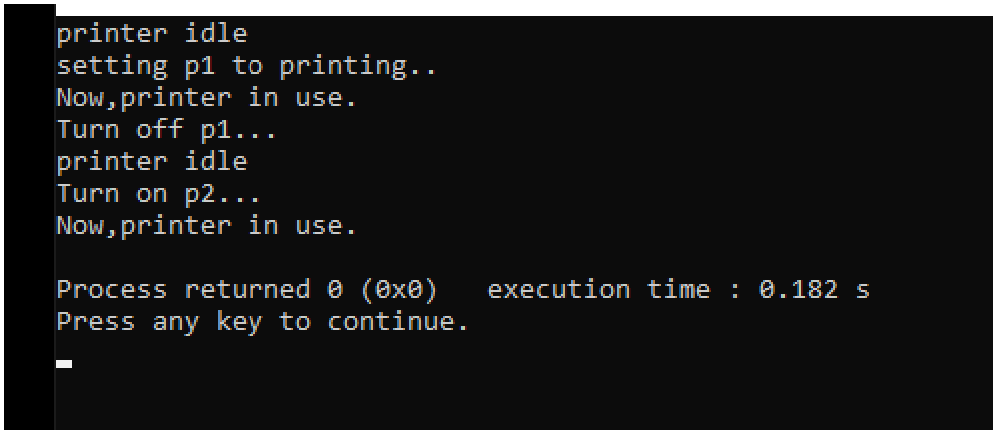
1] Imagine a situation in which two classes, called pr1 and pr2, shown here, share one printer. Further, imagine that other parts of your program need to know when the printer is in use by an object of either of these two classes. Create a function called inuse() that returns true when the printer is being used by either and false otherwise. Make this function a friend of both pr1 and pr2.(page_113)



```
printer idle
setting p1 to printing..
Now,printer in use.
Turn off p1...
printer idle
Turn on p2...
Now,printer in use.

Process returned 0 (0x0)   execution time : 0.182 s
Press any key to continue.
```

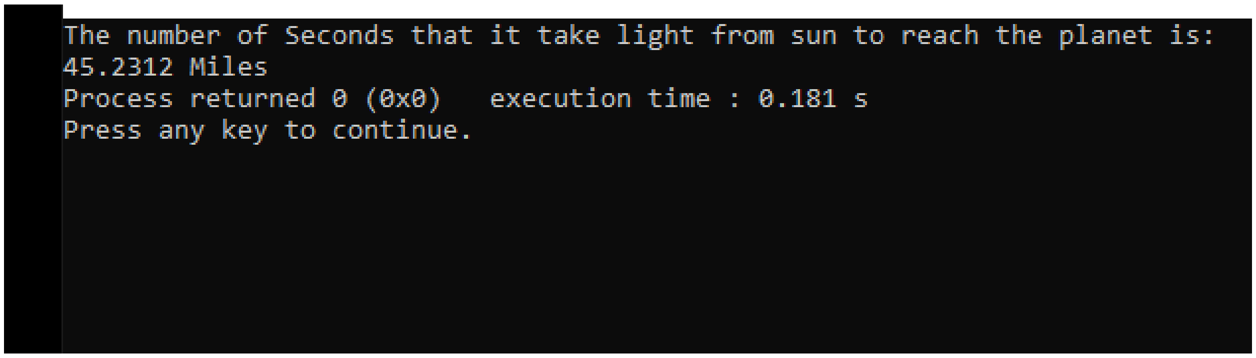
```
#include<iostream>
using namespace std;
    class pr2;
    class pr1{
        int printing;
    public:
        pr1( ){printing=0;}
        void set_print(int status){printing = status;}
        friend int inuse(pr1 o1,pr2 o2);
        };


    class pr2{
        int printing;
    public:
        pr2( ){printing = 0;}
        void set_print(int status) {printing=status;}
        friend int inuse(pr1 o1,pr2 o2);
        };
```

```
int inuse(pr1 o1,pr2 o2)
{
    if(o1.printing || o2.printing) return 1;
    else return 0;
}
int main()
{  pr1 p1;
  pr2 p2;
if(!inuse(p1,p2))cout<<"printer idle\n";
cout<<"setting p1 to printing..\n";
p1.set_print(1);
if(inuse(p1,p2))cout<<"Now,printer in use.\n";

cout<<"Turn off p1...\n";
p1.set_print(0);
if(!inuse(p1,p2))cout<<"printer idle\n";

cout<<"Turn on p2...\n";
p2.set_print(1);
if(inuse(p1,p2))cout<<"Now,printer in use.\n";

return 0;    }
```

