Biên soạn
NGUYỄN TIẾN ĐỨC

Tin học 11

Sử dụng ngôn ngữ C++

Hòa Bình, tháng 10 năm 2020

MỤC LỤC

Chương 1: Một số khái niệm về lập trình và ngôn ngữ lập trình

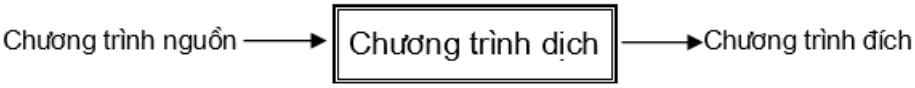
Bài 1. KHÁI NIỆM VỀ LẬP TRÌNH

Như ta biết, mọi bài toán có thuật toán đều có thể giải được trên máy tính điện tử. Khi giải bài toán trên máy tính điện tử, sau các bước xác định bài toán và xây dựng hoặc lựa chọn thuật toán khả thi là bước lập trình.

Lập trình là sử dụng cấu trúc dữ liệu và các câu lệnh của ngôn ngữ lập trình cụ thể để mô tả dữ liệu và diễn đạt các thao tác của thuật toán. Chương trình viết bằng ngôn ngữ lập trình bậc cao nói chung không phụ thuộc vào loại máy, nghĩa là một chương trình có thể thực hiện trên nhiều loại máy tính khác nhau. Chương trình viết bằng ngôn ngữ máy có thể được nạp trực tiếp vào bộ nhớ và thực hiện ngay, còn chương trình viết bằng ngôn ngữ lập trình bậc cao phải được chuyển đổi thành chương trình trên ngôn ngữ máy mới có thể thực hiện được.

Chương trình đặc biệt có chức năng chuyển đổi chương trình được viết bằng ngôn ngữ lập trình bậc cao thành chương trình thực hiện được trên máy tính cụ thể được gọi là *chương trình dịch*.

Chương trình dịch nhận đầu vào là chương trình viết bằng ngôn ngữ lập trình bậc cao (chương trình nguồn), thực hiện chuyển đổi sang ngôn ngữ máy (chương trình đích).



Hình 1.1 - Minh họa quá trình chuyển đổi chương trình nguồn thành chương trình đích

Xét ví dụ, bạn chỉ biết tiếng Việt nhưng cần giới thiệu về trường của mình cho đoàn khách đến từ nước Mĩ, chỉ biết tiếng Anh. Có hai cách để bạn thực hiện điều này.

Cách thứ nhất: Bạn nói bằng tiếng Việt và người phiên dịch giúp bạn dịch sang tiếng Anh. Sau mỗi câu hoặc một vài câu giới thiệu trọn một ý, người phiên dịch dịch sang tiếng Anh cho đoàn khách. Sau đó, bạn lại giới thiệu tiếp và người phiên dịch lại dịch tiếp. Việc giới thiệu của bạn và việc dịch của người phiên dịch luân phiên cho đến khi bạn kết thúc nội dung giới thiệu của mình. Cách dịch trực tiếp như vậy được gọi là *thông dịch*.

Cách thứ hai: Bạn soạn nội dung giới thiệu của mình ra giấy, người phiên dịch dịch toàn bộ nội dung đó sang tiếng Anh rồi đọc hoặc trao văn bản đã dịch cho đoàn khách đọc. Như vậy, việc dịch được thực hiện sau khi nội dung giới thiệu đã hoàn tất. Hai công việc đó được thực hiện trong hai khoảng thời gian độc lập, tách biệt nhau. Cách dịch như vậy được gọi là *biên dịch*.

Sau khi kết thúc, với cách thứ nhất không có một văn bản nào để lưu trữ, còn với cách thứ hai có hai bản giới thiệu bằng tiếng Việt và bằng tiếng Anh có thể lưu trữ để dùng lại về sau.

Tương tự như vậy, chương trình dịch có hai loại là *thông dịch* và *biên dịch*.

a) Thông dịch

Thông dịch (interpreter) được thực hiện bằng cách lặp lại dãy các bước sau:

- Kiểm tra tính đúng đắn của câu lệnh tiếp theo trong chương trình nguồn;
- Chuyển đổi câu lệnh đó thành một hay nhiều câu lệnh tương ứng trong ngôn ngữ máy;
- Thực hiện các câu lệnh vừa chuyển đổi được.

Như vậy, quá trình dịch và thực hiện các câu lệnh là luân phiên. Các chương trình thông dịch lần lượt dịch và thực hiện từng câu lệnh. Loại chương trình dịch này đặc biệt thích hợp cho môi trường đối thoại giữa người và hệ thống. Tuy nhiên, một câu lệnh nào đó phải thực hiện bao nhiêu lần thì nó phải được dịch bấy nhiêu lần.

Các ngôn ngữ khai thác hệ quản trị cơ sở dữ liệu, ngôn ngữ đối thoại với hệ điều hành,... đều sử dụng trình thông dịch. Python là ngôn ngữ thông dịch

b) Biên dịch

Biên dịch (compiler) được thực hiện qua hai bước:

- Duyệt, kiểm tra, phát hiện lỗi, kiểm tra tính đúng đắn của các câu lệnh trong chương trình nguồn;
- Dịch toàn bộ chương trình nguồn thành một chương trình đích có thể thực hiện trên máy và có thể lưu trữ để sử dụng lại khi cần thiết.

Như vậy, trong thông dịch, không có chương trình đích để lưu trữ, trong biên dịch cả chương trình nguồn và chương trình đích có thể lưu trữ lại để sử dụng vềsau.

Thông thường, cùng với chương trình dịch còn có một số dịch vụ liên quan như biên soạn, lưu trữ, tìm kiếm, cho biết các kết quả trung gian,... Toàn bộ các dịch vụ trên tạo thành một môi trường làm việc trên một ngôn ngữ lập trình cụ thể. Ví dụ, Turbo Pascal 7.0, Free Pascal1.2, Visual Pascal 2.1,... trên ngôn ngữ Pascal, Turbo C++, Visual C++,... trên ngôn ngữ C++.

Các môi trường lập trình khác nhau ở những dịch vụ mà nó cung cấp, đặc biệt là các dịch vụ nâng cấp, tăng cường các khả năng mới cho ngôn ngữ lập trình.

BÀI ĐỌC THÊM 1

Có thể sử dụng bài viết từ địa chỉ : https://vi.wikipedia.org/wiki/Ngôn ngữ lập trình.

Bài 2. CÁC THÀNH PHẦN CỦA NGÔN NGỮ LẬP TRÌNH

1. Các thành phần cơ bản

Mỗi ngôn ngữ lập trình thường có ba thành phần cơ bản là *bảng chữ cái*, *cú pháp* và *ngữ nghĩa*.

a) Bảng chữ cái

Là tập các kí tự được dùng để viết chương trình. Không được phép dùng bất kì kí tự nào ngoài các kí tự quy định trong bảng chữ cái.

Trong C++, bảng chữ cái bao gồm các kí tự:

- Các chữ cái in thường và các chữ cái in hoa của bảng chữ cái tiếng Anh:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

- 10 chữ số thập phân Ả Rập: `0 1 2 3 4 5 6 7 8 9`
- Các kí tự đặc biệt:

```
+     -     *     /        =        <     >     [     ]     .    ,
;     #     ^     $     @     &     (     )     {     }     :    '

           Dấu cách (mã ASCII là 32)                        _
```

Bảng chữ cái của các ngôn ngữ lập trình nói chung không khác nhau nhiều. Ví dụ, bảng chữ cái của ngôn ngữ lập trình C++ khác Pascal là sử dụng thêm các kí tự như dấu nháy kép ("), dấu sổ ngược (\), dấu chấm than (!) ...

b) Cú pháp

Là bộ quy tắc để viết chương trình. Dựa vào chúng, người lập trình và chương trình dịch biết được tổ hợp nào của các kí tự trong bảng chữ cái là hợp lệ và tổ hợp nào là không hợp lệ. Nhờ đó, có thể mô tả chính xác thuật toán để máy thực hiện.

c) Ngữ nghĩa

Xác định ý nghĩa thao tác cần phải thực hiện, ứng với tổ hợp kí tự dựa vào ngữ cảnh của nó.

Ví dụ

Phần lớn các ngôn ngữ lập trình đều sử dụng dấu cộng (+) để chỉ phép cộng. Xét các biểuthức:

```
A + B     (1)
I + J     (2)
```

Giả thiết *A, B* là các đại lượng nhận giá trị thực và *I, J* là các đại lượng nhận giá trị chuỗi ký tự. Khi đó dấu "+" trong biểu thức (1) được hiểu là cộng hai số thực, dấu "+" trong biểu thức (2) được hiểu là phép nối hai chuỗi. Như vậy, ngữ nghĩa dấu "+" trong hai ngữ cảnh khác nhau là khác nhau.

Tóm lại, cú pháp cho biết cách viết một chương trình hợp lệ, còn ngữ nghĩa xác định ý nghĩa của các tổ hợp kí tự trong chương trình.

Các lỗi cú pháp được chương trình dịch phát hiện và thông báo cho người lập trình biết. Chỉ có các chương trình không còn lỗi cú pháp mới có thể được dịch sang ngôn ngữ máy.

Các lỗi ngữ nghĩa khó phát hiện hơn. Phần lớn các lỗi ngữ nghĩa chỉ được phát hiện khi thực hiện chương trình trên dữ liệu cụ thể.

2. Một số khái niệm

a) Tên (Identifiers)

Mọi đối tượng trong chương trình đều phải được đặt tên theo quy tắc của ngôn ngữ lập trình và từng chương trình dịch cụ thể.

Trong C++, tên là một dãy liên tiếp có số kí tự tùy ý bao gồm chữ số, chữ cái hoặc dấu gạch dưới và bắt đầu bằng chữ cái hoặc dấu gạch dưới.

Ví dụ, trong ngôn ngữ C++:

- Các tên đúng:

```
A
R21
P21_c
_45
```

- Các tên sai:

```
A BC      (chứa kí tự trắng)
6Pq       (bắt đầu bằng chữ số)
X#Y       (chứa kí tự "#" không hợp lệ)
```

Ngôn ngữ C++ ***phân biệt chữ hoa, chữ thường*** trong tên. Khác với NNLT Pascal không phân biệt chữ hoa, chữ thường. Vídụ, *AB* và *Ab* là hai tên khác nhau trong C++, nhưng lại là cùng một tên trong Pascal.

Nhiều ngôn ngữ lập trình, trong đó có C++, phân biệt hai loại tên:

- Tên dành riêng;
- Tên do người lập trình đặt.

Tên dành riêng

Là loại tên được ngôn ngữ lập trình quy định dùng với ý nghĩa xác định, người lập trình không được sử dụng với ý nghĩa khác. Những tên này được gọi là tên dành riêng (còn được gọi là từ khoá).

Ví dụ. Một số tên dành riêng:

Trong Pascal: `while, else, true, if, define, for`

Trong C++: `main, include, if, while, void`

Tên do người lập trình đặt

Tên do người lập trình đặt được dùng với ý nghĩa riêng, xác định bằng cách khai báo trước khi sử dụng. Các tên này không được trùng với tên dành riêng.

Ví dụ

Tên do người lập trình đặt:

```
A1
DELTA
CT_Vidu
```

b) Hằng và biến

Hằng (constants)

Hằng là các đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.

Trong các ngôn ngữ lập trình thường có các hằng số học, hằng lôgic, hằng xâu.

- Hằng số học là các số nguyên hay số thực (dấu phẩy tĩnh hoặc dấu phẩy động), có dấu hoặc không có dấu.
- Hằng lôgic là giá trị đúng hoặc sai tương ứng với *true* hoặc *false*. Ngoài ra C++ còn coi số 0 là sai, số khác 0 là đúng.
- Hằng xâu là chuỗi kí tự trong bảng chữ cái. Trong C++, khi viết chuỗi kí tự được đặt trong cặp dấu nháy kép.

Ví dụ

- Hằng số học:

```
2            0        -5            +18
1.5          -22.36   +3.14159      0.5
-2.236E01    1.0E-6
```

- Hằng lôgic: Trong C++ chỉ viết chữ in thường

```
true     false
```

- Hằng ký tự: Trong C++ và Pascal đều viết hằng ký tự trong cặp dấu nháy đơn như sau:

```
'A'      '3'
```

- Hằng xâu:

+ Trong Pascal: Hằng xâu đặt trong cặp dấu nháy đơn

```
'Informatic'      'TIN HOC'
```

+ Trong C++: Hằng xâu đặt trong cặp dấu nháy kép, khác hằng ký tự.

```
"Informatic"      "TIN HOC"
```

Biến (variables)

Biến là đại lượng được đặt tên, dùng để lưu trữ giá trị và giá trị có thể được thay đổi trong quá trình thực hiện chương trình.

Tuỳ theo cách lưu trữ và xử lí, C++ phân biệt nhiều loại biến. Các biến dùng trong chương trình đều phải khai báo. Việc khai báo biến sẽ được trình bày ở các phần sau.

c) Chú thích (comments)

Có thể đặt các đoạn chú thích trong chương trình nguồn. Các chú thích này giúp cho người đọc chương trình nhận biết ngữ nghĩa của chương trình đó dễ hơn. Chú thích không ảnh hưởng đến nội dung chương trình nguồn và được chương trình dịch bỏ qua.

Trong C++ có hai loại chú thích: chú thích trên một dòng, chú thích gồm nhiều dòng

- Chú thích trên một dòng được bắt đầu bằng dấu //

  Ví dụ: Dòng sau đây là chú thích và được trình phiên dịch C++ bỏ qua

```
// Hello everybody
```

- Chú thích trên nhiều dòng được bắt đầu bằng ký hiệu /* và kết thúc bằng ký hiệu */

  Ví dụ: Các dòng sau đây là chú thích và được trình phiên dịch C++ bỏ qua

```
/* This is also a
perfect example of
multi-line comments */
```

TÓM TẮT

- Cần có chương trình dịch để chuyển chương trình nguồn thành chương trình đích.
- Có hai loại chương trình dịch: thông dịch và biên dịch.
- Các thành phần của ngôn ngữ lập trình: bảng chữ cái, cú pháp và ngữ nghĩa.
- Mọi đối tượng trong chương trình đều phải được đặt tên:
  - Tên dành riêng: Được dùng với ý nghĩa riêng, không được dùng với ý nghĩa khác.
  - Tên do người lập trình đặt: cần khai báo trước khi sử dụng.
- Hằng: Đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.
- Biến: Đại lượng được đặt tên. Giá trị của biến có thể thay đổi trong quá trình thực hiện chương trình.

Câu hỏi và bài tập

1. Tại sao người ta phải xây dựng các ngôn ngữ lập trình bậc cao?

2. Chương trình dịch là gì? Tại sao cần phải có chương trình dịch?

3. Biên dịch và thông dịch khác nhau như thế nào?

4. Hãy cho biết các điểm khác nhau giữa tên dành riêng và tên do người dùng đặt.

5. Hãy tự viết ra ba tên đúng theo quy tắc của C++.

6. Hãy cho biết những biểu diễn nào dưới đây không phải là biểu diễn hằng trong C++ và chỉ rõ lỗi trong từng trường hợp:

```
a) 150.0     b) -22       c) 6,23       d) <43>
e) A20       f) 1.06E-15  g) 4+6        h) 'C
i) 'TRUE'
```

BÀI ĐỌC THÊM 2

Có thể sử dụng bài viết từ wikipedia theo địa chỉ: https://vi.wikipedia.org/wiki/C%2B%2B

Chương 2: Chương trình đơn giản

Bài 3. CẤU TRÚC CHƯƠNG TRÌNH

1. Cấu trúc chung

Nói chung, chương trình được viết bằng một ngôn ngữ lập trình bậc cao thường gồm phần khai báo và phần thân. Phần thân chương trình nhất thiết phải có. Phần khai báo có thể có hoặc không tuỳ theo từng chương trình cụ thể.

Khi diễn giải cú pháp của ngôn ngữ lập trình người ta thường sử dụng ngôn ngữ tự nhiên. Các diễn giải bằng ngôn ngữ tự nhiên được đặt giữa cặp dấu < và >. Các thành phần của chương trình có thể có hoặc không được đặt trong cặp dấu [và].

Với quy ước trên, cấu trúc của một chương trình có thể được mô tả như sau:

```
<phần khai báo>
<phần thân>
```

2. Các thành phần của chương trình

a) Phần khai báo

Có thể có các khai báo: thư viện cần dùng, hằng, biến và chương trình con. Tuy nhiên phần này tùy theo từng chương trình mà có nhiều hoặc ít lời khai báo.

a.1) Khai báo thư viện

Mỗi ngôn ngữ lập trình thường có sẵn một số thư viện cung cấp một số chương trình thông dụng đã được lập sẵn. Để sử dụng các chương trình đó cần khai báo thư viện chứa nó.

Ví dụ. Khai báo thư viện các đối tượng vào/ra chuẩn trong C++

```
#include <iostream>
```

Thư viện *iostream* trong C++ cung cấp các tiện tích có sẵn để làm việc với bàn phím và màn hình. Ví dụ, muốn in ra màn hình xâu ký tự "HOA BINH XIN CHAO CAC BAN", ta viết:

```
cout << "HOA BINH XIN CHAO CAC BAN";
```

a.2) Khai báo hằng

Ví dụ. Khai báo hằng trong C++:

```
const int MaxN = 1000;
const float PI = 3.1416;
```

Khai báo hằng thường được sử dụng cho những giá trị cố định mà xuất hiện nhiều lần trong chương trình.

a.3) Khai báo biến

Tất cả các biến dùng trong chương trình đều phải đặt tên cho chương trình dịch biết để lưu trữ và xử lí. Biến chỉ nhận một giá trị tại mỗi thời điểm thực hiện chương trình được gọi là biến đơn.

Ví dụ

Khi khảo sát phương trình đường thẳng *ax + by + c = 0*, các hệ số *a, b, c* có thể được khai báo như những biến đơn.

Cách khai báo biến được trình bày riêng trong bài 5.

b) Phần thân chương trình

Trong C++ không có khái niệm chương trình chính, chương trình C++ được tổ chức dưới dạng các hàm, trong đó có hàm *main()* là nơi bắt đầu mọi hoạt động của chương trình. Để cho dễ tưởng tượng, hãy coi thân hàm *main()* như là một kiểu chương trình chính.

3. Ví dụ chương trình đơn giản

Dưới đây xét một vài ví dụ về những chương trình đơn giản.

Ví dụ 1.

Chương trình sau thực hiện việc đưa ra màn hình thông báo "*Xin chao cac ban!*".

Trong C++

```
#include <iostream>
main()
{
  cout << "Xin chao cac ban!";
}
```

- Phần khai báo chỉ có một câu lệnh *#include* khai báo thư viện *iostream*.
- Phần thân chương trình chỉ có một hàm *main()* trong đó có câu lệnh *cout* đưa thông báo ra màn hình.

Trong Pascal

```
program vi_du;
begin
  writeln('Xin chao cac ban!');
end.
```

- Phần khai báo chỉ có khai báo tên chương trình gồm tên dành riêng *program* và tên chương trình là *vi_du*.
- Phần thân chương trình chỉ có một câu lệnh *writeln*, đưa thông báo ra màn hình.

Ví dụ 2.

Chương trình C++ sau đưa các thông báo "*Xin chao cac ban!*" và "*Moi cac ban lam quen voi C++*" ra màn hình.

