# CƠ BẢN VỀ C++

## Chương trình C++ đầu tiên

Để thuận tiện diễn giải chương trình C++ đầu tiên, các tác giả dùng cột bên trái để đánh số các dòng lệnh của chương trình ở cột bên phải như bảng dưới đây:

|  |  |
| --- | --- |
| Số hiệu dòng | Chương trình |
| 1 | *//Chuong trinh dau tien* |
| 2 | #include <iostream> |
| 3 | #include <conio.h> |
| 4 | using namespace std; |
| 5 | int main() |
| 6 | { |
| 7 | cout << "Chao mung cac ban hoc C++"; |
| 8 | cout<<'\n'; |
| 9 | getch(); |
| 10 | return 0; |
| 11 | } |

*Dòng 1.* Dấu // ở đầu dòng là dấu chú thích, những chữ đi sau dấu này trên cùng một dòng chỉ nhằm giải thích rõ thêm về chương trình, không phải là lệnh thực hiện của chương trình.

*Dòng 2.* #include là chỉ thị tiền xử lý định hướng cho trình dịch. #include <iostream> là định hướng cho trình dịch biết chương trình này cần sử dụng thư viện chuẩn iostream.

*Dòng 3.* #include <conio.h> là định hướng cho trình dịch biết chương trình này cần sử dụng thư viện chuẩn conio.h.

*Dòng 4.* Khai báo dùng các tên của không gian tên trong thư viện bản mẫu chuẩn STD (Standard Template Library).

*Dòng 5.* Tên hàm chính là *main*. Hàm này luôn yêu cầu trả lại giá trị nguyên kiểu *int*.

*Dòng 6.* Dấu { báo hiệu bắt đầu hàm chính.

*Dòng 7*. Xuất dòng chữ “Chao mung cac ban hoc C++” ra màn hình. Trong đó cout là luồng ra chuẩn (thường là màn hình) đã được định nghĩa trước trong thư viện iostream.Còn kí hiệu << là phép toán gửi dòng chữ này đến luồng ra.

*Dòng 8.* Gửi kí tự ‘\n’ là ký tự điều khiển xuống dòng tới luồng ra. Kết quả sẽ điều chỉnh con trỏ trên màn hình về đầu dòng tiếp theo.

*Dòng 9.* Lệnh dừng kết quả đang hiện trên màn hình cho đến khi người dùng nhấn một phím nào đó. Lệnh này đòi hỏi phải khai báo dòng 3 (định hướng tiền xử lý tìm tới thư viện conio.h).

*Dòng 10.* Hàm chính trả về giá trị 0 (hàm chính kết thúc bình thường).

*Dòng 11.* Dấu } là dấu kết thúc hàm chính.

## Giao tiếp với bàn phím và màn hình

Trong C++, luồng (stream) là nguồn cho nhận hoặc xuất kí tự.

*Luồng ra* cho phép ghi (lưu trữ/xuất) lần lượt các kí tự trong luồng này tới tệp/màn hình (màn hình coi là tệp ra chuẩn).

Luồng ra chuẩn có tên là cout cho phép xuất các kí tự lên màn hình.

*Luồng vào chuẩn* là cin cho phép gán dữ liệu nhập từ bàn phím vào các biến.

Các luồng vào/ra được định nghĩa trong tệp tiêu đề iostream, vì vậy khi muốn dùng các luồng cout hoặc cin cần có dòng chỉ thị:

#include <iostream>

ở đầu chương trình.

**Ví dụ**: Nhập hai số nguyên n và m từ bàn phím. Viết chương trình hiện tổng n+m trên màn hình.

#include <iostream>

#include <conio.h> *// Tệp tiêu đề chứa lệnh tác động tới màn hình, bàn phím*

using namespace std;

long n,m; *// Khai báo hai biến n và m kiểu số nguyên 32 bit*

int main(void) {

cout << "Nhap n, m: "; *//Xuất dòng chữ ra màn hình*

cin >> n >> m; *// Nhập giá trị gõ từ bàn phím vào biến n và m*

cout << "Tong cua n va m la: "; *//Xuất dòng chữ này ra màn hình*

cout << n+m; *//Xuất tổng của n và m ra màn hình*

cout << '\n'; *// Điều khiển xuống dòng trên màn hình*

getch(); *// Giữ màn hình kết quả cho đến khi người dùng nhấn phím nào đó*

return 0;

}

Chương trình có thể viết gọn lại là:

#include <iostream>

#include <conio.h>

using namespace std;

long n,m; *// Khai báo hai biến n và m có kiểu số nguyên* ***long***

int main(void) {

cout << "Nhap n, m: ";

cin >> n >> m;

cout << "Tong cua n va m la: " << n+m << '\n';

getch();

return 0;

}

Để hiện trên màn hình số thực x với số chữ số thập phân có nghĩa nào đó (giả sử là 4 chữ số thập phân), trước tiên cần khai báo:

#include <iomanip>

và trước khi xuất x, cần có dòng lệnh:

cout << fixed << setprecision(4);

Ví dụ:

#include <iostream>

#include <iomanip>

using namespace std;

float x=3.14156972;

int main() {

cout << fixed << setprecision(4) << x;

system("pause");

return 0;

}

Sẽ cho kết quả trên màn hình là 3.1416

## Kiểu dữ liệu chuẩn, hằng, biến, lệnh, các phép toán,

### Các kiểu dữ liệu chuẩn

Các biến trong chương trình có thể thuộc nhiều kiểu dữ liệu khác nhau. Ví dụ kích thước hình chữ nhật có kiểu số thực, sĩ số học sinh có kiểu số nguyên, các kí tự cũng có kiểu của nó (gọi là kiểu kí tự, hay là char), đánh giá biểu thức là đúng hay sai cần dùng kiểu logic (gọi là kiểu bool)… Căn cứ vào số đơn vị bộ nhớ (là bao nhiêu byte) khi lưu trữ các biến trong bộ nhớ, mà phân loại thành những kiểu dữ liệu chuẩn sau đây:

|  |  |  |  |
| --- | --- | --- | --- |
| Tên kiểu | Số byte bộ nhớ | Mô tả kiểu | Miền giá trị, hoặc trị tuyệt đối |
| char | 1 | Kí tự (còn gọi là kiểu số nguyên 8 bit) | Có dấu: từ -128 đến 127  Không dấu: từ 0 đến 255 |
| short | 2 | Số nguyên 16 bit | Có dấu: -32763 đến 32762  Không dấu: 0 đến 65535 |
| long | 4 | Số nguyên 32 bit | Có dấu:-2147483648 đến 2147483647  Không dấu: 0 đến 4294967295 |
| long long | 8 | Số nguyên  64 bit | Có dấu: -263 đến 263-1  Không dấu: 0 đến 264-1 |
| int | (\*) | Số nguyên | (\*) Như short hoặc như long (tùy thuộc hệ thống) |
| float | 4 | Số thực (Dấu phảy động) | Trị tuyệt đối: 3.4E-38 →3.4E+38  (7 chữ số có nghĩa) |
| double | 8 | Số thực với độ chính xác gấp đôi | Trị tuyệt đối: 1.7E-308→1.7E308  (15 chữ số có nghĩa) |
| long double | 12 | Số thực với độ chính xác cao hơn double | Trị tuyệt đối:1.2E-4932→1.2E+4932  (19 chữ số có nghĩa) |
| bool | 1 | Kiểu lôgic | true hoặc false (đúng hoặc sai) |
| void |  | Không kiểu |  |

*Lưu ý.* Kiểu *void* là kiểu *không kiểu*. Khi muốn cho dữ liệu không có kiểu thì chuyển kiểu cho nó thành kiểu void.

### Tên, hằng và biến

***Tên.*** Mọi đối tượng trong chương trình đều phải được đặt tên theo quy tắc. Trong C++, quy tắc đặt tên là: *Tên chỉ chứa các kí tự chữ cái, chữ số, dấu gạch dưới; không được bắt đầu bằng chữ số; không được trùng với từ khóa* (là những tên đã được quy định dùng với một ý nghĩa đã định trước). Chú ý: Trong C++ luôn luôn phân biệt chữ hoa và chữ thường, ví dụ hai tên là abc và Abc là khác nhau.

***Hằng.*** Hằng là đại lượng không thay đổi giá trị ban đầu. Hằng cũng được đặt tên theo đúng qui định về tên. Ví dụ dòng lệnh sau khai báo hằng *max* kiểu số nguyên 32 bit và giá trị là 109*:*

const **long** max=1000000000;

hoặc

hằng *xuongdong* có giá trị là kí tự điều khiển xuống dòng ’\n’ (còn viết là endl) được khai báo như sau:

const char xuongdong=’\n’;

***Biến.*** Biến là đại lượng có giá trị có thể thay đổi trong quá trình thực hiện chương trình. Biến được khai báo bên trong một hàm thì chỉ bị tác động của lệnh bên trong hàm này, chúng gọi là các biến cục bộ. Biến được khai báo bên ngoài mọi hàm thì có thể bị tác động làm thay đổi giá trị của bất kỳ lệnh nào tại bất kì vị trí nào của chương trình, chúng gọi là biến toàn cục.

Ví dụ khai báo các biến dai, rong, dientich kiểu số thực với 15 chữ số có nghĩa:

**double** dai, rong, dientich;

### Câu lệnh, lệnh gán

Câu lệnh là những câu được viết theo đúng ngữ pháp của ngôn ngữ, để thể hiện các thao tác cần thực hiện của chương trình. Có câu lệnh đơn và câu lệnh ghép. Kết thúc mỗi câu lệnh là dấu chấm phảy. Câu lệnh ghép còn gọi là một khối lệnh gồm nhiều câu lệnh đơn nằm giữa cặp mở ngoặc ({) và đóng ngoặc (}).

*Câu lệnh gán* kí hiệu bởi dấu bằng (=) dùng để gán giá trị của hằng hoặc của biểu thức cho biến. Ví dụ:

dai=5;

rong=3;

dientich = dai\*rong;

### Các phép toán cơ sở.

* ***Các phép toán số học (*** +, -, \*, /, % ***)***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cộng | Trừ | Nhân | Chia | Lấy dư trong phép chia |
| + | - | \* | / | % |

Ví dụ.

a=143;

b=20;

c=a+b; *// gán cho c giá trị 163 (là tổng của 143 và 20)*

c=a%b; *// gán cho c giá trị 3 là số dư trong phép chia 143 cho 20*

a=a+10;*// gán cho a giá trị mới bằng giá trị cũ cộng thêm 10*

a+=10;*// chính là lệnh a=a+10;*

a/=10; *//chính là lệnh a=a/10; …*

* ***Phép toán tăng một đơn vị, giảm một đơn vị***

|  |  |
| --- | --- |
| x=12; |  |
| y=++x; | *//Tăng x một đơn vị, sau đó gán cho y (x=13, y=13)* |
| x=12; |  |
| y=x++; | *//Gán cho y giá trị của x, sau đó tăng x một đơn vị (y=12, x=13)* |
| x=20; |  |
| y=--x; | *//Giảm x một đơn vị, sau đó gán cho y (x=19, y=19)* |
| x=20; |  |
| y=x--; | *//Gán cho y giá trị của x, sau đó giảm x một đơn vị (y=20, x=19)* |

Các phép toán quan hệ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bằng nhau | Khác nhau | Lớn hơn | Nhỏ hơn | Không nhỏ hơn | Không lớn hơn |
| == | != | > | < | >= | <= |

Ví dụ.

