1.4. Lập trình C/C++ (chuyển đổi sơ đồ thành chương trình)

1.4.1. Nguyên tắc chung

Mỗi câu lệnh được kết thúc bằng dấu chấm phẩy (;)

Các từ khóa, câu lệnh trong ngôn ngữ C/C++ *phân biệt chữ hoa với chữ thường*.

Tập tin chương trình được lưu với tên `*.cpp`

Cấu trúc 1 chương trình viết bằng C/C++

```
<khai báo thư viện>

<khai báo hằng>
<khai báo biến>

<định nghĩa hàm>

int main()
{
    <Thân chương trình chính>;
    return 0;
}
```

Trước khi viết chương trình trên môi trường Code::Blocks, ta cần phải tạo một project để chứa chương trình.

Ví dụ một chương trình xuất ra màn hình thông điệp `Welcome to C/C++ language!`

Chương trình đầu tiên: Welcome.cpp

```
1  //*********************************************************
2  // Chương trình in ra màn hình thông điệp sau:
3  // Welcome to C/C++ language!
4  //*********************************************************
5
6  #include <iostream> //khai báo thư viện <iostream>
7
8  int main()
9  {
10     std::cout<<"Welcome to C/C++ language!";
11     return 0;
12 }
```

Output

```
Welcome to C/C++ language!
```

- Các dòng từ 1 đến 4 là phần chú thích (comment). Các dòng chú thích bắt đầu bằng cặp kí tự (`//`) viết liền nhau. Nội dung chú thích không ảnh hưởng đến kết quả thi hành của chương trình. Phần chú thích giúp chèn các giải thích của lập trình viên vào chương trình.
- Dòng 6 có chức năng khai báo thư viện `<iostream>` cho chương trình. Mỗi câu lệnh của C/C++ đều thuộc một thư viện nào đó nên việc khai báo thư viện là cần phải có trong một chương trình. Chẳng hạn `<iostream>` là thư viện chứa các câu lệnh liên quan đến các thao tác nhập và xuất dữ liệu cho chương trình.
- Dòng 8 chứa khai báo hàm `main()`.
- Dòng 10 là câu lệnh in chuỗi thông điệp ra thiết bị xuất chuẩn (màn hình hoặc tập tin). Chuỗi thông điệp cần in ra được đặt trong cặp dấu nháy kép (`""`) và ngay sau cặp kí tự `<<`. Cuối dòng lệnh này là dấu chấm phẩy (`;`) cho biết kết thúc một câu lệnh.

Mỗi câu lệnh của C/C++ đều thuộc một thư viện nào đó nên việc khai báo thư viện là cần phải có trong một chương trình. Một số thư viện thông dụng của C/C++ thuộc danh sách các thư viện chuẩn (Standard Library)

STT	Thư viện	Ngôn ngữ	Diễn giải
1	`<iostream>`	C++	Chứa các lệnh nhập/xuất luồng chuẩn của C++
2	`<stdio.h>/<cstdio>`	C/C++	Chứa các lệnh nhập/xuất luồng chuẩn
3	`<math.h>/<cmath>`	C/C++	Chứa các lệnh tính toán số học
4	`<string.h>/<cstring>`	C/C++	Chứa các lệnh xử lý chuỗi

1.4.2. Biến và kiểu dữ liệu

Chương trình đầu tiên chỉ thực hiện in ra màn hình thông điệp cố định mà lập trình viên chỉ định. Chương trình máy tính có thể làm được nhiều hơn như thế: nhận dữ liệu đầu vào (input), thực hiện một số thao tác xử lý trung gian và cho kết quả đầu ra (output). Như vậy, các chương trình máy tính có thể sẽ làm việc với các loại dữ liệu khác nhau trong quá trình thi hành. Dữ liệu của chương trình được lưu trữ trong bộ nhớ RAM và được chương trình truy xuất thông qua một đại lượng được gọi là biến.

Khái niệm

Biến là một vùng nhớ trên bộ nhớ RAM dùng để lưu trữ giá trị trong chương trình.

Các giá trị trong chương trình gồm có: input, output và các giá trị tính toán trung gian.

Trong ngôn ngữ C/C++, *mọi biến cần phải được khai báo trước khi sử dụng*. Tùy vào giá trị mà biến lưu trữ, *mỗi biến phải được xác định một kiểu dữ liệu tương ứng*. Như vậy, kiểu dữ liệu xác định miền giá trị mà biến được phép lưu trữ và kích thước vùng nhớ RAM dành để lưu giá trị cho chương trình.

Cú pháp khai báo biến: `<kiểu dữ liệu> <tên biến>;`

Có 4 loại kiểu dữ liệu cơ sở: kiểu số nguyên, kiểu số thực, kiểu kí tự và kiểu luận lý (kiểu logic). Các kiểu dữ liệu cơ sở thông dụng được cho trong bảng sau:

Loại dữ liệu	Kiểu dữ liệu	Kích thước (bits)	Miền giá trị
Kiểu số nguyên	`int`	32	$[-2^{31}..2^{31}-1]$
	`unsigned int`	32	$[0..2^{32}-1]$
	`long long`	64	$[-2^{63}..2^{63}-1]$
	`unsigned long long`	64	$[0..2^{64}-1]$
Kiểu số thực	`float`	32	$[-1.8^{-38}..3.4^{38}]$
	`double`	64	$[-2.2^{-308}..1.8^{308}]$
Kiểu kí tự	`char`	8	$[-128..127]$
	`unsigned char`	8	$[0..255]$
Kiểu luận lý	`bool`	8	[false..true]

Lưu ý:

- Các kí tự được lưu trữ bên trong máy tính dưới dạng các giá trị nguyên chính là mã ASCII tương ứng. Chẳng hạn `'A'` có mã ASCII 65, `'b'` có mã ASCII 98, …
- Các giá trị `false`, `true` của kiểu luận lý cũng được lưu trữ dưới dạng số nguyên với `false` tương ứng `0` và `true` tương ứng với `1`.

Ví dụ 1.4.1: Khai báo biến a kiểu số nguyên có dấu kích thước 32 bits:

```
int a;
```

Ví dụ 1.4.2: Khai báo 3 biến số thực x, y, z:

```
double x, y, z;
```

Ví dụ 1.4.3: Khai báo biến a kiểu nguyên không dấu kích thước 64 bits:

```
unsigned long long b;
```

Ví dụ 1.4.4: Khai báo biến a sau khi định nghĩa kiểu `ull`:

```
ull b;
```

Mẹo

Ta có thể sử dụng định nghĩa kiểu `typedef` để làm gọn câu lệnh khai báo:

```
typedef unsigned long long ull;
```

Sau câu lệnh trên, `ull` được xem như là một kiểu dữ liệu được định nghĩa thay cho `unsigned long long`.

1.4.3. Hằng số (constant)

Đôi khi lập trình viên cần sử dụng những đại lượng có giá trị không thay đổi trong suốt quá trình thi hành chương trình chẳng hạn như giá trị số `PI=3.14159`, giá trị gia tốc trọng trường `g=9.8m²/s` hoặc sức chứa tối đa của một phòng học `MAX_OCCUPANCY=80`. Các đại lượng này được gọi là hằng số hoặc gọi tắt là hằng.

Hằng bắt buộc phải được chỉ định giá trị ngay tại thời điểm khai báo. Cú pháp khai báo hằng: `const <kiểu dữ liệu> <tên hằng> = <giá trị>;`

Ví dụ 1.4.5: Khai báo hằng số thực pi:

```
const double PI = 3.1416;
```

Ví dụ 1.4.6: Khai báo hằng chuỗi:

```
const string msg = "Hello World!";
```

1.4.4. Câu lệnh gán (assignment)

Là câu lệnh cơ bản nhất trong lập trình được dùng để lưu trữ giá trị của chương trình vào bộ nhớ RAM thông qua biến tương ứng.

