Problems solved:

1 -Let's start with the mandatory ritual. Print the string: "Hello, World!" You can either use *printf*(preferred for this tutorial) or *cout*.

printf("Hello, World!");

**Sample Output**

Hello, World!

Solution

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

printf("Hello, World!");

return 0;

}

2- For any written program, a basic requirement is to take the input and print the expected output.

In C++, you can take the input using cin and print the output using cout. Here, you can use cinand cout, unlike C where you need the format specifier in printf and scanf.

Taking input:

If you want to input a number: cin>>n , where n is the number.

If you want to input a number and a string: cin>>n>>s, where s is the string.

Printing output:

If you want to output a single number: cout<<n

If you want to output a number and a string separated by a new line: cout<<n<<endl<<s

(where endl moves the printer to the new line and then the string is printed.)

In this exercise, take three numbers as inputs and print the sum of those three numbers.

**Input Format**

The first line of input contains three integers: AA, BB and CC.

1≤A,B,C≤10001≤A,B,C≤1000

**Output Format**

On a single line, print the sum of the three numbers.

**Sample Input**

1 2 7

**Sample Output**

10

**Explanation**

The sum of the three numbers: 1+2+7=101+2+7=10.

SOLUTION

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int a,b,c,sum;

cin>>a>>b>>c;

sum = a+b+c;

cout<<sum;

return 0;

}

3- A *for* loop is a programming language statement which allows code to be repeatedly executed.

The syntax for this is

for ( <expression\_1> ; <expression\_2> ; <expression\_3> )

<statement>

* *expression\_1* is used for intializing variables which are generally used for controlling terminating flag for the loop.
* *expression\_2* is used to check for the terminating condition. If this evaluates to false, then the loop is terminated.
* *expression\_3* is generally used to update the flags/variables.

A sample loop will be

for(int i = 0; i < 10; i++) {

...

}

**Input Format**

You will be given two positive integers, aa and bb (a≤ba≤b), separated by a newline.

**Output Format**

For each integer n∈[a,b]n∈[a,b] (so all numbers in that range):

* If 1≤n≤91≤n≤9, then print the English representation of it. That is "one" for 1, "two" for 2, and so on.
* Else if n>9n>9 and it is even, then print "even".
* Else if n>9n>9 and it is odd, then print "odd".

**Note:** [a,b][a,b] represents the interval, i.e., [a,b]={x∈Z| a≤x≤b}={a, a+1,…,b}[a,b]={x∈Z| a≤x≤b}={a, a+1,…,b}

**Sample Input**

8

11

**Sample Output**

eight

nine

even

odd

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

int a,b;

cin>>a;

cin>>b;

for(int i=a;i<=b;i++)

{

if(i<=9)

{

if (i==1){cout<<"one"<<endl;}

else if(i==2){cout<<"two"<<endl;}

else if(i==3){cout<<"three"<<endl;}

else if(i==4){cout<<"four"<<endl;}

else if(i==5){cout<<"five"<<endl;}

else if(i==6){cout<<"six"<<endl;}

else if(i==7){cout<<"seven"<<endl;}

else if(i==8){cout<<"eight"<<endl;}

else if(i==9){cout<<"nine"<<endl;}

}

else{

if(i%2==0){cout<<"even"<<endl;}

else {cout<<"odd"<<endl;}

}

}

return 0;

}

4 - A [pointer](http://en.wikipedia.org/wiki/Pointer_%28computer_programming%29) in C is a way to share a memory address among different contexts (primarily functions). They are primarily used whenever a function needs to modify the content of a variable, of which it doesn't have ownership.   
  
In order to access the memory address of a variable, valval, we need to prepend it with && sign. E.g., ‘‘&val"‘‘&val" returns the memory address of valval.   
  
This memory address is assigned to a pointer and can be shared among various functions. E.g. int∗p=&valint∗p=&val will assign the memory address of valval to pointer pp. To access the content of the memory to which the pointer points, prepend it with a ‘‘∗"‘‘∗". For example, ∗p∗p will return the value reflected by valval and any modification to it will be reflected at the source (valval).

void increment(int \*v) {

(\*v)++;

}

int main() {

int a;

scanf("%d", &a);

increment(&a);

printf("%d", a);

return 0;

}

You have to complete the function *void update(int \*a,int \*b)*, which reads two integers as argument, and sets aa with the sum of them, and bb with the absolute difference of them.

* a′=a+ba′=a+b
* b′=|a−b|b′=|a−b|

**Input Format**

Input will contain two integers, a and b, separated by a newline.

**Output Format**

You have to print the updated value of a and b, on two different lines.

*P.S.:* Input/ouput will be automatically handled. You only have to complete the *void update(int \*a,int \*b)* function.

**Sample Input**

4

5

**Sample Output**

9

1

**Explanation**

* a′=4+5=9a′=4+5=9
* b′=|4−5|=1

Solution:

#include <stdio.h>

void update(int \*a,int \*b) {

int tempA;

int tempB;

tempA=\*a+\*b;

if(\*a>\*b) {tempB=\*a-\*b;}

else {tempB= \*b-\*a;}

\*a=tempA;

\*b=tempB;

}

int main() {

int a, b;

int \*pa = &a, \*pb = &b;

scanf("%d %d", &a, &b);

update(pa, pb);

printf("%d\n%d", a, b);

return 0;

}

5 - An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

Declaration:

int arr[10]; //Declares an array named arr of size 10, i.e; you can store 10 integers.

Accessing elements of an array:

Indexing in arrays starts from 0.So the first element is stored at arr[0],the second element at arr[1]...arr[9]

You'll be given array of N integers and you have to print the integers in the reverse order.

**Input Format**

The first line of the input contains N, where N is the number of integers. The next line contains N integers separated by a space.

**Constraints**

1<=N<=10001<=N<=1000

1<=Ai<=100001<=Ai<=10000, where Ai is the ith integer in the array.

**Output Format**

Print the NN integers of the array in the reverse order in a single line separated by a space.

**Sample Input**

4

1 4 3 2

**Sample Output**

2 3 4 1

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int N;

cin>>N;

int arr[N];

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

{

cin>>arr[i];

}

for(int i=(N-1);i>=0;i--)

{

cout<<arr[i]<<" ";

}

return 0;

}

6 - *if* and *else* are two of the most heavily used conditionals in C/C++. They are used to execute zero or one statement among many statements.

They are be used in the following three ways.

1. *if:* It is used to execute a statement, given the condition is true.
2. if(condition) {
3. ...
4. }
5. *if - else:* It is used to execute exactly one of the two statements.
6. if(first condition) {
7. ...
8. }
9. else {
10. ...
11. }
12. *if - else if - else:* It is used to execute one of the multiple statements.
13. if(first condition) {
14. ...
15. }
16. else if(second condition) {
17. ...
18. }
19. .
20. .
21. .
22. else if((n-1)'th condition) {
23. }
24. else {
25. ...
26. }

You are given a positive integer, nn,:

* If 1≤n≤91≤n≤9, then print the English representation of it. That is "one" for 1, "two" for 2, and so on.
* Otherwise print "*Greater than 9*" (without quotes).

**Input Format**

Input will contain only one integer, nn.

**Output Format**

Print the output as described above.

**Sample Input**

5

**Sample Output**

five

**Sample Input #01**

8

**Sample Output #01**

eight

**Sample Input #02**

44

**Sample Output #02**

Greater than 9

Solution

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

unsigned int n;

cin>>n;

if(n>=1 && n<=9)

{

if (n==1){cout<<"one";}

else if (n==2){cout<<"two";}

else if (n==3){cout<<"three";}

else if (n==4){cout<<"four";}

else if (n==5){cout<<"five";}

else if (n==6){cout<<"six";}

else if (n==7){cout<<"seven";}

else if (n==8){cout<<"eight";}

else if (n==9){cout<<"nine";}

}

else {cout<<"Greater than 9";}

return 0;

}

7 - Functions are a bunch of statements glued together. A function is provided with zero or more arguments, and it executes the statements on it. Based on the return type, it either returns nothing (void) or something.   
  
A sample syntax for a function is

return\_type function\_name(arg\_type\_1 arg\_1, arg\_type\_2 arg\_2, ...) {

...

...

...

[if return\_type is non void]

return something of type `return\_type`;

}

For example, a function to read four variables and return the sum of them can be written as

int sum\_of\_four(int a, int b, int c, int d) {

int sum = 0;

sum += a;

sum += b;

sum += c;

sum += d

return sum;

}

You have to write a function *int max\_of\_four(int a, int b, int c, int d)* which reads four arguments and returns the greatest of them.