(2\*4==8) là biểu thức đúng (có giá trị *true*),

(3!=4) là biểu thức đúng (có giá trị *true*)

(x\*x<0) là biểu thức sai (có giá trị *false*)

* ***Các phép toán lôgic***

|  |  |  |
| --- | --- | --- |
| Phép **phủ định** (not) | Phép **và** (and) | Phép **hoặc** (or) |
| ! | && | || |

*Phủ định của true* là *false*, ngược lại *phủ định của false* là *true*.

Khi hai biểu thức a và b cùng có giá trị true thì a&&b là true, những trường hợp còn lại thì a&&b nhận giá trị false

Khi hai biểu thức a và b cùng nhận giá trị false thì a||b là false, những trường hợp còn lại thì a||b nhận giá trị true.

* ***Phép toán điều kiện***

Phép toán điều kiện được viết như sau:

<biểu thức logic> ? <giá trị thứ 1> : <giá trị thứ 2>;

Nếu *biểu thức logic* có giá trị *true* thì phép toán cho *giá trị thứ nhất*, ngược lại cho *giá trị thứ hai*. Ví dụ:

a=(4>2) ? 1 : 5; *// a được gán bằng 1 vì biểu thức 4>2 là true*

x= (a>b) ? a : b; *// x là giá trị lớn nhất của a và b*

Ký hiệu ? kết hợp với các dấu >, < và phép gán còn được dùng với ý nghĩa sau:

a <?= b; *// a được gán bằng giá trị nhỏ nhất trong hai giá trị a và b*

a >?= b; *// a được gán bằng giá trị lớn nhất trong hai giá trị a và b*

* ***Phép toán chuyển kiểu***

Trong C++ các kiểu dữ liệu được phân biệt rất ngiêm khắc. Các biến không thể nhận giá trị có kiểu khác với kiểu đã khai báo cho nó. Vì vậy trong một số trường hợp cần chuyển kiểu cho biến. C++ sẽ tự động chuyển kiểu cho phù hợp quy luật này hoặc cung cấp các hàm chuyển kiểu để người dùng tự xử lý.

*Ví dụ 1:* Giả sử a có kiểu **float**, b có kiểu **long**, c có kiểu **int** thì b\*c+2\*a được C++ tự động cho nó có kiểu **float** (kiểu có kích cỡ bộ nhớ lớn nhất), vì vậy biến x muốn nhận giá trị này thì biến x phải được khai báo trước là kiểu **float**.

*Ví dụ 2:* j=5.1234 (kiểu float), i có kiểu int thì có thể dùng lệnh:

i= int(j); hoặc i= (int) j;// i nhận giá trị nguyên của j là 5.

* ***Phép toán tính kích cỡ của biến***

Để đo kích thước lưu trữ (đo bằng byte) của một biến, của một kiểu chuẩn trong bộ nhớ, ta dùng hàm sizeof().

Ví dụ a:=sizeof(char); thì a=1.

### Bộ nhớ và kiểu con trỏ

Bộ nhớ máy tính gồm nhiều ô nhớ đơn vị (1 byte) xếp liền nhau. Các ô nhớ được đánh địa chỉ. C/C++ cung cấp công cụ để người lập trình có thể can thiệp bộ nhớ của máy tính ở mức độ thấp giúp cho việc lập trình có nhiều uyển chuyển và chạy nhanh hơn so với những ngôn ngữ lập trình khác. Khi thực hiện chương trình, các giá trị của biến được lưu trữ trong một số ô của bộ nhớ máy tính. Chúng ta có thể thông qua tên của biến để lấy các giá trị này.

Ngoài ra, chúng ta cũng có thể truy xuất thẳng vào địa chỉ của ô nhớ để lấy dữ liệu tại ô nhớ đó.

a) ***Toán tử lấy địa chỉ (&) và biến con trỏ***

Khi chúng ta khai báo một biến, chúng ta không biết biến này sẽ được đặt ở những ô nhớ nào. Muốn lấy địa chỉ các ô nhớ này, có thể thực hiện bằng cách đặt trước tên biến một toán tử lấy địa chỉ là &.

*Ví dụ* 1: Gán:

p\_x=&x;

thì địa chỉ ô nhớ đầu tiên trong các ô nhớ lưu giá trị của x được gán cho biến p\_x.

Biến p\_x gọi là biến con trỏ. Biến con trỏ là biến dùng để lưu địa chỉ trong bộ nhớ. Trong ví dụ trên, biến p\_x lưu địa chỉ ô nhớ chứa giá trị của biến x. Ta nói con trỏ p\_x trỏ tới địa chỉ ô nhớ của biến x.

b) ***Toán tử tham chiếu \****

Ngược lại, chúng ta có thể truy xuất trực tiếp đến giá trị một biến được trỏ tới bởi biến con trỏ bằng cách đặt trước tên biến con trỏ một dấu sao (\*).

*Ví dụ* 2: Câu lệnh z= \*p\_x; gán cho z giá trị của ô nhớ mà p\_x trỏ tới (chính là giá trị của x trong ví dụ 1).

c) ***Khai báo biến kiểu con trỏ***

Khai báo của một biến con trỏ sẽ có dạng sau:

***<type> \* <tên biến con trỏ>*;**

trong đó ***type*** là kiểu dữ liệu trong ô nhớ được con trỏ trỏ tới.

*Ví dụ*. Khai báo 3 biến con trỏ kiểu int, char và float như sau:

int \* p1;

char \* p2;

float \* p3;

Các biến p1, p2, p3 trỏ tới các kiểu dữ liệu khác nhau nhưng cả ba đều là biến con trỏ và chúng chiếm một lượng bộ nhớ như nhau (nghĩa là có kích thước như nhau; kích thước của một biến con trỏ tùy thuộc vào hệ điều hành). Nhưng dữ liệu mà chúng trỏ tới không chiếm lượng bộ nhớ như nhau, p1 trỏ tới dữ liệu kiểu **int** chiếm 2 byte, p2 trỏ tới dữ liệu kiểu **char** chiếm 1 byte và p3 trỏ tới dữ liệu kiểu **float** chiếm 4 byte.

Ta gọi int \* là kiểu con trỏ int, char \* là kiểu con trỏ ký tự, float \* là kiểu con trỏ float.

Tương tự khai báo biến p là biến con trỏ không kiểu như sau:

void \* p;

Giả sử x là biến con trỏ kiểu long, muốn ép kiểu thành con trỏ không kiểu thì viết:

p=(void \*) x;

Sau khi ép x thành con trỏ kiểu void, theo một yêu cầu nào đó chúng ta lại có thể ép nó trở lại kiểu cũ (là long chẳng hạn): (long \*) p; và dùng toán tử tham chiếu \* để lấy giá trị của x trỏ tới để gán cho biến y (kiểu long chẳng hạn) bằng lệnh:

y= \* (long \*) p;

## Các cấu trúc điều khiển

### Cấu trúc rẽ nhánh if – else

*Dạng khuyết:*

if (điều kiện) <thực hiện lệnh>;

Ví dụ: Nếu số a lớn hơn 0 thì thông báo a là số dương:

if (a>0) cout <<”a la so duong”;

*Dạng đầy đủ:*

if (điều kiện) <thực hiện lệnh 1>;

else <thực hiện lệnh 2>;

Ví dụ: Nếu số a lớn hơn 0 thì thông báo a là số dương, còn không thì thông báo a là số không dương.

if (a>0) cout <<”a la so duong”;

else cout <<”a la so khong dương”;

### Cấu trúc lặp while

Dạng:

while (điều kiện) {

<các lệnh>;

}

*Ý nghĩa.* Khi biểu thức điều kiện có giá trị đúng thì thực hiện các lệnh trong vòng lặp sau đó quay về kiểm tra lại điều kiện. Quá trình tiếp tục cho đến khi điều kiện sai thì thoát vòng lặp.

Ví dụ: Phân tích số a dương chẵn thành dạng a=d.2s (s≥0;d lẻ)

s = 0;

while (a%2==0) {

a=a/2; s++;

}

d=a;

Sau đoạn chương trình này, sẽ có giá trị của *d* và *s*

### Cấu trúc lặp do-while

do {

<các lệnh>;

} while (điều kiện);

*Ý nghĩa.* Thực hiện lệnh trong vòng lặp, sau đó kiểm tra điều kiện, nếu điều kiện còn đúng thì quay về đầu vòng lặp để thực hiện các lệnh trong vòng lặp. Quá trình tiếp tục cho đến khi điều kiện sai thì thoát vòng lặp.

Ví dụ: Phân tích số a dương chẵn thành dạng a=d.2s (s≥0;d lẻ)

s = 0;

do {

a/=2; /*/ có nghĩa là a nhận giá trị mới bằng nửa giá trị cũ;*

s++;

} while (a%2==0);

d=a;

*Lưu ý.* Với a là số chẵn thì ví dụ trong mục b và c đều giải đúng, nhưng khi ban đầu a là số lẻ thì giải như ví dụ trong mục c là sai. Tại sao?

### Cấu trúc lặp for

Dạng:

for (khởi tạo; điều kiện lặp; cập nhật ) {

<các lệnh>

}

*Ví dụ:* Thực hiện nhập N phần tử của mảng A từ bàn phím và tính trung bình cộng của chúng, xuất kết quả ra màn hình với độ chính xác tới 2 chữ số thập phân sau dấu phẩy:

#include <iostream>

#include <conio.h>

#include <iomanip>

using namespace std; *// Khai báo dùng Không gian các tên mẫu*

int n, sum, a[100];

int main() {

cout << "Nhap so phan tu cua mang: ";

cin >> n;

cout << "Nhap cac phan tu cua mang " << '\n';

sum = 0;

for (int i=0; i!=n; i++) {

cout <<"Nhap phan tu thu " << i+1 << " = ";

cin>> a[i];

sum +=a[i]; *//sum= sum + a[i];*

}

cout << fixed << setprecision(2);

cout << "Dap so: " <<(float) sum/n;

getch();

}

## Kiểu định nghĩa

1. Định nghĩa một hằng bằng một tên:

#define *tên* [giá trị]

Ví dụ. Khai báo hằng 1000000 bởi tên MaxN như sau:

#define MaxN 1000000

Với định nghĩa này thì MaxN trong toàn bộ chương trình luôn luôn có giá trị không đổi là 106.

1. Định nghĩa một đoạn chương trình bằng một tên:

#define *tên (<tham biến 1>,<tham biến 2>,…) lệnh*

Ví dụ. Khai báo đoạn chương trình tìm giá trị lớn nhất của hai số a và b là:

#define Max(x,y) (x>y?x:y)

Hoặc:

#define Max(x,y) (x>?=y)

Chương trình sau tìm số lớn nhất trong hai số nhập từ bàn phím:

#include <iostream>

using namespace std;

#define Max(x,y) (x>?=y)

int x,y;

int main(){

cout << "Nhap hai so nguyen: ";

cin >> x >> y;

cout << "So lon nhat la: ";

cout << Max(x,y)<<endl;

system("pause");

}

## Kiểu cấu trúc Bản ghi

Một biến quản lí các dữ liệu gồm một số kiểu khác nhau được khai báo thuộc kiểu **struct** (kiểu cấu trúc). Các biến thành phần tạo nên kiểu cấu trúc gọi là các *trường* của kiểu cấu trúc đó.

