toán tử
1	!
2	> >= < <=
3	== !=
4	&& ||

2.4.3 Các Toán Tử Thao Tác Bit

Khác với các Lập trình cơ bản cấp cao khác, C++ cung cấp nhiều toán tử có thể tác động tới những bit thực sự ở bên trong 1 biến.

Các toán tử thao tác bit chỉ dùng cho số nguyên hoặc ký tự.

Các toán tử thao tác bit bao gồm:

Toán tử	Ý nghĩa	Ví dụ
&	Phép và (AND) theo bit	a&b
|	Phép hoặc (OR) theo bit	a|b
^	Phép hoặc loại trừ (XOR) theo bit	a^b
<<	Dịch trái (Shift left)	a << 4
>>	Dịch phải (Shift right)	a>>4
~	Lấy phần bù theo bit (Not)	~a

2.4.4 Các toán tử khác

2.4.4.1 Toán tử sizeof

Toán tử sizeof cho ta kích thước (tính theo byte) của 1 kiểu dữ liệu cũng như một đối tượng dữ liệu. Cách viết toán tử như sau:

- sizeof (kiểu dữ liệu)
- sizeof (đối tượng dữ liệu)

Kiểu dữ liệu có thể là các kiểu chuẩn như int, float hoặc kiểu dữ liệu được định nghĩa bằng từ khóa typedef.

Đối tượng dữ liệu có thể là biến, mảng, cấu trúc,...(tên của vùng nhớ dữ liệu).

2.4.4.2 Toán tử ( )

Dùng để xác định trình tự ưu tiên các thành phần trong biểu thức.

Nếu có nhiều toán tử () lồng nhau thì thực hiện ưu tiên từ trong ra ngoài.

Nếu có nhiều toán tử () rời nhau thì thực hiện từ trái sang phải.

2.4.4.3 Toán tử dấu phảy ‘ , ‘ (Comma operator)

Được dùng để tạo sự thi hành tuần tự cho các thao tác, thường dùng trong câu lệnh for hay biểu thức vế phải của câu lệnh gán.

Trong vế phải câu lệnh gán, thì giá trị toàn thể biểu thức là giá trị của biểu thức cuối cùng trong danh sách các biểu thức được tách biệt bởi dấu phảy.

2.5 Lệnh, khối lệnh

Một chương trình C++ có thể gồm nhiều tập tin chương trình nguồn, mỗi tập tin chương trình là một văn bản chứa một dãy các chỉ thị và các chỉ thị điều khiển biên dịch. Các chỉ thị được phân thành 2 loại:

Chỉ thị kiểu: Gồm định nghĩa các kiểu dữ liệu mới, biến, hằng và hàm.

Chỉ thị thực hiện (câu lệnh): Được định nghĩa bằng những phép toán hay việc xử lý thực hiện trên các biến của chương trình.

Tất cả các chỉ thị đều phải kết thúc bằng dấu ; (chấm phảy).

Cả 2 loại chỉ thị này có thể hợp với nhau bằng một cú pháp qui định để hình thành một chỉ thị duy nhất được gọi là khối lệnh. Một khối lệnh được đặt trong cặp dấu ngoặc nhọn: { các chỉ thị }

Sơ đồ tổng quát của chương trình C++

Dạng thức tổng quát:

```
// Các chỉ thị điều khiển biên dịch
// Các định nghĩa toàn cục
// Khai báo nguyên mẫu các hàm
// Hàm main
int main() // void main()
{
// dãy tuần tự các lệnh
```

}

//Phần định nghĩa hàm.
KDL Ham(Danh_Sach_Cac_Doi)
{
// dãy tuần tự các lệnh
}
. . . . .

Một số quy tắc cần nhớ khi viết chương trình

- Quy tắc 1: Mỗi dòng có thể viết 1 hay nhiều chỉ thị.
- Quy tắc 2: Mỗi chỉ thị phải kết thúc bằng dấu chấm phảy (;).
- Quy tắc 3: Quy tắc viết lời giải thích. Các lời giải thích viết:
  - Trên nhiều dòng, một dòng hoặc trên 1 phần của dòng phải đặt vào giữa các dấu /* và */.
  - Trên một dòng hoặc trên phần còn lại của một dòng phải đặt sau //

  Các lời giải thích được trình biên dịch bỏ qua.
- Quy tắc 4: Quy tắc sử dụng các hàm chuẩn:

  Trước khi sử dụng một hàm chuẩn nào cần phải biết nó nằm trong tập tin thư viện nào của C++ để khai báo, và khai báo bằng chỉ thị biên dịch #include:

  #include <Ten_Tap_Tin> // không có dấu chấm phảy như câu lệnh

  Chẳng hạn, khi sử dụng hàm cout, cin. Vì các hàm này nằm trong thư viện vào ra chuẩn iostream.h, nên ta cần khai báo:

  #include < iostream.h >

2.6 Câu lệnh gán, lệnh xuất nhập

Lệnh gán : Có dạng:

b = bt;

Trong đó b là biến. bt là một biểu thức (một công thức toán học nào đó). Trước tiên tính biểu thức bt và sau đó gán giá trị tính được cho biến b.

Lệnh xuất nhập :

1. Nhập dữ liệu từ bàn phím :

Lệnh nhập có thể mô tả như sau:

cin >> Bien; //Nhập giá trị 1 biến

cin >> Bien1 >> Bien2 >> ... >> Bienn;//Nhập giá trị n biến.

Trong đó:

cin là đối tượng được khai báo trong <iostream.h>.

>> là toán tử nhập.

Biến-i là biến mà đối với loại của nó >> được định nghĩa.

2. Xuất dữ liệu ra màn hình :

Lệnh xuất có thể mô tả như sau:

cout << Bt; // cho 1 biểu thức

cout << Bt1 << Bt2 << ... << Btn;//n biểu thức

Trong đó:

cout là đối tượng đã được khai báo trong <iostream.h>.

<< là toán tử xuất.

Biểu-Thức-i là biểu thức mà đối với kiểu của nó << được định nghĩa.

Từ khóa: typedef

Từ khóa typedef được dùng để đổi lại tên một kiểu dữ liệu đã có như char, int, float, mảng... thành một tên mới.

Cách viết là đặt từ khóa typedef vào trước 1 khai báo thông thường.

2.7 Cách chạy chương trình

Chương trình đầu tiên

```
//Chuong trinh C++ dau tien trong moi truong Windows
// tạo bằng Windows 32 Console Application trong VC++ 6.0
#include <iostream.h>
int main()
{
    cout<<"Chuong trinh C++ dau tien trong moi truong Windows!\n";
    return 0;
}
```

Chương trình sẽ xuất ra màn hình chuỗi sau:
Chuong trinh C++ dau tien trong moi truong Windows!

Bài tập

Bài 1:

Viết chương trình khai báo 3 biến : x kiểu số thực, c kiểu ký tự, i kiểu số nguyên.Nhập, xuất giá trị cho các biến đó.

Bài 2:

Viết chương trình nhập vào 2 biến số nguyên x, y. Tính giá trị của x+y, xuất kết quả ra màn hình

Hướng dẫn:

- Khai báo 3 biến kiểu số nguyên: x,y và z (z: là biến lưu kết quả).
- Nhập giá trị cho x,y.
- Thực hiện phép gán: z = x+y.
- Xuất giá z ra màn hình.

Bài 3: Viết chương trình tính chu vi, diện tích của hình chữ nhật với chiều dài, rộng nhập từ bàn phím.

Bài 4: Khai báo hằng PI có giá trị 3.14 sử dụng hằng PI để tính diện tích hình tròn với bán kính được nhập từ bàn phím.

Bài 5: Khai báo hằng MAX có giá trị 60. Nhập số giây, quy đổi thời gian giây thành giờ, phút, giây. Xuất kết quả ra màn hình dưới dạng: gio:phut:giay

Ví dụ: số giây nhập = 3770

- *số giờ =(số_giây_nhập)/3600 = 1*
- *số phút = (số_giây_nhập%3600)/MAX=2*
- *số giây =(số_giây_nhập %3600)%MAX=50*

Xuất ra màn hình dưới dạng: 1:2:50

Hướng dẫn:

- Khai báo 4 biến kiểu số nguyên.
- Nhập vào số giây.

- Thực hiện phép toán quy đổi.
- Xuất kết quả ra màn hình.

Bài 6: Khai báo biến x, y kiểu số nguyên. Khởi gán x =20, y=6. Thực hiện các câu lệnh

sau và xuất kết quả của x, y trước và sau khi thực hiện từng câu lệnh này.

x++

x--

++x

--x

x=x/y

y= x%y

x=x*y

Bài tập 7:

Viết chương trình nhập vào một số nguyên và xuất ra màn hình

(Chú ý: bạn hãy nhập số lớn và giải thích kết quả).

Bài 8:

Viết chương trình tính và xuất ra diện tích tam giác:

S = $\sqrt{p(p-a)(p-b)(p-c)}$, a, b ,c được nhập từ bàn phím, p = $\frac{a+b+c}{2}$

(Sử dụng hàm tính căn bậc 2: *sqrt* trong thư viện math.h)

Bài 9:

Viết chương trình tính $x^2 + y^5$, với x và y là 2 số thực được nhập từ bàn phím.

(Sử dụng hàm tính tính mũ: *pow* trong thư viện math.h)

Bài 10:

Viết chương trình tính x^n , với x là số thực và n là số nguyên được nhập từ bàn phím. (Sử dụng hàm tính tính mũ: *pow* trong thư viện math.h)

CHƯƠNG 3 CÁC LỆNH CẤU TRÚC

Mã bài: MH 09.3

Mục tiêu của bài:

- Trình bày được ý nghĩa, cú pháp của các cấu trúc lệnh;
- Viết được một số đoạn chương trình cụ thể để thực hiện các yêu cầu đơn giản;
- Nghiêm túc, tỉ mỉ, sáng tạo trong quá trình học và vận dụng vào thực hành.