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     cout <<"Xin chao cac ban!\n";
5.     cout <<"Moi cac ban lam quen voi C++\n";
6.     return 0;
7. }
```

Như vậy, khi muốn đưa một thông báo nào đó ra màn hình trong C++, ta có thể dùng câu lệnh *cout*, dòng thông báo được viết như một hằng xâu và đặt sau ký hiệu <<.

Bài 4. MỘT SỐ KIỂU DỮ LIỆU CHUẨN

Các bài toán trong thực tế thường có dữ liệu vào và kết quả ra thuộc những kiểu dữ liệu quen biết như số nguyên, số thực, kí tự,... Khi lập trình cho những bài toán như vậy, khi cần người lập trình sử dụng các kiểu dữ liệu đó thường gặp một số hạn chế nhất định, phụ thuộc vào một số yếu tố như dung lượng bộ nhớ, khả năng xử lí của CPU,...

Vì vậy, mỗi ngôn ngữ lập trình thường cung cấp một số kiểu dữ liệu chuẩn cho biết phạm vi giá trị có thể lưu trữ, dung lượng bộ nhớ cần thiết để lưu trữ và các phép toán tác động lên dữ liệu. Dưới đây xét một số kiểu dữ liệu chuẩn thường dùng cho các biến đơn trong C++.

1. Kiểu nguyên

Kiểu	Bộ nhớ lưu trữ một giá trị	Phạm vi giá trị
`char`	1 bye	[0, 255]
`short`	2 byte	$[-2^{16}, 2^{16}-1]$
`int`	4 byte	$[-2^{32}, 2^{31}-1]$
`long long`	8 byte	$[-2^{64}, 2^{63}-1]$

2. Kiểu thực

Kiểu	Bộ nhớ lưu trữ một giá trị	Phạm vi giá trị
`float`	4 bye	(±1.18E-38, ±3.4E38), độ chính xác đến 9 chữ số thập phân
`double`	8 byte	(±2.23E-308, ±1.80E308), độ chính xác đến 15 chữ số thập phân
`long double`	10 byte	(±3.36E-4932, ±1.18E4932), độ chính xác đến 36 chữ số thập phân

3. Kiểu kí tự

Kí tự là các kí hiệu thuộc bảng mã ASCII, gồm 256 kí tự có mã ASCII thập phân từ 0 đến 255.

Ví dụ, kí tự 'A' có mã ASCII là 65, kí tự 'a' có mã ASCII là 97. Kí tự đặc biệt, dùng để thể hiện sự ngăn cách giữa hai từ viết liên tiếp trong các văn bản, là dấu cách. Dấu cách được gõ bằng phím Space - phím dài nhất trên bàn phím và có mã ASCII bằng 32.

Kiểu	Bộ nhớ lưu trữ một giá trị	Phạm vi giá trị
`char`	1 bye	256 kí tự trong bộ mã ASCII

4. Kiểu lôgic

Kiểu	Bộ nhớ lưu trữ một giá trị	Phạm vi giá trị
`bool`	1 bye	true/false

Ghi chú

- Một số ngôn ngữ lập trình mô tả giá trị lôgic bằng cách khác.

- Người lập trình cần tìm hiểu đặc trưng của các kiểu dữ liệu chuẩn được xác định bởi bộ dịch và sử dụng để khai báo biến.

Bài 5. KHAI BÁO BIẾN

Như đã nêu ở trên, mọi biến dùng trong chương trình đều cần khai báo tên và kiểu dữ liệu của biến. Tên biến dùng để xác lập quan hệ giữa biến với địa chỉ bộ nhớ nơi lưu trữ giá trị biến. Mỗi biến chỉ được khai báo một lần. Trong phần này ta chỉ xét khai báo các biến đơn.

Trong C++, khai báo biến có cú pháp như sau:

```
<kiểu dữ liệu> <danh sách biến>;
```

Trong đó:

- *danh sách biến* là một hoặc nhiều tên biến, các tên biến được viết cách nhau bởi dấu phẩy;
- *kiểu dữ liệu* thường là một trong các kiểu dữ liệu chuẩn hoặc kiểu dữ liệu do người lập trình định nghĩa;

Cấu trúc khai báo biến có thể xuất hiện nhiều lần và ở vị trí bất kỳ trong chương trình.

Ví dụ 1

Giả sử trong chương trình cần các biến thực *A, B, C, D, X1, X2* và các biến nguyên *M, N*. Khi đó có thể khai báo các biến đó như sau:

```
float A, B, C, X1, X2, D;
int  M, N;
```

Ví dụ 2

Xét khai báo biến:

```
float X, Y, Z;
char C;
short I, J;
int N;
```

Trong khai báo này có ba biến thực *X, Y, Z*. Bộ nhớ cấp phát cho ba biến này là 12 byte (3 × 4 = 12). *C* là biến kí tự (và cũng là 1 kiểu số nguyên) và bộ nhớ dành cho nó là 1 byte. Các biến *I, J* bộ nhớ dành cho mỗi biến là 2 byte. Biến *N* cũng nhận các giá trị nguyên, nhưng bộ nhớ cấp phát cho biến *N* là 4 byte. Như vậy, tổng bộ nhớ dành cho các biến đã khai báo là:

4 + 4 + 4 + 1 + 2 + 2 + 4 = 21 (byte).

Một số chú ý khi khai báo biến:

- Cần đặt tên biến sao cho gợi nhớ đến ý nghĩa của biến đó. Điều này rất có lợi cho việc đọc, hiểu và sửa đổi chương trình khi cần thiết.

Ví dụ, không nên vì cho ngắn gọn mà đặt tên biến là *d1, d2* mà nên đặt là *dtoan, dtin* gợi nhớ tới ngữ nghĩa của các biến đó là điểm toán, điểm tin của học sinh.

- Không nên đặt tên biến quá ngắn hay quá dài, dễ mắc lỗi khi viết nhiều lần tên biến. Ví dụ, không nên dùng *d1, d2* hay *diemmontoan, diemmontin* cho điểm toán, điểm tin của học sinh.

- Khi khai báo biến cần đặc biệt lưu ý đến phạm vi giá trị của nó. Ví dụ, khi khai báo biến biểu diễn số học sinh của một lớp có thể sử dụng kiểu *char*, nhưng biến biểu diễn số học sinh của toàn trường thì phải là kiểu *short*.
- Trong C++ cho phép ta vừa khai báo biến vừa khởi tạo giá trị cho biến:

```
int cnt = 0, a = 1, b = 2;
```

Bài 6. PHÉP TOÁN, BIỂU THỨC VÀ LỆNH GÁN

Để mô tả các thao tác trong thuật toán, mỗi ngôn ngữ lập trình đều xác định và sử dụng một số khái niệm cơ bản: phép toán, biểu thức, gán giá trị cho biến. Ta sẽ đi xét các khái niệm đó trong C++.

1. Phép toán

Tương tự trong toán học, trong các ngôn ngữ lập trình đều có những phép toán số học như cộng, trừ, nhân, chia trên các đại lượng thực, các phép toán chia nguyên và lấy phần dư, các phép toán quan hệ,… Bảng dưới đây là kí hiệu các phép toán đó trong toán và trong C++:

a. Phép toán số học với số nguyên

Tên phép toán	Trong C++	Trong Pascal	Ví dụ
cộng	+	+	x + y
trừ	-	-	x - y
nhân	*	*	x * y
chia nguyên	/	DIV	x / y
lấy phần dư	%	MOD	x % y
tăng thêm 1 đơn vị	++	không có	x++
giảm đi 1 đơn vị	--	không có	x--

b. Phép toán số học với số thực

Tên phép toán	Trong C++	Trong Pascal	Ví dụ
cộng	+	+	x + y
trừ	-	-	x - y
nhân	*	*	x * y
chia	/	/	x / y

Chú ý: trong C++ phép chia (/) hai giá trị nguyên thì cho kết quả nguyên, còn nếu có một trong hai vế của phép chia là giá trị thực thì kết quả là thực, vấn đề này rất dễ nhầm lẫn.

c. Phép toán quan hệ

Tên phép toán	Trong C++	Trong Pascal	Ví dụ
so sánh bằng	==	=	x == y
nhỏ hơn	<	<	x < y
lớn hơn	>	>	x > y
nhỏ hơn hoặc bằng	<=	<=	x <= y
lớn hơn hoặc bằng	>=	>=	x >= y
khác	!=	<>	x != y

d. Phép toán logic

Tên phép toán	Trong C++	Trong Pascal	Ví dụ
phủ định	!	not	!x
hoặc	||	or	x || y
và	&&	and	x && y

Chú ý:

- Kết quả của các phép toán quan hệ là kiểu lôgic.

- Một trong những ứng dụng của phép toán lôgic là để tạo ra các biểu thức phức tạp từ các quan hệ đơn giản.

2. Biểu thức số học

Trong lập trình, biểu thức số học là một biến kiểu số hoặc một hằng số hoặc các biến kiểu số và các hằng số liên kết với nhau bởi một số hữu hạn phép toán số học, các dấu ngoặc đơn (và) tạo thành một biểu thức có dạng tương tự như cách viết trong toán học với những quy tắc sau:

- Chỉ dùng các cặp ngoặc đơn để xác định trình tự thực hiện phép toán trong trường hợp cần thiết;
- Viết lần lượt từ trái qua phải;
- Không được bỏ qua dấu nhân (*) trong tích.

Các phép toán được thực hiện theo thứ tự:

- Ưu tiên thực hiện các phép toán trong ngoặc trước;
- Trong dãy các phép toán không chứa ngoặc thì thực hiện từ trái sang phải, theo thứ tự các phép toán nhân (*), chia (/), lấy phần dư (%) thực hiện trước và các phép toán cộng (+), trừ (-) thực hiện sau.

Ví dụ

Biểu thức trong toán học	**Biểu thức trong C++**
`5a + 6b`	`5*a + 6*b`
$\frac{xy}{z}$	`x*y/z`
`Ax`^{2} `+ Bx + C`	`A*x*x + B*x + C`
$\frac{x+y}{x-\frac{1}{2}} - \frac{x-z}{xy}$	`(x+y)/(x-1/2) - (x-z)/(x*y)`

Chú ý

a) Nếu biểu thức chứa một hằng hay biến kiểu thực thì ta có biểu thức số học thực, giá trị của biểu thức cũng thuộc kiểu thực.

b) Trong một số trường hợp nên dùng biến trung gian để có thể tránh được việc tính một biểu thức nhiều lần.

c) Phép toán tăng ++*i* và *i*++ sẽ cùng tăng *i* lên 1 đơn vị, 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 hoạ thông qua ví dụ sau, 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`

3. Hàm số học chuẩn

Để lập trình được dễ dàng, thuận tiện hơn, các ngôn ngữ lập trình đều có thư viện chứa một số chương trình tính giá trị những hàm toán học thường dùng. Các chương trình như vậy được gọi là các hàm số học chuẩn. Mỗi hàm chuẩn có tên chuẩn riêng. Đối số của hàm là một hay nhiều biểu thức số học và được đặt trong cặp ngoặc đơn (và) sau tên hàm. Bản thân hàm chuẩn cũng được coi là một biểu thức số học và nó có thể tham gia vào biểu thức số học như một toán hạng (giống như biến và hằng). Kết quả của hàm có thể là nguyên hoặc thực hay phụ thuộc vào kiểu của đối số.

Bảng dưới đây cho biết một số hàm chuẩn thường dùng.

Hàm	Biểu diễn toán học	Biểu diễn C++	Kiểu đối số	Kiểu kết quả
lũy thừa	x^y	**pow**(x,y)	Thực	Theo kiểu đối số
căn bậc 2	$\sqrt{x}$	**sqrt**(x)	Thực hoặc nguyên	Thực
giá trị tuyệt đối	$\|x\|$	**abs**(x)	Thực hoặc nguyên	Theo kiểu đối số
logarit cơ số tự nhiên	lnx	**log**(x)	Thực	Theo kiểu đối số
logarit cơ số 10	$log_{10}x$	**log10**(x)	Thực	Theo kiểu đối số
lũy thừa cơ số e	e^x	**exp**(x)	Thực	Theo kiểu đối số
hàm sin	$sinx$	**sin**(x)	Thực	Theo kiểu đối số
hàm cos	$cosx$	**cos**(x)	Thực	Theo kiểu đối số
hàm tang	tg	**tan**(x)	Thực	Theo kiểu đối số

Để dùng các hàm toán học phải khai báo bao gồm thư viện *cmath*.

Ví dụ:

Biểu thức toán học $\frac{-b+\sqrt{b^2-4ac}}{2a}$ trong C++ có thể viết dưới dạng:

```
(-b + sqrt(b*b - 4*a*c)/(2*a)
```

Hoặc

```
(-b + sqrt(b*b - 4*a*c)/ 2/a
```

Ngoài những hàm số học chuẩn trên, còn có các hàm chuẩn khác được giới thiệu trong những phần sau.

4. Biểu thức quan hệ

Hai biểu thức cùng kiểu liên kết với nhau bởi phép toán quan hệ cho ta một biểu thức quan hệ.

Biểu thức quan hệ có dạng:

```
<biểu thức 1> <phép toán quan hệ> <biểu thức 2>
```

trong đó, *biểu thức 1* và *biểu thức 2* cùng là xâu hoặc cùng là biểu thức số học.

Ví dụ

```
x < 5
i+1 >= 2*j
```

Biểu thức quan hệ được thực hiện theo trình tự:

- Tính giá trị các biểu thức.
- Thực hiện phép toán quan hệ.

Kết quả của biểu thức quan hệ là giá trị lôgic: *true* (đúng) hoặc *false* (sai).

Trong ví dụ trên, nếu *x* có giá trị 3, thì biểu thức *x < 5* có giá trị *true*. Nếu *i* có giá trị 2 và *j* có giá trị 3 thì biểu thức *i + 1 >= 2*j* sẽ cho giá trị *false*.

Ví dụ

Điều kiện để điểm *M* có toạ độ (*x; y*) thuộc hình tròn tâm *I(a; b)*, bán kính *R* là:

```
sqrt((x-a)*(x-a) + (y-b)*(y-b)) <= R
```

hoặc

```
(x-a)*(x-a) + (y-b)*(y-b) <= R*R
```

5. Biểu thức lôgic

Biểu thức lôgic đơn giản là biến lôgic hoặc hằng lôgic.

Biểu thức lôgic gồm các biểu thức lôgic đơn giản, các biểu thức quan hệ liên kết với nhau bởi phép toán lôgic. Giá trị biểu thức lôgic là *true* hoặc *false*. Các biểu thức quan hệ thường được đặt trong cặp ngoặc đơn (và).

Dấu phép toán ! được viết trước biểu thức cần phủ định, ví dụ:

`!(x < 1)` thể hiện phát biểu "*x không nhỏ hơn 1*" và điều này tương đương với biểu thức quan hệ *x* >= 1.

Các phép toán `&&` (AND) và `||` (OR) dùng để kết hợp nhiều biểu thức lôgic hoặc quan hệ, thành một biểu thức thường được dùng để diễn tả các điều kiện phức tạp.

Ví dụ 1

Để thể hiện điều kiện $5 \le x \le 11$, trong C++ có thể tách thành phát biểu dưới dạng "$5 \le x$ và $x \le 11$" và được viết như sau:

```
(5 <= x) && (x <= 11)
```

Ví dụ 2

Giả thiết *M* và *N* là hai biến nguyên. Điều kiện xác định *M* và *N* đồng thời chia hết cho 3 hoặc đồng thời không chia hết cho 3 được thể hiện trong C++ như sau:

```
(m % 3 == 0 && n % 3 == 0) || (m % 3 != 0 && n % 3 != 0)
```

6. Câu lệnh gán

Lệnh gán là một trong những lệnh cơ bản nhất của các ngôn ngữ lập trình.

Trong C++ câu lệnh gán có dạng sau (đừng nhầm với phép toán so sánh bằng ==):

```
<tên biến> = <biểu thức>
```

Trong trường hợp đơn giản, tên biến là tên của biến đơn. Kiểu của biến sẽ bị thay đổi theo giá trị của biểu thức.

Chức năng của lệnh gán là đặt cho biến có tên ở vế trái dấu "=" giá trị mới bằng giá trị của biểu thức ở vế phải, đồng thời cũng làm cho biến nhận kiểu giá trị mới.

Ví dụ

```
x1 = (-b - sqrt(b*b - 4*a*c))/(2*a);
x2 = -b/a - x1;
z = z - 1;
i = i + 1;
```

Trong ví dụ trên, ý nghĩa của lệnh gán thứ ba là giảm giá trị của biến *z* một đơn vị. Ý nghĩa của lệnh gán thứ tư là tăng giá trị của biến *i* lên một đơn vị.

Ngoài ra, C++ còn hỗ trợ một số cách viết phép gán như sau:

Biểu thức gán trong C++	**Ý nghĩa**
`x += 2`	Tăng *x* lên 2 đơn vị
`x -= 2`	Giảm *x* đi 2 đơn vị
`x *= 2`	Tăng *x* lên 2 lần
`x /= 2`	Giảm *x* đi 2 lần
`x %= 2`	Thay *x* bằng phần dư của *x* chia 2
`x = y = 3`	Gán cho cả *x* và *y* giá trị 3

Bài 7. CÁC HÀM CHUẨN VÀO/RA ĐƠN GIẢN

Để khởi tạo giá trị ban đầu cho biến, ta có thể dùng lệnh gán để gán một giá trị cho biến. Như vậy, mỗi chương trình luôn làm việc với một bộ dữ liệu vào. Để chương trình có thể làm việc với nhiều bộ dữ liệu vào khác nhau, thư viện của các ngôn ngữ lập trình cung cấp một số chương trình dùng để đưa dữ liệu vào và đưa dữ liệu ra.

Những chương trình đưa dữ liệu vào cho phép đưa dữ liệu từ bàn phím hoặc từ đĩa vào và gán cho các biến, làm cho chương trình trở nên linh hoạt, có thể tính toán với nhiều bộ dữ liệu đầu vào khác nhau. Kết quả tính toán được lưu trữ tạm thời trong bộ nhớ. Những chương trình đưa dữ liệu ra dùng để đưa các kết quả này ra màn hình, in ra giấy hoặc lưu trên đĩa.

Các chương trình đưa dữ liệu vào và ra đó được gọi chung là các *hàm chuẩn vào/ra đơn giản*.

Trong phần này, ta sẽ xét các *hàm chuẩn vào/ra đơn giản* của C++ để nhập dữ liệu vào từ bàn phím và đưa thông tin ra màn hình.

Trước hết phần đầu của khai báo chương trình phải có khai báo sử dụng thư viện vào/ra và khai báo sử dụng tên miền chuẩn *std*. Các lệnh này luôn xuất hiện trong mọi chương trình:

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

1. Nhập dữ liệu vào từ bàn phím

Việc nhập dữ liệu từ bàn phím được thực hiện bằng lệnh *cin* :

```
cin >> biến1 >>  biến2 >>...>> biếnN;
```

trong đó *biến1, biến2, ..., biếnN* là một hoặc nhiều tên biến đơn, mỗi biến đơn được viết theo sau một ký hiệu >>.

Ví dụ:

```
cin >> n;
cin >> a >> b >> c;
```

Lệnh thứ nhất để nhập một giá trị từ bàn phím và gán giá trị đó cho biến *n*. Lệnh thứ hai dùng để nhập lần lượt ba giá trị từ bàn phím và gán các giá trị đó tương ứng cho ba biến *a, b* và *c*.

Khi nhập giá trị cho nhiều biến, những giá trị này được gõ cách nhau bởi ít nhất một dấu cách hoặc nhóm kí tự xuống dòng (gõ phím Enter). Các giá trị ứng với biến nguyên phải được biểu diễn dưới dạng nguyên (không có dấu chấm thập phân). Các giá trị ứng với biến thực có thể gõ dưới dạng số nguyên, số thực dạng thông thường hoặc số thực dạng dấu phẩy động.

Ví dụ, để nhập các giá trị 1, -5 và 6 cho các biến thực *a, b, c* trong lệnh nhập thứ hai trong ví dụ trên, có thể gõ:

```
1 -5 6 ↵
```

hoặc

```
1.0 -5 ↵
6 ↵
```

2. Đưa dữ liệu ra màn hình

Để đưa dữ liệu ra màn hình, C++ cung cấp đối tượng chuẩn *cout*:

```
cout << kq1 << kq2 << ... << kqN << endl;
```

trong đó *kq1, kq2, ..., kqN* có thể là tên các biến đơn, biểu thức hoặc hằng. Các hằng xâu thường được dùng để tách các kết quả hoặc đưa ra chú thích. Mỗi thành phần trong kết quả ra được viết theo sau ký hiệu **<<**.

Quá trình in kết quả ra màn hình được thực hiện thứ tự từ trái sang phải, gặp từ ***endl*** thì con trỏ sẽ di chuyển xuống dòng tiếp theo (để cho dễ nhớ *endl* có thể hiểu là *end line* - kết thúc dòng).

Ví dụ 1

Để nhập giá trị nguyên cho biến *M* từ bàn phím, người ta thường dùng lệnh:

```
cout << "Hãy nhập giá trị M: ";
cin >> M;
```

Khi thực hiện các lệnh này, trên màn hình xuất hiện dòng thông báo:

```
Hay nhap gia tri M:
```

và con trỏ sẽ ở vị trí tiếp theo trên dòng, chờ ta gõ giá trị của *M*.

Để chương trình được sử dụng một cách tiện lợi, khi nhập giá trị từ bàn phím cho biến, ta nên có thêm xâu kí tự nhắc nhở việc nhập giá trị cho biến nào, kiểu dữ liệu gì,... Ví dụ, khi cần nhập một số nguyên dương *N* ($N \leq 100$) từ bàn phím, ta có thể sử dụng cặp lệnh sau:

```
cout << "Nhap so nguyen duong N <= 100: ";
cin << N;
```

Ví dụ 2

Sau đây là một chương trình hoàn chỉnh có sử dụng các hàm vào/ra chuẩn của C++.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     cout << "Lop ban co bao nhieu nguoi? ";
6.     int N;
7.     cin >> N;
8.     cout <<"Vay la ban co " << N-1 <<" nguoi ban trong lop." << endl;
9.     cout <<"Go ENTER de ket thuc chuong trinh.";
10.    return 0;
11.}
```

Ghi nhớ:

- Để có thể sử dụng được các lệnh *cin, cout* cho việc nhập/xuất dữ liệu, phần khái báo của chương trình phải có khai báo sử dụng thư viện vào/ra *iostream* và khai báo sử dụng tên miền chuẩn *std.*

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

- Khi thực hiện lệnh *cout*, sau mỗi kết quả ra (biến, hằng, biểu thức) có thể có quy ước định dạng đầu ra. Cụ thể:

+ Với kết quả thực, lệnh *cout* cần có thêm các thông số định dạng đi kèm với cú pháp như sau:

```
cout << fixed << setw(n) << setprecision(p) << <giá trị thực> <<...
```

Giải thích: Thông số *fixed* ấn định giá trị thực được in ra ở dạng dấu chấm tĩnh, *setw(n)* ấn định độ rộng của giá trị in ra trên màn hình là *n*, *setprecision(p)* ấn định giá trị được in ra có *p* chữ số thập phân.

+ Đối với các kết quả khác (kiểu nguyên, kiểu xâu) thì chỉ cần thêm thông số *setw(n)* vào lệnh *cout* để ấn định số chỗ trên màn hình cho giá trị sắp in ra là *n*.