* **Khai báo kiểu struct như sau:**

struct [<Tên\_kiểu>] {

[<Kiểu\_biến\_1> <Tên\_biến\_trường1\_1[, Tên\_biến\_trường1\_2, ...]>] ;

[<Kiểu\_biến\_2> <Tên\_biến\_trường2\_1[, Tên\_biến\_trường2\_2, ...]>] ;

……………..

} [<danh sách các biến kiểu cấu trúc>] ;

*Ví dụ*

struct hoc\_sinh {

int sbd;

char ht[30];

char ngsinh[9]; // “15/09/92”+ký tự ’\0’

bool nam\_nu;

double toan, ly, ngoaingu;

bool do\_truot;

};

Trong cấu trúc hoc\_sinh có các trường: sbd, ht, ngsinh, nam\_nu, toan, ly, ngoaingu, do\_truot.

* **Khai báo biến có kiểu cấu trúc:**

Cách 1

struct Tên\_kiểu {

<Khai báo các biến thành phần>

} <danh sách các biến>;

Cách 2

struct Tên\_kiểu {

<Khai báo các biến thành phần>

};

Tên\_kiểu <Tên\_biến>;

*Ví dụ*. Có thể khai báo hai biến *x* và *y* quản lý thông tin về hai học sinh x và y nào đó) và mảng lop[60] quản lý thông tin về một lớp nào đó như sau:

struct hoc\_sinh {

int sbd;

char ht[30];

char ngsinh[9]; // “15/09/92”

bool nam\_nu;

double toan, ly, ngoaingu;

bool do\_truot;

}x,y, lop[60];

Hoặc:

struct hoc\_sinh {

int sbd;

char ht[30];

char ngsinh[9]; // “15/09/92”

bool nam\_nu;

double toan, ly, ngoaingu;

bool do\_truot;

};

hoc\_sinh x, y, lop[60];

* **Truy cập các phần tử của kiểu dữ liệu struct**

Để truy cập tới dữ liệu của một trường trong cấu trúc, ta viết tên cấu trúc và tên trường nối nhau bởi dấu chấm.

Sau khi khai báo kiểu hoc\_sinh và khai báo biến x có kiểu hoc\_sinh, ta có thể truy cập x.ht là họ và tên của học sinh x và x.toan là điểm toán của học sinh x,...

*Ví dụ*. Một hoá đơn bán hàng có 3 thông tin là: tên hàng, số lượng, đơn giá. Hãy nhập 3 hoá đơn bán hàng. Viết chương trình hiện nội dung từng hoá đơn, số thành tiền của mỗi hoá đơn và tổng tiền hàng bán được ở 3 hoá đơn này.

#include <iostream>

#include <iomanip>

using namespace std;

const int n = 3;

struct hoa\_don {

char tenhang[30];

long soluong;

double dongia;

} hd, taphd[n];

int main() {

for (int i=0; i!=n; i++) {

cout << "\nNhap hoa don " << (i+1) ;

cout << "\nTen hang : ";

char s[30];

cin.getline(s,30); strcpy(taphd[i].tenhang,s);

cout << "So luong : ";

cin.getline(s,30); taphd[i].soluong = atol(s);

cout << "Don gia : ";

cin.getline(s,30); taphd[i].dongia = atof(s);

}

double thanhtien, tong=0;

for (int i=0; i!=n; i++){

hd = taphd[i];

thanhtien = hd.soluong\*hd.dongia;

cout << setw(30) << hd.tenhang << " : "

<< setw(10) << hd.soluong << " x "

<< setw(10) << hd.dongia << " = "

<< setw(15)<< setprecision(10)<<thanhtien<<endl;

tong += thanhtien;

}

cout << "\nTong tien cua cac hoa don la : "

<< setw(15)<< setprecision(10) << tong << endl;

system("pause");

return 0;

}

*Chú ý.* Để đưa kết quả hiện trên màn hình với khuôn dạng nhất định, C++ sử dụng nhiều bộ thao tác định khuôn dạng trong thư viện <iomanip>, hoặc sử dụng các thuộc tính của luồng chuẩn cout. Ví dụ:

- Định dạng setw(n) cho kết quả xuất ra trên khoảng rộng n kí tự.

- Định dạng setfill(c) sẽ lấp kí tự *c* vào phần đầu của khoảng rộng dành cho kết quả mà kết quả không dùng hết (kết quả căn theo lề phải của khoảng rộng).

- Định dạng setprecision(n) cho số lượng các chữ số có nghĩa là n trong kết quả số thực..

- Thuộc tính cout.precision(n) cho số lượng các chữ số có nghĩa là n trong kết quả số thực.

* **Con trỏ kiểu cấu trúc**

*Ví dụ* 2. Sử dụng con trỏ kiểu cấu trúc giải lại bài ví dụ 1.

#include <iostream>

#include <iomanip>

using namespace std;

const int n = 3;

struct hoa\_don {

char tenhang[30];

long soluong;

double dongia;

} taphd[n];

hoa\_don \* p; *//Khai báo con trỏ p, trỏ tới kiểu cấu trúc hoa\_don*

int main(){

for (int i=0; i!=n; i++) {

*//Các câu lệnh như trong Ví dụ 1*

}

double thanhtien, tong=0;

for (int i=0; i!=n; i++) {

p = new hoa\_don; *//Xin cấp phát bộ nhớ động cho con trỏ p*

\*p = taphd[i]; *// Tham chiếu tới dữ liệu của hóa đơn thứ i*

thanhtien = (p->soluong)\*(p->dongia);

cout << setw(30) << (p->tenhang) << " : "

<< setw(10) << (p->soluong) << " \*"

<< setw(10) << (p->dongia) << " = "

<< setw(15)<<setprecision(10)<<thanhtien<<'\n' ;

tong += thanhtien;

}

cout << "\nTong tien cua cac hoa don la : "

<< setw(15) << setprecision(10) << tong<<'\n';

system("pause");

return 0;

}

Trong vòng *for* thứ hai, các câu lệnh:

p = new hoa\_don;

\*p = taphd[i];

nhằm xin cấp phát bộ nhớ động cho con trỏ *p* và đặt *p* tham chiếu tới (trỏ tới) taphd[i]. Sau đó có thể truy cập tới giá trị các trường của *p* bằng cách sử dụng toán tử ->

Ví dụ: p->soluong chính là taphd[i].soluong, cũng là (\*p).soluong

Ghi chú: Toán tử -> gồm dấu trừ gõ liền với dấu lớn hơn dùng để tham chiếu tới các trường của biến con trỏ có kiểu **struct** (cấu trúc).

## Hàm

Một bài toán cho ban đầu có thể phân chia thành nhiều nhiệm vụ nhỏ riêng rẽ. Những đoạn chương trình hoàn thành từng nhiệm vụ nhỏ đó có thể viết thành một chương trình con. Những chương trình con này hợp thành chương trình giải bài toán ban đầu. Trong một số ngôn ngữ lập trình, chương trình con có thể là hàm hoặc thủ tục. Nhưng trong C++, mọi chương trình con đều gọi là hàm. Mỗi hàm được đặt tên theo đúng qui định về tên. Hàm có thể nhận các giá trị tham số đưa vào để thực hiện nhiệm vụ của mình và trả lại cho chương trình gọi hàm một giá trị thông qua tên hàm.

Điểm đặc biệt trong C++, hàm có thể trả lại một giá trị kiểu *void.* Trong trường hợp này ta ngầm định coi như là hàm không cần trả lại giá trị nào (khi đó coi như hàm chỉ làm nhiệm vụ một thủ tục).

Dạng thức **định nghĩa hàm** như sau:

***type name* ( *type1 argument1*, *type2*  *argument2*, ...) {**

***statements;***

***return value;***

***}***

trong đó:

***type*** là kiểu dữ liệu được trả về của hàm;

***name*** là tên gọi của hàm;

***argument1, argument2, …***  là tên các tham số hình thức (có nhiều bao nhiêu cũng được tuỳ theo nhu cầu). Còn ***type1, type2, …*** tương ứnglà các kiểu của các tham số ***argument1, argument2,…*** Các tham số hình thức khác nhau được ngăn cách bởi các *dấu phẩy*.

Dòng:

***type name* ( *type1 argument1*, *type2*  *argument2*, ...)** được gọi là phần đầu của hàm (chú ý cuối dòng không có dấu chấm phảy)

***statements*** là các câu lệnh taọ thành thân của hàm. Nó có thể là một lệnh đơn hay một khối lệnh thực hiện trong hàm.

***Lệnh return value;*** để hàm trả lại giá trị là *value* cho chương trình đã gọi hàm.

*Ví dụ 1*. Viết chương trình có xây dựng hàm *add* thực hiện phép cộng hai số

#include <iostream>

#include <conio.h>

using namespace std;

**int add (int a, int b) {**

**return (a+b);**

**}**

int main () {

int x,y,z;

cout << "nhap 2 so : "; cin >> x >> y ;

z = add (x,y);

cout << "Tong hai so la: " << z;

getch();

return 0;

}

Trong chương trình có hai hàm là **add** và **main.** Hàm **add** thực hiện nhiệm vụ: nhận vào 2 giá trị a và b, trả lại một giá trị là tổng a+b.

Hàm **main** là hàm chính. Chương trình được bắt đầu thực hiện từ câu lệnh đầu tiên của hàm **main.** Kiểu hàm chính luôn là **int,** giá trị trả lại là 0.

Tuy nhiên, có thể viết lại chương trình trên bằng sử dụng kiểu định nghĩa đoạn chương trình, được chương trình sau hiệu quả hơn:

#include <iostream>

#define add(a,b) (a+b)

using namespace std;

int main () {

int x,y;

cout << "nhap 2 so : ";

cin >> x >> y ;

cout << "Tong hai so la: " << add(x,y)<<endl;

system("pause"); *// Dừng màn hình kết quả đến khi có phím được nhấn*

return 0;

}

*Ví dụ 2.*Viết chương trình tìm số lớn nhất trong 3 số nguyên.

#include <iostream>

#include <conio.h>

using namespace std;

int a, b, c;

int max(int a, int b) {

if (a>b) return a;

else return b;

}

int max(int a,int b, int c) {

return max(max(a,b),c);

}

int main() {

cout << "Nhap ba so a, b, c: "; cin>>a>>b>>c;

cout << "Max(" << a << "," << b <<") la:"

<< max(a,b) << endl;

cout << "Max(" << a << "," << b << "," << c << ") la:"

<<max(a,b,c);

getch();

return 0;

}

Trong chương trình trên có hai hàm đều có tên là *max* trong đó hàm thứ nhất nhận 2 tham số và hàm thứ hai nhận 3 tham số. C/C++ có khả năng nhận biết sự khác biệt giữa hai hàm trên vì trong lời gọi hàm người dùng đã cho biết danh sách các tham số (gồm các biến nào và chúng thuộc kiểu nào). Tính chất này gọi là tính quá tải của hàm.