3.1 Câu lệnh if

3.1.1 Cú pháp

Dạng 1: if (Bt) Kl	 Nếu biểu thức Bt đúng thực hiện Kl

Dạng 2:
if (Bt)
Kl1
else
Kl2
	
	Nếu biểu thức Bt đúng
	thực hiện Kl1
	Ngược lại
	thực hiện Kl2

- Bt: là biểu thức có giá trị Đ (khác 0), hay sai (bằng 0).
- Kl: Khối lệnh.

3.1.2 Hoạt động

Trước tiên biểu thức Bt được xác định giá trị.

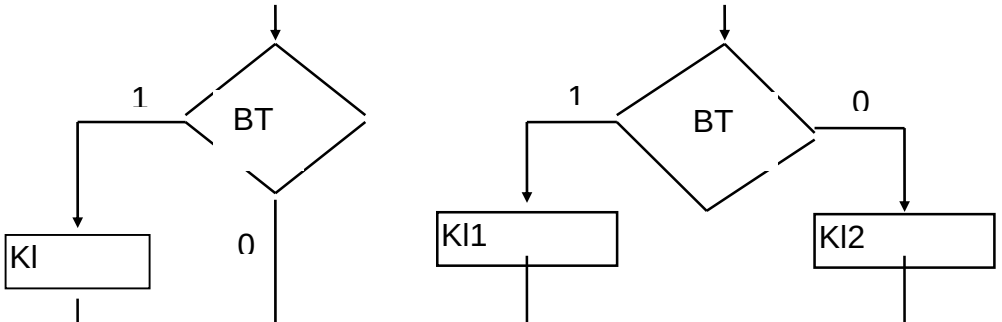
- Dạng 1:

  Nếu Bt đúng (có giá trị khác 0) thì thực hiện Kl. Nếu sai (giá trị bằng 0) thì khối lệnh được bỏ qua.

- Dạng 2:

  Nếu Bt đúng (có giá trị khác 0) thì thực hiện Kl1. Nếu sai (giá trị bằng 0) thì thực hiện Kl2.

3.1.3 Lưu đồ:



Ghi chú:

Nếu có nhiều cấu trúc if - else lồng nhau, để xác định một "else" là của "if" nào ta theo qui tắc: "else" là của một "if" gần với nó nhất mà không có "else".

Tức là:	Tương đương với:

if (Bt1)
  if (Bt2)
  Kl2
  else
  Kl3
else
  Kl1
	if (Bt1)
	{ if (Bt2)
	  Kl2
	  else
	  Kl3
	}
	else
	  Kl1

3.2 Câu lệnh switch

Câu lệnh cho phép chọn một trong nhiều nhánh rẽ.

3.2.1 Cú pháp

```
switch (Bt)
{
    case H1:    NL1
                [ break;]
```

```
        case H2:    NL2
                    [ break;]
            .....
        case Hn:    NLn
                    [ break;]
        [default:   NLn+1]
    }
```

Trong đó Bt là nguyên, các Hk có thể là hằng nguyên, hằng ký tự, hoặc biểu thức hằng.

3.2.2 Hoạt động của câu lệnh switch

- Trước tiên biểu thức nguyên được tính trị.
- Nếu trị biểu thức bằng Hk thì máy sẽ thực hiện NLk.

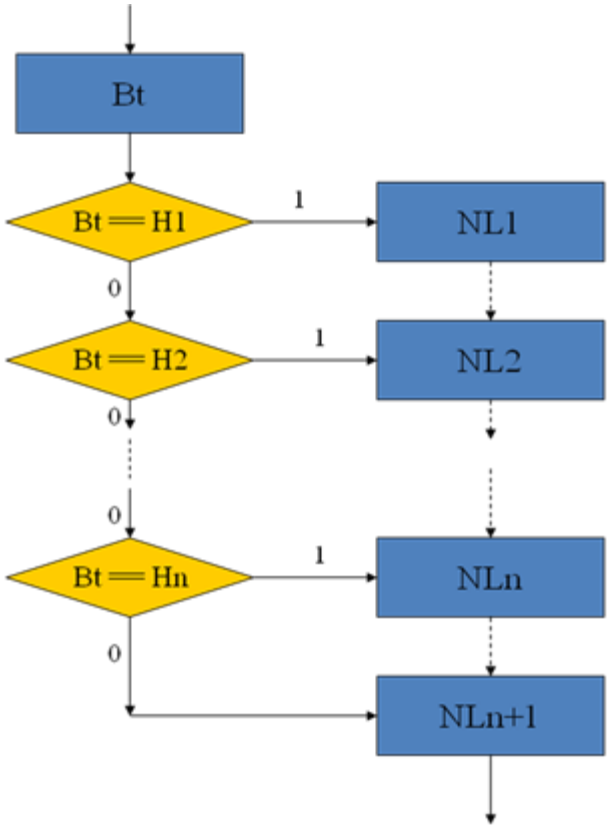
Sau đó:

  - Nếu không có câu lệnh break thì thực hiện tiếp NL k+1 của nhãn case Hk+1... cho tới khi gặp lệnh break hoặc gặp dấu } cuối cùng thì ra khỏi switch.
  - Nếu có câu lệnh break thì ra khỏi switch.

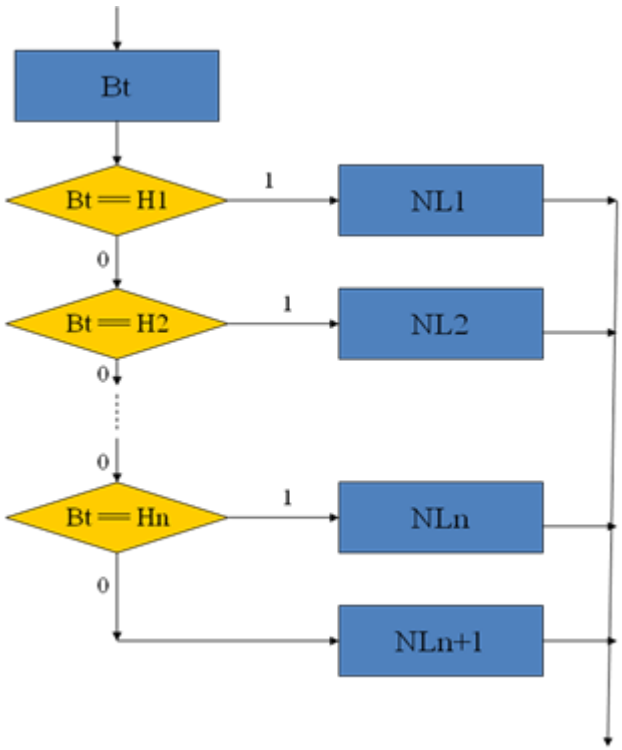
- Khi giá trị biểu thức khác tất cả các Hk thì:
  - Khi có default thì máy nhảy tới thực hiện NLn+1 trong default.
  - Khi không có default thì ra khỏi câu lệnh switch.

3.2.3 Lưu đồ: (Có thành phần default)

a) Không có câu lệnh break cuối mỗi nhóm lệnh



b) Có câu lệnh break cuối mỗi nhóm lệnh:



3.3 Câu lệnh for

3.3.1 Cú pháp

Câu lệnh for cho ta một cách xây dựng vòng lặp, có dạng sau:

```
for(bt1 ; bt2 ; bt3)
        Khối lệnh     // Câu lệnh
```

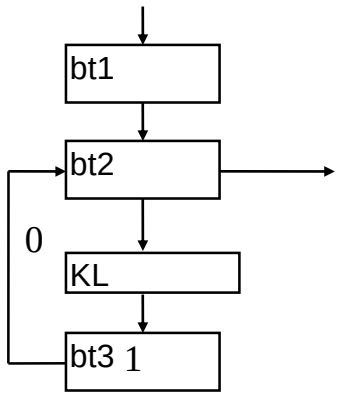
Trong đó:

- Mỗi thành phần bt là một biểu thức hay là một dãy các biểu thức được phân tách bởi dấu phảy.
- Bất kỳ một thành phần nào trong 3 thành phần bt cũng có thể vắng mặt, nhưng các dấu chấm phảy (;) luôn luôn có.

Chức năng của các thành phần bt thường là:

- bt1 là một phép gán để tạo ra giá trị ban đầu cho biến điều khiển.
- bt2 là một một quan hệ logic biểu thị điều kiện để tiếp tục vòng lặp.
- bt3 là một phép gán dùng để thay đổi giá trị của biến điều khiển.

3.3.2 Hoạt động của câu lệnh for

Hoạt động theo các bước: 1. Xác định biểu thức 1. 2. Xác định biểu thức 2. 3. Căn cứ vào giá trị đúng (khác 0), sai (bằng 0), sẽ lựa chọn một trong 2 nhánh: - Nếu biểu thức 2 sai , sẽ ra khỏi câu lệnh for và chuyển đến câu lệnh sau thân for. - Nếu biểu thức 2 đúng, sẽ thực hiện các câu lệnh trong thân for.Khi gặp dấu ngoặc nhọn đóng } cuối cùngcủa thân for,hoặc gặp câu lệnh continue máy sẽ chuyển sang bước 4 (khởi đầu lại). 4. Tính biểu thức 3, sau đó quay trở lại bước 2 để bắt đầu một vòng mới của vòng lặp.	