**Input Format**

Input will contain four integers - a,b,c,da,b,c,d , one in each line.

**Output Format**

Print the greatest of the four integers.   
*PS:* I/O will be automatically handled.

**Sample Input**

3

4

6

5

**Sample Output**

6

Solution:

#include <iostream>

#include <cstdio>

using namespace std;

int max\_of\_four(int a, int b, int c, int d){

int max=a;

if(b>max){max=b;}

if(c>max){max=c;}

if(d>max){max=d;}

return max;

}

int main() {

int a, b, c, d;

scanf("%d %d %d %d", &a, &b, &c, &d);

int ans = max\_of\_four(a, b, c, d);

printf("%d", ans);

return 0;

}

8 - You are given a class - *Complex*.

class Complex

{

public:

int a,b;

};

Operators are overloaded by means of operator functions, which are regular functions with special names. Their name begins with the operator keyword followed by the operator sign that is overloaded. The syntax is:

type operator sign (parameters) { /\*... body ...\*/ }

You need to overload operators + and << for the *Complex* class.

The operator + should add complex numbers according to the rules of complex addition:

(a+ib)+(c+id) = (a+c) + i(b+d)

Overload the stream insertion operator << to add "a+iba+ib" to the stream:

cout<<c<<endl;

The above statement should print "a+iba+ib" followed by a newline where a=c.aa=c.a and b=c.bb=c.b.

**Input Format**

The overloaded operator + should receive two complex numbers (a+iba+ib and c+idc+id) as parameters. It must return a single complex number.

The overloaded operator << should add "a+iba+ib" to the stream where aa is the real part and bbis the imaginary part of the complex number which is then passed as a parameter to the overloaded operator.

**Sample Input**

3+i4

5+i6

**Sample Output**

8+i10

Solution

//Operator Overloading

#include<iostream>

using namespace std;

class Complex

{

public:

int a,b;

void input(string s)

{

int v1=0;

int i=0;

while(s[i]!='+')

{

v1=v1\*10+s[i]-'0';

i++;

}

while(s[i]==' ' || s[i]=='+'||s[i]=='i')

{

i++;

}

int v2=0;

while(i<s.length())

{

v2=v2\*10+s[i]-'0';

i++;

}

a=v1;

b=v2;

}

};

Complex operator + (Complex k,Complex l )

{ Complex z;

z.a=k.a+l.a;

z.b=k.b+l.b;

return z;

}

ostream& operator << (ostream& o,Complex z)

{

return o << z.a << "+" << "i" << z.b << endl;

}

int main()

{

Complex x,y;

string s1,s2;

cin>>s1;

cin>>s2;

x.input(s1);

y.input(s2);

Complex z=x+y;

cout<<z<<endl;

}

9 - Some *C++* data types, their format specifiers, and their most common bit widths are as follows:

* *Int ("%d"):* 32 Bit integer
* *Long ("%ld"):* 32 bit integer (same as Int for modern systems)
* *Long Long ("%lld"):* 64 bit integer
* *Char ("%c"):* Character type
* *Float ("%f"):* 32 bit real value
* *Double ("%lf"):* 64 bit real value

**Reading**   
To read a data type, use the following syntax:

scanf("`format\_specifier`", &val)

For example, to read a *character* followed by a *double*:

char ch;

double d;

scanf("%c %lf", &ch, &d);

For the moment, we can ignore the spacing between format specifiers.

**Printing**   
To print a data type, use the following syntax:

printf("`format\_specifier`", val)

For example, to print a *character* followed by a *double*:

char ch = 'd';

double d = 234.432;

printf("%c %lf", ch, d);

**Note:** You can also use *cin* and *cout* instead of *scanf* and *printf*; however, if you are taking a million numbers as input and printing a million lines, it is faster to use *scanf* and *printf*.

**Input Format**

Input consists of the following space-separated values: *int*, *long*, *long long*, *char*, *float*, and*double*, respectively.

**Output Format**

Print each element on a new line in the same order it was received as input.

**Sample Input**

3 444 12345678912345 a 334.23 14049.30493

**Sample Output**

3

444

12345678912345

a

334.23

14049.30493

**Explanation**

Print *int* 33,   
followed by *long* 444444,   
followed by *long long* 1234567891234512345678912345,   
followed by *char* aa,   
followed by *float* 334.23334.23,   
followed by *double* 14049.3049314049.30493.

Solution

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

int a;

long b;

long long c;

char d;

float e;

double f;

scanf("%d %ld %lld %c %f %lf", &a, &b, &c, &d, &e, &f);

printf("%d\n%ld\n%lld\n%c\n%f\n%lf", a , b,c,d,e,f);

return 0;

}

10 - You are given an array of integers of size NN. Can you find the sum of the elements in the array?

**Input**   
The first line of input consists of an integer NN. The next line contains NN space-separated integers representing the array elements.   
Sample:

66

11 22 33 44 1010 1111

**Output**   
Output a single value equal to the sum of the elements in the array.   
For the sample above you would just print 3131 since 1+2+3+4+10+11=311+2+3+4+10+11=31.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main(){

int n;

int sum=0;

cin >> n;

vector<int> arr(n);

for(int arr\_i = 0;arr\_i < n;arr\_i++){

cin >> arr[arr\_i];

}

for(int arr\_i = 0;arr\_i < n;arr\_i++){

sum+= arr[arr\_i];

}

cout<<sum;

return 0;

}

11 - You are given an array of integers of size NN. You need to print the sum of the elements in the array, keeping in mind that some of those integers may be quite large.

**Input**

The first line of the input consists of an integer NN. The next line contains NN space-separated integers contained in the array.

**Constraints**   
1≤N≤101≤N≤10   
0≤A[i]≤10100≤A[i]≤1010

**Sample Input**   
5  
1000000001 1000000002 1000000003 1000000004 1000000005

**Output**   
Print a single value equal to the sum of the elements in the array. In the above sample, you would print 50000000155000000015.

**Note:** The range of the 32-bit integer is (−231) to (231−1) or [−2147483648,2147483647](−231) to (231−1) or [−2147483648,2147483647].  
When we add several integer values, the resulting sum might exceed the above range. You might need to use long long int in C/C++ or long data type in Java to store such sums.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main(){

int n;

cin >> n;

long long int sum=0;

vector<int> arr(n);

for(int arr\_i = 0;arr\_i < n;arr\_i++){

cin >> arr[arr\_i];

}

for(int arr\_i = 0;arr\_i < n;arr\_i++){

sum+= arr[arr\_i];

}

cout<<sum;

return 0;

}

12 - #include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <stdio.h>

#include <string.h>

using namespace std;

int main() {

int hh, mm, ss ;

char t12[2];

scanf("%d:%d:%d%s", &hh, &mm, &ss, t12) ;

if (strcmp(t12,"PM")==0 && hh!=12) hh += 12 ;

if (strcmp(t12,"AM")==0 && hh==12) hh = 0 ;

printf("%02d:%02d:%02d", hh, mm, ss) ;

return 0;

}

12 - This problem is to get you familiar with virtual functions. Create three classes *Person, Professor* and *Student*. The class *Person* should have data members name and age. The classes *Professor* and *Student* should inherit from the class *Person*.

The class *Professor* should have two data members: *publications* and *cur\_{id}*. There will be two member functions: *getdata* and *putdata*. The function *getdata* should get the input from the user: the *name, age* and *publications* of the professor. The function *putdata* should print the *name, age, publications* and the *cur\_{id}* of the professor.

The class *Student* should have two data members: *marks*, which is an array of size 66 and*cur\_{id}*. It has two member functions: *getdata* and *putdata*. The function *getdata* should get the input from the user: the *name, age*, and the *marks* of the student in 66 subjects. The function *putdata* should print the *name, age*, *sum* of the marks and the *cur\_{id}* of the student.

For each object being created of the *Professor* or the *Student* class, sequential id's should be assigned to them starting from 11.

Solve this problem using virtual functions, constructors and static variables. You can create more data members if you want.

**Input Format**

There are two types of input. If the object being created is of the *Professor* class, you will have to input the *name, age* and *publications* of the professor.

If the object is of the *Student* class, you will have to input the *name, age* and the *marks* of the student in 66 subjects.

**Constraints**

1≤lenname≤1001≤lenname≤100, where lennamelenname is the length of the name.   
1≤age≤801≤age≤80   
1≤publications≤10001≤publications≤1000   
0≤marks≤1000≤marks≤100, where marks is the marks of the student in each subject.

