GIÁO TRÌNH

LẬP TRÌNH CĂN BẢN

(BASIC PROGRAMMING)

Vĩnh Long - 2024

LỜI NÓI ĐẦU

-----ᘓ✧ᘒ-----

Để đáp ứng nhu cầu học tập và nghiên cứu của các bạn sinh viên, đặc biệt là sinh viên chuyên ngành Công nghệ thông tin, Khoa Công nghệ thông tin - Trường Đại học Sư phạm Kỹ thuật Vĩnh Long đã tiến hành biên soạn các giáo trình, bài giảng chính trong chương trình học. Giáo trình **Lập trình căn bản** này được biên soạn chủ yếu dựa trên quyển " ***C++ Program Design – An Introduction to Programming and Object-Oriented Design*** " của James P. Cohoon and Jack W.Davidson. Giáo trình này cũng được biên soạn dựa trên kinh nghiệm giảng dạy nhiều năm môn **Lập trình căn bản** của các giáo viên trong khoa chúng tôi. Ngoài ra chúng tôi cũng đã tham khảo rất nhiều tài liệu của các trường đại học trong và ngoài nước.

Tài liệu này được biên soạn dựa theo đề cương chi tiết môn học **Lập trình căn bản** của Khoa Công nghệ thông tin Trường Đại học Sư phạm Kỹ thuật Vĩnh Long dùng cho sinh viên chuyên ngành Công nghệ thông tin bao gồm 7 chương:

Chương 1: Tổng quan về ngôn ngữ C++

Chương 2: Các cấu trúc điều khiển

Chương 3: Dữ liệu kiểu mảng

Chương 4: Dữ liệu kiểu chuỗi

Chương 5: Con trỏ và hàm

Chương 6: Dữ liệu kiểu cấu trúc

Chương 7: Dữ liệu kiểu tập tin

Mục tiêu của nó nhằm giúp các bạn sinh viên chuyên ngành có một tài liệu cô đọng dùng làm tài liệu học tập và nghiên cứu nhưng chúng tôi cũng không loại trừ toàn bộ các đối tượng khác tham khảo. Chúng tôi nghĩ rằng các bạn sinh viên thuộc các chuyên ngành khác và những người quan tâm đến lập trình sẽ tìm được những điều bổ ích trong quyển giáo trình này.

Mặc dù chúng tôi đã rất cố gắng trong quá trình biên soạn nhưng chắc chắn giáo trình này sẽ còn nhiều thiếu sót và hạn chế. Rất mong nhận được sự đóng góp ý kiến quý báu của các sinh viên và bạn đọc để giáo trình ngày càng hoàn thiện hơn.

Chân thành cảm ơn!

Tác giả

CHƯƠNG 1: TỔNG QUAN VỀ NGÔN NGỮ C++

-----ଓ✧ଌ-----

1.1 KHÁI NIỆM VỀ LẬP TRÌNH

1.1.1 Giới thiệu chung

Máy tính là một công cụ để giải quyết hàng loạt các bài toán lớn. Một lời giải cho một bài toán nào đó được gọi là một giải thuật hay thuật toán (algorithm); nó mô tả một chuỗi các bước cần thực hiện để giải quyết bài toán.

Ví dụ: Bài toán về sắp xếp một danh sách các số theo thứ tự tăng dần.

Giải thuật của bài toán trên là: Giả sử danh sách đã cho là *list1*; ta tạo ra một danh sách rỗng *list2*, để lưu danh sách đã sắp xếp. Lặp đi lặp lại công việc, tìm số nhỏ nhất trong *list1*, xóa nó khỏi *list1* và thêm vào phần tử kế tiếp trong danh sách *list2*, cho đến khi *list1* là rỗng.

Giải thuật được diễn giải bằng các thuật ngữ trừu tượng mang tính chất dễ hiểu. Ngôn ngữ thật sự được hiểu bởi máy tính là ngôn ngữ máy. Chương trình được diễn đạt bằng ngôn ngữ máy được gọi là có thể thực thi. Một chương trình được viết bằng bất kỳ một ngôn ngữ nào trước hết cần được dịch sang ngôn ngữ máy để máy tính có thể hiểu và thực thi nó.

Ngôn ngữ máy cực kỳ khó hiểu đối với lập trình viên vì thế họ không thể sử dụng trực tiếp ngôn ngữ máy để viết chương trình. Một sự trừu tượng khác là ngôn ngữ assembly. Nó cung cấp những tên dễ nhớ cho các lệnh và một ký hiệu dễ hiểu hơn cho dữ liệu. Bộ dịch được gọi là assembler chuyển ngôn ngữ assembly sang ngôn ngữ máy.

Ngay cả những ngôn ngữ assembly cũng khó sử dụng. Những ngôn ngữ cấp cao như: C, C++, Pascal, … cung cấp các ký hiệu thuận tiện hơn nhiều cho việc lập trình cũng như thi hành các giải thuật.

Các ngôn ngữ cấp cao này giúp cho các lập trình viên không phải nghĩ nhiều về các ngôn ngữ cấp thấp vì thế giúp họ chỉ tập trung vào giải thuật. Trình biên dịch (compiler) sẽ đảm nhiệm việc dịch chương trình viết bằng ngôn ngữ cấp cao sang ngôn ngữ assembly. Mã assembly được tạo ra bởi trình biên dịch sau đó sẽ được tập hợp lại để cho ra một chương trình có thể thực thi.

1.1.2 Định nghĩa

Lập trình là kỹ thuật tổ chức dữ liệu và xây dựng quy trình xử lý cho máy tính làm việc thông qua ngôn ngữ lập trình chẳng hạn như: C, C++, Pascal, …

- Tổ chức dữ liệu: sắp xếp các thông tin nhằm phục vụ cho yêu cầu nào đó.

- Quy trình xử lý: bao gồm các chỉ thị để thực hiện các công việc như: tạo thông tin ban đầu, tính toán, sao chép, di chuyển, tìm kiếm, in kết quả, …

1.1.3 Giải thuật (Algorithm)

Muốn viết được một chương trình nào đó cho máy tính thực hiện điều trước tiên là ta phải tìm được giải thuật (thuật toán) cho công việc cần viết chương trình đó.

Giải thuật (thuật toán) là một tập hợp có thứ tự các bước tiến hành công việc nhằm đạt được kết quả mong muốn.

Ví dụ: Để viết được chương trình giải phương trình bậc 2 ($ax^2 + bx + c = 0$) ta cần phải tìm được các bước để thực hiện công việc này, đó chính là giải thuật:

Bước 1: Nhập 3 hệ số a, b và c

Bước 2: Tính $\Delta = b^2 - 4ac$

Bước 3: Xét dấu Δ

Nếu $\Delta < 0$ thì phương trình vô nghiệm

Ngược lại, nếu $\Delta = 0$ thì phương trình có nghiệm kép $x_1 = x_2 = -\frac{b}{2a}$

ngược lại ($\Delta > 0$) thì phương trình có 2 nghiệm phân biệt:

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \; ; \; x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

1.1.4 Đặc tính của giải thuật

- Phải kết thúc sau một số bước hữu hạn.
- Các bước trong giải thuật phải được máy chấp nhận và có thể thực hiện được.
- Xét hết tất cả các trường hợp có thể xảy ra.
- Áp dụng được cho tất cả các bài toán cùng loại (cùng dạng).

1.1.5 Các lưu ý

- Một bài toán có thể có nhiều giải thuật khác nhau.
- Một giải thuật được gọi là tốt nếu nó có các tính chất sau:
  + Đơn giản, dễ hiểu.
  + Tiết kiệm vùng nhớ.
  + Thời gian thực hiện nhanh.

1.1.6 Các công cụ thể hiện giải thuật

Để thể hiện giải thuật ta có thể dùng nhiều công cụ khác nhau, trong đó có 2 công cụ được dùng nhiều nhất là ngôn ngữ giả và lưu đồ.

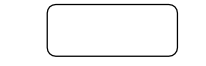
1.1.6.1 Ngôn ngữ giả

Ta có thể sử dụng các từ ngữ, ký hiệu sao cho ngắn gọn, dễ hiểu và có đánh số thứ tự các bước thực hiện.

Ví dụ: Giải thuật giải phương trình bậc 2 ở trên được thể hiện bằng ngôn ngữ giả.

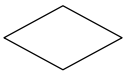
1.1.6.2 Lưu đồ (Flow chart)

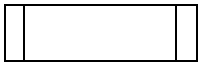
Ta sử dụng các khối theo qui định để thể hiện giải thuật, các khối được dùng là:

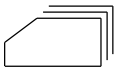

: Bắt đầu / Kết thúc

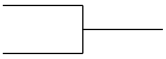

: Nhập / Xuất


: Thi hành (Thực hiện)


hoặc 
: Lựa chọn (Điều kiện)


: Chương trình con

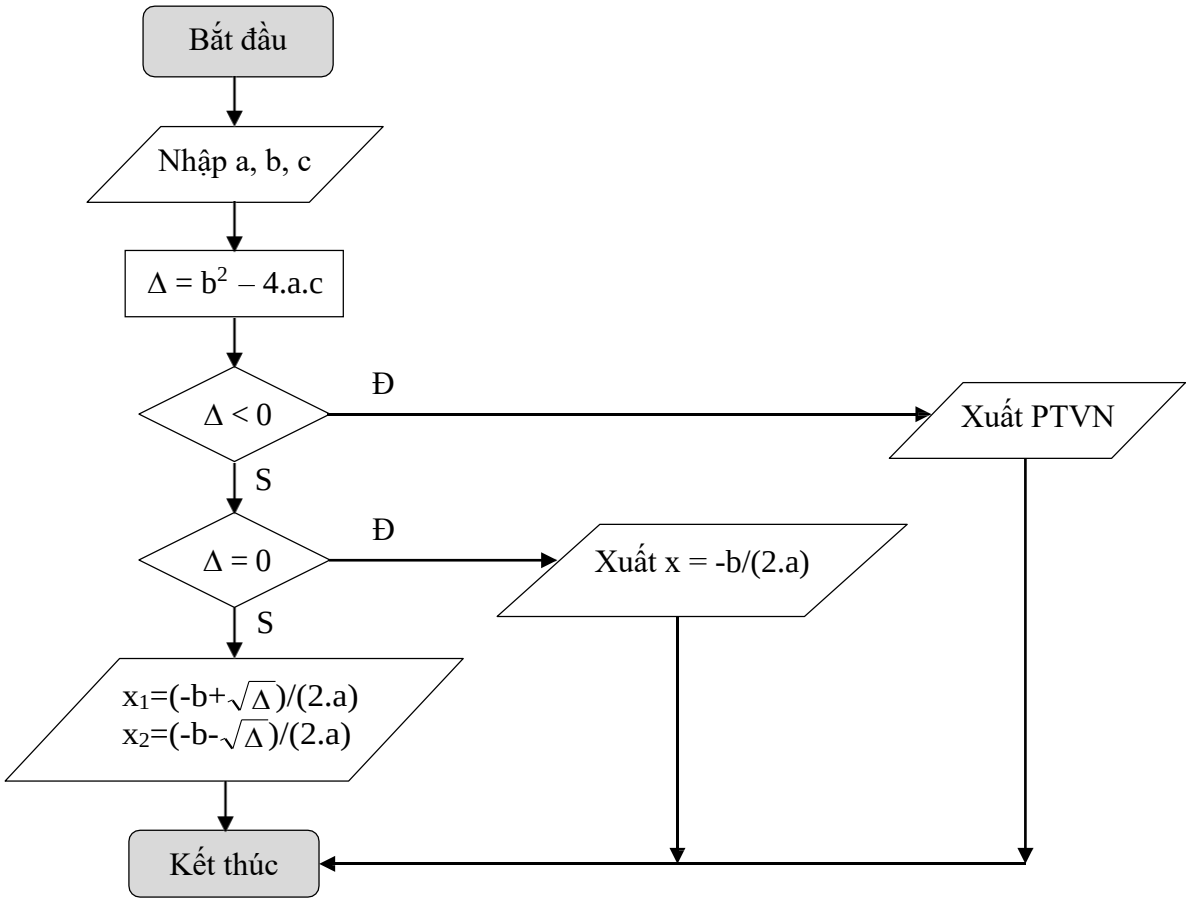

: Tập hợp các tập tin dữ liệu


: Các ghi chú, giải thích


: Nối

: Đường đi

Ví dụ: Giải thuật giải phương trình bậc 2 ở trên được thể hiện bằng lưu đồ như sau:



1.2 CÁC THÀNH PHẦN CƠ BẢN TRONG NGÔN NGỮ C++

Một ngôn ngữ lập trình (NNLT) cấp cao cho phép người lập trình (NLT) biểu diễn ý tưởng của mình để giải quyết một vấn đề, một bài toán bằng cách diễn đạt gần với ngôn ngữ thông thường thay vì phải diễn đạt theo ngôn ngữ máy (dãy các ký hiệu 0, 1). Hiển nhiên, các ý tưởng NLT muốn trình bày phải được viết theo một cấu trúc chặt chẽ thường được gọi là giải thuật hay thuật toán và theo đúng các qui tắc của ngôn ngữ gọi là cú pháp hoặc văn phạm.

1.2.1 Bảng ký tự của C++

Dưới đây là bảng ký tự được dùng để tạo nên những câu lệnh của ngôn ngữ C++:

- Các chữ cái La tinh (viết thường và viết hoa): a..z và A..Z. Cùng một chữ cái nhưng viết thường phân biệt với viết hoa. Ví dụ chữ cái 'a' là khác với 'A'.

- Dấu gạch dưới: _

- Các chữ số thập phân: 0..9.

- Các ký hiệu toán học: +, -, *, /, % , &, ||, !, >, <, = ...

- Các ký hiệu đặc biệt khác: , ;: [ ], {}, #, dấu cách, ...

1.2.2 Từ khóa

Một từ khoá là một từ được qui định trước trong NNLT với một ý nghĩa cố định, thường dùng để chỉ các loại dữ liệu hoặc kết hợp thành câu lệnh. Người lập trình có thể tạo ra những từ mới để chỉ các đối tượng của mình nhưng ***không được phép trùng với từ khóa***. Dưới đây là một vài từ khoá thường gặp, ý nghĩa của các từ này sẽ được trình bày dần trong các đề mục liên quan: asm, break, case, char, continue, default, do, double, else, extern, float, for, goto, if, int, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, while ... Một đặc trưng cần nhớ của C++ là ***các từ khóa luôn luôn được viết bằng chữ thường***.

1.2.3 Tên gọi

Để phân biệt các đối tượng với nhau chúng cần có một tên gọi. Hầu hết một đối tượng được viết ra trong chương trình thuộc 2 dạng, một dạng đã có sẵn trong ngôn ngữ (chẳng hạn như các từ khoá, tên các hàm chuẩn, ...), dạng còn lại là do người lập trình tạo ra dùng để đặt tên cho hằng, biến, kiểu, hàm, ...

Các tên gọi do người lập trình tự đặt phải tuân theo một số qui tắc sau:

- Là dãy ký tự liên tiếp (không chứa dấu cách và các ký hiệu) và phải bắt đầu bằng chữ cái hoặc dấu gạch dưới.

- Phân biệt ký tự in hoa và thường.

- Không được trùng với từ khóa.

- Số lượng ký tự dùng để phân biệt tên gọi là tùy ý.

Ví dụ:

Các tên gọi sau đây là đúng (được phép): i, i1, j, tinhoc, tin_hoc, luu_luong

Các tên gọi sau đây là sai (không được phép): 1i, tin hoc, luu-luong

Các tên gọi sau đây là khác nhau: ha_noi, Ha_noi, HA_Noi, HA_NOI

1.2.4 Chú thích trong chương trình

Một chương trình thường được viết một cách ngắn gọn, do vậy thông thường bên cạnh các câu lệnh chính thức của chương trình, người lập trình còn được phép viết vào chương trình các câu chú thích (ghi chú) nhằm giải thích rõ nghĩa hơn cho câu lệnh,

đoạn chương trình hay chương trình. Một chú thích có thể ghi chú về nhiệm vụ, mục đích, cách thức của thành phần đang được chú thích như biến, hằng, hàm hoặc công dụng của một đoạn lệnh, ... Các chú thích sẽ làm cho chương trình dễ hiểu hơn vì vậy dễ bảo trì, sửa chữa về sau hơn.

Có 2 cách báo cho chương trình biết một đoạn chú thích:

- Nếu chú thích là một đoạn ký tự bất kỳ liên tiếp nhau (trong 1 hàng hoặc trên nhiều hàng) ta đặt đoạn chú thích đó giữa cặp dấu đóng mở chú thích /* (mở) và */ (đóng).

- Nếu chú thích bắt đầu từ một vị trí nào đó cho đến hết hàng, thì ta đặt dấu // ở vị trí đó. Như vậy // sử dụng cho các chú thích chỉ trên 1 hàng.

Như đã nói ở trên, vai trò của đoạn chú thích là làm cho chương trình dễ hiểu đối với người đọc, vì vậy đối với máy các đoạn chú thích sẽ được bỏ qua. Lợi dụng đặc điểm này của chú thích đôi khi để tạm thời bỏ qua một đoạn lệnh nào đó trong chương trình (nhưng không xoá hẳn để khỏi phải gõ lại khi cần dùng đến) ta có thể đặt các dấu chú thích bao quanh đoạn lệnh này (ví dụ khi chạy thử chương trình, gỡ lỗi, ...), khi cần sử dụng lại ta có thể bỏ các dấu chú thích.

✧ ***Chú ý***: Cặp dấu chú thích /* ... */ không được phép viết lồng nhau.

1.2.5 Cấu trúc một chương trình trong C++

Một chương trình C++ có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trong một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Thông thường một chương trình gồm có các nội dung sau:

- *Phần khai báo các file nguyên mẫu:* Khai báo tên các file chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà người lập trình sẽ dùng trong chương trình.

- *Phần khai báo các kiểu dữ liệu, các biến, hằng ...*: Do người lập trình định nghĩa và được dùng chung trong toàn bộ chương trình.

- *Danh sách các hàm của chương trình*: Do người lập trình viết và bao gồm cả hàm main().

Ví dụ 1: Viết chương trình nhập vào số tuổi của bạn (t) và in ra màn hình hàng chuỗi: "Bây giờ bạn đã t tuổi."

```
#include <iostream>
using namespace std;
```

```
main()
{
      int t;
       cout << "Nhap vao so tuoi cua ban : ";
       cin >> t;
       cout << "Bay gio ban da "<<t<<" tuoi. "<<endl;
}
```

Hàng đầu tiên của chương trình là khai báo tiền xử lý là file nguyên mẫu `iostream`. Đây là khai báo bắt buộc vì trong chương trình có sử dụng các phương thức chuẩn "cout <<" (in ra màn hình) và "cin>>" (nhập dữ liệu từ bàn phím), các phương thức này được khai báo và định nghĩa sẵn trong `iostream`.

Hàng thứ haidùng đểnhómcác thực thể cóliên quan lại với nhautheo tên sau từ khóa namespace.

Không riêng hàm main(), mọi hàm khác đều phải bắt đầu tập hợp các câu lệnh của mình bởi dấu { và kết thúc bởi dấu }. Tập các lệnh bất kỳ bên trong cặp dấu này được gọi là khối lệnh. Khối lệnh là một cú pháp cần thiết trong các câu lệnh có cấu trúc như ta sẽ thấy trong các chương tiếp theo.

Ta cũng có thể sử dụng cách thức nhập và xuất của ngôn ngữ C bằng cách dùng phương thức scanf và printf. Ví dụ trên được viết lại như sau:

```
#include <stdio.h>
#include <conio.h>
main()
{
      int t;
       printf("Nhap vao so tuoi cua ban : ");
       scanf("%d",&t);
       printf("Bay gio ban da %d tuoi.\n",t);
       getch();
}
```

1.3 CÁC BƯỚC ĐỂ TẠO VÀ THỰC THI MỘT CHƯƠNG TRÌNH

1.3.1 Qui trình viết và thực thi chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

a. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output). Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó. Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x_1 và x_2 của phương trình. Kiểu của a, b, c, x_1, x_2 là các số thực.

b. Xác định giải thuật.

c. Cụ thể hóa các khai báo kiểu và thuật toán thành dãy các lệnh. Tức là viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.

d. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.

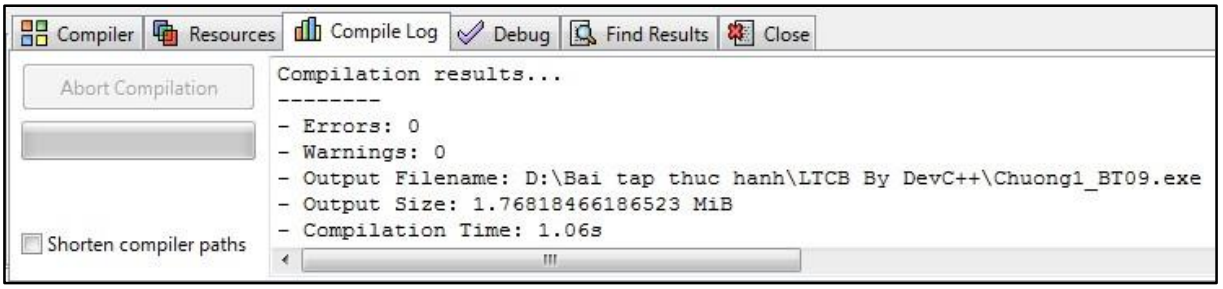
e. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NLT.

1.3.2 Soạn thảo tập tin chương trình nguồn

Soạn thảo chương trình nguồn là một công việc gõ nội dung của chương trình (đã viết ra giấy) vào trong máy. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp C++ (Dev-C++, Borland C++, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của máy. Văn bản chương trình cần được trình bày rõ ràng, dễ hiểu. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

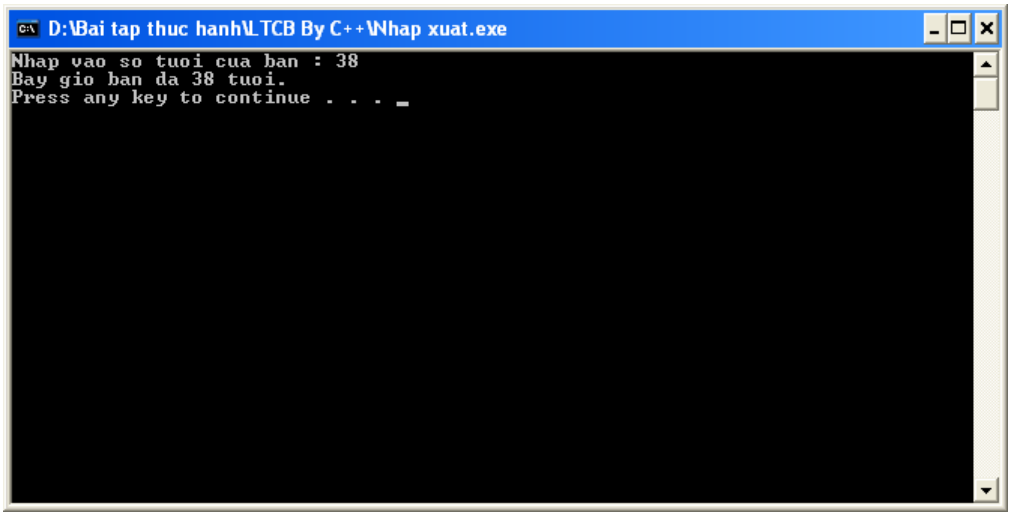
1.3.3 Dịch chương trình

Trong Dev-C++, sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (vào menu Execute rồi chọn Compile hoặc ấn tổ hợp phím Ctrl-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NLT có thể chọn tiếp các lỗi khác để sửa tiếp ở cửa sổ Compiler. Quá trình sửa lỗi − dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp. Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.Tệp này có thể lưu tạm thời trong bộ nhớ phục vụ cho quá trình chạy chương trình hoặc lưu lại trên đĩa tùy theo tùy chọn khi dịch của NLT. Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc xuất hiện hộp thoại thông báo chương trình đã được dịch thành công.



1.3.4 Thực thi chương trình

Sau khi đã sửa hết lỗi trong chương trình ta có thể cho chương trình thực thi (vào menu Execute rồi chọn Run hoặc ấn tổ hợp phím Ctrl-F10), nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NLT kiểm tra.



Nếu kết quả chưa được như mong muốn, quay lại chương trình nguồn để chỉnh sửa và chạy lại chương trình để kiểm tra kết quả. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình đang chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả.

1.4 NHẬP XUẤT TRONG C++

Trong phần này chúng ta làm quen một số lệnh đơn giản cho phép NLT nhập dữ liệu vào từ bàn phím hoặc xuất kết quả ra màn hình.

1.4.1 Nhập dữ liệu từ bàn phím

Để nhập dữ liệu vào cho các biến có tên bien_1, bien_2, bien_3 chúng ta sử dụng câu lệnh:

cin >> bien _1;

cin >> bien _2;

cin >> bien _3;

hoặc:

cin >> bien _1 >> bien _2 >> bien _3;

bien_1, bien_2, bien_3 là các biến được sử dụng để lưu trữ các giá trị NLT nhập vào từ bàn phím. Khái niệm biến sẽ được mô tả cụ thể hơn trong phần sau, ở đây bien_1, bien_2, bien_3 được hiểu là các tên gọi để chỉ 3 giá trị khác nhau. Hiển nhiên có thể nhập dữ liệu nhiều hơn 3 biến bằng cách tiếp tục viết tên biến vào bên phải sau dấu >> của câu lệnh.

Khi chạy chương trình nếu gặp các câu lệnh trên chương trình sẽ "tạm dừng" để chờ NLT nhập dữ liệu vào cho các biến. Sau khi NLT nhập xong dữ liệu, chương trình sẽ tiếp tục chạy từ câu lệnh tiếp theo sau của các câu lệnh trên.

Cách thức nhập dữ liệu của NLT phụ thuộc vào loại giá trị của biến cần nhập mà ta gọi là kiểu, ví dụ nhập một số có cách thức khác với nhập một chuỗi ký tự.

Giả sử cần nhập độ dài hai cạnh của một hình chữ nhật, trong đó cạnh dài được qui ước bằng tên biến cd và chiều rộng được qui ước bởi tên biến cr. Câu lệnh nhập sẽ như sau:

```
cin >> cd >> cr ;
```

Khi máy dừng chờ nhập dữ liệu NLT sẽ gõ giá trị cụ thể của các chiều dài, rộng theo đúng thứ tự trong câu lệnh. Các giá trị này cần cách nhau bởi ít nhất một dấu trắng (ta qui ước gọi dấu trắng là một trong 3 loại dấu được nhập bởi các phím sau: phím spacebar (dấu cách), phím tab (dấu tab) hoặc phím Enter (dấu xuống hàng)). Các giá trị NLT nhập vào cũng được hiển thị trên màn hình để NLT dễ theo dõi.

Ví dụ: Nếu NLT nhập vào 23 11 ↵thì chương trình sẽ gán giá trị 23 cho biến cd và 11 cho biến cr.

✧ ***Chú ý***: Giả sử NLT nhập 23 và 11 không có dấu cách thì chương trình sẽ xem 23 và 11 là một giá trị và gán cho cd. Máysẽ tạm dừng chờ NLT nhập tiếp giá trị cho biến cr.

1.4.2 Xuất dữ liệu ra màn hình

Để xuất/in giá trị của các biểu thức ra màn hình ta dùng câu lệnh sau:

```
cout << bt_1;
cout << bt_2;
cout << bt_3;
```

hoặc:

```
cout << bt_1 << bt_2 << bt_3;
```

Cũng giống câu lệnh nhập ở đây chúng ta cũng có thể mở rộng lệnh xuất/in với nhiều hơn 3 biểu thức. Câu lệnh trên cho phép in giá trị của các biểu thức bt_1, bt_2, bt_3. Các giá trị này có thể là tên của biến hoặc các kết hợp tính toán trên biến.

Ví dụ: Để in câu "Chiều dài là " và số 23 và tiếp theo là chữ " mét", ta có thể sử dụng 3 lệnh sau đây:

```
cout << "Chiều dài là " ;
cout << 23 ;
cout << " mét";
```

hoặc có thể chỉ bằng 1 lệnh:

```
cout << "Chiều dài là 23 mét" ;
```

Trường hợp chưa biết giá trị cụ thể của chiều dài, chỉ biết hiện tại giá trị này đã được lưu trong biến cd (ví dụ đã được nhập vào là 23 từ bàn phím bởi câu lệnh cin >> cd trước đó) và ta cần biết giá trị này là bao nhiêu thì có thể sử dụng câu lệnh in ra màn hình.

```
cout << "Chiều dài là " << cd << " mét" ;
```

Khi đó trên màn hình sẽ hiện ra hàng chữ: "Chiều dài là 23 mét". Như vậy trongtrường hợp này ta phải dùng đến ba lần dấu phép toán << chứ không phải một như câu lệnh trên. Ngoài ra phụ thuộc vào giá trị hiện được lưu trong biến cd, chương trình sẽ in ra số chiều dài thích hợp chứ không chỉ in cố định thành "Chiều dài là 23mét".

Ví dụ nếu cd được nhập là 15 thì lệnh trên sẽ in câu "Chiều dài là 15 mét".

Một giá trị cần in không chỉ là một biến như cd, cr, ... mà còn có thể là một biểu thức, điều này cho phép ta dễ dàng yêu cầu máy xuất ra diện tích và chu vi của hình chữ nhật khi đã biết cd và cr bằng các câu lệnh sau:

```
cout << "Dien tich = " << cd * cr ;
cout << "Chu vi = " << 2 * (cd + cr) ;
```

hoặc gộp tất cả thành 1 câu lệnh:

```
cout << "Dien tich = " << cd * cr << '\n' << " Chu vi = " << 2 * (cd + cr) ;
```

Ở đây có một ký tự đặc biệt: đó là ký tự '\n' ký hiệu cho ký tự xuống hàng (ta cũng có thể sử dụng cout<<endl), khi gặp ký tự này chương trình sẽ in các phần tiếp theo ở đầu hàng kế tiếp. Do đó kết quả của câu lệnh trên là 2 hàng sau đây trên màn hình:

```
Dien tich = 253
Chu vi = 68
```

Ở đây 253 và 68 lần lượt là các giá trị mà máy tính được từ các biểu thức cd * cr và 2 * (cd + cr) trong câu lệnh in ở trên.

✧ ***Chú ý***: Để sử dụng các câu lệnh nhập và xuất trong phần này, đầu chương trình phải có hàng khai báo

```
#include <iostream>
using namespace std;
```

Thông thường ta hay sử dụng lệnh in để in câu thông báo nhắc NLT nhập dữ liệu trước khi có câu lệnh nhập. Khi đó trên màn hình sẽ hiện hàng thông báo này rồi mới tạm dừng chờ dữ liệu nhập vào từ bàn phím. Nhờ vào thông báo này NLT sẽ biết phải nhập dữ liệu, nhập nội dung gì và như thế nào, ...

Ví dụ:

```
cout << "Hay nhap chieu dai: ";
cin >> cd;
cout << "Hay nhap chieu rong: ";
cin >> cr;
```

Khi đó máy sẽ in hàng thông báo "Hay nhap chieu dai: " và chờ sau khi NLT nhập xong 23 ↵, máy sẽ thực hiện câu lệnh tiếp theo tức in hàng thông báo "Hay nhap chieu rong: " và chờ đến khi NLT nhập xong 11 ↵ chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp theo.

Ví dụ 2: Từ các thảo luận trên ta có thể viết một cách đầy đủ chương trình tính diện tích và chu vi của một hình chữ nhật. Để chương trình có thể tính với các bộ giá trị khác nhau của chiều dài và rộng ta cần lưu giá trị này vào trong các biến chẳng hạn như cd và cr.

```
#include <iostream>
using namespace std;
main()
{
    float cd, cr ;
    cout << "Hay nhap chieu dai: " ;
    cin >> cd ;
    cout << "Hay nhap chieu rong: " ;
    cin >> cr ;
    cout << "Dien tich = " << cd * cr << '\n' ;
    cout << "Chu vi = " << 2 * (cd + cr) << '\n';
}
```

Khi chạy đến câu lệnh nhập, chương trình dừng để chờ nhận chiều dài và chiều rộng, NLT nhập các giá trị cụ thể, chương trình sẽ tiếp tục thực hiện và in ra kết quả. Thông qua câu lệnh nhập dữ liệu và 2 biến cd, cr NLT có thể yêu cầu chương trình cho kết quả của một hình chữ nhật bất kỳ chứ không chỉ trong trường hợp hình có chiều dài 23 và chiều rộng 11 như trong ví dụ cụ thể trên.

1.4.3 Định dạng thông tin cần in ra màn hình

Một số định dạng đơn giản được trình bày trước ở đây. Các định dạng chi tiết và phức tạp hơn sẽ được trình bày trong các phần sau của giáo trình. Để sử dụng các định dạng này cần khai báo file nguyên mẫu <iomanip> ở đầu chương trình bằng chỉ thị `#include <iomanip>`

- **endl:** Tương đương với ký tự xuống hàng '\n'.

- **setw(n):** Bình thường các giá trị được in ra bởi lệnh cout << sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này qui định độ rộng dành để in ra các giá trị là n cột màn hình. Nếu n lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.

- **setprecision(n):** Chỉ định số chữ số thể hiện là n. Số sẽ được làm tròn trước khi in ra.

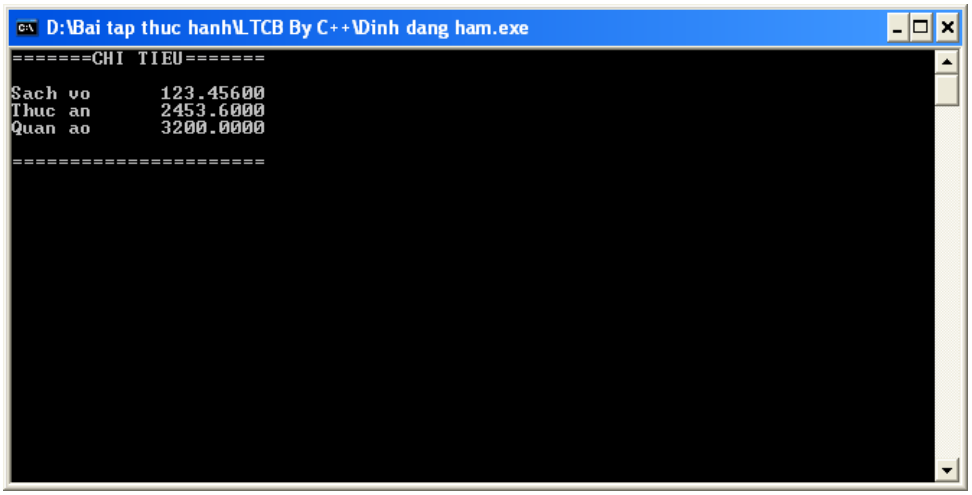
- **setiosflags(ios::showpoint):** Phương thức setprecision chỉ có tác dụng trên một hàng in. Để cố định các giá trị đã đặt cho mọi hàng in (cho đến khi đặt lại giá trị mới) ta sử dụng phương thức setiosflags(ios::showpoint).

Ví dụ 3: Viết chương trình in có định dạng mức chi tiêu của một sinh viên.

```
#include <iostream>
#include <iomanip>
#include <conio.h> // De su dung ham getch()
using namespace std;
```

```
main()
{
    cout << "=======" << "CHI TIEU" << "=======" << endl ;
    cout<<endl;
    cout << setiosflags(ios::showpoint) << setprecision(8) ;
    cout << "Sach vo" << setw(15) << 123.456 << endl;
    cout << "Thuc an" << setw(15) << 2453.6 << endl;
    cout << "Quan ao" << setw(15) << 3200.0 << endl;
    cout <<endl;
    cout << "======================" << endl ;
    getch();
}
```

Chương trình này khi chạy sẽ in ra kết quả như sau:



1.4.4 Một số lưu ý

Toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập ký tự hoặc chuỗi (chuỗi) ký tự, C++ cung cấp các phương thức (hàm) sau:

- **cin.get(c):** Cho phép nhập một ký tự vào biến ký tự c.

- **cin.getline(s, n):** Cho phép nhập tối đa n-1 ký tự vào chuỗi s.

Các hàm trên khi thực hiện sẽ lấy các ký tự còn lại trong bộ nhớ đệm (của lần nhập trước) để gán cho c hoặc s. Do toán tử cin >> x sẽ để lại ký tự xuống hàng trong bộ đệm nên ký tự này sẽ làm trôi các lệnh sau đó như cin.get(c), cin.getline(s,n) (máy không dừng để nhập cho c hoặc s).

Vì vậy trước khi sử dụng các phương thức cin.get(c) hoặc cin.getline(s, n) ta nên sử dụng phương thức cin.ignore(1) để lấy ra ký tự xuống hàng còn sót lại trong bộ đệm.

- **gets(st):** cho phép nhập vào một chuỗi ký tự tùy ý cho biến st.

Ví dụ 4: Viết chương trình nhập lần lượt giá trị cho 4 biến kiểu số nguyên, ký tự, số thực và chuỗi. Sau đó in ra kết quả giá trị của các biến đó.

```
#include <iostream>
#include <conio.h>
using namespace std;
main()
{
    int x; float y;
    char c; char st[20];
    cout << "Nhap so nguyen x = "; cin >> x;
    cin.ignore(1);
    cout << "Nhap ky tu c = "; cin.get(c);
    cout << "Nhap so thuc y = "; cin >> y;
    cin.ignore(1);
    cout << "Nhap chuoi st = "; gets(st);
    cout << "Ket qua cua cac bien da nhap la:" << endl;
    cout << "So nguyen x = " << x << endl;
    cout << "Ky tu c = " << c << endl;
    cout << "So thuc y = " << y << endl;
    cout << "Chuoi st = " << st << endl;
}
```

1.5 CÁC KIỂU DỮ LIỆU CƠ BẢN

1.5.1 Khái niệm về kiểu dữ liệu

Thông thường dữ liệu hay dùng là số và chữ. Tuy nhiên việc phân chia chỉ 2 loại dữ liệu như thế là không đủ. Để dễ dàng hơn cho lập trình, hầu hết các NNLT đều phân chia dữ liệu thành nhiều kiểu khác nhau được gọi là các kiểu cơ bản hay kiểu chuẩn. Trên cơ sở kết hợp các kiểu dữ liệu chuẩn, NLT có thể tự đặt ra các kiểu dữ liệu mới để phục vụ cho chương trình giải quyết bài toán của mình. Có nghĩa là lúc đó mỗi đối tượng được quản lý trong chương trình sẽ là một tập hợp nhiều thông tin hơn và được tạo thành từ nhiều loại (kiểu) dữ liệu khác nhau.

Một biến là một số ô nhớ liên tiếp nào đó trong bộ nhớ dùng để lưu trữ dữ liệu (vào, ra hay kết quả trung gian) trong quá trình hoạt động của chương trình. Để quản lý chặt chẽ các biến, NLT cần khai báo cho chương trình biết trước tên biến và kiểu của dữ liệu được chứa trong biến. Việc khai báo này sẽ làm chương trình quản lý các biến dễ dàng hơn

như trong việc cấp phát bộ nhớ cũng như quản lý các tính toán trên biến theo nguyên tắc: chỉ có các dữ liệu cùng kiểu với nhau mới được phép làm toán với nhau. Do đó, khi đề cập đến một kiểu chuẩn của một NNLT, thông thường chúng ta sẽ xét đến các yếu tố sau:

- ***Tên kiểu***: là một từ dành riêng để chỉ định kiểu của dữ liệu.

Ví dụ: Kiểu số nguyên có tên là `int`.

- ***Kích thước***: là số byte vùng nhớ cần thiết để lưu trữ một đơn vị dữ liệu thuộc kiểu này. Thông thường số byte này phụ thuộc vào các trình biên dịch và hệ thống máy tính khác nhau, ở đây ta chỉ xét đến hệ thống máy PC thông dụng hiện nay.

Ví dụ: Kiểu số nguyên `(int)` có kích thước là 2 bytes.

- ***Miền giá trị***: là tập giá trị mà một biến thuộc kiểu dữ liệu này sẽ có thể nhận được, người ta thường ghi giá trị nhỏ nhất và lớn nhất là bao nhiêu. Hiển nhiên các giá trị này phụ thuộc vào số byte mà hệ thống máy tính qui định cho từng kiểu. NLT cần nhớ đến miền giá trị này để khai báo kiểu cho các biến cần sử dụng một cách thích hợp.

Ví dụ: Kiểu số nguyên `(int)` có miền giá trị là -32768 .. 32767.

Bảng 1.1 sau đây mô tả tóm tắt một số kiểu dữ liệu chuẩn được sử dụng trong ngôn ngữ lập trình C++

LOẠI DỮ LIỆU	TÊN KIỂU	KÍCH THƯỚC	MIỀN GIÁ TRỊ
Luận lý	bool	1 byte	false, true
Ký tự	[signed] char	1 byte	-128 .. 127
	unsigned char	1 byte	0 .. 255
Số nguyên	int	2 bytes	-32768 .. 32767
	unsigned int	2 bytes	0 .. 65535
	short	2 bytes	-32768 .. 32767
	long [int]	4 bytes	-2^{31} .. $2^{31} - 1$
	unsigned long	4 bytes	0 .. $2^{32} - 1$
Số thực	float	4 bytes	$3.4*10^{-38}$ .. $3.4*10^{+38}$
	double	8 bytes	$1.7*10^{-308}$ .. $1.7*10^{+308}$
	long double	10 bytes	$3.4*10^{-4932}$ .. $1.1*10^{+4932}$

1.5.2 Kiểu luận lý

Kiểu luận lý còn được gọi là kiểu logic (bool) chỉ gồm 2 giá trị là false (sai) và true (đúng). Kết quả của một biểu thức điều kiện hay biểu thức so sánh luôn là kiểu luận lý.

Trong ngôn ngữ lập trình trình C chuẩn và các phiên bản trước của Dev-C++ không có kiểu bool, khi đó để thể hiện giá trị false và true người ta sử dụng kiểu số nguyên (int) với giá trị bằng 0 tương ứng là false và giá trị khác 0 tương ứng là true.

1.5.3 Kiểu ký tự

Một ký tự là một ký hiệu trong bảng mã ASCII. Như đã biết một số ký tự có mặt chữ trên bàn phím (ví dụ các chữ cái, chữ số) trong khi một số ký tự lại không (ví dụ ký tự biểu diễn việc lùi lại một ô trong văn bản, ký tự chỉ việc kết thúc một hàng haykết thúc một văn bản).

Do vậy để biểu diễn một ký tự người ta dùng chính mã ASCII của ký tự đó trong bảng mã ASCII và thường gọi là giá trị của ký tự. Ví dụ phát biểu "Cho ký tự 'A'" là tương đương với phát biểu "Cho ký tự 65" (65 là mã ASCII của ký tự 'A') hay "Xóa ký tự xuống hàng" là tương đương với phát biểu "Xóa ký tự 13" vì 13 là mã ASCII của ký tự xuống hàng. Như vậy một biến kiểu ký tự có thể được nhận giá trị theo 2 cách tương đương là chữ hoặc số.

Ví dụ: Giả sử c là một biến ký tự thì câu lệnh gán c = 'A' cũng tương đương với câu lệnh gán c = 65. Tuy nhiên để sử dụng giá trị số của một ký tự c nào đó ta phải yêu cầu đổi c sang giá trị số bằng câu lệnh int(c).

