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

/**
    2019 1b
*/

class A{
public:
    static int a;
    static int getA(){
        return a;
    }
};

int A::a = 257;

int main()
{
    cout << A::a << endl; // 257 /** a must be public */
    cout << A::getA() << endl; // 257
    return 0;
}
```

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

/**
    2019 2a
*/

class A{
    int a;
public:
    A(int aa = 2){ a = aa; }
    void show(){ cout << a << endl; }

};

int main()
{
    A a(5); a.show(); // 5
    A b; b.show(); // 2. Working like default constructor

    return 0;
}
```

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

/**
    2019 2a
*/

class A{
public:
    A(){ cout << "C" << endl; }
    ~A(){ cout << "D" << endl; }
};

int main()
{
    // explicit call to constructor
    A a = A(); // C
    A(); // CD

    // explicit call to dest
    a.~A(); // D

    // D
    return 0;
}
```

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

/**
    2019 2a
*/

class Complex{
    int real, img;
public:
    Complex(int r, int i){
        real = r;
        img = i;
    }
    Complex(){}
    Complex operator++(){ // ++a
        real++;
        img++;
        return *this;
    }
    Complex operator++(int){ //a++
        real++;
        img++;
        Complex cmp(real-1,img-1);
        return cmp;
    }
    void show(){ cout << real << "+" << img << "i" << endl; }

};

int main()
{
    Complex c1(2,3);
    ++c1;
    c1.show(); // 3+4i

    Complex c2;
    c2 = c1++;
    c1.show(); // 4+5i
    c2.show(); // 3+4i

    return 0;
}
```

```
#ifndef COMPLEXNUMBER_H
#define COMPLEXNUMBER_H

template<class T>
class Complex;

#endif // COMPLEXNUMBER_H
```

```cpp
#include<iostream>
#include "ComplexNumber.h"

template<class T>
class Complex{
    T real, img;
public:
    Complex(){}
    Complex(T r, T i){ real = r; img = i; }
    void show(){ std::cout << real << "+" << img << "i "; }

    Complex operator+(Complex& cmp){
        T rl = real + cmp.real;
        T im = img + cmp.img;

        Complex cm(rl,im);
        return cm;
    }

};
```

```cpp
#include<iostream>
#include "ComplexNumber.h"
#include "ComplexNumber.cpp" // program should run without
this line, but not working, may be because of building issue.
idk :)

using namespace std;


template<class T>
class Matrix{
    Complex<T> mat[2][2];
public:
    Matrix(){}
    Matrix(int *arr){
        mat[0][0] = Complex<T>(arr[0],arr[1]);
        mat[0][1] = Complex<T>(arr[2],arr[3]);
        mat[1][0] = Complex<T>(arr[4],arr[5]);
        mat[1][1] = Complex<T>(arr[6],arr[7]);
    }

    Matrix<int> operator+(Matrix mt){
        Matrix<int> ans;
        for(int i=0;i<2;i++){
            for(int j=0;j<2;j++){
                ans.mat[i][j] = mat[i][j] + mt.mat[i][j];
            }
        }
        return ans;
    }

    void show(){
        for(int i=0;i<2;i++){
            for(int j=0;j<2;j++){
                mat[i][j].show();
            }
            cout << endl;
        }
        cout << endl;
    }

};

int main(){


    int arr[] = {0,0,1,1,2,2,3,3};
    Matrix<int> mat1(arr);
    int brr[] = {1,1,2,2,3,3,4,4};
    Matrix<int> mat2(brr);

    mat1.show();
```

```
    mat2.show();

    Matrix<int> mat = mat1+mat2;
    mat.show();
}
```

```cpp
#include<iostream>

using namespace std;

/**
    2019 3c
    insertion operator(<<) can't be overloaded using member
function
    left shift operator(<<) can be overloaded using member
function
*/

class Complex{
    int real, img;
public:
    Complex(){}
    Complex(int r, int im){
        real = r;
        img = im;
    }
    void show(){ cout << real << "+" << img << "i" << endl; }
    Complex operator<<(int r){
        int rr = real<<r;
        int im = img<<r;
        Complex cmp(rr,im);
        return cmp;
    }
};


int main()
{
    Complex c1(5,9);
    Complex c2;
    c2 = c1 << 1;
    // 5<<1 = 10, 9<<1 = 18
    c1.show(); // 5+9i
    c2.show(); // 10+18i

    return 0;
}
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 4b
*/


void calculateAndWrite(){
    ifstream fin;
    fin.open("input.txt");
    ofstream ofn("output.txt");
    int a,b;
    while(fin >> a >> b){
        ofn << (a+b) << endl;
    }
    fin.close();
    ofn.close();
}

void read(){
    ifstream fin("output.txt");
    int a;
    while(fin >> a){
        cout << a << endl;
    }
}

int main()
{

    calculateAndWrite();
    read();
    return 0;
}
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 4b
*/

template<class T>
void fun(const T &x)//const na dile error
{
    static int count = 0;
    cout << "x = " << x << " count = " << count++ << endl;
    return;
}

int main()
{
    fun<int> (1);
    fun<int> (2);
    fun<double>(1.1);
    return 0;
}
/**
    output:
    x = 1 count = 0