**Output Format**

There are two types of output depending on the object.

If the object is of type *Professor*, print the space separated *name, age, publications* and *id* on a new line.

If the object is of the *Student* class, print the space separated *name, age*, the *sum of the marks*in 66 subjects and *id* on a new line.

**Sample Input**

4

1

Walter 56 99

2

Jesse 18 50 48 97 76 34 98

2

Pinkman 22 10 12 0 18 45 50

1

White 58 87

**Sample Output**

Walter 56 99 1

Jesse 18 403 1

Pinkman 22 135 2

White 58 87 2

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

static int professorID, studentID;

class Person

{

private:

string name;

int age;

public:

virtual void getdata() {

cin >> name;

cin >> age;

}

virtual void putdata() {

cout << name << " " << age << " ";

}

};

class Professor : public Person

{

private:

int publication;

int id;

public:

Professor() {

professorID++;

}

void getdata() {

Person::getdata();

cin >> publication;

id = professorID;

}

void putdata() {

Person::putdata();

cout << publication << " " << id << endl;

}

};

class Student : public Person

{

private:

int marks[6];

int id;

public:

Student() {

studentID++;

}

void getdata() {

Person::getdata();

for(int i = 0; i < 6; ++i)

cin >> marks[i];

id = studentID;

}

void putdata() {

Person::putdata();

int sum = 0;

for(int i = 0; i < 6; ++i)

sum += marks[i];

cout << sum << " " << id << endl;

}

};

int main(){

int n, val;

cin>>n; //The number of objects that is going to be created.

Person \*per[n];

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

cin>>val;

if(val == 1){

// If val is 1 current object is of type Professor

per[i] = new Professor;

}

else per[i] = new Student; // Else the current object is of type Student

per[i]->getdata(); // Get the data from the user.

}

for(int i=0;i<n;i++)

per[i]->putdata(); // Print the required output for each object.

return 0;

}

13 - C++ provides a nice alternative data type to manipulate strings, and the data type is conveniently called *string*. Some of its widely used features are the following:

* *Declaration:*
* string a = "abc";
* *Size:*
* int len = a.size();
* *Concatenate two strings:*
* string a = "abc";
* string b = "def";
* string c = a + b; // c = "abcdef".
* *Accessing*ithith*element:*
* string s = "abc";
* char c0 = s[0]; // c0 = 'a'
* char c1 = s[1]; // c1 = 'b'
* char c2 = s[2]; // c2 = 'c'
* s[0] = 'z'; // s = "zbc"

*P.S.:* We will use *cin/cout* to read/write a string.

**Input Format**

You are given two strings, aa and bb, separated by a new line. Each string will consist of lower case Latin characters ('a'-'z').

**Output Format**

In the first line print two space-separated integers, representing the length of aa and bbrespectively.   
In the second line print the string produced by concatenating aa and bb (a+ba+b).   
In the third line print two strings separated by a space, a′a′ and b′b′. a′a′ and b′b′ are the same as aaand bb, respectively, except that their first characters are swapped.

**Sample Input**

abcd

ef

**Sample Output**

4 2

abcdef

ebcd af

**Explanation**

* a=‘‘abcd"a=‘‘abcd"
* b=‘‘ef"b=‘‘ef"
* |a|=4|a|=4
* |b|=2|b|=2
* a+b=‘‘abcdef"a+b=‘‘abcdef"
* a′=‘‘ebcd"a′=‘‘ebcd"
* b′=‘‘af"

Solution:

#include <iostream>

#include <string>

using namespace std;

int main() {

string a;

string b;

string a1;

string b1;

cin>>a;

cin>>b;

cout<<a.size()<<" "<<b.size()<<"\n";

cout<<a+b<<"\n";

a1=a;

b1=b;

a1[0]=b[0];

b1[0]=a[0];

cout<<a1<<" "<<b1;

return 0;

}

14 - Classes define new types in C++. Types in C++ not only interact by means of constructions and assignments but also via operators. For example:

int a=2, b=1, c;

c = b + a;

The result of variable *c* will be 3. Similarly, classes can also perform operations using operator overloading. Operators are overloaded by means of operator functions, which are regular functions with special names. Their name begins with the operator keyword followed by the operator sign that is overloaded. The syntax is:

type operator sign (parameters) { /\*... body ...\*/ }

You are given a main() function which takes a set of inputs to create two matrices and prints the result of their addition. You need to write the class *Matrix* which has a member *a* of type*vector<vector<int> >*. You also need to write a member function to overload the operator **+**. The function's job will be to add two objects of *Matrix* type and return the resultant *Matrix*.

**Input Format**

First line will contain the number of test cases TT. For each test case, there are three lines of input.

The first line of each test case will contain two integers NN and MM which are the sizes of the*rows* and *columns* respectively of the two matrices that will follow on the next two lines. These next two lines will each contain N∗MN∗M elements of both the matrices in a row-wise format.

**Constraints**   
1<=T<=10001<=T<=1000   
1<=N<=1001<=N<=100   
1<=M<=1001<=M<=100   
1<=Ai,j<=101<=Ai,j<=10 , where Ai,jAi,j is the element in the ithith row and jthjth column of the matrix.

**Output Format**

The code provided in the editor will use your class MatrixMatrix and overloaded operator function to add the two matrices and give the output.

**Sample Input**

1

2 2

2 2 2 2

1 2 3 4

**Sample Output**

3 4

5 6

**Explanation**

The sum of first matrix and the second matrix is the matrix given in the output.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

class Matrix{

public:

vector<vector<int> > a;

Matrix & operator + (const Matrix &y){

for (int m=0; m<y.a.size(); ++m) {

for (int n=0; n<y.a[0].size(); ++n) {

this->a[m][n] = this->a[m][n] + y.a[m][n];

}

}

return \*this;

}

};

int main () {

int cases,k;

cin >> cases;

for(k=0;k<cases;k++) {

Matrix x;

Matrix y;

Matrix result;

int n,m,i,j;

cin >> n >> m;

for(i=0;i<n;i++) {

vector<int> b;

int num;

for(j=0;j<m;j++) {

cin >> num;

b.push\_back(num);

}

x.a.push\_back(b);

}

for(i=0;i<n;i++) {

vector<int> b;

int num;

for(j=0;j<m;j++) {

cin >> num;

b.push\_back(num);

}

y.a.push\_back(b);

}

result = x+y;

for(i=0;i<n;i++) {

for(j=0;j<m;j++) {

cout << result.a[i][j] << " ";

}

cout << endl;

}

}

return 0;

}

15 - You are given NN integer sequences and QQ queries. Each query is in the following format: "aa bb" where aa denotes the index of the sequence, and bb denotes the index of the element in that sequence. Your task is to find the value of the element described in each query.

**Input Format**

The first line consists of NN and QQ separated by a space.

The following NN lines contain sequences in this format: "kk s0s0 s1s1 s2s2...... sk−1sk−1"

The following QQ lines contain queries in this format: "aa bb".

**Constraints**

1≤N≤1051≤N≤105  
1≤Q≤1051≤Q≤105  
1≤∀k≤3.1051≤∀k≤3.105  
N≤∑k≤3.105N≤∑k≤3.105  
0≤si≤1060≤si≤106  
0≤∀a<N0≤∀a<N  
0≤∀b<0≤∀b< size of the sequence

**Output Format**

Output QQ lines, the iith line contains the answer of the iith query.

**Sample Input**

2 2

3 1 5 4

5 1 2 8 9 3

0 1

1 3

**Sample Output**

5

9

**Explanation**

For the first query, the sequence is [1,5,4][1,5,4]. Hence, the answer is 55.

For the second query, the sequence is [1,2,8,9,3][1,2,8,9,3]. Hence, the answer is 99.

**Please note that the problem uses**00**-based indexing**

Solution:

#include <iostream>

using namespace std;

int main() {

int n,q;

cin>>n>>q;

int\*\* seq=new int\* [n];

for(int i=0;i<n;i++)

{

int a;

cin>>a;

int\* b=new int [a];

for(int j=0;j<a;j++)

{

int e;

cin>>e;

b[j]=e;

}

\*(seq+i)=b;

}

for(int i=0;i<q;i++)

{

int r,s;

cin>>r>>s;

cout<<seq[r][s]<<endl;

}

return 0;

}

16 - *stringstream* is a stream class to operate on strings. It basically implements input/output operations on memory (string) based streams. *stringstream* can be helpful in different type of parsing. The following operators/functions are commonly used here

* *Operator >>* Extracts formatted data.
* *Operator <<* Inserts formatted data.
* *Method str()* Gets the contents of underlying string device object.
* *Method str(string)* Sets the contents of underlying string device object.