1.5.4 Kiểu số nguyên

Các số nguyên được phân chia thành 4 loại kiểu khác nhau với các miền giá trị tương ứng được cho trong bảng 1.1. Đó là kiểu số nguyên ngắn (short) tương đương với kiểu số nguyên (int) sử dụng 2 bytes và số nguyên dài (long int hoặc có thể viết gọn hơn là long) sử dụng 4 bytes. Kiểu số nguyên thường được chia làm 2 loại có dấu (int) và không dấu (unsigned int hoặc có thể viết gọn hơn là unsigned).

Ta thường sử dụng kiểu int cho các số nguyên trong các bài toán với miền giá trị vừa phải chẳng hạn như các biến đếm trong các vòng lặp, năm sinh của một người, ...

1.5.5 Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu float hoặc double mà miền giá trị của chúng được cho trong bảng 1.1. Các giá trị số kiểu double được gọi là số thực với độ chính xác kép vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu float để đảm bảo số số lẻ sau một số thực có thể tăng lên đảm bảo tính chính xác cao hơn so với số kiểu float. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu float là đủ.

Như đã nhắc đến trong phần nhập/xuất, liên quan đến việc in ấn số thực ta có một vài cách thiết lập dạng in theo ý muốn, ví dụ độ rộng tối thiểu để in một số hay số chữ số cần in là bao nhiêu, ...

Ví dụ 5: Chương trình sau đây sẽ in diện tích và chu vi của một hình tròn có bán kính 3cm với 6 số chữ số được in ra.

```
#include <iostream>
#include <conio.h>
#include <iomanip>
using namespace std;
main()
{
    float r = 3;
    cout<< setiosflags(ios::showpoint);
    cout<< "DT = "<<setprecision(6)<<r*r*3.1416<<endl;
    cout<< "CV = "<<setprecision(6)<<2*3.1416*r<<endl;
}
```

1.6 HẰNG SỐ

1.6.1 Định nghĩa

Hằng số (constant) là một giá trị cố định trong suốt quá trình thực thi chương trình.

Ví dụ:

3 là hằng số nguyên

'A' là hằng ký tự

5.0 là hằng số thực

"Viet Nam" là hằng chuỗi ký tự

Một giá trị có thể được hiểu dưới nhiều kiểu khác nhau, do vậy khi viết hằng ta cũng cần có dạng viết thích hợp.

1.6.2 Một số hằng thông dụng

Đối với một số hằng ký tự thường dùng nhưng không có mặt chữ tương ứng, hoặc các ký tự được dành riêng với nhiệm vụ khác, khi đó thay vì phải nhớ giá trị của chúng ta có thể viết theo qui ước sau:

'\n'	:	biểu thị ký tự xuống hàng (cũng tương đương với endl)
'\t'	:	ký tự tab
'\a'	:	ký tự chuông (tức thay vì in ký tự, loa sẽ phát ra một tiếng 'bíp')
'\r'	:	xuống hàng
'\f'	:	kéo trang
'\\'	:	dấu \
'\?'	:	dấu chấm hỏi ?

'\"	:	dấu nháy đơn '

'\'"	:	dấu nháy kép "

'\kkk'	:	ký tự có mã là kkk trong hệ 8

'\xkk'	:	ký tự có mã là kk trong hệ 16

Ví dụ: `cout << "Hom nay troi \t nang \a \a \a \n" ;` sẽ in ra màn hình hàng chữ "Hôm nay trời" sau đó bỏ một khoảng cách bằng một tab (khoảng 8 dấu cách) rồi in tiếp chữ "nắng", tiếp theo phát ra 3 tiếng chuông và cuối cùng con trỏ trên màn hình sẽ nhảy xuống đầu hàng mới.

1.6.3 Khai báo

Một giá trị cố định (hằng) được sử dụng nhiều lần trong chương trình đôi khi sẽ thuận lợi hơn nếu ta đặt cho nó một tên gọi, thao tác này được gọi là khai báo hằng. Ví dụ một chương trình quản lý sinh viên với giả thiết số sinh viên tối đa là 50. Nếu số sinh viên tối đa không thay đổi trong chương trình ta có thể đặt cho nó một tên gọi như **sosv** chẳng hạn. Trong suốt chương trình bất kỳ chỗ nào xuất hiện giá trị 50 ta đều có thể thay nó bằng **sosv**.

Việc sử dụng tên hằng thay cho hằng có nhiều điểm thuận lợi như sau:

- Chương trình dễ đọc hơn, vì thay cho các con số ít có ý nghĩa, một tên gọi sẽ làm NLT dễ hình dung vai trò, nội dung của nó. Ví dụ, khi gặp tên gọi **sosv** NLT sẽ dễ hình dung "đây là số sinh viên tối đa trong một lớp", trong khi số 50 có thể là số sinh viên mà cũng có thể là số thứ tự của một sinh viên nào đó.

- Chương trình dễ sửa lỗi hơn, ví dụ bây giờ nếu muốn thay đổi chương trình sao cho bài toán quản lý được thực hiện với số sinh viên tối đa là 60, khi đó ta cần tìm và thay thế hàng trăm vị trí xuất hiện của 50 thành 60. Việc thay thế như vậy dễ gây ra lỗi vì có thể không tìm thấy hết các số 50 trong chương trình hoặc thay nhầm số 50 với ý nghĩa khác như số thứ tự của một sinh viên nào đó chẳng hạn. Nếu trong chương trình sử dụng hằng **sosv**, bây giờ việc thay thế trở nên chính xác và dễ dàng hơn bằng thao tác khai báo lại giá trị hằng **sosv** bằng 60. Lúc đó trong chương trình bất kỳ nơi nào gặp tên hằng **sosv** đều được chương trình hiểu với giá trị 60.

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng

hoặc:

const <kiểu> tên_hằng = giá_trị_hằng;

Ví dụ:

```
#define sosv 50
#define MAX 100
const int sosv = 50;
const float MAX = 100.0;
```

1.7 BIẾN

1.7.1 Định nghĩa

Biến (variable – biến số) là các tên gọi để lưu giá trị khi làm việc trong chương trình. Các giá trị được lưu có thể là các giá trị dữ liệu ban đầu, các giá trị trung gian tạm thời trong quá trình tính toán hoặc các giá trị kết quả cuối cùng. Khác với hằng, giá trị của biến có thể thay đổi trong quá trình làm việc bằng các lệnh đọc vào từ bàn phím hoặc lệnh gán. Hình ảnh cụ thể của biến là một số ô nhớ trong bộ nhớ được sử dụng để lưu các giá trị của biến.

1.7.2 Khai báo

Mọi biến phải được khai báo trước khi sử dụng. Một khai báo như vậy sẽ báo cho chương trình biết về một biến mới gồm có: tên của biến, kiểu của biến (tức là kiểu của giá trị dữ liệu mà biến sẽ lưu giữ). Thông thường với nhiều NNLT tất cả các biến phải được khai báo ngay từ đầu chương trình hay đầu của hàm, tuy nhiên để thuận tiện hơn cho người lập trình C++ cho phép khai báo biến ngay bên trong chương trình hoặc hàm, có nghĩa là bất kỳ lúc nào NLT thấy cần thiết sử dụng biến mới, họ có quyền khai báo và sử dụng nó từ đó trở đi.

Cú pháp khai báo biến gồm tên kiểu, tên biến và có thể có hay không khởi tạo giá trị ban đầu cho biến. Để khởi tạo hoặc thay đổi giá trị của biến ta dùng lệnh gán (=).

1.7.2.1 Khai báo không khởi tạo

tên_kiểu tên_biến_1;

tên_kiểu tên_biến_2;

tên_kiểu tên_biến_3;

Nhiều biến cùng kiểu có thể được khai báo trên cùng một hàng:

tên_kiểu tên_biến_1, tên_biến_2, tên_biến_3;

Ví dụ:

```
int i, j;
float x;
char c, d[100];
```

1.7.2.2 Khai báo có khởi tạo

Trong câu lệnh khai báo, các biến có thể được gán ngay giá trị ban đầu bởi phép toán gán (=) theo cú pháp:

tên_kiểu tên_biến_1 = gt_1;

tên_kiểu tên_biến_2 = gt_2,

tên_kiểu tên_biến_3 = gt_3;

Trong đó các giá trị gt_1, gt_2, gt_3 có thể là các hằng, biến hoặc biểu thức.

Nhiều biến cùng kiểu có thể được khai báo trên cùng một hàng:

tên_kiểu tên_biến_1 = gt_1, tên_biến_2 = gt_2, tên_biến_3 = gt_3;

Ví dụ:

```
int i = 2, j , k = (i + 5) * 4;
float eps = 1.0e-6;
char d[100] = "Cong nghe thong tin";
```

1.7.3 Phạm vi của biến

Như đã biết chương trình là một tập hợp các hàm, các câu lệnh cũng như các khai báo. Phạm vi tác dụng của một biến là nơi mà biến có tác dụng, tức là hàm nào, câu lệnh nào được phép sử dụng biến đó. Một biến xuất hiện trong chương trình có thể được sử dụng bởi hàm này nhưng không được bởi hàm khác hoặc bởi cả hai, điều này phụ thuộc chặt chẽ vào vị trí nơi biến được khai báo. Một nguyên tắc đầu tiên là biến sẽ có tác dụng kể từ vị trí nó được khai báo cho đến hết khối lệnh chứa nó.

1.7.4 Gán giá trị cho biến

Trong các ví dụ trước chúng ta đã sử dụng phép gán hay lệnh gán (assign) dù nó chưa được trình bày, đơn giản một phép gán mang ý nghĩa tạo giá trị mới cho một biến. Khi biến được gán giá trị mới, giá trị cũ sẽ được tự động xóa đi bất kể trước đó nó chứa giá trị nào (hoặc chưa có giá trị, ví dụ chỉ mới vừa khai báo xong).

Cú pháp của phép gán như sau:

tên_biến = biểu thức;

Khi gặp phép gán chương trình sẽ tính toán giá trị của biểu thức bên vế phải sau đó gán giá trị này cho biến bên vế trái.

Ví dụ:

```
int n, i = 3;       // khởi tạo i bằng 3
n = 10;             // gán cho n giá trị 10 hay gán giá trị 10 vào biến n
cout << n <<", " << i << endl;              // in ra: 10, 3
i = n / 2;          // gán lại giá trị của i bằng n/2 = 5
cout << n <<", " << i << endl;              // in ra: 10, 5
```

1.7.5 Một số điểm lưu ý về phép gán

- Với ý nghĩa thông thường của phép toán (nghĩa là tính toán và cho lại một giá trị) thì phép gán còn một nhiệm vụ nữa là trả lại một giá trị. Giá trị trả lại của phép gán chính là giá trị của biểu thức sau dấu bằng. Lợi dụng điều này C++ cho phép chúng ta gán "kép" cho nhiều biến nhận cùng một giá trị bởi cú pháp:

biến_1 = biến_2 = … = biến_n = gt;

Với cách gán này tất cả các biến sẽ nhận cùng một giá trị là gt.

Ví dụ:

```
int i, j, k;
i = j = k = 1;
```

Biểu thức gán trên có thể được viết lại như (i = (j = (k = 1))), có nghĩa đầu tiên để thực hiện phép gán giá trị cho biến i chương trình phải tính biểu thức (j = (k = 1)), tức phải tính k = 1, đây là phép gán, gán giá trị 1 cho k và trả lại giá trị 1, giá trị trả lại này sẽ được gán cho j và trả lại giá trị 1 để tiếp tục gán cho i.

- Ngoài việc gán kép như trên, phép gán còn được phép xuất hiện trong bất kỳ biểu thức nào, điều này cho phép trong một biểu thức có phép gán, nó không chỉ tính toán mà còn gán giá trị cho các biến. Ví dụ n = 3 + (i = 2) sẽ cho ta i = 2 và n = 5.

Việc sử dụng nhiều chức năng của một câu lệnh làm cho chương trình gọn gàng hơn (trong một số trường hợp) nhưng cũng trở nên khó đọc, chẳng hạn câu lệnh trên có thể viết tách thành 2 câu lệnh i = 2; n = 3 + i; sẽ dễ đọc hơn ít nhất đối với những người mới bắt đầu tìm hiểu về lập trình.

1.8 PHÉP TOÁN, BIỂU THỨC VÀ CÂU LỆNH

1.8.1 Phép toán

1.8.1.1 Các phép toán số học: +, -, *, /, %

- Các phép toán + (cộng), - (trừ), * (nhân) được hiểu theo nghĩa thông thường trong số học.

- Phép toán a / b (chia) được thực hiện theo kiểu của các toán hạng, tức nếu cả hai toán hạng là số nguyên thì kết quả của phép chia chỉ lấy phần nguyên, ngược lại nếu 1 trong 2 toán hạng là số thực thì kết quả là số thực.

Ví dụ:

```
13/5 = 2                                  // vì 13 và 5 là 2 số nguyên
13.0/5 = 13/5.0 = 13.0/5.0 = 2.6          // vì có ít nhất 1 toán hạng là thực
```

- Phép toán a % b (lấy phần dư) trả lại phần dư của phép chia a / b, trong đó a và b là 2 số nguyên.

Ví dụ:

```
13%5 = 3      // phần dư của 13/5
5%13 = 5      // phần dư của 5/13
```

1.8.1.2 Các phép toán tự tăng, tự giảm: i++, ++i, i--, --i

- Phép toán ++i và i++ sẽ cùng tăng i lên 1 đơn vị tức tương đương với câu lệnh i = i+1. Tuy nhiên nếu 2 phép toán này nằm trong câu lệnh hoặc biểu thức thì ++i khác với i++. Cụ thể ++i sẽ tăng i, sau đó i mới được tham gia vào tính toán trong biểu thức, ngược lại i++ sẽ tăng i sau khi biểu thức được tính toán xong (với giá trị i cũ). Điểm khác biệt này được minh họa thông qua ví dụ sau với giả sử i = 3, j = 15.

Phép toán	Tương đương	Kết quả
i = ++j ; // tăng trước	j = j + 1 ; i = j ;	i = 16 , j = 16
i= j++ ; // tăng sau	i = j ; j = j + 1 ;	i = 15 , j = 16
j = ++i + 5 ;	i = i + 1 ; j = i + 5;	i = 4, j = 9
j = i++ + 5 ;	j = i + 5; i = i + 1;	i = 4, j = 8

✧ ***Ghi chú***: Việc kết hợp phép toán tự tăng, tự giảm vào trong biểu thức hoặc câu lệnh sẽ làm cho chương trình ngắn gọn hơn nhưng có thể khó hiểu hơn.

1.8.1.3 Các phép toán so sánh và logic

Đây là các phép toán mà giá trị trả lại là đúng (true) hoặc sai (false). Nếu giá trị của biểu thức là đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của 2 khái niệm "đúng" và "sai". Mở rộng hơn C++ quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai".

- *Các phép toán so sánh*: == (bằng nhau), != (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Hai toán hạng của các phép toán này phải cùng kiểu.

✧ ***Chú ý***: Cần phân biệt phép toán gán (=) và phép toán so sánh (==). Phép gán vừa gán giá trị cho biến vừa trả lại giá trị bất kỳ (là giá trị của toán hạng bên phải), trong khi phép so sánh luôn luôn trả lại giá trị 1 hoặc 0.

- *Các phép toán logic*: && (và), || (hoặc), ! (không, phủ định)

Hai toán hạng của loại phép toán này phải có kiểu logic tức chỉ nhận một trong hai giá trị "đúng" (được thể hiện bởi các số nguyên khác 0) hoặc "sai" (thể hiện bởi 0). Khi đó giá trị trả lại của phép toán là 1 hoặc 0 và được cho trong *bảng 1.2* sau:

A	B	A && B	A || B	! A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

✧ ***Chú ý***: Việc đánh giá biểu thức được tiến hành từ trái sang phải và sẽ dừng khi biếtkết quả mà không chờ đánh giá hết biểu thức. Cách đánh giá này sẽ cho những kết quả phụ khác nhau nếu trong biểu thức ta "tranh thủ" đưa thêm vào các phép toán tự tăng, tự giảm.

Ví dụ: Cho i = 2, j = 3, xét 2 biểu thức sau đây:

x = (++i < 4 && ++j > 5) cho kết quả x = 0 , i = 3 , j = 4

y = (++j > 5 && ++i < 4) cho kết quả y = 0 , i = 2 , j = 4

Cách viết hai biểu thức là như nhau (ngoại trừ hoán đổi vị trí 2 toán hạng của phép toán &&). Với giả thiết i = 2 và j = 3 ta thấy cả hai biểu thức trên cùng nhận giá trị 0.

Tuy nhiên các giá trị của i và j sau khi thực hiện xong hai biểu thức này sẽ có kết quả khác nhau. Cụ thể với biểu thức đầu vì ++i < 4 là đúng nên chương trình phải tiếp tục tính tiếp ++j > 5 để đánh giá được biểu thức. Do vậy sau khi đánh giá xong cả i và j đều được tăng lên 1 nên ta có i = 3, j = 4.

Trong khi đó với biểu thức sau do ++j > 5 là sai nên chương trình có thể kết luận được toàn bộ biểu thức là sai mà không cần tính tiếp ++i < 4. Có nghĩa là chương trình sau khi đánh giá xong ++j > 5 sẽ dừng và vì vậy chỉ có biến j được tăng 1, từ đó ta có i = 2, j = 4 khác với kết quả của biểu thức trên. Ví dụ này một lần nữa nhắc ta chú ý kiểm soát kỹ việc sử dụng các phép toán tự tăng, tự giảm trong biểu thức và trong câu lệnh.

1.8.2 Các phép gán

- Phép gán thông thường: Đây là phép gán đã được trình bày trong mục trước.

- Phép gán có điều kiện:

biến = (Điều_kiện) ? a : b ;

Điều_kiện là một biểu thức logic, a, b là các biểu thức bất kỳ cùng kiểu với kiểu của biến. Phép toán này gán giá trị a cho biến nếu điều kiện đúng và gán giá trị b cho biến nếu ngược lại.

Ví dụ:

```
x = (3 + 4 < 7) ? 10 : 20      // x = 20 vì 3 + 4 < 7 là sai
x = (3 + 4) ? 10 : 20          // x = 10 vì 3 + 4 khác 0, tức điều kiện đúng
x = (a > b) ? a : b            // x = số lớn hơn trong 2 số a, b.
```

- Cách viết gọn của phép gán: Một phép gán dạng x = x @ a ; có thể được viết gọn dưới dạng x @= a trong đó @ là các phép toán số học, xử lý bit ...

Ví dụ:

Thay cho viết x = x + 2 có thể viết x += 2;

hoặc x = x / 2; x = x * 2 có thể được viết lại như x /= 2; x *= 2;

1.8.3 Biểu thức

Biểu thức (expression) là dãy ký hiệu kết hợp giữa các toán hạng, phép toán và cặp dấu () theo một qui tắc nhất định. Các toán hạng có thể là hằng, biến, hàm. Biểu thức cung cấp một cách thức để tính giá trị mới dựa trên các toán hạng và toán tử trong biểu thức.

Ví dụ:

```
(x + y) * 2 - 4
3 - x + sqrt(y)
(-b + sqrt(delta)) / (2*a)
```

1.8.3.1 Thứ tự ưu tiên của các phép toán

Để tính giá trị của một biểu thức cần có một trật tự tính toán cụ thể và thống nhất.

Ví dụ:

Xét biểu thức x = 3 + 4 * 2 + 7

- Nếu tính theo đúng trật tự từ trái sang phải, thì x = ((3 + 4) * 2) + 7 = 21
- Nếu ưu tiên dấu + được thực hiện trước dấu *, thì x = (3 + 4) * (2 + 7) = 63
- Nếu ưu tiên dấu * được thực hiện trước dấu +, thì x = 3 + (4 * 2) + 7 = 18

C++ qui định trật tự tính toán theo các mức độ ưu tiên như sau:

1. Các biểu thức trong cặp dấu ngoặc ()
2. Các phép toán 1 ngôi (tự tăng, tự giảm, lấy địa chỉ, lấy nội dung con trỏ, …)
3. Các phép toán số học
4. Các phép toán so sánh, logic
5. Các phép gán

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, −, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái.

Ví dụ:

Theo mức ưu tiên đã qui định, biểu thức tính x trong ví dụ trên sẽ có giá trị là 18

Phần lớn các trường hợp muốn tính toán theo một trật tự nào đó ta nên sử dụng cụ thể các dấu ngoặc (vì các biểu thức trong dấu ngoặc được tính trước).

1.8.3.2 Phép chuyển đổi kiểu

Khi tính toán một biểu thức phần lớn các phép toán đều yêu cầu các toán hạng phải cùng kiểu. Chẳng hạn để phép gán thực hiện được thì giá trị của biểu thức phải có cùng kiểu với biến. Trong trường hợp kiểu của giá trị biểu thức khác với kiểu của biến thì hoặc là chương trình sẽ tự động chuyển kiểu giá trị biểu thức về thành kiểu của biến được gán (nếu được) hoặc sẽ báo lỗi. Do vậy khi cần thiết NLT phải sử dụng các câu lệnh để chuyển kiểu của biểu thức cho phù hợp với kiểu của biến. Có 2 cách chuyển đổi kiểu là chuyển kiểu tự động và ép kiểu.

- *Chuyển kiểu tự động*: Về mặt nguyên tắc, khi cần thiết các kiểu có giá trị thấp sẽ được chương trình tự động chuyển lên kiểu cao hơn cho phù hợp với phép toán. Cụ thể phép chuyển kiểu có thể được thực hiện theo sơ đồ như sau:

char ↔ int → long int → float → double

Ví dụ:

int i = 3;

```
float f ; f
= i + 2;
```

Trong ví dụ trên i có kiểu nguyên và vì vậy i + 2 cũng có kiểu nguyên trong khi f có kiểu thực. Tuy nhiên phép toán gán này là hợp lệ vì chương trình sẽ tự động chuyển kiểu của i + 2 (bằng 5) sang kiểu thực (bằng 5.0) rồi mới gán cho f.

- *Ép kiểu*: Trong chuyển kiểu tự động, chương trình chuyển các kiểu từ thấp lên cao, tuy nhiên chiều ngược lại không thể thực hiện được vì nó có thể gây mất dữ liệu. Do đó nếu cần thiết NLT phải ra lệnh cho chương trình chuyển kiểu từ cao xuống thấp.

Ví dụ:

```
int i;
float f = 3 ;      // tự động chuyển 3 thành 3.0 và gán cho f
i = f + 2 ;        // sai (báo lỗi) mặc dù f + 2 = 5 nhưng không gán được cho i
```

Trong ví dụ trên để câu lệnh i = f + 2 thực hiện được (không báo lỗi) ta phải ép kiểu của biểu thức f + 2 về thành kiểu nguyên. Cú pháp tổng quát như sau:

(tên_kiểu)biểu_thức

hoặc:

tên_kiểu(biểu_thức)

Dưới đây ta sẽ xét một số ví dụ về lợi ích của việc ép kiểu.

- Phép ép kiểu từ một số thực về số nguyên sẽ cắt bỏ tất cả phần lẻ của số thực, chỉ giữ lại phần nguyên. Như vậy để tính phần nguyên của một số thực x ta chỉ cần ép kiểu của x về thành kiểu nguyên, có nghĩa int(x) là phần nguyên của số thực x bất kỳ.

Ví dụ 6:

Để kiểm tra một số nguyên n có phải là số chính phương, ta cần tính căn bậc hai của n. Nếu căn bậc hai của n là số nguyên x thì n là số chính phương, tức là nếu int(x) = x thì x nguyên và khi đó n là chính phương. Chương trình cụ thể như sau:

```
#include <iostream>
#include <math.h>
using namespace std;
main()
{
    int n;
    cout << "Nhap n = ";
    cin >> n;
    float x = sqrt(n); // Ham sqrt de tinh can bac hai
```

```
    if (int(x) == x)
        cout << n << " la so chinh phuong" << endl;
    else
        cout << n << " khong la so chinh phuong" << endl;
}
```

1.8.4 Câu lệnh và khối lệnh

Một câu lệnh trong C++ được thiết lập từ các từ khóa và các biểu thức,… và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ nhập/xuất hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
cin >> x >> y ;
x =  x + 1 ;
y = sqrt(x) ;
cout << x << endl;
cout << y << endl;
```

Các câu lệnh được phép viết trên cùng một dòng hoặc nhiều dòng. Một số câu lệnh được gọi là lệnh có cấu trúc, tức là bên trong nó lại chứa dãy lệnh khác. Dãy lệnh này phải được bao giữa cặp dấu ngoặc {} và được gọi là khối lệnh.

Ví dụ tất cả các lệnh trong một hàm (như hàm main()) luôn luôn là một khối lệnh. Một đặc điểm của khối lệnh là các biến được khai báo trong khối lệnh nào thì chỉ có tác dụng trong khối lệnh đó. Chi tiết hơn về các đặc điểm của lệnh và khối lệnh sẽ được trình bày trong các chương tiếp theo của giáo trình.

1.9 MỘT SỐ HÀM THÔNG DỤNG

1.9.1 Nhóm hàm toán học

Các hàm này đều được khai báo trong file nguyên mẫu ***math.h***.

- abs(x), labs(x), fabs(x) : trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.

- pow(x, y) : hàm mũ, trả lại giá trị x lũy thừa y (x^y).

- exp(x) : hàm mũ, trả lại giá trị e mũ x (e^x).

- log(x), log10(x) : trả lại lôgarit cơ số e và logarit thập phân của x (lnx, logx) .

- sqrt(x) : trả lại căn bậc 2 của x.

- sin(x), cos(x), tan(x), asin(x), acos(x), atan(x) : trả lại các giá trị sinx, cosx, tgx, arcsinx, arccosx, arctgx.

1.9.2 Nhóm hàm kiểm tra ký tự

Các hàm này đều được khai báo trong file nguyên mẫu ***ctype.h***.

- isalnum(kt) : kiểm tra ký tự kt có phải là chữ cái hoặc chữ số hay không.
- isalpha(kt) : kiểm tra ký tự kt có phải là chữ cái haykhông.
- isdigit(kt) : kiểm tra ký tự kt có phải là chữ số haykhông.
- islower(kt) : kiểm tra ký tự kt có phải là chữ cái thường (a .. z) hay không.
- isupper(kt) : kiểm tra ký tự kt có phải là chữ cái hoa (A .. Z) hay không.
- isspace(kt) : kiểm tra ký tự kt có phải là ký tự trống haykhông.

1.9.3 Nhóm hàm chuyển đổi dữ liệu

Các hàm này đều được khai báo trong file nguyên mẫu ***ctype.h***.

- toupper(c) : chuyển từ chữ thường sang chữ hoa.
- tolower(c) : chuyển từ chữ hoa sang chữthường.
- atof(s) : chuyển chuỗi s sang giá trị double.
- atoi(s) : chuyển chuỗi s sang giá trị int.
- atol(s) : chuyển chuỗi s sang giá trị long.

BÀI TẬP CHƯƠNG 1

-----ଓ✧ଌ-----

1. Viết chương trình in nội dung một bài thơ mà bạn thích.

2. Viết chương trình nhập vào cạnh của hình vuông, in ra chu vi và diện tích.

3. Viết chương trình nhập vào 4 giá trị lần lượt là số thực, nguyên, nguyên dài và ký tự. In ra màn hình các giá trị này.

4. Viết chương trình nhập vào một ký tự, in ra ký tự đó và mã ASCII của nó.

5. Viết chương trình nhập 3 số thực a, b, c, in ra màn hình hàng chữ phương trình có dạng ax^2 + bx + c = 0, trong đó các giá trị a, b, c chỉ in 2 số lẻ (ví dụ với a = 5.141, b = −2, c = 0.8 **in ra** 5.14 x^2 −2.00 x + 0.80 = 0).

6. Viết chương trình in ra tổng, tích, hiệu và thương của 2 số thực được nhập vào từ bàn phím.

7. Viết chương trình in ra trung bình cộng, trung bình nhân của 3 số nguyên được nhập vào từ bàn phím.

8. Viết chương trình nhập vào 4 chữ số. In ra tổng của 4 chữ số này và chữ số hàng chục, hàng đơn vị của tổng đó (ví dụ 4 chữ số 3, 1, 8, 5 có tổng là 17 và chữ số hàng chục là 1 và hàng đơn vị là 7, **in ra** 17, 1, 7).

9. Viết chương trình nhập vào bán kính của hình tròn, in ra diện tích, chu vi của nó.

10. Viết chương trình nhập vào một số nguyên (có 4 chữ số). In ra tổng của 4 chữ số này và chữ số đầu, chữ số cuối (ví dụ số 3185 có tổng các chữ số là 17, chữ số đầu và cuối là 3 và 5, **in ra** 17, 3, 5).

11. Viết chương trình nhập vào hệ số lương (là số thực có 2 chữ số lẻ) của 1 nhân viên. In ra lương của nhân viên đó, với lương = 1.150.000 * hệ số lương.

12. Viết chương trình nhập 2 số nguyên a và b, hãy đổi giá trị của a và b theo 2 cách:

- Dùng biến phụ t: t = a; a = b; b = t;
- Không dùng biến phụ: a = a + b; b = a - b; a = a - b;

In kết quả ra màn hình giá trị của a và b trước và sau khi đổi.

13. Viết chương trình nhập vào năm sinh của 1 người. Cho biết người đó năm nay được bao nhiêu tuổi.

14. Viết chương trình tính chỉ số pignet của 1 người khi biết chiều cao (cm), vòng ngực trung bình (cm) và trọng lượng (kg).

Với: chỉ số pignet = chiều cao - (vòng ngực trung bình + trọng lượng)

15. Viết chương trình nhập vào chiều dài 3 cạnh a, b, c của 1 tam giác. Tính chu vi và diện tích của tam giác đó theo công thức sau:

Chu vi : CV = a + b + c

Diện tích : DT = $\sqrt{p*(p-a)*(p-b)*(p-c)}$ với p = CV/2.

CHƯƠNG 2: CÁC CẤU TRÚC ĐIỀU KHIỂN

-----ଓ✧ଌ-----

2.1 CẤU TRÚC RẼ NHÁNH

Nói chung việc thực hiện chương trình là hoạt động tuần tự, tức là thực hiện từng lệnh một từ câu lệnh bắt đầu của chương trình cho đến câu lệnh cuối cùng. Tuy nhiên, để việc lập trình hiệu quả hơn hầu hết các NNLT cấp cao đều có các câu lệnh rẽ nhánh và các câu lệnh lặp cho phép thực hiện các câu lệnh của chương trình không theo cấu trúc tuần tự.

2.1.1 Cấu trúc if

2.1.1.1 Ý nghĩa

Một câu lệnh if cho phép chương trình có thể thực hiện khối lệnh này hay khối lệnh khác phụ thuộc vào điều kiện được viết trong câu lệnh là đúng hay sai. Nói cách khác câu lệnh if cho phép chương trình rẽ nhánh (chỉ thực hiện 1 trong 2 nhánh).

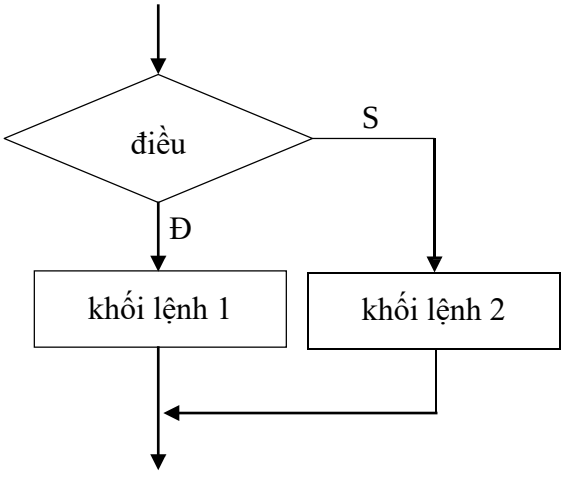
2.1.1.2 Cú pháp

```
if (điều kiện)
    khối lệnh 1;
[else
    khối lệnh 2;]
```

Trong cú pháp trên câu lệnh if có hai dạng: có else và không có else. Điều kiện là một biểu thức logic tức nó có giá trị đúng (khác 0) hoặc sai (bằng 0).

Khi chương trình thực hiện câu lệnh if nó sẽ tính biểu thức điều kiện. Nếu điều kiện đúng chương trình sẽ tiếp tục thực hiện các lệnh trong khối lệnh 1, ngược lại nếu điều kiện sai chương trình sẽ thực hiện khối lệnh 2 (nếu có else) hoặc không làm gì (nếu không có else).

2.1.1.3 Lưu đồ cú pháp



Hình 2.1: Lưu đồ cú pháp cấu trúc if

2.1.1.4 Đặc điểm

- Đặc điểm chung của các câu lệnh có cấu trúc là bản thân nó chứa các câu lệnh khác. Điều này cho phép các câu lệnh if có thể lồng nhau.

- Nếu nhiều câu lệnh if (có else và không else) lồng nhau việc hiểu if và else nào đi với nhau cần phải chú ý. Qui tắc là else sẽ đi với if gần nó nhất mà chưa được ghép cặp với else khác.

Ví dụ:

Câu lệnh:

```
if (n > 0)
    if (a > b)
        c = a;
    else
        c = b;
```

Tương đương với câu lệnh:

```
if (n > 0)
    {
        if (a > b)
            c = a;
        else
            c = b;
    }
```

2.1.1.5 Các lưu ý

- Điều kiện (hay biểu thức) sau if phải đặt trong cặp dấu ngoặc đơn ( ).

- Nếu khối lệnh có nhiều hơn 1 câu lệnh thì phải bao chúng trong cặp dấu ngoặc móc{ }.

2.1.1.6 Ví dụ minh họa

Ví dụ 1: Viết chương trình nhập vào 2 số nguyên, in ra số lớn hơn trong 2 số đó.

```
#include <iostream>
using namespace std;
main()
{
     int a, b, max;
     cout<<"Nhap vao 2 so nguyen a va b : ";
     cin>>a>>b;
    if (a > b)
```

```
        max = a;
    else
        max = b;
    cout<<"So lon hon la : "<<max<<endl;
}
```

Ví dụ 2: Viết chương trình nhập vào 1 năm bất kỳ, cho biết năm đó có phải là năm nhuận không. Biết rằng năm thứ n là nhuận nếu nó chia hết cho 4, nhưng không chia hết cho 100 hoặc chia hết cho 400.

```
#include <iostream>
using namespace std;
main()
{
    int n;
    cout<<"Nhap vao mot nam bat ky : ";
    cin>>n;
    if (n % 4 == 0 && n % 100 !=0 || n % 400 == 0)
        cout<<n<<" la nam nhuan"<<endl;
    else
        cout<<n<<" la nam khong nhuan"<<endl;
}
```

Ví dụ 3: Viết chương trình giải phương trình bậc hai: $ax^2 + bx + c = 0$ ($a \neq 0$).

```
#include <iostream>
#include <math.h>
using namespace std;
main()
{ float a, b, c;
  cout<<"Nhap vao 3 he so a, b, c : ";
  cin>>a>>b>>c;
  float delta;
  delta = b*b - 4*a*c ;
  if (delta < 0) cout << "Phuong trinh vo nghiem"<<endl;
  else
    if (delta == 0)
```

```
      {
        cout<<"Phuong trinh co nghiem kep x = ";
        cout<<-b/(2*a)<<endl;
      }
    else
      { float x1, x2;
        x1=(-b+sqrt(delta))/(2*a);
        x2 = (-b-sqrt(delta))/(2*a);
        cout<<"Phuong trinh co 2 nghiem phan biet"<<endl;
        cout<<"x1 = " <<x1<<" va x2 = "<<x2<<endl;
      }
}
```

2.1.2 Cấu trúc switch

2.1.2.1 Ý nghĩa

Câu lệnh if cho ta khả năng được lựa chọn một trong hai nhánh để thực hiện, do đó nếu muốn rẽ theo nhiều nhánh ta phải sử dụng cấu trúc if lồng nhau. Tuy nhiên trong trường hợp như vậy chương trình sẽ phức tạp và khó đọc hơn, vì vậy C++ cung cấp thêm một câu lệnh cấu trúc khác cho phép chương trình có thể chọn một trong nhiều nhánh để thực hiện, đó là câu lệnh switch.

2.1.2.2 Cú pháp

```
switch (biểu thức điều khiển)
     {
          case biểu_thức_1: dãy lệnh 1;
          case biểu_thức_2: dãy lệnh 2;
          …
          case biểu_thức_n: dãy lệnh n;
          [default: dãy lệnh n+1;]
     }
```

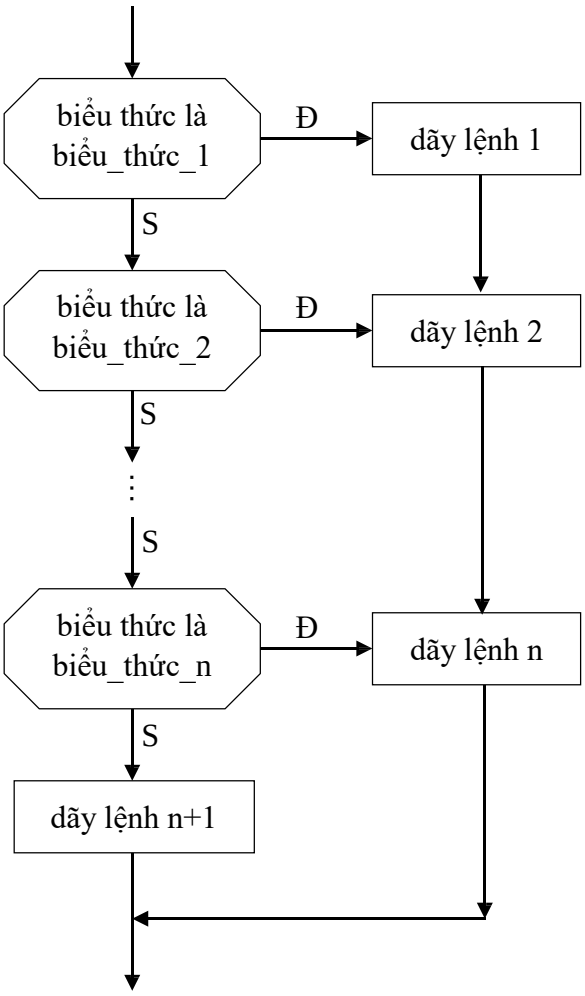
- Biểu thức điều khiển: phải có kiểu nguyên hoặc ký tự.

- Các biểu_thức_i: phải có kiểu nguyên hoặc ký tự tương ứng.

- Các dãy lệnh có thể rỗng và không cần bao dãy lệnh bởi cặp dấu ngoặc móc { }.

- Nhánh default có thể có hoặc không và vị trí của nó có thể nằm bất kỳ trong câu lệnh nào (giữa các nhánh case) nhưng thông thường người ta đặt nó nằm ở cuối cùng.

2.1.2.3 Lưu đồ cú pháp



Hình 2.2: Lưu đồ cú pháp cấu trúc switch

2.1.2.4 Cách thực hiện

Để thực hiện câu lệnh switch đầu tiên chương trình tính giá trị của biểu thức điều khiển (btđk), sau đó so sánh kết quả của btđk với giá trị của các biểu_thức_i bên dưới lần lượt từ biểu thức đầu tiên (thứ nhất) cho đến biểu thức cuối cùng (thứ n), nếu giá trị của btđk bằng giá trị của biểu thức thứ i đầu tiên nào đó thì chương trình sẽ thực hiện dãy lệnh thứ i và tiếp tục thực hiện tất cả dãy lệnh còn lại (từ dãy lệnh thứ i+1) cho đến hết (gặp dấu ngoặc đóng } của lệnh switch). Nếu quá trình so sánh không gặp biểu thức (nhánh case) nào bằng với giá trị của btđk thì chương trình thực hiện dãy lệnh trong default và tiếp tục cho đến hết (sau default có thể còn những nhánh case khác). Trường hợp câu lệnh switch không có nhánh default và btđk không khớp với bất kỳ nhánh case nào thì chương trình không làm gì, coi như đã thực hiện xong lệnh switch.

Nếu muốn lệnh switch chỉ thực hiện nhánh thứ i (khi btđk == biểu_thức_i) mà không phải thực hiện thêm các lệnh còn lại của các nhánh khác thì cuối dãy lệnh thứ i thông thường ta đặt thêm lệnh break; đây là lệnh cho phép thoát ra khỏi một lệnh cấu trúc bất kỳ.

2.1.2.5 Ví dụ minh họa

Ví dụ 1: Viết chương trình in ra số ngày của một tháng bất kỳ nào đó được nhập từ bàn phím, nếu tháng là 2 thì cho biết thêm năm nào.

```
#include <iostream>
using namespace std;
main()
{ int th, n;
  cout<<"Nhap vao mot thang bat ky : ";
  cin>>th;
  switch (th)
  {
      case 1: case 3: case 5: case 7: case 8: case 10:
      case 12: cout<<"Thang nay co 31 ngay"; break;
      case 4: case  6:
      case 9: case 11: cout<<"Thang nay co 30 ngay"; break;
      case 2:
        cout<<"Cho biet nam nao : ";
        cin>>n;
        if (n % 4 == 0 && n % 100 !=0 || n % 400 == 0)
          cout<<"Thang nay co 29 ngay";
        else cout<<"Thang nay co 28 ngay";
        break;
        default: cout<<"Ban da nhap so thang khong hop le";
    }
  cout<<endl;
}
```

Ví dụ 2: Viết chương trình nhập vào 2 số thực a và b từ bàn phím, sau đó nhập 1 ký tự thể hiện 1 trong 4 phép toán: cộng, trừ, nhân, chia. In ra kết quả thực hiện phép toán đó trên 2 số a, b.

```
#include <iostream>
#include <iomanip.h>
using namespace std;
main()
```

```
{ float a, b, kq; char pt;
  cout<<"Hay nhap 2 so thuc a, b: ";
  cin>>a>>b;
  cout<<"Hay nhap 1 phep toan: ";
  cin>>pt;
  switch (pt)
  {
      case '+': kq = a + b; break;
      case '-': kq = a - b; break;
      case 'x': case '.': case '*': kq = a * b ; break ;
      case ':': case '/':
        if (b != 0)
          {
            kq = a / b;
            break;
          }
        else
          {
            cout<<"Khong chia duoc";
            goto kt;
          }
      default:
        cout<<"Nhap phep toan sai"<<endl;
        goto kt;
  }
  cout<<"Ket qua la: "<<setprecision(6)<<kq<<endl;
  kt: system("PAUSE");
}
```

2.1.3 Lệnh nhảy goto

2.1.3.1 Ý nghĩa

Một dạng khác của rẽ nhánh là lệnh nhảy goto cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình. Nhãn là một tên gọi do NLT tự đặt theo các qui tắt đặt tên. Lệnh goto thường

được sử dụng để nhảy qua một số câu lệnh và thực hiện tiếp các câu lệnh sau đó trong chương trình, hoặc kết hợp với cấu trúc if để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này nên hạn chế sử dụng nếu không cần thiết.

2.1.3.2 Cú pháp

goto <nhãn>;

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng ngay sau nhãn và dấu hai chấm (:). Ví dụ 2 trong cấu trúc switch có sử dụng lệnh nhảy goto.

2.2 CẤU TRÚC LẶP

Một trong những cấu trúc quan trọng của lập trình cấu trúc là các câu lệnh cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình. Đó chính là các cấu trúc lặp.

2.2.1 Cấu trúc for

2.2.1.1 Cú pháp

```
for (dãy biểu thức 1; điều kiện lặp; dãy biểu thức 2)
     khối lệnh lặp;
```

- Các biểu thức trong các dãy biểu thức 1, 2 cách nhau bởi dấu phẩy (,). Có thể có nhiều biểu thức trong các dãy này hoặc dãy biểu thức cũng có thể trống.

- Điều kiện lặp: là biểu thức logic (có giá trị đúng hoặc sai).

✧ Các dãy biểu thức và điều kiện có thể trống tuy nhiên vẫn giữ lại các dấu chấm phẩy (;) để ngăn cách các thành phần với nhau.

2.2.1.2 Cách thực hiện

Khi gặp câu lệnh for trình tự thực hiện của chương trình như sau:

- Thực hiện dãy biểu thức 1 (thường là các lệnh khởi tạo giá trị cho các biến).

- Kiểm tra điều kiện lặp, nếu đúng thì thực hiện khối lệnh lặp → thực hiện dãy biểu thức 2 → quay lại kiểm tra điều kiện lặp và lặp lại quá trình trên cho đến khi việc kiểm tra điều kiện lặp cho kết quả sai thì dừng.

Tóm lại, dãy biểu thức 1 sẽ được thực hiện 1 lần duy nhất ngay từ đầu quá trình lặp sau đó thực hiện các câu lệnh trong khối lệnh lặp và dãy biểu thức 2 cho đến khi nào không còn thỏa điều kiện lặp nữa thì dừng.

Tương tự như cấu trúc if, nếu khối lệnh lặp có nhiều hơn 1 câu lệnh thì phải bao chúng trong cặp ngoặc móc { }.

2.2.1.3 Ví dụ minh họa

Ví dụ 1: Viết chương trình tính tổng N số nguyên dương đầu tiên, với N được nhập từ bàn phím.

Chương trình dùng một biến đếm i được khởi tạo từ 1, và một biến kq để chứa tổng.

Mỗi bước lặp chương trình sẽ cộng dồn i vào kq và sau đó tăng i lên 1 đơn vị. Chương trình còn lặp khi nào i còn chưa vượt quá N. Khi i lớn hơn N thì chương trình dừng.