Có thể thay chương trình trên bằng chương trình hiệu quả hơn có sử dụng kiểu định nghĩa đoạn chương trình:

#include <iostream>

#define max2(x,y) (x>y?x:y)

#define max3(x,y,z) (max2(max2(x,y),z))

using namespace std;

int a, b, c;

int main() {

cout << "Nhap ba so a, b, c: ";

cin >> a >> b >> c;

cout << max3(a,b,c)<< endl;

system("pause");

return 0;

}

*Ví dụ 3.*Viết chương trình tính số cách chọn k phần tử từ tập có n phần tử theo công thức*:*

#include <iostream>

#include <conio.h>

using namespace std;

int n, k;

int giaithua(int m) {

int gt=1;

for (int i = 2; i <= m; i++) gt \*= i;

return gt;

}

int main() {

cout << "Nhap k và n: "; cin>>k>>n;

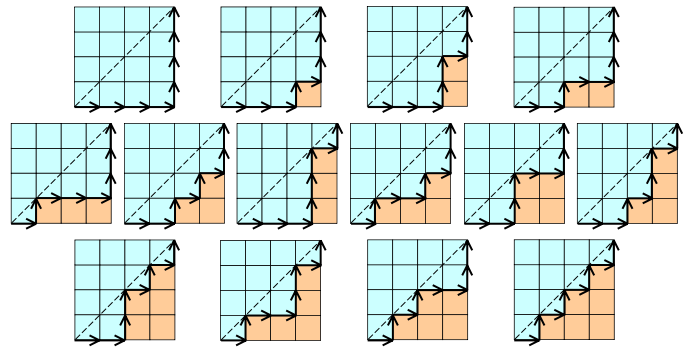
cout << giaithua(n)/(giaithua(n-k)\*giaithua(k));

getch();

}

*Ví dụ 4.* Tìm số Catalan thứ n

Dãy số Catalan là dãy T0=1, T1=1, T2=2, T3=5, T4=14, T5=42, … trong đó số thứ n là bằng số đường đi khác nhau dọc theo các cạnh ô vuông của lưới n×n ô vuông mà đường đi chỉ ở dưới một đường chéo (hình vẽ minh họa T4). Hạn chế n<30



#include <iostream>

#include <conio.h>

#include <iomanip>

using namespace std;

int n;

long double catalan(int n) {

long double t=1.0;

for (int i=n+2; i<=2\*n; i++) t = (t\*i)/(i-n-1);

t = t/n;

return t;

}

int main() {

cout << "Nhap n: "; cin>>n;

cout << fixed << setprecision(6);

cout << catalan(n);

getch();

return 0;

}

## Mảng

Trong khuôn khổ cuốn sách này, chúng ta đề cập nhiều về hai kiểu dữ liệu thông dụng nhất là mảng (array) và xâu ký tự (string).

### Mảng một chiều

Mảng là dãy hữu hạn các phần tử cùng kiểu. Các phần tử được đánh chỉ số và có thể truy cập trực tiếp. Các phần tử được lưu trên một vùng liên tục trong bộ nhớ và địa chỉ tăng dần theo thứ tự tăng dần của chỉ số.

* Dạng thức khai báo mảng là:

***type name* [*size*];**

Trong đó ***type***  là kiểu phần tử của mảng, ***name*** là tên mảng, ***size*** là số phần tử của mảng còn gọi là kích thước của mảng. Các phần tử của mảng được đánh số từ 0. Mỗi phần tử của mảng được xác định bởi tên mảng và chỉ số của phần tử đó.

*Ví dụ*. Khai báo mảng a có 6 phần tử là a[0], a[1], a[2], a[3], a[4], a[5]cùng kiểu int như sau:

int a[6];

* Khởi tạo mảng.

Theo mặc định, nếu mảng khai báo ngoài mọi hàm thì các phần tử được tự động khởi trị là 0. Nhưng *khi mảng khai báo trong một hàm thì mảng không được C++ khởi trị sẵn, chúng ta cần khởi trị mảng*.

Thường khởi trị cùng với khai báo mảng bằng cách gán giá trị cho các phần tử của nó như sau:

int a[6] = {10, 4, 5, 2, 1, 0};

hoặc

int a[ ] = {10, 4, 5, 2, 1, 0};

Trong cách thứ hai, C++ không khai báo trực tiếp số phần tử của mảng, kích thước của mảng chính bằng số lượng các giá trị đã cho trong cặp ngoặc { }

Trong một số trường hợp cần tạo bộ test ngẫu nhiên thường khởi tạo mảng bằng các giá trị ngẫu nhiên sinh ra từ hàm rand() (xem phụ lục 1).

* Truy xuất các phần tử của mảng

Trong phạm vi khai báo của mảng, có thể truy xuất tới từng phần tử của mảng qua kí hiệu ***name[index]*** trong đó ***name*** là tên mảng, ***index*** là chỉ số của phần tử.

*Ví dụ*

a[2] = 13; //gán giá trị 13 cho phần tử thứ ba của mảng a

x = a[0] //gán giá trị của phần tử thứ nhất của mảng a cho biến x

cout << a[2]; // xuất giá trị của phần tử thứ ba của mảng a lên màn hình

*Lưu ý.* Trong C++, trình dịch không phát hiện lỗi truy cập quá giới hạn của mảng. *Ví dụ* chương trình sau đây trình dịch không báo lỗi:

#include <iostream>

#include <conio.h>

int a[6];

int main()

{

a[8] = 10;

cout << a[8];

getch();

return 0;

}

Điều này gây rắc rối và sai lầm khi thực hiện chương trình tạo ra kết quả không mong muốn (vì tại ô nhớ a[8] có thể chứa dữ liệu từ lệnh cũ làm ảnh hưởng xấu tới thực hiện chương trình). Vậy chúng ta cần lưu ý khi chọn kích thước của mảng và chỉ truy cập phần tử có chỉ số không quá kích thước.

*Ví dụ.* Cho mảng gồm 8 phần tử là các số nguyên 4, 12, 3, 6, 10, 7, 8, 5

Tính giá trị trung bình của phần tử và số lượng phần tử lớn hơn giá trị trung bình.

#include <iostream>

#include <numeric>

using namespace std;

int main() {

const int N = 8;

int dem=0;

int a[N] = {4, 12, 3, 6, 10, 7, 8, 5}, x = 0;

int sum = accumulate(a, a+N, x);

float trungbinh=sum/N;

for (int i = 0; i != N; i++)

if (a[i] > trungbinh) dem++;

cout << "Tong: " <<sum << endl;

cout << "Gia tri trung binh: " << trungbinh << endl;

cout << "So phan tu tren trung binh: " << dem << endl;

system("pause");

return 0;

}

Trong chương trình trên có một số điểm cần lưu ý:

+ Tên mảng được coi là con trỏ chỉ tới phần tử 0 của mảng

+ Hàm accumulate(a, a+N,x)góp nhặt các phần tử của mảng a từ vị trí 0 đến vị trí N cộng tích lũy lại thành một tổng, rồi cộng với số x. Cũng có thể thay dòng lệnh: int sum=accumulate(a, a+N,x);

bằng đoạn sau:

int sum=0;

for (int i=0; i != N; i++) sum +=a[i];

Ví dụ. Nhập không quá 10 số nguyên dương vào đầu của mảng a (có thể từ a[0] đến a[9]), kết thúc nhập bởi số 0; đồng thời nhập số nguyên x. Tìm xem x có trong mảng a hay không, nếu có thì nêu rõ chỉ số của phần tử bằng x trong mảng a.

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int a[10], x, n = 0;

cout << "Nhap 10 so nguyen duong,ket thuc la so 0\n";

while (cin >> x, x != 0 && n < 10) a[n++] = x;

cout << "Gia tri can tim là: "; cin >> x;

int \*p = find(a, a+n, x);

if (p == a+n)

cout << "Khong tim thay\n";

else {

cout << "Tim thay";

if (p== a)

cout << " la phan tu dau tien";

else cout << " la phan tu thu " << p-a+1;

}

cout << endl;

system("pause");

return 0;

}

*Giải thích.* Trong thư viện algorithm có hàm **find(a,a+n,x)** cho phép tìm số x trong n phần tử đầu tiên của mảng a. Giá trị trả về của hàm là con trỏ mảng p chỉ tới vị trí phần tử của a bằng x (nếu có) hoặc trả về con trỏ mảng p chỉ tới vị trí sau phần tử cuối cùng của a (nghĩa là không có x trong a).

* Gán giá trị cho mảng

Cho hai mảng cùng kiểu a và b , có thể gán giá trị của mảng nguồn a cho mảng đích b bằng câu lệnh sau:

**memmove(b, a, sizeof(a));**

### Mảng hai chiều

Mảng hai chiều được coi như mảng một chiều có các phần tử là các mảng một chiều cùng kiểu và cùng kích thước. Mảng hai chiều còn được coi như một mô hình tương ứng với một bảng hai chiều gồm các hàng và các cột.

* Khai báo mảng hai chiều

***type* *name*[n1][n2];**

Trong khai báo nêu trên, type là kiểu của các phần tử của mảng, name là tên của mảng, n1 là kích thước chiều thứ nhất, n2 là kích thước chiều thứ hai của mảng.

*Ví dụ*: Khai báo mảng số nguyên có hai chiều: chiều thứ nhất kích thước là 3, chiều thứ hai kích thước là 5 :

int a[3][5];

Mảng này tương ứng với một bảng hai chiều có 3 dòng và 5 cột. Lưu ý chỉ số của dòng và của cột đều đánh số bắt đầu từ 0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Cột 0** | **Cột 1** | **Cột 2** | **Cột 3** | **Cột 4** |
| **Dòng 0** | 11 | 7 | 25 | 6 | 11 |
| **Dòng 1** | 45 | 68 | 79 | 35 | 5 |
| **Dòng 2** | 6 | 4 | 14 | 21 | 71 |

* Truy xuất tới phần tử

Có thể truy xuất tới một phần tử qua tên mảng cùng chỉ số dòng và chỉ số cột. Trong ví dụ nêu trên, a[2,4] có giá trị là 71, a[0][0] có giá trị là 11, a[1][3] có giá trị là 35.

* Chuyển mảng hai chiều thành mảng một chiều

Mảng hai chiều a[n1][n2] được coi tương đương với mảng một chiều b[n1\*n2]. Khi đó a[i][j] ứng với b[i\*n2 + j]. *Ví dụ* mảng a[3][5] được coi tương đương với mảng b[15], phần tử a[1][3] tương đương với phần tử b[8] (vì 1\*5+ 3=8).

### Mảng ba chiều

Trong một số bài toán chúng ta còn có thể gặp mảng nhiều hơn 2 chiều. Khi đó khai báo cũng tương tự. *Ví dụ* khai báo mảng 3 chiều như sau:

***type* *name*[n1][n2][n3];**

Trong khai báo, các số n1, n2, n3 lần lượt là kích thước chiều thứ nhất, thứ hai, thứ ba. Số phần tử của mảng sẽ là n1\*n2\*n3, dễ trở thành số lớn.

### Vector - Mảng trong thư viện chuẩn (std)

Điểm nổi bật của C++ là trong thư viện chuẩn STL (Standard Template Library) có nhiều lớp mẫu và hàm mẫu được xây dựng sẵn, người lập trình có thể sử dụng. Các lớp mẫu thông dụng nhất là: vectơ, stack, queue.

Về một phương diện nào đó, có thể dùng vectơ thay cho mảng với số phần tử hữu hạn nhưng tùy ý không cần khai báo ban đầu. Một đối tượng thuộc kiểu vectơ sẽ được sử dụng các hàm xây dựng cho vectơ.