2] Given this class fragment, class samp{ double *p; public: samp(double d){ p=(double *)malloc(sizeof(double)); if(!p) exit(1); //allocation error *p=d;} ~samp() {free(p);} // ... }; // ... samp ob1(123.09), ob2(0.0); // ... ob2 = ob1; what problem is caused by the assignment of ob1 to ob2?(page_114,2)

The trouble with the assignment of ob1 to ob2 is that the memory pointed to by ob2's initial value of p is now lost because this value is overwritten by the assignment. This memory thus become impossible to free, and the memory pointed to by ob1's p is freed twice when it is destroyed-possibly causing damage to the dynamic allocation system.

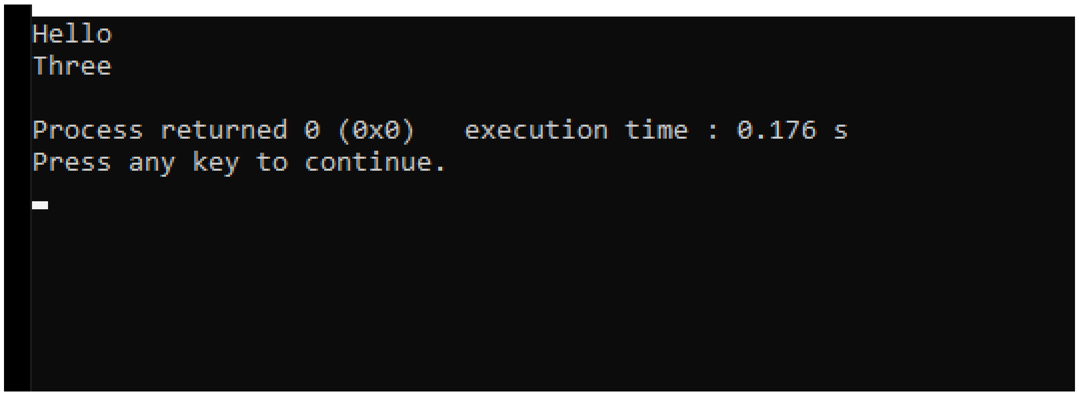
3] Given this class, class planet{ int moons; double dist_from_sun; // in miles double diameter; double mass; public: // ... double get_miles() {return dist_from_sun;} }; Create a function called light() that takes as an argument an object of type planet and returns the number of seconds that it takes light from the sun to reach the planet.(Assume that light travels at 186,000 miles per second and that dist_from_sun is specified in miles.) (page_114,3)



```
The number of Seconds that it take light from sun to reach the planet is:
45.2312 Miles
Process returned 0 (0x0)   execution time : 0.181 s
Press any key to continue.
```

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

    class planet{
        int moons;
        double dist_from_sun;
        double diameter;
        double mass;

    public:
        planet(int a,double b,double c, double d){
            moons=a;
            dist_from_sun=b;
            diameter=c;
            mass=d;
        }
        double get_miles( ){
        return dist_from_sun;
        }
        double light(planet p);
    };

    double planet::light(planet p){
    return (p.get_miles( )/186000);
    }
    int main( ){
        planet p1(1,8413000,45000,155641);
        cout<<"The number of Seconds that it take light from sun to
reach the planet is: \n";
        cout<<p1.light(p1)<<" Miles";
        return 0;
    }
```

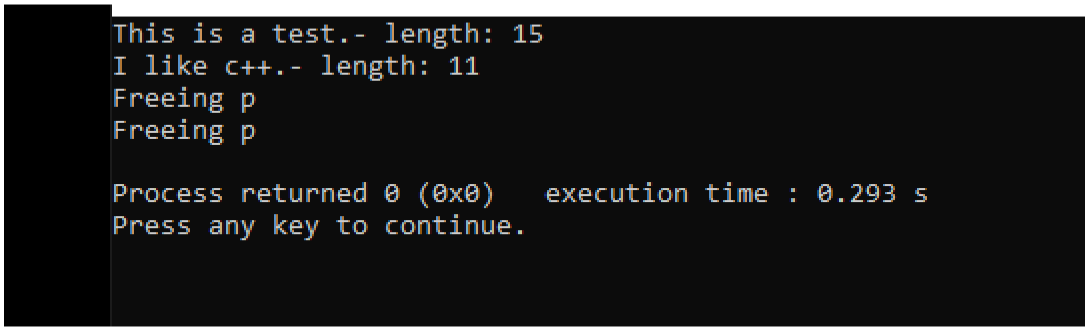
4] What is wrong with the following program? Show how it can be fixed by using reference parameter. //This program has an error. (page_148)