3.4 Câu lệnh while (Lặp với điều kiện được kiểm tra trước)

3.4.1 Cú pháp

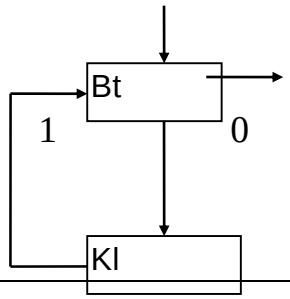
while (Bt)

Kl // Thân vòng lặp

Trong đó:

- Bt có thể là 1 biểu thức hay là một dãy các biểu thức được phân tách nhau bởi dấu phảy.
- Giá trị của Bt là Đúng (khác 0) hoặc sai (bằng 0).

3.4.2 Lưu đồ

Ngữ nghĩa: Trong khi Bt còn đúng Thực hiện Kl	

3.4.3 Hoạt động của câu lệnh while

Theo trình tự:

1. Xác định giá trị của bt.

2. Tùy thuộc vào tính đúng sai của bt, máy sẽ lựa chọn 1 trong 2 nhánh:

a. Nếu bt có giá trị 0 (sai) máy sẽ ra khỏi vòng lặp và chuyển tới câu lệnh sau thân while.

b. Nếu bt có giá trị khác 0 (đúng) máy sẽ thực hiện các câu lệnh trong thân while. Khi gặp dấu ngoặc nhọn } đóng cuối cùng của thân while máy sẽ trở lại bước 1; hoặc gặp câu lệnh continue máy sẽ chuyển đến đầu một vòng lặp mới của vòng lặp, tức là máy sẽ bỏ qua các câu lệnh còn lại trong thân vòng lặp để trở về bước tính và kiểm tra bt.

3.5 Câu lệnh do.. while (Lặp với điều kiện được kiểm tra sau)

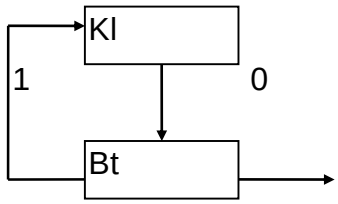
3.5.1 Cú pháp

```
do
    Kl      // thân vòng lặp
while(bt);
```

Trong đó:

- Bt có thể là 1 biểu thức hay là một dãy các biểu thức được phân tách bởi dấu phảy.
- Giá trị của Bt là Đúng (khác 0) hoặc sai (bằng 0).

3.5.2 Lưu đồ

Ngữ nghĩa: Thực hiện Kl trong khi Bt còn đúng	

3.5.3 Hoạt động của câu lệnh do

Theo trình tự:

1. Thực hiện khối lệnh trong thân do.

2. Khi gặp dấu ngoặc nhọn } đóng cuối cùng của thân do máy sẽ tính và xác định giá trị của Bt sau từ khóa while.

3. Tùy thuộc vào tính đúng sai của bt, máy sẽ lựa chọn 1 trong 2 nhánh:

a. Nếu Bt có giá trị bằng 0 (sai) máy sẽ ra khỏi vòng lặp và chuyển tới câu lệnh sau thân do.

b. Nếu Bt có giá trị khác 0 (đúng) máy sẽ trở về bước 1 để tiếp tục thực hiện vòng mới của vòng lặp.

3.6 Câu lệnh goto và nhãn

3.6.1 Nhãn

Nhãn là một tên và có dấu hai chấm (:) đứng sau.

Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình.

Chẳng hạn: tiep_tuc: s + = x;

nghĩa là: tiep_tuc là nhãn câu lệnh gán s += x.

3.6.2 Câu lệnh goto

1. Cú pháp:

goto nhãn;

2. Tác dụng:

chương trình sẽ nhảy tới thực hiện câu lệnh có nhãn viết sau từ khóa goto;

Ghi chú:

1. Câu lệnh goto và nhãn chỉ nằm trong 1 hàm. goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân 1 hàm. Nó không thể nhảy từ hàm này sang hàm khác.

2. Không cho phép dùng câu lệnh goto nhảy từ ngòai khối lệnh vào trong khối lệnh. Nhưng ngược lại, tức là nhảy từ trong ra ngoài khối lệnh là hợp lệ.

3. Câu lệnh goto thường được dùng để thóat khỏi các câu lệnh switch, và các vòng lặp hoặc kết hợp với if tạo ra vòng lặp.

Ví dụ:

```
nhan: clrscr();
. . .
goto nhan;
```

3.7 Các câu lệnh break, continue

3.7.1 break

Break cho phép ra khỏi switch , và các vòng lặp for, while, do mà không cần kiểm tra điều kiện kết thúc của vòng lặp.

Lệnh break cho phép thoát ra ngay khỏi vòng lặp bên trong nhất chứa lệnh break;

Mọi câu lệnh break có thể thay bằng câu lệnh goto và nhãn thích hợp.

3.7.2 continue

Lệnh continue tạo ra việc bắt đầu lặp lại của vòng lặp chứa nó.

Trong while và do, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra.

Đối với for, điều khiển được chuyển về bước khởi đầu lại. Continue chỉ áp dụng cho vòng lặp, không cho switch.

3.8 Câu lệnh rổng

Lệnh rổng là lệnh chỉ có dấu chấm phảy (;)

(Lệnh này thường được sử dụng trong thân các vòng lặp mà ở đó không muốn có 1 lệnh nào.)

3.9 Vòng lặp vô hạn

Thường có các dạng sau:

a) Dạng for:

```
for (; ;)
    khối lệnh
```

Mệnh đề trong phần thân vòng lặp thường là mệnh đề kép chứa:

- Những mệnh đề mà nó thực hiện lặp lại để giải quyết bài toán.

- Một mệnh đề mà nó sẽ kết thúc sự thực hiện vòng lặp khi một điều kiện nào đó được thỏa. Mệnh đề này thường là if kết hợp với break.

b) Dạng while:

```
while(1)
    khối lệnh
```

c) Dạng do..while:

```
do
    khối lệnh
while(1);
```

Khi sử dụng vòng lặp ta cần lưu ý đến điều kiện lặp có thể dừng vòng lặp được không. Các vòng lặp dạng trên muốn dừng, cần phải có sự can thiệp của các câu lệnh break, goto, return …

Bài tập

Bài 1: Tính tổng n số nguyên dương đầu tiên S=1+2+3...n

Bài 2: viết chương trình tính n!. n!=1*2*3*…*n.

Bài 3: Tính tổng n số nguyên dương lẽ: S=1+3+5...+n. n lẽ.

Bài 4: Tính tổng n số nguyên dương chẳn: S=2+4+6...+n. n chẳn.

Bài 5: Sử dụng cấu trúc while và do…while để viết chương trình tính:

- Tính tổng n số nguyên dương đầu tiên S=1+2+3...n
- Tính tổng n! với n là số nguyên n!=1*2*3*...*n.

CHƯƠNG 4 HÀM

Mã bài : MH 09.4

Mục tiêu của bài:

– Trình bày được khái niệm, phân loại hàm;

– Trình bày được qui tắc xây dựng hàm, cách sử dụng hàm trong một chương trình;

– Trình bày được khái niệm tham số, tham trị và cách truyền tham số;

– Viết được một số hàm đơn giản và sử dụng các hàm đó trong các chương trình cụ thể;

– Nghiêm túc, tỉ mỉ, sáng tạo trong quá trình học và vận dụng vào thực hành.

4.1 Khái niệm

C/ C++ đưa ra khái niệm hàm và trở thành công cụ mạnh mẽ để hỗ trợ cho phương pháp lập trình có cấu trúc. Hàm có thể xem là một đơn vị độc lập của chương trình.

Các hàm có vai trò ngang nhau, vì vậy không cho phép xây dựng một hàm bên trong các hàm khác.

Các hàm thường che dấu những chi tiết thực hiện đối với các phần khác trong chương trình, do đó làm chương trình sáng sủa, dễ sửa đổi.

Một hàm có thể là:

- Nằm ngay trong modul văn bản (có các khai báo,các lệnh có trong hàm) hoặc được đưa một cách tự động vào văn bản của chương trình (bằng đường dẫn #include) hay được dịch riêng rẽ (sẽ được nối kết vào chương trình trong giai đoạn liên kết)
- Được gọi từ chương trình chính, hoặc từ một hàm khác, hoặc từ chính nó (đệ qui).
- Có hay không có đối (tham số hình thức).
- Có hay không có giá trị trả về.
- . . .

Một hàm:

- Chỉ có 1 đầu vào ({).
- Có thể có nhiều điểm ra (return hay }).

4.2 Quy tắc xây dựng một hàm

4.2.1 Cấu trúc của 1 hàm

Mọi hàm đều có dạng:

[Kdl] Ten_Ham ([danh_sách_kiểu_và_Đối]) // Dòng tiêu đề

{

// Các chỉ thị về kiểu

// Các câu lệnh;

[return [biểu_thức];]

}

Trong đó:

a) Dòng tiêu đề:

Chứa các thông tin:

- Kdl: Kiểu dữ liệu của hàm,
- Ten_Ham: Tên của hàm, kiểu và tên đối.
- Danh_sách_kiểu_và_Đối: Là các đối và kiểu tương ứng của nó.

Cuối dòng tiêu đề không có dấu chấm phẩy (;) như kết thúc câu lệnh.

1. Kdl: Có thể có hoặc không.

- Trương hợp không:

Không có kiểu dữ liệu của hàm, Một cách ngầm định C++ coi đó là kiểu int.