Its header file is *sstream*.

One common use of this class is to parse comma-separated integers from a string (e.g., "23,4,56").

stringstream ss("23,4,56");

char ch;

int a, b, c;

ss >> a >> ch >> b >> ch >> c; // a = 23, b = 4, c = 56

You have to complete the function *vector parseInts(string str)*. *str* will be a string consisting of comma-separated integers, and you have to return a vector of int representing the integers.

**Note** If you want to know how to push elements in a vector, solve the first problem in the STL chapter.

**Input Format**

The first and only line consists of n integers separated by commas.

**Output Format**

Print the integers after parsing it.   
  
*P.S.:* I/O will be automatically handled. You need to complete the function only.

**Sample Input**

23,4,56

**Sample Output**

23

4

56

Solution:

#include <sstream>

#include <vector>

#include <iostream>

using namespace std;

vector<int> parseInts(string str) {

stringstream ss(str);

vector<int> final\_vector;

int temp;

char ch;

while(ss>>temp){

final\_vector.push\_back(temp);

ss>>ch;

}

return final\_vector;

}

int main() {

string str;

cin >> str;

vector<int> integers = parseInts(str);

for(int i = 0; i < integers.size(); i++) {

cout << integers[i] << "\n";

}

return 0;

}

17 - *struct* is a way to combine multiple fields to represent a composite data structure, which further lays the foundation for Object Oriented Programming. For example, we can store details related to a student in a struct consisting of his *age (int), first\_name (string), last\_name (string) and standard (int)*.   
  
*struct* can be represented as

struct NewType {

type1 value1;

type2 value2;

.

.

.

typeN valueN;

};

You have to create a struct, named *Student*, representing the student's details, as mentioned above, and store the data of a student.

**Input Format**

Input will consist of four lines.   
The first line will contain an integer, representing *age*.   
The second line will contain a string, consisting of lower-case Latin characters ('a'-'z'), representing the *first\_name* of a student.   
The third line will contain another string, consisting of lower-case Latin characters ('a'-'z'), representing the *last\_name* of a student.   
The fourth line will contain an integer, representing the *standard* of student.

*Note:* The number of characters in *first\_name* and *last\_name* will not exceed 50.

**Output Format**

Output will be of a single line, consisting of *age*, *first\_name*, *last\_name* and *standard*, each separated by one white space.

*P.S.:* I/O will be handled by HackerRank.

**Sample Input**

15

john

carmack

10

**Sample Output**

15 john carmack 10

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

struct Student {

int age;

string first\_name;

string last\_name;

int standard;

};

int main() {

Student st;

cin >> st.age >> st.first\_name >> st.last\_name >> st.standard;

cout << st.age << " " << st.first\_name << " " << st.last\_name << " " << st.standard;

return 0;

}

18 - Classes in C++ are user defined types declared with keyword class that has data and functions . Although classes and structures have the same type of functionality, there are some basic differences. The data members of a class are private by default and the members of a structure are public by default. Along with storing multiple data in a common block, it also assigns some functions (known as methods) to manipulate/access them. It serves as the building block of Object Oriented Programming.

It also has access specifiers, which restrict the access of member elements. The primarily used ones are the following:

* *public:* Public members (variables, methods) can be accessed from anywhere the code is visible.
* *private:* Private members can be accessed only by other member functions, and it can not be accessed outside of class.

Class can be represented in the form of

class ClassName {

access\_specifier1:

type1 val1;

type2 val2;

ret\_type1 method1(type\_arg1 arg1, type\_arg2 arg2,...)

...

access\_specifier2:

type3 val3;

type4 val4;

ret\_type2 method2(type\_arg3 arg3, type\_arg3 arg3,...)

...

};

It's a common practice to make all variables private, and set/get them using public methods. For example:

class SampleClass {

private:

int val;

public:

void set(int a) {

val = a;

}

int get() {

return val;

}

};

We can store details related to a student in a class consisting of his *age (int), first\_name (string), last\_name (string) and standard (int)*.   
  
You have to create a class, named *Student*, representing the student's details, as mentioned above, and store the data of a student. Create setter and getter functions for each element; that is, the class should at least have following functions:

* *get\_age*, *set\_age*
* *get\_first\_name*, *set\_first\_name*
* *get\_last\_name*, *set\_last\_name*
* *get\_standard*, *set\_standard*

Also, you have to create another method *to\_string()* which returns the string consisting of the above elements, separated by a comma(*,*). You can refer to *stringstream* for this.

**Input Format**

Input will consist of four lines.   
The first line will contain an integer, representing the *age*. The second line will contain a string, consisting of lower-case Latin characters ('a'-'z'), representing the *first\_name* of a student.   
The third line will contain another string, consisting of lower-case Latin characters ('a'-'z'), representing the *last\_name* of a student.   
The fourth line will contain an integer, representing the *standard* of student.

*Note:* The number of characters in *first\_name* and *last\_name* will not exceed 50.

**Output Format**

The code provided by HackerRank will use your class members to set and then get the elements of the *Student* class.

**Sample Input**

15

john

carmack

10

**Sample Output**

15

carmack, john

10

15,john,carmack,10

Solution:

#include <iostream>

#include <sstream>

using namespace std;

class Student{

private:

int age, standard;

string first\_name, last\_name;

stringstream ss;

char ch = ',';

public:

void set\_age(int age\_){

age = age\_;

}

int get\_age(){

return age;

}

void set\_first\_name(string first\_name\_){

first\_name = first\_name\_;

}

string get\_first\_name(){

return first\_name;

}

void set\_last\_name(string last\_name\_){

last\_name = last\_name\_;

}

string get\_last\_name(){

return last\_name;

}

void set\_standard(int standard\_){

standard = standard\_;

}

int get\_standard(){

return standard;

}

string to\_string()

{

ss<<age<<ch<<first\_name<<ch<<last\_name<<ch<<standard;

return ss.str();

}

};

int main() {

int age, standard;

string first\_name, last\_name;

cin >> age >> first\_name >> last\_name >> standard;

Student st;

st.set\_age(age);

st.set\_standard(standard);

st.set\_first\_name(first\_name);

st.set\_last\_name(last\_name);

cout << st.get\_age() << "\n";

cout << st.get\_last\_name() << ", " << st.get\_first\_name() << "\n";

cout << st.get\_standard() << "\n";

cout << "\n";

cout << st.to\_string();

return 0;

}

19 - A *class* defines a blueprint for an object. We use the same syntax to declare objects of a class as we use to declare variables of other basic types. For example:

Box box1; // Declares variable box1 of type Box

Box box2; // Declare variable box2 of type Box

Kristen is a contender for valedictorian of her high school. She wants to know how many students (if any) have scored higher than her in the 55 exams given during this semester.

Create a class named StudentStudent with the following specifications:

* An instance variable named scoresscores to hold a student's 55 exam scores.
* A *void input()* function that reads 55 integers and saves them to scoresscores.
* An *int calculateTotalScore()* function that returns the sum of the student's scores.

**Input Format**

Most of the input is handled for you by the locked code in the editor.

In the void Student::input() function, you must read 55 scores from stdin and save them to your scoresscores instance variable.

**Constraints**   
1≤n≤1001≤n≤100   
0≤examscore≤500≤examscore≤50

**Output Format**

In the int Student::calculateTotalScore() function, you must return the student's total grade (the sum of the values in scoresscores).

The locked code in the editor will determine how many scores are larger than Kristen's and print that number to the console.

**Sample Input**

The first line contains nn, the number of students in Kristen's class. The nn subsequent lines contain each student's 55 exam grades for this semester.