```
#include <iostream>
using namespace std;
main()
{     int n, i;
      long kq = 0;
      cout<<"Nhap vao so nguyen n = ";
      cin>>n;
      for (i = 1; i <= n; i++)
          kq = kq + i; // Co the viet kq += i;
      cout<<"Tong "<<n<<" so nguyen duong dau tien la : ";
     cout<<kq<<endl;
}
```

Ví dụ 2: Viết chương trình tính tổng $S = \sum_{i=1}^{n} \frac{1}{i}$

```
#include <iostream>
using namespace std;
main()
{     int n, i;
      float kq = 0;
      cout<<"Nhap vao so nguyen n = ";
      cin>>n;
      for (i = 1; i <= n; i++)
          kq += 1.0/i; // Co the viet kq = kq + 1.0/i;
      cout<<"Tong S = "<<kq<<endl;
}
```

Ví dụ 3: Viết chương trình in ra màn hình dãy giảm dần các số lẻ bé hơn một số n nào đó được nhập vào từ bàn phím.

```
#include <iostream>
using namespace std;
main()
{     int n, i;
```

```
        cout<<"Nhap vao so nguyen n = ";
        cin>>n;
        if (n % 2 == 0) i = n - 1;
        else i = n - 2;
        for ( ; i >= 1; i = i - 2)
            cout<<i<<"    ";
        cout<<endl;
}
```

2.2.1.4 Đặc điểm

Thông qua phần giải thích cách hoạt động của câu lệnh for trong ví dụ 3 có thể thấy các thành phần của for có thể để trống, tuy nhiên các dấu chấm phẩy vẫn phải giữ lại để ngăn cách các thành phần với nhau.

2.2.1.5 Lệnh for lồng nhau

Trong khối lệnh lặp có thể chứa cả lệnh for, tức các lệnh for cũng được phép lồng nhau như các lệnh có cấu trúc khác.

Ví dụ 4: Bài toán cổ: vừa gà vừa chó, bó lại cho tròn, 36 con, 100 chân chẵn. Hỏi có mấy gà và mấy con chó.

Để giải bài toán này ta gọi g là số gà và c là số chó. Theo điều kiện bài toán ta thấy g có thể đi từ 0 (không có con nào) và đến tối đa là 50 (vì chỉ có 100 chân), tương tự c có thể đi từ 0 đến 25. Như vậy ta có thể cho g chạy từ 0 đến 50 và với mỗi giá trị cụ thể của g lại cho c chạy từ 0 đến 25, lần lượt với mỗi cặp (g, c) cụ thể đó ta kiểm tra 2 điều kiện: **g + c == 36?** (số con) và **2g + 4c == 100?** (số chân). Nếu cả 2 điều kiện đều thỏa thì cặp (g, c) cụ thể đó chính là nghiệm cần tìm. Từ đó ta có chương trình với 2 vòng for lồng nhau, một vòng for cho g và một vòng for cho c.

```
#include <iostream>
using namespace std;
main()
{     int g, c;
      for (g = 0; g <= 50; g++)
          for (c = 0; c <= 25; c++)
              if (g + c == 36 && 2 * g + 4 * c == 100)
                 cout<<"Ga = "<<g<<" Cho = "<<c<<endl;
}
```

Ví dụ 5: Viết chương trình tìm tất cả các phương án để đổi 100$ ra các tờ giấy bạc loại 10$, 20$ và 50$.

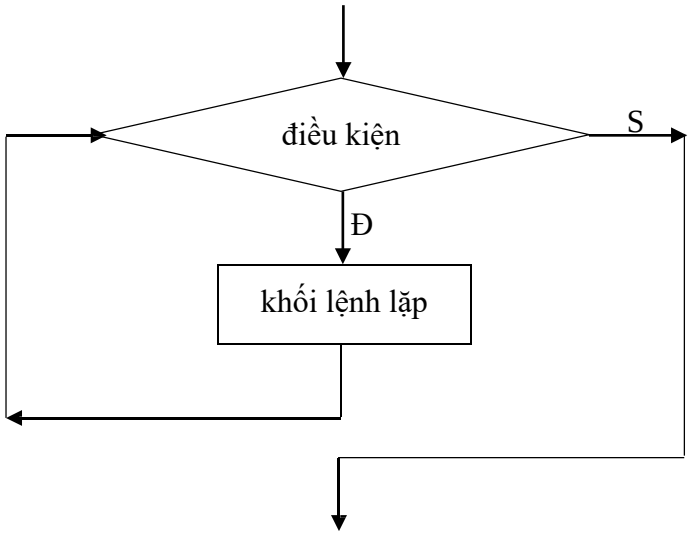
```
#include <iostream>
using namespace std;
main()
{ int st10, st20, st50;
  int sopa = 0;
  for (st10 = 0 ; st10 <= 10 ; st10++)
    for (st20 = 0 ; st20 <= 5 ; st20++)
      for (st50 = 0 ; st50 <= 2 ; st50++)
        if (st10 * 10 + st20 * 20 + st50 * 50 == 100)
        {   sopa++;
            cout<<"Phuong an "<<sopa<<" : ";
            if (st10) cout << st10 << " to 10$";
            if (st20) cout << " " << st20 << " to 20$";
            if (st50) cout << " " << st50 << " to 50$";
            cout << endl;
        }
  cout << "Tong so phuong an la : " << sopa <<endl;
}
```

2.2.2 Cấu trúc while

2.2.2.1 Cú pháp

```
while (điều kiện)
     khối lệnh lặp;
```

2.2.2.2 Lưu đồ cú pháp



Hình 2.3: Lưu đồ cú pháp cấu trúc while

2.2.2.3 Thực hiện

Khi gặp lệnh while chương trình thực hiện như sau: đầu tiên chương trình sẽ kiểm tra điều kiện, nếu đúng thì thực hiện khối lệnh lặp, sau đó quay lại kiểm tra điều kiện và tiếp tục, nếu điều kiện sai thì dừng vòng lặp. Tóm lại có thể mô tả một cách ngắn gọn về câu lệnh while như sau: lặp lại khối lệnh trong khi điều kiện vẫn còn đúng.

2.2.2.4 Đặc điểm

- Khối lệnh lặp có thể không được thực hiện lần nào nếu điều kiện sai ngay từ đầu.

- Để vòng lặp không lặp vô tận thì trong khối lệnh thông thường phải có ít nhất một câu lệnh nào đó gây ảnh hưởng đến kết quả của điều kiện, tức là làm cho điều kiện đang đúng trở thành sai.

- Nếu điều kiện luôn luôn nhận giá trị đúng (ví dụ biểu thức điều kiện là 1) thì trong khối lệnh lặp phải có câu lệnh kiểm tra dừng và lệnh break.

- Nếu khối lệnh lặp có nhiều hơn 1 câu lệnh thì phải bao chúng trong cặp ngoặc móc { }.

2.2.2.5 Ví dụ minh họa

Ví dụ 1: Bài toán cổ: vừa gà vừa chó, bó lại cho tròn, 36 con, 100 chân chẵn. Hỏi có mấy gà và mấy con chó.

```
#include <iostream>
using namespace std;
main()
{ int c, g = 0;
  while (g <= 36) //số gà tối đa là 50 nhưng đề chỉ cho 36
  {
      c = 0;
      while (c <= 25) //số chó chỉ tối đa 25 là đủ 100 chân
      {
          if (g + c == 36 && 2 * g + 4 * c == 100)
            cout<<"Ga = "<<g<<" Cho = "<<c<<endl;
          c++;
      }
      g++;
  }
}
```

Ví dụ 2: Viết chương trình nhập vào 2 số nguyên dương (giả sử m và n), in ra ước chung lớn nhất (UCLN) của 2 số này.

Áp dụng thuật toán Euclide bằng cách liên tiếp lấy số lớn trừ đi số nhỏ khi nào 2 số bằng nhau thì đó là UCLN. Trong chương trình ta qui ước m là số lớn và n là số nhỏ. Thêm biến phụ r để tính hiệu của 2 số. Sau đó đặt lại m hoặc n bằng r sao cho m > n và lặp lại. Vòng lặp dừng khi m = n.

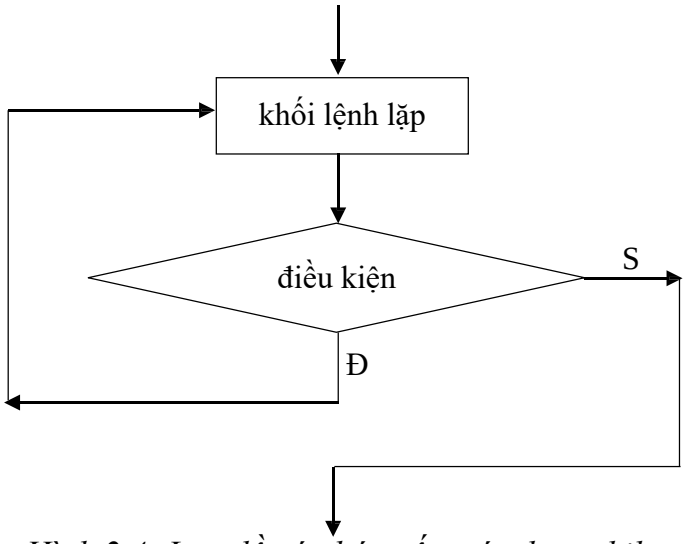
```
#include <iostream>
using namespace std;
main()
{   int m, n, r;
    cout<<"Nhap m, n: " ;
    cin>>m>>n ;
    if (m < n)        // Nếu m < n thì đổi vai trò hai số
    {
        int t = m; m = n; n = t;
    }
    while (m != n)
    {
        r = m - n;
        if (r > n)
              m = r;
        else
        {
              m = n;
              n = r;
        }
    }
    cout<<"UCLN = "<<m<<endl;
}
```

2.2.3 Cấu trúc do ... while

2.2.3.1 Cú pháp

```
do
    khối lệnh lặp;
while (điều kiện);
```

2.2.3.2 Lưu đồ cú pháp



Hình 2.4: Lưu đồ cú pháp cấu trúc do…while

2.2.3.3 Thực hiện

Đầu tiên chương trình sẽ thực hiện khối lệnh lặp, tiếp theo kiểm tra điều kiện, nếu điều kiện còn đúng thì quay lại thực hiện khối lệnh và quá trình tiếp tục cho đến khi điều kiện trở thành sai thì dừng.

2.2.3.4 Đặc điểm

Các đặc điểm của câu lệnh do … while cũng giống với câu lệnh lặp while trừ điểm khác biệt, đó là khối lệnh trong do … while sẽ được thực hiện ít nhất một lần vì điều kiện được kiểm tra sau, trong khi khối lệnh trong while có thể không được thực hiện lần nào.

2.2.3.5 Ví dụ minh họa

Ví dụ 1: Viết chương trình kiểm tra một số nguyên n (n > 1) có phải là số nguyên tố không, với n được nhập từ bàn phím.

Để kiểm tra một số n > 1 có phải là số nguyên tố hay không ta lần lượt chia n cho các số i đi từ 2 đến một nửa của n. Nếu có i sao cho n chia hết cho i thì n là hợp số (không phải là số nguyên tố) ngược lại n là số nguyên tố.

```
#include <iostream>
using namespace std;
main()
{   int i, n;
    cout<<"Nhap so n can kiem tra: ";
    cin>>n;
    i = 2;
    if(n > 2)
```

```
        do
        {
            if (n % i == 0)
            {
                cout<<n<<" la hop so";
                goto kt; break;
            }
            i++;
        } while (i <= n/2);
    cout<<n<<" la so nguyen to";
    kt: system("PAUSE");
}
```

Chương trình trên có thể viết bằng cách khác như sau:

```
#include <iostream>
using namespace std;
main()
{   int n, i=1;
    cout<<"Nhap so n can kiem tra: ";
    cin>>n;
    do
    {
         i++;
    } while (i <= n/2 && n % i != 0);
    if (i > n/2)
        cout<<n<<" la so nguyen to"<<endl;
    else
        cout<<n<<" la hop so"<<endl;
}
```

Ví dụ 2: Viết chương trình nhập vào một dãy ký tự rồi thống kê các loại chữ hoa, chữ thường, chữ số và các ký tự khác đến khi gặp dấu chấm câu thì dừng.

```
#include <iostream>
using namespace std;
```

```
main()
{   char kt;
    int ch, ct, cs, ck ;
    ch = ct = cs = ck = 0;
    cout << "Hay nhap day ky tu:"<<endl;
    do
    {   cin>>kt;
        if ('A' <= kt && kt <= 'Z') ch++;
        else if ('a' <= kt && kt <= 'z') ct++;
                 else if ('0' <= kt && kt <= '9') cs++;
                       else ck++;
    } while (kt != 46);
    cout<<"Trong day co:"<<endl;
    cout<<ch<<" chu hoa"<<endl;
    cout<<ct<<" chu thuong"<<endl;
    cout<<cs<<" chu so"<<endl;
    cout<<ck<<" ky tu khac"<<endl;
}
```

2.2.4 Lối ra của vòng lặp

2.2.4.1 Lệnh break

Lệnh break có thể xuất hiện bên trong vòng lặp (for, while hay do … while) hoặc trong lệnh switch. Nó tạo nên bước nhảy ra bên ngoài những lệnh này và vì thế kết thúc chúng. Lệnh break có tác dụng cho vòng lặp hoặc lệnh switch gần nó nhất. Sử dụng lệnh break bên ngoài vòng lặp hay lệnh switch sẽ gây ra lỗi.

Ví dụ: Đoạn chương trình sau dùng để kiểm tra việc nhập password là đúng hay sai và chỉ được phép nhập sai tối đa n lần.

```
for (i = 1; i <= n; i++)
{
    cout << "Nhap vao password: ";
    cin>> password;
    if (Kiemtra(password)) // Nếu password đúng
        break;                // Thoát khỏi vòng lặp
    cout<< "Password sai!"<<endl;
}
```

Ở đây chúng ta phải giả sử rằng có một hàm được gọi ***Kiemtra*** dùng để kiểm tra một password và trả về true nếu như password đúng và ngược lại là false.

Chúng ta có thể viết lại vòng lặp mà không cần lệnh break bằng cách sử dụng thêm một biến luận lý (kt) và thêm nó vào điều kiện lặp:

```
kt = 0;
for (i = 1; i < = n && !kt; i++)
{
    cout << "Nhap vao password: ";
    cin>> password;
    kt = Kiemtra(password));
    if (!kt) cout<< " Password sai!"<<endl;
}
```

✧ Người ta cho rằng phiên bản có sử dụng lệnh break đơn giản hơn nên thường được ưa chuộng hơn.

2.2.4.2 Lệnh continue

Lệnh continue dừng lần lặp hiện tại của một vòng lặp và nhảy tới lần lặp kế tiếp. Nó áp dụng tức thì cho vòng lặp gần với lệnh continue nhất. Sử dụng lệnh continue bên ngoài vòng lặp sẽ gây ra lỗi.

Trong vòng lặp while và vòng lặp do … while, vòng lặp kế tiếp mở đầu từ điều kiện lặp. Trong vòng lặp for, lần lặp kế tiếp bắt đầu từ dãy biểu thức 2 của vòng lặp.

Ví dụ: Một vòng lặp thực hiện nhập một số, xử lý nó nhưng bỏ qua những số âm, và dừng khi số là 0, có thể diễn giải như sau:

```
do
{
    cin>>num;
    if (num < 0) continue;
    // xử lý số ở đây ...
} while (num != 0);
```

Điều này tương đương với:

```
do
{   cin >> num;
    if (num >= 0)
        // xử lý số ở đây ...
} while (num != 0);
```

Một biến thể của vòng lặp này để đọc chính xác một số *n* lần (hơn là cho tới khi số đó là 0) có thể được diễn giải như sau:

```
for (i = 1; i <= n; i++)
{
    cin>>num;
    if (num < 0) continue; // làm cho nhảy tới i++
    // xử lý số ở đây ...
}
```

Khi lệnh continue xuất hiện bên trong vòng lặp được lồng vào thì nó áp dụng trực tiếp lên vòng lặp gần nó chứ không áp dụng cho vòng lặp bên ngoài. Ví dụ, trong một tập các vòng lặp được lồng nhau sau đây, lệnh continue áp dụng cho vòng lặp for và không áp dụng cho vòng lặp while.

```
while (more)
{
    for (i = 1; i <= n; i++)
    {
        cin >> num;
        if (num < 0) continue;    //làm cho nhảy tới i++
        // xử lý số ở đây ...
    }
    // Các câu lệnh của while
}
```

BÀI TẬP CHƯƠNG 2

-----ଓ✧ଓ-----

1. Viết chương trình nhập một ký tự, cho biết ký tự đó có phải là chữ cái hay không.

2. Viết chương trình nhập vào một số nguyên, hãy cho biết số đó là âm hay dương, chẵn hay lẻ.

3. Viết chương trình nhập 3 số a, b, c in ra max, min của 3 số đó.

4. Viết chương trình nhập 3 số a, b, c, hãy cho biết 3 số trên có thể là độ dài 3 cạnh của một tam giác.

5. Viết chương trình nhập vào điểm cơ bản (đcb) và điểm nâng cao (đnc) cho 1 học viên. Cho biết học viên này được xếp loại gì, với cách xếp loại dựa vào điểm trung bình (đtb) như sau:

- Nếu đtb >= 9 và không có điểm nào dưới 8 thì được xếp loại xuất sắc
- Nếu đtb >= 8 và không có điểm nào dưới 7 thì được xếp loại giỏi
- Nếu đtb >= 7 và không có điểm nào dưới 6 thì được xếp loại khá
- Nếu đtb >= 5 và không có điểm nào dưới 3 thì được xếp loại trung bình
- Còn lại thì ghi không đạt

6. Viết chương trình làm việc như 1 máy tính bỏ túi:

- Nhập vào 2 số
- Hỏi toán tử là +, -, * hay /, tương ứng in ra tổng, hiệu, tích, thương.
- Nếu không phải là các toán tử trên thì kết thúc chương trình.

7. Viết chương trình tính tích N số nguyên dương đầu tiên, với N được nhập từ bàn phím.

8. Viết chương trình tính tổng S = 1/2 + 2/3 + 3/4 + … + n/(n+1), với n là số nguyên dương được nhập từ bàn phím.

9. Viết chương trình tính tổng S = 1 + 2 - 3 + 4 + 5 - 6 + 7 + 8 - 9 + … +/- n, với n là số nguyên dương được nhập từ bàn phím.

10. Viết chương trình tính tổng S = $\sum_{i=1}^{n}\sum_{j=1}^{m} i+2j$, với n, m là số nguyên dương được nhập từ bàn phím.

11. Viết chương trình tính $e = 1 + \frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + ... + (-1)^{n+1}\frac{x^n}{n!}$, với n là số nguyên dương, x là số thực được nhập từ bàn phím.

12. Viết chương trình tính e = 1 + 1/1! + 1/2! + 1/3! + … + 1/n!, cho đến khi 1/(n+1)! nhỏ hơn Epsilon, với Epsilon là 1 lượng khá nhỏ được nhập từ bàn phím.

13. Viết chương trình nhập vào số nguyên dương n.

In ra tam giác có dạng sau:

@
@ @
@ @ @
@ @ @ @

(Nếu n nhập vào là 4)

14. Viết chương trình nhập vào chiều dài và chiều rộng cho 1 hình chữ nhật. In ra hình chữ nhật đó bằng các dấu *. Giả sử ta nhập cd = 6, cr = 4 thì in:

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

15. Viết chương trình nhập vào lần lượt các số nguyên, quá trình nhập kết thúc khi nhập số nguyên là 0. Cho biết có bao nhiêu số dương, bao nhiêu số âm trong các số nguyên đã nhập.

16. Viết chương trình nhập vào lần lượt các số thực, quá trình nhập kết thúc khi nhập số thực là 0. Cho biết trị trung bình của các số đã nhập.

17. Viết chương trình làm nhiều lần công việc sau: Nhập 1 số thực, in số đối của số thực đó. Chương trình kết thúc khi nào số thực nhập vào là 0.

18. Viết chương trình in ra tất cả các số nguyên tố < N, N là trị nhập.

19. Viết chương trình nhập vào 1 số nguyên dương, in ra ước số lẻ lớn nhất của nó. Ví dụ ta nhập số 60 thì in ra là 15.

20. Viết chương trình tính số hạng thứ n của dãy Fibonacci. Hướng dẫn: Dãy Fibonacci là dãy số gồm các số hạng F(n) vói F(n) = F(n - 1) + F(n - 2) và F(1) = F(2) = 1

Ví dụ: Dãy Fibonacci gồm 10 số hạng đầu tiên là: 1 1 2 3 5 8 13 21 34

21. Viết chương trình phân tích 1 số nguyên dương n thành tích của các thừa số nguyên tố.

22. Viết lại chương trình của bài 8 bằng lệnh nhảy goto.

23. Viết lại chương trình của bài 12 bằng lệnh nhảy goto.

24. Viết lại chương trình của bài 16 bằng lệnh nhảy goto.

CHƯƠNG 3: DỮ LIỆU KIỂU MẢNG

-----ଓ✧ଓ-----

3.1 ĐỊNH NGHĨA VÀ ỨNG DỤNG

3.1.1 Định nghĩa

Mảng là một tập hợp các phần tử có cùng kiểu dữ liệu gọi là kiểu phần tử. Kiểu phần tử có thể là các kiểu cơ bản, kiểu cấu trúc (struct), … Các phần tử của mảng được bố trí trong bộ nhớ là một dãy các ô nhớ liên tục nhau nên mảng còn có thể được gọi là dãy. Người ta thường chia mảng thành 2 loại mảng là mảng một chiều và mảng nhiều chiều (2 chiều trở lên).

3.1.2 Ứng dụng

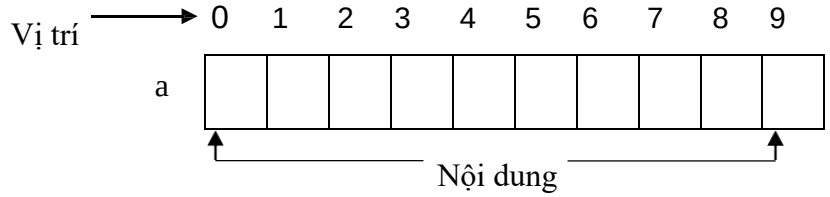
Sử dụng mảng nhằm tránh đi việc sử dụng quá nhiều biến đơn. Trong thực tế mảng thường dùng để biểu diễn danh sách các phần tử giống nhau, chẳng hạn như danh sách sinh viên, danh sách khách hàng, … Danh sách được lưu trữ như mảng được gọi là danh sách đặc.

3.2 MẢNG MỘT CHIỀU

3.2.1 Định nghĩa

Mảng một chiều hay véc-tơ là một dãy các phần tử có cùng kiểu được sắp kề nhau liên tục trong bộ nhớ. Tất cả các phần tử đều có cùng tên là tên của mảng. Để phân biệt các phần tử với nhau, các phần tử sẽ được đánh số thứ tự từ 0 cho đến hết mảng. Khi cần nói đến phần tử cụ thể nào đó của mảng ta sẽ dùng tên mảng và kèm theo vị trí/chỉ số của phần tử đó.

Hình 3.1 minh họa hình thức mảng 1 chiều gồm có 10 phần tử, các thành phần được đánh số từ 0 đến 9.



Hình 3.1: Minh họa hình thức mảng 1 chiều gồm 10 phần tử

3.2.2 Khai báo

<tên kiểu> <tên mảng>[số phần tử] ; *// không khởi tạo*

<tên kiểu> <tên mảng>[số phần tử] = { dãy giá trị }; *// có khởi tạo*

<tên kiểu> <tên mảng>[ ] = { dãy giá trị }; *// có khởi tạo*

- Tên kiểu là kiểu dữ liệu của các phần tử, các phần tử này có kiểu giống nhau. Thỉnh thoảng ta cũng gọi các phần tử là các thành phần.

- Cách khai báo trên giống như khai báo biến bình thường nhưng thêm số phần tử của mảng giữa cặp dấu ngoặc vuông [ ], còn được gọi là kích thước của mảng.

- Mỗi tên mảng là một biến và để phân biệt với các biến thông thường ta gọi là biến mảng.

- Dữ liệu hay nội dung các phần tử của 1 mảng được lưu trong bộ nhớ bởi 1 dãy các ô nhớ liên tiếp nhau. Số lượng ô nhớ bằng với số phần tử của mảng và kích thước (được tính bằng byte) của mỗi ô nhớ đủ để chứa thông tin của mỗi phần tử. Ô đầu tiên được đánh thứ tự là 0, ô tiếp theo là 1, ô kế tiếp là 2 và tiếp tục cho đến hết. Như vậy nếu mảng có **N** phần tử thì ô cuối cùng trong mảng sẽ được đánh số là **N - 1**.

- Dạng khai báo thứ 2 cho phép khởi tạo mảng bởi dãy giá trị trong cặp dấu { }, mỗi giá trị cách nhau bởi dấu phẩy (,), các giá trị này sẽ được gán lần lượt cho các phần tử của mảng bắt đầu từ phần tử thứ 0 cho đến hết dãy. Số giá trị có thể bé hơn số phần tử. Các phần tử trong mảng chưa có giá trị sẽ không được xác định cho đến khi trong chương trình nó được gán một giá trị nào đó.

- Dạng khai báo thứ 3 cho phép vắng mặt số phần tử, trường hợp này số phần tử được xác định bởi số giá trị của dãy khởi tạo. Do đó nếu vắng mặt cả dãy khởi tạo là không được phép (chẳng hạn khai báo int a[ ] là sai).

Ví dụ:

- Khai báo 2 biến chứa véc-tơ a, b trong không gian 3 chiều:

```
float a[3], b[3];
```

- Khai báo 3 phân số a, b, c; trong đó a = 1/3 và b = 3/5:

```
int a[2] = {1, 3} , b[2] = {3, 5} , c[2];
```

Ở đây ta ngầm qui ước phần tử đầu tiên (số thứ tự 0) là tử số và phần tử thứ hai (số thứ tự 1) là mẫu số của phân số.

- Khai báo mảng L chứa được tối đa 100 số nguyên dài:

```
long L[100];
```

- Khai báo mảng dong (hàng), mỗi dòng chứa được tối đa 80 ký tự:

```
char dong[80];
```

3.2.3 Cách sử dụng

- Để truy xuất đến phần tử thứ i (hay *vị trí* i hoặc *chỉ số* i) của 1 mảng ta viết tên mảng kèm theo *vị trí/chỉ số* đặt trong cặp ngoặc vuông [ ].

Ví dụ với các phân số trên a[0], b[0], c[0] để chỉ tử số và a[1], b[1], c[1] để chỉ mẫu số của 3 phân số a, b, c.

- Mặc dù mỗi mảng biểu diễn một đối tượng nhưng chúng ta không thể áp dụng các thao tác lên toàn bộ mảng mà phải thực hiện thao tác thông qua từng phần tử của mảng.

Ví dụ chúng ta không thể nhập dữ liệu cho mảng a[10] bằng câu lệnh:

```
cin >> a ;       //sai
```

Để nhập dữ liệu cho mảng a[10] ta phải nhập cho từng phần tử từ a[0] đến a[9] của a.

Dĩ nhiên trong trường hợp này ta phải cần đến lệnh lặp for:

```
int i;
for (i = 0 ; i < 10 ; i++)
      cin >> a[i];
```

Tương tự, giả sử chúng ta cần cộng 2 phân số a, b và đặt kết quả vào c.

Ta không thể viết:

```
c = a + b ;    //sai
```

Ta cần phải tính từng thành phần cho c như sau:

```
c[0] = a[0] * b[1] + a[1] * b[0] ;    //tử số
c[1] = a[1] * b[1] ;    //mẫu số
```

3.2.4 Các ví dụ minh họa

Ví dụ 1: Viết chương trình nhập vào 2 phân số, tìm tổng và tích của 2 phân số đó.

```
#include <iostream>
using namespace std;
main()
{   int a[2], b[2], tong[2], tich[2]; cout<<"Nhap
    tu cua phan so a = "; cin>>a[0]; cout<<"Nhap
    mau cua phan so a = "; cin>>a[1]; cout<<"Nhap
    tu cua phan so b = "; cin>>b[0]; cout<<"Nhap
    mau cua phan so b = "; cin>>b[1]; tong[0] =
    a[0] * b[1] + a[1] * b[0];
    tong[1] = a[1] * b[1];
    tich[0] = a[0] * b[0];
    tich[1] = a[1] * b[1];
    cout<<"Phan so Tong = "<<tong[0]<<'/'<<tong[1]<<endl;
    cout<<"Phan so Tich = "<<tich[0]<<'/'<<tich[1]<<endl;
}
```

Ví dụ 2: Viết chương trình nhập vào mảng 1 chiều gồm n phần tử kiểu số thực, in ra mảng vừa nhập, thống kê số phần tử dương, âm và bằng không có trong mảng.

```
#include <iostream>
using namespace std;
main()
```

```
{     float a[50];
      int i, n, sd, sa, s0;
      cout<<"Nhap so phan tu cua mang: ";
      cin>>n;
      for (i = 0; i < n; i++)
      {
         cout<<"a["<<i<<"] = ";
         cin>>a[i];
      }
      cout<<"Mang vua nhap la:"<<endl;
      for (i = 0; i < n; i++)
         cout<<a[i]<<'\t';
      cout<<endl;
      sd = sa = s0 = 0;
      for (i = 0; i < n; i++)
         if (a[i] > 0 ) sd++;
         else if (a[i] < 0 ) sa++;
              else s0++;
      cout << "So so duong = "<<sd<<endl;
      cout << "So so am = "<<sa<<endl;
      cout << "So so khong = "<<s0<<endl;
}
```

Ví dụ 3: Viết chương trình nhập vào mảng 1 chiều gồm n phần tử kiểu số nguyên. In ra mảng vừa nhập. Cho biết có bao nhiêu phần tử có nội dung là lẻ. Tính tổng các phần tử là bội số của k, với k được nhập từ bàn phím. Cho biết phần tử x xuất hiện đầu tiên ở vị trí thứ mấy, với x được nhập từ bàn phím. Sắp xếp mảng theo thứ tự tăng dần. In ra mảng sau khi sắp xếp.

```
#include <iostream>
using namespace std;
main()
{ int a[20];
      int n, i, j, d, s, k, x, tam;
      //Nhap va kiem tra so phan tu cua mang
```

```
do
{
    cout<<"Nhap so phan tu n = ";
    cin>>n;
} while (n <= 0 || n > 20);
//Nhap mang
for (i = 0; i < n; i++)
{
  cout<<"Nhap phan tu thu "<<i+1<<" : ";
  cin>>a[i];
}
//In mang vua nhap
cout<<"Mang vua nhap la :"<<endl;
for (i = 0; i < n; i++)
  cout<<a[i]<<'\t';
cout<<endl;
//Dem so phan tu co noi dung la le
d = 0;
for (i = 0; i < n; i++)
   if(a[i]%2 != 0) d++;
cout<<"Co "<<d<<" phan tu co noi dung la le"<<endl;
//Tinh tong cac phan tu la boi so cua k
cout<<"Nhap k = "; cin>>k;
s = 0;
for (i = 0; i < n; i++)
  if (a[i]%k == 0)
    s = s + a[i];
cout<<"Tong cac phan tu la boi so cua "<<k;
cout<<" la : "<<s<<endl;
//Xac dinh vi tri xuat hien dau tien cua x
cout<<"Nhap x : "; cin>>x;
```

```
      i = 0;
      while (i < n && a[i]!= x)
        i++;
      if (i < n)
      {
          cout<<"Phan tu "<<x<<" xuat hien dau tien";
          cout<<" o vi tri thu "<<i<<endl;
      }
      else
        cout<<"Khong co phan tu "<<x<<" trong mang"<<endl;
      //Sap xep mang theo thu tu tang dan
      for (i = 0; i < n-1; i++)
          for (j = i+1; j < n; j++)
               if (a[i] > a[j])
                    {
                         tam = a[i];
                         a[i] = a[j];
                         a[j] = tam;
                    }
      //In mang sau khi sap xep
      cout<<"Mang sau khi sap xep tang dan la :"<<endl;
      for (i = 0; i < n; i++)
        cout<<a[i]<<'\t';
      cout<<endl;
}
```

Ví dụ 4: Viết chương trình nhập vào mảng 1 chiều gồm n phần tử kiểu số thực. In ra mảng vừa nhập. Thêm 1 phần tử x vào tại vị trí vt, với x và vt nhập từ bàn phím. Cho biết phần tử lớn nhất xuất hiện sau cùng ở vị trí thứ mấy. Tính tích các phần tử dương nằm ở các vị trí lẻ. Xóa tất cả các phần tử có nội dung là y, với y nhập từ bàn phím.

```
#include <iostream>
using namespace std;
main()
{
```

```
float a[20], x, y, max, tich;
int n, i, j, vt;
//Nhap va kiem tra so phan tu cua mang
do
{     cout<<"Nhap so phan tu n = ";
      cin>>n;
} while (n <= 0 || n > 20);
//Nhap mang
for (i = 0; i < n; i++)
{
    cout<<"Nhap phan tu thu "<<i+1<<" : ";
    cin>>a[i];
}
//In mang vua nhap
cout<<"Mang vua nhap la :"<<endl;
for (i = 0; i < n; i++)
    cout<<a[i]<<'\t';
cout<<endl;
//Them phan tu vao vi tri vt
cout<<"Nhap phan tu can them : "; cin>>x;
do
{     cout<<"Nhap vi tri can them : ";
      cin>>vt;
} while (vt < 0 || vt > n);
for (i = n - 1; i >= vt; i--)
      a[i+1] = a[i];
a[vt] = x;
n = n + 1;
cout<<"Mang vua them la : "<<endl;
for (i = 0; i < n; i++)
      cout<<a[i]<<'\t';
cout<<endl;
```

```
    //Tim phan tu lon nhat xuat hien sau cung
    max = a[0];
    for (i = 1; i < n; i++)
         if (max < a[i]) max = a[i];
    i = n - 1;
    while (a[i] != max)
         i--;
    cout<<"Vi tri xuat hien sau cung cua ";
    cout<<"phan tu lon nhat la : "<<i<<endl;
    //Tinh tich cac phan tu duong o vi tri le
    tich = 1;
    for (i = 0; i < n; i++)
         if (i % 2 != 0 && a[i] > 0)
              tich = tich * a[i];
    cout<<"Tich cac phan tu duong nam o cac vi tri le ";
    cout<<"la : "<<tich<<endl;
    //Xoa tat ca cac phan tu co noi dung la y
    cout<<"Nhap phan tu can xoa : "; cin>>y;
    i = 0;
    while (i <= n - 1)
         if (a[i] == y)
         {
              for (j = i; j <= n - 2; j++)
                   a[j] = a[j + 1];
               n--;
         }
         else i = i + 1;
    cout<<"Mang sau khi xoa la : "<<endl;
    for (i = 0; i < n; i++)
        cout<<a[i]<<'\t';
    cout<<endl;
}
```

3.3 MẢNG HAI CHIỀU

3.3.1 Giới thiệu

Để thuận tiện trong việc biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu có nhiều chỉ tiêu, C++ đưa ra kiểu dữ liệu mảng nhiều chiều. Đối với mảng một chiều m phần tử, nếu mỗi phần tử của nó lại là mảng một chiều n phần tử thì ta gọi mảng là hai chiều với số phần tử hay kích thước của nó là m * n. Và nếu mỗi phần tử của chiều thứ 2 là mảng một chiều k phần tử thì ta gọi mảng là ba chiều với số phần tử hay kích thước của nó là m * n * k.

Như vậy tùy theo bài toán thực tế mà người lập trình có thể khai báo mảng với N chiều cho hợp lý. Tuy nhiên, việc sử dụng mảng càng nhiều chiều thì độ phức tạp sẽ càng tăng vì vậy trong giáo trình này chúng ta chỉ nghiên cứu mảng nhiều chiều đơn giản nhất chính là mảng hai chiều.

3.3.2 Định nghĩa

Mảng hai chiều là một mảng một chiều mà mỗi phần tử của nó là một mảng một chiều khác. Tất cả các phần tử đều có cùng tên là tên của mảng. Khi cần nói đến phần tử cụ thể nào đó của mảng ta sẽ dùng tên mảng và kèm theo vị trí/chỉ số của 2 chiều tương ứng của phần tử đó.

Ma trận là một minh họa cho hình ảnh của mảng hai chiều, nó gồm m hàng và n cột, tức là chứa m * n phần tử và hiển nhiên các phần tử này có cùng kiểu dữ liệu. Vì thế mảng hai chiều còn được gọi là ma trận.

Tuy nhiên, về mặt bản chất mảng hai chiều không phải là một tập hợp với m * n phần tử cùng kiểu mà là tập hợp với m thành phần, trong đó mỗi thành phần là một mảng một chiều với n phần tử.

	0	1	2	3
0				
1				
2				

Hình 3.2: Minh họa hình thức mảng 2 chiều có 3 hàng, 4 cột

Thực chất trong bộ nhớ tất cả 12 phần tử của mảng được sắp liên tiếp theo từng hàng của mảng như *hình 3.3*.

hàng 0 *hàng 1* *hàng 2*

0	**1**	**2**	**3**	4	5	6	7	8	9	10	11

Hình 3.3: Minh họa cách lưu trữ mảng 2 chiều có 3 hàng, 4 cột

3.3.3 Khai báo

<kiểu phần tử> <tên mảng> [m][n];

- m là số hàng, n số cột của mảng.

- kiểu phần tử là kiểu của m x n phần tử trong mảng.

- Trong khai báo cũng có thể được khởi tạo bằng dãy các hàng giá trị, các hàng cách nhau bởi dấu phẩy, mỗi hàng được bao bởi cặp ngoặc { } và toàn bộ giá trị khởi tạo nằm trong cặp dấu { }.

3.3.4 Cách sử dụng

- Tương tự mảng một chiều các chiều trong mảng cũng được đánh số từ 0.

- Không sử dụng các thao tác trên toàn bộ mảng mà phải thực hiện thông qua từng phần tử của mảng.

- Để truy nhập phần tử của mảng ta sử dụng tên mảng kèm theo 2 chỉ số chỉ vị trí hàng và cột của phần tử. Các chỉ số này có thể là các biểu thức thực nhưng khi đó C++ sẽ tự chuyển kiểu sang nguyên.

Ví dụ:

- Khai báo 2 ma trận A, B gồm 3 hàng 4 cột chứa các số nguyên:

```
int A[3][4], B[3][4];
```

- Khai báo có khởi tạo:

```
int A[3][4] = {{1, 2, 3, 4},{3, 2, 1, 4},{0, 1, 1, 0}};
```

với khởi tạo này ta có ma trận:

	0	1	2	3
0	**1**	**2**	**3**	**4**
1	**3**	**2**	**1**	**4**
2	**0**	**1**	**1**	**0**

Hình 3.4: Minh họa mảng 2 chiều có 3 hàng, 4 cột đã được khởi tạo các giá trị

Trong đó: A[0][0] = 1, A[0][1] = 2, A[1][0] = 3, A[2][3] = 0, …

Trong khai báo có thể vắng số hàng (không được vắng số cột), số hàng này được xác định thông qua khởi tạo.

Ví dụ: `float A[][3] = {{1, 2, 3}, {0, 1, 0}};` trong khai báo này chương trình tự động xác định số hàng là 2.

- Phép khai báo và khởi tạo sau đây cũng hợp lệ:

Ví dụ: `float A[][3] = {{1, 2}, {0}};`

Với khai báo trên NNLT cũng xác định được số hàng là 2 và số cột (bắt buộc phải khai báo) là 3 mặc dù trong khởi tạo không thể xác định được số cột. Các phần tử chưa khởi tạo sẽ chưa được xác định cho đến khi nào nó được nhập hoặc gán giá trị cụ thể. Trong ví dụ trên các phần tử A[0][2], A[1][1] và A[1][2] là chưa được xác định.

3.3.5 Các ví dụ minh họa

Ví dụ 1: Viết chương trình nhập vào ma trận m hàng, n cột, các phần tử kiểu số thực, in ra ma trận vừa nhập, cho biết giá trị nhỏ nhất và lớn nhất của các phần tử trong ma trận.

```
#include <iostream>
using namespace std;
main()
{     float a[10][10], max, min;
      int m, n, i, j;
      cout<<"Nhap so hang va so cot: ";
      cin>>m>>n;
      //Nhap ma tran
      for (i = 0; i < m; i++)
         for (j = 0; j < n; j++)
         {
              cout<<"a["<<i<<","<<j<<"] = ";
              cin>>a[i][j];
         }
      //In ma tran
      for (i = 0; i < m; i++)
      {
           for (j = 0; j < n; j++)
              cout<<a[i][j]<<'\t';
           cout<<endl;
      }
      //Tim max va min
      max = a[0][0];
      min = a[0][0];
```

```
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
            {
                if (max < a[i][j]) max = a[i][j];
                if (min > a[i][j]) min = a[i][j];
            }
        cout<<"Phan tu lon nhat la "<<max<<'\n';
        cout<<"Phan tu nho nhat la "<<min<<'\n';
}
```

Ví dụ 2: Viết chương trình nhập vào ma trận a gồm m hàng, n cột, các phần tử kiểu số nguyên. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu phần tử là số chính phương. Tính tích các phần tử khác 0 nằm trên cột c, với c nhập từ bàn phím. Sắp xếp các phần tử nằm trên hàng h theo thứ tự giảm dần, với h nhập từ bàn phím. In ra ma trận sau khi sắp xếp.