- Trường hợp có:

Kdl là kiểu dữ liệu của hàm, có thể là bất kỳ kiểu dữ liệu nào ngoại trừ mảng. C++ sẽ dùng từ khóa void để chỉ kiểu dữ liệu của hàm trong trường hợp hàm không trả về giá trị nào cả. Cần lưu ý là:

- Nếu hàm trả về kiểu int, ta có thể không khai báo (ngầm định là kiểu int).
- Nếu trả về khác kiểu int, ta phải khai báo tường minh Kiểu dữ liệu của hàm.

2. Ten_Ham: Là tên của hàm do người lập trình tự đặt theo quy tắc đặt tên.

3. Danh sách kiểu và đối:

Có thể có hoặc không.

- *Trường hợp không*: Nếu không có đối thì vẫn phải giữ các dấu ngoặc tròn ().Trong trường hợp này có thể thay bằng từ khóa void.
- *Trường hợp có*: Đó là các đối của hàm. Trước mỗi đối có kiểu dữ liêu tương ứng của nó, nếu có nhiều đối và kiểu tương ứng của nó, thì chúng phải cách nhau dấu phẩy(,).

Thân hàm:

Thân hàm bắt đấu bằng dấu ngoặc nhọn mở{, tiếp theo là các chỉ thị về kiểu (nếu có), các câu lệnh trong đó có thể có hay không câu lệnh return và kết thúc bằng dấu ngoặc nhọn đóng.

4.3 Sử dụng hàm

Hàm được sử dụng thông qua lời gọi hàm.

- Lời gọi hàm được viết như sau:

  Ten_Ham([Danh sách các tham số thực])

Trong đó:

- Số tham số thực phải bằng số các đối.
- Kiểu của tham số thực phải phù hợp với kiểu của đối tương ứng.

- Cách sử dụng như sau:
  - Nếu hàm trả về kiểu void, Viết lời gọi hàm như câu lệnh (thêm dấu ; cuối cùng), tức là:

    Ten_Ham([Danh sách các tham số thực]);
  - Nếu hàm trả về khác kiểu void, lời gọi hàm được sử dụng:
    - Như một toán hạng trong biểu thức.
    - Vế phải câu lệnh gán.
    - In giá trị của hàm.
    - . . .

Khai báo nguyên mẫu của hàm (prototype).

Trước khi sử dụng một hàm, ta có thể khai báo hoặc không khai báo nguyên mẫu của hàm trong chương trình.Vị trí khai báo có thể là ngoài tất cả các hàm, hoặc trong hàm... thường là đầu chương trình tại các định nghĩa toàn cục.

Dạng khai báo nguyên mẫu hàm là:

Kdl tên_hàm (Danh_sách_Kiểu_và _Đối); // Có dấu ;

Ví dụ:

float max(float a, float b);

Đối với C++, Các trường hợp sau nhất thiết phải khai báo nguyên mẫu:

- Vị trí của hàm đặt sau hàm main().
- Các hàm - không phải hàm main() - gọi lẫn nhau.

Dù rằng có các trường hợp không nhất thiết phải khai báo nguyên mẫu, nhưng tốt hơn cả là ta vẫn khai báo để trình biên dịch dễ phát hiện lỗi khi gọi hàm.

4.4 Nguyên tắc hoạt động của hàm

Khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện. Quá trình diễn ra theo trình tự:

a) Cấp phát bộ nhớ cho các đối và các biến địa phương.

b) Gán giá trị của các tham số thực cho các đối tương ứng.

c) Thực hiện các câu lệnh trong thân hàm.

d) Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xóa các đối, các biến địa phương và ra khỏi hàm.

4.5 Cánh truyền tham số

Có 2 cách: Truyền bằng trị và truyền bằng biến.

Truyền bằng trị: Tham số của hàm luôn được truyền theo tham trị. Điều này có nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các đối. Khi chúng ta gọi hàm với các tham số, những gì chúng ta truyền cho hàm là các giá trị chứ không phải bản thân các biến. Ví dụ, giả sử chúng ta gọi hàm cong như sau:

```
int x=5, y=3, z;
z = cong ( x , y );
```

Truyền bằng biến: Để thay đổi giá trị của biến qua lời gọi hàm, thì chúng ta phải truyền tham số theo dạng tham số biến

Ví dụ:

```
// truyền tham số theo dạng tham số biến
#include <stdio.h>
#include <conio.h>
void nhan2(int &a, int &b, int c);
void main()
{
```

```
 int x=4, y =10,c=8;
 clrscr();
 printf("Truoc goi ham x=%d, y=%d, c=%d \n",x,y,c);
 nhan2(x,y,c);
 printf("Sau goi ham x=%d, y=%d, c=%d \n",x,y,c);
 getch();
}
```

4.6 Câu lệnh return

Trong thân hàm có thể có hoặc không có câu lệnh này. Trong trường hợp có, có thể có một câu lệnh return, hoặc nhiều câu lệnh return ở những nơi khác nhau,

Nếu không có câu lệnh return thì chương trình sẽ ra khỏi hàm khi gặp dấu ngoặc nhọn đóng cuối cùng } của thân hàm để trở về nơi gọi nó.

Nếu có, Câu lệnh "return" là cơ chế chuyển giá trị từ hàm được gọi về nơi gọi. Khi gặp câu lệnh return máy sẽ không thực hiện các câu lệnh sau nó trong hàm chứa câu lệnh này.

Dạng tổng quát của câu lệnh này là:

return [Bt];

Ý nghĩa của các dạng có thể minh họa như sau:

- Dạng 1: **return;**

  Dùng để thoát ra khỏi 1 hàm và trở về hàm đã gọi nó, mà không trả về một giá trị nào. Trường hợp này khai báo kiểu dữ liệu của hàm là void. Câu lệnh return có thể dùng để ra khỏi thân switch, các vòng lặp.

- Dạng 2: **return Bt;**

  hoặc **return (Bt);**

  Giá trị của biểu thức Bt sẽ được chuyển kiểu cho phù hợp với kiểu của hàm trước khi gán cho hàm, và ra khỏi hàm chuyển về nơi gọi nó.

Hàm inline

Hàm có ưu điểm là được dùng lại chứ không phải viết lại cho các ứng dụng khác, tuy nhiên nó cũng có một nhược điểm là làm chậm chương trình một các đáng kể, ngay cả khi thân hàm chỉ có một vài câu lệnh, vì phải sao các đối, cất giữ trên thanh ghi, chương trình phải nhảy tới một vị trí mới.

Hàm inline đưa ra một giải pháp khắc phục tình trạng này, và xác định bởi từ khóa inline.

Một hàm được xác định là inline thì sẽ mở rộng “”trên dòng” tại điểm gọi, và làm cho phí tổn của hàm được loại bỏ.

Cách viết là chỉ thêm inline trước kiểu trả về của hàm.

4.7 Một số thư viện trong C++

Một trong những tính năng thuận tiện của C++ là cung cấp kèm theo một số lượng lớn các thư viện. Thư viện chứa các hằng, các hàm thực hiện các thao tác chuyên biệt nào đó. Muốn sử dụng thư viện, ta chỉ cần tham chiếu đến tập tin giao diện gọi là tập tin tiêu đề (tập tin header, có đuôi dạng.h) của thư viện đó, bằng cách sử dụng chỉ thị: #include <Ten_Tep>

4.8 Hàm đệ quy

4.8.1 Khái niệm hàm đệ quy

Đệ qui là 1 công cụ rất thường dùng trong khoa học máy tính và trong toán học để giải quyết các vấn đề. Trước hết, chúng ta hãy khảo sát thế nào là một vấn đề có đệ qui qua ví dụ sau:

Tính S(n) = 1 +2 +3 +4+ ... +n

Ta nhận thấy rằng, công thức trên có thể diễn đạt lại như sau:

S(n) = S(n-1) + n, và

S(n-1) = S(n-2) + (n-1)

.....

S(2) = S(1) + 2

S(1) = 1

Như vậy, *một vấn đề có đệ qui là vấn đề được định nghĩa lại bằng chính nó.*

4.8.2 Các bài toán dùng đệ quy

Một cách tổng quát, một chương trình đệ qui có thể được biểu diễn như bộ P gồm các mệnh đề cơ sở S (không chứa P) và bản thân P:

$P \equiv P(S_i, P)$

Để tính S(n): ta có kết quả của S(1), thay nó vào S(2), có S(2) ta thay nó vào S(3)...., cứ như vậy có S(n-1) ta sẽ tính được S(n)

Cũng như các lệnh lặp, các thủ tục đệ qui cũng có thể thực hiện các tính toán

không kết thúc, vì vậy ta phải xét đến vấn đề kết thúc các tính toán trong giải thuật đệ qui. Rõ ràng 1 thủ tục P được gọi đệ qui chỉ khi nào thỏa 1 điều kiện B, và dĩ nhiên điều kiện B này phải không được thỏa mãn tại 1 thời điểm nào đó. Như vậy mô hình về các giải thuật đệ qui là:

$P \equiv$ if (B) $P(S_i, P)$

hay $P \equiv P(S_i,$ if (B) $P)$.