- Để sử dụng được các hàm *setw()* và *setprecision()* ta cần khai báo sử dụng thêm thư viện:

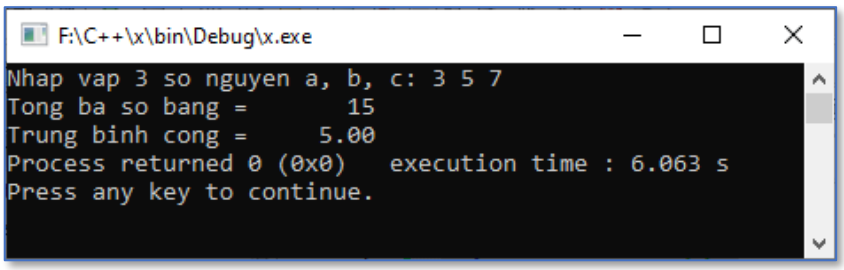
```
#include <iomanip>
```

Ví dụ 3

Nhập vào từ bàn phím giá trị 3 số nguyên *a, b, c* và in ra màn hình giá trị tổng và trung bình cộng của 3 số đó. Giá trị tổng phải được in với độ rộng 8, trung bình cộng phải được in với độ rộng 8 và có 2 chữ số thập phân.

```
1. #include <iostream>
2. #include <iomanip>
3. using namespace std;
4. int main()
5. {
6.     int a, b, c;
7.     cout << "Nhap vap 3 so nguyen a, b, c: ";
8.     cin >> a >> b >> c;
9.     int tong = a + b + c;
10.    double tb = tong / 3;
11.    cout <<"Tong ba so bang = " << setw(8)<< tong << endl;
12.    cout <<"Trung binh cong = " << fixed << setw(8) << setprecision(2)<< tb;
13.    return 0;
14.}
```

Chạy chương trình, xuất hiện màn hình nhập dữ liệu. Nhập vào 3 giá trị: 3 5 7, chương trình sẽ in ra màn hình kết quả sau:



Hình 2.1 - Minh họa kết quả chạy thử chương trình ví dụ 3 trong Code::Blocks

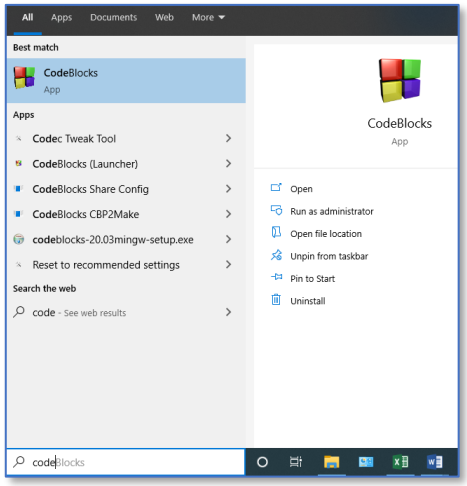
Bài 8. SOẠN THẢO, BIÊN DỊCH, THỰC HIỆN VÀ HIỆU CHỈNH CHƯƠNG TRÌNH.

Để có thể thực hiện chương trình được viết bằng một ngôn ngữ lập trình, ta cần soạn thảo chương trình, sử dụng chương trình dịch để dịch chương trình sang ngôn ngữ máy. Các hệ thống lập trình cụ thể thường cung cấp phần mềm phục vụ cho việc soạn thảo, dịch và hiệu chỉnh chương trình.

Với ngôn ngữ C++, bạn nên cài đặt phần mềm IDE Code::Blocks kèm bộ chương trình dịch GCC phổ biên hiện nay có tên là MinGW trên máy tính, Code::Blocks là bộ công cụ lập trình đơn giản và dễ sử dụng. Địa chỉ tải phần mềm: http://www.codeblocks.org/downloads, nhớ chọn đúng phiên bản có tên dạng: *codeblocks-xx.yymingw-setup.exe*, trong đó *xx.yy* là tên phiên bản có thể thay đổi theo thời gian và nhớ là chọn bản dành cho HĐH Windows nhé!

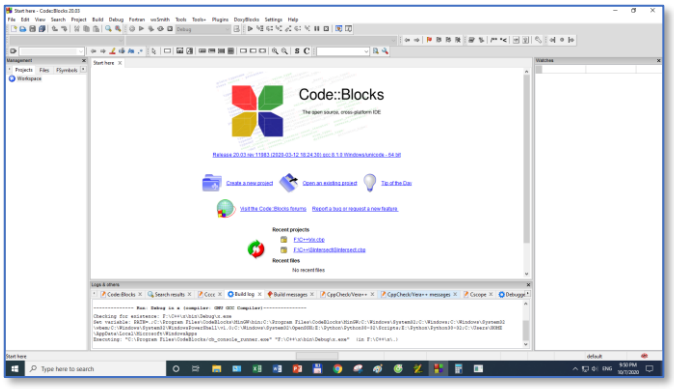
Từ đây cuốn sách này chỉ giới thiệu cách làm việc với IDE Code::Blocks.

Sau khi chắc chắn đã cài đặt IDE Code::Blocks kèm MinGW trên máy tính cài HĐH Windows, bạn có thể tìm thấy phần mềm Code::Blocks trong menu *Start* hoặc dùng chức năng tìm kiếm của *Windows* (nút hình kính lúp cạnh nút *start*) và gõ cụm từ *code* là có thể thấy Code::Blocks



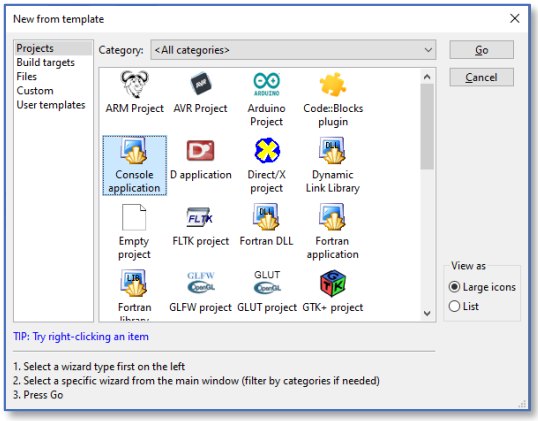
Hình 2.2 - Minh họa màn hình tìm kiếm công cụ IDE Code::Blocks trong Windows 10

Bấm đúp chuột vào biểu tượng, bạn sẽ thấy màn hình đầu tiên tương tự như sau:



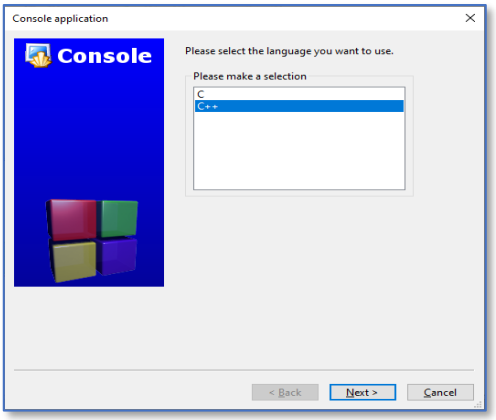
Hình 2.3 - Màn hình đầu tiên của Code::Blocks

Từ màn hình như hình 2.3, bạn bấm chuột vào biểu tượng *Create New Project* (hoặc chọn menu *File → New → Project...*), xuất hiện hộp thoại *New from template*.



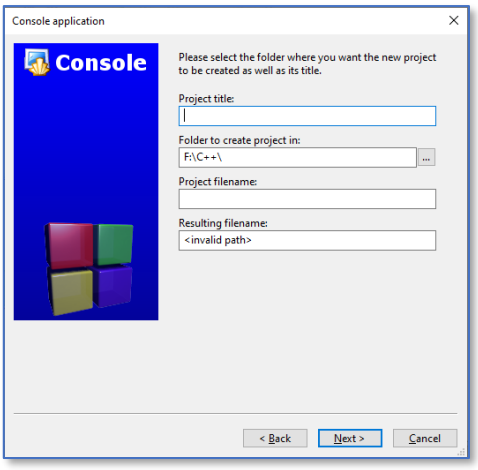
Hình 2.4 - Hộp thoại New from template

Bấm chuột vào biểu tượng *Console application*, và bấm nút *Go*, xuất hiện hộp thoại *Console application*.



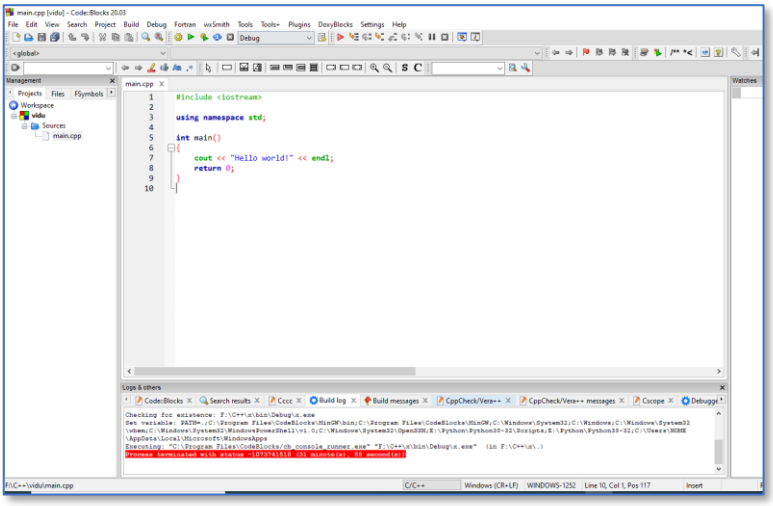
Hình 2.5 - Hộp thoại Console application

Bấm chọn dòng *C++*, và bấm nút *Next*, xuất hiện hộp thoại *Console application* như hình 2.6



Hình 2.6 - Hộp thoại Console application

Trong hộp *Project title* gõ vào tên chương trình (dự án), bấm nút ... (nút nhỏ nằm bên phải hộp văn bản *Folder to create project in*) để chọn thư mục lưu dự án trên đĩa và bấm nút *Next*, xuất hiện hộp thoại *Console application* kế tiếp thì bấm *Finish*. Xuất hiện cửa sổ chương trình như hình 2.7.



Hình 2.7 - Màn hình soạn thảo mã nguồn chương trình

Trong màn hình 2.7, phần bên trái là cửa sổ *management* quản lý các tệp của dự án, bạn sẽ thấy tên dự án mà bạn đã gõ vào như giai đoạn ở hình 2.6, nếu tệp mã nguồn *main.cpp* chưa được mở như hình 2.7 thì bạn bấm vào nút dấu + (cạnh biểu tượng *source*) ngay dưới tên dự án trong cửa sổ *management* và bấm đúp vào tệp *main.cpp* để mở nó.

Toàn bộ chương trình của bạn sẽ nằm trong tệp *main.cpp*. Theo mặc định, Code::Blocks đã tạo sẵn một chương trình mẫu như trong cửa sổ hình 2.7 (Chương trình *Hello World!* kinh điển).

1. Soạn thảo chương trình

Trong cửa sổ như hình 2.7, bạn có thể soạn thảo các lệnh của chương trình như cách soạn thảo văn bản thông dụng trong Windows và lưu chương trình vào đĩa bằng chức năng menu *File → Save* (phím tắt **CTRL + S**). Các chức năng soạn thảo như **Copy, Cut, Past, Find, Replace** có sẵn trong menu **Edit**.

2. Dịch và chạy chương trình

Sau khi soạn thảo xong chương trình, để dịch chương trình, bạn nhấn nút *Build* trên thanh công cụ (hoặc chọn menu *Build → Build*, phím tắt **CTRL + F9**)

Nếu có lỗi cú pháp, cửa sổ *Build log* phía bên dưới màn hình sẽ hiển thị và thông báo về lỗi.

Cần phải sửa hết lỗi cú pháp, khi hết lỗi thì chương trình sẽ thông báo dạng như sau:

```
Process terminated with status 0 (0 minute(s), 1 second(s)), 0
error(s), 0 warning(s) (0 minute(s), 1 second(s))
```

Trong cửa sổ *Build log*.

3. Chạy thử chương trình

Sau khi dự án chương trình đã được dịch thành công ta có thể chạy thử chương trình bằng cách bấm nút *Run* trên thanh công cụ (hoặc chọn menu *Build* → *Run*, phím tắt **CTRL + F10**).

Có thể thực hiện cả dịch và chạy chương trình bằng cách bấm nút *Build and run* trên thanh công cụ (hoặc chọn menu *Build* → *Build and run*, phím tắt **F9**).

4. Đóng dự án

Chọn menu *File* → *Close Project*.

5. Thoát khỏi môi trường Code::Blocks

Bấm nút đóng cửa sổ hoặc bấm **ALT + F4** hoặc chọn menu *File* → *Exit* hoặc bấm **CTRL + Q**.

TÓM TẮT

- Dữ liệu của bài toán được biểu diễn thông qua biến trong chương trình theo các quy tắc của ngôn ngữ lập trình cụ thể.
- Kiểu dữ liệu của mọi ngôn ngữ lập trình chỉ cho phép mô tả các đại lượng rời rạc và hữu hạn.
- Một chương trình thường có hai phần: Phần khai báo và phần thân chương trình. Phần khai báo có thể có hoặc không.
- Kiểu dữ liệu chuẩn: Kiểu nguyên, kiểu thực, kiểu chuỗi kí tự, kiểu lôgic.
- Các biến đều phải được khai báo và mỗi biến chỉ khai báo một lần.
- Các phép toán: số học, quan hệ và lôgic.
- Có ba loại biểu thức: số học, quan hệ và lôgic.
- Các ngôn ngữ lập trình có:
- Lệnh gán dùng để gán giá trị của biểu thức cho biến.
- Các thủ tục chuẩn dùng để đưa dữ liệu vào và ra.

BÀI TẬP VÀ THỰC HÀNH 1

1. Mục đích, yêu cầu

- Giới thiệu một chương trình C++ hoàn chỉnh đơn giản;
- Làm quen với một số dịch vụ cơ bản của IDE Code::Blocks trong việc soạn thảo, lưu trữ, dịch và thực hiện chương trình.

2. Nội dung

a) Tạo dự án mới trong Code::Blocks với tên GPTB2 và gõ vào chương trình sau:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int a, b, c, d;
6.     cout<<"Nhap a, b, c: ";
7.     cin >> a >> b >> c;
8.     d = b*b - 4*a*c;
9.     double x1 = (-b - sqrt(d))/(2*a), x2 = -b/a - x1;
10.    cout << "x1 = "<<fixed << setw(10) << setprecision(2) << x1 << endl;
11.    cout << "x2 = "<<fixed << setw(10) << setprecision(2) << x2 << endl;
12.    return 0;
13.}
```

Chú ý

- Mỗi câu lệnh được kết thúc bằng dấu chấm phảy (;) (các dòng *#include* thì không có)
- GCC cho phép khai báo *#include <bits/sdtc++.h>* sẽ cho phép ta dùng tất cả các hàm cần thiết mà không cần khai báo sử dụng thêm thư viện nào nữa.

b) Nhấn phím *CTRL + S* để lưu chương trình lên đĩa.

c) Nhấn tổ hợp phím *CTRL + F9* để dịch và sửa lỗi cú pháp (nếu có).

d) Khi hết lỗi tiếp tục nhấn phím *F9* để thực hiện chương trình. Nhập các giá trị 1; -3 và 2. Quan sát kết quả hiển thị trên màn hình *(x1 = 1.00 x2 = 2.00*). Bấm Enter trở về màn hình soạn thảo.

e) Nhấn *F9* rồi nhập các giá trị 1 0 -2. Quan sát kết quả trên màn hình (*x1 = -1.41 x2 = 1.41*).

f) Sửa lại chương trình trên sao cho không dùng biến trung gian *d*. Thực hiện chương trình đã sửa với các bộ dữ liệu trên.

g) Sửa lại chương trình nhận được ở mục c bằng cách thay đổi công thức tính *x2* (có hai cách để tính *x2*).

h) Thực hiện chương trình đã sửa với bộ dữ liệu 1; -5; 6. Quan sát kết quả trên màn hình (*x1 = 2 x2 = 3*).

i) Thực hiện chương trình với bộ dữ liệu 1; 1 ; 1 và quan sát kết quả trên mànhình.

Câu hỏi và bài tập

1. Hãy cho biết sự khác nhau giữa hằng có đặt tên và biến.

2. Tại sao phải dùng biến?

3. Trong C++, để một biến có kiểu nguyên thì phải làm thế nào?

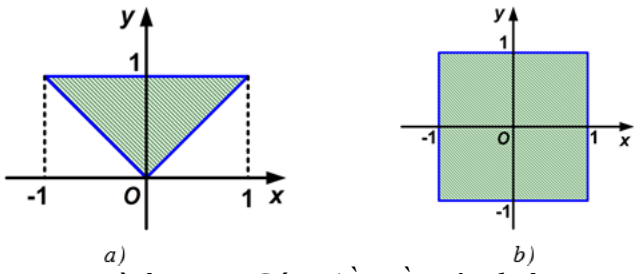
4. Hãy viết biểu thức toán học dưới đây trong C++:

$$(1+z)\frac{x+\frac{y}{z}}{a-\frac{1}{1+x^3}}$$

5. Hãy chuyển các biểu thức trong C++ dưới đây sang biểu thức toán học tương ứng:

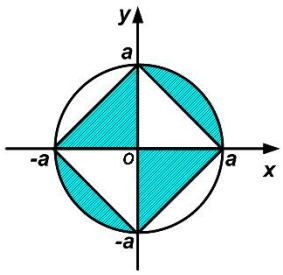
```
a) a/b*2;      b) a*b*c/2;      c) 1/a*b/c;    d) b/sqrt(a*a+b)
e) a**2 + 2*a*b + b**2          f) (x + y)**3  g) exp(5)
```

6. Hãy viết biểu thức lôgic cho kết quả *true* khi toạ độ (*x;y*) là điểm nằm trong vùng gạch chéo kể cả biên của các hình 2.11a và 2.11b.



Hình 2.11 - Các miền cần xác định

9. Hãy viết chương trình nhập số *a* (*a* > 0) rồi tính và đưa ra diện tích phần được gạch chéo trong hình 2.12 (kết quả làm tròn đến bốn chữ số thập phân).



Hình 2.12

10. Lập trình tính và đưa ra màn hình vận tốc *v* khi chạm đất của một vật rơi từ độ cao *h*, biết rằng $v = \sqrt{gh}$, trong đó *g* là gia tốc rơi tự do và *g* = 9,8m/s2. Độ cao *h* (m) được nhập vào từ bàn phím.

Chương 3. Cấu trúc rẽ nhánh và lặp

Bài 9. Cấu trúc rẽ nhánh

1. Rẽ nhánh

Có rất nhiều việc chỉ được thực hiện khi một điều kiện cụ thể nào đó được thoả mãn.

Ví dụ, Châu và Ngọc thường cùng nhau chuẩn bị các bài thực hành môn Tin học. Một lần Châu hẹn với Ngọc: "*Chiều mai nếu trời không mưa thì Châu sẽ đến nhà Ngọc*".

Một lần khác, Ngọc nói với Châu: "*Chiều mai nếu trời không mưa thì Ngọc sẽ đến nhà Châu, nếu mưa thì sẽ gọi điện cho Châu để trao đổi*".

Câu nói của Châu cho ta biết một việc làm cụ thể (*Châu đến nhà Ngọc*) sẽ được thực hiện nếu một điều kiện cụ thể (*trời không mưa*) thoả mãn. Ngoài ra không đề cập đến việc gì sẽ xảy ra nếu điều kiện đó không thoả mãn (trời mưa).

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề thiếu:

Nếu... thì…

Câu nói của Ngọc khẳng định một trong hai việc cụ thể (*Ngọc đến nhà Châu* hay *Ngọc gọi điện cho Châu*) chắc chắn sẽ xảy ra. Tuy nhiên, việc nào trong hai việc sẽ được thực hiện thì tuỳ thuộc vào điều kiện cụ thể (*trời không mưa*) thoả mãn hay không.

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề đủ:

Nếu… thì…, nếu không thì…

Từ đó có thể thấy, trong nhiều thuật toán, các thao tác tiếp theo sẽ phụ thuộc vào kết quả nhận được từ các bước trước đó.

Cấu trúc dùng để mô tả các mệnh đề có dạng như trên được gọi là *cấu trúc rẽ nhánh*.

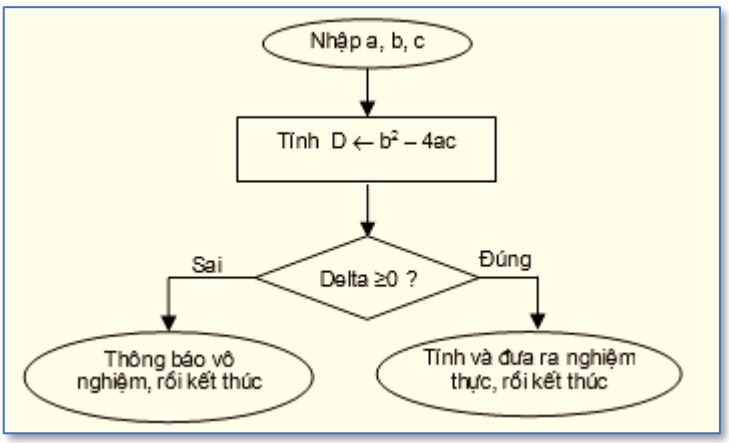
Ví dụ, để giải phương trình bậc hai:

ax2 + bx + c = 0, (a 0)

trước tiên ta tính biệt số delta $D = b^2 – 4ac$.

Nếu *D* không âm, ta sẽ đưa ra các nghiệm. Trong trường hợp ngược lại, ta phải thông báo là phương trình vô nghiệm.

Như vậy, sau khi tính *D*, tuỳ thuộc vào giá trị của *D*, một trong hai thao tác sẽ được thực hiện (hình 4). Mọi ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc rẽ nhánh.



Hình 3.1 - Sơ đồ thể hiện cấu trúc rẽ nhánh

2. Câu lệnh if

Để mô tả cấu trúc rẽ nhánh, C++ dùng câu lệnh ***if***. Tương ứng với hai dạng mệnh đề thiếu và đủ nói ở trên, C++ có hai dạng câu lệnh ***if***:

a) Dạng thiếu

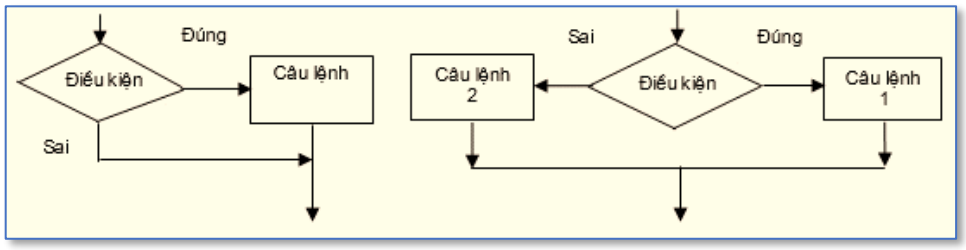
```
if (<điều kiện>) <câu lệnh >;
```

b) Dạng đủ

```
if (<điều kiện>)
     <câu lệnh 1>;
else
     <câu lệnh 2>
```

trong đó:

- *Điều kiện*: Biểu thức quan hệ hoặc lôgic có kết quả dạng `true / false` và luôn được đặt trong cặp ngoặc đơn (và).
- Câu lệnh, câu lệnh 1, câu lệnh 2 là một câu lệnh C++.



Hình 3.2 - Lưu đồ biểu diễn hai dạng lệnh if

- *Ở dạng thiếu*: *điều kiện* sẽ được tính và kiểm tra. Nếu *điều kiện* đúng (có giá trị *true*) thì câu lệnh sẽ được thực hiện, ngược lại thì *câu lệnh* sẽ bị bỏ qua (hình 3.2 trái).
- *Ở dạng đủ*: *điều kiện* cũng được tính và kiểm tra. Nếu *điều kiện* đúng thì *câu lệnh 1* sẽ được thực hiện, ngược lại thì *câu lệnh 2* sẽ được thực hiện (hình 3.2 phải).

Ví dụ 1.

