

LÀM QUEN NGÔN NGỮ LẬP TRÌNH C++

I- NGÔN NGỮ LẬP TRÌNH C++

1. Lý thuyết

1.1. Cấu trúc chương trình C++

```
#include <iostream> //Khai báo thư viện
using namespace std; // Sử dụng không gian tên std
// Thiết kế các hàm sử dụng trong chương trình
// hàm main() là nơi thực thi các câu lệnh của chương trình
int main()
{
    int a, b, tong;
    cin>>a>>b;
    tong=a+b;
    cout << "Tong= "<<tong; // In kết quả tổng a + b
    return 0;
}
```

Không sử dụng không gian tên std

```
#include <iostream> //Khai báo thư viện
// Thiết kế các hàm sử dụng trong chương trình
// hàm main() là nơi thực thi các câu lệnh của chương trình
int main()
{
    int a, b, tong;
    std::cin>>a>>b;
    tong=a+b;
    std::cout << "Tong= "<<tong; // In kết quả tổng a + b
    return 0;
}
```

1.2. Biến và hằng

a. Biến

- Biến là một vùng nhớ nào đó trong máy tính dùng để lưu trữ giá trị. Giá trị của biến có thể thay đổi.

- Mỗi vùng nhớ biến có một địa chỉ xác định.

Cú pháp: <kiểu dữ liệu> <tên biến>;

<kiểu dữ liệu> <tên biến 1>, <tên biến 2>...;

Ví dụ:

```
int a,b;
```

```
char c;
```

```
float d;
```

- Biến có thể được khởi tạo giá trị ban đầu trong phần khai báo biến.

Cú pháp: <kiểu> <tên biến>=<giá trị>;

Ví dụ:

```
int a=7,b=9;  
char c='A';  
float d=2.1;
```

b. Hằng

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

Cú pháp

const <kiểu> <tên hằng>=<giá trị>;

hoặc

#define <tên hằng> <giá trị>

Ví dụ:

```
#define PI 3.14           //Không có ;  
const double PI=3.14;
```

1.3. Kiểu dữ liệu

a. Kiểu số nguyên có dấu: n bit có dấu: $-2^{n-1} \dots +2^{n-1} - 1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
char	1	-128...127
short	2	-32,768 ... +32,767
int	4	-2,147,483,648 ... +2,147,483,647
long	4	-2,147,483,648 ... +2,147,483,647
long long	8	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807

b. Kiểu số nguyên không dấu: n bit không dấu: $0 \dots 2^n - 1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
unsigned char	1	0...255

unsigned short	2	0 ... 65,535
unsigned int	4	0 ... 4,294,967,295
unsigned long	4	0 ... 4,294,967,295
unsigned long long	8	0 ... 18,446,744,073,709,551,615

c. Kiểu số thực:

Ví dụ: $17.06 = 1.706 \times 10^1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
float	4	$3.4 \times 10^{-38} \dots 3.4 \times 10^{38}$
double	8	$1.7 \times 10^{-308} \dots 1.7 \times 10^{308}$
long double	10	3.4×10^{-4932} đến 1.1×10^{4932}

* float chính xác đến 7 số lẻ.

* double chính xác đến 19 số lẻ.

d. Kiểu luận lý:

- false (sai): giá trị 0.
- true (đúng): giá trị khác 0, thường là 1

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
bool	1	0 và 1

Ví dụ:

- 0 (false), 1 (true), 2 (true), 2.5 (true)
- $1 > 2$ (0, false), $1 < 2$ (1, true)

e. Kiểu ký tự:

- lưu mã ASCII của 256 ký tự
- gồm 2 loại có dấu và không dấu

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
char	1	-128...127
unsigned char	1	0...255

Ví dụ

- Lưu số 65 tương đương với ký tự 'A'...

- Lưu số 97 tương đương với ký tự ‘a’.