    x = 2 count = 1
    x = 1.1 count = 0
*/
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 4d
*/

class A{
public:
    int a,b;
    A(int aa, int bb):a(aa),b(bb){ }
    void show(){
        cout << a << " - " << b << endl;
    }
};

int main()
{
    A a(4,5);
    a.show(); // 4 - 5
    return 0;
}
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 6b
    helps to set different characteristics according to child
class
*/

class Vehicle{
protected:
    int price = 0;
    virtual void setPrice()=0;
public:
    void showPrice(){
        cout << price << endl;
    }
};

class Car : public Vehicle{
public:
    void setPrice(){ price = 500; }
};

class GhorarGari : public Vehicle{
public:
    void setPrice(){ price = 5000000; }
};

int main()
{
    Car car;
    car.setPrice();
    car.showPrice(); // 500

    GhorarGari gg;
    gg.setPrice();
    gg.showPrice(); // 5000000

    return 0;
}
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 6c
    compile time: function & operator overloading
    runtime: virtual function
*/

class Vehicle
{
protected:
    int price = 0;
    void setPrice() {}
public:
    virtual void showName()
    {
        cout << "Unknown" << endl;
    }

public:
    void showPrice()
    {
        cout << price << endl;
    }
};

class GhorarGari : public Vehicle
{
public:
    void setPrice()
    {
        price = 50000;
    }
    void setPrice(int incr)
    {
        price = 50000 + incr;
    }
    void setPrice(float disc)
    {
        price = 50000;
        price -= (price*(disc/100));
    }
    void showName()
    {
        cout << "Ghorar gari" << endl;
    }
};

Vehicle* random()
{
```

```
    switch(rand()%3)
    {
    case 0:
        return new Vehicle();
    case 1:
        return new GhorarGari();
    default:
        return NULL;
    }
}

int main()
{
    /** compile time, because at compile time, compiler
determine which one to be called */
    GhorarGari gg;
    gg.setPrice();
    gg.showPrice(); // 5000000

    gg.setPrice(5000);
    gg.showPrice(); // 55000

    gg.setPrice(5.0f); // 47500
    gg.showPrice();

    /** runtime */

    Vehicle *ptr;

    for(int i=0; i<8; i++)
    {
        ptr = random();
        if(ptr != NULL)
        {
            ptr->showName(); // showName() is called based on
the object ptr is pointing at runtime
        }

    }

    return 0;
}
```

```cpp
#include<bits/stdc++.h>

using namespace std;

/**
    2019 7c
*/

class Base
{
public:
    int a = 20;
    virtual void show(){ cout << a << endl; }
};

class Derived : public Base
{
public:
    int b = 10;
    void show(){ cout << b << endl; }
};

int main()
{
    Base *bp, bObj;
    Derived *dp, dObj;

    bObj.show(); // 20
    dObj.show(); // 10

    /** derived object to base object */
    dObj.a = 54;
    bObj = dynamic_cast<Base&>(dObj);
    bObj.show(); // 54

    /** derived pointer to base pointer */
    dObj.b = 45;
    bp = dynamic_cast<Base*>(&dObj);
    bp->show(); // 45

    /** base pointer to derived pointer */
    dObj.b = 57;
    bp = &dObj;
    dp = dynamic_cast<Derived*>(bp);
    dp -> show(); // 57

    /** base pointer object to derived object */
    bp = &dObj;
    dObj.a = 100;
    dObj.b = 200;
    dObj = dynamic_cast<Derived&>(*bp); //bad cast for bObj
    dObj.show(); // 2002019
```

```
    dObj.show(); // 2002019

    return 0;
}
```

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

/**
    2019 8c
    Array index out of bound exception for arr[10]
*/

int main()
{
    int arr[10] = {10};
    try{
        for (int i= 0; i <= 10; i++){
            if(i>=10) throw i;
            cout << arr[i] << endl;
        }
    }
    catch(int x){
        cout << "index out of bound" << endl;
    }

    //continue can be used also
}
```

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

/**
    2019 8d
*/

void test() throw(int,float){
    int arr[10] = {10};
    try{
        for (int i= 0; i <= 12; i++){
            if(i == 10) {
                continue;
                throw i;
            }
            else if(i==11) {
                continue;
                throw 10.0f;
            }
            else if(i==12) throw 'c';
            cout << arr[i] << endl;
        }
    }
    catch(int x){
        cout << "index out of bound" << endl;
    }
    catch(float x){// after adding first 'continue'
        cout << "out of bound again" << endl;
        throw x;
    }
    catch(char c){ // after adding second 'continue'
        cout << "Executed" << endl;
        //program terminated
        throw ; // rethrowing
    }
}

int main()
{
    try{
        test();
    }
    catch(float f){
        cout << "float -> " << f << endl;
    }
    catch(char c){
        cout << "never executed" << endl;
    }
    //continue can be used also
}
```