```
Hello
Three

Process returned 0 (0x0)   execution time : 0.176 s
Press any key to continue.
```

```
#include<iostream>
#include<cstring>
#include<cstdlib>
using namespace std;
    class strtype{
        char *p;
    public:
        strtype(char *s);
        ~strtype(){delete [] p;}
        char *get()
     {return p;}
};
    strtype::strtype(char *s){
        int l;
        l=strlen(s)+1;
        p=new char [1];
        if(!p){
            cout<<"Allocation error\n";
            exit(1);
        }
        strcpy(p,s);
    }
    void show(strtype &x)
    {        char *s;
        s=x.get();
        cout<<s<<"\n";
    }

    int main()
    {
        strtype a("Hello"),b("Three");
        show(a);
        show(b);
        return 0;
}
```

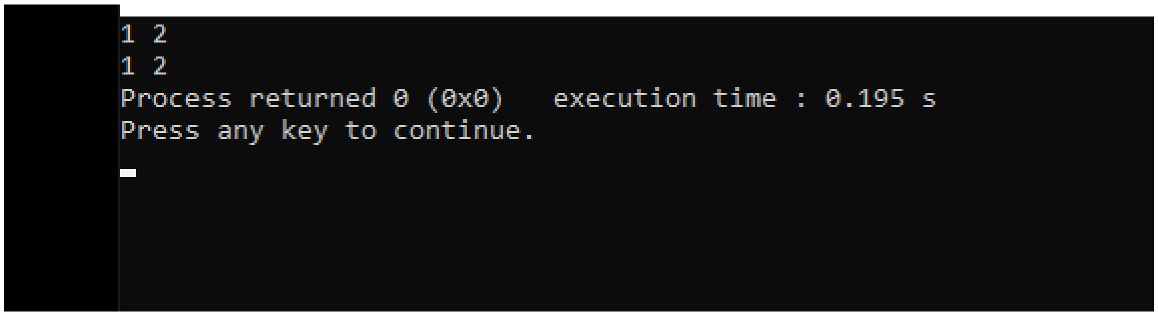
5] A strtype class was created that dynamically allocated space for a string. Rework the strtype class (shown here for your convenience) so it uses new and delete. (page_157)



```
This is a test.- length: 15
I like c++.- length: 11
Freeing p
Freeing p

Process returned 0 (0x0)   execution time : 0.293 s
Press any key to continue.
```

```
#include<iostream>

#include<cstring>

#include<cstdlib>

using namespace std;


    class strtype{

        char *p;

        int len;

    public:

        strtype(char *ptr);

        ~strtype( );

        void show( );

        };


    strtype::strtype(char *ptr)

    {         len=strlen(ptr);

        p=new char [len+1];

        if(!p){

            cout<<"Allocation error\n";

            exit(1);

        }
    strcpy(p,ptr); }
        strtype::~strtype()
    {
        cout<<"Freeing p\n";
        delete [] p;
    }
    void strtype::show()
    {
        cout<<p<<"- length: "<<len;
        cout<<"\n";
    }
    int main()
    {   strtype s1("This is a test."),s2("I like c++.");
        s1.show();
        s2.show();
        return 0;  }
```

6] Explain what is wrong with the following program and then fix it. //This program contains an error. (page_176)

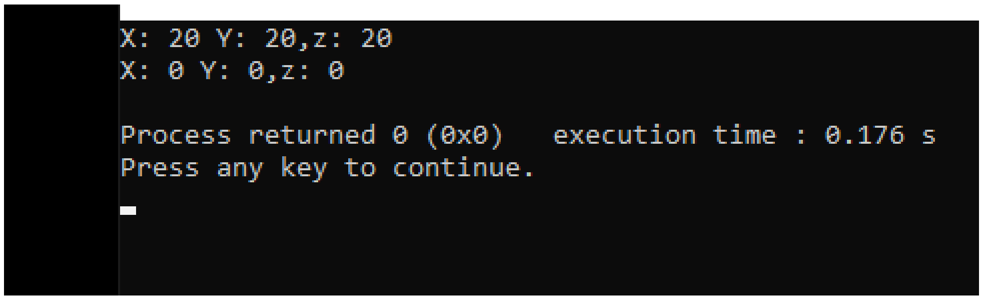


