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

using namespace std;


//2020 3a

class complexx{
    int real, img;
public:
    complexx(int r, int im){
        real = r;
        img = im;
    }

    complexx operator+(complexx& cmp){
        int r = real + cmp.real;
        int im = img + cmp.img;
        complexx cmp2(r,im);
        return cmp2;
    }

    complexx operator++(int){
        complexx tmp(real,img);
        real++;
        img++;
        return tmp;
    }

    void show(){
        cout << real << "+" << img << "i" << endl;
    }

};

int main()
{
    complexx c1(2,3), c2(4,5);
    c1.show(); // 2+3i
    c2.show(); // 4+5i

    complexx c3 = c1+c2;
    c3.show(); // 6+8i

    complexx c4 = c3++;
    c4.show(); // 6+8i

    c3.show(); // 7+9i
}
```

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

/**
    2020 3b
*/

/** not possible because for cin >> a;
    it is interpreted as cin.operator>>(a); for this, we have to define a function
    named operator>>(Complex&) inside cin class, which is not possible
*/

class Complex
{
    int r, im;
public:
    friend istream& operator>>(istream&, Complex&);
    Complex() {}
    Complex(int rr, int imm)
    {
        r = rr;
        im = imm;
    }
    void show()
    {
        cout << r << "+" << im <<"i" << endl;
    }
};
istream& operator>>(istream &strm, Complex& cmp)
{
    strm >> cmp.r >> cmp.im;
    return strm;
}
int main()
{
    Complex cmp1, cmp2;
    cin >> cmp1 >> cmp2;

    //input 4 5 5 45
    cmp1.show(); // 4+5i
    cmp2.show(); // 5+45i
}
```

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

using namespace std;

//2020 4a

class A
{
public:
    int a;
};

class B : virtual public A
{
public:
    int b;
};

class D : virtual public A
{
public:
    int d;
};
class E : virtual public A
{
public:
    int e;
};
class F : virtual public A
{
public:
    int f;
};
class C : virtual public A
{
public:
    int c;
};
class G : public D, virtual public E
{
public:
    int g;
};
class H : virtual public E, public F
{
public:
    int h;
};
class I : public B, public G, public H, public C
{
public:
    int i;
```

```
};

int main()
{
    I cc;
    cc.a = 6;
    cc.e = 9;
    cout << cc.a << " - " << cc.e << endl;// 6 - 9
    return 0;
}
```

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

using namespace std;

/**
    2020 4b
*/

/**
    there are 2 abnormalities
    1. program will crash becuase 'division by zero' for data[3].
        : use 'continue' or 'try-catch'
    2. int by int always generate int result. So 1/data[i] will always be zero(0) for given array
        : convert '1' or data[i] into float before dividing
*/


int main()
{
    int data[] = {10,340,200,0,50,60};
    float sum = 0;

    // using continue
    for(int i=0; i < sizeof(data)/sizeof(int); i++){
        if(data[i]==0) continue;
        sum += 1.0f/data[i];
    }
    cout << sum << endl; // 0.144608

    // using try-catch

    sum = 0;
    try{
        for(int i=0; i<sizeof(data)/sizeof(int); i++){
            if(data[i]==0) throw 0;
            sum += 1.0f/data[i];
        }
    }
    catch(int x){
        cout << "division by zero" << endl;
        sum = 0;
    }
    cout << sum << endl;

    return 0; // 0
}
```

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

using namespace std;

/**
    2020 5c

    Output: Base2
    Ambiguity can't occur in single inheritance.
    Because: Derived class function will override base class
function
*/

class Basel
{
public:
    void display() { cout << "Basel\n"; }
};

class Base2 : Basel
{
public:
    void display() { cout << "Base2\n"; }
};

class Derived : public Base2
{
public:
    void displays () { cout << "Derived\n"; }
};

int main()
{
    Derived dd;
    dd.display(); // Base 2
    return 0;
}
```

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

using namespace std;

/**
    2020 6a
*/

class DDD;

class DD{
protected:
    double a,b;
public:
    DD(double aa, double bb){
        a = aa;
        b = bb;
    }

    void cal_dis_distance(DD p1, DD p2){
        double da = p2.a - p1.a;
        double db = p2.b - p1.b;
        double ans = sqrt(da*da + db*db);
        cout << ans << endl;
    }
};

class DDD : public DD{
    double c;
public:
    DDD(double aa, double bb, double cc) : DD(aa,bb){
        c = cc;
    }
    void cal_dis_distance(DDD p1, DDD p2){

        double da = p2.a - p1.a;
        double db = p2.b - p1.b;
        double dc = p2.c - p1.c;

        double ans = sqrt(da*da + db*db + dc*dc);
        cout << ans << endl;
    }
};


int main()
{
    DD p1(0,0), p2(3,4);
    DDD d1(0,0,0), d2(2,3,4);

    DD *ptr = &p1;
    ptr->cal_dis_distance(p1,p2); // 5
```

```
    ptr = &d1;

    ptr->cal_dis_distance(d1,d2); /** 3.60555 wrong ans,
virtual e kaj hobe na & hocche o na (: */

    /** ar kono option paile bolis */
    ( static_cast<DDD*>(ptr) ) ->cal_dis_distance(d1,d2); //
5.38516

//    ( dynamic_cast<DDD*>(ptr) ) ->cal_dis_distance(d1,d2);
// 5.38516, virtual lagbe hudai

    return 0;
}
```

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

/**
    2020 6b
*/

class Creature
{
public:
    virtual void eat()
    {
        cout << "All creature eats\n";
    }
};

class Human : public Creature
{
public:
    int a = 5;
    void eat()
    {
        cout << "All human eats\n";
    }
};

int main()
{
    Creature *pp, oCr;
    Human *hum, oHum;

    /** dynamic_cast */

    /** base pointer to Derived pointer */
    // possible is base pointer is point to derived object
    pp = &oHum;
    hum = dynamic_cast<Human*>(pp);
    hum -> eat(); /** all human eats */

    /** derived object to base object */
    oCr = dynamic_cast<Creature&>(oHum);
    oCr.eat(); // all creatures eats

    /** derived pointer to base pointer */
    pp = dynamic_cast<Creature*>(&oHum);
    pp->eat(); // all human eats


    /** CONFUSED nicher sob. bujhle bujhas*/

    /** static cast */
```

```
    //pointer to pointer, derived to base
    pp = static_cast<Creature*>(&oHum);
    pp ->eat(); // All Humans eats, virtual

    //object to object
    oCr = static_cast<Creature>(oHum);
    oCr.eat(); // all creature eats

    /** reinterpret_cast*/

    pp = reinterpret_cast<Creature*>(&oHum);
    pp->eat(); // all humans eats, virtual

    hum = reinterpret_cast<Human*>(&oCr);
    hum->eat(); // all creates eats



    return 0;
}
```