1.4. Biểu thức

- Tạo thành từ các toán tử (Operator) và các toán hạng (Operand).
- Toán tử tác động lên các giá trị của toán hạng và cho giá trị có kiểu nhất định.
- Toán tử: +, -, *, /, %....
- Toán hạng: hàng, biến, lời gọi hàm...

Ví dụ:

2 + 3, a / 5, (a + b) * 5, ...

1.5. Các toán tử

a. Toán tử số học

Toán tử	Miêu tả	Ví dụ: A=10; B=20; C=2.0
+	Cộng hai toán hạng	A + B //kết quả là 30
-	Trừ toán hạng thứ hai từ toán hạng đầu	A - B //kết quả là -10
*	Nhân hai toán hạng	A * B //kết quả là 200 A*C //kết quả là 20.0
/	Phép chia	B / A //kết quả là 2 B/C //kết quả là 10.0
%	Phép lấy số dư	B % A //kết quả là 0
++	Toán tử tăng (++) , tăng giá trị toán hạng thêm một đơn vị	Đặt trước toán hạng Ví dụ ++x hay --x: thực hiện tăng/giảm trước. Đặt sau toán hạng Ví dụ x++ hay x--: thực hiện tăng/giảm sau. Ví dụ: x = 10; y = x++; // y = 10 và x = 11 x = 10; y = ++x; // x = 11 và y = 11
--	Toán tử giảm (--) , giảm giá trị toán hạng đi một đơn vị	

b. Toán tử quan hệ

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	(A == B) //là không đúng
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	(A != B) //là true

>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	(A > B) //là không đúng
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	(A < B) //là true
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	(A >= B) //là không đúng
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	(A <= B) //là true

c. Toán tử logic

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	(A && B) là false.
	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	(A B) là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	!(A && B) là true.

d. Toán tử gán

- Gán giá trị cho biến.

Cú pháp

- $<\text{biến}> = <\text{giá trị}>;$
- $<\text{biến}> = <\text{biến}>;$
- $<\text{biến}> = <\text{biểu thức}>;$
- Có thể thực hiện liên tiếp phép gán.

Ví dụ:

int a, b, c, d, e;

a=3; b=a+1; c=3*b;

$d=a=b=c;$

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	$C = A + B$ sẽ gán giá trị của $A + B$ vào trong C
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	$C += A$ tương đương với $C = C + A$
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	$C -= A$ tương đương với $C = C - A$
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	$C *= A$ tương đương với $C = C * A$
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	$C /= A$ tương đương với $C = C / A$
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	$C %= A$ tương đương với $C = C \% A$
<<=	Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C <<= 2$ tương đương với $C = C << 2$
>>=	Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C >>= 2$ tương đương với $C = C >> 2$
&=	Phép AND bit	$C \&= 2$ tương đương với $C = C \& 2$
^=	Phép OR loại trừ bit	$C ^= 2$ tương đương với $C = C ^ 2$
=	Phép OR bit.	$C = 2$ tương đương với $C = C 2$

e. Toán tử điều kiện

- Đây là toán tử 3 ngôi (gồm có 3 toán hạng)
- $<\text{biểu thức } 1> ? <\text{biểu thức } 2> : <\text{biểu thức } 3>$
 - $<\text{biểu thức } 1>$ đúng thì giá trị là $<\text{biểu thức } 2>$.
 - $<\text{biểu thức } 1>$ sai thì giá trị là $<\text{biểu thức } 3>$.

Ví dụ:

- $s1 = (1 > 2) ? 2912 : 1706;$

- int s2 = 0;
- 1 < 2 ? s2 = 2912 : s2 = 1706;

1.6. Nhập xuất dữ liệu

1.6.1. Nhập xuất dữ liệu từ bàn phím dùng cin, cout

- Phải khai báo sử dụng thư viện **iostream**

Nhập dữ liệu dùng hàm cin

- Cú pháp:

cin>>biến;

cin>>biến 1>>biến 2>>...>>biến n;

Xuất dữ liệu dùng hàm cout

- Cú pháp:

cout<<biểu thức;

cout<<biểu thức 1<<biểu thức<<...<<biểu thức n;

Trong đó: Biểu thức có thể chứa biến, hằng, hay kết quả trả về của một hàm.