Cú pháp lệnh gán: `<biến> = <biểu thức>;`

Chức năng: lưu trữ giá trị của `<biểu thức>` ở vế phải vào `<biến>` ở vế trái. Sau câu lệnh, `<biến>` sẽ mang giá trị của `<biểu thức>`.

Có thể gán giá trị cho biến ở bước khai báo để khởi tạo giá trị ban đầu cho biến. Cú pháp lệnh khởi tạo: `<kiểu dữ liệu> <tên biến> = <biểu thức>;`

Lưu ý:

- Giá trị của biểu thức được gán cho biến phải là loại dữ liệu có cùng kiểu với biến.
- Giá trị của biểu thức được gán phải nằm trong miền giá trị giới hạn của biến.

Ví dụ 1.4.7: Gán cho biến *a* kiểu `int` giá trị 10:

```
int a;
a = 10;
```

Ví dụ 1.4.8: Lưu giá trị của biến *a* vào biến *b*:

```
int a = 8, b = a;
```

Ví dụ 1.4.9: Lưu giá trị của biến *a* kiểu `int` vào biến *b* kiểu `double`:

```
int a = 5;
double b = a;
```

Mẹo

C/C++ cung cấp một số câu lệnh toán học sau:
- `sqrt(x)`: tính căn bậc 2 của x.
- `abs(x)`: tính giá trị tuyệt đối của x.
- `sin(x)`: tính giá trị sin của số thực x.
- `cos(x)`: tính giá trị cos của số thực x.

Ví dụ 1.4.10: Lưu giá trị $\sqrt{20}$ vào biến *a* kiểu `double`:

```
double a = sqrt(20);
```

1.4.5. Các phép toán

Phép toán (operator) trong lập trình được sử dụng để kết hợp các toán hạng (operand) tạo thành một biểu thức. Toán hạng trong lập trình có thể là một hằng, một biến hoặc một biểu thức.

Phép toán được phân loại thành phép toán 1 ngôi (unary operator) và phép toán 2 ngôi (binary operator). Phép toán 1 ngôi chỉ tác động lên 1 toán hạng, phép toán 2 ngôi phải có sự tham gia của 2 toán hạng.

1.4.5.1. Phép toán số học

Phép toán số học là phép toán được áp dụng để thực hiện các thao tác tính toán số học đối với các giá trị hoặc biến có kiểu dữ liệu là số nguyên hoặc số thực. Biểu thức có chứa các phép toán số học được gọi là biểu thức số học.

Bảng dưới đây liệt kê các phép toán số học theo thứ tự độ ưu tiên từ cao xuống thấp:

Độ ưu tiên	Phép toán	Diễn giải	Loại phép toán
1	-	Phép đảo dấu	1 ngôi
2	*	Phép nhân số học	2 ngôi
2	/	Phép chia/phép chia lấy phần nguyên	2 ngôi
2	%	Phép chia lấy phần dư *(chỉ áp dụng cho các kiểu số nguyên)*	2 ngôi
3	+	Phép cộng số học	2 ngôi
3	-	Phép trừ số học	2 ngôi

Ví dụ 1.4.11: Tính tổng của a và b và lưu vào c (a, b, c có kiểu số):

```
c = a + b;
```

Ví dụ 1.4.12: Tăng giá trị của a thêm 1 đơn vị (a có kiểu số):

```
a = a + 1;
```

Ví dụ 1.4.13: Gán d là chữ số hàng đơn vị của số nguyên n:

```
d = n % 10;
```

Ví dụ 1.4.14: Loại chữ số hàng đơn vị của số nguyên n ra khỏi n:

```
n = n/10;
```

1.4.5.2. Phép toán quan hệ

Phép toán quan hệ được dùng để so sánh hai giá trị hoặc biến. Biểu thức có chứa phép toán quan hệ được gọi là biểu thức logic. Miền giá trị của biểu thức logic là `false` hoặc `true`.

Phép toán quan hệ thường được sử dụng để thiết lập điều kiện rẽ nhánh hoặc điều kiện lặp trong lập trình.

Phép toán	Diễn giải
`>`	Kiểm tra toán hạng bên trái **lớn hơn** toán hạng bên phải
`<`	Kiểm tra toán hạng bên trái **nhỏ hơn** toán hạng bên phải
`>=`	Kiểm tra toán hạng bên trái **lớn hơn hoặc bằng** toán hạng bên phải
`<=`	Kiểm tra toán hạng bên trái **nhỏ hơn hoặc bằng** toán hạng bên phải
`==`	Kiểm tra 2 toán hạng **bằng nhau**
`!=`	Kiểm tra 2 toán hạng **khác nhau**

1.4.5.3. Phép toán logic

Phép toán logic được dùng để kết hợp các biểu thức logic. Có 3 toán tử logic: `AND` (phép và, kí hiệu `&&`), `OR` (phép hoặc, kí hiệu `||`), `NOT` (phép phủ định, kí hiệu `!`).

Xét 2 toán hạng A và B chỉ nhận giá trị `false` hoặc `true`, bảng chân trị của các phép toán logic được cho trong bảng sau:

A	B	!A	A & B	A || B
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

Ví dụ 1.4.15: Biểu thức kiểm tra giá trị x có thuộc đoạn $[a, b]$ hay không

```
(x >= a && x <= b)
```

Ví dụ 1.4.16: Biểu thức kiểm tra giá trị x có nằm ngoài đoạn $[a, b]$ hay không

`(x < a || x > b)` hoặc `!(x >= a && x <= b)`

Ví dụ 1.4.17: Biểu thức kiểm tra 3 giá trị a, b, c có bằng nhau không

```
(a == b && b == c)
```

Ví dụ 1.4.18: Biểu thức kiểm tra năm *y* có phải là năm nhuận hay không. Năm nhuận là năm thỏa 1 trong 2 điều kiện sau:

- Năm chia hết cho 4 và không chia hết cho 100
- Năm chia hết cho 400

```
((y % 4 == 0 && y % 100 != 0) || y % 400 == 0)
```

1.4.5.4. Phép gán phức hợp (compound)

STT	Biểu thức	Biểu thức tương đương
1	`x += a`	`x = x + a`
2	`x -= a`	`x = x - a`
3	`x *= a`	`x = x * a`
4	`x /= a`	`x = x / a`
5	`x %= a`	`x = x % a`

1.4.5.5. Phép tăng/giảm giá trị của biến

STT	Biểu thức	Diễn giải
1	`++x`	*Tăng* giá trị của biến thêm 1 đơn vị *trước* khi thi hành câu lệnh.
2	`--x`	*Giảm* giá trị của biến đi 1 đơn vị *trước* khi thi hành câu lệnh.
3	`x++`	*Tăng* giá trị của biến thêm 1 đơn vị *sau* khi thi hành câu lệnh.
4	`x--`	*Giảm* giá trị của biến đi 1 đơn vị *sau* khi thi hành câu lệnh.

Ví dụ 1.4.19:

```
a = 5, b = 10;
x = ++a, y = --b;
```

sau đoạn lệnh các biến sẽ có giá trị như sau: `a = 6, b = 9, x = 6, y = 9`

Ví dụ 1.4.20:

```
a = 5, b = 10;
c = a++ + b--;
```

sau đoạn lệnh các biến sẽ có giá trị như sau: `a = 6, b = 9, c = 15`.

1.4.6. Chuyển đổi các cấu trúc thuật toán thành câu lệnh

1.4.6.1. Câu lệnh nhập/xuất

Lệnh nhập dữ liệu

Lệnh nhập dữ liệu thực hiện input dữ liệu cho chương trình từ thiết bị nhập chuẩn (bàn phím hoặc tập tin).

Cú pháp lệnh nhập liệu:

- Ngôn ngữ C: `scanf("<đặc tả dữ liệu nhập>", <danh sách biến>);`
- Ngôn ngữ C++: `cin>><danh sách biến>;`

Chức năng: Đọc danh sách dữ liệu từ luồng nhập chuẩn (standard input: nhập từ bàn phím hoặc tập tin) và lưu vào `<danh sách biến>` tương ứng. Khi sử dụng hàm nhập của ngôn C, ta cần phải thêm các kí tự đặc tả dữ liệu nhập cho các biến tùy vào kiểu dữ liệu tương ứng của biến.