Thông thường trong các vòng lặp while, để đảm bảo cho vòng lặp kết thúc ta phải định nghĩa một hàm f(x) (x là 1 biến trong chương trình) sao cho nó phải trả về trị bằng 0 tại một thời điểm nào đó. Tương tự như vậy, chương trình đệ qui cũng có thể được chứng minh là sẽ dừng bằng cách chứng minh rằng hàm f(x) sẽ giảm sau mỗi lần thực hiện. Một cách thường làm là kết hợp một giá trị n với P và gọi P một cách đệ qui với giá trị tham số là n-1. Điều kiện B bây giờ là n > 0 thì sẽ đảm bảo được sự kết thúc của giải thuật đệ qui. Như vậy, ta có mô hình đệ qui mới:

$P(n) \equiv$ if (n >0) $P(S_i, P(n-1))$

Hay $P \equiv P(S_i,$ if (n>0) $P(n-1))$

4.8.2 Các ví dụ về hàm đệ quy

Ví dụ 1: Hàm giai thừa:

$$n! = \begin{cases} 1*2*3*.....*(n-1)*n & , n>0 \\ 1 & , n=0 \end{cases}$$

$$n! = \begin{cases} n*(n-1)! & , n>0 \\ 1 & , n= 0 \end{cases}$$

Nhận xét:

- Theo công thức trên, ta nhận thấy trong định nghĩa của n giai thừa (n!) có định nghĩa lại chính nó nên hàm giai thừa có đệ qui.

- Điều kiện dừng tính hàm giai thừa là n=0, khi đó n! = 1

- Hàm đệ qui:

```
long giaithua(int n)
{
  if (n == 0)
    return(1);
```

```
        else
            return(n * giaithua(n-1));
    }
```

hay:

```
    long giaithua(int n)
    {  return ((n==0) ? 1 : n*giaithua(n-1));
    }
```

- Hàm không đệ qui:

```
    long giaithua (int n)
    {  long gt=1;
       for (int i=1; i<=n; i++)
            gt= gt * i ;
      return (gt);
    }
```

Ví dụ 2: Hàm FIBONACCI:

$$F_n = \begin{cases} 1 & ; n = 0,1 \\ F_{n-1} + F_{n-2} & ; n > 1 \end{cases}$$

Nhận xét:

- Theo định nghĩa trên, hàm Fibonacci có lời gọi đệ qui.
- Quá trình tính dừng lại khi n= 1
- Hàm đệ qui:

```
    long fib(int n)
    {  if (n==0 || n==1)
         return 1 ;
       else return(fib(n-1) + fib(n-2));
    }
```

- Hàm không đệ qui:

```
    long fib(int n)
    {  long kq, Fn_1, Fn_2;
       kq = 1;
```

```
    if (n > 1)
    {
      Fn_1 = 1;
      Fn_2 = 1;
      for (int i=2; i<=n; i++)
      {
           kq = Fn_1 + Fn_2 ;
           Fn_2 = Fn_1;
           Fn_1 = kq;
      }
    }
      return (kq);
}
```

Bài tập

Viết chương trình thực hiện các thao tác trên số nguyên dương . Yêu cầu của chương trình là : viết 3 hàm tinhs1, tinhs2, tinhs3

Với:

1. Tính S1 = 1+2+3+4+...+ n , n là số nguyên
2. Tính S2 = 1+3+5+…+n (n lẽ)
3. Tính S3 = 2+4+6+8+..+n (n chẳn)

CHƯƠNG 5: MẢNG

Mã bài : MH 09.5

Mục tiêu của bài:

- Trình bày được khái niệm mảng, phân loại, cách khai báo mảng;
- Thực hiện cách truy xuất với các phần tử của mảng;
- Viết được các chương trình thực hiện các thao tác truy xuất trên mảng;
- Nghiêm túc, tỉ mỉ, sáng tạo trong quá trình học và vận dụng vào thực hành.

5.1. Khái niệm

- Mảng là một tập các biến cùng kiểu được gọi chung bằng một tên.
- Các phần tử của mảng được được truy cập đến bởi chỉ số (index) của mảng.
- Mảng có thể có 1 chiều hay nhiều chiều. Dữ liệu của mảng được chứa trong một vùng nhớ liên tục.

5.2 Mảng 1 chiều

I) Khai báo:

KDL Ten_Mang[KT];

Trong đó:

- KDL là kiểu dữ liệu của mảng, có thể là char, int, float,...
- Ten_Mang là một tên, chỉ tên của mảng.
- KT: Là một số nguyên dương chỉ kích thước khai báo của mảng: xác định số các phần tử của mảng.

II) Chỉ số của mảng:

Mỗi phần tử của mảng được xác định bởi chỉ số của mảng. Chỉ số của mảng phải có giá trị int không vượt quá kích thước của mảng. Chỉ số đầu tiên của mảng luôn là 0. Ký pháp a[i] để chỉ phần tử thứ i của mảng a.

Các phần tử của a được đánh số (mặc định) bởi: a[0], a[1],...

III) Lấy địa chỉ của mảng:

Các phần tử của mảng 1 chiều có địa chỉ liên tiếp nhau trong bộ nhớ.

Lấy địa chỉ của phần tử mảng 1 chiều bằng phép toán &, với cú pháp:

&a[i] // a[i] là phần tử thứ i của mảng a

Ghi chú: Địa chỉ đầu của mảng là tên mảng, vậy ta có: a == &a[0]

IV) Kích thước bộ nhớ (số bytes) được sử dụng để lưu trử mảng 1 chiều là:

Tong (Bytes) = sizeof(KDL)* KT

V) Một số thao tác thường gặp trên mảng một chiều:

1. Nhập và xuất dữ liệu cho mảng 1 chiều:

Thường liên kết với vòng lặp for.

2. Sắp xếp một mảng theo thứ tự tăng hay giảm.

3. Gán mảng:

Hai mảng a, b cùng kiểu, cũng không thể thực hiện trực tiếp việc gán a cho b bằng câu lệnh gán:

b = a;

Ta chỉ thực hiện được việc gán a cho b bằng cách gán giá trị từng phần tử của a tương ứng cho từng phần tử của b.

VI) Hàm và mảng 1 chiều

- Hàm không thể trả về một trị là mảng 1 chiều.
- Đối của hàm là tên của mảng 1 chiều: Khi đó tham số thực truyền cho đối tương ứng cũng là tên của mảng 1 chiều, cùng kiểu và cùng kích thước với đối. Tên của tham số thực và tên của đối có thể trùng nhau hoặc khác nhau.

Đối	Tham số thực
Tên mảng 1 chiều	Tên mảng 1 chiều (cùng kiểu, kích thước với đối)

5.3 Mảng 2 chiều

Trong các mảng nhiều chiều, hình thức đơn giản nhất là mảng 2 chiều.

I) Cách tiếp cận và khai báo:

Mảng 2 chiều là mảng 1 chiều của mảng 1 chiều.

Có thể khai báo như sau:

KDL Ten_Mang[KT1][KT2];

II) Công thức tính số bytes cần thiết trong bộ nhớ lưu trư mảng 2 chiều:

Tong (Bytes) = sizeof(KDL)* KT1* KT2.

III) Nhập, xuất mảng 2 chiều (ma trận):

Thường liên kết với 2 vòng for duyệt theo chỉ số để nhập dữ liệu cho từng phần tử, hoặc xuất dữ liệu của từng phần tử ra màn hình.

IV) Hàm và mảng 2 chiều:

- Hàm không thể trả về giá trị là một mảng 2 chiều.
- Đối của hàm là tên của mảng 2 chiều: Khi đó tham số thực truyền cho đối tương ứng cũng là tên của mảng 2 chiều, cùng kiểu và cùng kích thước với đối. Tên của tham số thực và tên của đối có thể trùng nhau hoặc khác nhau.

Đối	Tham số thực
Tên mảng 2 chiều	Tên mảng 2 chiều (cùng kiểu, kích thước với đối)

5.4 Kiểu mảng

Dùng từ khoá typedef để định nghĩa, bằng cách trước định nghĩa mảng thông thường, ta đặt từ khoá typedef.

Ví dụ:

a) typedef float D*ay10*[10];// Day*10* là kiểu dữ liệu mảng 1 chiều kích thước 10
Day10 a,b; // a,b là 2 biến kiểu mảng 1 chiều có kích thước 10.

b) typedef float M*at10_20* [10][20];
// M*at10_20* là kiểu dữ liệu mảng 2 chiều kích thước 10x20
Mat10_20 c,d; // c,d là 2 biến kiểu mảng 2 chiều có kích thước 10x20

5.5 Khởi đầu cho các mảng

1. Sử dụng các hằng

2. Khi khởi đầu có thể không cần chỉ ra kích thước của mảng. Khi đó máy sẽ dành cho mảng 1 vùng nhớ đủ để thu nhận danh sách giá trị khởi đầu. Trong trường hợp là mảng 2 chiều thì kích thước chiều 2 phải chỉ ra:

3. Đối với mảng 2 chiều có thể khởi đầu theo cách số giá trị của mỗi hàng có thể khác nhau với điều kiện là:

- Kích thước chiều 2 phải được chỉ ra.
- Kích thước chiều 1 hoặc là không được chỉ ra, hoặc có chỉ ra thì giá trị phải lớn hơn kích thước thực sự của mảng

Ví dụ:

```
int a[6][3] = {
                {1,2},
                {3},
                {4,5,6},
                {0,3,1}
            };
// Thực sự đây là mảng a[3][3]
```

5.6 Các kỹ thuật xử lý cơ bản trên mảng

1. Kỹ thuật thử và sai

Cần xác định Kq khi biết $Kq \in \{a_1,..,a_n\}$:

- Giả sử $Kq = a_1$;
- Duyệt các phần tử còn lại để chính xác giá trị Kq.

2. Kỹ thuật duyệt

- Toàn cục: duyệt tất cả các phần tử của tập hợp.
- Cục bộ: Chỉ trên một miền con của tập hợp có thể xét. Miền con này xác định từ các giá trị của hàm, hay là lập bảng.

3. Kỹ thuật kiểm tra tính đúng, sai:

Dạng 1:

- Đúng: nếu $\forall i$, a_i phải thỏa mãn.
- Sai: nếu $\exists i$, a_i không thỏa mãn.