Ví dụ:

```
#include<iostream>
using namespace std;
int main(){
    int a, b, tong;
    cin>>a>>b; //Nhập giá trị a, b từ bàn phím
    tong=a+b; //gán giá trị a+b vào biến tong
    cout<<"Tong= "<<tong; // xuất giá trị biến tong
}
```

1.6.2. Nhập xuất dữ liệu từ tập tin (file) dùng freopen, cin, cout

- Phải khai báo sử dụng thư viện **iostream**
- Phải khai báo sử dụng thư viện làm việc với file **fstream** hoặc **cstdio**

Đọc dữ liệu từ file

- Cú pháp: **freopen(filename, "r", stdin);**

Ghi dữ liệu ra file

- Cú pháp: **freopen(filename, "w", stdout);**

Trong đó: filename là địa chỉ file mục tiêu, “r” và “w” tương ứng chế độ đọc và chế độ ghi

Ví dụ:

Tính tổng a, b với a, b nhập từ file tong.inp và xuất kết quả ra file tong.out

Tong.inp	Tong.out
5 6	11

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    int a, b, tong;
    freopen("tong.inp","r",stdin); // mở file tong.inp để đọc dữ liệu
    freopen("tong.out","w",stdout); // mở file tong.out để ghi dữ liệu
    cin>>a>>b; //Nhập giá trị a, b từ bàn phím
    tong=a+b; //gán giá trị a+b vào biến tong
    cout<<tong; // xuất giá trị biến tong
}
```

1.6.3. Nhập xuất dữ liệu từ tập tin (file) dùng tên biến file

Thư viện làm việc với file

```
#include <fstream>
```

a. Nhập file

Khai báo kiểu dữ liệu file nhập:

```
ifstream <tên biến file>;
```

Mở file:

```
<tên biến file>.open(<tên file>);
```

Nhập dữ liệu từ file:

```
[<tên biến file>] >> [<biến chứa giá trị nhập từ file>];
```

Đóng file:

```
<tên biến file>.close()
```

Ví dụ:

```
ifstream fi;
fi.open("snt.inp");
```

```
int N;  
fi>> N;  
fi.close();
```

b. Xuất file

Khai báo kiểu dữ liệu file xuất và mở file:

```
ofstream <tên biến file> (<tên file>);
```

Ghi dữ liệu vào file:

```
[<tên biến file>] << [<giá trị cần ghi vào file>];
```

Đóng file:

```
<tên biến file>.close()
```

Ví dụ:

```
ofstream fo ("snt_out.txt");  
fo<<m;  
fo.close();
```

1.7. Câu lệnh IF

Câu lệnh IF thiêú:

Cú pháp:

```
if (<bíểu thức điều kiện>){  
    <lệnh>;  
}
```

Câu lệnh IF đú:

Cú pháp:

```
if (<bíểu thức điều kiện>){  
    <lệnh 1>;  
}  
else{  
    <lệnh 2>;  
}
```

Ví dụ: Tìm giá trị lớn nhất trong 3 số nguyên a, b, c

```
Int a=1; int b=2; int c=3;  
Int max;  
If(a>b){  
    max=a;  
}Else{  
    max=b;  
}
```

```
if(max<c){  
    max=c;  
}
```

1.8. Câu lệnh FOR

Lệnh for thực thi việc lặp lại một câu lệnh, một khối lệnh nhiều lần với số lần lặp xác định trước.

Cú pháp:

```
For (<giá trị khởi tạo>;<điều kiện lặp>; <giá trị tăng biến đếm>){  
    <Khối lệnh>;
```