Một số kí tự đặc tả định dạng dữ liệu nhập (dành cho ngôn ngữ C) cho một số kiểu dữ liệu thường dùng

STT	Kí tự đặc tả	Kiểu dữ liệu
1	%d	int
2	%u	unsigned int
3	%lld	long long
4	%llu	unsigned long long
5	%lf	double

Ví dụ 1.4.21:

Nhập 2 số nguyên từ bàn phím và lưu vào 2 biến tương ứng a, b

Ngôn ngữ C	Ngôn ngữ C++
`int a, b;` `scanf("%d%d", &a, &b);`	`int a, b;` `cin>>a>>b;`

Ví dụ 1.4.22:

Nhập 1 số nguyên và một số thực từ bàn phím và lưu vào 2 biến tương ứng x, y

Ngôn ngữ C	Ngôn ngữ C++

`int x;`
`double y;`
`scanf("%d%lf", &x, &y);`
	`int x;`
	`double y;`
	`cin>>x>>y;`

Lưu ý: ta bắt buộc phải thêm dấu & trước mỗi biến của hàm `scanf`.

Lệnh xuất dữ liệu

Lệnh xuất dữ liệu thực hiện output dữ liệu ra thiết bị xuất chuẩn (màn hình hoặc tập tin).

Cú pháp lệnh xuất:

- Ngôn ngữ C: `printf("<đặc tả>/<thông điệp xuất>", <danh sách biến>);`
- Ngôn ngữ C++: `cout<<"<thông điệp xuất>"/<danh sách biến>;`

Chức năng: Xuất giá trị của `<danh sách biến>` ra luồng xuất chuẩn (standard output: xuất ra màn hình hoặc tập tin). `<thông điệp xuất>` cần phải được đặt trong cặp dấu "". Tương tự như hàm `scanf`, ta cần phải thêm các kí tự đặc tả dữ liệu xuất cho các biến tùy vào kiểu dữ liệu tương ứng của biến. Các kí tự đặc tả dữ liệu xuất của hàm `printf` tương tự như các kí tự đặc tả của hàm `scanf`.

Dữ liệu xuất ra luồng chuẩn trong C có thể được điều khiển bằng các hằng kí tự tương tự như các hằng kí tự đặc tả định dạng dữ liệu nhập.

Ví dụ 1.4.23:

Xuất giá trị của 2 biến nguyên a, b ra màn hình và cách nhau khoảng trắng

Ngôn ngữ C	Ngôn ngữ C++
`printf("%d %d", a, b);`	`cout<<a<<" "<<b;`

Ví dụ 1.4.24:

Xuất giá trị của 3 biến nguyên a, b, c ra màn hình, mỗi biến nằm trên một dòng

Ngôn ngữ C	Ngôn ngữ C++
`printf("%d\n%d\n%d", a, b, c);`	`cout<<a<<"\n"<<b<<"\n"<<c;`

Ví dụ 1.4.25:

Xuất kết quả các phép tính: cộng, trừ, nhân và chia của số nguyên a cho số nguyên b. Mỗi biểu thức trên một dòng.

Ngôn ngữ C	Ngôn ngữ C++
`printf("%d + %d = %d\n", a,b,a+b);`	`cout<<a<<" + "<<b<<" = "<<a+b<<"\n";`
`printf("%d - %d = %d\n", a,b,a-b);`	`cout<<a<<" - "<<b<<" = "<<a-b<<"\n";`
`printf("%d * %d = %d\n", a,b,a*b);`	`cout<<a<<" * "<<b<<" = "<<a*b<<"\n";`
`printf("%d / %d = %d\n", a,b,a/b);`	`cout<<a<<" / "<<b<<" = "<<a/b<<"\n";`

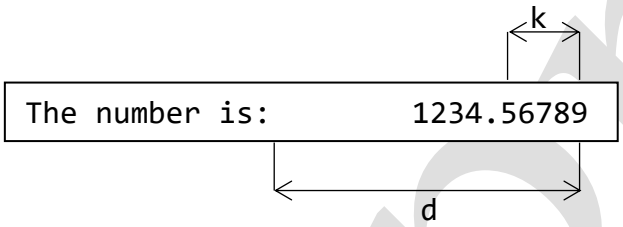
Ví dụ 1.4.26:

Xuất giá trị 2 biến số thực x, y ra màn hình và cách nhau khoảng trắng, giá trị của mỗi biến được làm tròn 3 chữ số thập phân

Ngôn ngữ C	Ngôn ngữ C++
`printf("%0.3lf %.3lf", x,y);`	`cout<<setprecision(3)<<fixed<<x<<" "<<y;`

Lưu ý:

- Để sử dụng hàm `setprecision(3)` trong C++ ta cần phải khai báo thư viện `<iomanip>`
- Câu lệnh `printf("The number is:%[d.k]lf", x)` xuất số thực x ra màn hình và được làm tròn k chữ số thập phân với d vị trí dành trước. Kết quả xuất được minh họa như hình



- Chuỗi `<thông điệp xuất>` có thể được chèn các kí tự điều khiển để định dạng kết quả xuất. Các kí tự điều khiển được đặt trong chuỗi và bắt đầu bằng kí tự \. Một số kí tự điều khiển thông thường

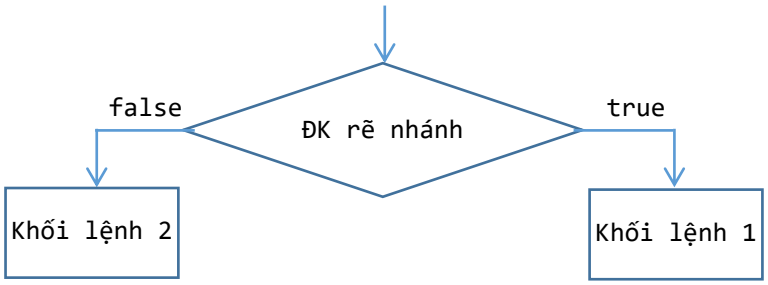
STT	Kí tự điều khiển	Diễn giải
1	`\n`	Chuyển con nháy xuống dòng mới để in giá trị tiếp theo. Tương tự thao tác nhấn Enter.
2	`\t`	Dịch con nháy đến vị trí tab tiếp theo để in giá trị tiếp theo. Tương tự thao tác nhấn phím Tab.
3	`\b`	Chuyển con nháy lui về 1 vị trí và xóa kí tự tại đó. Tương tự thao tác nhấn phím Backspace.
4	`\r`	Chuyển con nháy trở về đầu dòng. Tương tự thao tác nhấn phím Home
5	`\\`	Xuất kí tự \ ra luồng xuất chuẩn.
6	`\'`	Xuất kí tự ' ra luồng xuất chuẩn.
7	`\"`	Xuất kí tự " ra luồng xuất chuẩn.

1.4.6.2. Lệnh rẽ nhánh

Cấu trúc rẽ nhánh được sử dụng để đưa ra các quyết định thi hành trong lập trình. Có 2 dạng rẽ nhánh phổ dụng: rẽ nhánh đầy đủ và rẽ nhánh khuyết.

Rẽ nhánh đầy đủ

Sơ đồ thực thi




```
if (<ĐK rẽ nhánh>)
{
   <Khối lệnh 1>;
}
else
{
   <Khối lệnh 2>;
}
```

<Khối lệnh 1> được thực thi nếu <điều kiện rẽ nhánh> có giá trị true, ngược lại <Khối lệnh 2> được thực thi. Như vậy chỉ 1 trong 2 khối lệnh này được thực thi.