```
1. if (Delta < 0)
2.     cout << "Phuong trinh vo nghiem.";
```

Ví dụ 2.

```
1. if (a % 3 == 0)
2.     cout <<"a chia het cho 3";
3. else
4.     cout <<"a khong chia het cho 3";
```

Ví dụ 3.

Để tìm số lớn nhất *max* trong hai số *a* và *b*, có thể thực hiện bằng hai cách sau:

- Dùng câu lệnh gán *max* = *a* và lệnh *if* dạng thiếu:

```
1. if (b > a)
2.     max = b;
```

- Dùng một lệnh *if* dạng đủ:

```
1. if (b > a)
2.     max = b;
3. else
4.     max = a;
```

3. Câu lệnh ghép

Theo cú pháp, sau một số mệnh đề điều khiển (như *if* và *else*) phải là một câu lệnh. Nhưng trong nhiều trường hợp, các thao tác sau những mệnh đề đó khá phức tạp, đòi hỏi không phải chỉ một mà là nhiều câu lệnh để mô tả. Trong các trường hợp như vậy, ngôn ngữ lập trình cho phép gộp một dãy câu lệnh thành một câu lệnh ghép (hay câu lệnh hợp thành). Sau một mệnh đề điều khiển nào đó của C++, câu lệnh ghép được viết trong cặp móc nhọn {}.

Câu lệnh, Câu lệnh 1, Câu lệnh 2 trong các cú pháp *if* ở trên có thể là câu lệnh ghép.

Thuật ngữ câu lệnh được hiểu chung cho câu lệnh đơn và câu lệnh ghép.

Ví dụ

```
1. if (D < 0)
2.     cout  <<"Phuong trinh vo nghiem.";
3. else
4. {
5.     x1 = (-b – sqrt(b*b – 4*a*c))/(2*a);
6.     x2 = -b/a-x1;
7. }
```

Trong ví dụ trên lệnh sau mệnh đề *else* là một lệnh ghép gồm hai lệnh đơn được viết trong cặp móc {}.

4. Một số ví dụ

Ví dụ 1.

Tìm nghiệm thực của phương trình bậc hai:

$$ax^2 + bx + c = 0 \text{ , với } a \neq 0.$$

Input: Các hệ số *a, b, c* nhập từ bàn phím.

Output: Đưa ra màn hình các nghiệm thực hoặc thông báo "*Phuong trinh vo nghiem*".

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int a, b, c;
6.     cout << "a, b, c: ";
7.     cin >> a >> b >> c;
8.     double d = b*b - 4*a*c;
9.     if (d < 0)
10.        cout <<"Phuong trinh vo nghiem";
11.    else
12.    {
13.        double x1 = (-b - sqrt(d))/(2*a);
14.        double x2 = -b/a - x1;
15.        cout <<"x1 = " << fixed << setw(8) <<setprecision(3) << x1;
16.        cout <<" x2 = " << fixed << setw(8) <<setprecision(3) << x2;
17.    }
18.    return 0;
19.}
```

Ví dụ 2.

Tìm số ngày của năm *N*, biết rằng năm nhuận là năm chia hết cho 400 hoặc chia hết cho 4 nhưng không chia hết cho 100. Ví dụ, các năm 2000, 2004 là năm nhuận và có số ngày là 366, các năm 1900, 1945 không phải là năm nhuận và có số ngày là 365.

Input: *N* nhập từ bàn phím.

Output: Đưa số ngày của năm *N* ra màn hình.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int n, sn = 365;
6.     cout << "Nam: "; cin >> n;
7.     if (n % 400 == 0 || (n % 4 == 0 && n % 100 != 0)) sn = 366;
8.     cout <<"So ngay cua nam "<< n << " la " << sn;
9.     return 0;
10.}
```

Bài 10. CẤU TRÚC LẶP

1. Lặp

Với *a* là số nguyên và *a* > 2, xét các bài toán sau đây:

Bài toán 1. Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + \cdots + \frac{1}{a+100}$$

Bài toán 2. Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + \cdots + \frac{1}{a+N} + \cdots$$

cho đến khi $\frac{1}{a+N} < 0.0001$

Với cả hai bài toán, dễ thấy cách để tính tổng *S* có nhiều điểm tương tự:

- Xuất phát, *S* được gán giá trị $\frac{1}{a}$
- Tiếp theo, cộng vào tổng *S* một giá trị $\frac{1}{a+N}$ với *N* = 1, 2, 3, 4, 5,… việc cộng này được lặp lại một số lần.

Đối với bài toán 1, số lần lặp là 100 và việc cộng vào tổng *S* sẽ kết thúc khi đã thực hiện việc cộng 100 lần.

Đối với bài toán 2, số lần lặp chưa biết trước nhưng việc cộng vào tổng *S* sẽ kết thúc khi điều kiện $\frac{1}{a+N} < 0.0001$ được thoả mãn.

Nói chung, trong một số thuật toán có những thao tác phải thực hiện lặp đi lặp lại một số lần. Một trong các đặc trưng của máy tính là có khả năng thực hiện hiệu quả các thao tác lặp. Cấu trúc lặp mô tả thao tác lặp và được phân biệt hai loại là *lặp với số lần biết trước* và *lặp với số lần chưa biết trước*.

Các ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc điều khiển lặp.

2. Lặp có số lần lặp biết trước và câu lệnh lặp `for`

Có hai thuật toán *Tong_1a* và *Tong_1b* để giải bài toán 1 như sau:

Thuật toán Tong_1a

```
Bước 1. S ← 1/a; N ← 0;  {Khởi tạo S và N}
Bước 2. N ← N + 1;
Bước 3. Nếu N > 100 thì chuyển đến bước 5;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2;
Bước 5. Đưa S ra màn hình, rồi kết thúc.
```

Thuật toán Tong_1b

```
Bước 1. S ← 1/a; N ← 101;    {Khởi tạo S và N}
Bước 2. N ← N – 1;
Bước 3. Nếu N < 1 thì chuyển đến bước 5;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2;
Bước 5. Đưa S ra màn hình rồi kết thúc.
```

Lưu ý, số lần lặp của cả hai thuật toán trên là biết trước và như nhau (100 lần).

Trong thuật toán *Tong_1a*, khi bắt đầu tham gia vòng lặp giá trị *N* là 1 và sau mỗi lần lặp *N* tăng lên 1 cho đến khi *N* > 100 (*N* = 101) thì kết thúc lặp (thực hiện đủ 100 lần). Trong thuật toán *Tong_1b*, khi bắt đầu tham gia vòng lặp giá trị *N* là 100 và sau mỗi lần lặp *N* giảm đi 1 cho đến khi *N* < 1 (*N* = 0) thì kết thúc lặp (thực hiện đủ 100 lần). Ta nói cách lặp trong thuật toán *Tong_1a* là ***dạng tiến*** và trong thuật toán *Tong_1b* là ***dạng lùi***.

Để mô tả cấu trúc lặp với số lần biết trước, C++ dùng câu lệnh ***for*** với cú pháp sau:

```
for ([khởi tạo]; [điều kiện lặp]; [bước nhảy]) [lệnh];
```

Giải thích:

- Bước *khởi tạo* được thực hiện đầu tiên và chỉ một lần. Bước này cho phép bạn khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp nào. Bạn không bắt buộc phải viết phần *khởi tạo*, nhưng dấu chấm phẩy phải có. Tiếp theo, *điều kiện lặp* được đánh giá. Nếu nó là *true*, *lệnh* được thực thi. Nếu nó *false*, *lệnh* được bỏ qua và luồng điều khiển sẽ nhảy sang câu lệnh tiếp theo ngay sau vòng lặp ***for***. Sau khi *lệnh* thực thi, luồng điều khiển sẽ nhảy đến *bước nhảy* và thực hiện lệnh trong phần này. Câu lệnh trong phần *bước nhảy* có thể được để trống, miễn là có dấu chấm phẩy sau *điều kiện lặp*. *Điều kiện lặp* bây giờ được đánh giá lại. Nếu là *true*, vòng lặp sẽ thực thi và lặp lại quá trình trên. Sau khi *điều kiện lặp* trở thành *false*, vòng lặp ***for*** kết thúc.

Ví dụ 1.

Sau đây là hai chương trình cài đặt các thuật toán *Tong_1a*

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int a, n;
6.     cout << "Hay nhap gia tri a vao! ";
7.     cin >> a;
8.     double s = 1.0/a;           // Buoc 1
9.     for(n = 1; n <= 100; n++)   // Buoc 2, Buoc 3
10.        s = s + 1.0/(a+n);      // Buoc 4
11.    cout <<"Tong S la: ";
12.    cout << fixed << setw(8) << setprecision(4) << s; // Buoc 5
13.    return 0;
14.}
```

và *Tong_1b*

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, n;
    cout << "Hay nhap gia tri a vao! ";
    cin >> a;
    double s = 1.0/a;           // Buoc 1
    for(n = 100; n > 0; n--)    // Buoc 2, Buoc 3
        s = s + 1.0/(a+n);      // Buoc 4
    cout <<"Tong S la: ";
    cout << fixed << setw(8) << setprecision(4) << s; // Buoc 5
    return 0;
}
```

Ví dụ 2.

Chương trình sau thực hiện việc nhập từ bàn phím hai số nguyên dương *M* và *N* (*M* < *N*), tính và đưa ra màn hình tổng các số chia hết cho 3 hoặc 5 trong phạm vi từ *M* đến *N*.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int m, n, t = 0;
    cout << "Nhap so M nho hon so N: ";
    cin >> m >> n;
    for (int i = m; i <= n; i++)
        if( i % 3 == 0 || i % 5 == 0)
            t += i;
    cout << t;
    return 0;
}
```

3. Lặp với số lần chưa biết trước và câu lệnh lặp `while`

Có thể xây dựng thuật toán *Tong_2* để giải bài toán 2, như sau:

Thuật toán Tong_2

```
Bước 1. S ← 1/a; N ← 0;  {Khởi tạo S và N}
Bước 2. Nếu 1/(a  + N) < 0,0001 thì chuyển đến bước 5;
Bước 3. N ← N + 1;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2.
Bước 5. Đưa S ra màn hình, rồi kết thúc
```

Như vậy, việc lặp với số lần chưa biết trước sẽ chỉ kết thúc khi một điều kiện cho trước được thoả mãn.

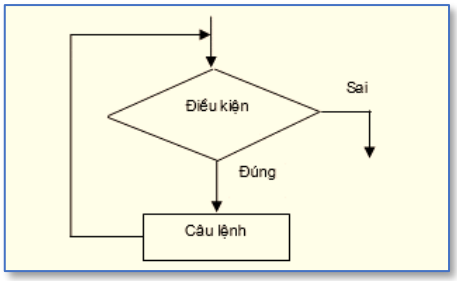
Để mô tả cấu trúc lặp như vậy, C++ dùng câu lệnh *while* có dạng:

```
while (<điều kiện lặp>) <câu lệnh>;
```

trong đó:

- *Điều kiện* là biểu thức quan hệ hoặc lôgic có giá trị `true/false`;
- *Câu lệnh* là một câu lệnh của C++.

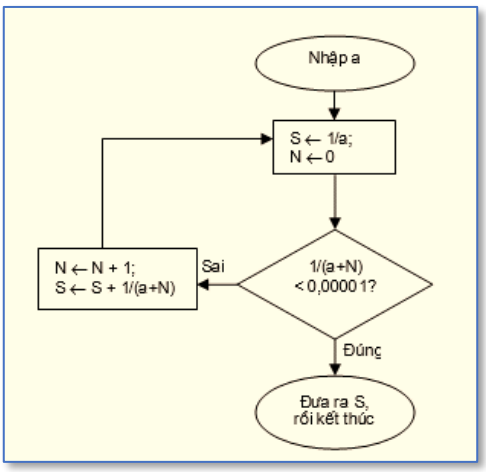
Việc thực hiện lệnh *while* được thể hiện trên sơ đồ ở hình 3.3.



Hình 3.3 - Sơ đồ lặp với số lần không biết trước

Ví dụ 1.

Sau đây là chương trình cài đặt thuật toán *Tong_2*.



Hình 3.4 - Sơ đồ khối của thuật toán Tong_2

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int a, n;
6.     cout << "Hay nhap gia tri a vao! ";
7.     cin >> a;
8.     n = 0;
9.     double s = 1.0/a;               // Buoc 1
10.    while(!(1.0/(a+n) < 0.0001)){   // Buoc 2
11.        n++;                        // Buoc 3
12.        s += 1.0/(a+n);             // Buoc 4
13.    }
14.    cout << fixed << setw(8) << setprecision(4) << s; // Buoc 5
15.    return 0;
16.}
```

Ví dụ 2.

Tìm ước chung lớn nhất (UCLN) của hai số nguyên dương *M* và *N*.

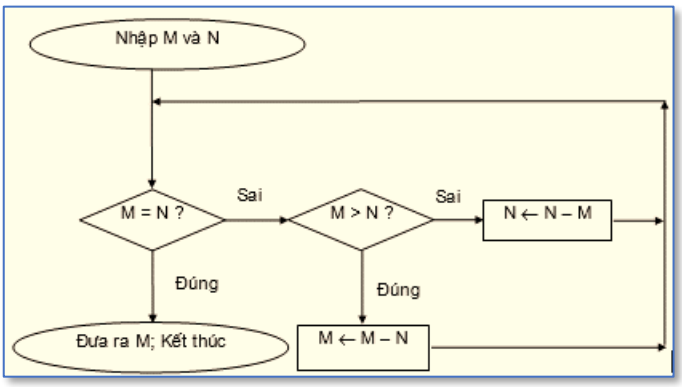
Có nhiều thuật toán khác nhau tìm UCLN của *M* và *N*. Sau đây là một thuật toán tìm UCLN.

Thuật toán

```
Bước 1.  Nhập M, N;
Bước 2.  Nếu M = N thì lấy M làm UCLN rồi chuyển đến bước5;
Bước 3.  Nếu M > N thì M ơ M - N rồi quay lại bước 2;
Bước 4.  N ơ N - M rồi quay lại bước 2;
Bước 5.  Đưa ra kết quả UCLN rồi kết thúc.
```

Chương trình sau thể hiện thuật toán tìm ước chung lớn nhất.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int m, n;
6.     cout << "M, N = ";
7.     cin >> m >> n;
8.     while(m != n){
9.         if (m > n)
10.            m -= n;
11.        else
12.            n -= m;
13.    }
14.    cout<<"USCLN = "<<m;
15.    return 0;
16. }
```



Hình 3.5 - Sơ đồ khối của thuật toán tìm ước chung lớn nhất

Chú ý

Các câu lệnh trong vòng lặp thường được lặp lại nhiều lần, vì vậy để tăng hiệu quả của chương trình thì những thao tác không cần lặp lại nên đưa ra ngoài vòng lặp.

TÓM TẮT

- Các ngôn ngữ lập trình đều có câu lệnh thể hiện cấu trúc rẽ nhánh và cấu trúc lặp.
- Câu lệnh rẽ nhánh có hai dạng:

  a) Dạng thiếu;

  b) Dạng đủ.
- Có thể gộp dãy câu lệnh thành câu lệnh ghép.

- Các câu lệnh mô tả cấu trúc lặp:

  a) Lặp với số lần biết trước;

  b) Lặp với số lần không biết trước.

Định lí Bohn Jacopini (Bon Ja-co-pi-ni): Mọi quá trình tính toán đều có thể thực hiện dựa trên ba cấu trúc cơ bản là cấu trúc tuần tự, cấu trúc rẽ nhánh và cấu trúc lặp.

BÀI TẬP VÀ THỰC HÀNH 2

1. Mục đích, yêu cầu

- Xây dựng chương trình có sử dụng cấu trúc rẽ nhánh;
- Làm quen với việc hiệu chỉnh chương trình.

2. Nội dung

Bài toán. Bộ số Pi-ta-go

Biết rằng bộ ba số nguyên dương *a, b, c* được gọi là bộ số Pi-ta-go nếu tổng các bình phương của hai số bằng bình phương của số còn lại. Viết chương trình nhập từ bàn phím ba số nguyên dương *a, b, c* và kiểm tra xem chúng có là bộ số Pi-ta-go hay không.

Ý tưởng: Kiểm tra xem có đẳng thức nào trong ba đẳng thức sau đây được thực hiện hay không:

$$a^2 = b^2 + c^2$$
$$b^2 = a^2 + c^2$$
$$c^2 = a^2 + b^2$$

Những công việc cần thực hiện:

a) Gõ chương trình sau:

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int a, b, c, a2, b2, c2;
    cout << "a, b, c: ";
    cin >> a >> b >> c;
    a2 = a, b2 = b, c2 = c;
    a2 *= a, b2 *= b, c2 *= c;
    if (a2 == b2 + c2 || b2 == a2 + c2 || c2 == a2 + b2)
        cout<< "Ba so da nhap la bo so Pi-ta-go" ;
    else
        cout<< "Ba so da nhap khong la bo so Pi-ta-go" ;
    return 0;
}
```

Chú ý:

a) Lưu chương trình với tên PITAGO lên đĩa.

b) Nhập các giá trị $a = 3$, $b = 4$, $c = 5$.

c) Lặp lại các bước trên với bộ dữ liệu $a = 700$, $b = 1000$, $c = 800$.

d) Nếu thay dãy lệnh

```
a2 = a;
b2 = b;
c2 = c;
a2 = a*a;
```

```
b2 = b*b;
c2 = c*c;
```

bằng dãy lệnh

```
a2 = a*a;
b2 = b*b;
c2 = c*c;
```

và chạy lại chương trình, thì kết quả có gì thay đổi với bộ dữ liệu cho ở câu c?

Câu hỏi và bài tập

1. Hãy cho biết sự giống và khác nhau của hai dạng câu lệnh ***if*** .

2. Câu lệnh ghép là gì? Tại sao phải có câu lệnh ghép?

3. Có thể dùng câu lệnh while để thay cho câu lệnh for được không? Nếu được, hãy thực hiện điều đó với chương trình *Tong_1a*.

4. Viết câu lệnh ***if*** tính:

a) $z = \begin{cases} x^2 + y^2 & nếu\ x^2 + y^2 \le 1 \\ x + y & nếu\ x^2 + y^2 > 1\ và\ y \ge x \\ 0.5 & nếu\ x^2 + y^2 > 1\ và\ y < x \end{cases}$

b) z = = $\begin{cases} |x| + |y| & nếu\ điểm\ (x, y) thuộc\ hình\ tròn\ bán\ kính\ r\ (r > 0), tâm\ (a; b) \\ x + y & trong\ trường\ hợp\ còn\ lại \end{cases}$

5. Lập trình tính:

a) $Y = \sum_{n=1}^{50} \frac{n}{n+1}$;

b) $e(n) = 1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!} + \cdots$, với n lần lượt bằng 3, 4, ..., cho đến khi $\frac{1}{n!} < 2 \times 10^{-6}$. Đưa các giá trị *e(n)* ra màn hình.

6. Lập trình giải bài toán cổ sau:

Vừa gà vừa chó.

Bó lại cho tròn.

Ba mươi sáu con,

Một trăm chân chẵn.

Hỏi bao nhiêu con mỗi loại?

7. Nhập từ bàn phím tuổi của cha và con (tuổi của cha hơn tuổi con ít nhất là25). Đưa ra màn hình bao nhiêu năm nữa thì tuổi cha gấp đôi tuổi con.

8. Một người gửi tiết kiệm không kì hạn với số tiền A đồng với lãi suất 0,2% mỗi tháng. Hỏi sau t tháng, người đó rút tiền thì sẽ nhận được số tiền là bao nhiêu. Biết rằng tiền gửi tiết kiệm không kì hạn không được tính lãi kép.

Chương 4. Kiểu dữ liệu có cấu trúc

Bài 11. KIỂU MẢNG

1. Kiểu mảng một chiều

Mảng một chiều là dãy hữu hạn các phần tử cùng kiểu. Mảng được đặt tên và mỗi phần tử của nó có một chỉ số. Để mô tả mảng một chiều cần xác định kiểu của các phần tử và cách đánh số các phần tử của nó.

Để người lập trình có thể xây dựng và sử dụng kiểu mảng một chiều, các ngôn ngữ lập trình có quy tắc cách thức cho phép xác định:

- Tên kiểu mảng một chiều;
- Số lượng phần tử;
- Kiểu dữ liệu của phần tử;
- Cách khai báo biến;
- Cách tham chiếu đến phần tử.

Ví dụ, xét bài toán nhập vào nhiệt độ (trung bình) của mỗi ngày trong tuần. Tính và đưa ra màn hình nhiệt độ trung bình của tuần và số lượng ngày trong tuần có nhiệt độ cao hơn nhiệt độ trung bình của tuần.

Ta có thể dùng bảy biến thực để lưu trữ nhiệt độ của các ngày trong tuần. Chương trình giải bài toán có thể được viết bằng C++ như sau:

```
#include <bits/stdc++.h>
using namespace std;

float t1, t2, t3, t4, t5, t6, t7, tb;
int dem;

int main(){
    cout << "Nhap vao nhiet do cua 7 ngay: ";
    cin >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7;
    tb = (t1 + t2 + t3 + t4 + t5 + t6 + t7)/7;
    dem = 0;
    if(t1 > tb) dem++;
    if(t2 > tb) dem++;
    if(t3 > tb) dem++;
    if(t4 > tb) dem++;
    if(t5 > tb) dem++;
    if(t6 > tb) dem++;
    if(t7 > tb) dem++;
    cout<< "Nhiet do trung binh tuan: " ;
    cout << fixed << setw(4) << setprecision(2)<<tb<<endl;
    cout<< "So ngay nhiet do cao hon trung binh: "<<dem;
    return 0;
}
```

Khi cần giải bài toán trên với *N* ngày (*N* khá lớn) thì cách làm tương tự không những đòi hỏi một khối lượng khai báo khá lớn, mà đoạn chương trình tính toán cũng khá dài.

Để giải quyết vấn đề đó, ta sử dụng kiểu dữ liệu mảng một chiều để mô tả dữ liệu. Chương trình giải bài toán tổng quát với *N* ngày trong C++ có thể như sau:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. const int Max = 366; // gia thiet N lon nhat la 366
4. typedef float Kmang1[Max];
5. Kmang1 nhietdo;
6. int dem, i, n;
7. float tong, trungbinh;
8.
9. int main(){
10.     cout << "Nhap so ngay: "; cin >> n;
11.     tong = 0;
12.     for(i = 0; i < n; i++)
13.     {
14.         cout<<"Nhap nhiet do ngay "<<i+1<<": ";
15.         cin >> nhietdo[i];
16.         tong += nhietdo[i];
17.     }
18.     dem = 0;
19.     trungbinh = tong/n;
20.     for(i = 0; i < n; i++)
21.         if(nhietdo[i] > trungbinh) dem++;
22.     cout<<"Nhiet do trung binh "<<n<<" ngay: ";
23.     cout<<fixed<<setw(8)<<setprecision(3)<<trungbinh<<endl;
24.     cout<<"So ngay nhiet do cao hon trung binh: "<< dem;
25.     return 0;
26. }
```

Chương trình này đã khai báo biến mảng một chiều (thông qua khai báo kiểu mảng) như sau:

`typedef float Kmang1[Max];`	Khai báo kiểu mảng một chiều gồm *Max* số thực với tên là *Kmang1*.
`Kmang1 nhietdo;`	Khai báo biến mảng *nhietdo* qua kiểu mảng.

a) Khai báo

Cú pháp khai báo biến mảng một chiều gồm hai cách sau:

- *Cách 1*. Khai báo trực tiếp biến mảng một chiều:

```
<kiểu phần tử> <tên biến mảng > [<kích thước>];
```

- *Cách 2*. Khai báo gián tiếp biến mảng qua kiểu mảng một chiều:

```
typedef  <kiểu phần tử>  <tên kiểu mảng>[<kích thước>];
<tên kiểu mảng> <tên biến mảng >;
```

trong đó:

- *Kích thước* thường là một số nguyên cho biết số phần tử của mảng, chỉ số của phần tử mảng trong C++ luôn được tính từ 0.
- *Kiểu phần tử* là kiểu của phần tử mảng.

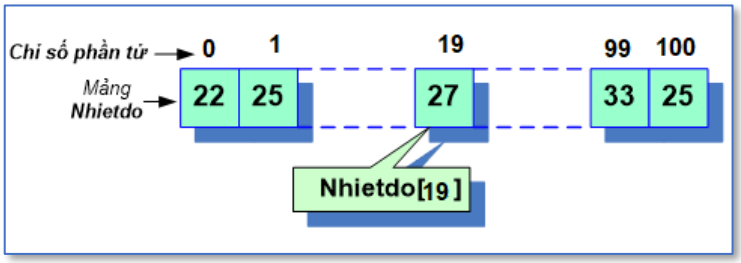
Ví dụ . Các khai báo kiểu mảng một chiều sau đây là hợp lệ:

```
typedef  float ArrayReal[300];
typedef  bool ArrayBool[2*n];
int A[100];
```

trong đó, *n* là hằng nguyên.

Tham chiếu tới phần tử của mảng một chiều được xác định bởi tên mảng cùng với chỉ số, được viết trong cặp ngoặc [và].

Ví dụ, tham chiếu tới nhiệt độ của ngày thứ 20, trong chương trình trên, được viết là *nhietdo[19]*. Chú ý, chỉ số mảng trong C++ được tính từ 0.



Hình 4.1 - Minh họa mảng một chiều

b) Một số ví dụ

Ta xét chương trình có sử dụng mảng một chiều cài đặt một số thuật toán giải những bài toán tìm kiếm và sắp xếp.

Ví dụ 1. Tìm phần tử lớn nhất của dãy số nguyên

Input: Số nguyên dương N ($N \leq 250$) và dãy N số nguyên dương $a_1, a_2, ..., a_N$, mỗi số đều không vượt quá 500.

Output: Chỉ số và giá trị của phần tử lớn nhất trong dãy số đã cho (nếu có nhiều phần tử lớn nhất chỉ cần đưa ra một trong số chúng).

Bước 1. Nhập N và dãy $a_1,..., a_N$;

Bước 2. Max $\leftarrow a_1$, $i \leftarrow 2$;

Bước 3. Nếu $i > N$ thì đưa ra giá trị Max rồi kết thúc;

Bước 4.

Bước 4.1. Nếu $a_i >$ Max thì Max $\leftarrow a_i$;

Bước 4.2. $i \leftarrow i + 1$ rồi quay lại bước 3;

Hình 4.2 - Thuật toán tìm phần tử lớn nhất của dãy số

Chương trình dưới đây thực hiện việc duyệt tuần tự các phần tử để tìm ra số lớn nhất của dãy số nguyên.

```
#include <bits/stdc++.h>
using namespace std;

const int Nmax = 250;
typedef float ArrInt[Nmax];
int N, i, Max, csmax;
ArrInt A;

int main(){
    cout <<"Nhap so luong phan tu cua day so, N = ";
    cin >> N;
    for (i = 0; i < N; i++)
    {
        cout<<"Phan tu thu "<< i+1 <<" = ";
        cin >> A[i];
    }
    Max = A[0]; csmax = 0;
    for (int i = 1; i < N; i++)
        if (A[i]> Max)
        {
            Max = A[i];
            csmax = i;
        }
    cout <<"Gia tri cua phan tu Max: "<< Max << endl;
    cout <<"Chi so cua phan tu Max: "<< csmax+1;
    return 0;
}
```

Ví dụ 2. Sắp xếp dãy số nguyên bằng thuật toán tráo đổi

Input: Số nguyên dương N ($N \leq 250$) và dãy A gồm N số nguyên dương a_1, a_2,..., a_N, mỗi số đều không vượt quá 500.

Output: Dãy số A đã được sắp xếp thành dãy không giảm.

Để giải bài toán này, ta sử dụng thuật toán tráo đổi như mô tả trong sách giáo khoa Tin học 10.

```
/* Chuong trinh giai bai toan sap xep day so */
#include <bits/stdc++.h>
using namespace std;

const int Nmax = 250;
typedef int ArrInt[Nmax];
int N, i, j, t;
ArrInt A;

int main()
{
    cout <<"Nhap so luong phan tu cua day so, N = "; cin >> N;
    for (i = 0; i < N; i++)
    {
        cout <<"Phan tu thu: "<< i+1<<" = ";
        cin >> A[i];
    }
    for (j = N-1; j > 0; j--)
    {
        for (i = 0 ; i < j; i++)
            if (A[i]> A[i+1])
            {
```

```
                //Trao doi A[i] va A[i+1]
                t = A[i];
                A[i] = A[i+1];
                A[i+1] = t;
            }
    }
    cout <<"Day so duoc sap xep la: "<<endl;
    for (i = 0 ; i < N; i++) cout<< setw(4)<< A[i];
    return 0;
}
```

Ví dụ 3. Tìm kiếm nhị phân

Input: Dãy *A* là dãy tăng gồm *N* ($N \leq 250$) số nguyên dương $a_1, a_2,..., a_N$ và số nguyên *k*.

Output: Chỉ số *i* mà $a_i = k$ hoặc thông báo "*Khong tim thay*" nếu không có số hạng nào của dãy *A* có giá trị bằng *k*.

Để giải bài toán này, chúng ta sẽ sử dụng thuật toán tìm kiếm nhị phân được trình bày trong sách giáo khoa Tin học 10.

Bước 1. Nhập *N*, các số hạng $a_1, a_2,..., a_N$ và khoá *k*;

Bước 2. *Dau* ← *1*, *Cuoi* ← *N*;

Bước 3. $Giua \leftarrow \left[\frac{Dau + Cuoi}{2}\right]$;

Bước 4. Nếu $a_{Giua} = k$ thì thông báo chỉ số *Giua*, rồi kết thúc;

Bước 5. Nếu $a_{Giua} > k$ thì đặt *Cuoi* = *Giua* – *1* rồi chuyển đến bước 7;

Bước 6. *Dau* ← *Giua* + *1*;

Bước 7. Nếu *Dau* > *Cuoi* thì thông báo dãy A không có số hạng có giá trị bằng *k*, rồi kết thúc;

Bước 8. Quay lại bước 3.

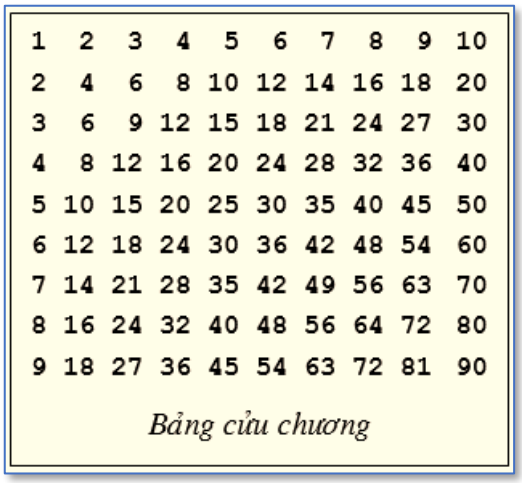
Hình 4.3 - Thuật toán tìm kiếm nhị phân

```
/* Chuong trinh cai dat thuat toan tim kiem nhi phan */
#include <bits/stdc++.h>
using namespace std;

const int Nmax = 250;
typedef int ArrInt[Nmax];
int N, i, k, dau, cuoi, giua;
ArrInt A;
bool tim_thay;

int main()
{
    cout <<"Nhap so luong phan tu cua day so, N = "; cin >> N;
    cout <<"Nhap cac phan tu cua day so tang: "<<endl;
    for (i = 0; i < N; i++)
    {
        cout <<"Phan tu thu: "<< i+1<<" = ";
        cin >> A[i];
    }
    cout <<"Nhap gia tri k = "; cin >> k;
    dau = 0; cuoi = N-1; tim_thay = false;
```

```
22.     while (dau <= cuoi && !tim_thay)
23.     {
24.         giua = (dau + cuoi) / 2;
25.         if (A[giua] == k)
26.              tim_thay = true;
27.         else
28.             if (A[giua] > k) cuoi = giua-1;
29.             else dau = giua + 1;
30.     }
31.     if (tim_thay) cout <<"Chi so tim duoc la: "<< giua+1;
32.     else cout <<"Khong tim thay";
33.     return 0;
34. }
```

2. Kiểu mảng hai chiều

Xét bài toán tính và đưa ra màn hình bảng cửu chương.



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90

Bảng cửu chương

Hình 4.4 - Minh họa bảng cửu chương

Ta thấy bảng cửu chương có dạng bảng gồm các giá trị cùng kiểu. Ta có thể biểu diễn bảng cửu chương bằng kiểu dữ liệu mảng hai chiều.

Mảng hai chiều là bảng các phần tử cùng kiểu.

Nhận xét rằng mỗi hàng của mảng hai chiều có cấu trúc như một mảng một chiều cùng kích thước. Nếu ta coi mỗi hàng của mảng hai chiều là một phần tử thì ta có thể nói mảng hai chiều là mảng một chiều mà mỗi phần tử là mảng một chiều.

Như vậy, ta cũng có thể mô tả dữ liệu của bảng cửu chương là kiểu mảng một chiều gồm 10 phần tử, mỗi phần tử lại là mảng một chiều có 9 phần tử, mỗi phần tử là một số nguyên.

Tương tự như với kiểu mảng một chiều, với kiểu mảng hai chiều, các ngôn ngữ lập trình cũng có các quy tắc, cách thức cho phép xác định:

- Tên kiểu mảng hai chiều;
- Số lượng phần tử của mỗi chiều;
- Kiểu dữ liệu của phần tử;

- Cách khai báo biến;
- Cách tham chiếu đến phần tử.

Ví dụ, biến mảng hai chiều B lưu trữ bảng cửu chương có thể được khai báo bằng C++ như sau:

```
int B[9][10];
```

a) Khai báo

Tổng quát, khai báo biến mảng hai chiều trong C++ như sau:

- Cách 1. Khai báo trực tiếp biến mảng hai chiều:

```
<kiểu phần tử> <tên biến mảng> [số dòng][số cột];
```

- Cách 2. Khai báo gián tiếp biến mảng qua kiểu mảng hai chiều:

```
typedef  <kiểu phần tử> <tên kiểu mảng> [số dòng][số cột];
<tên kiểu mảng>  <tên biến mảng>;
```

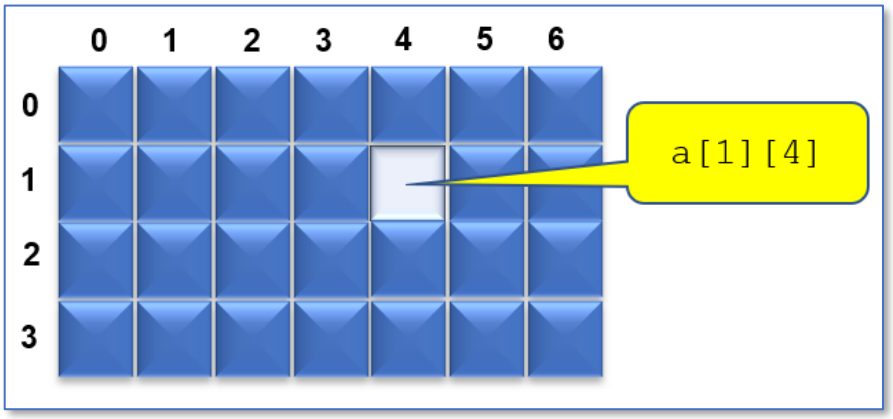
Ví dụ. Các khai báo sau đây là hợp lệ:

```
typedef  float ArrayReal[300][300];
typedef  bool ArrayBoolean[2*n][n];
int a[4][7];
long long ArrayLong[3*(n+1)][n];
```

trong đó, *n* là hằng nguyên.

Tham chiếu tới phần tử của mảng hai chiều được xác định bởi tên mảng cùng với hai chỉ số, mỗi chỉ số được viết trong cặp ngoặc [và]. Các chỉ số luôn được tính từ 0.

Ví dụ. Tham chiếu tới phần tử ở dòng thứ 2, cột thứ 5 của biến mảng *a* khai báo trong ví dụ trên được viết là: *a*[1][4].



Hình 4.5 - Minh họa mảng hai chiều

b) Một số ví dụ

Ví dụ 1. Chương trình sau tính và đưa ra màn hình bảng cửu chương.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. // B: bien mang hai chieu luu bang cuu chuong
```

```
int B[9][10];
int i, j;

int main(){
    for (i = 0; i < 9; i++)
        for (j = 0; j < 10; j++)
            B[i][j] = (i+1)*(j+1);
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 10; j++) cout << setw(4)<< B[i][j];
        cout << endl;
    }
    return 0;
}
```

Ví dụ 2. Chương trình sau nhập vào từ bàn phím các phần tử của mảng hai chiều *B* gồm 5 dòng, 7 cột với các phần tử là các số nguyên và số nguyên *k*. Sau đó, đưa ra màn hình các phần tử của mảng có giá trị nhỏ hơn *k*.

```
#include <bits/stdc++.h>
using namespace std;
int b[5][7];
int d, i, j, k;

int main(){
    cout <<"Nhap cac phan tu cua mang theo dong: "<<endl;
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 7; j++) cin >> b[i][j];
    }
    cout <<"Nhap vao gia tri k = "; cin >> k;
    d = 0;
    cout <<"DS cac phan tu mang nho hon "<< k <<": ";
    for (i = 0; i < 5; i++)
        for (j = 0; j < 7; j++)
            if (b[i][j] < k){
                cout<< b[i][j]<<" ";
                d++;
            }
   if (d == 0) cout << "Khong co phan tu nao nho hon "<< k;
   return 0;
}
```

Chú ý

- Các biến mảng thường gồm số lượng lớn các phần tử nên cần lưu ý phạm vi sử dụng chúng để khai báo kích thước và kiểu dữ liệu để tiết kiệm bộ nhớ.
- Ngoài hai kiểu mảng một chiều và hai chiều, còn có kiểu mảng nhiều chiều.

BÀI TẬP VÀ THỰC HÀNH 3

1. Mục đích, yêu cầu

- Nâng cao kĩ năng sử dụng một số câu lệnh và một số kiểu dữ liệu thông qua việc tìm hiểu, chạy thử các chương trình có sẵn;
- Biết giải một số bài toán tính toán, tìm kiếm đơn giản trên máy tính.

2. Nội dung

Bài 1.

Tạo mảng *A* gồm *n* (n ≤ 100) số nguyên, mỗi số có trị tuyệt đối không vượt quá 300. Tính tổng các phần tử của mảng là bội số của một số nguyên dương *k* cho trước.

a) Hãy tìm hiểu và chạy thử chương trình sau đây:

```
#include <bits/stdc++.h>
using namespace std;

const int nmax=100;
typedef int MyArray[nmax];
MyArray A;
int s, n, i, k;

int main()
{
    /* khoi dong bo tao so ngau nhien, lay thoi gian thuc lam hat giong*/
    srand(time(0));
    cout <<"Nhap n = "; cin >> n;
    /*Tao ngau nhien mang gom n so nguyen,
    ham rand() sinh ngau nhien so nguyen trong doan [0, 32767]
    */
    for (i = 0; i < n; i++) A[i] = rand() % 300 - rand()%300;
    for (i = 0; i < n; i++) cout <<setw(5)<< A[i]; //in ra mang vua tao
    cout << endl;
    cout <<"Nhap k = "; cin >> k;
    s = 0;
    for (i = 0; i < n; i++)
        if (A[i] % k == 0) s += A[i];
    cout <<"Tong can tinh la: " << s;
    return 0;
}
```

Chú ý:

Hàm *rand ()* sinh số nguyên ngẫu nhiên trong khoảng [0, 32767]. Còn hàm *srand(time(0))* được dùng để khởi động cơ chế tạo số ngẫu nhiên, dùng đồng hồ trên máy tính làm hạt giống để tạo.

b) Hãy đưa các câu lệnh sau đây vào những vị trí cần thiết nhằm sửa đổi chương trình trong câu a) để có được chương trình đưa ra số các số dương và số các số âm trong mảng.

```
int posi = 0;
int neg = 0;
if (A[i]> 0) posi++;
```

```
else if (A[i] < 0) neg++;
cout << posi<<' '<< neg;
```

Bài 2.

Viết chương trình tìm phần tử có giá trị lớn nhất của mảng và đưa ra màn hình chỉ số và giá trị của phần tử tìm được. Nếu có nhiều phần tử như vậy thì đưa ra phần tử có chỉ số nhỏ nhất.

a) Hãy tìm hiểu chương trình sau đây:

```
1. // Tim gia tri lon nhat trong day so
2. #include <bits/stdc++.h>
3. using namespace std;
4.
5. const int Nmax= 100;
6. typedef int MyArray[Nmax];
7. MyArray A;
8. int n, i, j;
9.
10.int main()
11.{
12.    cout <<"Nhap so luong phan tu cua day so, N = "; cin >> n;
13.    for (i = 0; i < n; i++)
14.    {
15.        cout <<"Phan tu thu " << i+1 <<" = ";
16.        cin >> A[i];
17.    }
18.    j = 0;
19.    for (i = 1; i < n; i++)
20.        if (A[i] > A[j]) j = i;
21.    cout <<"Chi so: "<< j+1 <<" Gia tri: " << setw(4) << A[j];
22.    return 0;
23.}
```

b) Chỉnh sửa chương trình trên để đưa ra chỉ số của các phần tử có cùng giá trị lớn nhất.

BÀI TẬP VÀ THỰC HÀNH 4

1. Mục đích, yêu cầu

- Biết nhận xét, phân tích đề xuất thuật toán giải bài toán sao cho chương trình chạy nhanh hơn;
- Làm quen với dữ liệu có cấu trúc và bài toán sắp xếp.

2. Nội dung

Bài 1.

a) Hãy tìm hiểu và chạy thử chương trình thực hiện thuật toán sắp xếp dãy số nguyên bằng tráo đổi với các giá trị khác nhau dưới đây. Qua đó, nhận xét về thời gian chạy của chương trình.