```
1 2
1 2
Process returned 0 (0x0)   execution time : 0.195 s
Press any key to continue.
```

```
#include<iostream>
#include<cstdlib>
using namespace std;
    class myclass{
        int *p;
```

```
public:
    myclass(int i);
    myclass(const myclass &o);
    ~myclass(){delete p;}
    friend int getval(myclass o);
};

myclass:: myclass(int i)
{
    p=new int;
    if(!p){
        cout<<"Allocation error\n";
        exit(1);
    }
    *p=i;
}

myclass::myclass(const myclass &o)
{
    p=new int;
    if(!p){
        cout<<"Allocation error\n";
        exit(1);
    }
    *p=*o.p;
}

int getval(myclass o)
{
    return *o.p;
}


int main()
{
    myclass a(1),b(2);
    cout<<getval(a)<<" "<<getval(b);
    cout<<"\n";
    cout<<getval(a)<<" "<<getval(b);
    return 0;     }
```

7] Overload the + for the three_d class in question 2 so that it accepts the following types of oprations: ob + int; int + ob; [ques-2] class three_d{ int x,y,z; public: three_d(int i,int j,int k){ x=I; y=j; z=k; } three_d() { x=0; y=0; z=0;} void get(int &amp;i, &amp;j, &amp;k) { i=x; j=y; k=z; } }; (page_228)

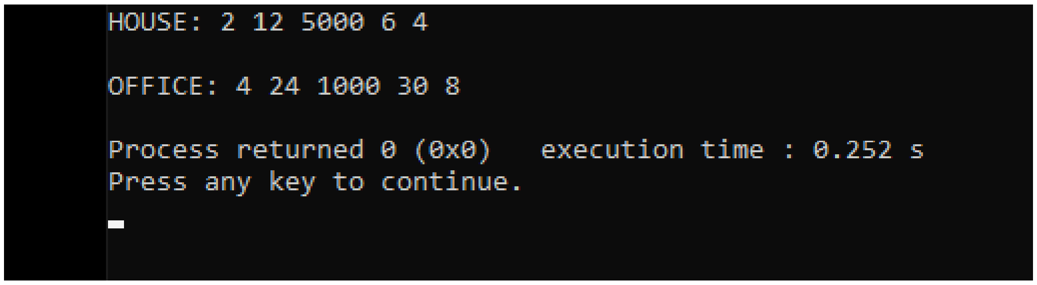


```
X: 20 Y: 20,z: 20
X: 0 Y: 0,z: 0

Process returned 0 (0x0)   execution time : 0.176 s
Press any key to continue.
```

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


    class three_d{
        int x,y,z;
    public:
        three_d(int i,int j,int k)
        {
        x=i;y=j;z=k;
        }
        three_d(){x=0;y=0;z=0;}
        void get(int &i,int&j,int &k)
        {
        i=x;j=y;k=z;
        }
        friend three_d operator+(three_d ob,int i);
        friend three_d operator+(int i,three_d ob);
    };

    three_d operator +(three_d ob,int i)
    {
        three_d temp;
        temp.x=ob.x+i;
        temp.y=ob.y+i;
        temp.z=ob.z+i;
        return temp;
    }
    three_d operator +(int i,three_d ob)
    {
        three_d temp;
        temp.x=ob.x+i;
        temp.y=ob.y+i;
        temp.z=ob.z+i;
        return temp;
    }
    int main()
    {
        three_d o1(10,10,10);
        int x,y,z;
        o1=o1+10;
        o1.get(x,y,z);
```

```
        cout<<"X: "<<x<<" Y: "<<y;
        cout<<",z: "<<z<<"\n";

        o1=-20+o1;
        o1.get(x,y,z);
        cout<<"X: "<<x<<" Y: "<<y;
        cout<<",z: "<<z<<"\n";

        return 0;
    }
```