Ví dụ 1.4.27:

```
if (a % 2 == 0)
{
    a = a / 2;
    ++cnt;
}
```

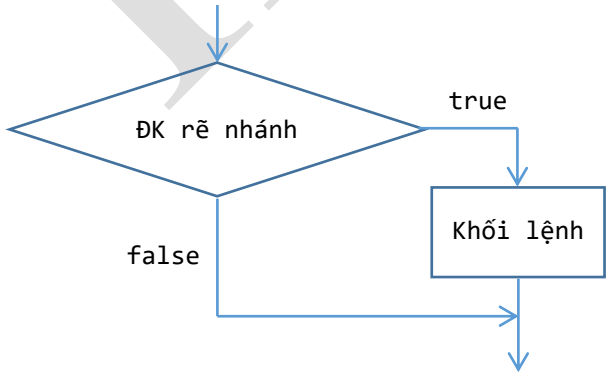
```
else
{
    a = a*3 + 1;
    ++cnt;
}
```

Lưu ý: Cặp dấu ngoặc được dùng để bao một khối lệnh. Nếu như khối lệnh chỉ có 1 câu lệnh thì ta có thể bỏ chúng đi.

Ví dụ 1.4.28: Đoạn chương trình trên có thể được viết lại như sau

```
if (a % 2 == 0)
    a = a / 2;
else a = a*3 + 1;
++cnt;
```

Rẽ nhánh khuyết




```
if (<ĐK rẽ nhánh>)
{
   <Khối lệnh>;
}
```

`<Khối lệnh>` được thực thi nếu `<điều kiện rẽ nhánh>` có giá trị `true`. Rẽ nhánh khuyết được sử dụng khi không cần quan tâm trường hợp `<điều kiện rẽ nhánh>` có giá trị `false`.

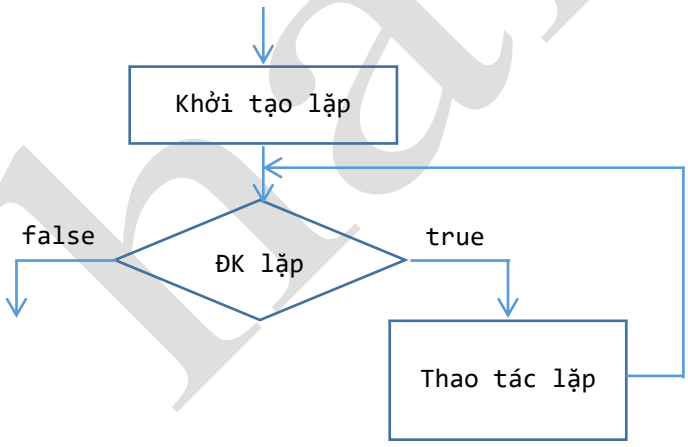
Ví dụ 1.4.29:

Cho 4 số nguyên a, b, c, d. Các số nguyên được đánh thứ tự tương ứng từ 1 đến 4. Hãy cho biết thứ tự của số lớn nhất. Nếu có nhiều số lớn nhất thì tìm thứ tự nhỏ nhất.

Đoạn chương trình thực hiện

```
int maxValue = a, pos = 1;
if (b > maxValue)
{
    maxValue = b;
    pos = 2;
}
if (c > maxValue)
{
    maxValue = c;
    pos = 3;
}
if (d > maxValue)
{
    maxValue = d;
    pos = 4;
}
```

1.4.6.3. Lệnh lặp




```
<Khởi tạo lặp>;
while (<ĐK lặp>)
{
   <Thao tác lặp>;
}
```

Ví dụ 1.4.30:

Cho số nguyên dương n. Tính tổng các chữ số của n

Đoạn chương trình thực hiện

```
int s = 0;
while (n > 0)
{
    int d = n%10;
    s = s + d;
    n = n/10;
}
```

Ví dụ 1.4.31:

Cho số nguyên dương n. Tính giá trị biểu thức

$$S = \sqrt{2 + \sqrt{2 + \cdots + \sqrt{2}}} \ (n \ dấu \ căn)$$

Đoạn chương trình thực hiện:

```
double S = 0;
int i = 1;
while (i <= n)
{
    S = sqrt(S + 2);
    ++i;
}
```

Ví dụ 1.4.32:

Phòng thí nghiệm (PTN) cần có n cá thể virus để phục vụ nghiên cứu. Các virus sẽ tự nhân đôi số lượng sau mỗi ngày nuôi cấy trong môi trường PTN. Ban đầu PTN sử dụng a cá thể virus để nuôi cấy. Lập trình tìm số ngày nuôi cấy tối thiểu để đạt đủ n cá thể virus.

Phân tích:

Input: $n, a \in \mathbb{Z}^+$.

Output: $\text{numDays} \in \mathbb{Z}^+$ là số ngày nuôi cấy tối thiểu.

Thao tác lặp: Sau mỗi ngày nuôi cấy các cá thể virus nhân đôi số lượng, biểu thức biểu diễn:

$a = a * 2$

$\text{numDays} = \text{numDays} + 1$

Khởi tạo lặp: Khi chưa bắt đầu nuôi cấy thì số ngày đếm được là 0, biểu thức biểu diễn: $\text{numDays} = 0$

Điều kiện lặp: Điều kiện còn tiếp tục nuôi cấy (khi số virus đang nuôi cấy chưa đủ n), biểu thức biểu diễn: $a < n$.

Đoạn chương trình thực hiện:

```
int numDays = 0;
while (a < n)
{
    a = a*2;
    numDays = numDays + 1;
}
```

Cấu trúc lặp được chia thành 2 loại:

- *Lặp không xác định*: không xác định được số lần lặp (ví dụ 1)
- *Lặp xác định*: xác định được số lần lặp (ví dụ 2, ví dụ 3)

Để tăng tính tiện dụng cho công việc lập trình, C/C++ cung cấp lệnh lặp `for` được dùng với ngữ nghĩa xác định được số lần lặp của thuật toán.

Cú pháp

```
for (<khởi tạo đếm>; <điều kiện lặp>; <thay đổi đếm>)
{
    <thao tác lặp>;
}
```

Đoạn chương trình ở *ví dụ 2* có thể được viết lại như sau:

```
int S = 0;
for (int i = 1; i <= n; ++i)
    S = sqrt(S + 2);
```

Ví dụ 1.4.33:

Cho số nguyên dương n. Tính giá trị biểu thức

$$S = 1 + \sqrt{2} + \cdots + \sqrt{n}$$

Đoạn chương trình thực hiện:

```
double S = 0;
for (int i = 1; i <= n; ++i)
    S = S + sqrt(i);
```

Lưu ý: Trong ngôn ngữ C/C++, vòng lặp `for` thực chất là một cách *"bố trí lại các câu lệnh"* của vòng lặp `while`. Về bản chất thì 2 vòng lặp này là như nhau.

1.4.7. Hàm (function)

Hàm (function) là một đơn vị của chương trình gồm một nhóm các câu lệnh thực thi một công việc cụ thể. Hàm còn được gọi là chương trình con (subprogram hoặc subroutine) giải quyết một *bài toán con* nào đó do đó hàm cũng nhận input, thực thi một số câu lệnh cụ thể để tạo ra output.

Hàm có thể được gọi thực thi trong các hàm khác hoặc trong chương trình chính (hàm `main`). Hàm có tính tái sử dụng nhằm tránh việc viết lại những đoạn lệnh thực hiện những tác vụ tương tự nhau nhiều lần và do đó giúp cho chương trình mang tính trong sáng, dễ sửa lỗi, dễ cải tiến.

Các ngôn ngữ lập trình luôn cung cấp các hàm cơ bản được xây dựng sẵn chứa trong các thư viện tích hợp. Đồng thời cho phép lập trình viên có thể tự định nghĩa các hàm của riêng mình phục vụ các nhu cầu đặc thù.