Cách thực hiện như sau:

- Kq = 1; // Đúng
- Duyệt để tìm điều kiện gán Kq = 0; // Sai

Dạng 2:

- Đúng: nếu $\exists i$, a_i thỏa mãn.
- Sai: nếu $\forall i$, a_i không thỏa mãn.

Cách thực hiện như sau:

- Kq = 0; //Sai
- Duyệt để tìm điều kiện gán Kq = 1; // Đúng

Mã bài : MH 09.6

Mục tiêu của bài:

- Trình bày được khái niệm về con trỏ và địa chỉ của con trỏ;
- Khai báo con trỏ, cách truy xuất giá trị cho các nút;
- Viết được các chương trình thực hiện các thao tác truy xuất trên đối tượng kiểu con trỏ, kết hợp được con trỏ với mảng và hàm;
- Nghiêm túc, tỉ mỉ, sáng tạo trong quá trình học và vận dụng vào thực hành.

6.1 Khái niệm con trỏ và địa chỉ

6.1.1 Khái niệm:

Biến con trỏ là biến lưu địa chỉ của những biến khác hay lưu địa chỉ của một ô nhớ nào đó trong máy tính.

Chúng ta đã biết các biến chính là các ô nhớ mà chúng ta có thể truy xuất dưới các tên. Các biến này được lưu trữ tại những chỗ cụ thể trong bộ nhớ. Đối với chương trình của chúng ta, bộ nhớ máy tính chỉ là một dãy gồm các ô nhớ 1 byte, mỗi ô có một địa chỉ xác định.

Một sự mô hình tốt đối với bộ nhớ máy tính chính là một phố trong một thành phố. Trên một phố tất cả các ngôi nhà đều được đánh số tuần tự với một cái tên duy nhất nên nếu chúng ta nói đến số 27 phố Trần Hưng Đạo thì chúng ta có thể tìm được nơi đó mà không lầm lẫn vì chỉ có một ngôi nhà với số như vậy.

Cũng với cách tổ chức tương tự như việc đánh số các ngôi nhà, hệ điều hành tổ chức bộ nhớ thành những số đơn nhất, tuần tự, nên nếu chúng ta nói đến vị trí 1776 trong bộ nhớ chúng ta biết chính xác ô nhớ đó vì chỉ có một vị trí với địa chỉ như vậy.

6.1.1 Toán tử lấy địa chỉ (&)

Vào thời điểm mà chúng ta khai báo một biến thì nó phải được lưu trữ trong một vị trí cụ thể trong bộ nhớ. Nói chung chúng ta không quyết định nơi nào biến đó được đặt - thật may mắn rằng điều đó đã được làm tự động bởi trình biên dịch và hệ điều hành, nhưng một khi hệ điều hành đã gán một địa chỉ cho biến thì chúng ta có thể muốn biết biến đó được lưu trữ ở đâu.

Điều này có thể được thực hiện bằng cách đặt trước tên biến một dấu và (&), có nghĩa là "*địa chỉ của*". Ví dụ:

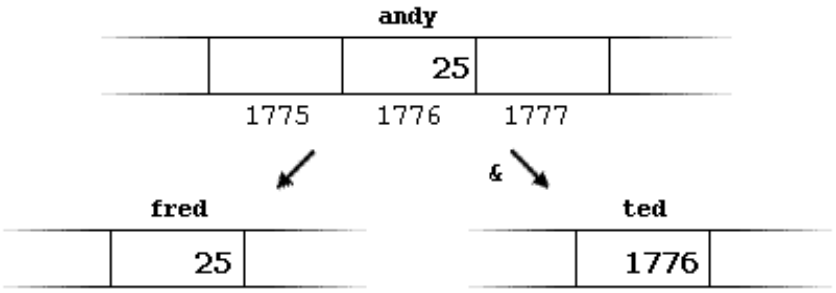
```
ted = &andy;
```

sẽ gán cho biến **ted** địa chỉ của biến **andy**, vì khi đặt trước tên biến **andy** dấu và (&) chúng ta không còn nói đến nội dung của biến đó mà chỉ nói đến địa chỉ của nó trong bộ nhớ.

Giả sử rằng biến **andy** được đặt ở ô nhớ có địa chỉ **1776** và chúng ta viết như sau:

andy = 25;fred = andy;ted = &andy;

kết quả sẽ giống như trong sơ đồ dưới đây:



Chúng ta đã gán cho **fred** nội dung của biến **andy** như chúng ta đã làm rất lần nhiều khác trong những phần trước nhưng với biến **ted** chúng ta đã gán *địa chỉ* mà hệ điều hành lưu giá trị của biến **andy**, chúng ta vừa giả sử nó là **1776**.

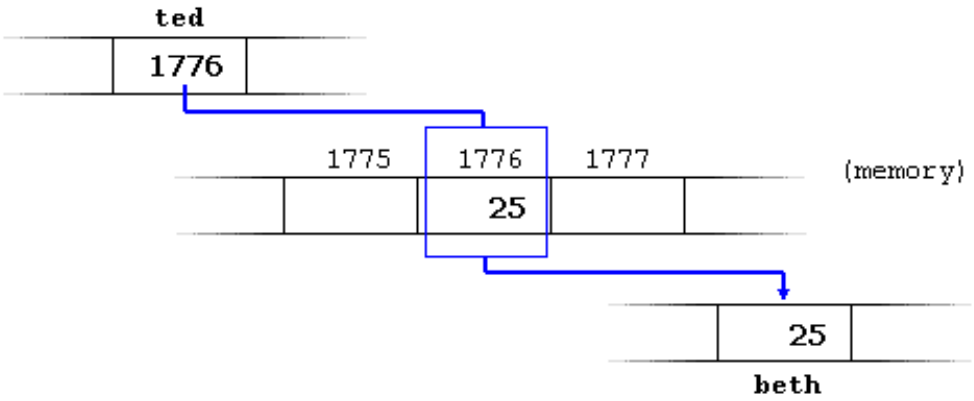
Những biến lưu trữ địa chỉ của một biến khác (như **ted** ở trong ví dụ trước) được gọi là **con trỏ**. Trong C++ con trỏ có rất nhiều ưu điểm và chúng được sử dụng rất thường xuyên, Tiếp theo chúng ta sẽ thấy các biến kiểu này được khai báo như thế nào.

6.1.2 Toán tử tham chiếu (*)

Bằng cách sử dụng con trỏ chúng ta có thể truy xuất trực tiếp đến giá trị được lưu trữ trong biến được trỏ bởi nó bằng cách đặt trước tên biến con trỏ một dấu sao (*) - ở đây có thể được dịch là "**giá trị được trỏ bởi**". Vì vậy, nếu chúng ta viết:

beth = *ted;

(chúng ta có thể đọc nó là: "beth bằng giá trị được trỏ bởi ted" **beth** sẽ mang giá trị **25**, vì **ted** bằng **1776** và giá trị trỏ bởi **1776** là **25**.



Bạn phải phân biệt được rằng **ted** có giá trị **1776**, nhưng ***ted** (với một dấu sao đằng trước) trỏ tới giá trị được lưu trữ trong địa chỉ **1776**, đó là **25**. Hãy chú ý sự khác biệt giữa việc có hay không có dấu sao tham chiếu.

beth = ted; *// beth bằng ted (1776)*

beth = *ted; *// beth bằng giá trị được trỏ bởi (25)*

Toán tử lấy địa chỉ (&)Nó được dùng như là một tiền tố của biến và có thể được dịch là "**địa chỉ của**", vì vậy &variable1 có thể được đọc là "địa chỉ của *variable1"*.

Toán tử tham chiếu (*)Nó chỉ ra rằng cái cần được tính toán là nội dung được trỏ bởi biểu thức được coi như là một địa chỉ. Nó có thể được dịch là "**giá trị được trỏ bởi**"..*mypointer được đọc là *"giá trị được trỏ bởi mypointer"*.

Vào lúc này, với những ví dụ đã viết ở trên

andy = 25;ted = &andy;

bạn có thể dễ dàng nhận ra tất cả các biểu thức sau là đúng:

andy == 25

&andy == 1776

ted == 1776

*ted == 25

6.2 Khai báo biến kiểu con trỏ

Vì con trỏ có khả năng tham chiếu trực tiếp đến giá trị mà chúng trỏ tới nên cần thiết phải chỉ rõ kiểu dữ liệu nào mà một biến con trỏ trỏ tới khai báo nó. Vì vậy, khai báo của một biến con trỏ sẽ có mẫu sau:

*type * pointer_name* **;**

trong đó *type* là kiểu dữ liệu được trỏ tới, không phải là kiểu của bản thân con trỏ. Ví dụ:

int * number;char * character;float * greatnumber;

đó là ba khai báo của con trỏ. Mỗi biến đầu trỏ tới một kiểu dữ liệu khác nhau nhưng cả ba đều là con trỏ và chúng đều chiếm một lượng bộ nhớ như nhau (kích thước của một biến con trỏ tùy thuộc vào hệ điều hành). nhưng dữ liệu mà chúng trỏ tới không chiếm lượng bộ nhớ như nhau, một kiểu **int**, một kiểu **char** và cái còn lại kiểu **float**.

Cần phải nhấn mạnh lại rằng dấu sao (*) mà chúng ta đặt khi khai báo một con trỏ chỉ có nghĩa rằng: đó là một con trỏ và hoàn toàn không liên quan đến toán tử tham chiếu mà chúng ta đã xem xét trước đó. Đó đơn giản chỉ là hai tác vụ khác nhau được biểu diễn bởi cùng một dấu.