3

30 40 45 10 10

40 40 40 10 10

50 20 30 10 10

**Sample Output**

1

**Explanation**

Kristen's grades are on the first line of grades. Only 11 student scored higher than her.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <cassert>

using namespace std;

class Student {

private:

int scores[5];

public:

void input(){

int exam\_score;

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

cin>>exam\_score;

scores[i]=exam\_score;

}

}

int calculateTotalScore(){

int total\_score=0;

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

total\_score+=scores[i];

}

return total\_score;

}

};

int main() {

int n; // number of students

cin >> n;

Student \*s = new Student[n]; // an array of n students

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

s[i].input();

}

// calculate kristen's score

int kristen\_score = s[0].calculateTotalScore();

// determine how many students scored higher than kristen

int count = 0;

for(int i = 1; i < n; i++){

int total = s[i].calculateTotalScore();

if(total > kristen\_score){

count++;

}

}

// print result

cout << count;

return 0;

}

20 - A class template provides a specification for generating classes based on parameters. *Class templates* are generally used to implement containers. A class template is instantiated by passing a given set of types to it as template arguments. Here is an example of a class, MyTemplate, that can store one element of any type and that has just one member function*divideBy2*, which divides its value by 2.

template <class T>

class MyTemplate {

T element;

public:

MyTemplate (T arg) {element=arg;}

T divideBy2 () {return element/2;}

};

It is also possible to define a different implementation of a template for a specific type. This is called *Template Specialization*. For the template given above, we find that a different implementation for type *char* will be more useful, so we write a function *printElement* to print the *char* element:

// class template specialization:

template <>

class MyTemplate <char> {

char element;

public:

MyTemplate (char arg) {element=arg;}

char printElement ()

{

return element;

}

};

You are given a main() function which takes a set of inputs. The type of input governs the kind of operation to be performed, i.e. concatenation for *strings* and addition for *int* or *float*. You need to write the class template *AddElements* which has a function *add()* for giving the sum of *int* or *float* elements. You also need to write a template specialization for the type*string* with a function *concatenate()* to concatenate the second string to the first string.

**Input Format**

Input will consist of *N+1* lines where *N* is the number given in the first line of the input followed by *N* lines.

From the second line forward, the type of the following two elements will be provided. The type will be one of *int*, *float* or *string* types only. Out of the following two elements, you have to concatenate or add the second element to the first element.

**Constraints**   
1<=N<=5000001<=N<=500000   
1.0<=valuefloat<=10.01.0<=valuefloat<=10.0, where valuefloatvaluefloat is any float value   
1<=valueint<=1000001<=valueint<=100000, where valueint is any int value   
0<=lenstring<=100<=lenstring<=10, where lenstringlenstring is the length of any string

**The time limit for this challenge is 4 seconds**

**Output Format**

The code provided in the code editor will use your class template to add/append elements and give the output.

**Sample Input**

3

string John Doe

int 1 2

float 4.0 1.5

**Sample Output**

JohnDoe

3

5.5

**Explanation**

"Doe" when appended with "John" gives "JohnDoe". 2 added to 1 gives 3, and 1.5 added to 4.0 gives 5.5.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <cassert>

using namespace std;

/\*Write the class AddElements here\*/

template <class T>

class AddElements{

public:

T element1,element2;

AddElements(T el){

element1=el;

}

T add(T el2){

return(element1+el2);

}

T concatenate(T el2){

return(element1+el2);

}

};

int main () {

int n,i;

cin >> n;

for(i=0;i<n;i++) {

string type;

cin >> type;

if(type=="float") {

double element1,element2;

cin >> element1 >> element2;

AddElements<double> myfloat (element1);

cout << myfloat.add(element2) << endl;

}

else if(type == "int") {

int element1, element2;

cin >> element1 >> element2;

AddElements<int> myint (element1);

cout << myint.add(element2) << endl;

}

else if(type == "string") {

string element1, element2;

cin >> element1 >> element2;

AddElements<string> mystring (element1);

cout << mystring.concatenate(element2) << endl;

}

}

return 0;

}

21 - Design a class named *Box* whose dimensions are integers and private to the class. The dimensions are labeled: length ll, breadth bb, and height hh.

The default constructor of the class should initialize ll, bb, and hh to 00.

The parameterized constructor *Box(int length, int breadth, int height)* should initialize *Box*'s l,bl,b and hh to length, breadth and height.

The copy constructor *Box*((*Box* BB) should set l,bl,b and hh to BB's l,bl,b and hh, respectively.

Every constructor should increment the global variable *BoxesCreated*.

The destructor should increment the global variable *BoxesDestroyed*.

Apart from the constructor and destructor, the class should have 44 functions:

* *int getLength()* - Return box's length
* *int getBreadth()* - Return box's breadth
* *int getHeight()* - Return box's height
* *long long CalculateVolume()* - Return the volume of the box

Overload the operator << for the class *Box*. *Box* AA << *Box* BB if:

1. A.lA.l < B.lB.l
2. A.bA.b < B.bB.b and A.lA.l==B.lB.l
3. A.hA.h < B.hB.h and A.bA.b==B.bB.b and A.lA.l==B.lB.l

Overload operator <<<< for the class *Box()*.   
If BB is an object of class *Box*:

cout<<Bcout<<B should print B.lB.l, B.bB.b and B.hB.h on a single line separated by spaces.

**Constraints**

0≤l,b,h≤1050≤l,b,h≤105   
Two boxes being compared using the << operator will not have all three dimensions equal.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int BoxesCreated,BoxesDestroyed;

class Box {

private:

long l,b,h;

public:

Box(){

l=0;

b=0;

h=0;

BoxesCreated++;

}

Box(int length, int breadth, int height){

l=length;

b=breadth;

h=height;

BoxesCreated++;

}

Box(Box& B){

l=B.l;

b=B.b;

h=B.h;

BoxesCreated++;

}

~Box()

{

BoxesDestroyed++;

}

int getLength(){

return l;

}

int getBreadth ()

{

return b;

}

int getHeight (){

return h;

}

long long CalculateVolume(){

long long Volume;

Volume=l\*b\*h;

return Volume;

}

bool operator<(Box &B){

if ((l<B.l)||(b<B.b && l==B.l) || (h<B.h && b==B.b && l==B.l) )

{return true;}

else {return false;}

}

friend ostream& operator<<(ostream& out, Box B){

out<<B.l<<" "<<B.b<<" "<<B.h;

return out;

}

};

void check2()

{

int n;

cin>>n;

Box temp;

for(int i=0;i<n;i++)

{

int type;

cin>>type;

if(type ==1)

{

cout<<temp<<endl;

}

if(type == 2)

{

int l,b,h;

cin>>l>>b>>h;

Box NewBox(l,b,h);

temp=NewBox;

cout<<temp<<endl;

}

if(type==3)

{

int l,b,h;

cin>>l>>b>>h;

Box NewBox(l,b,h);

if(NewBox<temp)

{

cout<<"Lesser"<<endl;

}

else

{

cout<<"Greater"<<endl;

}

}

if(type==4)

{

cout<<temp.CalculateVolume()<<endl;

}

if(type==5)

{

Box NewBox(temp);

cout<<NewBox<<endl;

}

}

}

int main()

{

BoxesCreated = 0;

BoxesDestroyed = 0;

check2();

cout<<"Boxes Created : "<<BoxesCreated<<endl<<"Boxes Destroyed : "<<BoxesDestroyed<<endl;

}

22 - A student signed up for nn workshops and wants to attend the maximum number of workshops where no two workshops overlap. You must do the following:

Implement 22 [structures](http://www.cplusplus.com/doc/tutorial/structures/):

1. *struct Workshop* having the following members:
   * The workshop's start time.
   * The workshop's duration.
   * The workshop's end time.
2. *struct Available\_Workshops* having the following members:
   * An integer, nn (the number of workshops the student signed up for).
   * An array of type *Workshop* array having size nn.

Implement 22 [functions](http://www.cplusplus.com/doc/tutorial/functions/):

1. *Available\_Workshops\* initialize (int start\_time[], int duration[], int n)*   
   Creates an *Available\_Workshops* object and initializes its elements using the elements in the start\_time[]start\_time[] and duration[]duration[] parameters (both are of size nn). Here, start\_time[i]start\_time[i]and duration[i]duration[i] are the respective start time and duration for the ithith workshop. This function must return a pointer to an *Available\_Workshops* object.
2. *int CalculateMaxWorkshops(Available\_Workshops\* ptr)*   
   Returns the maximum number of workshops the student can attend—without overlap. The next workshop cannot be attended until the previous workshop ends.

**Note:** An array of unkown size (nn) should be declared as follows:

DataType\* arrayName = new DataType[n];

**Input Format**

Input from stdin is handled by the locked code in the editor; you simply need to write your functions to meet the specifications of the problem statement above.

**Constraints**

* 1≤N≤1051≤N≤105
* 0≤start\_timei≤1030≤start\_timei≤103
* 0≤durationi≤1030≤durationi≤103

**Output Format**

Output to stdout is handled for you.

Your *initialize* function must return a pointer to an *Available\_Workshops* object.   
Your *CalculateMaxWorkshops* function must return maximum number of non-overlapping workshops the student can attend.

**Sample Input**

6

1 3 0 5 5 8

1 1 6 2 4 1

**Sample Output**

*CalculateMaxWorkshops* should return 44.