```
}
```

Ví dụ: Tính tổng từ 1 đến 10

```
int sum=0;  
For (int i=1; i<=10; i++){ sum = sum + i;}
```

1.9. Câu lệnh WHILE

Lệnh while thực thi việc lặp lại một khối lệnh khi điều kiện kiểm tra là đúng. Điều kiện sẽ được kiểm tra **trước** khi vào thân vòng lặp do đó nếu có thay đổi giá trị kiểm tra ở trong thân vòng lặp thì khối lệnh vẫn được thực thi cho đến khi kết thúc khối lệnh. Nếu điều kiện kiểm tra là sai (FALSE) ngay từ đầu thì khối lệnh sẽ không được thực hiện dù chỉ là một lần.

Cú pháp:

```
While (<điều kiện>){  
    <Khối lệnh>;  
}
```

Ví dụ: Tính tổng từ 1 đến 10

```
int sum=0;  
int i=1;  
while (i<=10)  
{  
    Sum=sum+i;  
    i++;  
}
```

1.10. Câu lệnh DO...WHILE

Lệnh do..while thực thi việc lặp lại một khối lệnh khi điều kiện kiểm tra là đúng. Điều kiện sẽ được kiểm tra **sau** khi thực hiện khối lệnh. Nếu điều kiện kiểm tra là sai (FALSE) ngay từ đầu thì khối lệnh sẽ được thực hiện 1 lần.

Cú pháp:

```
do{  
    <Khối lệnh>;  
} while (<điều kiện>)
```

Ví dụ: Tính tổng từ 1 đến 10

```

int sum=0;
int i=1;
do{
    Sum=sum+i;
    i++;
} while (i<=10)

```

1.11. Câu lệnh break và continue

a. Câu lệnh break

- Lệnh **break** dùng để thoát khỏi một cấu trúc lặp **for, while, do...while**

Ví dụ:

```

For(int i=1;i<=10;i++){
    If(i==5) break;
    Cout<< i<< " "; //Kết quả: 1 2 3 4
}

```

b. Câu lệnh continue

- Lệnh **continue** dùng để kết thúc vòng lặp hiện tại và bắt đầu vòng lặp tiếp theo.
- Lệnh **continue** chỉ được dùng trong thân các cấu trúc lặp như **for, while, do...while**.
- Câu lệnh **continue** thường đi kèm với câu lệnh **if**.

Ví dụ:

```

For(int i=1;i<=10;i++){
    If(i==5) continue;
    Cout<< i<< " "; //Kết quả: 1 2 3 4 6 7 8 9 10
}

```

1.12. Mảng 1 chiều

Khai báo:

Cú pháp:

<Kiểu dữ liệu> <Tên mảng>[<Số lượng phần tử tối đa của mảng>];

Lưu ý: Mảng có chỉ số đầu tiên là 0

Truy xuất giá trị phần tử của mảng:

Cú pháp: <Tên mảng>[<chỉ số phần tử>];

Ví dụ:

int a[5]; //Khai báo mảng a có 5 phần tử nguyên

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50

//Khai báo mảng a có 5 phần tử, mỗi phần tử có giá trị như mô tả trên

int a[5]={10, 20, 30, 40, 50};

a[4] //truy xuất phần tử 4 giá trị 10

1.13. Mảng nhiều chiều

Khai báo mảng nhiều chiều

Cú pháp

<kiểu dữ liệu> <tên mảng>[n₁][n₂]...[n_k];

- n₁, n₂, ..., n_k: số phần tử của mỗi chiều
- <tổng số phần tử> = n₁*n₂*...*n_k

- Bộ nhớ=sizeof(kiểu dữ liệu)*<tổng số phần tử>

Ví dụ: int a[3][4];

Bộ nhớ=4*12=48(bytes)

1.14. Kiểu vector (Tham khảo thêm tại: <https://motoo.in/tutorial-thu-vien-stl-trong-c-vector/>)

Vector có thể xem như là một mảng động (dynamic array): một mảng có thể thay đổi kích thước.

Thư viện phải khai báo: #include <vector>

* Vector 1 chiều:

Cú pháp: vector <[kiểu dữ liệu]> [tên biến vector];

Ví dụ: vector <int> v1;

* Vector 2 chiều:

Cú pháp: vector <vector <[kiểu dữ liệu]>> [tên biến vector];

Lưu ý các dấu ngoặc không viết liền nhau

Ví dụ: vector <vector <int>> v2;

Các phương thức thành viên của vector:

Gọi phương thức: <đối tượng vector>.<phương thức thành viên>

Ví dụ: vector <int> v1;

v1.size(); //cho biết kích thước của vector

Capacity:	
size()	Trả về số lượng phần tử
empty()	Trả về true(1) nếu vector rỗng, ngược lại là false (0)
Element access:	
operator [int i]	Truy cập giá trị phần tử thứ i của vector, tương tự như đối mảng thông thường
at(int i)	Truy cập phần tử thứ i của vector
front()	Truy cập phần tử đầu tiên của vector
back()	Truy cập phần tử cuối cùng của vector
Modifier:	
push_back(const x)	Thêm phần tử có giá trị x vào cuối vector
pop_back()	Loại bỏ phần tử cuối ra khỏi vector
insert (iterator positon,const x) insert (iterator positon,int n, const x)	Chèn phần tử có giá trị x vào trước vị trí position. Chèn n phần tử có giá trị x vào trước vị trí position. Chèn vào trước vị trí position tất cả các phần tử trong nửa khoảng [a,b) của một
insert (iterator positon,iterator a, iterator b)	

b)	vector khác.
erase (iterator position) erase (iterator first, iterator last)	Xóa phần tử ở vị trí position Xóa tất cả các phần tử trong nửa khoảng [first,last), tức là từ phần tử thứ first đến phần tử thứ (last-1)
swap(vector v)	Hoán đổi các phần tử của vector hiện hành và vector v
clear()	Xóa vector

1.15. Kiểu chuỗi (string)

- Xử lý chuỗi thông qua lớp string.
- Khai báo `#include<string>` để sử dụng thư viện string.

Cú pháp:

```
string <tên biến>;
string <tên biến>=<Hàng chuỗi>;
```

Ví dụ:

```
string ho="Nguyen", ten="Nam", diachi;
```

1.16. Chương trình con

Cú pháp:

```
<Tên kiểu kết quả> <Tên hàm>([<kiểu tham số 1> <tham số 1>], [...])
{
    [<Khai báo biến cục bộ và các lệnh thực hiện hàm>]
    [return <Biểu thức>];
}
```

Trong đó:

- Tên kiểu kết quả: là kiểu dữ liệu trả về, có thể là `int, float, char...`. Trong trường hợp hàm không có kết quả trả về thì ta nên dùng kiểu `void`.
- Tên hàm: phải hợp lệ và không trùng với biến hoặc từ khóa.
- Kiểu tham số: là kiểu dữ liệu của tham số.
- Tham số: là tham số dữ liệu truyền vào cho hàm. Nếu có nhiều tham số, mỗi tham số cách nhau dấu phẩy (,).

Lệnh `return` để trả về kết quả thông qua tên hàm. Ngoài ra lệnh `return` còn dùng để thoát khỏi một hàm. Khi gặp lệnh `return`, chương trình sẽ thoát khỏi hàm ngay lập tức và trả về giá trị của biểu thức sau `return`

Cú pháp:

`return; //không trả về giá trị`

`return <biểu thức>; //trả về giá trị của biểu thức`

***Chú ý:** hàm có kết quả trả về, bắt buộc ta phải sử dụng câu lệnh *return* để trả kết quả cho hàm.

Ví dụ:

Thực thi hàm:

Cú pháp:

<Tên hàm> (<Danh sách các tham số>);

Ví dụ:

Viết chương trình tính x^n