```
1. /* Chuong trinh giai bai toan sap xep day so */
2. #include <bits/stdc++.h>
3. using namespace std;
4.
5. const int Nmax = 250;
6. typedef int ArrInt[Nmax];
7. int n, i, j, y;
8. ArrInt A;
9.
10.int main()
11.{
12.    srand(time(0));
13.    cout << "Nhap n = "; cin >> n;
14.
15.    //Tao ngau nhien mang gom n so nguyen
16.    for (i = 0; i < n; i++) A[i] = rand() % 300 - rand() % 300;
17.
18.    for (i = 0; i < n; i++) cout << setw(5) << A[i]; //in mang vua tao
19.    cout << endl;
20.    for (j = n-1; j > 0; j--){
21.        for (i = 0; i < j; i++)
22.            if (A[i]> A[i+1]){
23.                //Trao doi A[i] va A[i+1] *)
24.                swap(A[i], A[i+1]);
25.            }
26.    }
27.    cout << "Day so duoc sap xep: " << endl;
28.    for (i = 0; i < n; i++) cout << setw(5) << A[i];
29.    return 0;
30.}
```

b) Khai báo thêm biến nguyên *Dem* và bổ sung vào chương trình những câu lệnh cần thiết để biến *Dem* tính số lần thực hiện tráo đổi trong thuật toán. Đưa kết quả tìm được ra màn hình.

Bài 2.

a) Hãy đọc và tìm hiểu những phân tích để viết chương trình giải bài toán:

Cho mảng *A* gồm *n* phần tử. Hãy viết chương trình tạo mảng *B[1..n]*, trong đó *B[i]* là tổng của *i* phần tử đầu tiên của *A* (Dãy *B* còn có tên là dãy tổng tiền tố của dãy *A*)

Thoạt đầu có thể viết chương trình sau để giải bài toán này:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. const int nmax = 100;
5. typedef int MyArray[nmax];
6. MyArray A, B;
7. int n;
8.
9. int main()
10.{
11.    srand(time(0));
12.    cout << "Nhap n = "; cin >> n;
13.
14.    //Tao ngau nhien mang gom n so nguyen
15.    for (int i = 0; i < n; i++) A[i] = rand() % 300 - rand() % 300;
16.
17.    for (int i = 0; i < n; i++) cout << setw(7) << A[i]; cout << endl;
18.    // Bat dau
19.    for (int i = 0; i < n; i++){
20.        B[i] = 0;
21.        for (int j = 0; j <= i; j++) B[i] += A[j];
22.    }
23.    // Ket thuc
24.    for (int i = 0; i < n; i++) cout << setw(7) << B[i]; cout << endl;
25.    return 0;
26.}
```

Để ý rằng ta có các hệ thức sau:

```
B[1]= A[1]
B[i]= B[i-1]+A[i], 1 < i ≤ n
```

Phân tích đó cho phép thay đoạn chương trình từ chỗ *// Bat dau* đến *//Ket thuc* bởi hai lệnh sau:

```
B[0] = A[0];
for (int i = 1; i < n; i++)   b[i] = b[i-1] + a[i];
```

Với hai lệnh này, máy chỉ phải thực hiện *n - 1* phép cộng, trong khi với đoạn chương trình trên máy phải thực hiện $\frac{n(n+1)}{2}$ phép cộng.

Nhờ việc phân tích ta có thể tiết kiệm được một lượng tính toán đáng kể.

Tuy tốc độ tính toán của máy tính nhanh nhưng có giới hạn. Do đó, trong khi viết chương trình, ta nên tìm cách viết sao cho chương trình thực hiện càng ít phép toán càng tốt.

Bài 12. KIỂU XÂU KÝ TỰ

Dữ liệu trong các bài toán không chỉ thuộc kiểu số mà cả kiểu phi số - dạng kítự. Dữ liệu kiểu xâu là dãy các kí tự.

Ví dụ. Các xâu kí tự đơn giản:

"Bach khoa" "KI SU" "2007 la nam Dinh Hoi"

Xâu (chuỗi) là dãy các kí tự trong bảng mã ASCII, mỗi kí tự được gọi là một phần tử của xâu. Số lượng kí tự trong một xâu được gọi là độ dài của xâu. Xâu có độ dài bằng 0 gọi là xâu rỗng.

Các ngôn ngữ lập trình đều có quy tắc, cách thức cho phép xác định:

- Tên kiểu xâu;
- Cách khai báo biến kiểu xâu;
- Số lượng kí tự của xâu;
- Các phép toán thao tác với xâu;
- Cách tham chiếu tới phần tử xâu.

Biểu thức gồm các toán hạng là biến xâu hoặc hằng xâu được gọi là biểu thức xâu. Với dữ liệu kiểu xâu có thể thực hiện phép toán ghép xâu và các phép toán quan hệ.

Có thể xem xâu là danh sách một chiều mà mỗi phần tử là một kí tự. Trong C++, các kí tự của xâu được đánh số thứ tự bắt đầu từ 0.

Tương tự như với mảng, việc tham chiếu tới phần tử của xâu được xác định bởi tên biến xâu và chỉ số đặt trong cặp ngoặc vuông [và].

Ví dụ, giả sử có biến *Hoten = 'Nguyen Le Huyen'* thì *Hoten[5]* cho ta chữ '*n*' là chữ thứ 6 trong xâu *Hoten*.

Dưới đây trình bày cách khai báo kiểu dữ liệu xâu, các thao tác xử lí xâu và một số ví dụ sử dụng kiểu xâu trong C++.

1. Khai báo

Biến kiểu xâu có thể khai báo như sau:

```
string <tên biến>;
```

Ví dụ

```
string Hoten;
```

Trong C++, độ dài xâu là giới hạn bộ nhớ và không cần khai báo độ dài xâu từ trước.

2. Các thao tác cơ bản trên xâu

a) Phép ghép xâu, kí hiệu là dấu cộng (+), được sử dụng để ghép nhiều xâu thành một. Có thể thực hiện phép ghép xâu đối với các hằng xâu và biến xâu.

Ví dụ

Phép ghép xâu:

```
"Ha"+ " Noi" +" - "+"Viet Nam"
```

cho xâu kết quả là *"Ha Noi - Viet Nam*".

b) Các phép so sánh bằng (=), khác (!=), nhỏ hơn (<), lớn hơn (>), nhỏ hơn hoặc bằng (<=), lớn hơn hoặc bằng (>=) có thứ tự ưu tiên thực hiện thấp hơn phép ghép xâu và thực hiện việc so sánh hai xâu theo các quy tắc sau:

- Xâu A là lớn hơn xâu B nếu như kí tự đầu tiên khác nhau giữa chúng kể từ trái sang trong xâu A có mã ASCII lớn hơn.
- Nếu A và B là các xâu có độ dài khác nhau và A là đoạn đầu của B thì A là nhỏ hơn B.

  Ví dụ

  ```
  "May tinh" < "May tinh cua toi"
  "12723" < "2345"
  ```

- Hai xâu được coi là bằng nhau nếu như chúng giống nhau hoàn toàn.

  Ví dụ

  ```
  "TIN HOC" == "TIN HOC"
  ```

Để xử lí các xâu có thể sử dụng các hàm thành viên của kiểu xâu dưới đây:

c) Hàm ***st.length()*** trả về độ dài (số ký tự) của xâu ***st***.

Giá trị `st`	Lệnh	Kết quả
`"abcdef"`	`st.length();`	`6`
`"Song Hong"`	`st.length();`	`9`

d) Hàm ***st.erase(****vt, n****)*** thực hiện việc xoá *n* kí tự của xâu ***st*** bắt đầu từ vị trí *vt*.

Giá trị `st`	Lệnh	Kết quả
`"abcdef"`	`st.erase(4, 2);`	`"abcd"`
`"Song Hong"`	`st.erase(st, 0, 5);`	`"Hong"`

e) Hàm ***st.insert(****s, vt****)*** thực hiện chèn xâu *s* vào xâu ***st*** bắt đầu từ vị trí *vt*.

Giá trị `st`	Giá trị `s`	Lệnh	Kết quả
`"IBM486"`	`" PC "`	`st.insert(s,3);`	`"IBM PC 486"`
`"Hinh .2"`	`"1"`	`st.insert(s,5);`	`"Hinh 1.2"`

f) Hàm ***st.substr (****vt, n)* thực hiện sao chép *n* ký tự của xâu ***st*** bắt đầu từ vị trí *vt*. Nếu thiếu tham số *n* thì mặc định là sao chép đến hết xâu *st*.

Giá trị `st`	Lệnh	Kết quả
`"Bai hoc thu 9"`	`st.substr(8,5);`	`"thu 9"`
`"abcdef"`	`st.substr(1);`	`"bcdef"`

g) Hàm ***st.find(****s, vt)* tìm kiếm vị trí xuất hiện lần đầu tiên của xâu *s* trong xâu ***st***, vị trí bắt đầu tìm là *vt*. Nếu thiếu tham số *vt* thì mặc định là tìm từ đầu xâu *st*. Nếu không tìm thấy thì hàm trả về giá trị hằng số trong C++ có tên `string::npos`.

Giá trị `st`	Lệnh	Kết quả
`"abcdef"`	`st.find("cd",1)`	`2`
`"abcdef"`	`st.find("cd",3)`	`string::npos`
`"abcdef"`	`st.find("k")`	`string::npos`

Ngoài các hàm thành viên của kiểu *string*, C++ còn cung cấp thư viện `<cctype>` có các hàm chuẩn trên kiểu dữ liệu ký tự (char):

Cú pháp	Ý nghĩa	Ví dụ	Kết quả
`toupper(c)`	Nếu *c* là chữ cái in thường thì trả về mã ASCII của ký tự in hoa tương ứng với ký tự *c*, ngược lại thì trả về mã ASCII của ký tự *c*	`x = toupper('a')`	`65` mã của chữ A
`tolower(c)`	Nếu *c* là chữ cái in hoa thì trả về mã ASCII của ký tự in thường tương ứng với ký tự *c*, ngược lại thì trả về mã ASCII của ký tự *c*	`x = tolower('N')`	`110` mã của chữ n

Ngoài ra còn có nhiều hàm khác, các bạn có thể tra cứu trên trang *http://www.cplusplus.com/*

3) Một số ví dụ

Ví dụ 1

Chương trình dưới đây nhập họ tên của hai người vào hai biến xâu và đưa ra màn hình xâu dài hơn, nếu bằng nhau thì đưa ra xâu nhập sau.

```
#include <bits/stdc++.h>
using namespace std;
string a, b;
int main(){
    cout<<"Nhap ho va ten nguoi thu nhat: "; getline(cin, a);
    cout<<"Nhap ho va ten nguoi thu hai: "; getline(cin, b);
    if (a.length() > b.length())
        cout << a;
    else
        cout << b;
    return 0;
}
```

Trong ví dụ 1, để nhập xâu họ tên ta phải dùng lệnh ***getline(cin**, a)* vì lệnh ***cin*** chỉ nhận được chuỗi ký tự không chứa dấu cách, mà xâu họ tên lại có thể chứa dấu cách. Nếu chắc chắn xâu dữ liệu không chứa dấu cách thì ta có thể sử dụng *cin* để nhập dữ liệu như các biến khác.

Ví dụ 2

Chương trình dưới đây nhập hai xâu từ bàn phím và kiểm tra kí tự đầu tiên của xâu thứ nhất có trùng với kí tự cuối cùng của xâu thứ hai không.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. string a, b;
4. int main(){
5.     cout<<"Nhap xau thu nhat: "; getline(cin, a);
6.     cout<<"Nhap xau thu hai: "; getline(cin, b);
7.     int n = b.length();
8.     // Xac dinh do dai xau b de biet vi tri cua ky tu cuoi cung
9.     if (a[0] == b[n-1])
10.        cout <<"Trung nhau";
11.    else
12.        cout << "Khac nhau";
13.    return 0;
14. }
```

Ví dụ 3

Chương trình sau nhập một xâu vào từ bàn phím và đưa ra màn hình xâu đó nhưng được viết theo thứ tự ngược lại.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. string a;
4. int main(){
5.     cout<<"Nhap xau: "; getline(cin, a);
6.     int n = a.length(); // Xac dinh do dai xau a
7.     for(int i = n-1; i >= 0; i--) cout <<a[i];
8.     return 0;
9. }
```

Ví dụ 4

Chương trình sau nhập một xâu vào từ bàn phím và đưa ra màn hình xâu thu được từ nó bởi việc loại bỏ các dấu cách.

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. string a, b;
5. int main(){
6.     cout << "Nhap xau: "; getline(cin, a);
7.     int k = a.length();
8.     b = ""; // Khoi tao xau rong
9.     for (int i = 0; i < k; i++)
10.        if (a[i] != ' ') b += a[i];
```

```
    cout <<"Ket qua: "<< b;
    return 0;
}
```

Ví dụ 5

Chương trình sau nhập vào từ bàn phím xâu kí tự s1, tạo xâu s2 gồm tất cả các chữ số có trong s1 (giữ nguyên thứ tự xuất hiện của chúng) và đưa kết quả ra màn hình.

```
#include <bits/stdc++.h>
using namespace std;
string s1, s2;
int main(){
    cout<<"Nhap xau s1: "; getline(cin, s1);
    s2 = "";    // Khoi tao xau rong
    for(int i = 0; i < s1.length(); i++)
        if ('0' <= s1[i] && s1[i] <= '9') s2 += s1[i];
    cout<<"Ketqua: " << s2;
    return 0;
}
```

BÀI TẬP VÀ THỰC HÀNH 5

1. Mục đích, yêu cầu

Làm quen với việc tìm kiếm, thay thế và biến đổi xâu.

2. Nội dung

Bài 1. Nhập vào từ bàn phím một xâu. Kiểm tra xâu đó có phải là xâu đối xứng, tức là đọc nó từ phải sang trái cũng như từ trái sang phải hay không (các xâu có tính chất này được gọi là palindrome).

a) Hãy chạy thử chương trình sau:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. string a, b;
4. int main(){
5.     cout<<"Nhap xau: "; getline(cin, a);
6.     int n = a.length(); // Xac dinh do dai xau a
7.     b = "";
8.     for(int i = n-1; i >= 0; i--) b += a[i];
9.     if (a == b)
10.       cout<<"Xau la palindrome";
11.    else
12.       cout<<"Xau khong la palindrome";
13.    return 0;
14. }
```

b) Hãy viết lại chương trình trên trong đó không cần có biến xâu *p*.

Bài 2. Viết chương trình nhập từ bàn phím một xâu kí tự *S* và thông báo ra màn hình số lần xuất hiện của mỗi chữ cái tiếng Anh trong *S* (không phân biệt chữ hoa hay chữ thường).

Bài 3. Nhập vào từ bàn phím một xâu. Thay thế tất cả các cụm kí tự "anh" bằng cụm kí tự "em".

Gợi ý bài 3:

Trong C++, kiểu xâu có hàm *replace()* cho phép thay một xâu bằng một xâu khác.

Ví dụ: Giả sử ta có hai xâu *s1, s2* như sau:

```
string s1 = "this is a test string.";
string s2 = "n example";
```

khi thực hiện lệnh :

```
s1.replace(9, 5, s2);
```

Kết quả xâu sẽ là:

```
s1 = "this is an example string."
```

Bài 13. KIỂU CẤU TRÚC (struct)

Dữ liệu kiểu *cấu trúc* (struct) dùng để mô tả các đối tượng có cùng một số thuộc tính mà các thuộc tính có thể có các kiểu dữ liệu khác nhau.

Ví dụ, bảng kết quả thi tốt nghiệp gồm thông tin về các thí sinh như họ và tên, ngày sinh, giới tính, điểm các môn thi,... mà những thông tin này thuộc các kiểu dữ liệu khác nhau.

Bảng kết quả thi tốt nghiệp

Họ và tên	Ngày sinh	Giới tính	Điểm Tin	Điểm Toán	Điểm Lí	Điểm Hoá	Điểm Văn	Điểm Sử	Điểm Địa
Nguyễn Thị Minh Huệ	12/12/1990	Nữ	9	10	7	8	8	7	8
Dương Trúc Lâm	2/1/1990	Nam	9	10	8	8	9	6	7
Đào Văn Bình	5/12/1990	Nam	8	8	9	8	7	7	6
...	...	...	...	...	...	...	...	...	...

Hình 4.6 - Minh họa về cấu trúc

Một ví dụ khác, khi xem xét doanh số của một cửa hàng, ta quan tâm đến tập hoá đơn bán hàng, mỗi hoá đơn đều có các thuộc tính như tên hàng, đơn giá, chủng loại, số lượng bán, giá thành, người bán, người mua, ngày bán,...

Để mô tả các đối tượng như vậy, ngôn ngữ lập trình C++ cho phép xác định kiểu dữ liệu ***cấu trúc*** (struct). Mỗi đối tượng được mô tả bằng một cấu trúc. Mỗi thuộc tính của đối tượng tương ứng với một thành viên của cấu trúc. Các thành viên khác nhau có thể có các kiểu dữ liệu khácnhau.

Ngôn ngữ lập trình đưa ra quy tắc, cách thức xác định:

- Tên kiểu cấu trúc;
- Tên các thuộc tính (thành viên);
- Kiểu dữ liệu của mỗi thành viên;
- Cách khai báo biến;
- Cách tham chiếu đến thành viên.

Dưới đây giới thiệu cách khai báo kiểu, biến, tham chiếu đến thành viên và phép gán giá trị cấu trúc trong C++.

1. Khai báo

Các thông tin cần khai báo bao gồm tên kiểu cấu trúc, tên các thuộc tính, kiểu dữ liệu của mỗi thuộc tính.

Do dữ liệu kiểu cấu trúc thường dùng để mô tả nhiều đối tượng nên ta thường định nghĩa một kiểu cấu trúc và sau đó dùng nó để khai báo các biến liên quan.

Kiểu cấu trúc thường được định nghĩa như sau:

```
struct <tên kiểu cấu trúc>{
     <kiểu thành viên 1> <thành viên 1>;
     . . .
     <kiểu thành viên k> <thành viên k>;
};
```

Sau khi có kiểu cấu trúc, biến kiểu cấu trúc có thể được khai báo như sau:

```
<tên kiểu cấu trúc>  <tên biến bản ghi>;
```

Ví dụ

Để xử lí bảng kết quả thi tốt nghiệp nêu trên ta có thể khai báo *Lop* là biến mảng một chiều, mỗi phần tử mảng là một cấu trúc *HocSinh* (dữ liệu về một học sinh). Mỗi cấu trúc *HocSinh* gồm các thông tin: *Hoten, NgaySinh, GioiTinh* và điểm 7 môn thi: *Tin, Toan, Li, Hoa, Van, Su, Dia*.