```
#include <iostream>
#include <math.h>
using namespace std;
main()
{    int a[10][10], m, n, i, j, d, c, h, tam;
     long t;
     //Nhap va kiem tra so hang, so cot
     do
      {
          cout<<"Nhap so hang va so cot : ";
          cin>>m>>n;
      }while (m<0 || m>=10 || n<0 || n>=10);
     //Nhap ma tran
     for (i=0; i<m; i++)
         for (j=0; j<n; j++)
         {
              cout<<"Nhap a["<<i<<","<<j<<"]= ";
              cin>>a[i][j];
          }
```

```
//In ma tran
for (i=0; i<m; i++)
{
     for (j=0; j<n; j++)
          cout<<a[i][j]<<'\t';
     cout<<endl;
}
//Dem so phan tu la so chinh phuong
d=0;
for (i=0; i<m; i++)
    for (j=0; j<n; j++)
         if (sqrt(a[i][j])==(int)sqrt(a[i][j]))
              d++;
cout<<"Trong ma tran co "<<d;
cout<<" phan tu la so chinh phuong"<<endl;
//Nhap va tinh tich cac phan tu tren cot c
do
{    cout<<"Nhap cot can tinh tich : ";
     cin>>c;
}while (c<0 || c>=n);
t=1;
for (i=0; i<m; i++)
     if (a[i][c]!=0)
          t=t*a[i][c];
cout<<"Tich cac phan tu khac 0 nam tren cot "<<c;
cout<<" la: "<<t<<endl;
//Nhap va sap xep giam dan cac phan tu tren hang h
do
{    cout<<"Nhap hang can sap xep : ";
     cin>>h;
}
while (h<0 || h>=m);
```

```
     for (i=0; i<n-1; i++)
          for (j=i+1; j<n; j++)
               if (a[h][i]<a[h][j])
               {
                    tam=a[h][i];
                    a[h][i]=a[h][j];
                    a[h][j]=tam;
               }
     //In ma tran sau khi sap xep
     for (i=0; i<m; i++)
     {
          for (j=0; j<n; j++)
               cout<<a[i][j]<<'\t';
          cout<<endl;
     }
}
```

✧ ***Ghi chú*:** Khi làm việc với ma trận do thói quen chúng ta thường tính chỉ số từ 1 (thay vì 0), do vậy trong ma trận ta có thể bỏ qua hàng 0, cột 0 bằng cách khai báo số hàng và cột tăng lên 1 so với số hàng, cột thực tế của ma trận và từ đó có thể làm việc từ hàng 1, cột 1 trở đi.

Ví dụ 3: Viết chương trình nhập vào ma trận vuông cấp n, các phần tử kiểu số thực. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu phần tử có phần nguyên là chẵn. In ra các phần tử nằm trên đường chéo chính. Tính tổng các phần tử nằm trên đường chéo phụ. In ra các phần tử nằm trên đường biên của ma trận. Cho biết phần tử có nội dung nhỏ nhất nằm ở các vị trí nào trong ma trận.

```
#include <iostream>
using namespace std;
main()
{  float a[11][11], s, min;
   int n, i, j, d;
   do
   {    cout<<"Nhap so hang va so cot : ";
        cin>>n;
   } while (n<=0 || n>10);
```

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
    {
           cout<<"Nhap a["<<i<<","<<j<<"]= ";
           cin>>a[i][j];
    }
 for (i=1; i<=n; i++)
 {
      for (j=1; j<=n; j++)
           cout<<a[i][j]<<"  ";
      cout<<endl;
 }
 d=0;
 for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      if (int(a[i][j])%2==0)
           d++;
 cout<<"Co "<<d<<" p.tu co phan nguyen la chan"<<endl;
 cout<<"Cac phan tu nam tren duong cheo chinh : ";
 for (i=1; i<=n; i++)
      cout<<a[i][i]<<'\t';
 cout<<endl;
 s=0;
 for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      if (i+j == n+1)
           s += a[i][j];
 cout<<"Tong cac p.tu nam tren duong cheo phu ";
 cout<<"la :"<<s<<endl;
 for (i=1; i<=n; i++)
 {
      for (j=1; j<=n; j++)
```

```
            if (i==1 || i==n || j==1 || j==n)
                cout<<a[i][j]<<"  ";
            else cout<<"    ";
        cout<<endl;
    }
    min=a[1][1];
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        if (min>a[i][j]) min=a[i][j];
    cout<<"Cac p.tu nho nhat nam o cac vi tri : "<<endl;
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        if (a[i][j] == min)
            cout<<"Hang "<<i<<" Cot "<<j<<endl;
}
```

BÀI TẬP CHƯƠNG 3

-----ଓ✧ଓ-----

1. Viết chương trình nhập vào mảng 1 chiều gồm n phần tử kiểu số nguyên. In ra mảng vừa nhập theo thứ tự ngược. Cho biết có bao nhiêu phần tử có nội dung là số nguyên tố. Tính tích các phần tử là ước số của k, với k được nhập từ bàn phím. Cho biết phần tử X xuất hiện ở lần thứ m tại vị trí thứ mấy, với X và m được nhập từ bàn phím. Sắp xếp mảng theo thứ tự giảm dần. In ra mảng sau khi sắp xếp.

2. Viết chương trình nhập vào 2 mảng 1 chiều có cùng số phần tử và nội dung các phần tử kiểu số thực. Tạo ra mảng thứ 3 bằng tổng của 2 dãy đó (c[i] = a[i] + b[i]) in ra dãy vừa tạo.

3. Viết chương trình nhập vào 2 mảng 1 chiều có cùng số phần tử và nội dung các phần tử kiểu bool. In ra 2 mảng vừa nhập trên 2 hàng khác nhau. Tạo ra mảng thứ 3 là kết quả của việc thực hiện phép toán AND trên từng phần tử của 2 mảng đã nhập. Tạo ra mảng thứ 4 là kết quả của việc thực hiện phép toán OR trên từng phần tử của 2 mảng đã nhập. In ra 2 mảng vừa tạo trên 2 hàng khác nhau.

4. Viết chương trình thực hiện các công việc sau:

- Nhập mảng 1 chiều gồm n phần tử kiểu số nguyên.
- In ra mảng vừa nhập.
- In ra vị trí của các phần tử lớn nhất có trong dãy.
- Tính trị trung bình của các phần tử dương có trong dãy.
- Đếm số phần tử là lũy thừa của K, với K nhập từ bàn phím.

5. Viết chương trình thực hiện các công việc sau:

- Nhập mảng 1 chiều gồm n phần tử kiểu số nguyên.
- In ra các phần tử nằm ở các vị trí chẵn.
- Kiểm tra xem dãy có thứ tự hay không?
- Kiểm tra xem dãy có đối xứng không?
- Tạo ra 1 mảng mới được copy từ mảng nhập gồm M phần tử bắt đầu từ phần tử thứ K, với M và K được nhập từ bàn phím.
- In ra mảng vừa tạo.

6. Viết chương trình nhập vào ma trận a gồm m hàng, n cột, các phần tử kiểu số thực. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu phần tử có phần nguyên là chẵn. Tính tích các phần tử dương nằm trên hàng h, với h nhập từ bàn phím. Sắp xếp các phần tử nằm trên cột c theo thứ tự tăng dần, với c nhập từ bàn phím. In ra ma trận sau khi sắp xếp.

7. Viết chương trình nhập vào ma trận vuông cấp n, các phần tử kiểu ký tự. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu ký tự 'T'. In ra các phần tử nằm trên đường chéo phụ. Cho biết ký tự lớn nhất nằm trên đường chéo chính là gì. Cho biết ma trận có hàng nào có thứ tự tăng dần không?

8. Viết chương trình nhập vào ma trận vuông cấp n, các phần tử kiểu số nguyên. In ra ma trận vừa nhập. In ra các phần tử nằm trên đường biên của ma trận. Cho biết phần tử nhỏ nhất nằm ở vị trí nào. Cho biết hàng nào có tổng các phần tử là lớn nhất. Tính giá trị trung bình của tất cả các phần tử dương có trong ma trận.

9. Viết chương trình nhập vào 2 ma trận gồm m hàng, n cột, các phần tử kiểu số nguyên. Tạo ra ma trận thứ 3 là ma trận tổng của 2 ma trận vừa nhập. Tạo ra ma trận thứ 4 là ma trận hiệu của 2 ma trận vừa nhập.

10. Viết chương trình nhập vào 2 ma trận các phần tử kiểu số thực. Kiểm tra xem hai ma trận này có thể tính tích được không? Nếu được thì tính tích hai ma trận này và lưu thành một ma trận mới.

CHƯƠNG 4: DỮ LIỆU KIỂU CHUỖI

-----ଓ✧ଌ-----

4.1 ĐỊNH NGHĨA

Một chuỗi ký tự là một dãy bất kỳ các ký tự (kể cả khoảng trắng (space)) và được lưu trữ trong một vùng nhớ liên tục. Một chuỗi ký tự có thể được xem như một mảng một chiều với kiểu phần tử là kiểu ký tự (char). Chuỗi ký tự còn được gọi là xâu ký tự, từ tiếng anh là *string*.

Tuy nhiên để máy có thể nhận biết được mảng ký tự này là một chuỗi thì cần phải có một ký tự kết thúc chuỗi, theo qui ước là ký tự có mã 0 (tức '\0') tại vị trí nào đó trong mảng. Khi đó chuỗi là dãy ký tự bắt đầu từ phần tử đầu tiên (thứ 0) đến ký tự kết thúc chuỗi đầu tiên (không kể các ký tự còn lại trong mảng).

Ví dụ:

0	1	2	3	4	5	6	7
H	E	L	L	O	\0		
H	E	L	\0				
\0							

Ví dụ trên minh họa 3 chuỗi, mỗi chuỗi được chứa trong mảng ký tự có độ dài tối đa là 8. Nội dung chuỗi thứ nhất là "HELLO" có độ dài thực sự là 5 ký tự, chiếm 6 ô trong mảng (thêm ô chứa ký tự kết thúc '\0'). Chuỗi thứ hai có nội dung "HEL" với độ dài 3 (chiếm 4 ô) và chuỗi cuối cùng biểu thị một chuỗi rỗng (chiếm 1 ô). Chú ý mảng ký tự được khai báo với độ dài 8 tuy nhiên các chuỗi có thể chỉ chiếm một số ký tự nào đó trong mảng này và tối đa là 7 ký tự.

4.2 KHAI BÁO

char <tên chuỗi>[độ dài] ; // không khởi tạo

char <tên chuỗi>[độ dài] = chuỗi ký tự ;// có khởi tạo

char <tên chuỗi>[ ] = chuỗi ký tự ; // có khởi tạo

- Độ dài chuỗi là số ký tự tối đa có thể có trong chuỗi. Độ dài thực sự của chuỗi chỉ tính từ đầu chuỗi đến dấu kết thúc chuỗi (không kể dấu kết thúc chuỗi '\0').

- Do một chuỗi phải có dấu kết thúc chuỗi nên trong khai báo độ dài của chuỗi cần phải khai báo thừa ra một phần tử. Thực chất độ dài tối đa của chuỗi = độ dài mảng - 1.

Ví dụ: Nếu muốn khai báo mảng s chứa được chuỗi có độ dài tối đa 80 ký tự, ta cần phải khai báo `char s[81];`

- Cách khai báo thứ hai có kèm theo khởi tạo chuỗi, đó là dãy ký tự đặt giữa cặp dấu nháy kép.

Ví dụ: `char hoten[40];` // chuỗi họ tên chứa tối đa 39 ký tự

```
char monhoc[31] = "NNLT C++" ;
```

Chuỗi môn học chứa tối đa 30 ký tự, được khởi tạo với nội dung "NNLT C++" với độ dài thực sự là 8 ký tự (chiếm 9 ô đầu tiên trong mảng monhoc[31]).

- Cách khai báo thứ 3 thì chương trình sẽ quyết định độ dài của chuỗi bởi chuỗi khởi tạo (bằng độ dài chuỗi + 1).

Ví dụ: `char thang[] = "Muoi hai" ;` // độ dài chuỗi = 8 + 1 = 9

4.3 CÁCH SỬ DỤNG

Tương tự như các mảng dữ liệu khác, chuỗi ký tự có những đặc trưng như mảng, tuy nhiên chúng cũng có những điểm khác biệt. Dưới đây là các điểm giống và khác nhau giữa chuỗi ký tự và mảng thông thường.

- Truy cập một ký tự trong chuỗi: cú pháp giống như mảng.

Ví dụ:

```
char s[50] = "I\'m a student";    // ký tự ' phải được viết là \'
cout << s[0];        // in ký tự đầu tiên, tức ký tự 'I'
s[1] = 'a';          // đặt lại ký tự thứ 2 là 'a'
```

- Không được thực hiện các phép toán trực tiếp trên chuỗi như phép gán, các phép so sánh.

Ví dụ:

```
char s[20] = "Hello", t[20] ;
t = "Hello" ;      // ISO C++ forbids assignment of arrays
t = s ;       // ISO C++ forbids assignment of arrays
if (s < t) cout<<"s nho hon t"; //khong bao loi nhung
else cout<<"s khong nho hon t"; //cho ket qua sai
```

- Toán tử nhập dữ liệu >> vẫn dùng được nhưng có nhiều hạn chế.

Ví dụ:

```
char s[60];
cout << "Nhap chuoi s : ";
cin >> s;
cout << s;
```

Nếu chuỗi nhập vào là "Tin hoc" chẳng hạn thì toán tử >> chỉ nhập "Tin" cho s (bỏ tất cả các ký tự đứng sau dấu trắng), vì vậy khi in ra trên màn hình chỉ có từ "Tin".

4.4 PHƯƠNG THỨC NHẬP CHUỖI

Do toán tử nhập >> có nhiều hạn chế đối với việc nhập chuỗi ký tự nên trong C++ người ta thường sử dụng hàm (còn được gọi là phương thức) **cin.getline(s, n)** để nhập chuỗi ký tự.

Hàm **cin.getline(s, n)** có 2 đối số trong đó s là chuỗi cần nhập nội dung và n-1 là số ký tự tối đa của chuỗi. Khi gặp hàm cin.getline(s, n) chương trình sẽ tạm dừng chờ NLT nhập dữ liệu (dãy ký tự) vào từ bàn phím. NLT có thể nhập vào dãy với độ dài bất kỳ cho đến khi nhấn Enter, chương trình sẽ lấy ra n-1 ký tự đầu tiên gán cho s, phần còn lại bị bỏ qua (nếu chuỗi nhập có nhiều hơn n-1 ký tự). Hiển nhiên, sau khi gán các ký tự cho s, chương trình sẽ tự động đặt ký tự kết thúc chuỗi vào ô tiếp theo của chuỗi.

Ví dụ: Xét đoạn lệnh sau:

```
char s[10];
cin.getline(s, 10);
cout << s << endl;
```

Giả sử ta nhập vào từ bàn phím hàng ký tự: 1234567890abcd↵. Khi đó lệnh cin.getline(s, 10) sẽ gán chuỗi "123456789" (9 ký tự) cho s, phần còn lại bị bỏ qua, tiếp theo s được in ra màn hình. Như vậy trên màn hình sẽ xuất hiện kết quả là:

```
123456789
```

4.5 MỘT SỐ HÀM XỬ LÝ CHUỖI

4.5.1 Hàm strcpy(s, t)

Sao chép nội dung của chuỗi t cho chuỗi s (thay cho phép gán "=" không được sử dụng). Hàm sẽ sao chép toàn bộ nội dung của chuỗi t (kể cả ký tự kết thúc chuỗi) vào cho chuỗi s. Để sử dụng hàm này cần đảm bảo độ dài của chuỗi s ít nhất cũng bằng độ dài của chuỗi t. Trong trường hợp ngược lại ký tự kết thúc chuỗi sẽ không được ghi vào s và điều này có thể gây treo máy khi thực thi chương trình.

Ví dụ:

```
char s[10], t[10];
strcpy(t, "Face");
strcpy(s, t);
cout << s << " to " << t << endl;
```

4.5.2 Hàm strncpy(s, t, n)

Sao chép n ký tự đầu tiên của t vào s. Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn ký tự kết thúc chuỗi cho s. Do vậy NLT phải thêm câu lệnh đặt ký tự '\0' vào cuối chuỗi s sau khi sao chép xong.

Ví dụ:

```
char s[10], t[10] = "Steven";
strncpy(s, t, 5);
s[5] = '\0';
cout << s << " is young brother of " << t << endl;
```

✧ Nếu ta muốn sao chép n ký tự tính từ ký tự thứ k của t vào s thì ta sử dụng hàm **strncpy(s, t+k, n)**.

Ví dụ:

```
char s[10], t[10] = "Steven";
 strncpy(s, t+2, 4);
 s[4] = '\0';
 cout << s << endl; // in ra la: even
```

4.5.3 Hàm strcat(s, t)

Ghép (nối) một bản sao của t vào sau s (thay cho phép +). Hiển nhiên hàm sẽ loại bỏ ký tự kết thúc chuỗi s trước khi ghép t vào. Việc nối sẽ đảm bảo lấy cả ký tự kết thúc của chuỗi t vào cho s (nếu s đủ chỗ). Vì vậy NLT không cần thêm ký tự này vào cuối chuỗi. Tuy nhiên, hàm không kiểm tra xem liệu độ dài của s có đủ chỗ để ghép thêm t hay không, việc kiểm tra này phải do NLT đảm nhiệm.

Ví dụ 1:

```
char a[40] = "Thanh", b[5] = "Minh";
strcat(a, " va ");
strcat(a, b);
cout << a << endl;
```

Hai câu lệnh gọi hàm strcat trên có thể gom chung lại thành 1 câu lệnh như sau:

```
strcat(strcat(a, " va "), b);
```

Ví dụ 2:

```
char s[10], t[10] = "Steve";
strncpy(s, t, 3);
s[3] = '\0';
strcat(s, "p");
cout << t << " goes "<< s << " by " << s  << endl;
```

4.5.4 Hàm strncat(s, t, n)

Ghép bản sao n ký tự đầu tiên của chuỗi t vào sau chuỗi s. Hàm tự động đặt thêm dấu kết thúc chuỗi vào s sau khi ghép xong (tương phản với strncpy()). Cũng giống như hàm strcat hàm đòi hỏi độ dài của s phải đủ chứa kết quả. Tương tự, có thể sử dụng cách viết **strncat(s, t+k, n)** để ghép n ký tự từ vị trí thứ k của chuỗi t cho s.

Ví dụ:

```
char s[20] = "Khoa ";
char t[] = "cong nghe thong tin";
```

```
strncat(s, t, 9);  // s = "Khoa cong nghe"
strcpy(s,"Khoa ");
strncat(s, t+10, 9);    // s = "Khoa thong tin"
```

4.5.5 Hàm strcmp(s, t)

Hàm so sánh 2 chuỗi s và t (thay cho các phép toán so sánh). Giá trị trả về là hiệu 2 ký tự khác nhau đầu tiên của s và t. Từ đó, nếu s < t thì hàm trả về giá trị là -1, nếu s==t thì hàm trả về giá trị là 0, và nếu s > t thì hàm trả về giá trị là 1.

Tiêu chí để so sánh là số nguyên tương ứng trong bảng mã ASCII. Vì thế hàm này có phân biệt chữ hoa và chữ thường (tức là 'A' khác với 'a') và 'A' < 'a' (vì 65 < 97).

Trong trường hợp chỉ quan tâm đến so sánh bằng, nếu hàm trả về giá trị là 0 là 2 chuỗi bằng nhau và nếu giá trị trả về khác 0 thì 2 chuỗi khác nhau.

Ví dụ:

```
char s[] = "Ha Noi" , t[] = "ha noi" ;
cout << strcmp(s, t)<<endl; // in ra -1
strcpy(s, "VIET NAM");
strcpy(t, s);
cout << strcmp(s, t)<<endl; // in ra 0
strcpy(s, "THONG TIN");
strcpy(t, "CONG NGHE");
cout << strcmp(s, t)<<endl; // in ra 1
```

4.5.6 Hàm strncmp(s, t, n)

Giống như hàm strcmp(s, t) nhưng chỉ so sánh tối đa n ký tự đầu tiên của hai chuỗi.

Ví dụ:

```
char s[] = "Ha Noi" , t[] = "Ha noi" ;
cout << strcmp(s, t)<<endl; // in ra -1
cout << strncmp(s, t, 3)<<endl; // in ra 0
```

4.5.7 Hàm strcmpi(s, t)

Giống như hàm strcmp(s, t) nhưng không phân biệt chữ hoa, thường.

Ví dụ:

```
char s[] = "Ha Noi" , t[] = "ha noi" ;
cout << strcmpi(s, t)<<endl; // in ra 0
```

4.5.8 Hàm strupr(s)

Hàm đổi chuỗi s thành in hoa, và cũng trả lại chuỗi in hoa đó.

Ví dụ:

```
char s[10] = "Ha Noi";
cout << s << endl; // Ha Noi
cout << strupr(s) << endl;   // HA NOI
cout << s << endl; // HA NOI (s cung thay doi)
```

4.5.9 Hàm strlwr(s)

Hàm đổi chuỗi s thành in thường, kết quả trả về là chuỗi in thường đó.

Ví dụ:

```
char s[10] = "Ha Noi";
cout << s << endl; // Ha Noi
cout << strlwr(s) << endl;   // ha noi
cout << s << endl; // ha noi (s cung thay doi)
```

4.5.10 Hàm strlen(s)

Hàm trả về giá trị là độ dài thực sự của chuỗi s.

Ví dụ:

```
char s[10] = "Ha Noi";
cout << strlen(s) << endl;   // in ra 6
```

4.6 CÁC VÍ DỤ MINH HỌA

4.6.1 Ví dụ 1

Viết chương trình nhập vào một chuỗi bất kỳ, thống kê số chữ cái, số chữ số, số khoảng trắng có trong chuỗi đó.

```
#include <iostream>
#include <string.h>
using namespace std;
main()
{    char s[100];
     int i, cd, dc, ds, dt;
     dc=ds=dt=0;
     cout<<"Nhap vao chuoi khong qua 99 ky tu:";
     cin.getline(s, 100);
     cd=strlen(s);
```

```
     for(i=0; i<cd; i++)
        if(isalpha(s[i])) dc++;
        else if(isdigit(s[i])) ds++;
           else if (s[i]== ' ') dt++;
     cout<<"Trong chuoi co "<<dc<<" chu cai, ";
     cout<<ds<<" chu so va "<<dt<<" khoang trang"<<endl;
}
```

4.6.2 Ví dụ 2

Viết chương trình nhập vào mật khẩu (không quá 12 ký tự). In ra "Đúng rồi. Mời vào!" nếu là "VinhLong2013", "Sai rồi. Nhập lại!" nếu ngược lại và chưa hết số lần thử, nếu hết số lần thử thì in ra "Bạn đã hết quyền nhập mật khẩu!". Chương trình cho phép nhập tối đa 3 lần. Nhập riêng rẽ từng ký tự (bằng hàm getch()) cho mật khẩu. Hàm getch() không hiện ký tự NLT nhập vào mà thay vào đó chương trình sẽ hiển thị ký tự 'X' để che giấu mật khẩu.

```
#include <iostream>
#include <string.h>
#include <conio.h>
using namespace std;
main()
{   char pw[13], kt;
    int i, solan = 0; //Cho phep nhap 3 lan
    do
    {   cout<<"Nhap mat khau vao:";
        i = 0;
        do
        {
            kt=getch();
            if(kt != 13) pw[i]=kt;
            else pw[i]= '\0';
            cout<<'X' ;
            i++;
        } while (kt != 13);
      cout<<endl;
```

```
      if (strcmp(pw, "VinhLong2013")==0)
      {
          cout<<"Dung roi. Moi vao!"<<endl;
          break;
      }
      else
      {
          solan++ ;
          if(solan < 3)
          cout<<"Sai roi. Nhap lai!"<<endl;
       }
    } while (solan < 3);
    if (solan == 3)
        cout<<"Ban da het quyen nhap mat khau!"<<endl;
}
```

4.6.3 Ví dụ 3

Viết chương trình nhập vào một chuỗi bất kỳ, cho biết trong chuỗi có bao nhiêu ký tự không phải là chữ cái. Đổi tất cả các chữ cái thành dạng chuẩn Upper (toàn bộ là ký tự hoa). Cho biết trong chuỗi các chữ số nào xuất hiện nhiều nhất và chúng xuất hiện bao nhiêu lần.

```
#include <iostream>
#include <string.h>
using namespace std;
main()
{
    char st[100];
    int i, max, dkt=0, a[10];
    cout<<"Nhap chuoi:";
    cin.getline(st,100);
    for (i=0; i<strlen(st); i++)
        if (isalpha(st[i])==0) dkt++;
    cout<<"Co "<<dkt<<" ky tu khong phai la chu cai"<<endl;
    for (i=0; i<strlen(st); i++)
```

```
        if (islower(st[i])) st[i]=toupper(st[i]);
    cout<<"Chuoi sau khi doi la:"<<endl;
    cout<<st<<endl;
    for (i=0; i<=9; i++)
        a[i]=0;
    for (i=0; i<strlen(st); i++)
        if (isdigit(st[i]))
           a[(int)st[i]-48]=a[(int)st[i]-48]+1;
    max=a[0];
    for (i=1; i<=9; i++)
        if (max<a[i]) max=a[i];
    cout<<"Cac chu so xuan hien nhieu nhat la: ";
    for (i=0; i<=9; i++)
        if (a[i]==max) cout<<i<<'\t';
    cout<<"\nChung xuat hien "<<max<<" lan."<<endl;
}
```

4.6.4 Ví dụ 4

Viết chương trình nhập vào họ tên của một người, kiểm tra tính hợp lệ của họ tên đã nhập (chỉ gồm chữ cái và khoảng trắng nhưng không toàn là khoảng trắng), cắt bỏ các khoảng trắng thừa ở đầu và cuối, cho biết trong họ tên có mấy ký tự kt với kt được nhập từ bàn phím, in ra tên của người này.

```
#include <iostream>
#include <string.h>
using namespace std;
main()
{   char ht[40], ten[10], tam[40], kt;
    int i, dc, dt, dkt;
    do
    {  cout<<"Nhap ho ten:";
       cin.getline(ht, 40);
       dc=dt=0;
        for (i=0; i<strlen(ht); i++)
```

```
            if (isalpha(ht[i])) dc++;
            else if (ht[i]==' ')dt++;
    } while (dc+dt!=strlen(ht) || dc==0);
    i=0;
    while (ht[i]==' ') i++;
    strcpy(tam, "");
    strncpy(tam, ht+i, strlen(ht));
    tam[strlen(ht)-i]='\0';
    i=strlen(tam)-1;
    while (tam[i]==' ') i--;
    strcpy(ht,"");
    strncpy(ht, tam, i+1);
    ht[i+1]='\0';
    cout<<"Ho ten sau khi cat bo khoang trang la:";
    cout<<ht<<endl;
    cout<<"Nhap ky tu kt:";
    cin>>kt;
    dkt=0;
    for (i=0; i<strlen(ht); i++)
        if (ht[i]==kt) dkt++;
    cout<<"Trong ho ten co "<<dkt<<" ky tu "<<kt<<endl;
    i=strlen(ht)-1;
    while(ht[i]!=' ') i--;
    strncpy(ten, ht+i+1, strlen(ht)-1-i);
    ten[strlen(ht)-1-i]='\0';
    cout<<"Ten cua nguoi nay la:"<<ten<<endl;
}
```

BÀI TẬP CHƯƠNG 4

-----ଓ✧ଌ-----

1. Viết chương trình nhập vào một số nguyên hệ thập phân (cơ số 10), đổi số đó sang hệ nhị phân (cơ số 2).

2. Viết chương trình nhập vào một số nguyên hệ nhị phân (cơ số 2), đổi số đó sang hệ thập phân (cơ số 10).

3. Viết chương trình nhập vào một số nguyên hệ thập phân (cơ số 10), đổi số đó sang hệ thập lục phân (cơ số 16).

4. Viết chương trình nhập vào một số nguyên hệ bát phân (cơ số 8), đổi số đó sang hệ thập lục phân (cơ số 16).

5. Viết chương trình nhập vào một số nguyên hệ nhị phân (cơ số 2), đổi số đó sang hệ bát phân (cơ số 8).

6. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược theo từng ký tự.

Ví dụ: Nhập chuỗi ‘ABCD’ ⇨ Chuỗi đảo là: ‘DCBA’

7. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược theo từng từ.

Ví dụ: Nhập chuỗi ‘Cấm Không Được Câu Cá’

⇨ Chuỗi đảo là: ‘Cá Câu Được Không Cấm’

8. Viết chương trình nhập vào một số nguyên dương nhỏ hơn 1000, đọc số đó ra chữ.

Ví dụ: Nhập số 125 ⇨ Đọc là ‘Một trăm hai mươi lăm’

9. Viết chương trình nhập vào một chuỗi ký số (các ký tự từ 1 đến 9), hãy cho biết trong chuỗi có bao nhiêu dãy con tăng dần, in ra dãy con tăng dần dài nhất có trong chuỗi này.

10. Viết chương trình nhập vào một chuỗi bất kỳ, cho biết từ nào xuất hiện nhiều nhất.

CHƯƠNG 5: CON TRỎ VÀ HÀM

-----ଓ✧ଌ-----

5.1 CON TRỎ

5.1.1 Định nghĩa

Con trỏ là một biến dùng để giữ địa chỉ của biến khác. Nếu p là con trỏ giữ địa chỉ của biến x ta gọi p trỏ tới x và x được trỏ bởi p. Thông qua con trỏ ta có thể làm việc được với nội dung của những ô nhớ mà p trỏ đến.

Con trỏ là một đặc trưng mạnh của C++, nó cho phép chúng ta thâm nhập trực tiếp vào bộ nhớ để xử lý các bài toán khó chỉ bằng vài câu lệnh đơn giản của chương trình.

5.1.2 Khai báo

<Kiểu được trỏ> <*Tên biến con trỏ> ;

<Kiểu được trỏ*> <Tên biến con trỏ> ;

Địa chỉ của một biến là địa chỉ byte nhớ đầu tiên của biến đó. Vì vậy để lấy được nội dung của biến, con trỏ phải biết được số byte của biến, tức là kiểu của biến mà con trỏ sẽ trỏ tới. Kiểu này cũng được gọi là kiểu của con trỏ. Như vậy khai báo biến con trỏ cũng giống như khai báo một biến thường ngoại trừ cần thêm dấu * trước tên biến (hoặc sau tên kiểu).

Ví dụ:

int *p ; // khai báo biến p là biến con trỏ trỏ đến kiểu số nguyên

float *q, *r ; // khai báo 2 con trỏ thực q và r

5.1.3 Cách sử dụng

- Để con trỏ p trỏ đến biến x ta phải dùng phép gán p = địa chỉ của x. Nếu x không phải là biến mảng thì ta viết: p = &x. Còn nếu x là biến mảng thì ta viết: p = x hoặc p = &x[0].

- Ta không thể gán p cho một hằng địa chỉ cụ thể. Chẳng hạn ta viết p = 200 là sai.

- Phép toán * cho phép lấy nội dung nơi p trỏ đến, ví dụ để gán nội dung nơi p trỏ đến cho biến f ta viết f = *p.

- Phép toán & và phép toán * là 2 phép toán ngược nhau. Cụ thể nếu p = &x thì x = *p. Từ đó nếu p trỏ đến x thì bất kỳ nơi nào xuất hiện x đều có thể thay được bởi *p và ngược lại.

Ví dụ 1:

```
#include <iostream>
using namespace std;
main()
{    int i, j; // khai báo 2 biến nguyên i, j
     int *p, *q; // khai báo 2 con trỏ nguyên p, q
```

```
    p = &i; // cho p trỏ tới i
    q = &j; // cho q trỏ tới j
    cout << &i; // hỏi địa chỉ biến i
    cout << q; // hỏi địa chỉ biến j (thông qua q)
    i = 2; // gán i bằng 2
    *q = 5; // gán j bằng 5 (thông qua q)
    i++ ; cout << i;   // tăng i và in ra i = 3
    (*q)++ ; cout << j;// tăng j (thông qua q) và in j = 6
    (*p) = (*q) * 2 + 1; // gán lại i (thông qua p)
    cout << i; // i = 13
}
```

Qua ví dụ trên ta thấy mọi thao tác với i là tương đương với *p, với j là tương đương với *q và ngược lại.

5.1.4 Các phép toán

5.1.4.1 Phép toán gán

- Gán con trỏ với địa chỉ một biến: p = &x ;

- Gán con trỏ với con trỏ khác: p = q ; (sau phép toán gán này p, q chứa cùng một địa chỉ hay cùng trỏ đến một nơi).

Ví dụ 2:

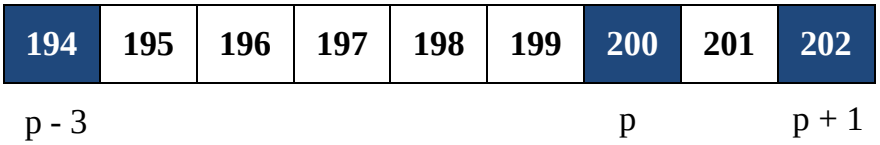
```
#include <iostream>
using namespace std;
main()
{    int i = 10;
     int *p, *q, *r;
     p = q = r = &i;
     *p = *q + 3;
     *q = *r * 2;
     cout << i << endl;
}
```

5.1.4.2 Phép toán tăng, giảm địa chỉ

p ± n: con trỏ trỏ đến thành phần thứ n sau (trước) p.

Một đơn vị tăng giảm của con trỏ bằng kích thước của biến được trỏ.

Ví dụ: Giả sử p là con trỏ nguyên (2 bytes) đang trỏ đến địa chỉ 200 thì p + 1 là con trỏ trỏ đến địa chỉ 202; tương tự, p + 5 là con trỏ trỏ đến địa chỉ 210; và p - 3 chứa địa chỉ 194.



Như vậy, phép toán tăng, giảm con trỏ cho phép làm việc thuận lợi trên mảng. Nếu con trỏ đang trỏ đến mảng (tức đang chứa địa chỉ đầu tiên của mảng), việc tăng con trỏ lên 1 đơn vị sẽ dịch chuyển con trỏ trỏ đến phần tử thứ hai, và cứ thế tiếp tục. Từ đó ta có thể cho con trỏ chạy từ đầu đến cuối mảng bằng cách tăng con trỏ lên từng đơn vị như trong câu lệnh for trong ví dụ dưới đây.

Ví dụ 3:

```
#include <iostream>
using namespace std;
main()
{    int a[10] = { 1, 2, 3, 4, 5, 6, 7 }, *p, *q;
     p = a; // cho p trỏ đến mảng a
     cout << *p ;  // *p = a[0] = 1
     p += 5;
     cout << *p ; // *p = a[5] = 6
     q = p - 4 ;
     cout << *q ; // q = a[1] = 2
     p = a;
     for (int i = 0;  i < 10;  i++)
          cout << *(p + i) << '\t' ;  // in toàn bộ mảng a
}
```

5.1.4.3 Phép toán tự tăng, tự giảm

p++, p--, ++p, --p: Tương tự p + 1 và p - 1, có chú ý đến tăng (giảm) sau hay trước.

Ví dụ sau minh họa kết quả kết hợp phép tự tăng, tự giảm với việc lấy giá trị nơi con trỏ trỏ đến. a là một mảng gồm 2 số, p là con trỏ trỏ đến mảng a.

Ví dụ 4:

```
#include <iostream>
using namespace std;
main()
{    int a[2] = {3, 7}, *p = a;
```

```
    cout << (*p)++ << endl; // in ra 3
    cout << *p << endl;     // in ra 4
    cout << ++(*p) << endl; // in ra 5
    cout << *p << endl;     // in ra 5
    cout << *(p++) << endl; // in ra 5
    cout << *p << endl;     // in ra 7
}
```

✧ ***Chú ý***: Ta phải phân biệt giữa p + 1 và p++ (hoặc ++p):

• p + 1 được xem như một con trỏ khác với con trỏ p. Con trỏ p + 1 trỏ đến phần tử sau p.

• p++ là con trỏ p nhưng trỏ đến phần tử khác. Con trỏ p++ trỏ đến phần tử đứng sau phần tử p trỏ đến ban đầu.

5.1.4.4 Phép hiệu của 2 con trỏ

Phép toán này chỉ thực hiện được khi p và q là 2 con trỏ cùng trỏ đến các phần tử của một dãy dữ liệu nào đó trong bộ nhớ (chẳng hạn cùng trỏ đến 1 mảng dữ liệu).

Khi đó hiệu p - q là số phần tử giữa p và q (chú ý p - q không phải là hiệu của 2 địa chỉ mà là số phần tử giữa p và q).

Ví dụ:

Giả sử p và q là 2 con trỏ số nguyên có kích thước 2 bytes, p có địa chỉ 200 và q có địa chỉ 208. Khi đó p - q = - 4 và q - p = 4 (4 là số phần tử nguyên từ địa chỉ 200 đến 208).

5.1.4.5 Phép toán so sánh

Các phép toán so sánh cũng được áp dụng đối với con trỏ, thực chất là so sánh giữa địa chỉ của hai nơi được trỏ bởi các con trỏ này. Thông thường các phép so sánh <, <=, >, >= chỉ áp dụng cho hai con trỏ trỏ đến phần tử của cùng một mảng dữ liệu nào đó. Thực chất của phép so sánh này chính là so sánh chỉ số của 2 phần tử được trỏ bởi 2 con trỏ đó.

Ví dụ 5:

```
#include <iostream>
using namespace std;
main()
{   float a[5] = {1, 2, 3, 4, 5}, *p, *q;
    p = a; // p trỏ đến mảng (tức p trỏ đến a[0])
    q = &a[3]; // q trỏ đến phần tử thứ 3 (a[3]) của mảng
    cout << (p < q) << endl;  // in ra 1 (true)
    cout << (p + 3 == q) << endl; // in ra 1 (true)
    cout << (p > q - 1) << endl;  // in ra 0 (false)
```

```
    cout << (p >= q - 2) << endl; // in ra 0 (false)
    for (p = a ; p < a + 5; p++)
        cout << *p << '\t'; // in toàn bộ mảng a
    cout << endl;
}
```

5.1.5 Cấp phát động, toán tử cấp phát và thu hồi vùng nhớ

5.1.5.1 Cấp phát động

Khi tiến hành chạy chương trình, chương trình dịch sẽ bố trí các ô nhớ cụ thể cho các biến được khai báo trong chương trình. Vị trí cũng như số lượng các ô nhớ này tồn tại và cố định trong suốt thời gian chạy chương trình, chúng xem như đã bị chiếm dụng và sẽ không được sử dụng vào mục đích khác và chỉ được giải phóng sau khi chấm dứt chương trình. Việc phân bổ bộ nhớ như vậy được gọi là ***cấp phát tĩnh*** (vì được cấp sẵn trước khi chạy chương trình và không thể thay đổi tăng, giảm kích thước hoặc vị trí trong suốt quá trình chạy chương trình).

Ví dụ nếu ta khai báo một mảng nguyên chứa 1000 phần tử thì trong bộ nhớ sẽ có một vùng nhớ liên tục 2000 bytes để chứa dữ liệu của mảng này. Khi đó dù trong chương trình ta chỉ nhập vào mảng và làm việc với một vài phần tử thì phần mảng rỗi còn lại vẫn không được sử dụng vào việc khác. Đây là hạn chế thứ nhất của kiểu mảng. Ở một hướng khác, một lần nào đó chạy chương trình ta lại cần làm việc với hơn 1000 số nguyên. Khi đó vùng nhớ mà chương trình dịch đã dành cho mảng là không đủ để sử dụng. Đây chính là hạn chế thứ hai của mảng được khai báo trước.

Để khắc phục các hạn chế trên của kiểu mảng, bây giờ chúng ta sẽ không khai báo (bố trí) trước mảng dữ liệu với kích thước cố định như vậy. Kích thước cụ thể sẽ được cấp phát trong quá trình chạy chương trình theo đúng yêu cầu của NLT. Nhờ vậy chúng ta có đủ số ô nhớ để làm việc mà vẫn tiết kiệm được bộ nhớ, và khi không dùng nữa ta có thể thu hồi (còn gọi là giải phóng) số ô nhớ này để chương trình sử dụng vào việc khác. Hai công việc cấp phát và thu hồi này được thực hiện thông qua các toán tử new và delete, để thực hiện được điều này ta cần phải sử dụng biến kiểu con trỏ. Thông qua biến con trỏ ta có thể làm việc với bất kỳ địa chỉ nào của vùng nhớ được cấp phát. Cách thức bố trí bộ nhớ như thế này được gọi là ***cấp phát động***.

5.1.5.2 Toán tử cấp phát vùng nhớ (new)

Cú pháp:

p = new <kiểu> ;

p = new <kiểu>[n] ;

Với p là biến con trỏ.

Ví dụ:

```
int *p;
```

```
p = new int;
p = new int[10];
```

Khi gặp toán tử new, chương trình sẽ tìm trong bộ nhớ một lượng ô nhớ còn rỗi và liên tục với số lượng đủ theo yêu cầu và cho p trỏ đến địa chỉ (byte đầu tiên) của vùng nhớ này. Nếu không có vùng nhớ với số lượng như vậy thì việc cấp phát là thất bại và khi đó p bằng NULL (NULL là một địa chỉ rỗng, không xác định). Do vậy ta có thể kiểm tra việc cấp phát có thành công hay không thông qua việc kiểm tra con trỏ p bằng hay khác NULL.

Ví dụ 6:

```
#include <iostream>
using namespace std;
main()
{   float *p; int n;
    cout << "So luong can cap phat = ";
    cin >> n;
    p = new float[n];
    if (p != NULL)
         cout << "Cap phat thanh cong!";
}
```

5.1.5.3 Toán tử thu hồi vùng nhớ (delete)

Để thu hồi hay giải phóng vùng nhớ đã cấp phát cho một biến (khi không cần sử dụng nữa) ta sử dụng toán tử delete.

Cú pháp:

delete p ;

Với p là con trỏ được cấp phát vùng nhớ bởi toán tử new.

Để giải phóng toàn bộ mảng được cấp phát thông qua con trỏ p ta dùng câu lệnh:

delete[ ] p ;

Với p là con trỏ trỏ đến mảng.

5.1.6 Ví dụ minh họa

Viết chương trình nhập vào dãy số nguyên (không dùng mảng). In ra dãy đã nhập. Sắp xếp dãy tăng dần và in ra dãy kết quả.

Trong ví dụ này chương trình yêu cầu cấp phát bộ nhớ đủ chứa n số nguyên và được trỏ bởi con trỏ head. Khi đó địa chỉ của số nguyên đầu tiên và cuối cùng sẽ là head và

head + n - 1. p và q là 2 con trỏ chạy trên dãy số này, so sánh và đổi nội dung của các số này với nhau để sắp thành dãy tăng dần và cuối cùng in kết quả.

```
#include <iostream>
using namespace std;
main()
{
     int *head, *p, *q, n, tam;
                         // head trỏ đến (đánh dấu) đầu dãy
     cout << "Cho biet so phan tu cua day: ";
     cin >> n ;
     head = new int[n] ;// cấp phát bộ nhớ chứa n số nguyên
     for (p = head; p < head + n; p++)    // nhập dãy
     {
          cout << "Nhap p.tu thu " << p-head+1 << ": " ;
          cin >> *p ;
     }
     cout << "Day vua nhap la : " << endl;
     for (p = head; p < head + n; p++)
          cout << *p << '\t';
    cout << endl;
     // Sắp xếp dãy tăng dần
    for (p = head; p < head + n - 1; p++)
          for (q = p + 1; q < head + n; q++)
               if (*q < *p)
               {
                    tam = *p; *p = *q; *q = tam;
               }
     cout << "Day sau khi sap xep la : " << endl;
    for (p = head; p < head + n; p++)
        cout << *p << '\t';
    cout << endl;
}
```

5.1.7 Con trỏ và chuỗi ký tự

Một con trỏ ký tự có thể xem như một biến chuỗi ký tự, trong đó chuỗi chính là tất cả các ký tự kể từ byte con trỏ trỏ đến cho đến byte '\0' gặp đầu tiên. Vì vậy ta có thể khai báo các chuỗi dưới dạng con trỏ ký tự như sau:

char *s ;

char *s = "Hello" ;

Các hàm trên chuỗi vẫn được sử dụng như khi ta khai báo nó dưới dạng mảng ký tự. Bên cạnh đó ta còn được phép sử dụng phép gán cho 2 chuỗi dưới dạng con trỏ.