**Explanation**

The first line denotes nn, the number of workshops.   
The next line contains nn space-separated integers where the ithith integer is the ithith workshop's start time.   
The next line contains nn space-separated integers where the ithith integer is the ithith workshop's duration.

The student can attend the workshops 0,1,3,0,1,3, and 55 without overlap, so*CalculateMaxWorkshops* returns 44 to *main* (which then prints 44 to stdout).

Solution: (With Mircea’s Help)

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

//Define the structs Workshops and Available\_Workshops.

//Implement the functions initialize and CalculateMaxWorkshops

struct Workshop {

int start;

int duration;

int end;

};

struct Available\_Workshops {

int n;

vector<Workshop> workshops;

};

struct sort\_key

{

inline bool operator() (const Workshop& struct1, const Workshop& struct2)

{

return (struct1.end < struct2.end);

}

};

Available\_Workshops\* initialize(int start\_times[], int durations[], int n) {

Available\_Workshops \*ptr = new Available\_Workshops();

ptr->n = n;

ptr->workshops = vector<Workshop>(n);

for (unsigned i=0; i<n; ++i) {

ptr->workshops[i].start = start\_times[i];

ptr->workshops[i].duration = durations[i];

ptr->workshops[i].end = start\_times[i] + durations[i];

}

return ptr;

}

int CalculateMaxWorkshops(Available\_Workshops \*ptr) {

int count = 1;

int current\_end\_index = 0;

// OLD SORT, produces good result but gives timeout (it's slow)

/\*for (unsigned i=0; i<ptr->n-1; ++i) {

for (unsigned j=i+1; j<ptr->n; ++j) {

if (ptr->workshops[j].end < ptr->workshops[i].end) {

Workshop temp = ptr->workshops[i];

ptr->workshops[i] = ptr->workshops[j];

ptr->workshops[j] = temp;

}

}

}\*/

// NEW SORT

std::sort(ptr->workshops.begin(), ptr->workshops.end(), sort\_key());

for (unsigned i=1; i<ptr->n; ++i) {

if (ptr->workshops[i].start >= ptr->workshops[current\_end\_index].end) {

++count;

current\_end\_index = i;

}

}

return count;

}

int main()

{

int n;

cin>>n;

int start\_time[n],duration[n];

for(int i=0;i<n;i++)

{

cin>>start\_time[i];

}

for(int i=0;i<n;i++)

{

cin>>duration[i];

}

Available\_Workshops \* ptr;

ptr=initialize(start\_time,duration,n);

cout<<CalculateMaxWorkshops(ptr)<<endl;

return 0;

}

My Solution: (Only passed 5 test cases) (problem with vector indexing??):

//Define the structs Workshops and Available\_Workshops.

// to be able to sort a vector of pairs only by the second element

struct Workshop {

int StartTime;

int Duration;

};

struct Available\_Workshops {

int n;

Workshop\* AvailWork;

};

//Implement the functions initialize

Available\_Workshops\* initialize (int start\_time[], int duration[], int n){

Available\_Workshops\* temp = new Available\_Workshops;

temp->AvailWork = new Workshop[n];

temp->n = n;

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

temp->AvailWork[i].StartTime = start\_time[i];

temp->AvailWork[i].Duration = duration[i];

}

return temp;

};

// CalculateMaxWorkshops function

int CalculateMaxWorkshops(Available\_Workshops\* ptr) {

//Initializing max\_workshop variable. Temp is a temporary variable to calculate the end time

int max\_workshop=1;

int temp=0;

//Initializing 2 vectors contiaing start and end times

vector<pair<int, int> > General\_Vector;

//Filling out the 2 vectors containing the start and end times

for(int i=0 ; i < ptr->n; i++){

temp = ptr->AvailWork[i].StartTime + ptr->AvailWork[i].Duration;

General\_Vector.push\_back(std::make\_pair(ptr->AvailWork[i].StartTime, temp));

}

//Sorting the elements in the vector by their end times

std:: stable\_sort(General\_Vector.begin(), General\_Vector.end(),

[](const pair<int, int>& lhs, const pair<int, int>& rhs) -> bool {

if (lhs.second == 0)

return true;

return lhs.second < rhs.second; } );

std::vector<pair<int, int> >::iterator it2 = General\_Vector.begin();

//Calculating the max workshop with greedy approach

for(std::vector<pair<int, int> >::iterator it = General\_Vector.begin() + 1 ; it != General\_Vector.end() ; ++it){

int p\_a = it->first;

int p\_b = it2->second;

if (General\_Vector[p\_a] >= General\_Vector[p\_b])

{

max\_workshop++;

it2 = it;

}

}

return max\_workshop;

};

* 23 - [**Problem**](https://www.hackerrank.com/challenges/vector-sort)
* [**Submissions**](https://www.hackerrank.com/challenges/vector-sort/submissions)
* [**Leaderboard**](https://www.hackerrank.com/challenges/vector-sort/leaderboard)
* [**Discussions**](https://www.hackerrank.com/challenges/vector-sort/forum)

You are given NN integers.Sort the NN integers and print the sorted order.  
Store the NN integers in a vector.Vectors are sequence containers representing arrays that can change in size.

* *Declaration:*
* vector<int>v; (creates an empty vector of integers)
* *Size:*
* int size=v.size();
* *Pushing an integer into a vector:*
* v.push\_back(x);(where x is an integer.The size increases by 1 after this.)
* *Popping the last element from the vector:*
* v.pop\_back(); (After this the size decreases by 1)
* *Sorting a vector:*
* sort(v.begin(),v.end()); (Will sort all the elements in the vector)

To know more about vectors, [Click Here](http://www.cplusplus.com/reference/vector/vector/)

**Input Format**

The first line of the input contains NN where NN is the number of integers. The next line contains NN integers.  
**Constraints**  
1<=N<=1051<=N<=105  
1<=Vi<=1091<=Vi<=109, where ViVi is the ithith integer in the vector.

**Output Format**

Print the integers in the sorted order one by one in a single line followed by a space.

**Sample Input**

5

1 6 10 8 4

**Sample Output**

1 4 6 8 10

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

int n;

cin>>n;

int temp;

vector<int> MyVector;

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

cin>>temp;

MyVector.push\_back(temp);

}

std::sort(MyVector.begin(),MyVector.end());

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

cout<<MyVector[i]<<" ";

}

return 0;

}

23 - You are given NN integers.Then you are given 22 queries.First query consists of 11 integer denoting the position which should be removed.Next query consists of 22 integers denoting the range that should be removed.  
The following are some useful vector functions:

* erase(int position):
* Removes the element present at position.<br>
* Ex: v.erase(v.begin()+4); (erases the `$5{th}$` element of the vector v)
* erase(int start,int end):
* Removes the elements in the range from start to end inclusive of the start and exclusive of the end.
* Ex:v.erase(v.begin()+2,v.begin()+5);(erases all the elements from the third element to the fifth element.)

**Input Format**

The first line of the input contains an integer NN.The next line contains NN space separated integers.The third line contains a single integer xx,denoting the position of an element that should be removed from the vector.The fourth line contains two integers aa and bb denoting the range that should be erased from the vector inclusive of a and exclusive of b.

**Constraints**  
1<=N<=1051<=N<=105  
1<=x<=N1<=x<=N  
1<=a<=b<=N−11<=a<=b<=N−1

**Output Format**

Print the size of the vector in the first line and the elements of the vector after the two erase operations in the second line separated by space.

**Sample Input**

6

1 4 6 2 8 9

2

2 4

**Sample Output**

3

1 8 9

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n;

cin>>n;

int x;

int a,b;

int temp;

vector<int> MyVector;

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

cin>>temp;

MyVector.push\_back(temp);

}

cin>>x;

cin>>a;

cin>>b;

MyVector.erase(MyVector.begin() + x - 1);

MyVector.erase(MyVector.begin() + a - 1 , MyVector.begin() + b - 1);

cout<<MyVector.size()<<"\n";

for(int i=0;i<MyVector.size();++i){

cout<<MyVector[i]<<" ";

}

return 0;

}

24 - You are given NN integers in the sorted order. Then you are given QQ queries. In each query you will be given an integer and you have to tell whether that integer is present in the array, if so you have to tell at which index it is present and if it is not present you have to tell the index at which the smallest integer that is just greater than the given number is present.  
Lower bound is a function that can be used with a sorted vector. Learn how to use lower bound to solve this problem by [Clicking Here](http://www.cplusplus.com/reference/algorithm/lower_bound/).