Hình 4.7 - Minh họa cấu trúc Hocsinh

Trong chương trình xử lí kết quả thi tốt nghiệp có thể sử dụng khai báo saudây:

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. const int Max =60; //gia thiet si so lop cao nhat la 60
5. struct HocSinh{
6.     string HoTen;
7.     string NgaySinh;
8.     bool GioiTinh;
9.     float Tin, Toan, Li, Hoa, Van, Su, Dia;
10.};
11.
12.HocSinh A, B;
13.HocSinh Lop[Max];
```

Nếu *A* là biến kiểu cấu trúc và *X* là tên một thuộc tính của *A*, thì tham chiếu đến thuộc tính *X*, được viết là: *A.X*

Để tham chiếu đến điểm tin học của một học sinh trong ví dụ trên ta viết: *A.Tin*

2. Gán giá trị

Có hai cách để gán giá trị cho biến bản ghi:

- *Dùng lệnh gán trực tiếp*: Nếu *A* và *B* là hai biến bản ghi cùng kiểu và thì ta có thể gán giá trị của *B* cho *A* bằng câu lệnh:

  `A = B;`

- *Gán giá trị cho từng thuộc tính*: Có thể thực hiện bằng lệnh gán hoặc nhập từ bàn phím.

Ví dụ, một lớp gồm N ($N \leq 60$) học sinh. Cần quản lí học sinh với các thuộc tính như họ và tên, ngày sinh, địa chỉ, điểm toán, điểm văn, xếp loại. Giả sử việc xếp loại được xác định như sau:

- Nếu tổng điểm toán và văn lớn hơn hoặc bằng 18 thì xếp loại A.
- Nếu tổng điểm toán và văn lớn hơn hoặc bằng 14 và nhỏ hơn 18 thì xếp loại B.
- Nếu tổng điểm toán và văn lớn hơn hoặc bằng 10 và nhỏ hơn 14 thì xếp loại C.
- Nếu tổng điểm toán và văn nhỏ hơn 10 thì xếp loại D.

Chú ý rằng, trong các thuộc tính cần quản lí, chỉ có năm thuộc tính đầu là độc lập, còn thuộc tính xếp loại được xác định dựa vào các điểm toán và văn. Để lưu trữ thông tin về học sinh, ta dùng kiểu bản ghi với sáu trường tương ứng với sáu thuộc tính cần quản lí.

Dưới đây là chương trình nhập vào từ bàn phím thông tin của từng học sinh trong lớp, thực hiện xếp loại và đưa ra màn hình kết quả xếp loại học sinh:

```
#include <bits/stdc++.h>
using namespace std;

const int Max =60; //gia thiet si so lop cao nhat la 60
struct HocSinh{
    string HoTen;
    string NgaySinh, DiaChi;
    float Toan, Van;
    char XepLoai;
};

HocSinh Lop[Max];
int n;

int main(){
    cout <<"So luong hoc sinh trong lop N = "; cin >> n;
    cin.ignore(255,'\n');
    for(int i = 1; i <= n; i++){
        cout <<"Nhap so lieu ve hoc sinh thu "<< i <<" : "<<endl;
        cout <<"Ho va ten: "; getline(cin, Lop[i].HoTen);
        cout <<"Ngaysinh: "; getline(cin,Lop[i].NgaySinh);
        cout <<"Dia chi: "; getline(cin,Lop[i].DiaChi);
        cout <<"Diem Toan: "; cin >> Lop[i].Toan;
        cout <<"Diem Van: "; cin >> Lop[i].Van;
        cin.ignore(255,'\n');
        if (Lop[i].Toan + Lop[i].Van >=18)
            Lop[i].XepLoai = 'A';
```

```
28.         else if (Lop[i].Toan + Lop[i].Van >= 14)
29.             Lop[i].XepLoai = 'B';
30.         else if (Lop[i].Toan+Lop[i].Van >= 10)
31.             Lop[i].XepLoai = 'C';
32.         else
33.             Lop[i].XepLoai = 'D';
34.     }
35.     cout <<"Danh sach xep loai hoc sinh trong lop:\n";
36.     for (int i = 1 ; i <= n; i++)
37.         cout<<setw(30) << Lop[i].HoTen <<" - Xep loai: " << Lop[i].XepLoai<<end
    l;
38.     return 0;
39. }
```

Chú ý: Dòng lệnh 17 và 25: *cin.ignore(255,'\n')* để bỏ qua ký tự Enter từ câu lệnh cin trước đó. Không có dòng lệnh này, lệnh *getline(cin, ...)* sẽ nhận vào ký tự Enter và bỏ qua lệnh nhập sau đó. Việc này chỉ xảy ra khi trước lệnh *getline(cin, ...)* có lệnh *cin*.

TÓM TẮT

- Kiểu dữ liệu có cấu trúc được xây dựng từ những kiểu dữ liệu đã có theo quy tắc, khuôn dạng do ngôn ngữ lập trình cung cấp.
- Mảng một chiều
- Mảng một chiều là dãy hữu hạn các phần tử cùng kiểu.
- Khai báo: tên mảng, kích thước, kiểu phần tử
- Tham chiếu phần tử mảng: tên biến mảng [chỉ số phần tử];
- Mảng hai chiều
- Mảng hai chiều là bảng các phần tử cùng kiểu.
- Khai báo: tên mảng, số dòng, số cột, kiểu phầntử.
- Tham chiếu phần tử mảng: tên biến mảng[chỉ số dòng] [chỉ số cột]
- Kiểu dữ liệu xâu
- Xâu là dãy các kí tự trong bảng mã ASCII.
- Các thao tác xử lí thường sử dụng:
- Phép ghép xâu;
- Phép so sánh;
- Các hàm thành viên của kiểu xâu.
- Kiểu cấu trúc
- Khai báo kiểu cấu trúc: tên cấu trúc, tên và kiểu các thuộc tính.

- Tham chiếu thuộc tính của cấu trúc: *tên biến cấu trúc.tên thuộc tính*

Câu hỏi và bài tập

1. Tại sao danh sách là kiểu dữ liệu có cấu trúc?

2. Có cần thiết phải khai báo kích thước của danh sách trước khi dùng không?

3. Các phần tử của danh sách có thể có những kiểu gì?

4. Tham chiếu đến phần tử của danh sách bằng cách nào?

5. Viết chương trình nhập từ bàn phím số nguyên dương N ($N \leq 100$) và dãy A gồm N số nguyên $A_1, A_2,..., A_N$, có giá trị tuyệt đối không lớn hơn 1000. Hãy cho biết dãy A có phải là một cấp số cộng hay không và thông báo kết quả ra mànhình.

6. Viết chương trình nhập từ bàn phím số nguyên dương N ($N \leq 100$) và dãy A gồm N số nguyên $A_1, A_2,..., A_N$, có trị tuyệt đối không lớn hơn 1000. Hãy đưa ra những thông tin sau:

a) Số lượng số chẵn và số lẻ trong dãy;

b) Số lượng số nguyên tố trong dãy;

7. Dãy F là dãy Fi-bô-na-xi nếu: *F0 = 1; F1 = 1; F2 = 2; FN = FN-1 + FN-2* với $N > 2$.

Viết chương trình nhập từ bàn phím số nguyên dương N và đưa ra màn hình số hạng thứ N của dãy Fi-bô-na-xi. Chương trình của em thực hiện được với giá trị lớn nhất của N là bao nhiêu?

8. Chương trình sau đây thực hiện những gì?

```
#include <bits/stdc++.h>
using namespace std;

const int NN = 50;
float a[NN][NN];
int n;

int main(){
    cout<<"Nhap N: "; cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 0; j < n; j++){
            cout<<"A["<<i<<","<<j<<"]= "; cin >> a[i][j];
        }
    }
    for(int i = 1; i <= n; i++){
        for(int j = 0; j < n; j++){
            float c = a[i][j];
            a[i][j] = a[n-i+1][j];
            a[n-i+1][j] = c;
        }
    }
    for(int i = 1; i <= n; i++){
        for(int j = 0; j < n; j++)
            cout<<fixed<<setw(5)<<setprecision(2) <<a[i][j]<<' ';
        cout<<endl;
    }
}
```

9. Cho mảng hai chiều kích thước $n \times n$ với các phần tử là những số nguyên. Tìm trong mỗi dòng phần tử lớn nhất rồi đổi chỗ nó với phần tử có chỉ số dòng bằng chỉ số cột.

Chương trình sau đây giải bài toán trên:

```
#include <bits/stdc++.h>
using namespace std;

const int NN = 15;
float a[NN][NN];
int n;

int main(){
    cout<<"Nhap N: "; cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cout<<"A["<<i<<","<<j<<"]= "; cin >> a[i][j];
        }
    }
    for(int i = 1; i <= n; i++){
        int Max = a[i][1], idx = 1;
        for(int j = 2; j <= n; j++){
            if(a[i][j] > Max){
                Max = a[i][j];
                idx = j;
            }
        }
        swap(a[i][i], a[i][idx]); // lenh swap() doi cho hai phan tu cho nhau
    }
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++)
            cout<<setw(10)<<a[i][j];
        cout<<endl;
    }
    return 0;
}
```

10. Viết chương trình nhập từ bàn phím xâu kí tự *S* có độ dài không quá 100. Hãy cho biết có bao nhiêu chữ số thập phân xuất hiện trong xâu *S*. Thông báo kết quả ra màn hình.

11. Hãy bổ sung thêm chương trình *Xep_loai* (ở bài 13) những lệnh cần thiết để chương trình đưa ra danh sách học sinh xếp loại A.

Chương 5. Tệp và thao tác với tệp

Bài 14. KIỂU DỮ LIỆU TỆP

1. Vai trò kiểu tệp

Tất cả các dữ liệu có các kiểu dữ liệu đã xét đều được lưu trữ ở bộ nhớ trong (RAM) và do đó dữ liệu sẽ bị mất khi tắt máy. Với một số bài toán, dữ liệu cần được lưu trữ để xử lí nhiều lần và với khối lượng lớn cần có kiểu dữ liệu tệp (file).

Kiểu dữ liệu tệp có những đặc điểm sau:

- Được lưu trữ lâu dài ở bộ nhớ ngoài (đĩa từ, CD,…) và không bị mất khi tắt nguồnđiện;
- Lượng thông tin lưu trữ trên tệp có thể rất lớn và chỉ phụ thuộc vào dung lượngđĩa.

2. Phân loại tệp và thao tác với tệp

Xét theo cách tổ chức dữ liệu, có thể phân tệp thành hai loại:

- *Tệp văn bản* là tệp mà dữ liệu được ghi dưới dạng các kí tự theo mã ASCII/UNICODE. Trong tệp văn bản, dãy kí tự kết thúc bởi nhóm kí tự xuống dòng hay kí tự kết thúc tệp.

Các dữ liệu dạng văn bản như sách, tài liệu, bài học, giáo án, các chương trình nguồn viết bằng ngôn ngữ bậc cao,... thường được lưu trữ dưới dạng tệp văn bản.

- *Tệp có cấu trúc* là tệp chứa dữ liệu được tổ chức theo một cách thức nhất định. Dữ liệu ảnh, âm thanh,... thường được lưu trữ dưới dạng tệp có cấu trúc.

Xét theo cách thức truy cập, có thể phân tệp thành hai loại:

- *Tệp truy cập tuần tự* cho phép truy cập đến một dữ liệu nào đó trong tệp chỉ bằng cách bắt đầu từ đầu tệp và đi qua lần lượt tất cả các dữ liệu trước nó.
- *Tệp truy cập trực tiếp* cho phép tham chiếu đến dữ liệu cần truy cập bằng cách xác định trực tiếp vị trí (thường là số hiệu) của dữ liệu đó.

Khác với mảng, số lượng phần tử của tệp không xác định trước.

Hai thao tác cơ bản đối với tệp là ghi dữ liệu vào tệp và đọc dữ liệu từ tệp.

Thao tác đọc/ghi với tệp được thực hiện với từng phần tử của tệp.

Để thao tác với kiểu dữ liệu tệp, người lập trình cần tìm hiểu cách thức NNLT cung cấp cách:

- Khai báo biến tệp;
- Mở tệp;
- Đọc/ghi dữ liệu;
- Đóng tệp.

Bài 15. KIỂU TỆP

Trong mục này ta chỉ xét cách khai báo, thao tác với tệp văn bản trong C++.

1. Khai báo

Trước khi làm việc với kiểu tệp, ta cần khai báo sử dụng thư viện <*fstream*>:

```
# include <fstream>
```

Tiếp theo là khai báo sử dụng biến tệp như sau:

+ Khai báo biến tệp để sau đó đọc dữ liệu

```
ifstream <tên biến tệp>;
```

+ Khai báo biến tệp để sau đó ghi dữ liệu

```
ofstream <tên biến tệp>;
```

2. Thao tác với tệp

a) Mở tệp

- Nếu đã khai báo biến tệp để đọc/ghi dữ liệu như ở trên thì ta có thể mở tệp như sau:

```
<tên biến tệp>.open(<xâu tên tệp>);
```

- Nếu chưa khai báo biến tệp thì ta có thể vừa khai báo biến tệp, đồng thời mở tệp:

  + để đọc dữ liệu, dùng cú pháp:

```
ifstream <tên biến tệp>(<xâu tên tệp>);
```

  + để ghi dữ liệu, dùng cú pháp:

```
ofstream <tên biến tệp>(<xâu tên tệp>);
```

  Sau lệnh trên, mọi thao tác làm việc với tệp đều gắn với *tên biến tệp*.

  Ví dụ.

- Lệnh sau đây sẽ gắn tệp *dulieu.txt* chứa trong thư mục *Mydata* của ổ đĩa *F:* với biến tệp *fi* và tệp được mở ở chế độ đọc dữ liệu.

```
ifstream fi("f:\mydata\dulieu.txt")
```

  Chú ý:

  Nếu mở tệp để đọc, mà tệp được chỉ định không tồn tại thì bạn sẽ nhận được thông báo lỗi.

- Lệnh sau đây sẽ gắn tệp *ketqua.txt* chứa trong thư mục *Mydata* của ổ đĩa *F:* với biến tệp *fo* và tệp được mở ở chế độ đọc dữ liệu.

```
Ofstream fo("f:\mydata\ketqua.txt")
```

  Chú ý:

  Nếu mở tệp để ghi, mà tệp được chỉ định đang tồn tại thì tệp sẽ bị ghi đè nội dung đang có.

b) Đọc dữ liệu từ tệp đang mở ở chế độ đọc

- Đọc tương tự như lệnh `cin` : Hay dùng để đọc dữ liệu kiểu số.

```
biến_tệp >> biến1 >> biến2 >> ... >> biếnK;
```

- Đọc tương tự như lệnh `getline()` : Hay dùng để đọc dữ liệu kiểu chuỗi.

```
getline(biến_tệp, biến_xâu);
```

c) Ghi dữ liệu vào tệp đang mở ở chế độ ghi

Sử dụng tương tự như lệnh `cout`, nhưng thay từ khóa *cout* bằng tên biến tệp đang mở:

```
biến_tệp << biến1 << biến2 << ... << biếnK;
```

e) Đóng tệp đang mở

Cú pháp

```
biến_tệp.close();
```

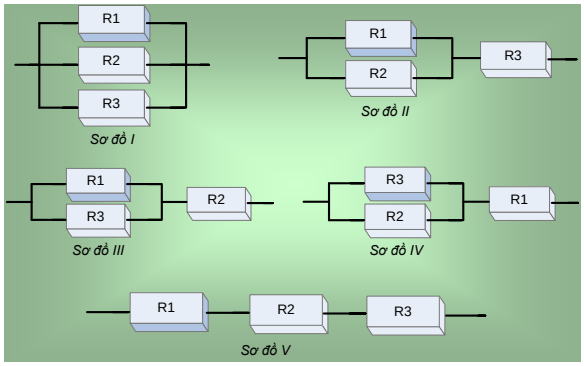
Sau khi kết thúc phiên làm việc với tệp thì ta cần đóng tệp và giải phóng tài nguyên. Giả sử tệp được mở đang gắn với biến tệp *f* thì ta chỉ cần sử dụng lệnh *f.close()* để đóng tệp.

Bài 16. VÍ DỤ LÀM VIỆC VỚI TỆP

Ví dụ 1

Tính điện trở tương đương.

Cho ba điện trở *R1, R2, R3*. Sử dụng cả ba điện trở ta có thể tạo ra năm điện trở tương đương bằng cách mắc các sơ đồ nêu ở hình 5.1.



Hình 5.1 - Sơ đồ mắc điện trở

Mỗi cách mắc sẽ cho một điện trở tương đương khác nhau. Ví dụ, nếu mắc theo sơ đồ I thì điện trở tương đương sẽ là:

$$R = \frac{R_1 * R_2 * R_3}{R_1 * R_2 + R_1 * R_3 + R_2 * R_3}$$

Còn nếu mắc theo sơ đồ V thì $R = R_1 + R_2 + R_3$.

Cho tệp văn bản RESIST.DAT gồm nhiều dòng, mỗi dòng chứa ba số thực R1, R2 và R3, các số cách nhau một dấu cách, $0 < R1, R2, R3 \leq 10^5$.

Chương trình sau đọc dữ liệu từ tệp RESIST.DAT, tính các điện trở tương đương và ghi kết quả ra tệp văn bản RESIST.EQU, mỗi dòng ghi năm điện trở tương đương của ba điện trở ở dòng dữ liệu vào tương ứng.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    ifstream fi ("resist.dat");
    ofstream fo ("resist.equ");
    float a[5], r1, r2, r3;
    //Doc den khi het tep thi dung
    while(fi >> r1 >> r2 >> r3){
        a[0] = r1*r2*r3/(r1*r2 + r1*r3 + r2*r3);
        a[1] = r1*r2/(r1+r2) + r3;
        a[2] = r1*r3/(r1+r3) + r2;
        a[3] = r3*r2/(r3+r2) + r1;
        a[4] = r1 + r2 + r3;
        // ghi danh sách a[i] ra tệp với định dạng số thực
        for (int i = 0; i < 5; i++)
            fo<<fixed<<setw(9)<<setprecision(3) << a[i];
        // xuống dòng trong tệp
        fo << endl;
    }
    fi.close();
    fo.close();
    return 0;
}
```

Ví dụ 2

Thầy hiệu trưởng tổ chức cho giáo viên và học sinh của trường đi cắm trại, sinh hoạt ngoài trời ở vườn quốc gia Cúc Phương. Để lên lịch đến thăm khu trại các lớp, thầy Hiệu trưởng cần biết khoảng cách từ trại của mình (ở vị trí có toạ độ (0;0)) đến trại các giáo viên chủ nhiệm lớp. Mỗi lớp có một khu trại, vị trí trại của mỗi thầy chủ nhiệm đều có tọa độ nguyên (x ; y), được ghi vào tệp văn bản TRAI.TXT (như vậy, tệp TRAI.TXT chứa các cặp số nguyên liên tiếp, các số cách nhau bởi dấu cách và không kết thúc bằng dấu xuống dòng).

Chương trình sau sẽ đọc từ tệp TRAI.TXT các cặp toạ độ, tính và đưa ra màn hình khoảng cách (với độ chính xác hai chữ số sau dấu chấm thập phân) từ trại mỗi giáo viên chủ nhiệm lớp đến trại của thầy hiệu trưởng.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    ifstream fi("trai.txt");
    int x, y;
    while (fi >> x >> y){
        float d = sqrt(x*x + y*y);
        cout <<fixed<<setw(10)<<setprecision(2) << d << endl;
    }
    fi.close();
```

```
12.     return 0;
13. }
```

TÓM TẮT

- Việc trao đổi dữ liệu với bộ nhớ ngoài được thực hiện thông qua kiểu dữ liệu tệp;
- Để có thể làm việc với tệp cần phải gắn tệp với biến tệp;
- Mỗi ngôn ngữ lập trình đều có các chương trình chuẩn để làm việc với tệp;
- Các thao tác với tệp văn bản:
- Cách khai báo biến tệp, mở tệp và đóng tệp.
- Đọc/ghi: tương tự như làm việc với bàn phím và màn hình.

Câu hỏi và bài tập

1. Nêu một số trường hợp cần phải dùng tệp.
2. Khi cần nhập dữ liệu từ tệp phải dùng những thao tác nào?
3. Tại sao cần phải mở tệp trước khi đọc/ghi tệp?
4. Tại sao phải dùng câu lệnh đóng tệp sau khi đã kết thúc ghi dữ liệu vào tệp?

Chương 6. Hàm và lập trình có cấu trúc

Bài 17. CHƯƠNG TRÌNH CON VÀ PHÂN LOẠI

1. Khái niệm chương trình con

Các chương trình giải các bài toán phức tạp thường rất dài, có thể gồm hàng trăm, hàng nghìn lệnh. Khi đọc những chương trình dài, rất khó nhận biết được chương trình thực hiện các công việc gì và việc hiệu chỉnh chương trình cũng khó khăn. Vì vậy, vấn đề đặt ra là phải cấu tạo chương trình như thế nào để cho chương trình dễ đọc, dễ hiệu chỉnh, nâng cấp.

Mặt khác, việc giải quyết một bài toán phức tạp thường đòi hỏi và nói chung có thể phân thành các bài toán con.

Xét bài toán tính tổng bốn luỹ thừa:

$$TLuythua = a^n + b^m + c^p + d^q$$

Bài toán đó bao gồm bốn bài toán con tính a^n, b^m, c^p, d^q, có thể giao cho bốn người, mỗi người thực hiện một bài. Giá trị *TLuythua* là tổng kết quả của bốn bài toán con đó. Với những bài toán phức tạp hơn, mỗi bài toán con lại có thể được phân chia thành các bài toán con nhỏ hơn. Quá trình phân rã làm "mịn" dần bài toán như vậy được gọi là cách thiết kế từ trên xuống.

Tương tự, khi lập trình để giải bài toán trên máy tính có thể phân chia chương trình thành các khối (môđun), mỗi khối bao gồm các lệnh giải một bài toán con nào đó. Mỗi khối lệnh sẽ được xây dựng thành *một chương trình con*. Sau đó, chương trình chính sẽ được xây dựng từ các chương trình con này. Có thể xem chương trình con cũng là một chương trình và nó cũng có thể được xây dựng từ các chương trình con khác.

Cách lập trình như vậy dựa trên phương pháp lập trình có cấu trúc và chương trình được xây dựng gọi là chương trình có cấu trúc.

Chương trình con là một dãy lệnh mô tả một số thao tác nhất định và có thể được thực hiện (được gọi) từ nhiều vị trí trong chương trình.

Ví dụ, chương trình nhập dữ liệu từ bàn phím, tính và đưa ra màn hình giá trị *TLuythua* được mô tả như trên với *a, b, c, d* có kiểu thực và *m, n, p, q* có kiểu nguyên thì chương trình trong C++ có thể như sau:

```
1. // Tinh tong
2. #include <bits/stdc++.h>
3. using namespace std;
4. float a, b, c, d;
5. int m, n, p, q;
6.
7. int main(){
```

```
8.      cout<<"Hay nhap du lieu theo thu tu a, b, c, d, m, n, p, q: ";
9.      cin >> a >> b >> c >> d >> m >> n >> p >> q;
10.
11.     double luythua1 = 1.0;
12.     for (int i = 1; i <= m; i++) luythua1 *= a;
13.
14.     double luythua2 = 1.0;
15.     for (int i = 1; i <= n; i++) luythua1 *= b;
16.
17.     double luythua3 = 1.0;
18.     for (int i = 1; i <= p; i++) luythua1 *= c;
19.
20.     double luythua4 = 1.0;
21.     for (int i = 1; i <= q; i++) luythua1 *= d;
22.
23.     double tluythua = luythua1 + luythua2 + luythua3 + luythua4;
24.     cout<<"Tong luy thua = "<<fixed<<setw(8)<<setprecision(4)<<tluythua<<endl;