* **Khai báo sử dụng thư viện vectơ:**

#include <vector>

* **Khai báo một đối tượng có kiểu vectơ:**

vector<*kiểu\_của\_phần\_tử* > *tên\_đối\_tượng*;

Ví dụ: Khai báo đối tượng kiểu vectơ gồm các phần tử số nguyên, tên là v (gọi vắn tắt là vectơ v) như sau:

vector<*int*> v;

* **Khởi trị vectơ:**

vector<*int*> v=(n,value);

Lệnh trên vừa khai báo vectơ *v* có *n* phần tử vừa khởi trị các phần tử có giá trị đều bằng *value*

* **Truy xuất các phần tử của vectơ** cũng thông qua chỉ số của phần tử, chẳng hạn v[i] cho giá trị của phần tử thứ i+1 của vectơ v. Ngoài ra có thể dùng hàm **at** để truy xuất nhanh hơn, ví dụ: **v.at(i)** cho giá trị phần tử thứ i+1 của vectơ v.
* **Nạp giá trị x vào vị trí cuối cùng của vectơ v:**

v.push\_back(x);

*Chú ý*. x phải cùng kiểu với kiểu phần tử của vectơ v.

Khi sử dụng hàm trong lớp vectơ, viết giữa tên vectơ và tên hàm là một dấu chấm.

* **reserve(*số\_ phần\_ tử)****.*

Để khai báo vectơ a cần bộ nhớ có n phần tử thì viết:

a.reserve(n);

* **resize(*số\_phần\_tử, giá\_trị\_phần\_tử).***

Để khai báo lại kích thước vectơ a chỉ còn *n* phần tử đều có giá trị bằng *value* thì viết:

a.resize(n,value);

*Ví dụ.* Nhập 5 phần tử của một vectơ kiểu int. Hiện phần tử thứ ba của vectơ:

#include <iostream>

#include <vector>

using namespace std;

vector<int> v;

int x, n;

int main(void) {

n = 5;

v.reserve(n); *// Khai báo vectơ v có n phần tử*

cout << " Nhap 5 phan tu cua vecto : " << endl;

for (int i=0; i!=5; i++) {

cout << "Nhap phan tu thu " << i+1 << " : ";

cin >> x;

v.push\_back(x); *// Nạp x làm phần tử cuối của vectơ*

}

cout << "phan tu thu ba la: " << v.at(2) << endl;

system("pause");

return 0;

}

* **assign(num, val)** gán giá trị *val* cho *num* phần tử đầu của vector và chỉ có các phần tử này được quản lí.

Ví dụ đã tồn tại vectơ v=(10,2,3,5,1,8) thì v.assign(4,2) sẽ tạo lại vector v chỉ gồm 4 phần tử cùng có giá trị là 2: v=(2,2,2,2).

* ***back , front***

Hàm back(). Để trả lại giá trị phần tử cuối cùng của vector *v* thì viết v.back();

Hàm front(). Để trả lại giá trị phần tử đầu của vector *v* thì viết v.front();

* ***Khai báo con trỏ lặp iterator***

Đó là con trỏ chỉ tới từng phần tử đang được quản lý của một vectơ. Khai báo một con trỏ lặp iterator như sau:

*vector<kiểu phần tử> :: iterator tên\_con\_trỏ;*

* ***begin, end***

Hàm begin() trả lại một con trỏ chỉ tới phần tử đầu của vector, con trỏ này thuộc loại *iterator* (gọi là đối tượng lặp).

Hàm end() trả lại một con trỏ chỉ tới phần tử cuối của vector, cũng thuộc loại *iterator* .

Ví dụ:

vector<int> v1( 5, 789 );*// v1 có 5 phần tử bằng 789*

vector<int>::iterator it;

for( it = v1.begin(); it != v1.end(); it++ )

cout << \*it << endl; *//Giá trị các phần tử của vector do* **it** *trỏ tới*

* ***size***

Hàm size() cho số lượng phần tử đang được quản lý của vector hiện tại.

* ***pop\_back***

Hàm pop\_back() xoá đi phần tử cuối cùng của vector (nghĩa là không quản lý phần tử cuối cùng này nữa).

Ví dụ: Tạo vectơ s chứa 10 kí tự liên tiếp A, B, C, D, E, F, G, H, I (có mã số ASCII từ 65 đến 74).

Xoá dần các kí tự cuối của *s* rồi hiện *s* sau mỗi lần xoá.

#include <iostream>

#include <vector>

using namespace std;

vector<char> s;

vector<char>::iterator p;

int main() {

for( int i=0; i < 10; i++ ) s.push\_back(i+65);

int size = s.size();

for( int i=0; i != size; i++ ) {

s.pop\_back();

for( p = s.begin(); p!= s.end(); p++ )

cout << \*p;

cout << endl;

}

system("pause");

return 0;

}

Kết quả chương trình trên xuất ra màn hình sẽ là:

ABCDEFGHI

ABCDEFGH

ABCDEFG

ABCDEF

ABCDE

ABCD

ABC

AB

A

* ***erase***

erase( iterator pos );

erase( iterator start, iterator end );

Cách 1: hàm erase() xoá đi phần tử mà poschỉ tới.

Cách 2: hàm erase() xoá đi các phần tử từ *start* đến *end*.

Hàm trả về đối tượng lặp chỉ vào phần tử sau phần tử cuối cùng bị xoá.

Ví dụ. Tạo vector s=(‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’, ‘I’, ‘J’), Dùng hàm erase xoá dần các kí tự đầu. Hiện vector *s* sau mỗi lần xoá:

#include <iostream>

#include <vector>

using namespace std;

vector<char> s;

vector<char>::iterator p;

int main() {

for( int i=0; i < 10; i++ ) s.push\_back(i+65);

int size = s.size();

for( int i=0; i != size; i++ ) {

p = s.begin(); *// p chỉ vào phần tử đầu*

s.erase( p ); *//xóa phần tử đầu*

*// Hiện các phần tử còn lại*

for(p=s.begin(); p!=s.end(); p++) cout << \*p ;

cout<<endl;

}

system("pause");

return 0;

}

Kết quả trên màn hình sẽ là:

BCDEFGHIJ

CDEFGHIJ

DEFGHIJ

EFGHIJ

FGHIJ

GHIJ

HIJ

IJ

J

* ***insert***

Kiểu 1: Thêm phần tử có giá trị bằng *val* vào trước phần tử được chỉ bởi *pos*

insert( iterator pos, val );

Kiểu 2: Chèn thêm *num* phần tử vào trước phần tử được chỉ bởi *pos*

insert( iterator pos, num, val );

Kiểu 3: Chèn các phần tử từ vị trí lặp *t1* đến *t2* của vectơ s2 vào trước phần tử chỉ bởi vị trí *pos* của vec tơ s1

insert(iterator *pos*,iterator *t1*,iterator *t2*);

Ví dụ.

#include <iostream>

#include <vector>

using namespace std;

vector<char> s, s2;

vector<char>::iterator p, t1, t2;

int main() {

*// Tạo vectơ s= (ABCDEFGHIJ)*

for(char i=65; i < 75; i++ ) s.push\_back(i);

*//Hiện vectơ s*

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p;

cout<<endl;

*//Tạo vec tơ s2=(abcdefghij) và hiện s2*

for(char i=97; i < 107; i++ ) s2.push\_back(i);

for (p=s2.begin(); p!=s2.end(); p++)

cout<< \*p;

cout<<endl;

*//****Kiểu chèn thứ nhất***

p=s.begin();*//p trỏ vào phần tử đầu của s*

s.insert(p,90); *//Chèn kí tự Z vào đầu s*

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p; *//Hiện lại s sau khi chèn: s=(ZABCDEFGHIJ)*

cout<<endl;

*//Xóa Z ở đầu vec tơ s, hiện lại s như cũ s=(ABCDEFGHIJ)*

p=s.begin();

s.erase(p);

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p;

cout<<endl;

*//****Kiểu chèn thứ hai***

p=s.begin();

s.insert(p,10,90); *// Chèn 10 kí tự Z (mã là 90) vào trước vị trí p*

*//Hiện s mới =(ZZZZZZZZZZABCDEFGHIJ)*

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p;

cout<<endl;

*//Xóa 10 kí tự Z*

for (int i=1; i!=10; i++) {

p=s.begin();

s.erase(p);

}

*//Hiện lại s như cũ s=(ABCDEFGHIJ)*

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p;

cout<<endl;

*//****Kiểu chèn thứ 3***

p=s.begin()+2; *//p trỏ vào phần tử thứ ba của s*

t1=s2.begin()+5; *//t1 trỏ vào phần tử thứ 6 của s2*

t2=t1+3; *//t2 trỏ vào phần tử thứ 8 của s2*

s.insert(p,t1,t2); *//chèn các phần tử s2[5]...s2[7] vào s (trước s[2])*

*//Hiện s mới: s=ZAfghBCDEFGHIJ)*

for (p=s.begin(); p!=s.end(); p++)

cout<< \*p;

cout<<endl;

system("pause");

return 0;

}

* ***empty***

Hàm empty() trả lại giá trị true nếu vector không có phần tử nào, ngược lại là false. Ví dụ: Vòng lặp xuất lần lượt các phần tử của vectơ từ cuối về đầu:

while( !v.empty() ) {

cout << v.back() << endl;

v.pop\_back();

}

* ***clear***

Để xóa toàn bộ nội dung của vectơ *v* viết:

v.clear();

* ***sort***

Sắp tăng các phần tử của vector *v*:

sort( v.begin(),v.end() );

Sắp giảm các phần tử của vector *v*:

sort(v.begin(),v.end(), greater<*kiểu phần tử của v*>() );

#include <iostream>

#include <vector>

using namespace std;

vector<int> v;

int main() {

v.reserve(10);

v.push\_back(2); v.push\_back(-4); v.push\_back(22);

v.push\_back(28); v.push\_back(12); v.push\_back(15);

v.push\_back(7); v.push\_back(9); v.push\_back(3);

v.push\_back(25);

sort(v.begin(), v.end());

for (int i=0; i!=10; i++) cout << v[i] << " ";

cout << endl;

system("pause");

return 0;

}

Còn có thể dùng hàm sort để sắp mảng một chiều. Ví dụ:

int a[] = { 23, -1, 9999, 0, 4 };  
int a\_size = 5;             
cout << "Before sorting: ";  
for( int i = 0; i < a\_size; i++ )  
   cout << a[i] << " ";  
cout << endl;              
sort( a, a + a\_size );*// Sắp tăng mảng a*               
cout << "After sorting: ";  
for( int i = 0; i < a\_size; i++ )  
cout << a[i] << " ";  
cout << endl;

Bây giờ, sử dụng vectơ chúng ta giải bài toán:

“*Một hoá đơn bán hàng có 3 thông tin là: tên hàng, số lượng, đơn giá. Một cửa hàng thống kê tình hình bán hàng của tháng Tết. Viết chương trình nhập các hoá đơn bán hàng trong tháng này (việc nhập kết thúc khi nhập tên hàng là “ketthuc”) và tính tổng tiền bán được của cửa hàng trong tháng Tết đó đồng thời cho biết tên hàng có số tiền bán được lớn nhất*”.