Một số hàm cơ bản thông dụng của ngôn ngữ C/C++:

STT	Hàm	Diễn giải
1	`sqrt(<x>)`	Trả về giá trị căn bậc 2 của `<x>`
2	`abs(<x>)`	Trả về giá trị tuyệt đối của `<x>`
3	`max(<x>,<y>)`	Trả về giá trị lớn hơn trong 2 giá trị `<x>` và `<y>`
4	`min(<x>,<y>)`	Trả về giá trị nhỏ hơn trong 2 giá trị `<x>` và `<y>`
5	`swap(<x>,<y>)`	Hoán vị giá trị của 2 biến `<x>` và `<y>`

1.4.7.1. Phân loại hàm

Hàm trả về giá trị

Hàm trả về giá trị là được sử dụng khi *bài toán con chỉ có đúng 1 output*. Giá trị trả về của hàm chính là *giá trị output*.

Cấu trúc hàm trả về giá trị:

```
<kiểu DL ouput> TênHàm(<kiểu DL> <d/s input>)
{
    <Thân hàm>;
    return <giá trị output>;
}
```

- `<danh sách input>` được gọi là tham số hình thức của hàm (formal parameter).
- Câu lệnh **`return`** trả giá trị *output* cho hàm và kết thúc hàm. Giá trị *output* của hàm có thể được tính thông qua một biểu thức có cùng kiểu dữ liệu với kiểu dữ liệu của *output*. Các lệnh nằm sau lời gọi thực thi lệnh **`return`** sẽ bị bỏ qua.

- Vì có trả về giá trị nên khi gọi thực thi hàm, ta có thể lưu trữ giá trị trả về của hàm vào một biến có cùng kiểu với dữ liệu được trả về.

Cú pháp gọi thực thi hàm có giá trị trả về:

```
<tên biến> = <TênHàm>(<d/s input>);
```

Ví dụ 1.4.34:

Cho số nguyên dương n. Xây dựng số nguyên dương m có các chữ số có thứ tự đảo ngược với thứ tự các chữ số của n

Phân tích:

Input: $Number \in \mathbb{Z}^+$.

Output: reverseNum $\in \mathbb{Z}^+$ là số đảo ngược của n.

Vì bài toán chỉ có 1 output là reverseNum nên ta sử dụng hàm có giá trị trả về:

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

int Reverse(int Number)
{
    int reverseNum = 0;
    while (Number > 0)
    {
        int lastDigit = Number % 10;
        reverseNum = (reverseNum * 10) + lastDigit;
        Number = Number / 10;
    }
    return reverseNum;
}

int main()
{
    int n;
    cin>>n;
    int m = Reverse(n); //gọi thực thi hàm có giá trị trả về
    cout<<m;
    return 0;
}
```

Ví dụ 1.4.35:

Cho số nguyên n. Lập trình kiểm tra n có phải là số nguyên tố và thông báo `YES` nếu n là số nguyên tố, ngược lại thông báo `NO`.

Phân tích:

Input: $n \in \mathbb{Z}^+$.

Output: **True** hoặc **False** tương ứng n là số nguyên tố hoặc không là số nguyên tố.

Vì bài toán chỉ có 1 output là nên sử dụng hàm có giá trị trả về:

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

bool IsPrime(int n)
{
    if (n < 2)
        return false;
    int m = sqrt(n);
    for (int i = 2; i <= m; ++i)
        if (n % i == 0)
            return false;

    return true;
}

int main()
{
    int n;
    cin>>n;
    if (IsPrime(n)) //gọi thực thi hàm có giá trị trả về
        cout<<"YES";
    else cout<<"NO";
    return 0;
}
```

Hàm không trả về giá trị

Hàm không trả về giá trị được sử dụng khi *bài toán con không cho output hoặc cho nhiều output* (từ 2 output trở lên).

Cấu trúc hàm không trả về giá trị

```
void TênHàm(<kiểu DL> <d/s input>, <kiểu DL> &<d/s output>)
{
    <Thân hàm>;
}
```

- `<d/s input>` và `<d/s output>` được gọi là tham số hình thức của hàm.
- `<d/s output>` chỉ được khai báo hình thức khi số lượng `<output>` là cố định cho mọi trường hợp. Trường hợp bài toán không có `<output>` hoặc có số lượng `<output>` không cố định thì `<d/s output>` sẽ được lược bỏ đi.
- Trước mỗi `<output>` trong `<d/s output>` hình thức đều phải thêm dấu **&**.
- **void** là một kiểu dữ liệu đặc biệt cho biết hàm không trả về giá trị.

Cú pháp gọi thực thi hàm không trả về giá trị: `<TênHàm>(<d/s input>, [<d/s output>]);`

Ví dụ 1.4.36:

Cho số nguyên dương h. Lập trình vẽ tam giác vuông cân có độ lớn cạnh góc vuông là h bằng các kí tự `'*'` có dạng như hình minh họa với $h = 4$:

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

Phân tích:

Input: $h \in \mathbb{Z}^+$

Output: không có.

Do bài toán không có output nên sử dụng hàm không trả về giá trị như sau:

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

void DrawTriangle(int h)
{
    for (int row = 1; row <= h; ++row)
    {
        for (int i = 1; i <= row; ++i)
            cout<<"* ";
        cout<<"\n";
    }
}

int main()
{
    int h;
    cin>>h;
    DrawTriangle(h); //gọi thực thi hàm không trả về giá trị
    return 0;
}
```

Ví dụ 1.4.37:

Cho số nguyên dương n. Liệt kê tất cả ước dương của n.

Phân tích:

Input: $n \in \mathbb{Z}^+$

Output: tất cả ước dương của n.

Do số lượng output của bài toán là không cố định nên sử dụng hàm không trả về giá trị như sau:

```
#include <iostream>
```

```
using namespace std;

void ListDivisors(int n)
{
    for (int i = 1; i <= n; ++i)
    {
        if (n % i == 0)
            cout<<i<<" ";
    }
}

int main()
{
    int n;
    cin>>n;
    ListDivisors(n);
    return 0;
}
```

Ví dụ 1.4.38:

Cho số nguyên dương n. Đếm số chữ số và tính tổng các chữ số của n.

Phân tích

Bài toán có 2 output là số chữ số và tổng các chữ số của n nên hàm xử lý là hàm không có giá trị trả về (Parse).

```
#include <iostream>

using namespace std;

void Parse(int n, int &sum, int &cnt)
{
    sum = cnt = 0;
    while (n > 0)
    {
        sum = sum + n % 10;
        ++cnt;
        n = n / 10;
    }
}

int main()
{
    int n, s, k;
    cin>>n;
    Parse(n, s, k);
    cout<<s<<" "<<k;
```

Mẹo

Input: $n \in \mathbb{Z}^+$

Output: $sum, cnt \in \mathbb{Z}^+$

Mẹo

Gọi thực thi hàm:

- `s` nhận giá trị của `sum`
- `k` nhận giá trị của `cnt`