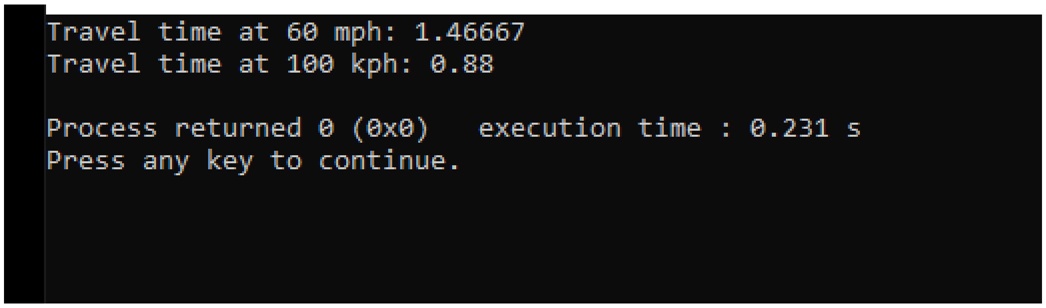
8] Create a generic base class called building that stores the number of floors a building has, the number of rooms and its total square footage. Create a derived class called house that inherits building and also stores the number of bedrooms and the number of bathrooms. Next, create a derived class called office that inherits building and also stores the number of fire extinguishers and the number of telephones. (page_262)



```
HOUSE: 2 12 5000 6 4

OFFICE: 4 24 1000 30 8

Process returned 0 (0x0)   execution time : 0.252 s
Press any key to continue.
```

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

    class building{
    protected:
        int floors;
        int rooms;
        double footage;
    };
    class house:public building{
        int bedrooms;
        int bathrooms;
        public:
        house(int f,int r,double ft,int be,int ba){
            floors=f;
            rooms=r;
            footage=ft;
            bedrooms=be;
            bathrooms=ba;
        }

        void showh(){
            cout<<floors<<" "<<rooms<<" "<<footage<<" "<<bedrooms<<"
"<<bathrooms<<endl;
        }
    };

    class office:public building{
        int extingusihers;
        int telephones;
        public:
        office(int f,int r,double ft,int ex,int te){
            floors=f;
            rooms=r;
            footage=ft;
            extingusihers=ex;
            telephones=te;
        }
        void showo(){
            cout<<floors<<" "<<rooms<<" "<<footage<<"
"<<extingusihers<<" "<<telephones<<endl;

        }

    };


    int main()

    {

        house h_ob(2,12,5000,6,4);

        office o_ob(4,24,1000,30,8);


        cout<<"HOUSE: ";

        h_ob.showh();

        cout<<"\nOFFICE: ";

        o_ob.showo();


        return 0; }
```

9] Write a program that creates a base class called dist that stores the distance between two points in a double variable. In dist, create a virtual function called trav_time() that outputs the time it takes to travel that distance, assuming that the distance is in miles and the speed is 60 miles per hour. In a derived class called metric, override trav_time() so that it outputs the travel time assuming that the distance is in kilometres and the speed is 100 kilometers per hour. (page_357)

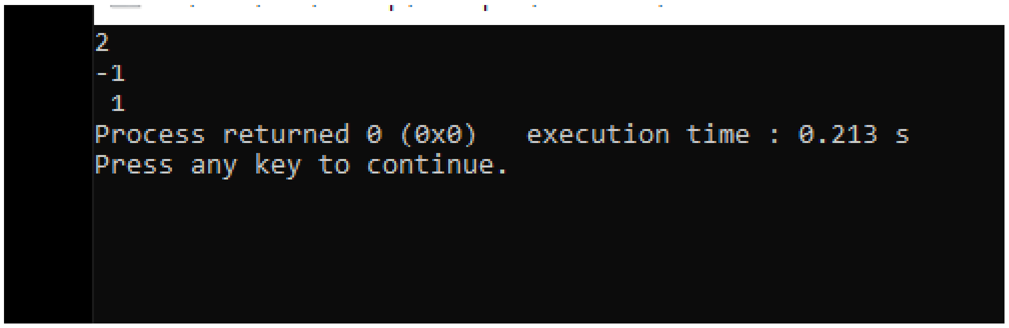


```
Travel time at 60 mph: 1.46667
Travel time at 100 kph: 0.88

Process returned 0 (0x0)   execution time : 0.231 s
Press any key to continue.
```