Ví dụ:

```
char *s, *t = "Cong nghe thong tin";
s = t;    // thay cho hàm strcpy(s, t);
```

Thực chất phép gán trên chỉ là gán 2 con trỏ với nhau, nó cho phép s bây giờ cũng được trỏ đến nơi mà t đang trỏ (tức là dãy ký tự "Cong nghe thong tin" đã bố trí sẵn trong bộ nhớ).

Khi khai báo chuỗi dạng con trỏ nó vẫn chưa có bộ nhớ cụ thể, vì vậy thông thường kèm theo khai báo ta cần phải xin cấp phát bộ nhớ cho chuỗi với độ dài cần thiết.

Ví dụ:

```
#include <iostream>
#include <string.h>
using namespace std;
main()
{   char *s = new char[30], *t;
    strcpy(s, "Hello");
    t = s;
    cout << "Chuoi s la : " << s << endl;
    cout << "Chuoi t la : " << t << endl;
}
```

Nếu không có lệnh cấp phát vùng nhớ cho t thì ta chỉ có thể gán s vào t mà không thể gọi hàm strcpy. Vì vậy để gọi được `strcpy(t, s);` ta cần phải có thêm câu lệnh `t = new char[30];` vào chương trình.

Chương trình trên được viết lại như sau:

```
#include <iostream>
#include <string.h>
using namespace std;
main()
```

```
{   char *s = new char[30], *t;
    strcpy(s, "Hello");
    t = new char[30];
    strcpy(t, s);
    cout << "Chuoi s la : " << s << endl;
    cout << "Chuoi t la : " << t << endl;
}
```

5.2 HÀM

5.2.1 Định nghĩa

Hàm (Function) là một chương trình con trong chương trình lớn. Hàm nhận (hoặc không) các đối số (tham số) và trả lại (hoặc không) một giá trị cho chương trình gọi nó. Trong trường hợp không trả lại giá trị, hàm hoạt động như một thủ tục trong các NNLT khác. Một chương trình là một tập hợp các hàm, trong đó có một hàm chính với tên gọi là main(), khi chạy chương trình, hàm main() sẽ được chạy đầu tiên và gọi đến các hàm khác. Kết thúc hàm main() cũng là kết thúc chương trình.

Hàm giúp cho việc phân đoạn chương trình thành những mô-đun riêng lẻ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa là một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, dễ dàng cho việc kiểm tra và bảo trì chương trình.

5.2.2 Các đặc trưng

- Nằm trong hoặc ngoài văn bản có chương trình gọi đến hàm, trong một văn bản có thể chứa nhiều hàm.

- Được gọi từ chương trình chính (hàm main()), từ hàm khác hoặc từ chính nó (tính đệ quy).

- Không lồng nhau.

5.2.3 Khai báo

Một hàm thường thực hiện chức năng: tính toán trên các đối số và trả lại kết quả, hoặc chỉ đơn thuần thực hiện một chức năng nào đó, không trả lại kết quả tính toán. Thông thường kiểu của kết quả trả về được gọi là kiểu của hàm. Các hàm thường được khai báo ở đầu chương trình. Các hàm viết sẵn được khai báo trong các tệp nguyên mẫu (tập tin tiêu đề) *.h.

Do đó, để sử dụng được các hàm này, cần có chỉ thị `#include <*.h>` ở ngay đầu chương trình, trong đó *.h là tên file cụ thể có chứa khai báo của các hàm được sử dụng (chẳng hạn để sử dụng các hàm toán học ta cần khai báo `#include <math.h>`). Đối với các hàm do người lập trình tự viết, cũng cần phải khai báo.

Khai báo một hàm như sau:

<Kiểu giá trị trả về> <Tên hàm> (Danh sách đối số) ;

Trong đó, kiểu giá trị trả về còn gọi là kiểu của hàm và có thể nhận bất kỳ kiểu chuẩn nào của C++ và cả kiểu do người lập trình khai báo. Đặc biệt nếu hàm không trả về giá trị thì kiểu giá trị trả về được khai báo là **void**.

Ví dụ:

```
long LapPhuong(int);
int NgauNhien();
void ChuoiHoa(char[ ]);
int  Cong(int, int);
```

5.2.4 Cấu trúc chung

Cấu trúc chung của một hàm bất kỳ được bố trí cũng giống như hàm main() trong các phần trước.

```
<Kiểu giá trị trả về> <Tên hàm> (Danh sách đối số hình thức)
    {
        Các khai báo cục bộ của hàm ; // chỉ dùng riêng cho hàm này
        Dãy lệnh của hàm ; // các câu lệnh xử lý
        return <Biểu thức trả về> ; // có thể nằm đâu đó trong dãy lệnh.
    }
```

- **Danh sách đối số hình thức** còn được gọi ngắn gọn là danh sách đối số gồm dãy các đối số cách nhau bởi dấu phẩy, đối số có thể là một biến thường, biến tham chiếu hoặc biến con trỏ. Mỗi đối số được khai báo giống như khai báo biến, tức là một cặp gồm **<kiểu đối số> <tên đối số>**.

- Với **hàm có trả lại giá trị** cần có câu lệnh return kèm theo sau là một biểu thức. Kiểu của giá trị biểu thức này chính là kiểu của hàm đã được khai báo ở phần tên hàm. Câu lệnh return có thể nằm ở vị trí bất kỳ trong phần dãy lệnh của hàm. Khi gặp câu lệnh return chương trình tức khắc thoát khỏi hàm và trả lại giá trị của biểu thức sau return như giá trị của hàm.

- Với **hàm không có trả lại giá trị** (tức kiểu hàm là void) thì không cần có câu lệnh return hoặc có câu lệnh return nhưng phía sau return không có <Biểu thức trả về>.

Ví dụ 1: Ví dụ sau định nghĩa hàm tính lập phương của số nguyên n (với n nguyên) và hàm đổi chuỗi bất kỳ về dạng chuẩn upper.

```
#include <iostream>
#include <string.h>
```

```
using namespace std;
long LapPhuong(int);
void ChuoiHoa(char *s);
main()
{   int n;
    cout<<"Nhap so nguyen n = "; cin>>n;
    cout<<"Lap phuong cua "<<n<<" la : ";
    cout<<LapPhuong(n)<<endl;
    cin.ignore(1);
    char *s = new char[100];
    cout<<"Nhap vao chuoi bat ky : ";
    cin.getline(s, 100);
    ChuoiHoa(s);
    cout<<"Chuoi dang upper la : "<<s<<endl;
}
long LapPhuong(int n)
{
    return n*n*n;
}
void ChuoiHoa(char *s)
{   int i;
    for(i = 0; i < strlen(s); i++)
        s[i] = toupper(s[i]);
}
```

✧ ***Các chú ý***:

- Danh sách đối số trong khai báo hàm có thể chứa hoặc không chứa tên đối số, thông thường ta chỉ khai báo kiểu đối số chứ không cần khai báo tên đối số, trong khi ở hàng đầu tiên của định nghĩa hàm phải có tên đối số đầy đủ.

- Cuối khai báo hàm phải có dấu chấm phẩy (;), trong khi cuối hàng đầu tiên của định nghĩa hàm không có dấu chấm phẩy.

- Hàm có thể không có đối số(danh sách đối số rỗng), tuy nhiên cặp dấu ngoặc sau tên hàm vẫn phải được viết. Chẳng hạn như: NgauNhien(), InTho(), …

- Một hàm có thể không cần phải khai báo nếu nó được định nghĩa trước khi có hàm nào đó gọi đến nó.

5.2.5 Lời gọi hàm

Lời gọi hàm được phép xuất hiện trong bất kỳ biểu thức, câu lệnh của hàm khác … Nếu lời gọi hàm lại nằm trong chính bản thân hàm đó thì ta gọi là đệ quy. Để gọi hàm ta chỉ cần viết tên hàm và danh sách các giá trị cụ thể truyền cho các đối số đặt trong cặp dấu ngoặc ( ).

Tên hàm(danh sách đối số thực tế);

- Danh sách đối số (tham số) thực tế còn gọi là danh sách giá trị gồm các giá trị cụ thể để gán lần lượt cho các đối số hình thức của hàm. Khi hàm được gọi thực hiện thì tất cả những vị trí xuất hiện của đối số hình thức sẽ được gán cho giá trị cụ thể của đối số thực tế tương ứng trong danh sách, sau đó hàm tiến hành thực hiện các câu lệnh của hàm (để tính kết quả).

- Danh sách đối số thực tế truyền cho đối số hình thức có số lượng bằng với số lượng đối số trong hàm và được truyền cho đối số theo thứ tự tương ứng. Các đối số thực tế có thể là các hằng, các biến hoặc biểu thức. Biến trong giá trị có thể trùng với tên đối số. Ví dụ ta có hàm in n lần ký tự c với khai báo `void inkitu(int n, char c);` và lời gọi hàm `inkitu(12, 'A');` thì n và c là các đối số hình thức, 12 và 'A' là các đối số thực tế hoặc giá trị. Các đối số hình thức n và c sẽ lần lượt được gán bằng các giá trị tương ứng là 12 và 'A' trước khi tiến hành các câu lệnh trong phần thân hàm. Giả sử hàm in ký tự được khai báo lại thành `inkitu(char c, int n);` thì lời gọi hàm cũng phải được thay lại thành `inkitu('A', 12);`

- Các giá trị tương ứng được truyền cho đối số phải có kiểu cùng với kiểu đối số (hoặc C++ có thể tự động chuyển về kiểu của đối số).

- Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện câu lệnh đầu tiên trong hàm được gọi. Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

Ví dụ 2: Giả sử ta cần tính giá trị của hàm $f = 2x^3 - 5x^2 - 4x + 1$, thay cho việc tính trực tiếp x^3 và x^2, ta có thể gọi hàm LuyThua() trong ví dụ trên để tính các giá trị này bằng cách gọi nó trong hàm main() như sau:

```
#include <iostream>
using namespace std;
double LuyThua(float x, int n)
{   int i;
    double kq = 1;
    for (i = 1; i <= n; i++)
        kq *= x;
    return kq;
}
```

```
main()
{   float x;
    double f;
    cout << "Nhap x = ";
    cin >> x;
    f = 2*LuyThua(x,3) - 5*LuyThua(x,2) - 4*x + 1;
    cout << "Gia tri cua ham f = " << f << endl;
}
```

5.2.6 Hàm với đối số mặc định

Trong phần trước chúng ta đã khẳng định số lượng đối số thực tế phải bằng số lượng đối số hình thức khi gọi hàm. Tuy nhiên, trong thực tế rất nhiều lần hàm được gọi với các giá trị của một số đối số hình thức được lặp đi lặp lại. Trong trường hợp như vậy lúc nào cũng phải viết một danh sách dài các đối số thực tế giống nhau cho mỗi lần gọi là một công việc không mấy thú vị. Từ thực tế đó C++ đưa ra một cú pháp mới về hàm sao cho một danh sách đối số thực tế trong lời gọi không nhất thiết phải viết đầy đủ nếu một số trong chúng đã có sẵn những giá trị định trước. Cú pháp này được gọi là hàm với đối số mặc định và được khai báo với cú pháp như sau:

<Kiểu hàm> <Tên hàm>(đs1, …, đsn, đsmđ1 = gt1, …, đsmđm = gtm) ;

- Các đối số đs1, …, đsn và các đối số mặc định đsmđ1, …, đsmđm đều được khai báo như cũ nghĩa là gồm có kiểu đối số và tên đối số.

- Riêng các đối số mặc định đsmđ1, …, đsmđm có gán thêm các giá trị mặc định gt1, …, gtm. Một lời gọi bất kỳ khi gọi đến hàm này đều phải có đầy đủ các đối số thực tế ứng với các đs1, …, đsn nhưng có thể có hoặc không các đối số thực tế ứng với các đối số mặc định đsmđ1, …, đsmđm. Nếu đối số nào không có đối số thực tế thì nó sẽ được tự động gán giá trị mặc định đã khai báo.

Ví dụ: Xét hàm `double LuyThua(float x, int n = 2);` Hàm này có một đối số mặc định là số mũ n, nếu lời gọi hàm bỏ qua số mũ này thì chương trình hiểu là tính bình phương của x (n = 2). Chẳng hạn lời gọi `LuyThua(4, 3)` được hiểu là tính 4^3, lời gọi `LuyThua(4, 2)` hay `LuyThua(4)` được hiểu là tính 4^2

✧ ***Lưu ý***: Các đối số mặc định phải nằm bên phải các đối số không mặc định.

5.2.7 Khai báo hàm trùng tên (Quá tải hàm)

Hàm trùng tên còn gọi là hàm chồng (đè) hay quá tải hàm (function overload). Đây là một kỹ thuật cho phép sử dụng cùng một tên gọi cho các hàm "***giống nhau***" (cùng mục đích) nhưng xử lý trên các kiểu dữ liệu khác nhau hoặc trên số lượng dữ liệu khác nhau.

Ví dụ:

Hàm sau tìm số lớn hơn trong 2 số nguyên:

```
int Max(int a, int b)
{
    return (a > b) ? a: b;
}
```

Chúng ta có thể định nghĩa chồng hàm trên để có thể tìm số lớn hơn trong 2 số thực:

```
float Max(float a, float b)
{
    return (a > b) ? a: b;
}
```

Khi đó tùy theo giá trị nhận vào của các đối số a và b thuộc kiểu int hay float mà phiên bản của hàm Max nào sẽ được chọn để thực hiện.

5.2.8 Các cách truyền đối số

Có 3 cách truyền đối số thực tế cho các đối số hình thức trong lời gọi hàm. Trong đó cách ta đã dùng cho đến thời điểm hiện nay được gọi là truyền bằng giá trị, còn được gọi là *tham trị* hay *truyền bằng giá trị*, tức các đối số hình thức sẽ nhận các giá trị cụ thể từ lời gọi hàm và tiến hành tính toán rồi trả lại giá trị.

5.2.8.1 Truyền bằng giá trị

Ta xét lại ví dụ hàm `LuyThua(float x, int n)` dùng để tính tính x^n. Giả sử trong chương trình chính (hàm main()) ta có các biến a, b, f đang chứa các giá trị a = 2, b = 3, và f chưa có giá trị. Để tính a^b và gán giá trị trả về vào f, ta có thể gọi `f = LuyThua(a,b);`

Khi gặp lời gọi này, chương trình sẽ tổ chức như sau:

- Tạo 2 biến mới (tức 2 ô nhớ trong bộ nhớ) có tên x và n. Gán nội dung các ô nhớ này bằng các giá trị trong lời gọi, tức gán 2 (a) cho x và 3 (b) cho n.

- Tới phần khai báo (của hàm), chương trình tạo thêm các ô nhớ mang tên là i và kq.

- Tiến hành tính toán (gán lại kết quả cho kq).

- Cuối cùng lấy kết quả trong kq gán cho ô nhớ f (là ô nhớ có sẵn đã được khai báo trước, nằm bên ngoài hàm).

- Kết thúc hàm quay về chương trình gọi. Do hàm LuyThua đã hoàn thành xong việc tính toán nên các ô nhớ được tạo ra trong khi thực hiện hàm (x, n, i, kq) sẽ được xóa khỏi bộ nhớ. Kết quả tính toán được lưu giữ trong ô nhớ f (không bị xóa vì không liên quan gì đến hàm).

Truyền đối số theo giá trị là cách truyền phổ biến. Tuy nhiên vấn đề đặt ra là giả sử ngoài việc tính f, ta còn muốn thay đổi các giá trị của các ô nhớ a, b (khi truyền nó cho hàm) thì có thể thực hiện được không? Để giải quyết vấn đề này ta cần phải truyền đối số bằng địa chỉ hoặc con trỏ. Hai cách truyền này có thể được gọi chung là *tham biến*.

5.2.8.2 Truyền bằng con trỏ

Xét ví dụ hoán đổi giá trị của 2 biến. Đây là một yêu cầu nhỏ nhưng được gặp nhiều lần trong chương trình, ví dụ để sắp xếp một mảng hay danh sách. Do vậy cần viết một hàm để thực hiện yêu cầu trên. Hàm không trả kết quả mà chỉ hoán đổi giá trị giữa 2 đối số. Do các biến cần hoán đổi là chưa được biết trước tại thời điểm viết hàm, nên ta phải đưa chúng vào hàm như các đối số, tức là hàm có 2 đối số (có thể đặt là x, y) đại diện cho các biến sẽ thay đổi giá trị sau này.

Từ các nhận xét trên, nếu hàm hoán đổi ngay sau đây sẽ không đáp ứng được yêu cầu.

```
void Swap(int x, int y)
{
    int t = x;
    x = y; y = t;
}
```

Hàm Swap trên không đáp ứng được yêu cầu vì giả sử trong chương trình chính ta có 2 biến x, y chứa các giá trị lần lượt là 2 và 5. Ta cần hoán đổi nội dung 2 biến này sao cho x = 5 còn y = 2 bằng cách gọi đến hàm Swap(x, y).

```
main()
{   int x = 2, y = 5;
    Swap(x, y);
    cout << "x = " << x << "va y = " << y << endl;
            // in ra x = 2 va y = 5 (x, y vẫn không đổi)
}
```

Thực tế sau khi chạy xong chương trình ta thấy giá trị của x và y vẫn không thay đổi !?. Như đã giải thích trong mục trên (gọi hàm LuyThua), việc đầu tiên khi chương trình thực hiện một hàm là tạo ra các biến mới (các ô nhớ mới, độc lập với các ô nhớ x, y đã có sẵn) tương ứng với các đối số, trong trường hợp này cũng có tên là x, y và gán nội dung của x, y (ngoài hàm) cho x, y (mới). Và việc cuối cùng của chương trình sau khi thực hiện xong hàm là xóa các biến mới này. Do vậy nội dung của các biến mới thực tế là có thay đổi, nhưng không ảnh hưởng gì đến các biến x, y cũ (ngoài hàm).

Như vậy ***hàm Swap cần được viết lại*** sao cho việc thay đổi giá trị không thực hiện trên các biến tạm mà phải thực hiện trên các biến ngoài. Muốn vậy thay vì truyền giá trị của các biến ngoài cho đối số, bây giờ ta sẽ truyền địa chỉ của nó cho đối số, và các thay đổi sẽ phải thực hiện trên nội dung của các địa chỉ này. Đó chính là lý do ta phải sử dụng con trỏ để làm đối số thay cho biến thường.

Cụ thể hàm Swap được viết lại như sau:

```
void Swap(int *p, int *q)
{
    int t = *p;
    *p = *q; *q = t;
}
```

Với cách tổ chức hàm như vậy rõ ràng nếu ta cho p trỏ tới biến x và q trỏ tới biến y thì hàm Swap sẽ làm thay đổi nội dung của x, y chứ không phải của p, q.

Từ đó lời gọi hàm sẽ là Swap(&x, &y) (tức truyền địa chỉ của x cho p để p trỏ tới x và tương tự q trỏ tới y).

5.2.8.3 Truyền bằng địa chỉ

Một hàm viết dưới dạng đối số được truyền bằng địa chỉ sẽ đơn giản hơn so với truyền bằng con trỏ và nó giống với cách truyền bằng giá trị hơn, trong đó chỉ có một điểm khác biệt đó là các đối số khai báo theo dạng địa chỉ.

Như vậy ***hàm Swap trên cũng có thể được viết*** theo cách truyền địa chỉ trực tiếp mà không thông qua con trỏ như sau:

```
void Swap(int &x, int &y)
{
    int t = x;
    x = y; y = t;
}
```

Và lời gọi hàm cũng đơn giản như cách truyền đối số bằng giá trị.

```
main()
{   int x = 2, y = 5;
    Swap(x, y);
    cout << "x = " << x << "va y = " << y << endl;
            // in ra x = 5 va y = 2 (x, y đã không đổi)
}
```

5.3 ĐỆ QUI

5.3.1 Khái niệm đệ qui

Một hàm gọi đến hàm khác là việc gọi hàm một cách thông thường, tuy nhiên nếu một hàm lại gọi đến chính bản thân nó thì ta gọi hàm là đệ qui. Khi thực hiện một hàm đệ qui, hàm sẽ phải thực hiện nhiều lần, trong mỗi lần chạy chương trình sẽ tạo nên một tập biến cục bộ mới trên ngăn xếp (các đối số, các biến riêng khai báo trong hàm) độc lập với lần chạy trước đó, từ đó dễ gây tràn ngăn xếp. Vì vậy đối với những bài toán có thể giải được bằng phương pháp lặp thì không nên dùng đệ qui.

Để minh họa ta hãy xét hàm tính n giai thừa. Để tính n! ta có thể dùng phương pháp lặp như sau:

```
#include <iostream>
using namespace std;
long GiaiThua(int n)
{    long kq = 1;
     for (int i = 1; i <= n; i++)
        kq = kq * i;
     return kq;
}
main()
{    int n;
     long kq;
     cout << "n = " ; cin >> n;
     kq = GiaiThua(n);
     cout << n << "! = " << kq << endl;
}
```

Mặt khác, n! cũng được tính thông qua (n-1)! bởi công thức truy hồi như sau:

n! = 1 nếu n = 0

n! = (n-1)! * n nếu n > 0

Do đó ta có thể xây dựng hàm đệ qui tính n! như sau:

```
#include <iostream>
using namespace std;
long GiaiThuaDQ(int n)
{
     if(n == 0)
         return 1;
     else
         return GiaiThuaDQ(n - 1) * n;
}
main()
{    int n;
     long kq;
     cout << "n = " ; cin >> n;
```

```
    kq = GiaiThuaDQ(n);
    cout << n << "! = " << kq << endl;
}
```

5.3.2 Các đặc điểm của hàm đệ qui

- Chương trình viết ngắn gọn.

- Việc thực hiện gọi đi gọi lại hàm nhiều lần phụ thuộc vào độ lớn của đầu vào. Chẳng hạn trong ví dụ trên hàm được gọi n lần, mỗi lần như vậy chương trình sẽ mất thời gian để lưu giữ các thông tin của hàm gọi trước khi chuyển điều khiển đến thực hiện hàm được gọi. Mặt khác các thông tin này được lưu giữ nhiều lần trong ngăn xếp sẽ tốn nhiều vùng nhớ và có thể dẫn đến tràn ngăn xếp nếu n lớn.

- Chương trình dễ viết và dễ đọc nhưng có thể khó hiểu. Trên thực tế có nhiều bài toán hầu như tìm một thuật toán lặp cho nó là rất khó trong khi viết theo thuật toán đệ qui thì lại rất dễ dàng.

5.3.3 Lớp các bài toán giải được bằng đệ qui

Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:

- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp suy biến hay cơ sở, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó.
- Đối với trường hợp tổng quát, bài toán có thể giải được bằng bài toán cùng dạng nhưng với đối số khác có kích thước hay giá trị nhỏ/lớn hơn đối số ban đầu. Và sau một số bước hữu hạn biến đổi cùng dạng, bài toán được đưa về trường hợp suy biến.

Như vậy trong trường hợp tính n! nếu n = 0 hàm cho ngay giá trị là 1 mà không cần phải gọi lại chính nó, đây chính là trường hợp suy biến. Trường hợp n > 0 hàm sẽ gọi lại chính nó nhưng với n giảm 1 đơn vị. Việc gọi này được lặp lại cho đến khi n = 0.

Một lớp rất rộng của bài toán dạng này là các bài toán có thể định nghĩa được dưới dạng đệ qui như các bài toán lặp với số bước hữu hạn biết trước, các bài toán UCLN, tháp Hà Nội, … Đặc biệt là các ứng dụng trong cấu trúc dữ liệu và giải thuật.

5.3.4 Cấu trúc chung của hàm đệ qui

Dạng thức chung của một chương trình đệ qui như sau:

```
if (trường hợp suy biến)
        { trình bày cách giải // thường là trả về kết quả }
else    // trường hợp tổng quát
        { gọi lại hàm với đối số "nhỏ/lớn" hơn }
```

5.3.5 Các ví dụ minh họa

Ví dụ 1: Tìm UCLN của 2 số nguyên dương a, b. Bài toán có thể được định nghĩa dưới dạng đệ qui như sau:

- Nếu a = b thì UCLN = a

- Nếu a < b thì UCLN(a, b) = UCLN(a, b - a)

- Nếu a > b thì UCLN(a, b) = UCLN(a - b, b)

Từ đó ta có chương trình đệ qui để tính UCLN của a và b như sau:

```
#include <iostream>
using namespace std;
int UCLN(int a, int b)
{
    if (a == b)
        return a;
    else
        if (a < b)
            return UCLN(a, b-a);
        else
            return UCLN(a-b, b);
}
main()
{    int a, b;
     cout << "Nhap 2 so nguyen duong a va b = " ;
     cin >> a >> b;
     cout << "UCLN(" << a << "," << b << ")= ";
     cout << UCLN(a,b) << endl;
}
```

Ví dụ 2: Tính số hạng thứ n của dãy Fibonacci là dãy f(n) được định nghĩa:

- f(1) = f(2) = 1

- f(n) = f(n - 1) + f(n - 2) với ∀ n > 2

Chương trình được viết như sau:

```
#include <iostream>
using namespace std;
long Fibo(int n)
{    long kq;
     if (n==1 || n==2)
        kq = 1;
     else
```

```
        kq = Fibo(n-1) + Fibo(n-2);
    return kq;
}
main()
{   int n;
    long kq;
    cout << "n = "; cin >> n;
    kq = Fibo(n);
    cout<<"So hang thu " <<n<<" trong day Fibonacci la ";
    cout << kq << endl;
}
```

Ví dụ 3: Viết chương trình tính $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$; $C_n^k = 1$ *nếu* $k = 0$ *hoặc* $k = n$

```
#include <iostream>
using namespace std;
long Tohop(int n, int k)
{
    if (k==0 || k==n)
        return 1;
    else
        return Tohop(n-1,k-1) + Tohop(n-1,k);
}
main()
{   int n, k;
    cout << "Nhap n va k : " ;
    cin >> n >> k;
    cout << "To hop " << n << " chap " << k << " = ";
    cout << Tohop(n,k) << endl;
}
```

Ví dụ 4: Viết chương trình in ra các số lẻ nhỏ hơn hoặc bằng n.

```
#include <iostream>
using namespace std;
void InLe(int n)
{
```

```
    if (n==1)
        cout << n;
    else
        if (n%2 != 0)
        {
            cout << n <<'\t';
            InLe(n-2);
        }
        else
            InLe(n-1);
}
main()
{    int n;
     cout << "Nhap n = " ;
     cin >> n ;
     cout << "Cac so le la : ";
     InLe(n);
     cout << endl;
}
```

✧ Hàm đệ qui InLe(int n) trên sẽ in các số lẻ theo thứ tự giảm dần. Nếu muốn in các số lẻ theo thứ tự tăng dần thì ta hiệu chỉnh hàm InLe lại như sau:

```
void InLe(int n, int i)
{
    if (n==1)
        cout << n;
    else
        if (i <= n)
        {
            cout << i <<'\t';
            InLe(n,i+2);
        }
}
```

Và khi đó ở lời gọi hàm này ta sẽ truyền giá trị cho đối số i là 1 (`InLe(n, 1);`)

BÀI TẬP CHƯƠNG 5

-----ଓ✧ଌ-----

1. Viết chương trình nhập vào chuỗi ký tự st (dạng con trỏ), in ra màn hình chuỗi đảo ngược.

2. Viết chương trình nhập vào chuỗi ký tự s(dạng con trỏ), hãy copy từ chuỗi s sang chuỗi t một đoạn bắt đầu tại vị trí m với độ dài n.

3. Xây dựng hàm tìm UCLN của 2 số nguyên. Áp dụng hàm này để tìm UCLN của 3 số nguyên được nhập từ bàn phím.

4. Xây dựng hàm tìm BCNN của 2 số nguyên. Áp dụng hàm này để tìm UCLN của 4 số nguyên được nhập từ bàn phím.

5. Xây dựng hàm kiểm tra một số nguyên dương n có là số nguyên tố. Áp dụng hàm này để tính tổng của K số nguyên tố đầu tiên, với K được nhập từ bàn phím.

6. Xây dựng hàm kiểm tra một năm có là năm nhuận không. Áp dụng hàm này in ra các năm nhuận từ năm 1000 đến 2035.

7. Xây dựng hàm chuẩn hoá một chuỗi bất kỳ về dạng Proper (cắt bỏ các khoảng trắng ở 2 đầu, cắt bớt các dấu trắng thừa ở giữa các từ (chỉ để lại 1), viết hoa đầu từ).

8. Xây dựng hàm đệ qui tính n!. Áp dụng hàm này để tính tổ hợp n chập k theo công thức truy hồi: C(n, k) = n!/(k! (n-k)!). Viết hàm main() minh họa tính đúng đắn của các hàm đã xây dựng.

9. Xây dựng các hàm sau bằng giải thuật không đệ qui:

   a. Kiểm tra xem số n có phải là số chính phương không?

   b. Tìm số lớn nhất trong 4 số thực.

   c. In ra n phần tử đầu tiên của dãy Fibonacci.

   d. Đếm xem có bao nhiêu số nguyên tố trong n số nguyên dương đầu tiên.

   e. In ra các ước số của số nguyên dương n.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

10. Xây dựng các hàm sau bằng giải thuật đệ qui:

   a. Tính tổng N số nguyên dương đầu tiên.

   b. Tính x^n với x là số thực còn n là số nguyên.

   c. Tìm bội số chung nhỏ nhất của 2 số nguyên dương.

   d. In ra các ước số của số nguyên dương n.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

11. Xây dựng các hàm sau cho dữ liệu kiểu mảng 1 chiều có n phần tử kiểu số nguyên:

a. Nhập mảng.

b. Xuất mảng.

c. Đếm số phần tử là số nguyên tố có trong mảng.

d. Tính tổng các phần tử là ước số của K.

e. Tìm BCNN của 2 phần tử đầu và cuối mảng.

f. Sắp xếp mảng theo thứ tự tăng dần.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

12. Xây dựng các hàm sau cho dữ liệu kiểu ma trận M hàng, N cột, phần tử kiểu số thực:

a. Nhập ma trận.

b. In ma trận.

c. Đếm số phần tử có phần nguyên là 0.

d. Tính tổng các phần tử nằm trên hàng H.

e. Cho biết vị trí của phần tử nhỏ nhất.

f. Sắp xếp các phần tử trên cột C theo thứ tự giảm dần.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

13. Xây dựng các hàm thực hiện các chức năng sau:

a. Trả về chiều dài của chuỗi.

b. Trả về vị trí của chuỗi stc trong chuỗi st.

c. Trả về chuỗi stc từ chuỗi st gồm num ký tự bắt đầu từ vị trí thứ pos.

d. Trả về chuỗi sau khi ghép 3 chuỗi lại với nhau.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

14. Xây dựng các hàm thực hiện các chức năng sau:

a. Chèn chuỗi stc vào chuỗi st bắt đầu từ vị trí thứ pos.

b. Xóa num ký tự trong chuỗi st bắt đầu từ vị trí thứ pos.

c. Đổi số nguyên thành chuỗi số tương ứng.

d. Đổi số thực thành chuỗi số tương ứng.

e. Đổi chuỗi số nguyên thành số tương ứng.

f. Đổi chuỗi số thực thành số tương ứng.

Viết hàm main() minh họa tính đúng đắn của các hàm trên.

CHƯƠNG 6: DỮ LIỆU KIỂU CẤU TRÚC

-----ଓ✧ଌ-----

6.1 ĐỊNH NGHĨA

Dữ liệu kiểu cấu trúc là một tập hợp gồm nhiều thành phần có thể thuộc nhiều kiểu dữ liệu khác nhau nhưng lại có liên quan với nhau. Dữ liệu kiểu cấu trúc không có sẵn mà người lập trình phải tự định nghĩa bằng từ khóa **struct**.

Kiểu cấu trúc còn được gọi là kiểu mẫu tin hay bản ghi (record) trong đó mỗi thành phần được gọi là 1 trường (field).

Ví dụ: Để quản lý sinh viên một lớp ta cần các thông tin như: mã số, họ tên, năm sinh, giới tính, điểm trung bình thì ta sẽ định nghĩa kiểu cấu trúc với các thành phần là các thông tin cần quản lý.

6.2 KHAI BÁO

Để tạo ra một kiểu cấu trúc người lập trình cần phải khai báo tên của kiểu (là một tên gọi do người lập trình tự đặt) và các thành phần dữ liệu có trong kiểu cấu trúc này.

Một kiểu cấu trúc được khai báo theo mẫu sau:

struct <tên kiểu cấu trúc>

{

thành phần 1;

thành phần 2;

…

thành phần n;

} [danh sách các biến];

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần còn được gọi là trường (field). Nếu các thành phần có cùng kiểu dữ liệu thì ta có thể khai báo chung.

- Phần danh sách các biến có thể có hoặc không. Tuy nhiên trong khai báo ký tự kết thúc cuối cùng phải là dấu chấm phẩy (;) phải có.

- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có kiểu là kiểu cấu trúc khác.

- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thành phần của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

- Ta cũng có thể khai báo biến kiểu cấu trúc giống như khai báo các biến kiểu cơ sở dưới dạng sau:

<tên kiểu cấu trúc> <danh sách biến>;

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên kiểu cấu trúc> <biến> = { giá trị khởi tạo };

Ví dụ:

- Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số:

```
struct Phanso
{   int tu;
    int mau;
} a, b, c;
```

Ta có thể khai báo bằng cách khác như sau:

```
struct Phanso
{
    int tu, mau;
};
Phanso a, b, c;
```

- Để quản lý sinh viên một lớp ta khai báo như sau:

```
struct Sinhvien
{
    char ms[9], ht[40];
    int ns, gt;
    float dtb;
};
```

- Kiểu Date gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Date
{   int ngay;
    int thang;
    int nam;
} holiday = {31, 12, 2013};
```

Biến holiday cũng được khai báo kèm theo kiểu này và được khởi tạo bởi bộ giá trị 31, 12, 2013. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 31, th = 12 và nam = 2013.

6.3 CÁCH SỬ DỤNG

Tùy theo cách khai báo biến kiểu cấu trúc mà ta có thể truy xuất đến các thành phần của nó bằng các cách khác nhau.

6.3.1 Đối với biến thường

<tên biến> . <tên thành phần>

Ví dụ:

```
struct Sinhvien
{
    char ms[9], ht[40];
    int ns, gt;
    float dtb;
};
Sinhvien sv1, sv2;
strcpy(sv1.ht, "Nguyen Thanh Liem");
sv1.dtb = 5.5;
sv2.dtb = sv1.dtb + 1;
cout<<sv1.ht<<'\t'<<sv2.ht<<endl;
cout<<sv1.dtb<<'\t'<<sv2.dtb<<endl;
```

6.3.2 Đối với biến con trỏ

<tên biến> → <tên thành phần>

Ví dụ:

```
struct Sinhvien
{
     char ms[9], ht[40];
     int ns, gt;
     float dtb;
}x, *p;
Sinhvien y = {"CNTT3801", "Le Trung Truc", 1995, 1, 5.5};
x.ns = y.ns;
p = new Sinhvien;
strcpy(p->ht, y.ht);
p->ns = x.ns - 1;
p->dtb = y.dtb + 2;
cout<<p->ht<<'\t'<<p->ns<<'\t'<<p->dtb<<endl;
```

6.3.3 Đối với biến mảng

Ta phải truy xuất thành phần mảng trước rồi đến thành phần cấu trúc sau.

Ví dụ:

```
struct Sinhvien
{
     char ms[9], ht[40];
     int ns, gt;
     float dtb;
}cntt13[80];
Sinhvien y = {"CNTT3801", "Le Trung Truc", 1995, 1, 5.5};
strcpy(cntt13[1].ht, "Nguyen Quoc An");
cntt13[1].ns = y.ns;
cntt13[1].dtb = 9.5;
cout<<cntt13[1].ht<<'\t'<<cntt13[1].ns<<'\t';
cout<<cntt13[1].dtb<<endl;
```

6.3.4 Đối với cấu trúc lồng nhau

Ta phải truy xuất thành phần của cấu trúc ngoài trước rồi đến thành phần của cấu trúc trong sau; và ta cũng phải sử dụng các phép toán **.** hoặc → (các phép toán lấy thành phần) một cách thích hợp.

Ví dụ:

```
struct Date
{
     int ngay, thang, nam;
};
struct Sinhvien
{
     char ms[9], ht[40];
     date ngaysinh;
     float dtb;
};
Sinhvien sv3 = {"CNTT3803", "Phan Hanh Phuc",
                            {27, 12, 1994}, 5.5};
cout<<"Sinh vien "<<sv3.ht<<" co ngay sinh la : ";
cout<<sv3.ngaysinh.ngay<<'/'<<sv3.ngaysinh.thang<<'/';
cout<<sv3.ngaysinh.nam<<endl;
```

6.4 CÁC THAO TÁC CƠ BẢN

6.4.1 Nhập, xuất dữ liệu

Cũng giống như các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Chẳng hạn muốn nhập, xuất một biến cấu trúc ta phải viết các câu lệnh nhập, xuất cho từng thành phần của biến đó.

Ví dụ:

```
struct Sinhvien
{
     char ms[9], ht[40];
     int ns, gt;
     float dtb;
} x;
cout << " Nhap du lieu cho sinh vien x:" << endl;
cout << "Nhap ma so:"; cin.getline(x.ms, 9);
cout << "Nhap ho ten:"; cin.getline(x.ht, 40);
cout << "Nhap nam sinh:"; cin >> x.ns;
cout << "Nhap gioi tinh:"; cin >> x.gt;
cout << "Nhap diem trung binh:"; cin >> x.dtb;
cout << "Thong tin ve sinh vien vua nhap la:" << endl;
cout << "Ma so: " << x.ms << endl;
cout << "Ho ten: " << x.ht << endl;
cout << "Nam sinh: " << x.ns << endl;
cout << "Gioi tinh: " << (x.gt == 1 ? "Nam" : "Nu") << endl;
cout << "Diem trung binh: " << x.dtb << endl;
```

6.4.2 Gán dữ liệu

Khác với biến mảng, đối với biến cấu trúc chúng ta có thể gán giá trị của 2 biến cấu trúc cho nhau. Phép gán này cũng tương đương với việc gán từng thành phần của 2 biến cấu trúc.

Ví dụ:

```
struct Sinhvien
{
     char ms[9], ht[40];
```

```
    int ns, gt;
    float dtb;
} x, y, *p;
cout << " Nhap du lieu cho sinh vien x:" << endl;
cout << "Nhap ma so:"; cin.getline(x.ms, 9);
cout << "Nhap ho ten:"; cin.getline(x.ht, 40);
cout << "Nhap nam sinh:"; cin >> x.ns;
cout << "Nhap gioi tinh:"; cin >> x.gt;
cout << "Nhap diem trung binh:"; cin >> x.dtb;
y = x;
cout << "Thong tin ve sinh vien y la:" << endl;
cout << "Ma so: " << y.ms << endl;
cout << "Ho ten: " << y.ht << endl;
cout << "Nam sinh: " << y.ns << endl;
cout << "Gioi tinh: " << (y.gt == 1 ? "Nam" : "Nu") << endl;
cout << "Diem trung binh: " << y.dtb << endl;
p = new Sinhvien;
*p = y;
cout << "Thong tin ve sinh vien p la:" << endl;
cout << "Ma so: " << p->ms << endl;
cout << "Ho ten: " << p->ht << endl;
cout << "Nam sinh: " << p->ns << endl;
cout << "Gioi tinh:" << (p->gt == 1 ? "Nam" : "Nu") << endl;
cout << "Diem trung binh: " << p->dtb << endl;
```

✧ ***Chú ý:*** Ta không thể gán bộ giá trị cụ thể cho biến cấu trúc được, cách gán này chỉ thực hiện được khi khởi tạo.

Ví dụ:

```
Sinhvien y = {"CNTT3801", "Le Trung Truc", 1995, 1, 5.5};
```

là đúng.

Nhưng:

```
Sinhvien z;
```

`z = {"CNTT3801", "Le Trung Truc", 1995, 1, 5.5};` là sai.

Và:

`z = y;` là đúng.

6.5 CÁC VÍ DỤ MINH HỌA

6.5.1 Ví dụ 1

Viết chương trình nhập vào 2 phân số, tính hiệu và thương của 2 phân số đó.

```
#include <iostream>
using namespace std;
main()
{
    struct Phanso
    {
        int tu;
        int mau;
    } a, b, h, t;
    cout<<"Nhap tu so cua phan so a:"; cin>>a.tu;
    cout<<"Nhap mau so cua phan so a:"; cin>>a.mau;
    cout<<"Nhap tu so cua phan so b:"; cin>>b.tu;
    cout<<"Nhap mau so cua phan so b:"; cin>>b.mau;
    cout<<"Phan so a da nhap la:"<<a.tu<<'/'<<a.mau<<endl;
    cout<<"Phan so b da nhap la:"<<b.tu<<'/'<<b.mau<<endl;
    h.tu=a.tu*b.mau-a.mau*b.tu;
    h.mau=a.mau*b.mau;
    cout<<"Hieu cua 2 phan so a va b la:";
    cout<<h.tu<<'/'<<h.mau<<endl;
    t.tu=a.tu*b.mau;
    t.mau=a.mau*b.tu;
    cout<<"Thuong cua 2 phan so a va b la:";
    cout<<t.tu<<'/'<<t.mau<<endl;
}
```

Tuy nhiên ta có thể khai báo kiểu cấu trúc bên ngoài hàm main() và để đơn giản hơn ta có thể xây dựng các hàm để thực hiện các công việc cụ thể trong chương trình như: nhập/xuất phân số, tính hiệu và thương 2 phân số. Khi đó chương trình 1 được viết lại như sau:

```
#include <iostream>
using namespace std;
struct Phanso
{
    int tu;
    int mau;
};
void NhapPS(Phanso &p)
{
    cout<<"Nhap tu so :"; cin>>p.tu;
    cout<<"Nhap mau so :"; cin>>p.mau;
}
void XuatPS(Phanso p)
{
    cout<<p.tu<<'/'<<p.mau<<endl;
}
Phanso Hieu(Phanso a, Phanso b)
{
    Phanso h;
    h.tu=a.tu*b.mau-a.mau*b.tu;
    h.mau=a.mau*b.mau;
    return h;
}
Phanso Thuong(Phanso a, Phanso b)
{
    Phanso t;
    t.tu=a.tu*b.mau;
    t.mau=a.mau*b.tu;
    return t;
}
```

```
main()
{   Phanso a, b;
    cout<<"Nhap phan so a:"<<endl;
    NhapPS(a);
    cout<<"Nhap phan so b:"<<endl;
    NhapPS(b);
    cout<<"Phan so a la: "; XuatPS(a);
    cout<<"Phan so b la: "; XuatPS(b);
    cout<<"Hieu cua 2 phan so a va b la: ";
    XuatPS(Hieu(a, b));
    cout<<"Thuong cua 2 phan so a va b la: ";
    XuatPS(Thuong(a, b));
}
```

6.5.2 Ví dụ 2

Viết chương trình quản lý sách cho 1 thư viện. Thông tin về mỗi quyển sách gồm: tựa sách, tên tác giả, tên nhà xuất bản, năm xuất bản và số trang của nó. Nhập danh sách n quyển sách, với n nhập từ bàn phím. In danh sách các quyển sách đã nhập. Cho biết có bao nhiêu quyển sách của tác giả X, với X nhập từ bàn phím. Tính tổng số trang của các quyển sách được xuất bản vào năm 2013. Cho biết quyển sách dày nhất có bao nhiêu trang. In ra tựa sách, tên tác giả của những quyển sách có số trang là P, với P nhập từ bàn phím.