```
    return 0;
}
```

1.4.7.2. Phạm vi của biến

Phạm vi của biến cho biết tầm hoạt động của biến trong một chương trình, nghĩa là phần nào của chương trình thì được hay không được phép truy xuất đến một biến nào đó. Phạm vi của biến được chia thành 2 dạng chính: biến cục bộ (local variable) và biến toàn cục (global variable).

Biến cục bộ

Biến cục bộ là biến được khai báo bên trong hàm hoặc bên trong một khối lệnh (bên trong cặp dấu ngoặc `{}`). Biến cục bộ có phạm vi hoạt động chỉ bên trong thân hàm hoặc bên trong khối lệnh được khai báo. Do đó các hàm hoặc các khối lệnh khác nhau có thể đặt biến cục bộ trùng tên nhau.

Biến cục bộ nếu không được khởi tạo giá trị thì sẽ mang giá trị bất kỳ (gọi là giá trị rác). Bộ nhớ dành cho biến cục bộ sẽ tự giải phóng khi ra khỏi phạm vi của hàm hoặc khối lệnh được khai báo.

Tham số hình thức của hàm đóng vai trò tương tự biến cục bộ của hàm đó.

Xét chương trình trong ví dụ 1.4.34:

- Biến `reverseNum` là biến cục bộ của hàm `Reverse` nên có phạm vi hoạt động chỉ bên trong hàm này.
- Biến `lastDigit` là biến cục bộ trong vòng lặp `while (n > 0)` nên có phạm vi hoạt động chỉ bên trong vòng lặp này.
- Biến `n, m` là biến cục bộ của hàm `main` nên có phạm vi hoạt động chỉ bên trong hàm `main`.

Biến toàn cục

Biến toàn cục là biến được khai báo ở ngoài tất cả hàm trong chương trình và có phạm vi hoạt động từ *vị trí khai báo đến cuối chương trình*. Bộ nhớ dành cho biến toàn cục chỉ được giải phóng khi kết thúc chương trình.

Biến toàn cục nếu không được khởi tạo thì sẽ được tự động khởi tạo giá trị 0.

Trường hợp biến toàn cục trùng tên với biến cục bộ hoặc tham số của hàm, thì các biến đóng vai trò cục bộ sẽ được ưu tiên trong phạm vi cục bộ.

Ví dụ 1.4.39:

Cho số nguyên dương n. Tìm số đảo ngược của n. (xét lại ví dụ 1.4.34)

Chương trình trong ví dụ 1.4.34 có thể được viết lại như sau:

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

int Number, reverseNum; //biến toàn cục

int Reverse(int Number)
{
    int reverseNum = 0; //biến cục bộ của hàm Reverse
    while (Number > 0)
    {
        int lastDigit = Number % 10;
        reverseNum = (reverseNum * 10) + lastDigit;
        Number = Number / 10;
    }
    return reverseNum;
}

int main()
{
    cin>>n;
    int m = Reverse(n);
    cout<<m;
    return 0;
}
```

- Biến `Number`, `reverseNum` toàn cục có phạm vi hoạt động trong toàn bộ chương trình. Tuy nhiên, hàm `Reverse` cũng có tham số `Number` và biến cục bộ `reverseNum` trùng tên với 2 biến toàn cục. Trong trường hợp này, `Number` và `reverseNum` cục bộ sẽ được ưu tiên bên trong phạm vi hàm `Reverse`.

Ví dụ 1.4.40:

Xét chương trình dưới đây và theo dõi giá trị của các biến cục bộ và toàn cục.

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

int a = 5, b = 12, c;

int Subtract(int a, int b)
{
    int c = a - b;
    cout<<"Local value: "<<a<<" "<<b<<" "<<c<<"\n";
    return c;
}
```

```
int main()
{
    c = Subtract(b, a);
    cout<<"Global value: "<<a<<" "<<b<<" "<<c<<"\n";
    return 0;
}
```

Kết quả output như sau

```
Local value: 12 5 7
Global value: 5 12 7
```

Lưu ý:

- Nên hạn chế sử dụng biến toàn cục vì có khả năng không kiểm soát được các hàm truy xuất giá trị của biến toàn cục.

Ví dụ 1.4.41:

Cho số nguyên dương n. Liệt kê các số nguyên tố không vượt quá n. Mỗi số trên một dòng.

Ví dụ này minh họa trường hợp sử dụng biến toàn cục gây lỗi lặp vô hạn

```
#include <iostream>
#include <cmath>

using namespace std;

int i, n, countDiv;

bool IsPrime(int n)
{
    if (n < 2) return false;
    int m = sqrt(n);
    for (i = 2; i <= m; ++i)
        if (n % i == 0)
            return false;
    return true;
}

void PrimeList(int n)
{
    for (i = 1; i <= n; ++i)
        if (IsPrime(i))
            cout<<i<<"\n";
}
```

```
int main()
{
    cin>>n;
    PrimeList(n);
    return 0;
}
```

- Chương trình trên bị lỗi lặp vô hạn do xung đột giá trị biến đếm `i` của hai vòng lặp **`for`** trong hàm `IsPrime` và hàm `PrimeList`.

BÀI TẬP LẬP TRÌNH

BÀI 1

Chuyển đổi tất cả bài tập trong phần thiết kế thuật toán thành chương trình.

BÀI 2

Viết chương trình nhập vào 3 số thực dương a, b, c. Cho biết a, b, c có là 3 cạnh của 1 tam giác hay không. Nếu có hãy xuất ra màn hình thông báo đó là tam giác loại gì trong các loại sau: đều, cân, vuông, vuông cân, nhọn, tù.

Dữ liệu: Nhập vào 3 số thực $a, b, c, (0 < a, b, c \leq 10^6)$.