```
#include<iostream>

using namespace std;


    class dist{
```

```
public:
    double d;
    dist(double f){d=f;}
    virtual void trav_time()
    {
        cout<<"Travel time at 60 mph: ";
        cout<<d/60<<"\n";
    }
};

class metric:public dist{
public:
    metric(double f):dist (f){}
    void trav_time()
    {
        cout<<"Travel time at 100 kph: ";
        cout<<d/100<<"\n";
    }
};

int main()
{        dist *p,mph(88.0);
    metric kph(88);

    p=&mph;
    p->trav_time();
    p=&kph;
    p->trav_time();
    return 0;
}
```

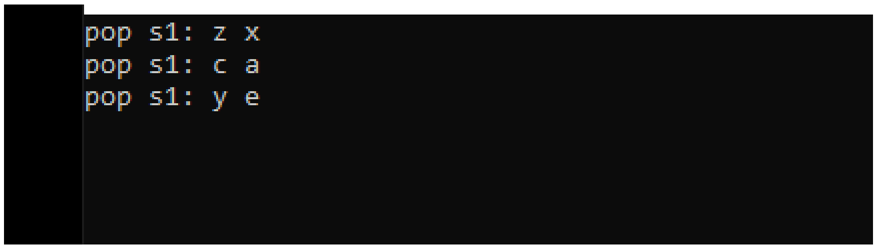
10] A good candidate for a template function is called find(). This function searches for an array for an object. It returns either the index os the matching object (if one is found) or -1 if no match is found. Here is the prototype for a specific version of find(). Convert find() into a generic function and demonstrate your solution within a program. (The size parameter specifies the number of elements in the array.) int find(int object, int *list, int size) { // … } (page_379)



```
2
-1
 1
Process returned 0 (0x0)   execution time : 0.213 s
Press any key to continue.
```

```
#include<iostream>

#include<cstring>

using namespace std;

    template <class X> int find(X object,X *list,int size)

    {

        int i;

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

        if(object==list[i])return i;

        return -1;

    }


    int main()

    {       int a[]={1,2,3,4};

        char *c = "this is test";

        double d[]={1.1,2.2,3.3};


        cout<<find(3,a,4);

        cout<<endl;

        cout<<find('a',c,(int)strlen(c));
        cout<<endl;
        cout<<find(0.0,d,3);
        return 0;
 }
```

11] Rework the stack class so that it can store pairs of different-type objects on the stack. Demonstrate your solution. (page_405)



```
pop s1: z x
pop s1: c a
pop s1: y e
```

```
#include<iostream>
using namespace std;
#define SIZE 10
template<class StackType>class stack{
    StackType stck[SIZE][2];
    int tos;
public:
    void init(){tos=0;}
    void push(StackType ob,StackType ob2);
    StackType pop(StackType &ob2);
};
template <class StackType>
void stack<StackType>::push(StackType ob,StackType ob2)
{    if(tos==SIZE){
        cout<<"Stack is full.\n";
        return; }
    stck[tos][0]=ob;
    stck[tos][1]=ob2;
    tos++;
}

template <class StackType>
StackType stack<StackType>::pop(StackType &ob2)
{
    if(tos==0){
        cout<<"Stack is empty.\n:";
        return 0;
    }
    tos--;
    ob2=stck[tos][1];
    return stck[tos][0];
}
int main()
{     stack<char>s1,s2;
    int i;
    char ch;
    s1.init();
    s1.init();

    s1.push('a','b');
    s1.push('x','z');
    s1.push('b','d');
    s1.push('y','e');
    s1.push('c','a');
    s1.push('z','x');

    for(i=0;i<3;i++){
        cout<<"pop s1: "<<s1.pop(ch);
        cout<<' '<<ch<<"\n";
    }
    for(i=0;i<3;i++){
        cout<<"pop s2: "<<s2.pop(ch);
        cout<<' ' <<ch<<"\n";
    }
    stack<double>ds1,ds2;
    double d;
    ds1.init();
    ds2.init();

    ds1.push(1.1,2.0);
    ds2.push(2.2,3.0);
    ds1.push(3.3,4.0);
    ds2.push(4.4,5.0);
    ds1.push(5.5,6.0);
    ds2.push(6.6,7.0);

    for(i=0;i<3;i++){
        cout<<"pop ds1: "<<ds1.pop(d);
        cout<<' '<<d<<"\n";
    }
    for(i=0;i<3;i++){
        cout<<"pop ds2: "<<ds2.pop(d);
        cout<<' '<<d<<"\n";
    }
return 0;
}
```