#include <iostream>

#include <iomanip>

#include <string>

#include <vector>

using namespace std;

int n=0;

struct hoa\_don {

char tenhang[30];

int soluong;

double dongia;

} e;

vector<hoa\_don> taphd;*//Khởi tạo tập hóa đơn* taphd *rỗng*

int main(void) {

int ii; *//biến dùng lưu lại chỉ số hóa đơn có thành tiền lớn nhất*

float tien, maxtien=0;

bool finish = false; *//Cờ báo hiệu chưa kết thúc nhập*

*//Nhập các hóa đơn e*

while (!finish) {

cout << "\nNhap hoa don " << (++n) ;

cout << "\nTen hang : ";

char s[30];

cin.getline(s,30); strcpy(e.tenhang,s);

if (strcmp(e.tenhang,"ketthuc")==0) finish = true;

else { *// Tên hàng khác “ketthuc” thì nhập tiếp soluong, dongia*

cout << "So luong : ";

cin.getline(s,30); e.soluong = atol(s);

cout << "Don gia : ";

cin.getline(s,30); e.dongia = atof(s);

taphd.push\_back(e); *// Nạp hoa đơn e vào cuối tập hóa đơn*

}

}

*// Hiện lại các hóa đơn đã nhập thành cột*

for (int i = 0; i != (int) taphd.size(); ++i) {

tien = taphd[i].soluong \* taphd[i].dongia;

cout << setw(30) << (taphd[i].tenhang) << " : "

<< setw(10) << (taphd[i].soluong) << " \*"

<< setw(10) << setprecision(10)

<< (taphd[i].dongia) << " = "

<< setw(15)<<setprecision(10)<<tien<<'\n' ;

if (tien>maxtien) {

maxtien=tien;

ii=i; *//Lưu chỉ số hóa đơn có thành tiền max*

}

}

*// Tính tổng doanh thu trong tháng*

double tongtien = 0;

for (int i = 0; i != (int) taphd.size(); ++i)

tongtien += taphd[i].soluong \* taphd[i].dongia;

*//Hiện các yêu cầu cần thống kê : tổng thành tiền, tên hàng có thành tiền max*

cout << "----------THONG KE:-------------- " << endl;

cout << "tong tien = " << setprecision(10)

<< tongtien << "\n";

cout << "Ten hang co thanh tien max: "

<< taphd[ii].tenhang << endl;

system("pause");

return 0;

}

## Kí tự và xâu kí tự

### Kí tự

Mọi văn bản đều gồm các ký tự (các kí hiệu, chữ cái, chữ số). Ký tự là dữ liệu kiểu *char*. Một biến kiểu *char* có giá trị từ 0 đến 255 (kiểu không dấu: *unsigned char*) hoặc là có giá trị từ -128 đến 127 (kiểu có dấu: *signed char*) và mỗi giá trị ứng với một kí tự trong văn bản.

Kiểu *char* thuộc kiểu có dấu hay không dấu là phụ thuộc hệ điều hành và trình dịch. Vì thế khi sử dụng kiểu ký tự chúng ta nên khai báo rõ nó thuộc kiểu *unsigned char* hay *signed char*.

*Ví dụ 1.*

#include<iostream>  
using namespace std;  
int main(){  
unsigned char c='A';  
cout<<c<<" "<<(int)c<<endl;*// In ra A và 65*c=68;  
cout<<c<<" "<<(int)c<<endl;*// In ra D và 68*return 0;  
}

Thực hiện chương trình trên, kết quả trên màn hình sẽ là:

A 65

D 68

Chú ý rằng c=68, khi in ra màn hình là D vì khai báo unsigned char c='A';.

*Ví dụ 2.* Chương trình sau hiện các mã ASCII của các ký tự được gõ từ bàn phím, chương trình dừng khi ký tự gõ là ‘q’.

#include<iostream>

using namespace std;

int main() {

unsigned char c;

while(cin>>c) {

cout<<c<<' '<<(int)c<<endl;

if (c=='q') break;

}

return 0;

}

### Xâu kí tự (chuỗi).

Các chương trình C++ có thể sử dụng chuỗi kí tự theo cách thức của *ngôn ngữ C* đó là coi chuỗi kí tự như một mảng gồm các phần tử kiểu ký tự và được kết thúc bởi ký tự null (‘\0’) có mã ASCII là 0; hoặc sử dụng các hàm khai báo trong thư viện *string*

***Định nghĩa***. Xâu kí tự (chuỗi kí tự) là dãy các kí tự được lưu trữ tại các byte liên tiếp trong bộ nhớ và có kí tự cuối cùng là kí tự null (‘\0’).

*Độ dài của xâu là số kí tự trong xâu, không kể kí tự cuối cùng là null*

***Khai báo xâu kí tự.*** Xâu kí tự **s** có thể khai báo theo hai cách sau:

char s[100]; *// Khai báo như một mảng có các phần tử là các kí tự*

char \* s; *// Khai báo như một biến con trỏ chỉ tới địa chỉ phần tử đầu của mảng*

***Cách truy cập các kí tự của xâu.*** Cho xâu **s** có độ dài N, gồm N kí tự. Các phần tử của nó được truy cập thông qua các kí hiệu s[0], s[1], …, s[i],…, s[N-1]. Xâu s được kết thúc bởi kí tự s[N]=’\0’ vậy cần mảng s có N+1 phần tử để lưu trữ xâu này..

***Cách khởi trị xâu kí tự.*** Khi khai báo xâu kí tự, chúng ta cũng có thể khởi trị xâu kí tự theo những cách sau:

char s[ ] = “Thanh”; *//Khởi trị cả mảng bằng hằng mảng kí tự*

char s[ ] = {‘T’, ‘h’, ‘a’, ‘n’, ‘h’,’\0’};*//Khởi trị từng phần tử*

char s[6] = {‘T’, ‘h’, ‘a’, ‘n’, ‘h’,’\0’};*//Khởi trị từng phần tử*

Mặt khác, do qui định *hằng xâu kí tự là một con trỏ kiểu kí tự chỉ vào địa chỉ kí tự đầu tiên của hằng*, nên còn có thể khởi trị một con trỏ kiểu kí tự bằng một hằng xâu kí tự như cách sau:

char \* s = “Thanh”;

***Nhập xâu kí tự từ bàn phím***

Để nhận từ bàn phím giá trị của một xâu kí tự, chúng ta có thể dùng phương thức cin.getline. Lời gọi phương thức có dạng sau:

cin.getline (char s[ ], int length, char delimiter = ‘\n’);

Trong đó s là mảng kí tự cần nhận dãy kí tự từ bàn phím, length là độ dài cực đại của mảng s, delimite là một kí tự kết thúc việc nhập, khi bỏ qua delimiter thường ngầm định là kí tự ‘\n’ (kí tự xuống dòng: nhấn phím Enter).

*Ví dụ 1.*

#include <iostream>

#include <conio.h>

using namespace std;

char s[100];

int main(){

cout << "Nhap mot xau chu so " ;

cin.getline(s,100);

cout << "Xau da nhap là" << s;

getch();

return 0;

}

*Ví dụ 2. Tạo một đối thoại ngắn giữa máy tính và người sử dụng*

#include <iostream.h>

#include <conio.h>

using namespace std;

int main (){

char s [100];

cout << "Ten ban la gi? ";

cin.getline (s,100);

cout << "Chao ban " << s << ".\n";

cout << "Ban thich doi bong nao? ";

cin.getline (s,100);

cout << "Toi cung thich doi bong " << s << "\n";

getch();

return 0;

}

*Chú ý*. Nếu sử dụng cin để nhập từ bàn phím một dãy kí tự vào xâu kí tự thì chỉ nhập được các kí tự cho đến khi gặp kí tự trống. *Ví dụ* dùng lệnh cin >> s; nhập dãy kí tự “*Tran van Thanh”* vào s thì s chỉ nhận “*Tran”*.

***Xuất xâu kí tự lên màn hình.***

Có thể xuất lần lượt từng kí tự của xâu kí tự. Chương trình sau cho kết quả trên 3 dòng, mỗi dòng một xâu “Thanh”**.**

*Ví dụ 3.*

#include <iostream>

#include <conio.h>

using namespace std;

char s[] = "Thanh";

char s2[] = {'T', 'h', 'a', 'n', 'h','\0'};

char \* p = "Thanh";

int main() {

for (int i=0; s[i]!='\0'; i++) cout << s[i];

cout << '\n';

for (int i=0; s2[i]!='\0'; i++) cout << s2[i];

cout << '\n';

for (int i=0; p[i]!='\0'; i++) cout << p[i];

cout << '\n';

getch();

return 0;

}

Luồng *cout* cho phép xuất ra toàn bộ một xâu kí tự nên có chương trình ngắn gọn hơn là:

#include <iostream>

#include <conio.h>

using namespace std;

char s[ ] = "Thanh";

char s2[ ] = {'T', 'h', 'a', 'n', 'h','\0'};

char \* p = "Thanh";

int main() {

cout << s << '\n';

cout << s2 << '\n';

cout << p << '\n';

getch();

return 0;

}

* Khi khai báo char \* p = "Thanh"; thì **p** là con trỏ chỉ tới **xâu kí tự** có địa chỉ đầu tiên là địa chỉ của kí tự ‘T’. Sử dụng con trỏ và luồng *cout*, ta có thể xuất ra các xâu: “hanh” , “anh”, “nh”, “h” bằng các câu lệnh tương ứng là:

cout << p+1;

cout << p+2;

cout << p+3;

cout << p+4;

* Chúng ta cũng cần phân biệt p+i khác với \*(p+i):

a) p+i là con trỏ chỉ tới xâu kí tự có kí tự đầu là kí tự thứ i+1 trong xâu p.

b) \*(p+i) là giá trị của kí tự thứ i+1 trong xâu p.

*Ví dụ* p+1 là “hanh” còn \*(p+1) là ‘h’.

***Cách gán giá trị cho xâu kí tự***

+ Khi khai báo xâu kí tự như biến con trỏ kí tự thì trong chương trình có thể gán xâu bằng một hằng xâu kí tự.

Khai báo: char \* p;

thì trong chương trình có thể dùng lệnh gán:

p = “Tran van Thanh”;

+ Khi khai báo xâu kí tự p như một mảng các kí tự thì lệnh gán trên là không hợp lệ (vì vế phải lệnh gán là hằng xâu kí tự nên là kiểu con trỏ kí tự, vế bên trái lệnh gán là biến có kiểu mảng). Trong trường hợp này phải gán từng kí tự cho các phần tử của mảng và gán cho phần tử cuối cùng của mảng là kí tự ‘\0’ (kí tự null).

Người ta còn thường dùng hàm **strcpy(s1, s2)** trong tệp tiêu đề string.h để sao chép các kí tự của hằng xâu s2 sang mảng kí tự s1.

*Ví dụ 4.*

#include <iostream>

#include <conio.h>

using namespace std;

char s[20];

int main() {

strcpy(s,"Tran van Thanh"); *// Gán s="Tran van Thanh";*

cout << s;

getch();

return 0;

}

***Một số hàm thao tác trên xâu kí tự***

C++ thừa kế một số hàm làm việc trên xâu kí tự của C. Sau đây là một số hàm thông dụng thao tác trên các xâu kí tự:

* Hàm **strcpy():** char \***strcpy**(char \*dest, const char \*src);Thường dùng để copy hằng xâu (nguồn) src sang mảng kí tự (xâu đích) dest.

*Ví dụ 5*

#include <iostream>

using namespace std;

int main(void){

char s[10];

char \*str1 = "Tran Thi Thu Ha";

strcpy(s, str1);

cout << s << endl;

system("pause");

return 0;

}

* Hàm **strcat():** char \***strcat**(char \*dest, const char \*src); Hàm này nối hằng xâu *src* vào cuối mảng kí tự *dest* và trả lại mảng *dest* đã nối.

*Ví dụ 6.*

#include <iostream>

using namespace std;

int main (){

char str[80];

strcpy (str,"these "); *//Chuyển hằng xâu vào mảng kí tự*

strcat (str,"strings ");*//Nối hằng xâu vào mảng kí tự*

strcat (str,"are ");

strcat (str,"concatenated.");

cout << str <<endl; //hoặc puts(str);

system("pause");

return 0;

}

Kết quả thực hiện chương trình là xâu sau hiện trên màn hình:

these strings are concattenated.