Kết quả: Xuất ra thông báo `KHONG PHAI TAM GIAC` nếu a, b, c không lập thành 3 cạnh của một tam giác hoặc một trong các thông báo sau: `TAM GIAC DEU`, `TAM GIAC CAN`, `TAM GIAC VUONG`, `TAM GIAC VUONG CAN`, `TAM GIAC NHON`, `TAM GIAC TU`.

Ví dụ:

INPUT	OUTPUT
3 5 10	KHONG PHAI TAM GIAC
6 6 6	TAM GIAC DEU

BÀI 3

Viết chương trình nhập vào 2 số nguyên a, b là số tuổi hiện tại của em và của anh. Cho biết sau bao nhiêu năm nữa hay cách hiện tại bao nhiêu năm trước thì tuổi của anh gấp đôi tuổi của em.

Dữ liệu: Nhập vào 2 số nguyên $a, b (0 < a < b \leq 10^6)$.

Kết quả: Xuất ra thông báo `sau x nam nua` hoặc `cach day x nam`. Trong đó `x` là số năm cách năm hiện tại mà tuổi anh gấp đôi tuổi em.

Ví dụ:

INPUT	OUTPUT
5 8	cach day 2 nam
5 12	sau 2 nam nua

BÀI 4

Hình chữ nhật được gọi là che phủ hoàn toàn hình tròn nếu ta có thể đặt hình chữ nhật lên hình tròn trong một mặt phẳng sao cho không có phần nào của hình tròn được nhìn thấy. Hãy kiểm tra hình chữ nhật có thể che phủ hoàn toàn hình tròn hay không.

Dữ liệu: Nhập vào 3 số thực $a, b, R (0 < a, b, R \leq 10^6)$.

Kết quả: Xuất ra thông báo `YES` hoặc `NO` tương ứng câu trả lời mảnh gỗ hình chữ nhật có kích thước $a \times b$ có thể che phủ được hoàn toàn một mảnh gỗ hình tròn bán kính R.

Ví dụ:

INPUT	OUTPUT
4 5 3	NO
7 10 3	YES

BÀI 5

Một hình chữ nhật che phủ hoàn toàn được hình chữ nhật khác nếu ta có thể đặt hình này lên hình còn lại sao cho không có phần nào của hình còn lại được nhìn thấy. Hãy kiểm tra hình chữ nhật thứ nhất có thể che phủ hoàn toàn hình chữ nhật thứ hai hay không.

Dữ liệu: Nhập vào 4 số nguyên $a_1, b_1, a_2, b_2 (0 < a_1, b_1, a_2, b_2 \leq 10^6)$.

Kết quả: Xuất ra thông báo `YES` hoặc `NO` tương ứng câu trả lời hình chữ nhật kích thước là $a_1 \times b_1$ có che phủ hoàn toàn được hình chữ nhật kích thước $a_2 \times b_2$.

Ví dụ:

INPUT	OUTPUT
4 5 3 6	NO
5 10 2 2	YES

BÀI 6

Một tiệm bánh bán 2 loại bánh hình tròn có chất lượng và độ dày như nhau nhưng khác nhau về bán kính. Loại thứ nhất có bán kính r_1 và giá tiền c_1; loại thứ hai có bán kính r_2 và giá tiền c_2. Cho biết mua bánh nào thì có lợi hơn theo nghĩa: cùng số tiền nhưng mua được lượng bánh nhiều hơn.

Dữ liệu: Nhập vào 4 số nguyên $r_1, c_1, r_2, c_2 (0 < r_1, c_1, r_2, c_2 \leq 10^6)$.

Kết quả: Xuất ra số nguyên `1` hay `2` cho biết loại bánh có lợi hơn.

Ví dụ:

INPUT	OUTPUT
10 10 50 20	2

BÀI 7

Một tiệm bánh bán 2 loại bánh: loại thứ nhất có giá a đồng/cái, loại thứ hai có giá b đồng/cái. Một người sử dụng toàn bộ số tiền c để mua 2 loại bánh trên (hoặc chỉ mua 1 loại bánh mua cả 2 loại bánh). Cho biết tất cả những cách mua bánh với số tiền đúng bằng c đồng.

Dữ liệu: Nhập vào 3 số nguyên $a, b, c (0 < a, b \leq 10^6; 0 < c \leq 10^9)$.

Kết quả: Xuất ra tất cả cách mua 2 loại bánh để số tiền phải trả đúng bằng c, mỗi cách trên một dòng. Với mỗi cách mua xuất theo định dạng $x\ y (x, y \geq 0)$ – tương ứng với số lượng bánh từng loại. Kết quả xuất tăng dần theo x.

Ví dụ:

INPUT	OUTPUT
2 3 10	2 2

	`5 0`

BÀI 8

Phòng thí nghiệm cần n tế bào để phục vụ nghiên cứu. Các tế bào sẽ nhân đôi số lượng sau mỗi ngày nuôi cấy trong môi trường phù hợp. Ban đầu phòng thí nghiệm sử dụng a tế bào để nuôi cấy. Hãy tìm số ngày nuôi cấy tối thiểu để được đủ n tế bào.

Dữ liệu: Nhập vào số nguyên $a, n (1 \le a \le 10^6; 1 \le n \le 10^9)$.

Kết quả: Xuất ra số ngày nuôi cấy tối thiểu.

Ví dụ:

INPUT	OUTPUT
`3 20`	`3`

BÀI 9

Một cây con có chiều cao ban đầu là a. Nếu được chăm sóc thích hợp, mỗi ngày chiều cao của cây tăng thêm $b\%$. Hãy tìm chiều cao của cây sau n ngày chăm sóc.

Dữ liệu: Nhập vào số nguyên $a (1 \le a \le 10^3)$, số thực $b (1 \le b \le 100)$ và số nguyên $n (1 \le n \le 10^5)$.

Kết quả: Xuất ra số thực là chiều cao của cây sau n ngày chăm sóc. Kết quả được làm tròn 3 chữ số thập phân.

Ví dụ:

INPUT	OUTPUT
`5 10 10`	`12.969`

BÀI 10

Cho số nguyên dương n. Hãy liệt kê n số nguyên tố đầu tiên ra màn hình.

Dữ liệu: Nhập vào số nguyên $n (1 \le n \le 10^4)$.

Kết quả: Xuất ra các số nguyên tố theo thứ tự tăng dần, mỗi số nằm trên một dòng.

Ví dụ:

INPUT	OUTPUT

`5`	`2`
	`3`
	`5`
	`7`
	`11`

Lưu ý:

Các bài tập sau đây có dữ liệu được nhập xuất từ file văn bản để có thể chấm tự động. Cách đặt tên *file bài làm (*.cpp), file input (*.inp) và output (*.out)* được quy định trong mỗi bài. Trong đó 3 tập tin này ở mỗi bài *(*.cpp, *.inp, *.out)* được đặt cùng tên nhau, chỉ khác nhau phần mở rộng.

Các câu lệnh dùng để nhập xuất file (trong thư viện `<cstdio>`)