Code:

```
// my first pointer
#include <iostream.h>
int main ()
{
int value1 = 5, value2 = 15;
int * mypointer;
mypointer = &value1;
*mypointer = 10;
mypointer = &value2;
*mypointer = 20;
cout << "value1==" << value1 << "/ value2==" << value2;
return 0;
}
```

Kết quả:

value1==10 / value2==20

Chú ý rằng giá trị của **value1** và **value2** được thay đổi một cách gián tiếp. Đầu tiên chúng ta gán cho **mypointer** địa chỉ của **value1**dùng toán tử lấy địa chỉ (&) và sau đó chúng ta gán **10** cho giá trị được trỏ bởi **mypointer**, đó là giá trị được trỏ bởi **value1** vì vậy chúng ta đã sửa biến **value1** một cách gián tiếp

Để bạn có thể thấy rằng một con trỏ có thể mang một vài giá trị trong cùng một chương trình chúng ta sẽ lặp lại quá trình với **value2**và với cùng một con trỏ.

Đây là một ví dụ phức tạp hơn một chút:

Code:

```
// more pointers
#include <iostream.h>
int main ()
{
int value1 = 5, value2 = 15;
int *p1, *p2;
p1 = &value1; // p1 = địa chỉ của value1
p2 = &value2; // p2 = địa chỉ của value2
*p1 = 10; // giá trị trỏ bởi p1 = 10
*p2 = *p1; // giá trị trỏ bởi p2 = giá trị trỏ bởi p1
p1 = p2; // p1 = p2 (phép gán con trỏ)
*p1 = 20; // giá trị trỏ bởi p1 = 20
cout << "value1==" << value1 << "/ value2==" << value2;
return 0;
}
```

Kết quả:

value1==10 / value2==20

Một dòng có thể gây sự chú ý của bạn là:

int *p1, *p2;

dòng này khai báo hai con trỏ bằng cách đặt dấu sao (*) trước mỗi con trỏ. Nguyên nhân là kiểu dữ liệu khai báo cho cả dòng là **int**và vì theo thứ tự từ phải sang trái, dấu sao được tính trước tên kiểu. Chúng ta đã nói đến điều này trong bài 1.3: Các toán tử.

6.3 Con trỏ và mảng một chiều.

Trong thực tế, tên của một mảng tương đương với địa chỉ phần tử đầu tiên của nó, giống như một con trỏ tương đương với địa chỉ của phần tử đầu tiên mà nó trỏ tới, vì vậy thực tế chúng hoàn toàn như nhau. Ví dụ, cho hai khai báo sau:

int numbers [20];int * p;

lệnh sau sẽ hợp lệ:

p = numbers;

Ở đây **p** và **numbers** là tương đương và chúng có cũng thuộc tính, sự khác biệt duy nhất là chúng ta có thể gán một giá trị khác cho con trỏ **p** trong khi **numbers** luôn trỏ đến phần tử đầu tiên trong số 20 phần tử kiểu int mà nó được định nghĩa với. Vì vậy, không giống như **p** - đó là một biến con trỏ bình thường, **numbers** là một con trỏ hằng. Lệnh gán sau đây là không hợp lệ:

numbers = p;

bởi vì **numbers** là một mảng (con trỏ hằng) và không có giá trị nào có thể được gán cho các hằng.

V ì con trỏ cũng có mọi tính chất của một biến nên tất cả các biểu thức có con trỏ trong ví dụ dưới đây là hoàn toàn hợp lệ:

Code:

```
// more pointers
#include <iostream.h>
int main ()
{
int numbers[5];
int * p;
p = numbers; *p = 10;
p++; *p = 20;
p = &numbers[2]; *p = 30;
p = numbers + 3; *p = 40;
p = numbers; *(p+4) = 50;
for (int n=0; n<5; n++)
cout << numbers[n] << ", ";
return 0;
}
```

Kết quả:

10, 20, 30, 40, 50,

Trong bài mảng chúng ta đã dùng dấu ngoặc vuông để chỉ ra phần tử của mảng mà chúng ta muốn trỏ đến. Cặp ngoặc vuông này được coi như là toán tử offset và ý

nghĩa của chúng không đổi khi được dùng với biến con trỏ. Ví dụ, hai biểu thức sau đây:

a[5] = 0; *// a [offset of 5] = 0*

*(a+5) = 0; *// pointed by (a+5) = 0*

là hoàn toàn tương đương và hợp lệ bất kể **a** là mảng hay là một con trỏ.

6.3.2 Khởi tạo con trỏ

Khi khai báo con trỏ có thể chúng ta sẽ muốn chỉ định rõ ràng chúng sẽ trỏ tới biến nào,

int number;int *tommy = &number;

là tương đương với:

int number;int *tommy;tommy = &number;

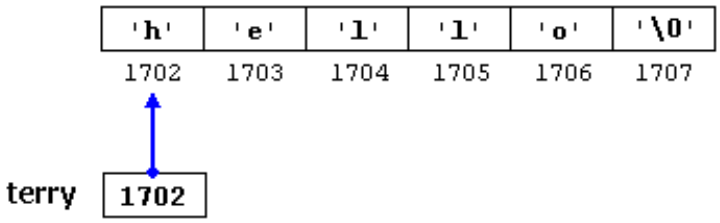
Trong một phép gán con trỏ chúng ta phải luôn luôn gán địa chỉ mà nó trỏ tới chứ không phải là giá trị mà nó trỏ tới. Bạn cần phải nhớ rằng khi khai báo một biến con trỏ, dấu sao (*) được dùng để chỉ ra nó là một con trỏ, và hoàn toàn khác với toán tử tham chiếu. Đó là hai toán tử khác nhau mặc dù chúng được viết với cùng một dấu. Vì vậy, các câu lệnh sau là không hợp lệ:

int number;int *tommy;*tommy = &number;

Như đối với mảng, trình biên dịch cho phép chúng ta khởi tạo giá trị mà con trỏ trỏ tới bằng giá trị hằng vào thời điểm khai báo biến con trỏ:

char * terry = "hello";

trong trường hợp này một khối nhớ tĩnh được dành để chứa "hello" và một con trỏ trỏ tới kí tự đầu tiên của khối nhớ này (đó là kí tự h') được gán cho **terry**. Nếu "hello" được lưu tại địa chỉ 1702, lệnh khai báo trên có thể được hình dung như thế này:



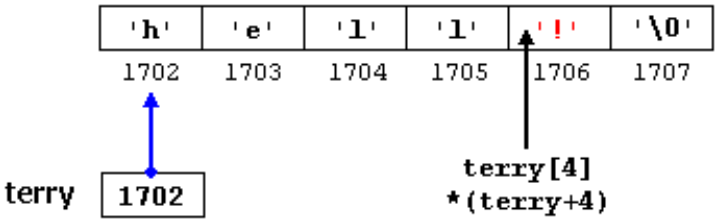
cần phải nhắc lại rằng **terry** mang giá trị **1702** chứ không phải là **'h'** hay **"hello"**.

Biến con trỏ **terry** trỏ tới một xâu kí tự và nó có thể được sử dụng như là đối với một mảng (hãy nhớ rằng một mảng chỉ đơn thuần là một con trỏ hằng). Ví dụ, nếu chúng

ta muốn thay kí tự **'o'** bằng một dấu chấm than, chúng ta có thể thực hiện việc đó bằng hai cách:

terry[4] = '!';*(terry+4) = '!';

hãy nhớ rằng viết **terry[4]** là hoàn toàn giống với viết ***(terry+4)** mặc dù biểu thức thông dụng nhất là cái đầu tiên. Với một trong hai lệnh trên xâu do **terry** trỏ đến sẽ có giá trị như sau:



6.3.2 Các phép tính số học với pointer

Việc thực hiện các phép tính số học với con trỏ hơi khác so với các kiểu dữ liệu số nguyên khác. Trước hết, chỉ phép cộng và trừ là được phép dùng. Nhưng cả cộng và trừ đều cho kết quả phụ thuộc vào *kích thước* của kiểu dữ liệu mà biến con trỏ trỏ tới.

Chúng ta thấy có nhiều kiểu dữ liệu khác nhau tồn tại và chúng có thể chiếm chỗ nhiều hơn hoặc ít hơn các kiểu dữ liệu khác. Ví dụ, trong các kiểu số nguyên, *char* chiếm 1 byte, *short* chiếm 2 byte và *long* chiếm 4 byte.

Giả sử chúng ta có 3 con trỏ sau:

char *mychar;short *myshort;long *mylong;

và chúng lần lượt trỏ tới ô nhớ **1000**, **2000** and **3000**.

Nếu chúng ta viết

mychar++;myshort++;mylong++;

Điều này đúng với cả hai phép toán cộng và trừ đối với con trỏ. Chúng ta cũng hoàn toàn thu được kết quả như trên nếu viết:

mychar = mychar + 1;myshort = myshort + 1;mylong = mylong + 1;

Cần phải cảnh báo rằng cả hai toán tử tăng (++) và giảm (--) đều có quyền ưu tiên lớn hơn toán tử tham chiếu (*), vì vậy biểu thức sau đây có thể dẫn tới kết quả sai:

*p++;*p++ = *q++;

Lệnh đầu tiên tương đương với *(p++) điều mà nó thực hiện là tăng **p** (địa chỉ ô nhớ mà nó trỏ tới chứ không phải là giá trị trỏ tới).