**Input Format**

The first line of the input contains the number of integers NN. The next line contains NNintegers in sorted order. The next line contains QQ, the number of queries. Then QQ lines follow each containing a single integer YY.  
**If the same number is present multiple times, you have to print the first index at which it occurs.**  
**The input is such that you always have an answer for each query.**

**Constraints**  
1<=N<=1051<=N<=105  
1<=Xi<=1091<=Xi<=109,where xixi is ithith element in the array.  
1<=Q<=1051<=Q<=105  
1<=Y<=1091<=Y<=109

**Output Format**

For each query you have to print "Yes"(without the quotes)if the number is present and at which index it is present separated by a space.  
If the number is not present you have to print "No"(without the quotes) followed by the index of the next smallest number just greater than that number.  
You have to output each query in a new line.

**Sample Input**

8

1 1 2 2 6 9 9 15

4

1

4

9

15

**Sample Output**

Yes 1

No 5

Yes 6

Yes 8

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n;

cin>>n;

int q;

int y;

int temp;

vector<int> MyVector;

vector<int> MyQuery;

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

cin>>temp;

MyVector.push\_back(temp);

}

cin>>q;

for(int i=0;i<q;++i){

cin>>y;

MyQuery.push\_back(y);

}

std::vector<int>::iterator low;

for(int i=0;i<q;++i){

low = lower\_bound(MyVector.begin(), MyVector.end(), MyQuery[i]);

if (MyVector[low - MyVector.begin()] == MyQuery[i]) {cout << "Yes " << (low - MyVector.begin()+1) << endl;}

else { cout << "No " << (low - MyVector.begin()+1) << endl;}

}

return 0;

}

* 25 - [**Problem**](https://www.hackerrank.com/challenges/cpp-sets)
* [**Submissions**](https://www.hackerrank.com/challenges/cpp-sets/submissions)
* [**Leaderboard**](https://www.hackerrank.com/challenges/cpp-sets/leaderboard)
* [**Discussions**](https://www.hackerrank.com/challenges/cpp-sets/forum)

Sets are a part of the C++ STL.Sets are containers that store unique elements following a specific order.The mainly used member functions of sets are:

* *Declaration:*
* set<int>st; //Creates a set of integers.
* *Size:*
* int length=s.size(); //Gives the size of the set.
* *Insert:*
* s.insert(x); //Inserts an integer x into the set s.
* *Erasing an element:*
* s.erase(val); //Erases an integer val from the set s.
* *Finding an element:*
* set<int>::iterator itr=s.find(val); //Gives the iterator to the element val if it is found otherwise returns s.end() .
* Ex: set<int>::iterator itr=s.find(100); //If 100 is not present then it==s.end().

To know more about sets [click Here](http://www.cplusplus.com/reference/set/set/). Coming to the problem,you will be give QQqueries.Each query is of one of the three types:  
  
11 xx:Add an element xx to the set.  
22 xx:Delete an element xx from the set. (If the number xx is not present in the set then do nothing).  
33 xx:If the number xx is present in the set then print "Yes"(without quotes) else print "No"(without quotes).

**Input Format**

The first line of the input contains QQ where QQ is the number of queries. The next QQ lines contain 11 query each. Each query consists of two integers yy and xx where yy is the type of the query and xx is an integer.

**Constraints**  
1<=Q<=1051<=Q<=105  
1<=y<=31<=y<=3  
1<=x<=1091<=x<=109

**Output Format**

For queries of type 33 print "Yes"(without quotes) if the number xx is present in the set and if not present then print "No"(without quotes).  
Each query of type 33 should be printed in a new line.

**Sample Input**

8

1 9

1 6

1 10

1 4

3 6

3 14

2 6

3 6

**Sample Output**

Yes

No

No

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <set>

#include <algorithm>

using namespace std;

int main() {

set<int> s;

int q;

cin>>q;

int x;

int y;

for(int i=0;i<q;++i){

cin>>y;

cin>>x;

switch (y){

case 1: s.insert(x);

break;

case 2: s.erase(x);

break;

case 3:

set<int>::iterator itr=s.find(x);

if (itr != s.end()) {cout<<"Yes"<<endl;}

else {cout<<"No"<<endl;}

break;

}

}

return 0;

}

26 - Maps are a part of the C++ STL.Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.The mainly used member functions of maps are:

* Map Template:
* std::map <key\_type, data\_type>
* Declaration:
* map<string,int>m; //Creates a map m where key\_type is of type string and data\_type is of type int.
* Size:
* int length=m.size(); //Gives the size of the map.
* Insert:
* m.insert(make\_pair("hello",9)); //Here the pair is inserted into the map where the key is "hello" and the value associated with it is 9.
* Erasing an element:
* m.erase(val); //Erases the pair from the map where the key\_type is val.
* Finding an element:
* map<string,int>::iterator itr=m.find(val); //Gives the iterator to the element val if it is found otherwise returns m.end() .
* Ex: map<string,int>::iterator itr=m.find("Maps"); //If Maps is not present as the key value then itr==m.end().
* Accessing the value stored in the key:
* To get the value stored of the key "MAPS" we can do m["MAPS"] or we can get the iterator using the find function and then by itr->second we can access the value.

To know more about maps [click Here](http://www.cplusplus.com/reference/map/map/).

You are appointed as the assistant to a teacher in a school and she is correcting the answer sheets of the students.Each student can have multiple answer sheets.So the teacher has QQqueries:

1 X Y1 X Y :Add the marks YY to the student whose name is XX.

2 X2 X: Erase the marks of the students whose name is XX.

3 X3 X: Print the marks of the students whose name is XX. (If XX didn't get any marks print 00.)

**Input Format**

The first line of the input contains QQ where QQ is the number of queries. The next QQ lines contain 11 query each.The first integer, typetype of each query is the type of the query.If query is of type 11, it consists of one string and an integer XX and YY where XX is the name of the student and YY is the marks of the student.If query is of type 22 or 33,it consists of a single string XX where XX is the name of the student.

**Constraints**

1≤Q≤1051≤Q≤105

1≤type≤31≤type≤3

1≤|X|≤61≤|X|≤6

1≤Y≤1031≤Y≤103

**Output Format**

For queries of type 33 print the marks of the given student.

**Sample Input**

7

1 Jesse 20

1 Jess 12

1 Jess 18

3 Jess

3 Jesse

2 Jess

3 Jess

**Sample Output**

30

20

0

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <set>

#include <map>

#include <algorithm>

#include <string>

using namespace std;

int main() {

map<string,int> m;

int q;

cin>>q;

int type;

string name;

int grade;

map<string,int>::iterator itr;

for(int i=0;i<q;++i){

cin>>type;

switch (type){

case 1: cin>>name;

cin>>grade;

itr = m.find(name);

if (itr == m.end()) {m.insert(make\_pair(name, grade));}

else {itr->second += grade;}

break;

case 2: cin>>name;

m.erase(name);

break;

case 3:cin>>name;

itr=m.find(name);

if(itr!=m.end()) {cout<<itr->second<<endl;}

else {cout<<"0"<<endl;}

break;

}

}

return 0;

}

27 - Double ended queue or Deque(part of C++ STL) are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back). The member functions of deque that are mainly used are:

* Deque Template:
* std::deque<value\_type>
* Declaration:
* deque<int> mydeque; //Creates a double ended queue of deque of int type
* Size
* int length = mydeque.size(); //Gives the size of the deque
* Push
* mydeque.push\_back(1); //Pushes element at the end
* mydeque.push\_front(2); //Pushes element at the beginning
* Pop
* mydeque.pop\_back(); //Pops element from the end
* mydeque.pop\_front(); //Pops element from the beginning
* Empty
* mydeque.empty() //Returns a boolean value which tells whether the deque is empty or not

To know more about deque, [click here](http://www.cplusplus.com/reference/deque/deque/)

Given a set of arrays of size NN and an integer KK, you have to find the maximum integer for each and every contiguous subarray of size KK for each of the given arrays.

**Input Format**

First line of input will contain the number of test cases T. For each test case, you will be given the size of array N and the size of subarray to be used K. This will be followed by the elements of the array Ai.

**Constraints**   
1<=T<=10001<=T<=1000   
1<=N<=100001<=N<=10000   
1<=K<=N1<=K<=N   
1<=Ai<=100001<=Ai<=10000 , where AiAi is the ithith element in the array AA.

**Output Format**

For each of the contiguous subarrays of size KK of each array, you have to print the maximum integer.