```
...

#include <cstdio>

...

int main()
{
    freopen("input.txt", "r", stdin); //mở file input.txt để đọc
    freopen("output.txt", "w", stdout); //mở file output.txt để ghi
    ...
    return 0;
}
```

Sau khi gọi câu lệnh `freopen` thì các thao tác đọc dữ liệu từ file hay ghi kết quả ra file được sử dụng bằng các lệnh `cin` và `cout` như thông thường.

SỐ HỮU NGHỊ

Hai số tự nhiên khác nhau được gọi là hữu nghị nếu tổng các ước (không kể 1 và chính nó) của số này bằng giá trị của số kia và ngược lại. Chẳng hạn 48 và 75 là cặp số hữu nghị vì $75 = 2 + 3 + 4 + 6 + 8 + 12 + 16 + 24$ và $48 = 3 + 5 + 15 + 25$.

Yêu cầu: Cho số nguyên dương n. Liệt kê tất cả cặp số hữu nghị không vượt quá n.

Dữ liệu: Vào từ tập tin văn bản **FNUM.INP** số nguyên dương $n(48 \le n \le 10^5)$.

Kết quả: Ghi ra tập tin văn bản **FNUM.OUT** tất cả cặp số hữu nghị dạng $x, y(x < y \le n)$. Mỗi cặp số trên một dòng. Các cặp số được liệt kê tăng dần theo x. Nếu không tìm được cặp số hữu nghị nào thì xuất -1.

Ví dụ:

FNUM.INP	FNUM.OUT
100	48 75

SỐ HẠNH PHÚC

Số nguyên dương x được gọi là hạnh phúc nếu x có $2 \times k$ chữ số và tổng của k chữ số đầu bằng tổng của k chữ số cuối. Chẳng hạn số 135027 là số hạnh phúc vì $1 + 3 + 5 = 0 + 2 + 7$.

Yêu cầu: Cho số nguyên dương n. Liệt kê tất cả số hạnh phúc không vượt quá n.

Dữ liệu: Vào từ tập tin văn bản **HNUM.INP** số nguyên dương $n(n \leq 10^7)$.

Kết quả: Ghi ra tập tin văn bản **HNUM.OUT** các số hạnh phúc có giá trị không vượt quá n, mỗi số trên một dòng và theo thứ tự tăng dần. Nếu không tìm được số hạnh phúc nào thì xuất -1.

Ví dụ:

HNUM.INP	HNUM.OUT
100	11 22 33 44 55 66 77 88 99

DỊCH CHỮ SỐ

Một phép dịch phải số nguyên dương n sẽ dịch chuyển các chữ số của n sang phải một vị trí, chữ số hàng đơn vị sẽ được chuyển lên hàng đầu tiên (hàng trái nhất). Chẳng hạn với $n = 2016$ thì sau một phép dịch phải ta nhận được giá trị 6201. Tiếp tục thực hiện thì ta lần lượt nhận được các giá trị tương ứng: 1620, $162, 216, 621$. Trong đó 6201 là giá trị lớn nhất nhận được.

Yêu cầu: Cho số nguyên dương n. Tìm giá trị lớn nhất khi liên tục dịch phải n.

Dữ liệu: Vào từ tập tin văn bản **RSHIFT.INP** số nguyên dương $n(n \leq 10^{15})$.

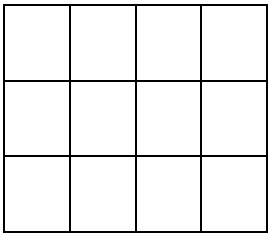
Kết quả: Ghi ra tập tin văn bản **RSHIFT.OUT** giá trị lớn nhất nhận được khi thực hiện dịch phải số n.

Ví dụ:

RSHIFT.INP	RSHIFT.OUT
2016	6201

ĐẾM SỐ HÌNH VUÔNG

Cho một lưới hình chữ nhật kích thước $n \times m$ gồm các ô vuông đơn vị như hình. Ví dụ với lưới có kích thước $n = 4$ và $m = 3$ như hình minh hoạ thì ta thấy có tổng cộng 20 hình vuông, trong đó có 12 hình vuông kích thước 1×1, 6 hình vuông kích thước 2×2 và 2 hình vuông kích thước 3×3.



Yêu cầu: Cho 2 số nguyên dương n, m. Cho biết số hình vuông có trong lưới.

Dữ liệu: Vào từ tập tin văn bản **CNTSQR.INP** 2 số nguyên $n, m(1 \leq n, m \leq 10^6)$.

Kết quả: Ghi ra tập tin văn bản **CNTSQR.OUT** tổng số hình vuông đếm được.

Ví dụ:

CNTSQR.INP	CNTSQR.OUT
4 3	20

TÍNH CƯỚC CUỘC GỌI

Cước gọi điện thoại di động của một công ty viễn thông được chia thành block và phút như sau:

- Giá cước p đồng /phút và t đồng/block (6 giây).
- Dưới 6 giây thì được tính là 1 block (6 giây).

Yêu cầu: Hãy cho biết số tiền mà khách hàng phải trả cho công ty viễn thông khi gọi điện trong n giây.

Dữ liệu: Vào từ tập tin văn bản **MOBILE.INP** chứa 3 số nguyên dương $p, t, n (1 \le p, n \le 10^9)$ – tương với giá cước 1 phút, giá cước 1 block và thời gian gọi [điện (tính theo giây).

Kết quả: Ghi ra tập tin văn bản **MOBILE.OUT** một số nguyên là số tiền mà khách hàng phải trả.

Ví dụ:

MOBILE.INP	MOBILE.OUT
100 12 200	348

Giải thích: 200 giây được chia thành 3 phút và 20 giây = 3 phút + 4 block. Do đó số tiền phải trả là $300 + 48 = 348$.

GAME THỦ

Một game thủ đang tích cực cày một game mới. Nhiệm vụ của game thủ là tiêu diệt các quái vật có level tăng dần từ level 1 trở đi. Mỗi lần tiêu diệt được quái vật ở level k thì game thủ được k điểm. Nghĩa là tiêu diệt quái vật ở level 1 được 1 điểm, tiêu diệt quái vật ở level 2 được 2 điểm, … và level phải được bắt đầu từ 1 trở đi. Game thủ hiện cần có n điểm để mua 1 bảo vật. Cho biết game thủ cần phải tiêu diệt xong quái vật ở level bao nhiêu.

Yêu cầu: Cho số nguyên dương $n (n \le 10^9)$. Cho biết level mà game thủ cần phải đạt đến để đủ n điểm thưởng.

Dữ liệu: Vào từ tập tin văn bản **GAMER.INP** số nguyên $n (1 \le n < 10^9)$.

Kết quả: Ghi ra tập tin văn bản **GAMER.OUT** số nguyên tương ứng với level mà game thủ cần đạt.

Ví dụ:

GAMER.INP	GAMER.OUT
20	6

SƠN TƯỜNG

Tí đang trang trí bằng cách quét sơn lên một dãy gồm n ô vuông nằm liên tiếp nhau như hình minh họa, các ô được đánh số từ 1 đến n. Tí dự định sẽ sơn các ô vuông bằng 2 màu sơn đen và trắng như sau:

- Ban đầu tất cả ô vuông đều được sơn màu trắng.
- Tiếp theo những ô vuông có số thứ tự là ước của n thì được sơn màu đen.

Tí nhận thấy rằng nếu làm như vậy, một số ô được quét 2 lần nước sơn, điều này gây lãng phí nên Tí cần phải tính toán trước số lượng ô cần sơn từng màu cụ thể.

Yêu cầu: Cho số nguyên dương n. Hãy đếm số lượng ô được sơn trắng và số lượng ô được sơn đen.

Dữ liệu: Vào từ tập tin văn bản **PAINTING.INP** số nguyên $n(1 \le n \le 10^{14})$.

Kết quả: Ghi ra tập tin văn bản **PAINTING.OUT** 2 số nguyên x, y tương ứng với số lượng ô cần được sơn màu trắng và màu đen. Các số xuất trên một dòng và cách nhau 1 khoảng trắng.

Ví dụ:

PAINTING.INP	PAINTING.OUT
8	4 4

MUA SỮA KHUYẾN MÃI

Công ty sữa đang có chương trình siêu khuyến mãi cho các khách hàng, khi mua n hộp sữa sẽ được tặng thêm 1 hộp. Mỗi hộp sữa có giá tiền a. Một khách hàng mang theo số tiền C và muốn dùng toàn bộ để mua sữa. Cho biết số hộp sữa nhiều nhất mà khách hàng có thể mua được.

Yêu cầu: Cho 3 số nguyên dương a, n, C. Tính số lượng hộp sữa mua được nhiều nhất.

Dữ liệu: Vào từ tập tin văn bản **BUYMILK.INP** chứa 3 số nguyên dương $a, n, C(a, n, C \le 10^{14})$.

Kết quả: Ghi ra tập tin văn bản **BUYMILK.OUT** số hộp sữa mà khách hàng mua được.

Ví dụ:

BUYMILK.INP	BUYMILK.OUT
5 10 100	22

MÁY ATM

Hiện trong máy ATM chỉ còn các tờ bạc mệnh giá 5 đồng, 10 đồng và 20 đồng với số lượng tương ứng a, b, c $(1 \le a, b, c \le 10^5)$ tờ. Một người cần rút một số tiền n đồng $(1 \le n \le 10^5)$. Cho biết máy ATM có bao nhiêu cách khác nhau để trả đủ n đồng cho khách hàng bằng số tiền hiện có trong máy.

Yêu cầu: Cho 4 số nguyên dương a, b, c, n. Cho biết số cách trả tiền khác nhau.

Dữ liệu: Vào từ tập tin văn bản **ATM.INP** chứa 4 số nguyên dương a, b, c, n.

Kết quả: Ghi ra tập tin văn bản **ATM.OUT** số cách trả tiền tìm được.

Ví dụ:

ATM.INP	ATM.OUT
5 3 2 50	6

Giải thích: có 6 cách trả tiền: $(0,1,2), (0,3,1), (2,0,2), (2,2,1), (4,1,1), (4,3,0)$

KHUYẾN MÃI

Công ty nước ngọt vừa đưa ra chương trình khuyến mãi vô cùng hấp dẫn cho các fan của loại nước uống này. Nắp chai nước ngọt sau khi khui ra sẽ được dùng để tích lũy đổi chai mới cùng loại, với k nắp chai sẽ được đổi 1 chai nước ngọt cùng loại và nắp của chai này cũng có giá trị đổi tiếp tục. Một người sau một thời gian đã tích lũy được n nắp chai. Cho biết với số nắp chai hiện tại thì trong thời gian tới người này có thể uống được bao nhiêu chai nước ngọt miễn phí.

Yêu cầu: Cho 2 số nguyên dương n, k. Tính số chai nước ngọt có thể nhận miễn phí.

Dữ liệu: Vào từ tập tin văn bản **PROMOTION.INP** chứa 2 số nguyên $n, k (2 \le k \le 10^9; 1 \le n \le 10^9)$.

Kết quả: Ghi ra tập tin văn bản **PROMOTION.OUT** số lượng chai nước ngọt nhận miễn phí.

Ví dụ:

PROMOTION.INP	PROMOTION.OUT
9 5	2

Giải thích: Với 9 nắp, người này lấy 5 nắp để đổi lấy 1 chai nước ngọt. Sau khi sử dụng xong chai vừa đổi được thì nắp của chai này cộng với 4 nắp còn dư thì đổi được thêm 1 chai nữa.

TÍNH TIỀN ĐIỆN

Đơn giá điện được công ty điện lực quy định như sau:

- Cho kW từ 1 đến 50 có đơn giá 14đ/kW
- Cho kW từ 51 đến 100 có đơn giá 15đ/kW
- Cho kW từ 101 đến 200 có đơn giá 16đ/kW
- Cho kW từ 201 đến 300 có đơn giá 17đ/kW
- Cho kW từ 301 đến 400 có đơn giá 18đ/kW
- Cho kW từ 401 trở lên có đơn giá 20đ/kW

Yêu cầu: Cho số nguyên dương n là số kW điện mà một gia đình đã tiêu thụ trong tháng. Hãy tính số tiền điện mà chủ hộ phải trả cho công ty điện lực.

Dữ liệu: Vào từ tập tin văn bản **BILL.INP** số nguyên dương $n (n \le 10^9)$.

Kết quả: Ghi ra tập tin văn bản **BILL.OUT** số tiền mà chủ hộ phải trả cho công ty điện lực.

Ví dụ:

BILL.INP	BILL.OUT
70	1000