Lệnh thứ hai, cả hai toán tử tăng (++) đều được thực hiện sau khi giá trị của *q được gán cho *p và sau đó cả q và p đều tăng lên 1. Lệnh này tương đương với:

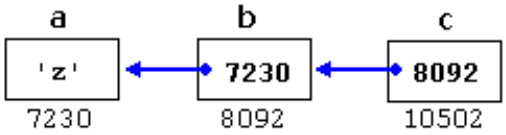
*p = *q;p++;q++;

6.3.3 Con trỏ trỏ tới con trỏ

C++ cho phép sử dụng các con trỏ trỏ tới các con trỏ khác giống như là trỏ tới dữ liệu. Để làm việc đó chúng ta chỉ cần thêm một dấu sao (*) cho mỗi mức tham chiếu.

char a;char * b;char ** c;a = 'z';b = &a;c = &b;

giả sử rằng a,b,c được lưu ở các ô nhớ **7230**, **8092** and **10502**, ta có thể mô tả đoạn mã trên như sau:



Điểm mới trong ví dụ này là biến **c**, chúng ta có thể nói về nó theo 3 cách khác nhau, mỗi cách sẽ tương ứng với một giá trị khác nhau:

c là một biến có kiểu (char **) mang giá trị 8092***c** là một biến có kiểu (char*) mang giá trị 7230****c** là một biến có kiểu (char) mang giá trị 'z'

6.3.4 Con trỏ không kiểu

Con trỏ không kiểu là một loại con trỏ đặc biệt. Nó có thể trỏ tới bất kì loại dữ liệu nào, từ giá trị nguyên hoặc thực cho tới một xâu kí tự. Hạn chế duy nhất của nó là dữ liệu được trỏ tới không thể được tham chiếu tới một cách trực tiếp (chúng ta không thể dùng toán tử tham chiếu * với chúng) vì độ dài của nó là không xác định và vì vậy chúng ta phải dùng đến toán tử chuyển kiểu dữ liệu hay phép gán để chuyển con trỏ không kiểu thành một con trỏ trỏ tới một loại dữ liệu cụ thể.

Một trong những tiện ích của nó là cho phép truyền tham số cho hàm mà không cần chỉ rõ kiểu

```
Code:
// integer increaser
#include <iostream.h>
void increase (void* data, int type)
{
switch (type)
```

```
{
case sizeof(char) : (*((char*)data))++; break;
case sizeof(short): (*((short*)data))++; break;
case sizeof(long) : (*((long*)data))++; break;
}
}
int main ()
{
char a = 5;
short b = 9;
long c = 12;
increase (&a,sizeof(a));
increase (&b,sizeof(b));
increase (&c,sizeof(c));
cout << (int) a << ", " << b << ", " << c;
return 0;
}
```

Kết quả:

6, 10, 13

sizeof là một toán tử của ngôn ngữ C++, nó trả về một giá trị hằng là kích thước tính bằng byte của tham số truyền cho nó, ví dụ **sizeof(char)** bằng **1** vì kích thước của **char** là 1 byte.

6.4 Con trỏ hàm

C++ cho phép thao tác với các con trỏ hàm. Tiện ích tuyệt vời này cho phép truyền một hàm như là một tham số đến một hàm khác. Để có thể khai báo một con trỏ trỏ tới một hàm chúng ta phải khai báo nó như là khai báo mẫu của một hàm nhưng phải bao trong một cặp ngoặc đơn () tên của hàm và chèn dấu sao (*) đằng trước.

Code:

```
// pointer to functions
#include <iostream.h>
```

```
int addition (int a, int b)
{ return (a+b); }
int subtraction (int a, int b)
{ return (a-b); }
int (*minus)(int,int) = subtraction;
int operation (int x, int y, int (*functocall)(int,int))
{
int g;
g = (*functocall)(x,y);
return (g);
}
int main ()
{
int m,n;
m = operation (7, 5, &addition);
n = operation (20, m, minus);
cout <<n;
return 0;
}
```

Kết quả:

8

Trong ví dụ này, **minus** là một con trỏ toàn cục trỏ tới một hàm có hai tham số kiểu **int**, con trỏ này được gám để trỏ tới hàm **subtraction**, tất cả đều trên một dòng:

int (* minus)(int,int) = subtraction;

CHƯƠNG 7: CHUỖI KÝ TỰ

Mã bài: MH09.7

Mục tiêu của bài:

- Trình bày được khái niệm về kiểu dữ liệu chuỗi ký tự;
- Khai báo biến chuỗi, cách thao tác trên chuỗi;
- Viết được các chương trình thực hiện một số thao tác xử lý các chuỗi ký tự ;
- Nghiêm túc, tỉ mỉ, sáng tạo trong quá trình học và vận dụng vào thực hành.

7.1.Khái niệm

Chuỗi (xâu ký tự) là mảng một chiều các ký tự được kết thúc bởi ký tự NULL (\0).

Số lượng các ký tự khác NULL trong xâu gọi là chiều dài của xâu.

7.2 Khai báo

```
char a[KT];
```

- a là tên, chỉ tên của xâu ký tự.
- KT là số nguyên dương, chỉ kích thước của xâu ký tự.
- a[i] là ký tự thứ i của a. Chỉ số đầu tiên luôn là 0.

Trong khi khai báo, ta phải khai báo xâu ký tự có chiều dài lớn hơn mảng được sử dụng 1 ký tự để đủ chỗ chứa ký tự NULL.

7.3 Kiểu xâu ký tự

Cách tạo là đặt từ khoá typedef trước khai báo xâu thông thường.

7.4 Các thao tác trên chuỗi ký tự

1. Xuất:

```
cout<< a;
```

Khi đó toán tử << của iostream sẽ in từng ký tự trong a cho đến khi gặp mã của NULL (\0).

2. Nhập:

```
cin>> a ;
```

Với a là một xâu ký tự không chứa ký tự tách (khoảng trắng, tab …), vì iostream xem các ký tự đó là ký tự tách chứ không phải dữ liệu.

Để nhập một xâu ký tự có chứa ký tự tách thì có thể dùng các cách sau:

a) Dùng hàm gets trong tệp tiêu đề <stdio.h> hoặc trong <string.h>:

gets(a);

b) Dùng hàm thành phần của iostream:

cin.getline(a, so_ky_tu);

Hàm này đọc vào tối đa so_ky_tu – 1 ký tự, kể cả ký tự trắng.

Các xử lý thường gặp trên xâu ký tự như xác định chiều dài xâu, chép xâu s vào xâu t, nối 2 xâu,....

7.4.1 Khởi đầu cho xâu ký tự

Cũng theo quy tắc chung của mảng (1 chiều) đã giới thiệu ở trên.Bộ khởi đầu của mảng 1 chiều ký tự có thể là:

- Danh sách các hằng ký tự, cuối cùng là ký tự '\0'.
- Hoặc là một hằng xâu ký tự.

7.4.2 Hàm và xâu ký tự

Hàm không thể trả về một giá trị là xâu ký tự.

Đối của hàm có thể là tên xâu ký tự, khi đó tham số thực phải là tên xâu ký tự

Đối	Tham số thực
Xâu ký tự	Tên xâu ký tự

7.4.3 Mảng các xâu ký tự

- Cách tiếp cận như là mảng 2 chiều các ký tự.
- Khai báo:

  char a[KT1][KT2];

- Nhập hoặc xuất mảng các xâu ký tự là nhập hoặc xuất từng xâu ký tự của mảng.
- Khởi đầu:

  Khởi đầu từng xâu ký tự bằng các hằng xâu ký tự.

7.4.4 Một số hàm xử lý chuỗi

Chuyển đổi xâu kí tự sang các kiểu khác.

Vì một xâu kí tự có thể biểu diễn nhiều kiểu dữ liệu khác như dạng số nên việc chuyển đổi nội dung như vậy sang dạng số là rất hữu ích. Ví dụ, một xâu có thể mang giá trị "1977"nhưng đó là một chuỗi gồm 5 kí tự (kể cả kí tự null) và không dễ gì

chuyển thành một số nguyên. Vì vậy thư viện cstdlib (stdlib.h) đã cung cấp 3 macro/hàm hữu ích sau:

atoi: chuyển xâu thành kiểu int.

atol: chuyển xâu thành kiểu long.

atof: chuyển xâu thành kiểu float.

Tất cả các hàm này nhận một tham số và trả về giá trị số (int, long hoặc float). Các hàm này khi kết hợp với phương thức getline của cin là một cách đáng tin cậy hơn phương thức cin>> cổ điển khi yêu cầu người sử dụng nhập vào một số.

Các hàm để thao tác trên chuỗi

Thư viện cstring (string.h) không chỉ có hàm strcpy mà còn có nhiều hàm khác để thao tác trên chuỗi. Dưới đây là giới thiệu lướt qua của các hàm thông dụng nhất:

strcat: char* strcat (char* dest, const char* src);

Gắn thêm chuỗi src vào phía cuối của dest. Trả về dest.

strcmp: int strcmp (const char* string1, const char* string2);

So sánh hai xâu string1 và string2. Trả về 0 nếu hai xâu là bằng nhau.

strcpy: char* strcpy (char* dest, const char* src);

Copy nội dung của src cho dest. Trả về dest.

strlen: size_t strlen (const char* string);

Trả về độ dài của string.

Chú ý: char* hoàn toàn tương đương với char[]

TÀI LIỆU THAM KHẢO

[1] Lê Mạnh Thạnh (2000), *Giáo trình môn lập trình C*, NXB Giáo dục.
[2] Nguyễn Linh Giang, Nguyễn Xuân Thực, Lê Văn Thái (2005), *Giáo trình kỹ thuật lập trình C*, NXB Giáo dục.
[3] Ngô Trung Việt (1995), *Giáo trình ngôn ngữ lập trình C và C++* , NXB Giao thông vận tải.
[4] B. Kernighan and D. Ritchie (1990), *The C programming language*, Prentice Hall.