**Sample Input**

2

5 2

3 4 6 3 4

7 4

3 4 5 8 1 4 10

**Sample Output**

4 6 6 4

8 8 8 10

**Explanation**

For the first case, the contiguous subarrays of size 2 are {3,4},{4,6},{6,3} and {3,4}. The 4 maximum elements of subarray of size 2 are: 4 6 6 4.   
  
For the second case,the contiguous subarrays of size 4 are {3,4,5,8},{4,5,8,1},{5,8,1,4} and {8,1,4,10}. The 4 maximum element of subarray of size 4 are: 8 8 8 10.

Solution:

#include <iostream>

#include <deque>

#include <vector>

using namespace std;

void printKMax(int arr[], int n, int k)

{

int max = arr[0];

int imax = 0;

for (int i=1; i<k; ++i)

{

if (arr[i] > max)

{

max = arr[i];

imax = i;

}

}

cout << max << ' ';

for (int i=k; i<n; ++i)

{

if (arr[i] > max || imax < i-k+1)

{

max = arr[i-k+1];

imax = i-k+1;

for (int j=i-k+2; j<=i; ++j)

{

if (arr[j] > max)

{

max = arr[j];

imax = j;

}

}

}

cout << max << ' ';

}

cout << '\n';

}

int main(){

int t;

cin >> t;

while(t>0) {

int n,k;

cin >> n >> k;

int i;

int arr[n];

for(i=0;i<n;i++)

cin >> arr[i];

printKMax(arr, n, k);

t--;

}

return 0;

}

28 - One of the important topics of Object Oriented Programming is Inheritance.Inheritance allows us to define a class in terms of another class, which allows us in the reusability of the code.Check out the code below:

class Triangle{

public:

void triangle(){

cout<<"I am a triangle\n";

}

};

The class Triangle has a function called triangle(). Now we create a class derived from the base class Triangle called Isosceles.

class Isosceles : public Triangle{

public:

void isosceles(){

cout<<"I am an isosceles triangle\n";

}

};

Now we can create a derived class object and use it to access the functions of the base class.

int main(){

Isosceles isc;

isc.isosceles();

isc.triangle();

return 0;

}

This code will print:

I am an isosceles triangle

I am a triangle

Now write a function in Isosceles class such that the output is as given below.

**Sample Output**

I am an isosceles triangle

In an isosceles triangle two sides are equal

I am a triangle

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

class Triangle{

public:

void triangle(){

cout<<"I am a triangle\n";

}

};

class Isosceles : public Triangle{

public:

void isosceles(){

cout<<"I am an isosceles triangle\n";

}

void description(){

cout<<"In an isosceles triangle two sides are equal\n";

}

//Write your code here.

};

int main(){

Isosceles isc;

isc.isosceles();

isc.description();

isc.triangle();

return 0;

}

* 29 - [**Problem**](https://www.hackerrank.com/challenges/rectangle-area)
* [**Submissions**](https://www.hackerrank.com/challenges/rectangle-area/submissions)
* [**Leaderboard**](https://www.hackerrank.com/challenges/rectangle-area/leaderboard)
* [**Discussions**](https://www.hackerrank.com/challenges/rectangle-area/forum)

Krish just finished reading about rectangles, and he has learned how to compute the area of a rectangle. However, he doesn't know whether his answer is correct.  
He needs your help! Write a class that computes the area of a rectangle.

Create two classes: "Rectangle" and "RectangleArea" with the following specifications:   
Class1: Rectangle   
Member1: width   
Member2: height   
Method1: Display(), to display the dimensions of rectangle.   
Class2: RectangleArea, derived from class Rectangle.   
Method1: Input(), to read the width and height of a rectangle.   
Method2: Display(), overloaded function to display the area of rectangle.

**Input Format**

The first and only line of input contains two space separated integers denoting the width and height of the rectangle.

**Constraints**   
1<=width<=100.1<=width<=100.   
1<=height<=100.1<=height<=100.

**Output Format**

In the first line, print the width and height of the rectangle with a single space in between.   
In the next line, print the area of the rectangle.

**Sample Input**

10 5

**Sample Output**

10 5

50

**Explanation**

Area of rectangle is calculated as: width\*height = 10\*5 = 50.

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <cassert>

using namespace std;

class Rectangle{

public:

int width;

int height;

void Display(){

cout<<width<<" "<<height<<"\n";

}

};

class RectangleArea: public Rectangle{

public:

void Input(){

cin>>width;

cin>>height;

}

void Display(){

int area;

area=width\*height;

cout<<area;

}

};

int main() {

RectangleArea r\_area;

r\_area.Input();

Rectangle \*r;

r=&r\_area;

r->Display();

r\_area.Display();

return 0;

}

30 - **Solve the problem Inheritance-Introduction before solving this problem.**

In the previous problem we learnt about Inheritance and how can a derived class object use the member functions of the base class.

In this we will see what multi-level inheritance is. Suppose we have a class A which is the base class and we have a class B which is derived from class A and we have a class C which is derived from class B, we can access the functions of both class A and class B by creating an object for class C. This is called Multi-Level inheritance. (B inherits A and C inherits B.)

Now create a class called Equilateral which inherits from Isosceles and should have a function such that the output is as given below.

**Sample Output**

I am an equilateral triangle

I am an isosceles triangle

I am a triangle

Solution:

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

class Triangle{

public:

void triangle(){

cout<<"I am a triangle\n";

}

};

class Isosceles : public Triangle{

public:

void isosceles(){

cout<<"I am an isosceles triangle\n";

}

};

class Equilateral : public Isosceles{

public:

void equilateral(){

cout<<"I am an equilateral triangle\n";

}

};

int main(){

Equilateral eqr;

eqr.equilateral();

eqr.isosceles();

eqr.triangle();

return 0;

}

31 - You are given three classes *A, B* and *C*. All three classes implement their own version of *func*.

In class *A*, *func* multiplies the value passed as a parameter by 22:

class A

{

protected:

void func(int & a)

{

a=a\*2;

}

}

In class *B*, *func* multiplies the value passed as a parameter by 33:

class B

{

protected:

void func(int & a)

{

a=a\*3;

}

}

In class *C*, *func* multiplies the value passed as a parameter by 55:

class C

{

protected:

void func(int & a)

{

a=a\*5;

}

}

You are given a class *D*:

class D

{

int val;

public:

//Initially, val is 1

D()

{

val=1;

}

//Implement this function

void update\_val(int new\_val)

{

}

}

You need to modify the class *D* and implement the function update\_val which sets *D*'s *val* to*new\_val* by manipulating the value by only calling the *func* defined in classes *A, B* and *C*.

**It is guaranteed that *new\_val* has only**2,32,3**and**55**as its prime factors.**

**Input Format**

Implement class *D*'s function *update\_val*. This function should update *D*'s *val* only by calling*A, B* and *C*'s *func*.

**Constraints**

1≤1≤ *new\_val* ≤10000≤10000   
**Note:** The *new\_val* only has 2,32,3 and 55 as its prime factors.

**Sample Input**

*new\_val* =30=30

**Sample Output**

*A*'s *func* will be called once.   
*B*'s *func* will be called once.   
*C*'s *func* will be called once.

**Explanation**

Initially, *val* =1=1.

*A*'s *func* is called once:

val = val\*2

val = 2

*B*'s *func* is called once:

val = val\*3

val = 6

*C*'s *func* is called once:

val = val\*5

val = 30

Solution:

#include<iostream>

using namespace std;

int callA=0;

int callB=0;

int callC=0;

class A

{

protected:

void func(int & a)

{

a=a\*2;

callA++;

}

};

class B

{

protected:

void func(int & a)

{

a=a\*3;

callB++;

}

};

class C

{

protected:

void func(int & a)

{

a=a\*5;

callC++;

}

};

class D: protected A,B,C

{

int val;

public:

//Initially val is 1

D()

{

val=1;

}

//Implement this function

void update\_val(int new\_val)

{

int a = new\_val;

while ( a %2 == 0)

{

a = a/2;

A::func(val);

}

while ( a % 3 == 0)

{

a = a/3;

B::func(val);

}

while ( a % 5 == 0)

{

a = a/5;

C::func(val);

}

}

//For Checking Purpose

void check(int); //Do not delete this line.

};

void D::check(int new\_val)

{

update\_val(new\_val);

cout<<"Value = "<<val<<endl<<"A's func called "<<callA<<" times "<<endl<<"B's func called "<<callB<<" times "<<endl<<"C's func called "<<callC<<" times"<<endl;

}

int main()

{

D d;

int new\_val;

cin>>new\_val;

d.check(new\_val);

}