```
#include <iostream>
using namespace std;
struct sach
{
    char tua[40], ttg[40], nxb[60];
    int namxb, trang;
} s[1000];
main()
{   int n, i, P;
    char X[40];
    //Nhap va kiem tra n
    do
    {
```

```
    cout<<"Nhap so sach co trong thu vien : ";
    cin>>n;
}while (n<=0 || n>1000);
//Nhap sach
for (i=0; i<n; i++)
{
    cout<<"Nhap quyen sach thu "<<i+1<<" : "<<endl;
    cin.ignore(1);
    cout<<"Nhap tua sach :"; cin.getline(s[i].tua,40);
    cout<<"Nhap ten tac gia :";
    cin.getline(s[i].ttg,40);
    cout<<"Nhap ten nha xuat ban :";
    cin.getline(s[i].nxb,60);
    cout<<"Nhap nam xuat ban :";
    cin>>s[i].namxb;
    cout<<"Nhap so trang :";
    cin>>s[i].trang;
}
//In cac quyen sach da nhap
for (i=0; i<n; i++)
{
    cout<<"Thong tin ve qs thu "<<i+1<< " la : "<<endl;
    cout<<"Tua : "<<s[i].tua<<endl;
    cout<<"Tac gia : "<<s[i].ttg<<endl;
    cout<<"Nha xuat ban : "<<s[i].nxb<<endl;
    cout<<"Nam xuat ban : "<<s[i].namxb<<endl;
    cout<<"So trang : "<<s[i].trang<<endl;
}
//Dem so sach duoc viet boi tac gia X
cin.ignore(1);
cout<<"Ban muon dem so sach cua tac gia nao? ";
cin.getline(X,40);
```

```
    int d=0;
    for (i=0; i<n; i++)
        if (strcmpi(s[i].ttg,X)==0) d++;
    cout<<"Tac gia "<<X<<" co "<<d;
    cout<<" quyen sach trong thu vien"<<endl;
    //Tinh tong so trang cua cac qs duoc xb trong nam 2013
    long ts=0;
    for (i=0; i<n; i++)
        if (s[i].namxb==2013)
            ts=ts+s[i].trang;
    cout<<"Tong so trang cac qs duoc xb nam 2013 la : ";
    cout<<ts<<endl;
    //Tim so trang cua quyen sach day nhat
    int pmax=s[0].trang;
    for (i=1; i<n; i++)
        if (pmax<s[i].trang)
            pmax=s[i].trang;
    cout<<"Quyen sach day nhat co "<<pmax<<" trang"<<endl;
    //In tua va tac gia cua cac quyen sach co so trang la P
    cout<<"Nhap so trang cua cac quyen sach can in : ";
    cin>>P;
    cout<<"Cac quyen sach co so trang la "<<P<<" :"<<endl;
    for (i=0; i<n; i++)
        if (s[i].trang==P)
        {
            cout<<"Tua sach : "<<s[i].tua<<endl;
            cout<<"Tac gia : "<<s[i].ttg<<endl;
        }
}
```

6.5.3 Ví dụ 3

Viết chương trình nhập vào danh sách sinh viên của 1 lớp, có tối đa 80 sinh viên. Mỗi sinh viên gồm các thông tin: mã số, họ tên, quê quán, giới tính, năm sinh, điểm trung bình và xếp loại. Kiểm tra dữ liệu nhập: mã số phải đủ 6 ký tự trong đó 3 ký tự đầu là

chữ cái và 3 ký tự cuối là chữ số, họ tên chỉ toàn chữ cái và khoảng trắng nhưng không toàn là khoảng trắng, năm sinh từ 1985 đến 1995, điểm trung bình từ 0.0 đến 4.0 và xếp loại dựa vào điểm trung bình. In ra danh sách các sinh viên đã nhập. Cho biết trong lớp có bao nhiêu sinh viên nữ. In ra họ tên các sinh viên có điểm trung bình cao nhất. Sắp xếp danh sách sinh viên giảm dần theo điểm trung bình.

```
#include <iostream>
using namespace std;
struct Sinhvien
{
    char ms[7], ht[40], qq[20], xl[11];
    int gt, ns;
    float dtb;
} sv[80];
main()
{   int n,i,j,p,d1,d2,dc,dt,dnu;
    float dmax;
    Sinhvien tam;
    do
    {  cout<<"Nhap so luong sinh vien : ";
       cin>>n;
    }while (n <=0 || n > 80);
    for (i=0; i<n; i++)
    {
       cout<<"Nhap sinh vien thu "<<i+1<<" : "<<endl;
       cin.ignore(1);
       do
       {   cout<<"Nhap ma so:";
           cin.getline(sv[i].ms,7);
           d1=0;
           for (j=0; j<3; j++)
               if (isalpha(sv[i].ms[j])) d1++;
           d2=0;
           for (j=3; j<6; j++)
```

```
            if (isdigit(sv[i].ms[j])) d2++;
    }while(strlen(sv[i].ms)!=6 || d1!=3 || d2!=3);
    do
    {   cout<<"Nhap ho ten:";
        cin.getline(sv[i].ht,40);
        dc=0; dt=0;
        for (j=0; j<strlen(sv[i].ht); j++)
           if (isalpha(sv[i].ht[j]))dc++;
           else if (isspace(sv[i].ht[j]))dt++;
    }while(dc+dt!=strlen(sv[i].ht) ||dc==0);
    cout<<"Nhap que quan:";
    cin.getline(sv[i].qq,20);
    cout<<"Nhap gioi tinh (Nam:1, Nu:0):";
    cin>>sv[i].gt;
    do
    {   cout<<"Nhap nam sinh:";
        cin>>sv[i].ns;
    }while (sv[i].ns<1985 || sv[i].ns>1995);
    do
    {   cout<<"Nhap diem trung binh:";
        cin>>sv[i].dtb;
    }while (sv[i].dtb<0 || sv[i].dtb>4);
    if (sv[i].dtb>=3.6)
        strcpy(sv[i].xl,"Xuat sac");
    else if (sv[i].dtb>=3.2)
             strcpy(sv[i].xl,"Gioi");
         else if (sv[i].dtb>=2.5)
                  strcpy(sv[i].xl,"Kha");
              else if (sv[i].dtb>=2)
                       strcpy(sv[i].xl,"Trung binh");
                   else strcpy(sv[i].xl,"Khong dat");
}
```

```
for (i=0; i<n; i++)
{
    cout<<"Thong tin ve sv thu "<<i+1<<" la : "<<endl;
    cout<<"Ma so : "<<sv[i].ms<<endl;
    cout<<"Ho ten : "<<sv[i].ht<<endl;
    cout<<"Que quan : "<<sv[i].qq<<endl;
    cout<<"Gioi tinh : ";
    if (sv[i].gt==0) cout<<"Nu"<<endl;
    else cout<<"Nam"<<endl;
    cout<<"Nam sinh : "<<sv[i].ns<<endl;
    cout<<"Diem trung binh : "<<sv[i].dtb<<endl;
    cout<<"Xep loai : "<<sv[i].xl<<endl;
}
dnu=0;
for (i=0; i<n; i++)
    if (sv[i].gt==0) dnu++;
cout<<"Trong lop co "<<dnu<<" sinh vien nu";
dmax=sv[0].dtb;
for (i=1; i<n; i++)
    if (dmax<sv[i].dtb)
       dmax=sv[i].dtb;
cout<<"Ho ten cac sinh vien co dtb cao nhat la : ";
for (i=0; i<n; i++)
    if (sv[i].dtb==dmax)
       cout<<sv[i].ht<<endl;
for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
       if (sv[i].dtb<sv[j].dtb)
       {
           tam=sv[i];
           sv[i]=sv[j];
           sv[j]=tam;
       }
```

```
    cout<<"Danh sach sv sau khi sap xep la : "<<endl;
    for (i=0; i<n; i++)
    {
        cout<<"Thong tin ve sinh vien thu "<<i+1<<" la : ";
        cout<<"Ma so : "<<sv[i].ms<<endl;
        cout<<"Ho ten : "<<sv[i].ht<<endl;
        cout<<"Que quan : "<<sv[i].qq<<endl;
        cout<<"Gioi tinh : ";
        if (sv[i].gt==0) cout<<"Nu"<<endl;
        else cout<<"Nam"<<endl;
        cout<<"Nam sinh : "<<sv[i].ns<<endl;
        cout<<"Diem trung binh : "<<sv[i].dtb<<endl;
        cout<<"Xep loai : "<<sv[i].xl<<endl;
    }
}
```

BÀI TẬP CHƯƠNG 6

-----ଓ✧ଌ-----

1. Một phân số gồm 2 trường là tử số và mẫu số. Viết chương trình nhập vào 2 phân số. In ra kết quả của các phép toán cộng, trừ, nhân và chia trên 2 phân số đó.

2. Giả sử để quản lý thông tin về họ tên, ngày sinh (phải bao gồm ngày, tháng, năm sinh) người ta định nghĩa 2 cấu trúc là NgaySinh và SinhVien. Từ các cấu trúc đó hãy viết chương trình nhập N sinh viên, in thông tin về các sinh viên, cho biết sinh viên nào sinh vào ngày giữa 15, đếm số sinh viên sinh vào mùa xuân (tháng 1, 2, 3), in họ tên sinh viên có tuổi đời trẻ nhất.

3. Viết chương trình quản lý học sinh của một lớp. Mỗi học sinh gồm các thông tin: họ tên, năm sinh, điểm HK1, điểm HK2. In ra họ tên, năm sinh và điểm cả năm của tất cả các học sinh, với điểm cả năm = (điểm HK1 + điểm HK2)/2. In ra tuổi lớn nhất và nhỏ nhất của lớp. Cho biết học sinh nào có điểm cả năm lớn nhất. Tìm điểm trung bình cả năm của lớp. Sắp xếp danh sách giảm dần theo điểm cả năm, nếu trùng thì sắp xếp giản dần theo điểm HK2. In ra danh sách sau khi sắp xếp.

4. Viết chương trình nhập vào N hóa đơn bán hàng, N không quá 100. Mỗi hóa đơn gồm các thông tin: mã số hóa đơn, tên người mua, tên hàng, loại hàng, đơn giá và số lượng. Khi nhập cần kiểm tra các điều kiện sau: mã số phải đủ 4 ký tự trong đó 2 ký tự đầu là chữ cái và 2 ký tự cuối la chữ số, tên người mua chỉ gồm chữ cái và khoảng trắng nhưng không toàn là khoảng trắng và sau khi nhập đổi sang dạng chuẩn proper, tên hàng chỉ gồm chữ cái và chữ số nhưng phải có ít nhất là 2 ký tự và sau khi nhập đổi sang dạng chuẩn upper, loại hàng chỉ nhận một trong ba giá trị là 1, 2, hoặc 3, đơn giá và số lượng phải dương. Cho biết thứ tự của hóa đơn có thành tiền cao nhất với thành tiền = số lượng * đơn giá. Cho biết số lượng trung bình của tấc cả các hóa đơn. Tính tổng thành tiền bán được theo từng loại hàng.

5. Viết chương trình quản lý nhân viên trong một cơ quan, mỗi nhân viên thông tin: mã số, họ tên, quê quán, ngày sinh, chức vụ, phụ cấp chức vụ, hệ số lương, tạm ứng và thực lĩnh. Chương trình cần thực hiện được các chức năng sau:

- Nhập danh sách nhân viên có kiểm tra dữ liệu nhập

  + Mã số gồm 6 ký số nhưng phải được tạo thành ít nhất từ 2 ký số khác nhau.

  + Họ tên chỉ gồm chữ cái và khoảng trắng nhưng không toàn là khoảng trắng và sau khi nhập đổi sang dạng chuẩn upper.

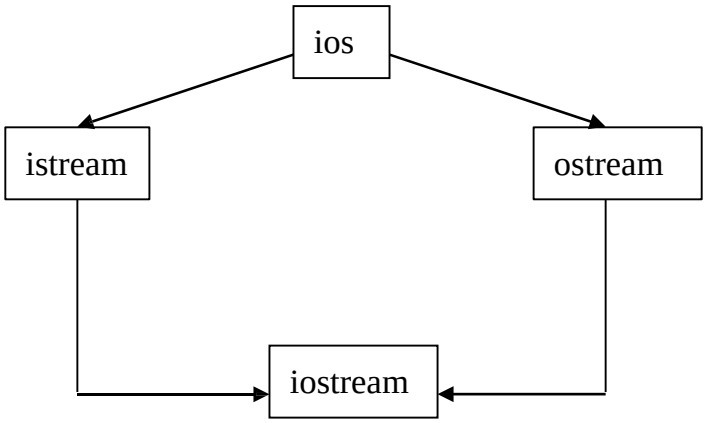
  + Hệ số lương từ 2.1 đến 6.8, tạm ứng phải không âm.
- In danh sách nhân viên.
- Tính thực lĩnh cho các nhân viên,
  với thực lĩnh = hệ số lương * 1.210.000 + phụ cấp chức vụ - tạm ứng.
- In họ tên nhân viên có hệ số lương cao nhất.
- Đếm số nhân viên có thực lĩnh trên 5000000.
- Tính tổng tạm ứng của tất cả những nhân viên có chức vụ được nhập từ bàn phím.
- Sắp xếp giảm dần theo thực lĩnh.
- In ra danh sách nhân viên sau khi sắp xếp.

CHƯƠNG 7: DỮ LIỆU KIỂU TẬP TIN

-----ଓ✧ଌ-----

7.1 GIỚI THIỆU

Trong C++ có sẵn một số lớp chuẩn chứa dữ liệu và các phương thức phục vụ cho các thao tác nhập/xuất dữ liệu của NLT, thường được gọi chung là ***stream*** (luồng/dòng). Trong số các lớp này, lớp có tên **ios** là lớp cơ sở, chứa các thuộc tính để định dạng việc nhập/xuất và kiểm tra lỗi. Mở rộng (kế thừa) lớp này có các lớp **istream, ostream** cung cấp thêm các toán tử nhập/xuất như >>, << và các hàm get, getline, read, ignore, put, write, flush … Một lớp rộng hơn có tên **iostream** là tổng hợp của 2 lớp trên. Bốn lớp nhập/xuất cơ bản này được khai báo trong các file tiêu đề có tên tương ứng (với đuôi *.h). Sơ đồ thừa kế của 4 lớp trên được thể hiện qua *hình 7.1*.



Hình 7.1: Sơ đồ thừa kế 4 lớp

Đối tượng của các lớp trên được gọi là các *luồng* dữ liệu. Một số đối tượng thuộc lớp iostream đã được khai báo sẵn (*chuẩn*) và được gắn với những thiết bị nhập/xuất cố định như các đối tượng **cin, cout, cerr, clog** gắn với bàn phím (cin) và màn hình (cout, cerr, clog). Điều này có nghĩa các toán tử >>, << và các hàm kể trên khi làm việc với các đối tượng này sẽ cho phép NLT nhập dữ liệu thông qua bàn phím hoặc xuất kết quả thông qua màn hình.

Để nhập/xuất thông qua các thiết bị khác (như máy in, file trên đĩa …), C++ cung cấp thêm các lớp **ifstream, ofstream, fstream** cho phép NLT khai báo các đối tượng mới gắn với thiết bị và từ đó nhập/xuất thông qua các thiết bị này.

Trong chương này, chúng ta sẽ xét các đối tượng thuộc các lớp **ifstream, ofstream, fstream** để làm việc với tập tin.

Làm việc với một tập tin (tệp/file) trên đĩa cũng được quan niệm như làm việc với các thiết bị khác của máy tính (chẳng hạn như làm việc với bàn phím, màn hình cùng với đối tượng chuẩn cin, cout, …). Các đối tượng này được khai báo thuộc lớp ifstream hay ofstream tùy thuộc vào việc ta muốn sử dụng file để đọc hay ghi.

Như vậy, để sử dụng một file dữ liệu đầu tiên chúng ta cần tạo đối tượng và gắn cho file này. Để tạo đối tượng có thể sử dụng các hàm tạo có sẵn trong hai lớp ifstream và ofstream. Đối tượng sẽ được gắn với tên file cụ thể trên đĩa ngay trong quá trình tạo đối tượng (tạo đối tượng với tham số là tên file) hoặc cũng có thể được gắn với tên file sau này bằng câu lệnh mở file. Sau khi đã gắn một đối tượng với file trên đĩa thì ta có thể sử dụng đối tượng này như cin, cout. Điều này có nghĩa là trong các câu lệnh in ra màn hình chỉ cần thay từ khóa cout bởi tên đối tượng, khi đó mọi dữ liệu cần in trong câu lệnh sẽ được ghi lên file mà đối tượng đại diện. Cũng tương tự nếu thay cin bởi tên đối tượng thì dữ liệu sẽ được đọc vào từ file thay cho từ bàn phím. Để tạo đối tượng dùng cho việc ghi ta khai báo chúng với lớp *ofstream* còn để dùng cho việc đọc ta khai báo chúng với lớp *ifstream*.

7.2 CÁC THAO TÁC CƠ BẢN

7.2.1 Tạo đối tượng gắn với tập tin

Mỗi lớp ifstream và ofstream cung cấp 4 phương thức để tạo tập tin. Ở giáo trình này chỉ giới thiệu 2 phương thức (2 cách) thường được sử dụng.

✧ ***Cách 1:*** **<Lớp> <Đối_tượng>;**

<Đối_tượng> . open(Tên_tập_tin, Chế_độ);

Lớp là một trong hai lớp ifstream và ofstream. Đối tượng là tên do NLT tự đặt. Chế độ là cách thức làm việc với tập tin được cho trong *bảng 7.1* sau:

Chế độ	Cách thức làm việc
ios::binary	quan niệm tập tin theo kiểu nhị phân, ngầm định là kiểu văn bản
ios::in	tập tin để đọc (ngầm định với đối tượng trong ifstream)
ios::out	tập tin để ghi (ngầm định với đối tượng trong ofstream), nếu tập tin đã có trên đĩa thì nội dung của nó sẽ bị ghi đè (bị xóa)
ios::app	bổ sung vào cuối tập tin
ios::trunc	xóa nội dung tập tin đã có
ios::ate	chuyển con trỏ đến cuối tập tin
ios::nocreate	không làm gì nếu tập tin chưa có

Bảng 7.1: Các chế độ làm việc trên tập tin

Cách này cho phép tạo trước một đối tượng chưa gắn với tập tin cụ thể nào. Sau đó dùng tiếp phương thức open để đồng thời mở tập tin và gắn với đối tượng vừa tạo.

Ví dụ: ifstream fi; // tạo đối tượng có tên fi để đọc

ofstream fo; // tạo đối tượng có tên fo để ghi

fi.open(“Songuyen.txt”); // mở tập tin Songuyen.txt và gắn với fi

fo.open(“Baitap.txt”); // mở tập tin Baitap.txt và gắn với fo

✧ ***Cách 2:*** **<Lớp> <Đối_tượng>(Tên_ tập_tin, Chế_độ)**

Cách này cho phép đồng thời mở tập tin cụ thể và gắn tập tin với tên đối tượng trong cùng câu lệnh.

Ví dụ: ifstream fi(“Songuyen.txt”); //mở tập tin Songuyen.txt gắn với đối tượng fi để đọc

ofstream fo(“Baitap.txt”); // mở tập tin Baitap.txt gắn với đối tượng fi để ghi

Sau khi mở tập tin và gắn với đối tượng (fi, fo), mọi thao tác trên đối tượng (fi, fo) cũng chính là làm việc với tập tin (Songuyen.txt, Baitap.txt).

Trong các câu lệnh trên các chế độ ở trạng thái ngầm định. Ta có thể chỉ định cùng lúc nhiều chế độ bằng cách ghi chúng liên tiếp nhau với toán tử hợp bit |.

Ví dụ: Để mở tập tin Baitap.txt như một file nhị phân và ghi tiếp theo vào cuối file ta dùng câu lệnh: ofstream f(“Baitap.txt”, ios::binary | ios::app);

7.2.2 Đóng tập tin và giải phóng đối tượng

Để đóng tập tin được đại diện bởi f, sử dụng phương thức close như sau:

<Đối_tượng> . close();

Sau khi đóng tập tin (và giải phóng mối liên kết giữa đối tượng với tập tin) ta có thể sử dụng lại đối tượng đó để gắn và làm việc với tập tin khác bằng phương thức open.

7.2.3 Kiểm tra sự tồn tại của tập tin, kiểm tra kết thúc tập tin

Việc mở một tập tin chưa có để đọc sẽ gây ra lỗi và làm dừng chương trình. Khi xảy ra lỗi mở tập tin, giá trị trả về của phương thức ***good()*** là 0. Vì thế ta có thể sử dụng phương thức này để kiểm tra một tập tin đã có trên đĩa hay chưa.

Ví dụ:

```
ifstream f("Baitap.txt");
if (!f.good())
{
    cout << "File Baitap.txt chua ton tai";
    getch();
    exit(1);
}
```

Khi đọc hoặc ghi, con trỏ tập tin sẽ chuyển dần về cuối tập tin. Khi con trỏ ở cuối tập tin, phương thức ***eof()*** sẽ trả về giá trị khác không. Do đó có thể sử dụng phương thức này để kiểm tra đã kết thúc tập tin hay chưa.

Ví dụ:

```
int dodai = 0;
while (!f.eof())
{
    f.get(ch));
    dodai++;
}
cout << "Do dai tap tin = " << dodai;
```

7.2.4 Đọc và ghi đồng thời trên tập tin

Để đọc và ghi đồng thời, tập tin phải được gắn với đối tượng của lớp fstream là lớp thừa kế của 2 lớp ifstream và ofstream. Khi đó chế độ phải được bao gồm chỉ định ios::in | ios::out.

Ví dụ: fstream f("Data.dat", ios::in | ios::out);

hoặc: fstream f;

f.open("Data.dat", ios::in | ios::out);

7.2.5 Di chuyển con trỏ tập tin

Các phương thức sau cho phép làm việc trên đối tượng của dòng xuất (ofstream).

<Đối_tượng> . seekp(n): Di chuyển con trỏ đến byte thứ n (các byte được tính từ 0)

<Đối_tượng> . seekp(n, vị trí xuất phát): Di chuyển đi n byte (có thể là số âm hoặc dương) từ vị trí xuất phát. Vị trí xuất phát có thể là:

- ios::beg: từ đầu tập tin
- ios::end: từ cuối tập tin
- ios::cur: từ vị trí hiện tại của con trỏ.

<Đối_tượng> . tellp( ): Cho biết vị trí hiện tại của con trỏ.

Để làm việc với dòng nhập tên các phương thức trên được thay thế tương ứng bởi các tên: **seekg** và **tellg**. Đối với các dòng nhập lẫn xuất có thể sử dụng được cả 6 phương thức trên.

Ví dụ:

```
fstream f("Baitap.txt");
f.seekg(0, ios::end);
cout << "Do dai tap tin = " << f.tellg();
```

7.3 TẬP TIN VĂN BẢN

7.3.1 Khái niệm

Trong tập tin văn bản mỗi byte được xem là một ký tự. Tuy nhiên nếu 2 byte 10 (LF), 13 (CR) đi liền nhau thì chỉ được xem là một ký tự và nó chính là ký tự xuống dòng. Như vậy tập tin văn bản là một tập hợp các dòng ký tự với ký tự xuống dòng có mã là 10. Ký tự có mã 26 được xem là ký tự kết thúc tập tin.

7.3.2 Các ví dụ minh họa

7.3.2.1 Ví dụ 1

Viết chương trình nhập vào N số nguyên từ bàn phím và ghi vào file Songuyen.txt, các số được ghi cách nhau 1 dấu cách. Sau đó in nội dung file này lên màn hình.

```
#include <iostream>
#include <fstream>
using namespace std;
main()
{    fstream f;
     int n, x;
     cout << "Ban muon nhap bao nhieu so nguyen: ";
     cin >> n;
     f.open("Songuyen.txt",ios::in|ios::out|ios::trunc);
     for (int i = 1; i <= n - 1; i++)
     {
          cin >> x;
          f << x << ' ';
     }
     cin >> x;
     f << x;
     cout << "Noi dung cua file Songuyen.txt la:" << endl;
     f.seekg(0);
     while(!f.eof())
     {    f >> N;
          cout << N << endl;
     }
     f.close();
}
```

7.3.2.2 Ví dụ 2

Viết chương trình nhập danh sách n nhân viên gồm các thông tin mã số, họ tên, năm sinh và tiền lương. Lưu vào file có tên được nhập từ bàn phím, trong đó dòng đầu tiên trong file ghi số lượng nhân viên, các dòng tiếp theo ghi thông tin của các nhân viên gồm mã số với độ rộng là 8, họ tên với độ rộng là 41, năm sinh với độ rộng là 6 và tiền lương với độ rộng 8. Sau đó đọc và in nội dung file ra màn hình.

```
#include <fstream>
#include <iomanip>
using namespace std;
struct Nhanvien
{    char ms[7], ht[40];
     int ns;
     long tl;
} nv[80];
main()
{   int n,i;
    do
     {    cout<<"Nhap so luong nhan vien : ";
          cin>>n;
    }while (n <=0 || n > 80);
     //Nhap thong tin cho cac nhan vien
    for (i=0; i<n; i++)
    {
          cout<<"Nhap nhan vien thu "<<i+1<<" : "<<endl;
          cin.ignore(1);
          cout<<"Nhap ma so:"; cin.getline(nv[i].ms,7);
          cout<<"Nhap ho ten:"; cin.getline(nv[i].ht,40);
          cout<<"Nhap nam sinh:"; cin>>nv[i].ns;
          cout<<"Nhap tien luong:"; cin>>nv[i].tl;
    }
     //Nhap ten file can luu
     char fname[20];
     cout<<"Nhap ten file can ghi: ";
     cin.ignore(1);
```

```
    cin.getline(fname, 20);
    fstream f;
    f.open(fname, ios::out | ios::trunc);
   //Luu so luong va thong tin cac nhan vien vao file
   f<<n;
   for (i=0; i<n; i++)
    {
        f<<endl<<setw(8)<<nv[i].ms<<setw(41)<<nv[i].ht;
        f<<setw(6)<<nv[i].ns<<setw(8)<<nv[i].tl;
    }
    f.close();
    //Doc file va in thong tin len man hinh
    char tam[80];
    cout<<"Noi dung cua file "<<fname<<" la: "<<endl;
    f.open(fname, ios::in);
   f>>n;
   cout<<"So luong nhan vien la: "<<n<<endl;
   while(!f.eof())
   {
        f.getline(tam, 80);
        cout<<tam<<endl;
   }
    f.close();
}
```

7.4 TẬP TIN NHỊ PHÂN

7.4.1 Khái niệm

Thông tin lưu trong tập tin nhị phân được xem như dãy byte bình thường. Mã kết thúc tập tin được chọn là -1, được định nghĩa là EOF trong stdio.h. Các thao tác trên tập tin nhị phân thường là đọc và ghi từng byte, không quan tâm đến ý nghĩa của byte.

Một số thao tác nhập/xuất sẽ có hiệu quả khác nhau khi ta mở tập tin dưới các dạng khác nhau.

Ví dụ: Giả sử ch = 10, khi đó f << ch sẽ ghi 2 byte 10 và 13 lên tập tin văn bản f, trong khi đó lệnh này chỉ ghi 1 byte 10 lên tập tin nhị phân. Ngược lại, nếu f là tập tin văn bản thì f.get(ch) sẽ trả về chỉ 1 byte 10 khi đọc được 2 byte 10, 13 liên tiếp nhau.

Một tập tin luôn ngầm định dưới dạng văn bản, do vậy để chỉ định tập tin là nhị phân ta cần sử dụng cờ ios::binary.

7.4.2 Đọc và ghi ký tự

get(c): đọc ký tự từ tập tin

put(c): ghi ký tự ra tập tin

7.4.3 Đọc và ghi chuỗi ký tự

read(char *buf, int n): đọc n ký tự từ buf vào dòng nhập

write(char *buf, int n): ghi n ký tự trong buf ra dòng xuất

gcount(): cho biết số ký tự mà read đọc được

7.4.4 Các ví dụ minh họa

7.4.4.1 Ví dụ 1

Viết chương trình cho phép người dùng nhập vào tên file. Nếu file chưa tồn tại thì báo lỗi và kết thúc, ngược lại thì in ra độ dài của file này.

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
main()
{    char fname[20], ch;
     long dodai = 0;
     cout << "Nhap ten file can mo: ";
     cin.getline(fname, 20);
     ifstream f(fname, ios::binary);
     if (!f.good())
    {
          cout << "File " << fname << " chua ton tai!";
          getch();
          exit(1);
     }
     while (!f.eof())
    {
        f.get(ch);
```

```
        dodai++;
    }
    f.close();
    cout << "Do dai cua file " << fname << " la: ";
    cout << dodai << endl;
}
```

✧ Tuy nhiên chương trình trên có thể được viết đơn giản hơn bằng cách sử dụng hàm seekg() và tellg() như sau:

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
main()
{   char fname[20], ch;
    long dodai = 0;
    cout<<"Nhap ten file can mo: ";
    cin.getline(fname, 20);
    ifstream f(fname, ios::binary);
    if (!f.good())
    {
        cout << "File "<<fname<<" chua ton tai";
        getch();
        exit(1);
    }
    f.seekg(0, ios::end);
    cout << "Do dai cua file "<<fname<< " la: ";
    cout << (int)f.tellg()+1 << endl;
    f.close();
}
```

7.4.4.2 Ví dụ 2

Viết chương trình mô phỏng việc thực hiện sao chép file. Yêu cầu cần phải sao chép và ghi từng byte một.

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
main()
{    char fnguon[20], fdich[20], ch;
     cout<<"Nhap ten file nguon: ";
     cin.getline(fnguon, 20);
     ifstream fi(fnguon, ios::in | ios::binary);
     if (!fi.good())
    {
          cout << "File "<<fnguon<<" chua ton tai";
          getch(); exit(1);
     }
     cout<<"Nhap ten file dich: ";
     cin.getline(fdich, 20);
     ifstream ft(fdich,ios::in | ios::binary);
     if (ft.good())
     {
          cout << "File "<<fdich<<" da ton tai";
          getch(); exit(1);
     }
     ft.close();
     ofstream fo(fdich,ios::out | ios::binary);
     while (!fi.eof())
     {
          fi.get(ch);
          fo.put(ch);
     }
     fi.close(); fo.close();
}
```

✧ Tuy nhiên chương trình trên có thể được viết lại bằng cách sử dụng hàm read(), write() và gcount() như sau:

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
main()
{    char fnguon[20], fdich[20], ch;
     cout<<"Nhap ten file nguon: ";
     cin.getline(fnguon, 20);
     ifstream fi(fnguon, ios::in | ios::binary);
     if (!fi.good())
    {
          cout << "File "<<fnguon<<" chua ton tai";
          getch(); exit(1);
     }
     cout<<"Nhap ten file dich: ";
     cin.getline(fdich, 20);
     ifstream ft(fdich,ios::in | ios::binary);
     if (ft.good())
     {
          cout << "File "<<fdich<<" da ton tai";
          getch(); exit(1);
     }
     ft.close();
     ofstream fo(fdich,ios::out | ios::binary);
     char buf[2000]; int n = 2000;
     while(n)
    {
          fi.read(buf, 2000);
          n = fi.gcount();
          fo.write(buf, n);
     }
     fi.close(); fo.close();
}
```

BÀI TẬP CHƯƠNG 7

-----ଓ✧ଌ-----

1. Viết chương trình in nội dung của một file văn bản và cho biết file đó có bao nhiêu dòng.

2. Viết chương trình đọc in từng kí tự của file văn bản ra màn hình, mỗi màn hình in 20 dòng.

3. Viết chương trình tìm từ dài nhất trong một file văn bản.

4. Viết chương trình ghép một file văn bản thứ hai vào file văn bản thứ nhất, trong đó tất cả chữ cái của file văn bản thứ nhất phải đổi thành chữ in hoa.

5. Viết chương trình in nội dung file ra màn hình và cho biết tổng số chữ cái, tổng số chữ số đã xuất hiện trong file.

6. Viết chương trình nhập 10 số thực từ bàn phím vào file INPUT.DAT. Sau đó đọc các số thực từ file trên và in tổng bình phương của chúng ra màn hình.

7. Viết chương trình nhập 10 số nguyên từ bàn phím vào file văn bản tên INPUT.DAT. Sau đó đọc các số nguyên từ file trên và ghi những số chẵn vào file EVEN.DAT còn các số lẻ vào file ODD.DAT.

8. Thông tin về một nhân viên trong cơ quan bao gồm: họ tên, chức vụ, số điện thoại, địa chỉ nhà riêng. Viết chương trình nhập từ bàn phím thông tin của N nhân viên và ghi vào file INPUT.DAT. Sau đó tìm và in ra thông tin nhân viên theo số điện thoại được nhập từ bàn phím.

TÀI LIỆU THAM KHẢO

-----ଓ✧ଌ-----

[1] Phạm Văn Ất, Nguyễn Hiếu Cường, Đỗ Văn Tuấn, Lê Trường Thông, ***Giáo trình Kỹ thuật lập trình C***, NXB Hồng Đức, 2009.

[2] Trần Đình Quế, Nguyễn Mạnh Hùng, ***Ngôn ngữ lập trình C++***, Học viện công nghệ bưu chính viễn thông, 2006.

[3] James P. Cohoon and Jack W.Davidson, ***C++ Program Design – An Introduction to Programming and Object-Oriented Design***, 2nd edition, WCB McGraw-Hill, 1999.

[4] Robert Sedgewick, *Cẩm nang Thuật Toán Vol.1,* Nhà xuất bản Khoa học Kỹ thuật.

[5] Robert Sedgewick, *Cẩm nang Thuật Toán Vol.2,* Nhà xuất bản Khoa học Kỹ thuật.

CÁC PHỤ LỤC

-----ଓ✧ଌ-----

PHỤ LỤC 1: Bảng mã ASCII với 128 ký tự đầu tiên

Hex	0	1	2	3	4	5	6	7
0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	STX 2	DC2 18	“ 34	2 50	B 66	R 82	b 98	r 114
3	♥ 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	♦ 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	♣ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
6	♠ 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
7	BEL 7	ETB 23	‘ 39	7 55	G 71	W 87	g 103	w 119
8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	EM 25	) 41	9 57	I 73	Y 89	I 105	y 121
A	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
E	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	DEL 127

PHỤ LỤC 2: Bảng mã ASCII với ký tự số 128 - số 255

Hex	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	░ 176	└ 192	╙ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	▒ 177	┴ 193	╤ 209	ß 225	± 241
2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
4	ä 132	ö 148	đ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
5	à 133	ò 149	Đ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
6	å 134	û 150	ª 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
7	ç 135	ù 151	º 167	╖ 183	╟ 199	╫ 215	τ 231	≈ 247
8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
9	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	Θ 233	∙ 249
A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Ω 234	· 250
B	ï 139	¢ 155	½ 171	╗ 187	╦ 203	█ 219	δ 235	√ 251
C	î 140	£ 156	¼ 172	╝ 188	╠ 204	■ 220	∞ 236	ⁿ 252
D	ì 141	¥ 157	¡ 173	╜ 189	═ 205	▌ 221	φ 237	² 253
E	Ä 142	₧ 158	« 174	╛ 190	╬ 206	▐ 222	ε 238	■ 254
F	Å 143	ƒ 159	» 175	┐ 191	╧ 207	▀ 223	∩ 239	 255

PHỤ LỤC 3: Các nguyên tắc chung khi sửa lỗi

Khi biên dịch chương trình thông thường ta nhận được 2 thông báo là error (lỗi) và warning (cảnh báo). Nguyên tắc chung là khi gặp lỗi thì người lập trình bắt buộc phải sửa lại cho đúng, còn khi gặp cảnh báo thì có thể bỏ qua. Tuy nhiên một số cảnh báo nếu không được sửa sẽ làm cho chương trình chạy không đúng.

Khi gặp thông báo lỗi thì ta nên tiến hành theo các bước sau để sửa lỗi:

1. Nhắp đúp chuột vào thông báo lỗi để nhảy đến vị trí có lỗi trong chương trình.
2. Đọc dòng chứa con trỏ để sửa lỗi.
3. Nếu ngay dòng chứa con trỏ không phát hiện lỗi thì đọc dòng ngay trên hoặc ngay dưới để tìm lỗi và sửa.
4. Nếu vẫn không phát hiện lỗi thì phải dò lỗi từ đầu chương trình đến dòng chứa con trỏ (vì có thể là do lỗi xuất hiện ở các dòng phía trên nữa).

✧ ***Lưu ý:*** Các lỗi mà ta có thể phát hiện và sửa theo các bước trên được gọi là lỗi cú pháp (syntax error). Tuy nhiên trong lập trình còn một loại lỗi mà trình biên dịch không phát hiện được đó là lỗi ngữ nghĩa hay lỗi giải thuật (chương trình vẫn thực thi nhưng kết quả sai). Để xử lý lỗi này thì người lập trình phải xem xét lại toàn bộ giải thuật để tìm ra nơi phát sinh lỗi (làm cho chương trình trả về kết quả sai) và sửa lại cho đúng.

Một số từ tiếng Anh thường xuất hiện **trong báo lỗi**:

declaration: khai báo

undeclared: không/chưa khai báo

unable: không thể

undefined: không xác định/định nghĩa

incorrectly: không chính xác

duplicate: trùng

object: đối tượng

statement: câu lệnh

parameter/argument: tham số, đối số

terminated: chấm dứt

syntax: cú pháp

constant: hằng số

expression: biểu thức

expected: mong đợi, chờ đợi

unexpected: không mong đợi

missing: thiếu

convert: chuyển đổi

identifier: tên, định danh

division by zero: chia cho không

illegal: không hợp lệ

incorrect: không đúng

too many/too much: quá nhiều

too large: quá lớn

mismatch/error: lỗi

call: lời gọi, lệnh gọi

ambiguity/ambiguous: không phân biệt

MỤC LỤC

-----ଓ✧ଓ-----

Lời nói đầu

- "C++ được đánh giá là ngôn ngữ mạnh vì tính mềm dẻo, gần gũi với ngôn ngữ máy. Ngoài ra, với khả năng lập trình theo mẫu (template), C++ đã khiến ngôn ngữ lập trình trở thành khái quát, không cụ thể và chi tiết như nhiều ngôn ngữ khác. Sức mạnh của C++ đến từ STL, viết tắt của Standard Template Library - một thư viện template cho C++ với những cấu trúc dữ liệu cũng như giải thuật được xây dựng tổng quát mà vẫn tận dụng được hiệu năng và tốc độ của C. Với khái niệm template, những người lập trình đã đề ra khái niệm lập trình khái lược (generic programming), C++ được cung cấp kèm với bộ thư viện chuẩn STL.

  Bộ thư viện này thực hiện toàn bộ các công việc vào ra dữ liệu (iostream), quản lý mảng (vector), thực hiện hầu hết các tính năng của các cấu trúc dữ liệu cơ bản (stack, queue, map, set...). Ngoài ra, STL còn bao gồm các thuật toán cơ bản: tìm min, max, tính tổng, sắp xếp (với nhiều thuật toán khác nhau), thay thế các phần tử, tìm kiếm (tìm kiếm thường và tìm kiếm nhị phân), trộn. Toàn bộ các tính năng nêu trên đều được cung cấp dưới dạng template nên việc lập trình luôn thể hiện tính khái quát hóa cao. Nhờ vậy, STL làm cho ngôn ngữ C++ trở nên trong sáng hơn nhiều."

  Trích **"Tổng quan về thư viện chuẩn STL"**
- STL khá là rộng nên tài liệu này mình chỉ viết để định hướng về cách sử dụng STL cơ bản để các bạn ứng dụng trong việc giải các bài toán tin học đòi hỏi đến cấu trúc dữ liệu và giải thuật.
- Mình chủ yếu sử dụng các ví dụ, cũng như nguồn tài liệu từ trang web www.cplusplus.com , các bạn có thể tham khảo chi tiết ở đó nữa.
- Để có thể hiểu được những gì mình trình bày trong này, các bạn cần có những kiến thức về các cấu trúc dữ liệu, cũng như một số thuật toán như sắp xếp, tìm kiếm...
- Việc sử dụng thành thạo STL sẽ là rất quan trọng nếu các bạn có ý định tham gia các kì thi như Olympic Tin Học, hay ACM. "STL sẽ nối dài khả năng lập trình của các bạn" (trích lời thầy Lê Minh Hoàng).
- Mọi ý kiến đóng góp xin gửi về địa chỉ: manhdjeu@gmail.com

Mục lục

- Thư viện mẫu chuẩn STL trong C++ chia làm 4 thành phần là:
    - Containers Library : chứa các cấu trúc dữ liệu mẫu (template)
        - Sequence containers
            - Vector
            - Deque
            - List
        - Containers adpators
            - Stack
            - Queue
            - Priority_queue
        - Associative containers
            - Set
            - Multiset
            - Map
            - Multimap
            - Bitset
    - Algorithms Library: một số thuật toán để thao tác trên dữ liệu
    - Iterator Library: giống như con trỏ, dùng để truy cập đến các phần tử dữ liệu của container.
    - Numeric library:

- Để sử dụng STL, bạn cần khai báo từ khóa "using namespace std;" sau các khai báo thư viện (các "#include", hay "#define",...)
- Ví dụ:

```
#include <iostream>
#include <stack>  //khai báo sử dụng container stack
#define n 100
using namespace std;  //khai báo sử dụng STL
main() {
      ....
}
```

- Việc sử dụng các hàm trong STL tương tự như việc sử dụng các hàm như trong class. Các bạn đọc qua một vài ví dụ là có thể thấy được quy luật.

I. ITERATOR (BIẾN LẶP):

- Trong C++, một biến lặp là một đối tượng bất kì, trỏ tới một số phần tử trong 1 phạm vi của các phần tử (như mảng hoặc container), có khả năng để lặp các phần tử trong phạm vi bằng cách sử dụng một tập các toán tử (operators) (như so sánh, tăng (++), ...)
- Dạng rõ ràng nhất của iterator là một con trỏ: Một con trỏ có thể trỏ tới các phần tử trong mảng, và có thể lặp thông qua sử dụng toán tử tăng (++). Tuy nhiên, cũng có

các dạng khác của iterator. Ví dụ: mỗi loại container (chẳng hạn như vector) có một loại iterator được thiết kế để lặp các phần tử của nó một cách hiệu quả.

- Iterator có các toán tử như:
    - So sánh: "==" , "!=" giữa 2 iterator.
    - Gán: "=" giữa 2 iterator.
    - Cộng trừ: "+","-" với hằng số và "++","—".
    - Lấy giá trị: "*".

II. CONTAINERS (THƯ VIỆN LƯU TRỮ)

- Một container là một đối tượng cụ thể lưu trữ một tập các đối tượng khác (các phần tử của nó). Nó được thực hiện như các lớp mẫu (class templates).
- Container quản lý không gian lưu trữ cho các phần tử của nó và cung cấp các hàm thành viên (member function) để truy cập tới chúng, hoặc trực tiếp hoặc thông qua các biến lặp (iterator – giống như con trỏ).
- Container xây dựng các cấu trúc thường sử dụng trong lập trình như: mảng động - dynamic arrays (vector), hàng đợi – queues (queue), hàng đợi ưu tiên – heaps (priority queue), danh sách kiên kết – linked list (list), cây – trees (set), mảng ánh xạ - associative arrays (map),...
- Nhiều container chứa một số hàm thành viên giống nhau. Quyết định sử dụng loại container nào cho nhu cầu cụ thể nói chung không chỉ phụ thuộc vào các hàm được cung cấp mà còn phải dựa vào hiệu quả của các hàm thành viên của nó (độ phức tạp (từ giờ mình sẽ viết tắt là ĐPT) của các hàm). Điều này đặc biệt đúng với container dãy (sequence containers), mà trong đó có sự khác nhau về độ phức tạp đối với các thao tác chèn/xóa phần tử hay truy cập vào phần tử.

1. Iterator:

Tất cả các container ở 2 loại: Sequence container và Associative container đều hỗ trợ các iterator như sau (ví dụ với vector, những loại khác có chức năng cũng vậy).