25.     return 0;
26. }
```

Trong chương trình trên có bốn đoạn lệnh tương tự nhau (các dòng 11-12, 14-15, 17-18, 20-21), việc lặp lại những đoạn lệnh tương tự nhau làm cho chương trình vừa dài vừa khó theo dõi. Để nâng cao hiệu quả lập trình, các ngôn ngữ lập trình bậc cao đều cung cấp khả năng xây dựng chương trình con dạng tổng quát "đại diện" cho nhiều đoạn lệnh tương tự nhau, chẳng hạn tính luỹ thừa *Luythua* = x^k, trong đó *Luythua* và *x* là giá trị kiểu thực còn *k* thuộc kiểu nguyên:

```
1. double Luythua(double x, int k){
2.      double lt = 1.0;
3.      for(int i = 1; i < k; i++) lt *= x;
4.      return lt;
5. }
```

Ta có thể đặt tên cho chương trình con này là *Luythua* và tên các biến chứa dữ liệu vào của nó là *x* và *k*. Khi cần tính luỹ thừa của những giá trị cụ thể ta chỉ cần viết tên gọi chương trình con và thay thế (*x, k*) bằng giá trị cụ thể tương ứng. Chẳng hạn để tính a^m, b^n, c^p, d^q ta viết *Luythua(a, n), Luythua(b, m), Luythua(c,p), Luythua(d, q)*.

Lợi ích của việc sử dụng chương trình con

- Tránh được việc phải viết lặp đi lặp lại cùng một dãy lệnh nào đó tương tự như trong ví dụ tính trung bình ở trên. Ngôn ngữ lập trình cho phép tổ chức dãy lệnh đó thành một chương trình con (hàm). Sau đó, mỗi khi chương trình chính cần đến dãy lệnh này chỉ cần gọi thực hiện chương trình con đó.
- Khi phải viết chương trình lớn hàng nghìn, hàng vạn lệnh, cần huy động nhiều người tham gia, có thể giao cho mỗi người (hoặc mỗi nhóm) viết một chương trình con, rồi sau đó lắp ghép chúng lại thành chương trình chính. Ví dụ, với các bài toán mà việc tổ chức dữ liệu vào và ra không đơn giản thường người ta chia bài toán thành ba bài toán con như nhập, xử lí và xuất dữ liệu, rồi viết các chương trình con tương ứng.

- Phục vụ cho quá trình trừu tượng hoá: Người lập trình có thể sử dụng các kết quả được thực hiện bởi chương trình con mà không phải quan tâm đến việc các thao tác này được cài đặt như thế nào. Trừu tượng hoá là tư tưởng chủ đạo để xây dựng chương trình nói chung và chương trình có cấu trúc nói riêng.
- Các ngôn ngữ lập trình thường cung cấp phương thức đóng gói các chương trình con nhằm cung cấp như một câu lệnh mới (tương tự như các lệnh gọi thực hiện các hàm và thủ tục chuẩn) cho người lập trình sử dụng mà không cần biết mã nguồn của nó như thế nào.
- Thuận tiện cho phát triển, nâng cấp chương trình. Do chương trình được tạo thành từ các chương trình con nên chương trình dễ đọc, dễ hiểu, dễ kiểm tra và hiệu chỉnh. Việc nâng cấp, phát triển chương trình con nào đó, thậm chí bổ sung thêm các chương trình con mới nói chung không gây ảnh hưởng đến các chương trình con khác.
- Hiện nay, ngày càng có nhiều thiết bị kĩ thuật số tiện ích như máy quay phim, máy ảnh, máy ghi âm, các thiết bị âm thanh, màn hình màu độ phân giải cao,... có thể được kết nối với máy tính. Việc thiết kế những chương trình con thực hiện các giao tiếp cơ bản với các thiết bị như vậy là rất cần thiết và giúp mở rộng khả năng ứng dụng của ngôn ngữ.

2. Phân loại và cấu trúc của chương trình con

a) Phân loại

Trong NNLT Pascal có các khái niệm chương trình chính và chương trình con, nhưng trong NNLT C++ không phải như thế. Tất cả các chương trình C++ đều được tổ chức dưới dạng một số hàm, vì vậy các hàm có thể được coi là chương trình con. Trong đó có một hàm đặc biệt gọi là hàm *main()* nơi bắt đầu của chương trình. Hàm trong C++ có thể chia làm hai loại:

- *Hàm có kết quả* (Fruitful functions) là loại hàm thực hiện một số thao tác nào đó và trả về một số giá trị theo sau lệnh ***return***. Ví dụ các hàm toán học:

  sin(x) nhận vào giá trị thực *x* và trả về giá trị *sinx*,

  sqrt(x) nhận vào giá trị *x* trả về giá trị căn bậc hai của *x*,

  abs(x) nhận vào giá trị *x* trả về giá trị tuyệt đối của *x*,

- *Hàm không có kết quả* (Void functions) là hàm thực hiện một số thao tác nhất định nhưng không trả về giá trị nào. Ví dụ hàm *setw(), setprecision(),...*

b) Cấu trúc của hàm

Hàm có cấu trúc tương tự chương trình, nhưng nhất thiết phải có phần đầu dùng để khai báo tên và phần thân chứa các lệnh:

```
<phần đầu>
<phần thân>
```

Phần đầu là khai báo tên hàm và danh sách tham số hình thức.

Phần thân có thể có khai báo biến cho dữ liệu vào, các hằng và biến dùng trong hàm. Thành phần chủ yếu của thân hàm là dãy các câu lệnh thực hiện để từ những dữ liệu vào ta nhận được dữ liệu ra hay kết quả mong muốn, và nếu là hàm có kết quả thì nó chứa lệnh *return* trả về kết quả đó.

Tham số hình thức

Các biến được khai báo cho dữ liệu vào/ra được gọi là *tham số hình thức* của hàm. Các biến được khai báo để dùng riêng trong thân hàm được gọi là *biến cục bộ*.

Ví dụ, trong hàm *Luythua()* ở trên thì *x, k* là các tham số hình thức và *lt* là biến cục bộ.

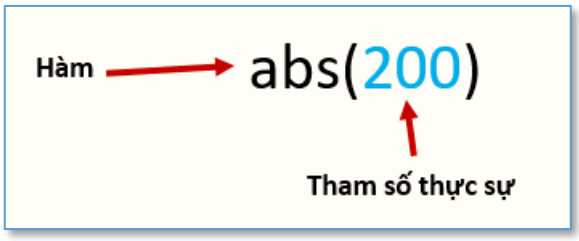
Nói chung, các hàm khác không nhìn thấy các biến cục bộ của một hàm nhưng mọi hàm đều sử dụng được các biến của chương trình. Do vậy, các biến của chương trình được gọi là *biến toàn cục*. Ví dụ, các biến *a, b, c, d, m, n, p, q* khai báo trong chương trình ở ví dụ trên là biến toàn cục.

Hàm có thể có hoặc không có tham số hình thức và biến cục bộ.

c) Thực hiện hàm

Tham số thực sự

Để thực hiện (gọi) một hàm, ta cần phải có lệnh gọi nó, tương tự lệnh gọi các hàm dựng sẵn của C++, bao gồm tên hàm với tham số (nếu có) là các hằng và biến chứa dữ liệu vào và ra tương ứng với các tham số hình thức đặt trong cặp ngoặc (và). Các hằng và biến này được gọi là các *tham số thực sự*.



Hình 6.1 - Minh họa về hàm và tham số thực sự

Khi thực hiện hàm, các tham số hình thức để nhập dữ liệu vào sẽ nhận giá trị của tham số thực sự tương ứng, còn các tham số hình thức để lưu trữ dữ liệu ra sẽ trả giá trị đó cho tham số thực sự tương ứng.

Ví dụ, khi thực hiện tính *tluythua* cần bốn lần gọi chương trình con *Luythua(x, k)* với các tham số *(a, n), (b, m), (c, p), (d, q)* và các tham số này là tham số thực sự tương ứng với tham số hình thức (*x, k*). Sau khi hàm kết thúc, lệnh tiếp theo lệnh gọi hàm sẽ được thực hiện.

Bài 18. VÍ DỤ VỀ CÁCH ĐỊNH NGHĨA VÀ SỬ DỤNG HÀM

Các ngôn ngữ lập trình đều có các quy tắc viết và sử dụng chương trình con. Trong mục này sẽ xét cách viết và sử dụng hàm trong C++.

1. Cách viết và sử dụng hàm không có kết quả

Xét ví dụ vẽ hình chữ nhật có dạng sau:

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

Ta có thể vẽ hình chữ nhật trên với ba câu lệnh:

```
cout<<"* * * * * * *"<<endl;
cout<<"*           *"<<endl;
cout<<"* * * * * * *"<<endl;
```

Như vậy, trong một chương trình, mỗi khi cần vẽ một hình chữ nhật như trên cần phải đưa vào ba câu lệnh này.

Trong chương trình sau, ta đưa ba câu lệnh trên vào một hàm có tên là *Ve_Hcn* (vẽ hình chữ nhật). Mỗi khi cần vẽ một hình chữ nhật ta cần đưa vào một câu lệnh gọi hàm đó. Chương trình gọi hàm *Ve_Hcn()* ba lần để vẽ ba hình chữ nhật.

```
1. #include <iostream>
2. using namespace std;
3.
4. void Ve_Hcn(){
5.     cout<<"* * * * * * *"<<endl;
6.     cout<<"*           *"<<endl;
7.     cout<<"* * * * * * *"<<endl;
8. }
9.
10.int main(){
11.    Ve_Hcn();
12.    cout<<endl<<endl;
13.    Ve_Hcn();
14.    cout<<endl<<endl;
15.    Ve_Hcn();
16.    return 0;
17.}
```

a. Định nghĩa hàm không có kết quả

Hàm không có kết quả có cú pháp định nghĩa như sau:

```
void <tên hàm>([danh sách tham số]){
     [<dãy các lệnh>]
}
```

Phần đầu của định nghĩa hàm gồm từ khóa ***void***, tiếp theo là tên hàm, sau đó là danh sách các tham số hình thức (nếu có) thì được liệt kê trong cặp ngoặc đơn và phân cách bằng dấu phảy.

Chú ý

- Dù định nghĩa hàm có tham số hình thức hay không thì cặp ngoặc đơn nhất thiết phải có.
- Các lệnh trong thân hàm phải nằm trong cặp móc nhọn { và }. Sau dấu } không có dấu ;

b) Ví dụ về hàm không có kết quả

Hàm *Ve_Hcn()* trong ví dụ trên chỉ vẽ được hình chữ nhật với kích thước cố định là 7×3. Giả sử chương trình cần vẽ nhiều hình chữ nhật với kích thước khác nhau. Để hàm *Ve_Hcn()* có thể thực hiện được điều đó, cần có hai tham số cho dữ liệu vào là chiều dài, chiều rộng và phần đầu của định nghĩa hàm được viết như sau:

```
void Ve_Hcn(int chdai, int chrong)
{
...
}
```

Khai báo này có nghĩa hàm *Ve_Hcn()* sẽ được thực hiện để vẽ hình chữ nhật có kích thước tuỳ theo giá trị của các tham số *chdai, chrong* và giá trị của các tham số *chdai* và *chrong* là nguyên (*int*).

Trong chương trình sau đây mô tả đầy đủ hàm *Ve_Hcn()* với các tham số *chdai, chrong* và sử dụng hàm này để vẽ các hình chữ nhật có kích thước khác nhau.

```
1. #include <iostream>
2. using namespace std;
3.
4. void Ve_Hcn(int chdai, int chrong){
5.     // Ve canh tren hinh chu nhat
6.     for(int i = 0; i < chdai; i++) cout<<"* "; cout<<endl;
7.     for(int j = 0; j < chrong-2; j++) // Ve 2 canh ben
8.     {
9.         cout<<"* ";
10.        for(int i = 0; i < chdai-2; i++) cout<<"  ";
11.        cout<<"* "<<endl;
12.    }
13.    // Ve canh duoi hinh chu nhat
14.    for(int i = 0; i < chdai; i++) cout<<"* "; cout<<endl;
15.}
16.
17.int main(){
18.    Ve_Hcn(25, 10);
19.    cout<<endl<<endl;
20.    Ve_Hcn(5, 10);
21.    cout<<endl<<endl;
22.    int a= 4, b = 2;
23.    for(int i = 0; i < 4; i++){
24.        Ve_Hcn(a, b);
25.        a *= 2, b *= 2;
26.    }
27.    return 0;
28.}
```

Trong lệnh gọi hàm, các tham số hình thức được thay bằng các tham số thực sự tương ứng là các giá trị cụ thể được gọi là các ***tham số giá trị*** (gọi tắt là *tham trị*).

Các tham số *chdai, chrong* của hàm *Ve_Hcn()* là tham trị. Trong lệnh gọi hàm *Ve_Hcn(5, 3)* (vẽ hình chữ nhật kích thước 5 × 3) tham số *chdai* được thay bởi số nguyên 5, tham số *chrong* được thay bởi số nguyên 3.

Còn trong lời gọi hàm *Ve_Hcn(a, b)* vẽ hình chữ nhật kích thước *a* × *b*, tham số *chdai* được thay bởi giá trị hiện thời của biến *a*, tham số *chrong* được thay bởi giá trị hiện thời của biến *b*.

Trong lệnh gọi hàm, các tham số hình thức được thay bằng các tham số thực sự tương ứng là tên các biến chứa dữ liệu ra được gọi là các ***tham số tham chiếu*** (gọi tắt là *tham chiếu*).

Để phân biệt tham chiếu và tham trị, C++ sử dụng ký hiệu & để khai báo những tham số là tham chiếu.

Ví dụ, hàm *Hoan_doi()* trong chương trình sau đổi giá trị của hai biến. Vì cả hai biến đều chứa dữ liệu ra nên cần sử dụng ký hiệu & để khai báo cho cả hai biến.

```
1. #include <iostream>
2. #include <iomanip>
3. using namespace std;
4.
5. void Hoan_doi(int &x, int &y){
6.     int tg = x;
7.          x = y;
8.          y = tg;
9. }
10.
11.int main(){
12.    int a =5, b = 10;
13.    cout << setw(6) << a << setw(6)<< b << endl;
14.    Hoan_doi(a,b);
15.    cout << setw(6) << a << setw(6)<< b << endl;
16.    return 0;
17.}
```

Khai báo sau xác định *x* và *y* là hai tham chiếu kiểu nguyên:

```
(int &x, int &y)
```

Trong lệnh gọi *Hoan_doi(a, b)* các tham biến *x* và *y* được thay thế bởi các biến nguyên tương ứng *a* và *b*. Giá trị của các biến này bị thay đổi khi thực hiện các lệnh:

```
tg = a; a = b; b = tg;
```

Do vậy, sau khi thực hiện hàm *Hoan_doi()*, biến *a* sẽ nhận giá trị tham chiếu của biến *b* và biến *b* nhận giá trị tham chiếu của biến *a*. Nếu giá trị của *a* là 5 còn giá trị của *b* là 10 thì trên màn hình có hai dòng:

```
     5    10
    10     5
```

Chương trình sau cho thấy sự khác nhau khi hàm dùng tham trị và tham chiếu (có và không khai báo với ký hiệu &):

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

void Hoan_doi(int x, int &y){
    int tg = x;
        x = y;
        y = tg;
}

int main(){
    int a =5, b = 10;
    cout << setw(6) << a << setw(6)<< b << endl;
    Hoan_doi(a,b);
    cout << setw(6) << a << setw(6)<< b << endl;
    return 0;
}
```

Khi chương trình thực hiện, sẽ nhận được kết quả:

```
     5    10
     5     5
```

2) Cách viết và sử dụng hàm có kết quả

Điểm khác nhau cơ bản giữa hàm không có kết quả và hàm có kết quả là việc thực hiện hàm có kết quả luôn trả về giá trị kết quả thuộc kiểu xác định và giá trị đó được trả về sau lệnh *return*.

Hàm có cấu kết quả có định nghĩa tương tự như hàm không có kết quả, tuy nhiên có khác nhau phần đầu và phần thân. Cụ thể cú pháp như sau:

```
<kiểu dữ liệu> <tên hàm>([<danh sách tham số>])
{
    <các câu lệnh>;
    return <biểu thức giá trị>;
}
```

- *Kiểu dữ liệu* là kiểu dữ liệu của giá trị mà hàm trả về và có thể là các kiểu *int, float, char, bool, string* hoặc một số kiểu dữ liệu phức hợp.
- Cũng giống như hàm không có kết quả, nếu hàm không có tham số hình thức thì không cần danh sách tham số nhưng vẫn phải có cặp ngoặc đơn (và) theo sau khai báo tên hàm. Trong thân hàm cần có lệnh trả về giá trị của hàm:

  `return <biểu thức>;`

Xét chương trình thực hiện việc rút gọn một phân số, trong đó có sử dụng hàm tính ước chung lớn nhất (UCLN) của hai số nguyên.

Ví dụ 1

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

int UCLN(int x, int y){
    while(y){
```

```
        int du = x % y;
        x = y;
        y = du;
    }
    return x;
}

int main(){
    int tu, mau;
    cout << "Nhap vao tu so, mau so: "; cin >> tu >> mau;
    int uc = UCLN(tu, mau);
    if(uc > 1){
        tu /= uc, mau /= uc;
    }
    cout << setw(5) << tu << setw(5)<< mau << endl;
    return 0;
}
```

*Chú ý: Trong chương trình này, các biến **tu, mau** và **uc** là các biến toàn cục, còn biến **du** là biến cục bộ.*

Vì hàm *UCLN()* có trả về kết quả, nên khi gọi hàm ta thường đặt hàm vào vế phải của phép gán để gán giá trị trả về của hàm cho một biến nào đó. Ví dụ:

```
a = 6*UCLN(tu, mau) + 1;
```

Ví dụ 2

Chương trình sau cho biết giá trị nhỏ nhất trong ba số nhập từ bàn phím, trong đó có sử dụng hàm tìm số nhỏ nhất trong hai số.

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

float Min(float x, float y){
    if(x < y) return x;
    return y;
}

int main(){
    float a, b, c;
    cout << "Nhap ba so: "; cin >> a >> b >> c;
    float m = Min(Min(a, b), c);
    cout <<"So nho nhat trong ba so la: ";
    cout << fixed << setw(8) << setprecision(2) << m << endl;
    return 0;
}
```

BÀI TẬP VÀ THỰC HÀNH 6

1. Mục đích, yêu cầu

- Rèn luyện các thao tác xử lí xâu;
- Nâng cao kĩ năng viết, sử dụng hàm.

2. Nội dung

a) Tìm hiểu việc xây dựng hai hàm sau đây

- Hàm *CatDan(S)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự, tạo ra xâu kết quả *t* thu được từ xâu *s* bằng việc chuyển kí tự đầu tiên của *s* xuống cuối xâu. Ví dụ nếu *s = 'abcd'* thì *t = 'bcda'*.

```
1. void CatDan(string s1, string &s2){
2.     s2 = s1.substr(1) + s1[0];
3. }
```

- Hàm *CanGiua(s)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự, bổ sung vào đầu *s* một số dấu cách để khi đưa ra màn hình xâu kí tự trong *s* ban đầu được căn giữa dòng gồm 80 kí tự.

```
1. void CanGiua(string &s){
2.     int n = s.length();
3.     n = (80-n) / 2;
4.     for(int i = 0; i < n; i++) s = ' ' + s;
5. }
```

b) Theo dõi cách sử dụng hai hàm trên, ta có thể viết chương trình sau đây để nhập một xâu kí tự từ bàn phím và đưa xâu đó ra màn hình có dạng dòng chữ chạy giữa màn hình văn bản 25×80.

```
1. #include <bits/stdc++.h>
2. #include <windows.h> // De goi cac ham phuc vu man hinh
3. #include <conio.h>   // Goi ham kbhit()
4. using namespace std;
5. COORD coord = {0, 0};// Khi bao kieu du lieu toa do
6.
7. // Ham sleep de tam dung chuong trinh trong thoi gian tinh bang miliseconds
8. void sleep(int milliseconds){
9.     clock_t time_end;
10.    time_end = clock() + milliseconds * CLOCKS_PER_SEC/1000;
11.    while (clock() < time_end){ }
12.}
13.
14.// Ham gotoxy de di chuyen con tro den vi tri (x, y) tren man hinh
15.void gotoxy (int x, int y){
16.    coord.X = x; coord.Y = y; // X and Y coordinates
17.    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
18.}
19.
20.void CatDan(string s1, string &s2){
21.    s2 = s1.substr(1) + s1[0];
22.}
23.
```

```
void CanGiua(string &s){
    int n = s.length();
    n = (80-n) / 2;
    for(int i = 0; i < n; i++) s = ' ' + s;
}

int main(){
    string s1, s2;
    cout <<"Nhap xau s1: "; getline(cin, s1);
    system("cls");
    CanGiua(s1);
    bool stop = false;
    while (!stop){
        gotoxy(12, 1); // Di chuyen con tro den cot 12, dong 1
        cout << s1;
        sleep(200); // tam dung 200 miligiay
        CatDan(s1, s2);
        s1 = s2;
        if(kbhit()) break; // Neu cos phim duoc bam thi dung chuong trinh
    }
    return 0;
}
```

Hãy chạy thử chương trình trên với dòng chữ:

'... *Mừng nghìn năm Thăng Long - Hà Nội!*... '

c) Viết hàm *ChuChay(s, dong)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự và biến nguyên *dong*, đưa ra xâu *s* có dạng chữ chạy ở dòng *dong*. Viết và chạy chương trình có sử dụng hàm này.