```
#include <iostream>
using namespace std;
//Hàm tính lũy thừa
int luythua(int x, int n)
{
    int r = 1 ;    // r để lưu kết quả
    for (int i=1; i<=n; i++)
        r *= x ;
    return r;
}
// Ham main() là nơi sử thực thi chương trình
int main()
{
    int a=5; int b=7;
    cout << luythua(a, b); // In giá trị hàm lũy thừa
}
```

Truyền tham số cho hàm

+ Truyền giá trị

- Truyền tham số cho hàm ở dạng giá trị.
- Tham số không thay đổi sau khi hàm được thực thi.

+ Truyền tham chiếu (reference)

- Truyền đối số cho hàm ở dạng tham chiếu. Bắt đầu bằng & trong khai báo.
- Khi muốn thay đổi tham số truyền vào.

Ví dụ:

```
void tang(int x){  
    x++;  
}  
  
void giam(int &x){  
    x--;  
}  
  
int main(){  
    int a=7;  
    tang(a);  
    cout<<a<<endl; //kết quả là 7  
    giam(a);  
    cout<<a; //kết quả là 6  
}
```

2. Một số ví dụ

Ví dụ 1: (TONG.CPP) Cho số nguyên dương n ($n \leq 10^6$). Tính tổng từ 1 đến n

Dữ liệu vào: File TONG.INP: Gồm số nguyên dương n

Dữ liệu ra: File TONG.OUT: Gồm giá trị tổng từ 1 đến n

Ví dụ

TONG.INP	TONG.OUT
10	55

Chương trình tham khảo:

```
#include<iostream>  
  
#include<fstream>  
  
using namespace std;  
  
int n;  
  
void nhap(){  
    freopen("TONG.INP","r",stdin);  
    cin>>n;  
}
```

```

long tong(int n){
    long s=0;
    for(int i=1;i<=n;i++){
        s+=i;
    }
    return s;
}

void xuat(){
    freopen("TONG.OUT","w",stdout);
    cout<<tong(n);
}

int main(){
    nhap();
    xuat();
}

```

Ví dụ 2: (HOANVIAB.CPP) Cho số nguyên a, b ($1 < a, b \leq 10^6$). Xuất ra giá trị đã hoán vị của a, b

Dữ liệu vào: File HOANVIAB.INP: Gồm 2 số nguyên a, b cách nhau bởi khoảng trắng

Dữ liệu ra: File HOANVIAB.OUT: Ghi ra giá trị a, b đã hoán vị, cách nhau bởi khoảng trắng

Ví dụ

HOANVIAB.INP	HOANVIAB.OUT
5 6	6 5

Chương trình tham khảo:

```

#include<iostream>
#include<fstream>
using namespace std;
int a,b;
void nhap(){
    freopen("HOANVIAB.INP","r",stdin);
    cin>>a>>b;
}

```

```

}

void hoanvi(int &a, int &b){

    int tmp=a;

    a=b;

    b=tmp;

}

void xuat(){

    freopen("HOANVIAB.OUT","w",stdout);

    cout<<a<<" "<<b;

}

int main(){

    nhap();

    xuat();