* Hàm **strcmp():** int  **strcmp**(const char \*s1, const char \*s2); Hàm so sánh hai xâu s1 và s2. Nếu s1<s2 thì kết quả là số âm, nếu s1==s2 thì kết quả là số 0, nếu s1>s2 thì kết quả là số dương.
* Hàm **strncpy(): char \*strncpy(char \*dest, const char \*src, size\_t maxlen);** Sao chép *maxlen* kí tự đầu của xâu nguồn *src* sang mảng đích *dest*. (*maxlen* không vượt quá chiều dài xâu nguồn và kích thước mảng đích).

*Ví dụ 7. Kết quả chương trình sau sẽ hiện “*Ha Noi*” trên màn hình?*

#include <iostream>

using namespace std;

int main(){

char \* src = "Ha Noi hoa binh";

char dest[100];

strncpy(dest, src, 6); dest[6]=’\0’;

cout << dest;

system("pause");

return 0;

}

* Hàm **strlen():** size\_t **strlen**(const char \*s); Hàm này trả lại độ dài của xâu. Chú ý kiểu hàm là size\_t đó là một số nguyên không dấu.
* Hàm **strchr:** const char \*strchr(const char \*s, int c); Trả về một con trỏ kí tự chỉ tới địa chỉ xâu con của xâu kí tự *s* (kể từ ký tự *c* cho đến hết)*.* Trả về NULL nếu *c* không có trong *s.*

Do đó có thể lấy địa chỉ xâu do hàm trả về trừ đi địa chỉ xâu s sẽ biết được *vị trí lần đầu tiên xuất hiện kí tự c trong xâu s*:

*Ví dụ 8*

#include <iostream>

using namespace std;

int main(void) {

char s[42];

char \* s1, c = 'T';

strcpy(s, "Hoang van Thu - Le Qui Don - Nguyen Trai");

s1 = strchr(s, c);

cout << s1 << '\n'; *// Hiện xâu kết quả do hàm strchr trả lại*

if (s1) *// Địa chỉ xâu s1 khác không*

cout << "Ki tu " <<c << " o vi tri : " << (s1-s);

else

cout << "Khong tim thay.";

system("pause");

return 0;

}

* Hàm **strstr():** const char \*strstr(const char \*s1, const char \*s2); hoặc char \*strstr(char \*s1, const char \*s2);

Hàm trả về xâu con của s1 kể từ vị trí xuất hiện đầu tiên của xâu s2 trong s1, kéo dài đến hết xâu s1. Nếu không tìm thấy, hàm trả về NULL.

*Ví dụ 9.* Chương trình dưới đây cho kết quả là “tuoi dep”.

#include <iostream>

using namespace std;

int main(void){

char \*s1="Viet Nam tuoi dep", \*s2 = "tu", \* ptr;

ptr = strstr(s1, s2);

cout << "Xau con la :" << ptr << endl;

system("pause");

return 0;

}

Tất nhiên vị trí của s2 trong s1 sẽ là ptr-s1

* Hàm **strset():** char \*strset(char \*s, int ch); Hàm thay mọi kí tự của xâu s bằng kí tự ch.

*Ví du 10*

#include <iostream>

#include <conio.h>

using namespace std;

int main(void){

char s[10] = "123456789";

char symbol = 'c';

cout <<"Truoc khi dung strset() : " << s << '\n';

strset(s, symbol);

cout <<"Sau khi dung strset() : " << s;

getch();

return 0;

}

*Ví dụ 11.* Hãy tạo một xâu là xâu nghịch đảo của một xâu cho trước.

#include <iostream>

#include <conio.h>

using namespace std;

int main(void){

char s[27], c;

cout << "Nhap mot xau chu so " ;

cin.getline(s,100);

cout << s << '\n';

int l = strlen(s);

for (int i=0;i<l/2;i++){ *// Tráo đổi hai ký tự ở vị trí đối xứng*

c = s[i];

s[i] = s[l-i-1];

s[l-i-1] = c;

}

cout << s << '\n';

getch();

return 0;

}

* Hàm **strrev()** đảo ngược chuỗi**:** char \*strrev(char \*s);

Dùng hàm này giải lại ví dụ 11 được chương trình ngắn gọn hơn:

#include <iostream>

#include <conio.h>

using namespace std;

char s[100];

int main(){

cout << "Nhap mot xau chu so " ;

cin.getline(s,100);

cout << "Xau dao nguoc la " << strrev(s);

getch();

return 0;

}

* Hàm **strupr() char \*strupr(char \*s);** Đổi chữ thường ra chữ hoa
* **Hàm strlwr() char \*strlwr(char \*s);** Đổi chữ hoa ra chữ thường.

*Ví dụ 12.* Xây dựng hàm chèn một kí tự vào chuỗi.

#include <iostream>

#include <conio.h>

using namespace std;

char c, s[100];

int i;

int main()

{

cout << "Nhap mot xau chu so " ;

cin.getline(s,100); *// Giả sử s= “123456789”*

cout <<"\nChen vao xau s ki tu c = ";

cin >> c; *//Giả sử c=’T’*

cout << "\nChen truoc s[i] voi i = ";

cin >> i; *//Giả sử i=3*

char s2[100];

strcpy(s2, &s[i]); *//*s2=”456789”

s[i] = c; s[i+1] = '\0'; *//s=”123T”*

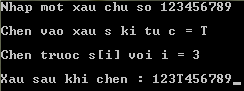
strcat(s,s2); *//Sau khi nối: s=”123T456789”*

cout << "\nXau sau khi chen : " << s;

getch(); return 0;

}

Kết quả trên màn hình:



1. ***Chuyển đổi xâu kí tự chữ số thành kiểu khác***

Trong tệp tiêu đề **stdlib.h** cung cấp một số hàm chuyển xâu các kí tự chữ số thành kiểu dữ liệu dạng số. Đó là các hàm sau:

* **atoi : int atoi(const char \*s);** chuyển xâu chữ số thành số kiểu *int*
* **atol :** chuyển xâu chữ số thành số kiểu *long*
* **atof :** chuyển xâu chữ số thành số kiểu *float*.
* **atold:** chuyển xâu chữ số thành số kiểu *long double*

*Lưu ý.* Các hàm trên chỉ chuyển đổi cho đến khi gặp kí tự khác chữ số trong xâu.

*Ví dụ 13*:

#include <iostream>

#include <iomanip>

using namespace std;

int main(void) {

char \*s = "12345.6789";

int n = atoi(s);

cout << n << endl; *//Hiện số nguyên 12345 (kiểu int)*

long m = atol(s);

cout << m << endl;  *// Hiện số nguyên 12345 (kiểu long)*

float h = atof(s);

cout << fixed << setprecision(3);

cout << h << endl; *//Hiện số thực 12345.679 (kiểu float)*

system("pause");

return 0;

}

### String

Trong trình dịch Dev-C++ có thư viện chuẩn STL (Standard Template Library) cung cấp kiểu *string* dùng thuận tiện khi cần xử lý xâu kí tự.

***Khai báo xâu kí tự s có kiểu string:***

#include <string>

string s;

***Thao tác với xâu kí tự kiểu string trong Dec-C++:***

* Có thể nhập xâu ký tự s từ bàn phím:

cin>>s;

nhưng chỉ nhập được các ký tự cho đến khi gặp ký tự trống của s.

* Nhập các ký tự từ bàn phím vào luồng *cin* rồi gán cho s:

getline(cin,s);

cách này nhập được cả ký tự trống.

* Có thể xuất xâu ký tự s ra màn hình:

cout<<s;

*Ví dụ 1a*. *Sử dụng getline*

#include <iostream>

#include <string>

using namespace std;

string s;

int main() {

cout<< "Nhap mot dong chu: "; getline(cin,s);

cout<< "Dong chu da nhap: "; cout<< s << endl;

system("pause");

return 0;

}

*Ví dụ 1b: Sử dụng getline*

#include <iostream>

#include <string>

using namespace std;

string s1="Tran Thi Thu Ha", s2;

int main() {

cout<<"Chuoi s1 la: "<<s1 <<endl;

cout<<"Nhap chuoi s2: ";

cin>>s2; // *giả sử gõ “Pham van Duong”*

cout<<"Chuoi s2: "<<s2<<endl; *//chỉ hiện ra”Pham”*

getline(cin,s2); *//gán phần còn lại trong luồng cin cho s2*

cout<<"Phan con lai cua s2: "<<s2;*//hiện s2=” van Duong”*

system(“pause”); *//Dừng kết quả trên màn hình cho đến khi nhấn phím*

return 0;

}

Kết quả thực hiện chương trình:



* Nối 2 xâu hoặc ghép một ký tự vào xâu bằng dấu + (điều này chỉ áp dụng với kiểu string, không áp dụng với mảng kí tự).
* Có thể so sánh theo tự điển 2 xâu kiểu string bằng các phép toán so sánh sau:

== (bằng nhau),

!= (khác nhau),

> (lớn hơn),

< (nhỏ hơn),

>= (không nhỏ hơn),

<= (không lớn hơn);

* Nếu s là *string* thì s.length() cho biết độ dài xâu ký tự.
* s[i] cho biết ký tự thứ *i*+1 trong s (0≤i<s.length().

Đây là điều khác biệt cần chú ý khi đối chiếu với string trong Pascal.

* Có thể dùng phép gán (=) với biến có kiểu string.
* Có thể coi *s* là xâu rỗng khi khai báo string s="";
* Giả sử p1 và p2 là hai vị trí lặp (kiểu iterator) của xâu string s thì:

string(p1, p2);

sẽ cho xâu con của s từ *p1* đến *p2-1.*

Ví dụ 2.

#include <iostream>

#include <string>

using namespace std;

string s;

int main() {

cout<< "Nhap mot dong chu: ";

getline(cin,s); *// Giả sử nhập “Do Thi Minh Xuan”*

string::iterator p1, p2;

p1=s.begin()+7;p2=p1+4;

cout << string(p1,p2) << endl; *// Hiện xâu con là “Minh”*

system("pause");

return 0;

}

***Ngoài ra còn có thể sử dụng những phương thức sau:***

* *s.substr(int pos, int nchar)* trích ra một chuỗi con của một chuỗi s. Vis dụ: s.substr(2,4) trả về chuỗi con gồm 4 ký tự của chuỗi s kể từ ký tự ở vị trí thứ 2 (*chú ý ký tự đầu tiên của chuỗi ở vị trí 0)*.

Trong ví dụ 2 nêu trên, các dòng lệnh:

p1=s.begin()+7;p2=p1+4;

cout << string(p1,p2) << endl; *// Hiện xâu con là “Minh”*

có thể thay bằng lệnh:

cout << s.substr(7,4)<<endl;

* *s.find(int ch, int pos);* trả về vị trí lặp của ký tự ch kể từ vị trí *pos* đến cuối chuỗi *s*

*s.find(char \*s1, int pos);* trả về vị trí lặp của mảng ký tự *s1* trong chuỗi *s* kể từ vị trí pos đến cuối

*s.find(string& s1, int pos);* trả về vị trí lặp của chuỗi *s1* trong chuỗi *s* kể từ vị trí *pos* đến cuối.

Trong s.find() nếu không cho *pos* thì hiểu mặc nhiên là pos=0; nếu tìm thấy thì trả về vị trí xuất hiện đầu tiên, ngược lại trả về giá trị -1.