```
/*khai báo iterator "it"*/
vector <int> :: iterator it;
/* trỏ đến vị trí phần tử đầu tiên của vector */
it=vector.begin();
/*trỏ đến vị trí kết thúc (không phải phần tử cuối cùng nhé) của vector) */
it=vector.end();
/* khai báo iterator ngược "rit" */
vector <int> :: reverse_iterator rit; rit = vector.rbegin();
/* trỏ đến vị trí kết thúc của vector theo chiều ngược (không phải phần tử đầu tiên nhé*/
rit = vector.rend();
```

Tất cả các hàm iterator này đều có độ phức tạp O(1).

2. Vector (Mảng động):

Khai báo vector:

```
#include <vector>
...
/* Vector 1 chiều */

/* tạo vector rỗng kiểu dữ liệu int */
vector <int> first;

//tạo vector với 4 phần tử là 100
vector <int> second (4,100);

// lấy từ đầu đến cuối vector second
vector <int> third (second.begin(),second.end())

//copy từ vector third
vector <int> four (third)

/*Vector 2 chiều*/

/* Tạo vector 2 chiều rỗng */
vector < vector <int> > v;

/* khai báo vector 5×10 */
vector < vector <int> > v (5, 10) ;

/* khai báo 5 vector 1 chiều rỗng  */
vector < vector <int> > v (5) ;

//khai báo vector 5*10 với các phần tử khởi tạo giá trị là 1
vector < vector <int> > v (5, vector <int> (10,1) ) ;
```

Các bạn chú ý 2 dấu "ngoặc" không được viết liền nhau.
Ví dụ như sau là **sai**:

```
/*Khai báo vector 2 chiều SAI*/
vector <vector <int>> v;
```

Các hàm thành viên:

Capacity:

- size : trả về số lượng phần tử của vector. ĐPT O(1).
- empty : trả về true(1) nếu vector rỗng, ngược lại là false(0). ĐPT O(1).

Truy cập tới phần tử:

- operator [] : trả về giá trị phần tử thứ []. ĐPT O(1).
- at : tương tự như trên. ĐPT O(1).
- front: trả về giá trị phần tử đầu tiên. ĐPT O(1).
- back: trả về giá trị phần tử cuối cùng. ĐPT O(1).

Chỉnh sửa:

- push_back : thêm vào ở cuối vector. ĐPT O(1).
- pop_back : loại bỏ phần tử ở cuối vector. ĐPT O(1).
- insert (iterator,x): chèn "x" vào trước vị trí "iterator" (x có thể là phần tử hay iterator của 1 đoạn phần tử…). ĐPT O(n).
- erase : xóa phần tử ở vị trí iterator. ĐPT O(n).
- swap : đổi 2 vector cho nhau (ví dụ: first.swap(second);). ĐPT O(1).
- clear: xóa vector. ĐPT O(n).

Nhận xét:

- Sử dụng vector sẽ tốt khi:
  - Truy cập đến phần tử riêng lẻ thông qua vị trí của nó O(1)
  - Chèn hay xóa ở vị trí cuối cùng O(1).
- Vector làm việc giống như một "mảng động".

Ví dụ 1: Ví dụ này chủ yếu để làm quen sử dụng các hàm chứ không có đề bài cụ thể.

```
#include <iostream>
#include <vector>
using namespace std;
vector <int> v; //Khai báo vector
vector <int>::iterator it;  //Khai báo iterator
vector <int>::reverse_iterator rit; //Khai báo iterator ngược
int i;
main() {
	for (i=1;i<=5;i++) v.push_back(i); // v={1,2,3,4,5}
	cout << v.front() << endl;         // In ra 1
	cout << v.back() << endl;          // In ra 5

	cout << v.size() << endl;          // In ra 5

	v.push_back(9);                    // v={1,2,3,4,5,9}
	cout << v.size() << endl;          // In ra 6

	v.clear();                         // v={}
	cout << v.empty() << endl;         // In ra 1 (vector rỗng)

	for (i=1;i<=5;i++) v.push_back(i); // v={1,2,3,4,5}
	v.pop_back();                      // v={1,2,3,4}
	cout << v.size() << endl;          // In ra 4

	v.erase(v.begin()+1);              // Xóa ptử thứ 1 v={1,3,4}
	v.erase(v.begin(),v.begin()+2);    // v={4}
	v.insert(v.begin(),100);           // v={100,4}
	v.insert(v.end(),5);               // v={100,4,5}

	/*Duyệt theo chỉ số phần tử*/
	for (i=0;i<v.size();i++) cout << v[i] << " "; // 100 4 5
	cout << endl;



   /*Chú ý: Không nên viết
```

```
        for (i=0;i<=v.size()-1;i++) ...
        Vì nếu vector v rỗng thì sẽ dẫn đến sai khi duyệt !!!
        */

        /*Duyệt theo iterator*/
        for (it=v.begin();it!=v.end();it++)
            cout << *it << " " ;
        //In ra giá trị ma iterator đang trỏ tới "100 4 5"
        cout << endl;

        /*Duyệt iterator ngược*/
        for (rit=v.rbegin();rit!=v.rend();rit++)
            cout << *rit << " "; // 5 4 100
        cout << endl;

        system("pause");
}
```

Ví dụ 2: Cho đồ thị vô hướng G có n đỉnh (các đỉnh đánh số từ 1 đến n) và m cạnh và không có khuyên (đường đi từ 1 đỉnh tới chính đỉnh đó).
Cài đặt đồ thị bằng danh sách kề và in ra các cạnh kề đối với mỗi cạnh của đồ thị.
Dữ liệu vào:

- Dòng đầu chứa n và m cách nhau bởi dấu cách
- M dòng sau, mỗi dòng chứa u và v cho biết có đường đi từ u tới v. Không có cặp đỉnh u,v nào chỉ cùng 1 đường đi.

Dữ liệu ra:

- M dòng: Dòng thứ i chứa các đỉnh kề cạnh i theo thứ tự tăng dần và cách nhau bởi dấu cách.

Giới hạn: 1 <= n,m <= 10000
Ví dụ:

INPUT	OUTPUT
6 7	2 3 5 6
1 2	1 3 6
1 3	1 2 5
1 5	
2 3	1 3
2 6	1 2
3 5	
6 1	

Chương trình mẫu:

```
#include <iostream>
#include <vector>
using namespace std;
vector < vector <int> > a (10001);
//Khai báo vector 2 chiều với 10001 vector 1 chiều rỗng
int m,n,i,j,u,v;
main() {
        /*Input data*/
```

```
        cin >> n >> m;
        for (i=1;i<=m;i++) {
            cin >> u >> v;
            a[u].push_back(v);
            a[v].push_back(u);
        }
        /*Sort cạnh kề*/
        for (i=1;i<=m;i++)
            sort(a[i].begin(),a[i].end());
        /*Print Result*/
        for (i=1;i<=m;i++) {
            for (j=0;j<a[i].size();j++) cout << a[i][j] << " ";
            cout << endl;
        }
        system("pause");
}
```

3. Deque (Hàng đợi hai đầu):

- Deque (thường được phát âm giống như "deck") là từ viết tắt của double-ended queue (hàng đợi hai đầu).
- Deque có các ưu điểm như:
    - Các phần tử có thể truy cập thông cua chỉ số vị trí của nó. O(1)
    - Chèn hoặc xóa phần tử ở cuối hoặc đầu của dãy. O(1)

Khai báo: `#include <deque>`

Capacity:

- size : trả về số lượng phần tử của deque. ĐPT O(1).
- empty : trả về true(1) nếu deque rỗng, ngược lại là false(0). ĐPT O(1).

Truy cập phần tử:

- operator [] : trả về giá trị phần tử thứ []. ĐPT O(1).
- at : tương tự như trên. ĐPT O(1).
- front: trả về giá trị phần tử đầu tiên. ĐPT O(1).
- back: trả về giá trị phần tử cuối cùng. ĐPT O(1).

Chỉnh sửa:

- push_back : thêm phần tử vào ở cuối deque. ĐPT O(1).
- push_front : thêm phần tử vào đầu deque. ĐPT O(1).
- pop_back : loại bỏ phần tử ở cuối deque. ĐPT O(1).
- pop_front : loại bỏ phần tử ở đầu deque. ĐPT O(1).
- insert (iterator,x): chèn "x" vào trước vị trí "iterator" (x có thể là phần tử hay iterator của 1 đoạn phần tử…). ĐPT O(n).
- erase : xóa phần tử ở vị trí iterator. ĐPT O(n).
- swap : đổi 2 deque cho nhau (ví dụ: first.swap(second);). ĐPT O(n).

clear: xóa vector. ĐPT O(1).

4. List (Danh sách liên kết):

- List được thực hiện như danh sách nối kép (doubly-linked list). Mỗi phần tử trong danh sách nối kép có liên kết đến một phần tử trước đó và một phần tử sau nó.

- Do đó, list có các ưu điểm như sau:
    - Chèn và loại bỏ phần tử ở bất cứ vị trí nào trong container. O(1).
- Điểm yếu của list là khả năng truy cập tới phần tử thông qua vị trí. O(n).
- Khai báo: #include <list>

Các hàm thành viên:

Capacity:
- size : trả về số lượng phần tử của list. ĐPT O(1).
- empty : trả về true(1) nếu list rỗng, ngược lại là false(0). ĐPT O(1).

Truy cập phần tử:
- front: trả về giá trị phần tử đầu tiên. ĐPT O(1).
- back: trả về giá trị phần tử cuối cùng. ĐPT O(1).

Chỉnh sửa:
- push_back : thêm phần tử vào ở cuối list. ĐPT O(1).
- push_front : thêm phần tử vào đầu list. ĐPT O(1).
- pop_back : loại bỏ phần tử ở cuối list. ĐPT O(1).
- pop_front : loại bỏ phần tử ở đầu list. ĐPT O(1).
- insert (iterator,x): chèn "x" vào trước vị trí "iterator" (x có thể là phần tử hay iterator của 1 đoạn phần tử…). ĐPT là số phần tử thêm vào.
- erase : xóa phần tử ở vị trí iterator. ĐPT là số phần tử bị xóa đi.
- swap : đổi 2 list cho nhau (ví dụ: first.swap(second);). ĐPT O(1).
- clear: xóa list. ĐPT O(n).

Operations:
- splice : di chuyển phần tử từ list này sang list khác. ĐPT O(n).
- remove (const) : loại bỏ tất cả phần tử trong list bằng const. ĐPT O(n).
- remove_if (function) : loại bỏ tất các phần tử trong list nếu hàm function return true . ĐPT O(n).
- unique : loại bỏ các phần tử bị trùng lặp hoặc thỏa mãn hàm nào đó. ĐPT O(n). Lưu ý: Các phần tử trong list phải được sắp xếp.
- sort : sắp xếp các phần tử của list. O(NlogN)
- reverse : đảo ngược lại các phần tử của list. O(n).

5. Stack (Ngăn xếp):

- Stack là một loại container adaptor, được thiết kế để hoạt động theo kiểu LIFO (Last - in first - out) (vào sau ra trước), tức là một kiểu danh sách mà việc bổ sung và loại bỏ một phần tử được thực hiển ở cuối danh sách. Vị trí cuối cùng của stack gọi là đỉnh (top) của ngăn xếp.

Khai báo: `#include <stack>`

Các hàm thành viên:
- size : trả về kích thước hiện tại của stack. ĐPT O(1).
- empty : true stack nếu rỗng, và ngược lại. ĐPT O(1).
- push : đẩy phần tử vào stack. ĐPT O(1).
- pop : loại bỏ phần tử ở đỉnh của stack. ĐPT O(1).
- top : truy cập tới phần tử ở đỉnh stack. ĐPT O(1).

Chương trình demo:

```
#include <iostream>
#include <stack>
using namespace std;
stack <int> s;
int i;
main() {
	for (i=1;i<=5;i++) s.push(i); // s={1,2,3,4,5}
	s.push(100);                  // s={1,2,3,4,5,100}
	cout << s.top() << endl;      // In ra 100
	s.pop();                      // s={1,2,3,4,5}
	cout << s.empty() << endl;    // In ra 0
	cout << s.size() << endl;     // In ra 5
	system("pause");
}
```

6. Queue (Hàng đợi):

- Queue là một loại container adaptor, được thiết kế để hoạt động theo kiểu FIFO (First - in first - out) (vào trước ra trước), tức là một kiểu danh sách mà việc bổ sung được thực hiển ở cuối danh sách và loại bỏ ở đầu danh sách.
- Trong queue, có hai vị trí quan trọng là vị trí đầu danh sách (front), nơi phần tử được lấy ra, và vị trí cuối danh sách (back), nơi phần tử cuối cùng được thêm vào.

Khai báo: `#include <queue>`

Các hàm thành viên:

- size : trả về kích thước hiện tại của queue. ĐPT O(1).
- empty : true nếu queue rỗng, và ngược lại. ĐPT O(1).
- push : đẩy vào cuối queue. ĐPT O(1).
- pop: loại bỏ phần tử (ở đầu). ĐPT O(1).
- front : trả về phần tử ở đầu. ĐPT O(1).
- back: trả về phần tử ở cuối. ĐPT O(1).

Chương trình demo:

```
#include <iostream>
#include <queue>
using namespace std;
queue <int> q;
int i;
main() {
	for (i=1;i<=5;i++) q.push(i); // q={1,2,3,4,5}
	q.push(100);                  // q={1,2,3,4,5,100}
	cout << q.front() << endl;    // In ra 1
	q.pop();                      // q={2,3,4,5,100}
	cout << q.back() << endl;     // In ra 100
	cout << q.empty() << endl;    // In ra 0
	cout << q.size() << endl;     // In ra 5
	system("pause");
}
```

7. Priority Queue (Hàng đợi ưu tiên):

- Priority queue là một loại container adaptor, được thiết kế đặc biệt để phần tử ở đầu luôn luôn lớn nhất (theo một quy ước về độ ưu tiên nào đó) so với các phần tử khác.
- Nó giống như một heap, mà ở đây là heap max, tức là phần tử có độ ưu tiên lớn nhất có thể được lấy ra và các phần tử khác được chèn vào bất kì.
- Phép toán so sánh mặc định khi sử dụng priority queue là phép toán less (Xem thêm ở thư viện **functional**)
- Để sử dụng priority queue một cách hiệu quả, các bạn nên học cách viết hàm so sánh để sử dụng cho linh hoạt cho từng bài toán.
- Khai báo: `#include <queue>`

```
/*Dạng 1 (sử dụng phép toán mặc định là less)*/
priority_queue <int> pq;

/* Dạng 2 (sử dụng phép toán khác) */
priority_queue <int,vector<int>,greater<int> > q; //phép toán greater
```

Phép toán khác cũng có thể do người dùng tự định nghĩa. Ví dụ:
Cách khai báo ở dạng 1 tương đương với:

```
/* Dạng sử dụng class so sánh tự định nghĩa */
struct cmp{
	bool operator() (int a,int b) {return a<b;}
};


main() {
	…
	priority_queue <int,vector<int>,cmp > q;
}
```

Các hàm thành viên:

- size : trả về kích thước hiện tại của priority queue. ĐPT O(1)
- empty : true nếu priority queue rỗng, và ngược lại. ĐPT O(1).
- push : đẩy vào priority queue. ĐPT O(logN).
- pop: loại bỏ phần tử ở đỉnh priority queue. ĐPT O(logN).
- top : trả về phần tử ở đỉnh priority queue. ĐPT O(1).

Chương trình Demo 1:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

main() {
	priority_queue <int> p;  // p={}
	p.push(1);               // p={1}
	p.push(5);               // p={1,5}
	cout << p.top() << endl; // In ra 5
	p.pop();                 // p={1}
	cout << p.top() << endl; // In ra 1
```

```
        p.push(9);              // p={1,9}
        cout << p.top() << endl; // In ra 9
        system("pause");
}
```

Chương trình Demo 2:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

main() {
        priority_queue < int , vector <int> , greater <int> > p;  // p={}
        p.push(1);              // p={1}
        p.push(5);              // p={1,5}
        cout << p.top() << endl; // In ra 1
        p.pop();                // p={5}
        cout << p.top() << endl; // In ra 5
        p.push(9);              // p={5,9}
        cout << p.top() << endl; // In ra 5
        system("pause");
}
```

Chương trình Demo 3:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

struct cmp{
        bool operator() (int a,int b) {return a<b;}
};

main() {
        priority_queue < int , vector <int> , cmp > p;  // p={}
        p.push(1);              // p={1}
        p.push(5);              // p={1,5}
        cout << p.top() << endl; // In ra 1
        p.pop();                // p={5}
        cout << p.top() << endl; // In ra 5
        p.push(9);              // p={5,9}
        cout << p.top() << endl; // In ra 5
        system("pause");
}
```

8. Set (Tập hợp):

- Set là một loại associative containers để lưu trữ các phần tử không bị trùng lặp (unique elements), và các phần tử này chính là các khóa (keys).
- Khi duyệt set theo iterator từ begin đến end, các phần tử của set sẽ tăng dần theo phép toán so sánh.
- Mặc định của set là sử dụng phép toán less, bạn cũng có thể viết lại hàm so sánh theo ý mình.

- Set được thực hiện giống như cây tìm kiếm nhị phân (Binary search tree).

Khai báo:

```
#include <set>
set <int> s;
set <int, greater<int> > s;
```

Hoặc viết class so sánh theo ý mình:

```
struct cmp{
	bool operator() (int a,int b) {return a<b;}
};
set <int,cmp > myset ;
```

Capacity:

- size : trả về kích thước hiện tại của set. ĐPT O(1)
- empty : true nếu set rỗng, và ngược lại. ĐPT O(1).

Modifiers:

- insert : Chèn phần tử vào set. ĐPT O(logN).
- erase : có 2 kiểu xóa: xóa theo iterator, hoặc là xóa theo khóa. ĐPT O(logN).
- clear : xóa tất cả set. ĐPT O(n).
- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy itarator trỏ về "end" của set. ĐPT O(logN).
- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của set. ĐPT O(logN).
- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của set.. ĐPT O(logN).
- count : trả về số lần xuất hiện của khóa trong container. Nhưng trong set, các phần tử chỉ xuất hiện một lần, nên hàm này có ý nghĩa là sẽ return 1 nếu khóa có trong container, và 0 nếu không có. ĐPT O(logN).

Chương trình Demo 1:

```
#include <iostream>
#include <set>

using namespace std;

main() {
	set <int> s;
	set <int> ::iterator it;
	s.insert(9);                // s={9}
	s.insert(5);                // s={5,9}
	cout << *s.begin() << endl; //In ra 5
	s.insert(1);                // s={1,5,9}
	cout << *s.begin() << endl; // In ra 1

	it=s.find(5);
	if (it==s.end()) cout << "Khong co trong container" << endl;
	else  cout << "Co trong container" << endl;

	s.erase(it);                // s={1,9}
	s.erase(1);                 // s={9}
```

```
        s.insert(3);                // s={3,9}
        s.insert(4);                // s={3,4,9}

        it=s.lower_bound(4);
        if (it==s.end()) cout << "Khong co phan tu nao trong set khong be hon 4" << endl;
        else cout << "Phan tu be nhat khong be hon 4 la " << *it << endl;  // In ra 4

        it=s.lower_bound(10);
        if (it==s.end()) cout << "Khong co phan tu nao trong set khong be hon 10" << endl;
        else cout << "Phan tu be nhat khong be hon 10 la " << *it << endl; // Khong co ptu nao

        it=s.upper_bound(4);
        if (it==s.end()) cout << "Khong co phan tu nao trong set lon hon 4" << endl;
        else cout << "Phan tu be nhat lon hon 4 la " << *it << endl;       // In ra 9

        /* Duyet set */

        for (it=s.begin();it!=s.end();it++) {
            cout << *it <<  " ";
        }
        // In ra 3 4 9

        cout << endl;
        system("pause");
}
```

Lưu ý: Nếu bạn muốn sử dụng hàm lower_bound hay upper_bound để tìm phần tử lớn nhất "bé hơn hoặc bằng" hoặc "bé hơn" bạn có thể thay đổi cách so sánh của set để tìm kiếm. Mời bạn xem chương trình sau để rõ hơn:

```
#include <iostream>
#include <set>
#include <vector>

using namespace std;

main() {
        set <int, greater <int> > s;
        set <int, greater <int> > :: iterator it; // Phép toán so sánh là greater

        s.insert(1);                    // s={1}
        s.insert(2);                    // s={2,1}
        s.insert(4);                    // s={4,2,1}
        s.insert(9);                    // s={9,4,2,1}

        /* Tim phần tử lớn nhất bé hơn hoặc bằng 5 */
        it=s.lower_bound(5);
        cout << *it << endl;            // In ra 4

        /* Tim phần tử lớn nhất bé hơn 4 */
        it=s.upper_bound(4);
        cout << *it << endl;            // In ra 2

        system("pause");
}
```

9. Mutiset (Tập hợp):

- Multiset giống như Set nhưng có thể chứa các khóa có giá trị giống nhau.

- Khai báo : giống như set.
- **Các hàm thành viên:**

Capacity:
- size : trả về kích thước hiện tại của multiset. ĐPT O(1)
- empty : true nếu multiset rỗng, và ngược lại. ĐPT O(1).

Chỉnh sửa:
- insert : Chèn phần tử vào set. ĐPT O(logN).
- erase :
  - xóa theo iterator ĐPT O(logN)
  - xóa theo khóa: xóa tất cả các phần tử bằng khóa trong multiset ĐPT: O(logN) + số phần tử bị xóa.
- clear : xóa tất cả set. ĐPT O(n).
- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:
- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy itarator trỏ về "end" của set. ĐPT O(logN). Dù trong multiset có nhiều phần tử bằng khóa thì nó cũng chỉ iterator đến một phần tử.
- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của set. ĐPT O(logN).
- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của set.. ĐPT O(logN).
- count : trả về số lần xuất hiện của khóa trong multiset. ĐPT O(logN) + số phần tử tìm được.

Chương trình Demo:

```
#include <iostream>
#include <set>
using namespace std;
main() {
	multiset <int> s;
	multiset <int> :: iterator it;
	int i;
	for (i=1;i<=5;i++) s.insert(i*10); // s={10,20,30,40,50}
	s.insert(30);                      // s={10,20,30,30,40,50}
	cout << s.count(30) << endl;       // In ra 2
	cout << s.count(20) << endl;       // In ra 1
	s.erase(30);                       // s={10,20,40,50}

	/* Duyet set */

	for (it=s.begin();it!=s.end();it++) {
	    cout << *it <<  " ";
	}
	// In ra 10 20 40 50
	cout << endl;
	system("pause");
}
```

10. Map (Ánh xạ):

- Map là một loại associative container. Mỗi phần tử của map là sự kết hợp của khóa (key value) và ánh xạ của nó (mapped value). Cũng giống như set, trong map không chứa các khóa mang giá trị giống nhau.
- Trong map, các khóa được sử dụng để xác định giá trị các phần tử. Kiểu của khóa và ánh xạ có thể khác nhau.
- Và cũng giống như set, các phần tử trong map được sắp xếp theo một trình tự nào đó theo cách so sánh.
- Map được cài đặt bằng red-black tree (cây đỏ đen) – một loại cây tìm kiếm nhị phân tự cân bằng. Mỗi phần tử của map lại được cài đặt theo kiểu pair (xem thêm ở thư viện **utility**).

Khai báo:

```
#include <map>
...
map <kiểu_dữ_liệu_1,kiểu_dữ_liệu_2>
// kiểu dữ liệu 1 là khóa, kiểu dữ liệu 2 là giá trị của khóa.
```

Sử dụng class so sánh:

Dạng 1:

```
struct cmp{
	bool operator() (char a,char b) {return a<b;}
};
.....
map <char,int,cmp> m;
```

- Truy cập đến giá trị của các phần tử trong map khi sử dụng iterator:

Ví dụ ta đang có một iterator là it khai báo cho map thì:

```
	(*it).first;    // Lấy giá trị của khóa, kiểu_dữ_liệu_1
	(*it).second;   // Lấy giá trị của giá trị của khóa, kiểu_dữ_liệu_2
	(*it)          // Lấy giá trị của phần tử mà iterator đang trỏ đến, kiểu pair

	it->first;      // giống như (*it).first
	it->second;     // giống như (*it).second
```

Capacity:

- size : trả về kích thước hiện tại của map. ĐPT O(1)
- empty : true nếu map rỗng, và ngược lại. ĐPT O(1).

Truy cập tới phần tử:

- operator [khóa]: Nếu khóa đã có trong map, thì hàm này sẽ trả về giá trị mà khóa ánh xạ đến. Ngược lại, nếu khóa chưa có trong map, thì khi gọi [] nó sẽ thêm vào map khóa đó. ĐPT O(logN)

Chỉnh sửa

- insert : Chèn phần tử vào map. Chú ý: phần tử chèn vào phải ở kiểu "pair". ĐPT O(logN).
- erase :
  - xóa theo iterator ĐPT O(logN)
  - xóa theo khóa: xóa khóa trong map. ĐPT: O(logN).
- clear : xóa tất cả set. ĐPT O(n).
- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy iterator trỏ về "end" của map. ĐPT O(logN).
- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).
- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).
- count : trả về số lần xuất hiện của khóa trong multiset. ĐPT O(logN)

Chương trình demo:

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;
main() {
	map <char,int> m;
	map <char,int> :: iterator it;

	m['a']=1;                              // m={{'a',1}}
	m.insert(make_pair('b',2));            // m={{'a',1};{'b',2}}
	m.insert(pair<char,int>('c',3) );      // m={{'a',1};{'b',2};{'c',3}}

	cout << m['b'] << endl;                // In ra 2
	m['b']++;                              // m={{'a',1};{'b',3};{'c',3}}

	it=m.find('c');                        // it point to key 'c'

	cout << it->first << endl;             // In ra 'c'
	cout << it->second << endl;            // In ra 3

	m['e']=100;                           //m={{'a',1};{'b',3};{'c',3};{'e',100}}

	it=m.lower_bound('d');                 // it point to 'e'
	cout << it->first << endl;             // In ra 'e'
	cout << it->second << endl;            // In ra 100

	system("pause");
}
```

11. Multi Map (Ánh xạ):

Giống như map nhưng có thể chứa các phần tử có khóa giống nhau, do đó nó khác map ở chỗ không có opearator[].

III. STL ALGORITHMS (THƯ VIỆN THUẬT TOÁN):

1. Giới thiệu tổng quan:

Thư viện algorithm nằm trong thư viện chuẩn (Standard Template Libarary) của C++, nó được thiết kế để thực hiện các thuật toán cơ bản, các thao tác thường sử dụng (như là tìm kiếm, sắp xếp, trộn, chèn, xóa …) trên một khoảng liên tiếp của dãy các phần tử có cùng kiểu dữ liệu.
Một khoảng là một dãy các đối tượng có thể truy cập thông qua các biến lặp (iterator) hoặc các con trỏ (pointer). Ví dụ như là một mảng hoặc một số STL container.
Các hàm STL Algorithm chia thành 7 loại cơ bản:

- Các thao tác không thay đổi đoạn phần tử (Non-modifying sequence operations)
- Các thao tác thay đổi đoạn phần tử (Modifying sequence operations)
- Sắp xếp (Sorting)
- Tím kiếm nhị phân (Binary Search)
- Trộn (Merge)
- Hàng đợi ưu tiên (Heap)
- Min/max

Để sử dụng thư viện algorithm bạn cần phải khai báo: #include <algorithm>

Sau đây sẽ là phần trình bày chi tiết của mỗi loại.

2. Các thao tác không thay đổi đoạn phần tử:

2.1. for_each:

```
template <class InputIterator, class Function>
   Function for_each (InputIterator first, InputIterator last, Function f);
```

Ý nghĩa:

- Thực hiện hàm *'f'* cho mỗi phần tử thuộc nửa đoạn *[first,last)*

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- f: hàm duy nhất nhận phần tử trong đoạn làm tham số. Nó có thể là con trỏ đến hàm hoặc đối tượng mà lớp overloads toán tử (). Xem ví dụ để hiểu rõ hơn.

Ví dụ:

```
// for_each example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

void myfunction (int i) {
  cout << " " << i;
}

struct myclass {
  void operator() (int i) {cout << " " << i;}
```

```
} myobject;

int main () {
  vector<int> myvector;
  myvector.push_back(10);
  myvector.push_back(20);
  myvector.push_back(30);

  cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myfunction);

  // or:
  cout << "\nmyvector contains:";
  for_each (myvector.begin(), myvector.end(), myobject);

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 10 20 30
myvector contains: 10 20 30
```

2.2. find:

Dạng:

```
template <class InputIterator, class T>
   InputIterator find ( InputIterator first, InputIterator last, const T& value );
```

Ý nghĩa:

- Tìm kiếm phần tử. Trả về con trỏ đến phần tử đầu tiên trong đoạn *[first,last)* mà so sánh bằng *'value'*, hoặc trả về *'last'* nếu không tìm thấy.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- value: phần tử tìm kiếm

Ví dụ:

```
// find example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int myints[] = { 10, 20, 30 ,40 };
  int * p;

  // pointer to array element:
  p = find(myints,myints+4,30);
  ++p;
  cout << "The element following 30 is " << *p << endl;

  vector<int> myvector (myints,myints+4);
  vector<int>::iterator it;

  // iterator to vector element:
```

```
  it = find (myvector.begin(), myvector.end(), 30);
  ++it;
  cout << "The element following 30 is " << *it << endl;

  return 0;
}
```

Output:

```
The element following 30 is 40
The element following 30 is 40
```

2.3. find_if:

Dạng:

```
template <class InputIterator, class Predicate>
   InputIterator find_if ( InputIterator first, InputIterator last, Predicate pred
);
```

Ý nghĩa:

- Tìm kiếm phần tử. Trả về con trỏ đến phần tử đầu tiên trong đoạn *[first,last)* mà khi áp dụng *'pred'* trả vê true, hoặc trả về *'last'* nếu không tìm thấy.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- pred: giống như *'f'* trong hàm 'for_each' ở trên, nhưng thay vì thực hiện một nhiệm vụ nào đó thì 'pred' phải trả về giá trị *'true'* hoặc *'false'* để so sánh.

Ví dụ:

```
// find_if example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool IsOdd (int i) {
  return ((i%2)==1);
}

int main () {
  vector<int> myvector;
  vector<int>::iterator it;

  myvector.push_back(10);
  myvector.push_back(25);
  myvector.push_back(40);
  myvector.push_back(55);

  it = find_if (myvector.begin(), myvector.end(), IsOdd);
  cout << "The first odd value is " << *it << endl;

  return 0;
}
```

Output:

```
The first odd value is 25
```

2.4. count:

Dạng:

```
template<class InputIterator, class Type>
   typename iterator_traits<InputIterator>::difference_type count(
      InputIterator _First,
      InputIterator _Last,
      const Type& _Val
   );
```

Ý nghĩa:

- Trả về số phần tử trong đoạn *[first,last)* mà bằng *'value'*.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- value: phần tử so sánh để đếm số lần xuất hiện

Ví dụ:

```
// count algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int mycount;

  // counting elements in array:
  int myints[] = {10,20,30,30,20,10,10,20};   // 8 elements
  mycount = (int) count (myints, myints+8, 10);
  cout << "10 appears " << mycount << " times.\n";

  // counting elements in container:
  vector<int> myvector (myints, myints+8);
  mycount = (int) count (myvector.begin(), myvector.end(), 20);
  cout << "20 appears " << mycount  << " times.\n";

  return 0;
}
```

Output:

```
10 appears 3 times.
   20 ppears 3 times.
```

2.5. count_if:

Dạng:

```
template<class InputIterator, class Predicate>
   typename iterator_traits<InputIterator>::difference_typecount_if(
      InputIterator _First,
      InputIterator _Last,
      Predicate _Pred
   );
```

Ý nghĩa:

- Trả về số phần tử trong đoạn *[first,last)* mà khi áp dụng *'pred'* trả về *'true'*.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).

- pred: giống như *'f'* trong hàm 'for_each' ở trên, nhưng thay vì thực hiện một nhiệm vụ nào đó thì 'pred' phải trả về giá trị *'true'* hoặc *'false'* để so sánh.

Ví dụ:

```
// count_if example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool IsOdd (int i) { return ((i%2)==1); }

int main () {
  int mycount;

  vector<int> myvector;
  for (int i=1; i<10; i++) myvector.push_back(i); // myvector: 1 2 3 4 5 6
7 8 9

  mycount = (int) count_if (myvector.begin(), myvector.end(), IsOdd);
  cout << "myvector contains " << mycount  << " odd values.\n";

  return 0;
}
```

Output:

```
myvector contains 5 odd values.
```

3. Các thao tác thay đổi đoạn phần tử:

3.1. Copy:

Dạng:

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy ( InputIterator first, InputIterator last, OutputIterator
result );
```

Ý nghĩa:

- Sao chép đoạn phần tử *[first,last)* đến đoạn phần tử mới bắt đầu từ *'result'*

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- result: trỏ đến phần tử đầu tiên của đoạn kết quả.

Ví dụ:

```
// copy algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int myints[]={10,20,30,40,50,60,70};
  vector<int> myvector;
  vector<int>::iterator it;

  myvector.resize(7);   // allocate space for 7 elements
```

```
  copy ( myints, myints+7, myvector.begin() );

  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 10 20 30 40 50 60 70
```

3.2. Swap:

Dạng:

```
template <class T> void swap ( T& a, T& b );
```

Ý nghĩa:

- Trao đổi giá trị của hai đối tượng. Gán giá trị của a cho b, và b cho a.

Tham số:

- a,b: 2 đối tượng có cùng kiểu.

Ví dụ:

```
// swap algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {

  int x=10, y=20;                          // x:10 y:20
  swap(x,y);                               // x:20 y:10
  return 0;
}
```

3.3. Replace:

Dạng:

```
template < class ForwardIterator, class T >
  void replace ( ForwardIterator first, ForwardIterator last,
                 const T& old_value, const T& new_value );
```

Ý nghĩa:

- Thay thế tất cả các phần tử trong đoạn *[first,last)* có giá trị *'old_value'* bằng giá trị mới *'new_value'*.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- old_value: giá trị cần thay thế.
- new_value: giá trị thay thế vào.

Ví dụ:

```
// replace algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int myints[] = { 10, 20, 30, 30, 20, 10, 10, 20 };
  vector<int> myvector (myints, myints+8);            // 10 20 30 30 20 10
10 20

  replace (myvector.begin(), myvector.end(), 20, 99); // 10 99 30 30 99 10
10 99

  cout << "myvector contains:";
  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 10 99 30 30 99 10 10 99
```

3.4. Remove:

Dạng:

```
template < class ForwardIterator, class T >
  ForwardIterator remove ( ForwardIterator first, ForwardIterator last,
                           const T& value );
```

Ý nghĩa:

- Xóa bỏ các phần tử trong đoạn *[first,last)* mà có giá trị là *'value'*.

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- value: giá trị cần thay thế.

Ví dụ:

```
// remove algorithm example
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};      // 10 20 30 30 20 10 10 20

  // bounds of range:
  int* pbegin = myints;                          // ^
  int* pend = myints+sizeof(myints)/sizeof(int); // ^
^

  pend = remove (pbegin, pend, 20);              // 10 30 30 10 10 ?  ?  ?
                                                 // ^              ^
  cout << "range contains:";
  for (int* p=pbegin; p!=pend; ++p)
    cout << " " << *p;
```

```
  cout << endl;

  return 0;
}
```

Output:

```
range contains: 10 30 30 10 10
```

3.5. Unique:

Dạng 1:

```
template <class ForwardIterator>
  ForwardIterator unique ( ForwardIterator first, ForwardIterator last );
```

Dạng 2:

```
template <class ForwardIterator, class BinaryPredicate>
  ForwardIterator unique ( ForwardIterator first, ForwardIterator last,
                           BinaryPredicate pred );
```

Ý nghĩa:

- Loại bỏ các phần tử liên tiếp có giá trị bằng nhau trong đoạn *[first,last)*.
- Sau khi 'unique', hàm trả về con trỏ đến vị trí kết thúc của đoạn mới (khác với phần tử cuối cùng), đoạn cũ vẫn giữ nguyên kích thước, một số phần tử sau vị trí kết thúc có giá trị không xác định.
- Việc so sánh giữa các yếu tố thực hiện bằng cách áp dụng toán tử so sánh *'=='* (dạng 1) hoặc tham số mẫu *'comp'* (dạng 2).

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- comp: hàm so sánh hai đối tượng (có cùng kiểu dữ liệu với đoạn phần tử), trả về *'true'* nếu 2 tham số bằng nhau. ('bằng nhau' ở đây do người sử dụng tự định nghĩa), và false nếu ngược lại.

Ví dụ:

```
// unique algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool myfunction (int i, int j) {
  return (i==j);
}

int main () {
  int myints[] = {10,20,20,20,30,30,20,20,10};    // 10 20 20 20 30 30 20
20 10
  vector<int> myvector (myints,myints+9);
  vector<int>::iterator it;

  // using default comparison:
  it = unique (myvector.begin(), myvector.end()); // 10 20 30 20 10 ?  ?  ?
?
                                                  //                ^

  myvector.resize( it - myvector.begin() );       // 10 20 30 20 10

  // using predicate comparison:
  unique (myvector.begin(), myvector.end(), myfunction);   // (no changes)
```

```
  // print out content:
  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 10 20 30 20 10
```

3.6. Reverse:

Dạng:

```
template <class BidirectionalIterator>
  void reverse ( BidirectionalIterator first, BidirectionalIterator last);
```

Ý nghĩa:

- Đảo ngược thứ tự các phần tử trong đoạn [first,last).

Tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).

Ví dụ:

```
// reverse algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  vector<int> myvector;
  vector<int>::iterator it;

  // set some values:
  for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

  reverse(myvector.begin(),myvector.end());       // 9 8 7 6 5 4 3 2 1

  // print out content:
  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 9 8 7 6 5 4 3 2 1
```

4. Sắp xếp:

4.1. Sort:

Dạng 1:

```
template <class RandomAccessIterator>
  void sort ( RandomAccessIterator first, RandomAccessIterator last );
```

Dạng 2:

```
template <class RandomAccessIterator, class Compare>
  void sort ( RandomAccessIterator first, RandomAccessIterator last,
Compare comp );
```

Ý nghĩa:

- Hàm sort có tác dụng sắp xếp các phần tử trong nửa đoạn [first,last] theo thứ tự tăng dần.
- Các phần tử được so sánh bằng cách sử dụng toán tử '<' trong phiên bản đầu tiên, và toán tử 'comp' trong phiên bản thứ 2.

Các tham số:

- first, last: biến lặp truy cập ngẫu nhiên đến vị trí đầu và cuối, bao gồm phần tử đầu tiên first và tất cả các phần tử giữa first và last (không có phần tử cuối cùng).
- comp: hàm so sánh hai đối tượng (có cùng kiểu dữ liệu với đoạn phần tử), trả về true nếu tham số đầu tiên 'bé hơn' tham số thứ hai ('bé hơn' ở đây do người sử dụng tự định nghĩa), và false nếu ngược lại.

Ví dụ:

```
// sort algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  vector<int> myvector (myints, myints+8);               // 32 71 12 45 26
80 53 33
  vector<int>::iterator it;

  // sử dụng toán tử mặc định (operator <):
  sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26
80 53 33

  // sử dụng hàm để so sánh
  sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26
33 53 80)

  // sử dụng đối tượng để so sánh
  sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45
53 71 80)
```

```
  // In ra các phần tử:
  cout << "myvector contains:";
  for (it=myvector.begin(); it!=myvector.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
myvector contains: 12 26 32 33 45 53 71 80
```

Độ phức tạp:
Khoảng $N \times logN$, với N là số phần tử sắp xếp.
Trong trường hợp xấu nhất là N^2 (rất khó xảy ra).

5. Tìm kiếm nhị phân (Đoạn đã sắp xếp):

5.1. lower_bound:

Dạng 1:

```
template <class ForwardIterator, class T>
  ForwardIterator lower_bound ( ForwardIterator first, ForwardIterator last,
                                const T& value );
```

Dạng 2:

```
template <class ForwardIterator, class T, class Compare>
  ForwardIterator lower_bound ( ForwardIterator first, ForwardIterator last,
                                const T& value, Compare comp );
```

Ý nghĩa:

- Trả về con trỏ đến phần tử đầu tiên trong đoạn [first,last) (đã sắp xếp) mà không bé hơn ‘value’. Phép toán so sánh mặc định là toán tử ‘<’ trong dạng 1, hoặc là ‘comp’ trong dạng 2.

Tham số:

- first, last: giống như trong hàm “sort”
- value: phần tử cần tìm kiếm.
- comp: giống như trong hàm “sort”

Ví dụ: xem ví dụ ở phần upper_bound.
Độ phức tạp: Khoảng log(last-first).

5.2. upper_bound:

Dạng 1:

```
template <class ForwardIterator, class T>
  ForwardIterator upper_bound ( ForwardIterator first, ForwardIterator last,
                                const T& value );
```

Dạng 2:

```
template <class ForwardIterator, class T, class Compare>
  ForwardIterator upper_bound ( ForwardIterator first, ForwardIterator last,
                                const T& value, Compare comp );
```

Ý nghĩa:

- Trả về con trỏ đến phần tử đầu tiên trong đoạn [first,last) (đã sắp xếp) mà lớn hơn ‘value’. Phép toán so sánh mặc định là toán tử ‘<’ trong dạng 1, hoặc là ‘comp’ trong dạng 2.

Tham số:

- first, last: giống như trong hàm "sort"
- value: phần tử cần tìm kiếm.
- comp: giống như trong hàm "sort"

Ví dụ:

```
// lower_bound/upper_bound example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  vector<int> v(myints,myints+8);           // 10 20 30 30 20 10 10 20
  vector<int>::iterator low,up;

  sort (v.begin(), v.end());                // 10 10 10 20 20 20 30 30

  low=lower_bound (v.begin(), v.end(), 20); //          ^
  up= upper_bound (v.begin(), v.end(), 20); //                   ^

  cout << "lower_bound at position " << int(low- v.begin()) << endl;
  cout << "upper_bound at position " << int(up - v.begin()) << endl;

  return 0;
}
```

Output:

```
lower_bound at position 3
upper_bound at position 6
```

Độ phức tạp: Khoảng log(last-first).