}

```

3. Bài tập vận dụng

Bài 1 (AMDUONG.CPP): Cho dãy số gồm n số nguyên a_1, a_2, \dots, a_n . Tính tổng các phần tử âm và tổng các phần tử dương của dãy số a.

Dữ liệu vào: File AMDUONG.INP:

Dòng 1: Gồm số nguyên dương n ($1 < n \leq 1000000$)

Dòng 2: gồm n số nguyên dương a_i mỗi giá trị cách nhau bởi khoảng trống ($|a_i| < 1000000$)

Dữ liệu ra: File AMDUONG.OUT: ghi ra tổng âm và tổng dương cách nhau bởi khoảng trắng

AMDUONG.INP	AMDUONG.OUT
4 2 5 4 6	0 17
5 -3 6 -10 8 7	-13 21

Bài 2 (CHANLE.CPP): Cho dãy số gồm n số nguyên a_1, a_2, \dots, a_n . Tính tổng các phần tử chẵn và tổng các phần tử lẻ của dãy số a.

Dữ liệu vào: File CHANLE.INP:

Dòng 1: Gồm số nguyên dương n ($1 < n \leq 1000000$)

Dòng 2: gồm n số nguyên dương a_i mỗi giá trị cách nhau bởi khoảng trống ($1 < a_i < 1000000$)

Dữ liệu ra: File CHANLE.OUT: ghi ra tổng chẵn và tổng lẻ cách nhau bởi khoảng trắng

CHANLE.INP	CHANLE.OUT
4 2 3 4 5	6 8
5 2 4 6 8 10	30 0

Bài 3: (VTMAX.CPP) Cho dãy số gồm n số nguyên a_1, a_2, \dots, a_n . Tìm vị trí tất cả phần tử có giá trị lớn nhất của dãy a .

Dữ liệu vào: File VTMAX.INP:

Dòng 1: Gồm số nguyên dương n ($1 < n \leq 1000000$)

Dòng 2: gồm n số nguyên dương a_i mỗi giá trị cách nhau bởi khoảng trống ($1 < a_i \leq 1000000$)

Dữ liệu ra: File VTMAX.OUT: ghi ra vị trí các phần tử có giá trị lớn nhất, mỗi vị trí cách nhau bởi khoảng trắng

VTMAX.INP	VTMAX.OUT
4 2 8 4 8	2 4

Bài 4: (DOIHEDEM.CPP)

Cho một giá trị x ở hệ 10 hoặc hệ 2. Hãy chuyển x sang hệ 2 hoặc hệ 10 theo yêu cầu

Dữ liệu vào: File DOIHEDEM.INP:

Dòng 1: Gồm 2 số nguyên a, b, với a là hệ đếm của x, b là hệ đếm cần chuyển đổi ($a, b \in \{2; 10\}$)

Dòng 2: Là giá trị x, x có thể là chuỗi nếu ở hệ đếm 2 hoặc số nguyên dương nếu ở hệ đếm 10

Dữ liệu ra: File DOIHEDEM.OUT: ghi ra giá trị đã được chuyển đổi của x ở hệ đếm b

DOIHEDEM.INP	DOIHEDEM.OUT
2 10 1111	15
10 2 12	1100

Bài 5: (BUOCNHAY.CPP) Nam suy ra một trò chơi về số, trên một đường thăng được tạo từ n số nguyên dương, người chơi chỉ được bước trên những số chẵn. Vị trí bắt đầu là số 0, mỗi bước trong phạm vi 3 số liên tiếp, người chơi thăng nếu có thể đi hết con đường. Nam muốn biết là với một dãy số bất kỳ thì người chơi có thể thăng trò chơi không.

Ví dụ: với dãy 0 1 3 4 5 8 7 6 3 1: lần đầu người chơi đi được tới số 4 vì trong phạm vi 3 số liên tiếp, lần 2 là số 8, lần 3 là số 6, lần cuối cùng là kết thúc con đường. Người chơi thăng

Với dãy 0 2 3 5 7: lần đầu người chơi nhảy đến vị trí số 2, lần 2 không thể tìm bước đi kế tiếp trong phạm vi 3 số liên tiếp nên người chơi thua

Dữ liệu vào: Vào từ file văn bản BUOCNHAY.INP

Dòng 1: gồm một số nguyên n ($1 < n \leq 1000000$)

Dòng 2: gồm n số nguyên dương a_i mỗi giá trị cách nhau bởi khoảng trống ($1 < a_i \leq 1000000$)

Dữ liệu ra: Ghi ra file văn bản BUOCNHAY.OUT, ghi số 1 nếu người chơi để thắng trò chơi, -1 nếu người chơi thua.

Ví dụ:

BUOCNHAY.INP	BUOCNHAY.OUT
9 0 1 3 4 5 8 7 6 3 1	1
5 0 2 3 5 7	-1