*Ví dụ 3. Tìm và tách riêng từng từ của chuỗi*

#include <iostream>

#include <string>

using namespace std;

int main() {

string s="Nghin nam Thang Long";

string s1="Thang";

string::iterator p, p2; *//Khai báo các biến lặp- iterator p và p2*

cout<<s.find(s1)<<endl; *// Hiện vị trí lặp của s1 trong chuỗi s*

for (p=s.begin(); p<s.end(); ){ *// Vị trí p chuyển từ đầu tới cuối s*

p2=find(p, s.end(),' '); *// vị trí lặp của dấu trống mới kể từ vị trí p*

if (p2!=p) *// Hai vị trí lặp khác nhau*

cout<<string(p, p2)<<endl;// *Xuất một từ giữa p và p2*

p = p2+1; *// vị trí mới của p là vị trí tiếp theo của p2*

}

system(“pause”); *//Dừng kết quả trên màn hình cho đến khi nhấn phím*

return 0;

}

Kết quả thực hiện chương trình:



* *s.compare(p1, n1,hằng xâu s2)* lấy chuỗi con dài *n1* ký tự trong *s1* từ vị trí *p1* so sánh với hằng xâu *s2.*

*s1.compare(p1, n1, s2, p2, n2)* lấy chuỗi con dài *n1* ký tự trong *s1* từ vị trí *p1* so sánh với chuỗi con dài *n2* ký tự lấy trong *s2* từ vị trí *p2*.

*Ví dụ 4. Sử dụng thuộc tính so sánh các chuỗi con trong hai chuỗi:*

#include <iostream>

#include <string>

using namespace std;

int main (){

string s1="green apple";

string s2="red apple";

if (s1!=s2) *//nếu s1 khác s2 thì*

cout << s1 << " is not " << s2 << "\n";

if (s1.compare(6,5,"apple") == 0)

*// nếu “apple” bằng chuỗi con 5 kí tự của s1 kể từ vị trí 6 thì*

cout << s1 << " is an apple\n";

if (s2.compare(4,5,"apple") == 0)

*// nếu “apple” bằng chuỗi con 5 kí tự của s2 kể từ vị trí 4 thì*

cout << "and " << s2 << " is also an apple\n";

if (s1.compare(6,5,s2,4,5) == 0)

*// nếu chuỗi con 5 kí tự của s1 kể từ vị trí 6 bằng chuỗi con 5 kí tự của s2 kể từ vị trí 4 thì*

cout << "therefore, both are apples\n";

system(“pause”); *//Dừng kết quả trên màn hình cho đến khi nhấn phím*

return 0;

}

Kết quả chạy chương trình là:



* *s.size()* cho số lượng kí tự của chuỗi s.
* *s.empty()* trả về 1 nếu chuỗi s rỗng, trả về 0 nếu ngược lại.
* *s.begin()* Trả về iterator (vị trí lặp có kiểu size\_t) đầu tiên của chuỗi s.
* *s.end()* Trả về iterator cuối cùng của chuỗi s.
* *s.front()* Trả về tham chiếu đến phần tử đầu tiên của chuỗi.
* *s.back()* Trả về tham chiếu đến phần tử cuối cùng của chuỗi.
* *s1.swap(s2)* Hoán đổi 2 chuỗi s1 và s2 với nhau.
* *s.insert(int pos, char\* a);* chèn mảng ký tự a vào vị trí *pos* của s.
* *s.insert(int pos, string s1);* chèn chuỗi s1 (kiểu string) vào vị trí *pos* của chuỗi s.
* *s.insert(int pos, int n, int ch)*; chèn n lần ký tự ch vào vị trí *pos* của chuỗi s.
* *s.erase(int pos, int n)* xóa n ký tự của chuỗi s kể từ vị trí *pos*. Nếu không quy định giá trị n thì tấtcả các ký tự của s từ vị trí *pos* trở đi sẽ bị xóa.

*Lưu ý.* Biến s kiểu string khác với biến ss kiểu xâu ký tự (là kiểu con trỏ kí tự char \* ss; hoặc kiểu mảng kí tự char ss[];) mặc dù chúng có thể dùng biểu diễn cùng một xâu như nhau. Trong nhiều trường hợp cần chuyển string thành mảng xâu ký tự có thể chuyển dần từng kí tự của s thành phần tử tương ứng của ss, rồi kết thúc ss bởi ký tự null (‘\0’).

* Chuyển string thành mảng xâu kí tự :

string s=”BALANCE.1”; *//Có 9 ký tự trong s*

char ss[10];

for (int k=0; k<10; k++) ss[k]=s[k];

ss[10]='\0';

* Ngược lại chuyển mảng ký tự ss thành string s như sau:

char ss[10]=”BALANCE.1”; *//Có 9 ký tự trong ss và kết thúc bới null*

string s=””;

for (int k=0; k<9; k++) s=s+ss[k];

*Ví dụ 5. Xây dựng hàm* ***toString(const tnumber x)*** *chuyển mảng số x thành một xâu kiểu string:*

#include <sstream>

#include <string>

#include <iomanip>

using namespace std;

*…*

*//giả sử đã có mảng x một chiều biểu diễn số lớn, mỗi phần tử là số có tối đa 8 chữ số*

*…*

inline string toString(const tnumber x) {

ostringstream o;

o << x[n-1];

for (int i=n-2; i>=0; i--) {

o << setfill('0') << setw(8) << x[i];

}

return o.str();

}

…

*Giải thích.* Sử dụng tệp tiêu đề <sstream> và khai báo luồng xuất string là *o* bằng lệnh:

ostringstream o;

Giả sử mảng x[0..n-1] có n phần tử, dùng biểu diễn một số nguyên lớn mỗi phần tử là một số nguyên có tối đa 8 chữ số. Ví dụ số nguyên lớn là: 12000003456700000800000089,

được chứa vào mảng x có 4 phần tử (n=4):

x[3]=12; x[2]=00000345=345;

x[1]=67000008; x[0]=00000089=89;

Chúng ta lần lượt đổ các phần tử x[3], rồi x[2], x[1], x[0] của mảng (từ cuối về đầu) vào luồng *o*, sau đó xuất ra string bằng lệnh:

return o.str();

đó là xâu “12000003456700000800000089” đang nằm trong luồng.

* Hàm **memcpy**() có dạng:

void \* memcpy(void \* destination,const void \* source, size\_t num );

Hàm này làm nhiệm vụ sao chép một khối bộ nhớ (là *num* byte) từ địa chỉ do con trỏ của nguồn *source,* vào địa chỉ do con trỏ đích *destination* trỏ tới*.*

*Ví dụ 6. Chuyển string sang kiểu char \* (kiểu con trỏ kí tự)*

*#include <iostream>*

*#include <string>*

*using namespace std;*

*int main() {*

*string s;*

*cout<<"Nhap xau ki tu: "; getline(cin,s);*

*int L=s.size(); //Độ dài string s*

*char \* a = new char[L+1]; //Xin cấp phát bộ nhớ cho biến a*

*a[L]='\0'; // Kết thúc sẵn cho mảng xâu kí tự a*

*memcpy(a,s.c\_str(),L); //Sao chép s.c\_str()với độ dài L sang a*

*cout << a << endl; //Xuất ra xâu kí tự do a trỏ tới*

*system("pause");*

*return 0;*

*}*

## Đọc và ghi dữ liệu tệp.

Để đọc hoặc ghi tệp, cần có dòng khai báo tệp tiêu đề fstream:

#include <fstream>

Sau đó, khai báo các đối tượng liên kết với *tệp tin để đọc dữ liệu* bằng cách dùng ifstream; hoặc khai báo các đối tượng liên kết với *tệp tin để ghi dữ liệu* bằng cách dùng ofstream.

Ví dụ khai báo đối tượng *fin* liên kết tới các tệp tin để đọc như sau:

ifstream fin; // (1)

Dùng đối tượng *fin* mở tệp tin “BAITAP01.IN” để sau đó có thể dùng lệnh đọc dữ liệu trong tệp tin này thì dùng lệnh:

fin.open (“BAITAP01.IN”); // (2)

Hoặc có thể viết gộp hai lệnh (1) và (2) trên thành một dòng lệnh:

ifstream fin (“BAITAP01.IN”);

Sau đó có thể đọc lần lượt giá trị trong tệp “BAITAP01.IN” có kiểu tương ứng với kiểu của biến x bằng lệnh:

f in>> x;

Tương tự, khai báo đối tượng *fout* liên kết tới tệp tin để ghi như sau:

ofstream fout; // (3)

Dùng đối tượng *fout* (khai báo ở trên) mở tệp tin “BAITAP01.OUT” để sau đó sẽ ghi (kết quả ra) vào nó thì dùng lệnh:

fout.open (“BAITAP01.OUT”); // (4)

Hoặc có thể viết gộp hai lệnh (3) và (4) trên thành:

ofstream fout (“BAITAP01.OUT”);

Để ghi một giá trị x vào tệp tin “BAITAP01.OUT” do đối tượng *fout* liên kết tới thì dùng lệnh:

fout << x;

Sau khi làm việc xong với tệp cần đóng tệp: Ví dụ để đóng tệp BAITAP01.IN do đối tượng fin đã mở ta dùng lệnh:

fin.close();

Để đóng tệp BAITAP01.OUT do đối tượng *fout* đã mở ta dùng lệnh fout.close();

***Cách đọc từng dòng của tệp:***

**Bài toán.** Cho N điểm, hai điểm trong chúng có thể nối với nhau bằng một cạnh. Tệp dữ liệu vào “PATH.IN” tổ chức như sau: Tệp gồm một số dòng, mỗi dòng ghi ở đầu dòng một số *i*, sau đó ghi các số khác trên cùng dòng là các số *j*, thể hiện có cạnh nối điểm *i* với các điểm *j*. Hãy đọc tệp để biết các điểm nào được nối với nhau (dùng mảng hai chiều A[][] để lưu lại, A[i][j]=1 và A[j][i]=1 thể hiện có cạnh nối điểm i với điểm j).

Ví dụ: Cho tệp PATH.IN như sau

13

0 3 4 5 6 9

1 3 5 6 7 9 10

2 3 6 7 8 9 10 11

3 7 10

4 7 10

5 6 7 8 10 11

6 7 10 11 12

7 11 12

8 12

Tệp này mô tả:

Có N=13 điểm được đánh số từ 0 đến 12;

Các dòng tiếp theo: số ở đầu dòng là điểm *i* có cạnh nối với các điểm *j* ghi trên cùng một dòng với *i*. Ví dụ dòng 6 trong các dòng tiếp theo là:

5 6 7 8 10 11

nên điểm 5 có cạnh nối với các điểm 6,7,8,10,11.

Sau đây là hàm đọc dữ liệu từ tệp PATH.IN

#include <fstream>

#include <sstream>

using namespace std;

ifstream fin (“PATH.IN”);

void doc(){

int i, j;

string s;

fin >> N; *// Đọc số đỉnh*

while (getline(fin, s)) { *// Đọc một dòng của tệp vào xâu s*

istringstream ss(s); *// Đổ xâu s vào luồng string là ss*

ss >> i; *// Lấy số thứ nhất ra khỏi luồng ss*

while (ss>>j) { *//Lần lượt lấy các số còn lại ra khỏi ss*

A[i][j] =1; *// Xác nhận cạnh nối điểm i và điểm j*

A[j][i] =1;

}

}

fin.close();