5.3. binary_search:

Dạng 1:

```
template <class ForwardIterator, class T>
  bool binary_search ( ForwardIterator first, ForwardIterator last,
                       const T& value );
```

Dạng 2:

```
template <class ForwardIterator, class T, class Compare>
  bool binary_search ( ForwardIterator first, ForwardIterator last,
                       const T& value, Compare comp );
```

Ý nghĩa:

- Kiểm tra xem giá trị value có tồn tại trong đoạn đã sắp xếp không.
- Trả về 'true' nếu có một phần tử trong nửa đoạn [first,last) bằng 'value', ngược lại trả về 'false'. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2. Một 'value' a được coi là tương đương (bằng) 'value' b khi: (!(a<b) && !(b<a)) hoặc (!comp(a,b) && !comp(b,a)).

Tham số:

- first, last: giống như trong hàm "sort"
- value: phần tử cần tìm kiếm.
- comp: giống như trong hàm "sort"

Ví dụ:

```
// binary_search example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool myfunction (int i,int j) { return (i<j); }

int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3
2 1

  // sử dụng toán tử so sánh mặc định '<':
  sort (v.begin(), v.end());

  cout << "looking for a 3... ";
  if (binary_search (v.begin(), v.end(), 3))
    cout << "found!\n"; else cout << "not found.\n";

  // sử dụng hàn myfunction để so sánh:
  sort (v.begin(), v.end(), myfunction);

  cout << "looking for a 6... ";
  if (binary_search (v.begin(), v.end(), 6, myfunction))
    cout << "found!\n"; else cout << "not found.\n";

  return 0;
}
```

Output:

```
looking for a 3... found!
looking for a 6... not found.
```

Độ phức tạp: Khoảng log(last-first).

6. Trộn (thực hiện trên đoạn các phần tử đã sắp xếp):

6.1. Merge:

Dạng 1:

```
template <class InputIterator1, class InputIterator2, class OutputIterator>
  OutputIterator merge ( InputIterator1 first1, InputIterator1 last1,
                         InputIterator2 first2, InputIterator2 last2,
                         OutputIterator result );
```

Dạng 2:

```
template <class InputIterator1, class InputIterator2,
          class OutputIterator, class Compare>
  OutputIterator merge ( InputIterator1 first1, InputIterator1 last1,
                         InputIterator2 first2, InputIterator2 last2,
                         OutputIterator result, Compare comp );
```

Ý nghĩa:

- Kết hợp các phần tử đã được sắp xếp trong đoạn [first1,last1) và [first2,last2) vào đoạn mới bắt đầu bằng *'result'* với các phần tử đã được sắp xếp. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- first1, last1, first2, last2: giống như first và last của hàm 'sort'

- result: trỏ đến phần tử đầu tiên của đoạn kết quả. Chiều dài của đoạn phần tử này bằng tổng chiều dài của 2 đoạn cần 'merge'.
- comp: giống như hàm 'sort'

Ví dụ:

```
// merge algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int first[] = {5,10,15,20,25};
  int second[] = {50,40,30,20,10};
  vector<int> v(10);
  vector<int>::iterator it;

  sort (first,first+5);
  sort (second,second+5);
  merge (first,first+5,second,second+5,v.begin());

  cout << "The resulting vector contains:";
  for (it=v.begin(); it!=v.end(); ++it)
    cout << " " << *it;

  cout << endl;

  return 0;
}
```

Output:

```
The resulting vector contains: 5 10 10 15 20 20 25 30 40 50
```

Độ phức tạp: tổng số phần tử của hai đoạn đem trộn.

7. Hàng đợi ưu tiên:

7.1. push_heap:

Dạng 1:

```
template <class RandomAccessIterator>
  void push_heap ( RandomAccessIterator first, RandomAccessIterator last );
```

Dạng 2:

```
template <class RandomAccessIterator, class Compare>
  void push_heap ( RandomAccessIterator first, RandomAccessIterator last,
                   Compare comp );
```

Ý nghĩa:

- Đẩy phần tử vào heap.
- Ban đầu có một heap là nửa đoạn [first,last-1), hàm này sẽ đẩy phần tử (last-1) vào vị trí tương ứng của nó.
- Đoạn phần tử là heap có thể xây dựng bằng cách gọi hàm "make_heap" (trình bày ở dưới), sau khi tạo ra, các tính chất của heap sẽ được giữ nguyên bằng cách sử dụng "push_heap" và "pop_heap" tương ứng để bổ sung và loại bỏ phần tử.
- Cần phải nói thêm rằng là: tính chất của heap đó là phần tử đầu tiên luôn có độ ưu tiên lớn nhất (theo phép so sánh), phần tử được 'pop' luôn là phần tử có độ ưu tiên lớn nhất, mỗi khi 'push' vào, phần tử sẽ được đẩy vào chỗ mà phù hợp với độ ưu tiên của nó (xem thêm cấu trúc dữ liệu heap để hiểu rõ). Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- first, last: giống như trong hàm "sort"
- comp: giống như trong hàm "sort"

Ví dụ: (Xem ở hàm sort_heap)
Độ phức tạp: log (last – first).

7.2. pop_heap:

Dạng 1:

```
template <class RandomAccessIterator>
  void pop_heap ( RandomAccessIterator first, RandomAccessIterator last );
```

Dạng 2:

```
template <class RandomAccessIterator, class Compare>
  void pop_heap ( RandomAccessIterator first, RandomAccessIterator last,
                  Compare comp );
```

Ý nghĩa:

- Loại bỏ phần tử có độ ưu tiên lớn nhất ra khỏi heap (chính là phần tử đầu tiên của đoạn phần tử đã là heap). Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- first, last: giống như trong hàm "sort"
- comp: giống như trong hàm "sort"

Ví dụ: (Xem ở hàm sort_heap)
Độ phức tạp: Khoảng (2*log(last-first))

7.3. make_heap:

Dạng 1:

```
template <class RandomAccessIterator>
  void make_heap ( RandomAccessIterator first, RandomAccessIterator last );
```

Dạng 2:

```
template <class RandomAccessIterator, class Compare>
  void make_heap ( RandomAccessIterator first, RandomAccessIterator last,
                   Compare comp );
```

Ý nghĩa:

- Tạo heap từ đoạn phần tử bằng cách sắp xếp lại các phần tử trong nửa đoạn [first,last) theo quy tắc để chúng tạo thành một heap. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.
- Bên trong, một heap là một cây mà mỗi nút liên kết đến các phần tử có độ ưu tiên không lớn hơn nó.

Tham số:

- first, last: giống như trong hàm "sort"
- comp: giống như trong hàm "sort"

Ví dụ: (Xem ở hàm sort_heap)
Độ phức tạp: Khoảng (3*(last-first))

7.4. sort_heap:

Dạng 1:

```
template <class RandomAccessIterator>
  void sort_heap ( RandomAccessIterator first, RandomAccessIterator last );
```

Dạng 2:

```
template <class RandomAccessIterator, class Compare>
  void sort_heap ( RandomAccessIterator first, RandomAccessIterator last,
                   Compare comp );
```

Ý nghĩa:

- Sắp xếp lại các phần tử của heap (không phải theo độ ưu tiên mà là theo thứ tự các phần tử - giống như săp xếp thông thường). Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- first, last: giống như trong hàm "sort"
- comp: giống như trong hàm "sort"

Ví dụ:

```
// range heap example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int myints[] = {10,20,30,5,15};
  vector<int> v(myints,myints+5);
  vector<int>::iterator it;

  make_heap (v.begin(),v.end());
  cout << "initial max heap   : " << v.front() << endl;

  pop_heap (v.begin(),v.end()); v.pop_back();
  cout << "max heap after pop : " << v.front() << endl;

  v.push_back(99); push_heap (v.begin(),v.end());
  cout << "max heap after push: " << v.front() << endl;

  sort_heap (v.begin(),v.end());

  cout << "final sorted range :";
  for (unsigned i=0; i<v.size(); i++) cout << " " << v[i];

  cout << endl;

  return 0;
}
```

Output:

```
initial max heap   : 30
max heap after pop : 20
max heap after push: 99
final sorted range : 5 10 15 20 99
```

Độ phức tạp: Khoảng NlogN (với N = last – first)

8. Min / Max:

8.1. min:

Dạng 1:

```
template <class T> const T& min ( const T& a, const T& b );
```

Dạng 2:

```
template <class T, class Compare>
  const T& min ( const T& a, const T& b, Compare comp );
```

Ý nghĩa:

- Trả về phần tử nhỏ hơn trong hai phần tử. Nếu hai phần tử bằng nhau, trả về phần tử a. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- a,b: hai phần tử so sánh
- comp: giống như trong hàm "sort"

Ví dụ:

```
// min example
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
  cout << "min(1,2)==" << min(1,2) << endl;
  cout << "min(2,1)==" << min(2,1) << endl;
  cout << "min('a','z')==" << min('a','z') << endl;
  cout << "min(3.14,2.72)==" << min(3.14,2.72) << endl;
  return 0;
}
```

Output:

```
min(1,2)==1
min(2,1)==1
min('a','z')==a
min(3.14,2.72)==2.72
```

8.2. max:

Dạng 1:

```
template <class T> const T& max ( const T& a, const T& b );
```

Dạng 2:

```
template <class T, class Compare>
  const T& max ( const T& a, const T& b, Compare comp );
```

Ý nghĩa:

- Trả về phần tử lớn hơn trong hai phần tử. Nếu hai phần tử bằng nhau, trả về phần tử a. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.

Tham số:

- a,b: hai phần tử so sánh
- comp: giống như trong hàm "sort"

Ví dụ:

```
// max example
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
  cout << "max(1,2)==" << max(1,2) << endl;
  cout << "max(2,1)==" << max(2,1) << endl;
  cout << "max('a','z')==" << max('a','z') << endl;
```

```
  cout << "max(3.14,2.73)==" << max(3.14,2.73) << endl;
  return 0;
}
```

Output:

```
max(1,2)==2
max(2,1)==2
max('a','z')==z
max(3.14,2.73)==3.14
```

8.3. next_permutation:

Dạng 1:

```
template <class BidirectionalIterator>
  bool next_permutation (BidirectionalIterator first,
                         BidirectionalIterator last );
```

Dạng 2:

```
template <class BidirectionalIterator, class Compare>
  bool next_permutation (BidirectionalIterator first,
                         BidirectionalIterator last, Compare comp);
```

Ý nghĩa:

- Biến đổi đoạn phần tử về hoán vị kế tiếp của nó theo thứ tự từ điển. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.
- Hàm sẽ trả về 'true' nếu tìm được hoán vị tiếp theo (tức là chưa phải là hoán vị cao nhất theo thứ tự từ điển), ngược lại là 'false'

Tham số:

- first, last: giống như trong hàm "sort".
- comp: giống như trong hàm "sort".

Ví dụ:

```
// next_permutation
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
  int myints[] = {1,2,3};

  cout << "The 3! possible permutations with 3 elements:\n";

  sort (myints,myints+3);

  do {
    cout << myints[0] << " " << myints[1] << " " << myints[2] << endl;
  } while ( next_permutation (myints,myints+3) );

  return 0;
}
```

Output:

```
The 3! possible permutations with 3 elements:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Độ phức tạp: Số các phép so sánh, khoảng (last-first).

8.4. prev_permutation:

Dạng 1:

```
template <class BidirectionalIterator>
  bool prev_permutation (BidirectionalIterator first,
                         BidirectionalIterator last );
```

Dạng 2:

```
template <class BidirectionalIterator, class Compare>
  bool prev_permutation (BidirectionalIterator first,
                         BidirectionalIterator last, Compare comp);
```

Ý nghĩa:

- Biến đổi đoạn phần tử về hoán vị trước đó của nó theo thứ tự từ điển. Phép toán so sánh của đoạn phần tử mặc định là toán tử '<' trong dạng 1, hoặc là 'comp' trong dạng 2.
- Hàm sẽ trả về 'true' nếu tìm được hoán vị trước đó (tức là chưa phải là hoán vị thấp nhất theo thứ tự từ điển), ngược lại là 'false'

Tham số:

- first, last: giống như trong hàm "sort".
- comp: giống như trong hàm "sort".

Ví dụ:

```
// prev_permutation
#include <iostream>
#include <algorithm>
using namespace std;

int main () {
  int myints[] = {1,2,3};

  cout << "The 3! possible permutations with 3 elements:\n";

  sort (myints,myints+3);
  reverse (myints,myints+3);

  do {
    cout << myints[0] << " " << myints[1] << " " << myints[2] << endl;
  } while ( prev_permutation (myints,myints+3) );

  return 0;
}
```

Output:

```
The 3! possible permutations with 3 elements:
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3
```

Độ phức tạp: Số các phép so sánh, khoảng (last-first).

9. Một số bài tập áp dụng:

Bài 1: Hoán vị kế tiếp (Link http://www.spoj.pl/PTIT/problems/BCNEPER/):

Trong bài này, bạn hãy viết chương trình nhận vào một chuỗi (có thể khá dài) các ký tự số và đưa ra màn hình hoán vị kế tiếp của các ký tự số đó (với ý nghĩa là hoán vị có giá trị lớn hơn tiếp theo nếu ta coi chuỗi đó là một giá trị số nguyên).

Chú ý: Các ký tự số trong dãy có thể trùng nhau.

Ví dụ:

123 -> 132

279134399742 -> 279134423799

Cũng có trường hợp sẽ không thể có hoán vị kế tiếp. Ví dụ như khi đầu vào là chuỗi 987.

Dữ liệu vào

Dòng đầu tiên ghi số nguyên t là số bộ test ($1 \le t \le 1000$). Mỗi bộ test có một dòng, đầu tiên là số thứ tự bộ test, một dấu cách, sau đó là chuỗi các ký tự số, tối đa 80 phần tử.

Dữ liệu ra

Với mỗi bộ test hãy đưa ra một dòng gồm thứ tự bộ test, một dấu cách, tiếp theo đó là hoán vị kế tiếp hoặc chuỗi "BIGGEST" nếu không có hoán vị kế tiếp.

Example

```
Input:

3

1 123

2 279134399742

3 987

Output:

1 132

2 279134423799

3 BIGGEST
```

Hướng dẫn:

- Bài này có thể sử dụng thuật toán sinh hoán vị kế tiếp. Nhưng cách đơn giản nhất là áp dụng hàm next_permutation.
- Chương trình mẫu:

```
#include <iostream>
#include <string>
using namespace std;
string nextPermutation(string s) {
	if (next_permutation(s.begin(),s.end())) return s;
	else return "BIGGEST";
}
main() {
	long nTest,test;
	string s;
	cin>>nTest;
	while (nTest) {
		cin>>test>>s;
		printf("%d ",test);
		cout<<nextPermutation(s)<<endl;
		nTest--;
	}
}
```

Bài tập tương tự: http://www.spoj.pl/PTIT/problems/BCPERMU/

Liệt kê hoán vị của n phần tử của một tập gồm các số từ 1->n.

Input

Dòng duy nhất chứa số n (1<=n<=8)

Output

Các hoán vị sắp xếp theo thứ tự từ điển tăng dần.

Example

```
Input:
3

Output:
123
132
213
231
312
321
```

Bài 2:

Cho N (1<=N<=30000) sinh viên, mỗi sinh viên có 3 thông tin là: mã sinh viên, điểm học tập và điểm rèn luyện.

Yêu cầu: Sắp xếp các sinh viên theo thứ tự tăng dần của điểm học tập, nếu 2 sinh viên có điểm học tập bằng nhau thì sắp xếp theo điểm rèn luyện tăng dần.

Dữ liệu:

- Dòng 1: chứa số N
- Dòng 2..N+1: mỗi dòng chứa 3 số nguyên lần lượt là mã sinh viên, điểm học tập và rèn luyện.

Kết quả: Danh sách sau khi đã sắp xếp theo yêu cầu:

- N dòng, mỗi dòng chứa 3 số nguyên mã sinh viên, điểm học tập, điểm rèn luyện.

Lưu ý: Các số trong input và output trên cùng 1 dòng cách nhau bởi dấu cách.

Ví dụ:

```
Input:
3
1 10 9
2 8 7
3 8 8
Output:
2 8 7
3 8 8
1 10 9
```

Hướng dẫn:

- Với N<=30000 bạn cần sử dụng thuật toán sắp xếp có độ phức tạp cỡ khoảng NlogN, nên ta có thể áp dụng hàm *sort* hoặc cấu trúc *heap* trong thư viện algorithm. Khi sử dụng cần viết lại hàm so sánh cho đúng với cách sắp xếp của bài này.
- Chương trình mẫu:
    - Sử dụng hàm *sort*:

```
#include <iostream>
#include <algorithm>
#define maxn 30000
using namespace std;
struct SV {
	long maSV,DHT,DRL;
};
bool comp(SV a,SV b) {
	return (a.DHT<b.DHT || (a.DHT == b.DHT && a.DRL < b.DRL));
}
main() {
	SV a[maxn];
	long n;
	cin >> n;
	for (long i=0;i<n;i++) {
		cin >> a[i].maSV >> a[i].DHT >> a[i].DRL;
	}
	sort(a,a+n,comp);
	for (long i=0;i<n;i++) {
		cout << a[i].maSV << " " << a[i].DHT << " " << a[i].DRL << endl;
	}
}
```

    - Sử dụng heap : Hàm so sánh có thay đổi một chút, vì phần tử ở vị trí *pop* của heap luôn có độ ưu tiên cao nhất (tức là nếu sử dụng phép toán thấp hơn, thì *pop* luôn là phần tử lớn nhất, trong bài này mỗi lần cần lấy ra phần tử *pop* là bé nhất, nên phép toán so sánh sẽ là lớn hơn).

```
#include <iostream>
#include <algorithm>
#define maxn 30000
using namespace std;
struct SV {
	long maSV,DHT,DRL;
};
bool comp(SV a,SV b) {
	return (a.DHT>b.DHT || (a.DHT == b.DHT && a.DRL > b.DRL));
}
main() {
	SV a[maxn];
	long n;
```

```
        cin >> n;
        for (long i=0;i<n;i++) {
                cin >> a[i].maSV >> a[i].DHT >> a[i].DRL;
                if (i>0) push_heap(a,a+i,comp);
        }
        for (long i=0;i<n;i++) {
                /*phần tử thấp nhất luôn là phần tử đầu tiên (a[0]) */
                cout << a[0].maSV << " " << a[0].DHT << " " << a[0].DRL << endl;
                /*loại bỏ phần tử thấp nhất*/
                pop_heap(a,a+n-i,comp);
        }
}
```

Một số bài áp dụng:

- Bài 1: http://www.spoj.pl/PTIT/problems/BCSAPXEP/

Sắp xếp dãy tăng dần.

Input

- Dòng đầu chứa số n (số phần tử của dãy 1<=n<=1000)

- n dòng sau, mỗi dòng là 1 phần tử của dãy (giá trị tuyệt đối không quá 1000)

Output

Mỗi phần tử của dãy in trên 1 dòng, theo thứ tự tăng dần.

Example

```
Input:
3
3
2
1

Output:
1
2
3
```

- Bài 2: http://www.spoj.pl/PTIT/problems/BCTELEPH/

Cho một danh sách các số điện thoại, hãy xác định danh sách này có số điện thoại nào là phần trước của số khác hay không? Nếu không thì danh sách này được gọi là nhất quán. Giả sử một danh sách có chứa các số điện thoại sau:

- Số khẩn cấp: 911

- Số của Alice: 97625999

- Số của Bob: 91125426

Trong trường hợp này, ta không thể gọi cho Bob vì tổng đài sẽ kết nối bạn với đường dây khẩn cấp ngay khi bạn quay 3 số đầu trong số của Bob, vì vậy danh sách này là không nhất quán.

Dữ liệu vào

Dòng đầu tiên chứa một số nguyên $1 \le t \le 40$ là số lượng bộ test. Mỗi bộ test sẽ bắt đầu với số lượng số điện thoại n được ghi trên một dòng, $1 \le n \le 10000$. Sau đó là n dòng, mỗi dòng ghi duy nhất 1 số điện thoại. Một số điện thoại là một dãy không quá 10 chữ số.

Dữ liệu ra

Với mỗi bộ dữ liệu vào, in ra "**YES**" nếu danh sách nhất quán và "**NO**" trong trường hợp ngược lại.

INPUT	OUTPUT
2 3 911 97625999 91125426 5 113 12340 123440 12345 98346	**NO** **YES**

- Bài 3: http://vn.spoj.pl/problems/QBHEAP/

Cho trước một danh sách rỗng. Người ta xét hai thao tác trên danh sách đó:

Thao tác "+V" (ở đây V là một số tự nhiên <= 1000000000): Nếu danh sách đang có ít hơn 15000 phần tử thì thao tác này bổ sung thêm phần tử V vào danh sách; Nếu không, thao tác này không có hiệu lực.

Thao tác "-": Nếu danh sách đang không rỗng thì thao tác này loại bỏ tất cả các phần tử lớn nhất của danh sách; Nếu không, thao tác này không có hiệu lực

Input

Gồm nhiều dòng, mỗi dòng ghi một thao tác. Thứ tự các thao tác trên các dòng được liệt kê theo đúng thứ tự sẽ thực hiện

Output

Dòng 1: Ghi số lượng những giá trị còn lại trong danh sách.

Các dòng tiếp theo: Liệt kê những giá trị đó theo thứ tự giảm dần, mỗi dòng 1 số

Example

Input:

```
+1
+3
+2
+3
-
+4
+4
-
+2
+9
+7
+8
-
```

Output:

```
4
8
2
2
1
```

IV. THƯ VIỆN STRING C++:

- String là một kiểu đặc biệt của container, thiết kế đặc để hoạt động với các chuỗi kí tự.
- Khai báo: `#include <string>`
- Iterator: Tương tự như trong container, string cũng hỗ trợ các iterator như begin, end, rbegin, rend với ý nghĩa như trước.
- Nhập, xuất string:
  - Sử dụng toán tử >> : tương tự như trong C. Nhập đến khi gặp dấu cách.
  - Sử dụng getline: giống như gets trong C. Nhập cả một dòng kí tự (chứa cả dấu cách nếu có) cho string.
  - Xuất string: sử dụng toán tử << như bình thường.

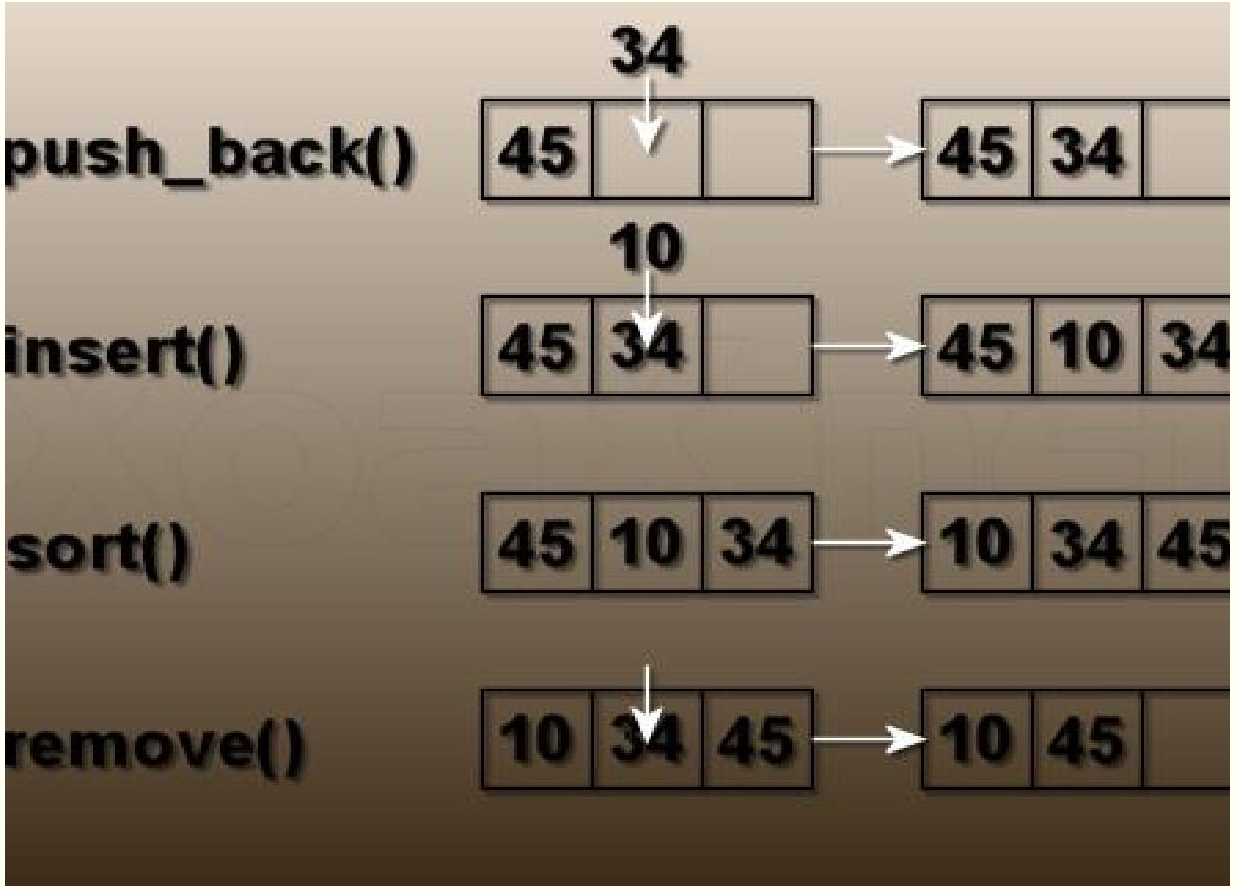
Các hàm thành viên:

- Trong string bạn có thể sử dụng các toán tử "=" để gán giá trị cho string, hay toán tử "+" để nối hai string với nhau. Ngoài ra, khi so sánh hai string với nhau thì cũng dùng

các toán tử so sánh như '>','<' để so sánh 2 string theo thứ tự từ điển. Chức năng này khá giống với xâu ở trong ngôn ngữ pascal.

- Capacity:
  - size: trả về độ dài của string
  - length: trả về độ dài của string
  - clear : xóa string
  - empty: return true nếu string rỗng, false nếu ngược lại
- Truy cập đến phần tử:
  - operator [chỉ_số]: lấy kí tự vị trí chỉ_số của string
- Chỉnh sửa:
  - push_back: chèn kí tự vào sau string
  - insert (n,x): chèn x vào string ở trước vị trí thứ n. x có thể là string, char,...
  - erase (pos,n): xóa khỏi string "n" kí tự bắt đầu từ vị trí thứ "pos".
  - erase (iterator): xóa khỏi string phần tử ở vị trí iterator.
  - replace (pos, size, s1) : thay thế string từ vị trí "pos", số phần tử thay thế là "size" và thay bằng xâu s1.
  - swap (string_cần_đổi): đổi giá trị 2 xâu cho nhau.
- String operations:
  - c_str : chuyển xâu từ dạng string trong C++ sang string trong C.
  - substr (pos,length): return string được trích ra từ vị trí thứ "pos", và trích ra "length" kí tự.

BÀI 9
VECTOR, STRUCT



NỘI DUNG

- Vector
- Vector của vector
- Struct

KHÁI NIỆM VỀ VECTOR

- **Vector** là mảng có thể thay đổi được số phần tử (mảng động)
- Các phần tử lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ
- Cung cấp các phương thức để thao tác với các phần tử
- Khi sử dụng **Vector** cần khai báo thêm:

```
#include <vector>
```

KHAI BÁO VECTOR

- Cú pháp:

```
vector<kieudulieu> tenvector;
```

```
vector<kieudulieu> tenvector(số-phần-tử);
```

```
vector<kieudulieu> tenvector(số-phần-tử, giá-trị);
```

- Ví dụ:

```
vector<string> A;   //Khai báo vector A chưa có phần tử nào
vector<int> A(10);  //Khai báo vector có 10 phần tử
vector<float> B(10, 2.5); //Khai báo có khởi gán giá trị
vector<float> C(B);    //vector C là bản sao của vector B
```

VÍ DỤ

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main()
5  {
6      vector<int> V(3); //khai bao vector V kieu so nguyen
7      V[0] = 5;//Gan gia tri cho phan tu dau tien
8      V[1] = 10;
9      V[2] = 15;
10     for(int i=0; i<V.size(); i++)
11       cout<<V[i]<<" ";//In cac phan tu ra man hinh
12
13   return 0;
14 }
```

MẢNG MỘT CHIỀU

- Ví dụ:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n, a[100];
6      cout<<"nhap so phan tu cua day n=";cin>>n;
7      for(int i=0;i<n;i++)
8      {
9          cout<<"a["<<i<<"]=";
10         cin>>a[i];
11     }
12     cout<<"Day so vua nhap"<<endl;
13     for(int i=0;i<n;i++)
14       cout<<a[i]<<" ";
15     return 0;
16 }
```

SỬ DỤNG VỚI VECTOR

- Ví dụ:

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main()
5  {
6      int n;
7      cout<<"Nhap so phan tu cua day n="; cin>>n;
8      vector<float> V(n); //khai bao vector V co n phan tu
9
10     for(int i=0; i<V.size(); i++)
11     {
12       cout<<"V["<<i<<"]=";
13       cin>>V[i]; //Nhap tung phan tu
14     }
15
16     cout<<"Day vua nhap la:"<<endl;
17     for(int i=0; i<V.size(); i++)
18       cout<<V[i]<<" ";
19   return 0;
20 }
```

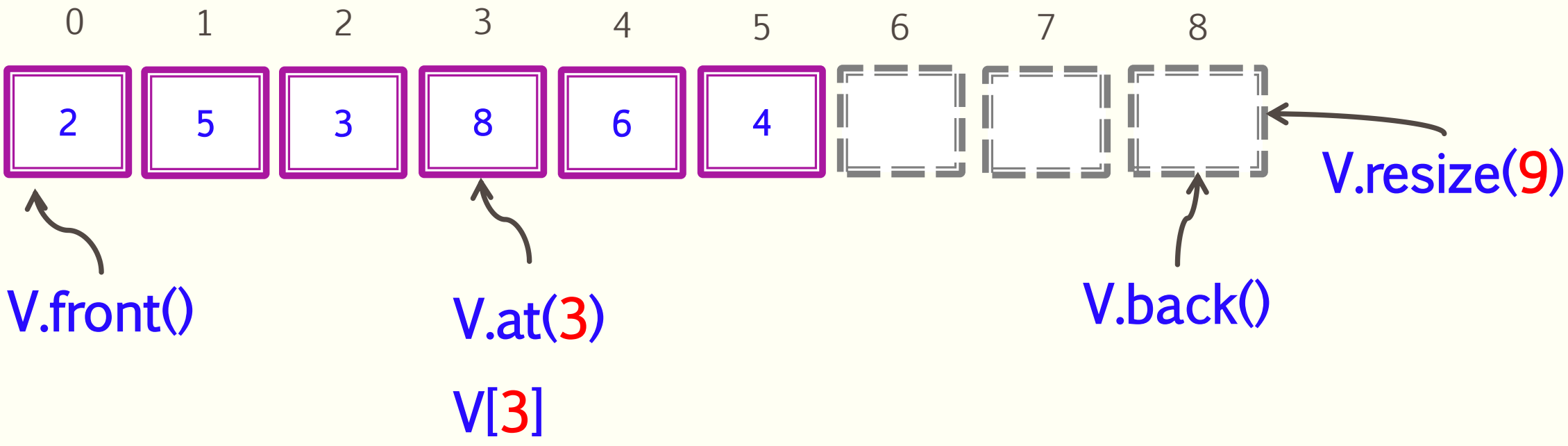
CÁC TOÁN TỬ VÀ PHƯƠNG THỨC

Toán tử/Phương thức	Mô tả
=	Gán vector
[chỉ-số]	Truy nhập tới phần tử của vector theo chỉ số
.size()	Lấy số phần tử của vector
.resize(n)	Thay đổi số phần tử của vector (có n phần tử)
.at(chỉ-số)	Truy nhập tới phần tử của vector theo chỉ số
.front()	Truy nhập vào phần tử đầu tiên của vector
.back()	Truy nhập vào phần tử cuối cùng của vector

BÀI TẬP

- **Bài 1:** **Nhập vào một dãy n số nguyên. Tính tổng dãy số vừa nhập**

CÁC TOÁN TỬ VÀ PHƯƠNG THỨC



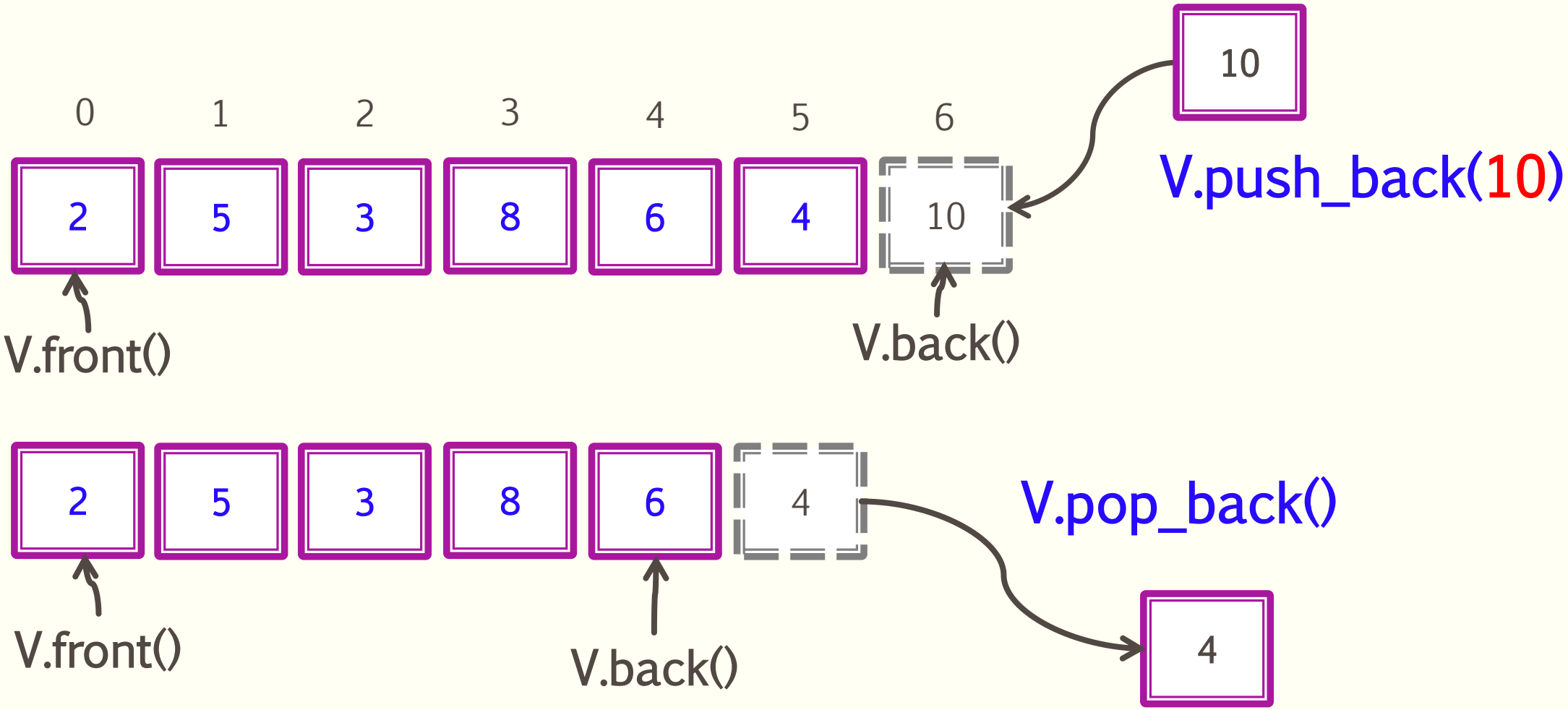
MÀN HÌNH HIỂN THỊ GÌ

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main()
5  {
6      vector<float> V(3); //khai bao vector V co 3 phan tu
7      V[0]=15;
8      V[1]=24;
9      V[2]=10;
10     V.front() += V.back()*2;
11     V.at(2) = 50;
12     V[1] = -V[2];
13     V.resize(4);
14     for(int i=0; i<V.size(); i++)
15       cout<<V[i]<<" ";
16   return 0;
17 }
```

CÁC PHƯƠNG THỨC (tiếp)

Phương thức	Mô tả
.push_back(pt)	Thêm phần tử **pt** vào cuối dãy. Số phần tử của dãy tăng lên 1.
.pop_back()	Xoá phần tử khỏi dãy. Số phần tử của dãy giảm 1
.insert(pos, giatri)	Chèn 1 phần tử vào vị trí **pos** của dãy
.insert(pos, n, giatri)	Chèn **n** phần tử vào vị trí **pos** của dãy
.erase(pos)	Xóa phần tử vị trí thứ **pos** của dãy.
.erase(vt1, vt2)	Xóa phần tử từ vị trí 1 đến vị trí 2 trong dãy
.clear()	Xóa hết các phần tử của vector
.swap(vector2)	Hoán đổi 2 vector

CÁC PHƯƠNG THỨC (tiếp)



VÍ DỤ

Nhập vào một dãy số thực cho đến khi gặp số 0. Tính tổng dãy vừa nhập.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main()
5  {
6      vector<double> V;
7      double x,tong=0;
8      do
9      {
10         cout<<"Nhap vao 1 so:"; cin>>x;
11         V.push_back(x);//Them phan tu vao cuoi day
12     } while (x!= 0);
13     V.pop_back();//Xoa phan tu cuoi cung (chua gia tri 0)
14     cout<<"Day vua nhap la:"<<endl;
15     for(int i=0;i<V.size();i++)
16        {
17          cout<<V[i]<<" ";
18          tong+= V[i];
19        }
20     cout<<"\nTong cac so ="<<tong;
21   return 0;
22 }
```

THÊM PHẦN TỬ VÀO DÃY

```cpp
#include <iostream>
#include <vector>
using namespace std;
void Inday(vector<int> v)
{
    for(int i=0; i<v.size(); i++)
      cout<<v[i]<<"  ";
}
int main()
{
    vector<int> v;
    for(int i=1; i<10; i++)
       v.push_back(i);
    cout<<"Day ban dau: "<<endl;
    Inday(v);
    v.insert(v.begin()+2, 10);//Them vao vi tri thu 2
    v.insert(v.begin()+5, 3,20);//Them vao 3 so vao vi tri thu 5
    v.insert(v.end(),50);//Them vao cuoi day
    cout<<"\nDay sau khi them"<<endl;
    Inday(v);
  return 0;
}
```

XÓA PHẦN TỬ TRONG DÃY

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5  void Inday(vector<int> A, string tbao)
6  {
7      cout<<tbao<<endl;
8      for(int i=0; i<A.size(); i++)
9        cout<<A[i]<<"  ";
10 }
11 int main()
12 {
13     vector<int> v;
14     for(int i=1; i<10; i++)
15        v.push_back(i);
16     Inday(v, "Day ban dau:");
17     v.erase(v.begin());//Xoa phan tu dau tien
18     Inday(v,"\nDay da xoa phan tu dau tien:");
19     v.erase(v.end()-3, v.end());//Xoa 3 phan tu cuoi
20     Inday(v,"\nXoa 3 phan tu cuoi:");
21   return 0;
22 }
```

BÀI TẬP

- Bài 2: **Nhập vào một dãy n số nguyên. Đưa ra dãy mới toàn các số chẵn.**

- Bài 3: **Nhập vào một dãy n số thực. Đưa ra dãy mới là dãy đảo ngược của dãy ban đầu.**

BÀI TẬP

- **Bài 4:** Đọc vào n số nguyên từ bàn phím. Xóa bỏ số ở vị trí thứ k của dãy, với k đọc vào từ bàn phím. Đưa dãy đã xóa ra màn hình.

- **Bài 5:** Đọc vào n số (2<n<50). Đọc thêm một số x, chèn số x vào vị trí thứ 2 của dãy. Đưa dãy sau khi chèn ra màn hình

- **Bài 6:** Đọc vào n số. Đọc thêm một số x, chèn số x vào vị trí thứ k của dãy với k đọc vào từ bàn phím. Nếu k>n thì thêm vào vị trí thứ n. Đưa dãy sau khi chèn ra màn hình

MẢNG HAI CHIỀU – VECTOR CỦA VECTOR

- Ma trận:

$$\begin{bmatrix} 3 & 4 & 5 & 2 \\ 2 & 7 & 6 & 4 \\ 8 & 5 & 9 & 1 \end{bmatrix}$$

- Mảng 2 chiều:

```
int a[3][4];
```

- Vector:

```
vector<vector<int> > a(3, vector<int>(4));
```

Dấu cách

MẢNG HAI CHIỀU

- Ví dụ:

```
#include <iostream>
using namespace std;
int main ()
{
   int a[20][30]; //khai bao ma tran
   int i, j, cot, hang;
   cout<<"Nhap so hang, cot cua ma tran: ";
   cin>>hang>>cot;
   for(i=0; i< hang; i++)//nhap gia tri cho mang
     for(j=0; j<cot; j++)
       {
        cout<<"a["<<i<<"]["<<j<<"]="; cin>>a[i][j];
       }
    cout<<"\nMa tran vua nhap la:"<<endl;
    for(i=0; i< hang; i++)//hien thi
    {
       for(j=0; j<cot; j++)
          cout<<a[i][j]<<" ";
        cout<<endl;
     }
}
```

VECTOR CỦA VECTOR

- Ví dụ:

```
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
   int i, j, cot, hang;
   cout<<"Nhap so hang, cot cua ma tran: ";
   cin>>hang>>cot;
   vector<vector<int> > a(hang, vector<int> (cot));//Khai bao ma tran
   for(i=0; i< hang; i++)//nhap gia tri cho mang
     for(j=0; j<cot; j++)
       {
        cout<<"a["<<i<<"]["<<j<<"]="; cin>>a[i][j];
       }
    cout<<"\nMa tran vua nhap la:"<<endl;
    for(i=0; i< hang; i++)//hien thi
    {
       for(j=0; j<cot; j++)
          cout<<a[i][j]<<" ";
        cout<<endl;
     }
}
```

KIỂU CẤU TRÚC - STRUCT

- Là kiểu dữ liệu mới với các phần tử có **kiểu dữ liệu khác nhau**
- **Định nghĩa kiểu cấu trúc:**

```
struct ten
{
   kieudulieu ten-truong-1;
   kieudulieu ten-truong-2;
  ....
   kieudulieu ten-truong-n;
}
```

KIỂU CẤU TRÚC - STRUCT

- Ví dụ:

```
struct Sinhvien
{
   string hoten;
   int namsinh;
   string diachi;
   bool gioitinh;
}
```

KIỂU CẤU TRÚC - STRUCT

- Ví dụ:

```
#include <iostream>
#include <cmath>
using namespace std;
struct Toado
{
    float x;
    float y;
};
float Tinhdodai(Toado a, Toado b){
    return sqrt(pow(a.x-b.x,2) + pow(a.y-b.y,2));;
}
int main ()
{
   Toado A,B,C;
   cout<<"Nhap vao toa do diem A:"; cin>>A.x>>A.y;
   cout<<"Nhap vao toa do diem B:"; cin>>B.x>>B.y;
   cout<<"Nhap vao toa do diem C:"; cin>>C.x>>C.y;
   cout<<"\nDo dai canh AB="<<Tinhdodai(A,B);
   cout<<"\nDo dai canh AC="<<Tinhdodai(A,C);
   cout<<"\nDo dai canh BC="<<Tinhdodai(B,C);
   return 0;
}
```

BÀI TẬP

- **Bài 7:** Tạo một struct Thời gian có 3 trường là giờ, phút, giây. Nhập vào 2 thời điểm, tính khoảng cách ra giây giữa 2 thời điểm trên.

- **Bài 8:** Tạo một struct Nhân viên có 2 trường Họ tên và Lương. Nhập vào danh sách n nhân viên, đưa ra nhân viên có mức lương cao nhất.

ÔN TẬP!