University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 12

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

OVERLOADING CONSTRUCTORS

```
#include <iostream>
using namespace std;
class Sample{
public:
    int x;
    int y;
    Sample(){x=y=0;}
    Sample(int i){x=y=i;}
    Sample(int i, int j){x=i;y=j;}
};
int main()
{
    Sample t;
    Sample t1(5);
    Sample t2(9,10);
    cout<<t.x<<" "<<t.y<<endl;
    cout<<t1.x<<" "<<t1.y<<endl;
    cout<<t2.x<<" "<<t2.y<<endl;
    return 0;
}
```

PROBLEM 12.1

Create a class to hold a 2D point. A point 2D point has 2 coordinates, x and y (float is fine).

Your class should have getters and setters for its member variables.
It should also allow the user to create objects when the don’t know the values of x and y, and when they do know them.

ASSIGNING OBJECTS

Objects must be of the same class!

```
#include <iostream>
using namespace std;
class Test{
    int a,b;
public:
    void setab(int i, int j){a=i, b=j;}
    void showab(){
        cout<<"a: "<<a<<endl;
        cout<<"b: "<<b<<endl;
    }
};
```

```
int main()
{
    Test obj1, obj2;
    obj1.setab(10,20);
    obj2.setab(0,0);
    cout<<"Objects before assignment: "<<endl;
    obj1.showab();
    obj2.showab();
    obj2 = obj1;
    cout<<"New Obj2"<<endl;
    obj2.showab();
    return 0;
}
```

PASSING OBJECTS TO FUNCTIONS

```
#include <iostream>
using namespace std;
class MyClass {
    int val;
public:
    MyClass(int i){
    val=i;
    cout<<"Inside Constructor"<<endl;}
    ~MyClass(){cout<<"Destructing"<<endl;}
    int getval() {return val;}
    void setval(int i){val=i;}
};
void display(MyClass ob)
{cout<<ob.getval()<<endl;}
int main()
{
    MyClass a(10);
    cout<<"before calling display()."<<endl;
    display(a);
    cout<<"After display()."<<endl;
    return 0;
}
```

What do you think the output is?

What if I change the code to:

void display(MyClass &ob)

POTENTIAL ISSUES WHEN PASSING OBJECTS

Even if you pass by value you may still break the original object.
For instance:

- If the object allocates system resources and frees it when it is destroyed.
- The original object will still try to use the (freed!) resource.

If you pass an object by reference, there is no call to the destructor inside the method.
If that is not possible you can write your own copy constructor.

RETURNING OBJECTS

```
Before constructing.
Inside constructor
After constructing.
Before Display.
10
Destructing
After Display.
Before mkbigger
Inside constructor
Destructing
After mkbigger
Before 2nd Display.
20
Destructing
After 2nd Display.
Destructing
```

```
#include <iostream>
using namespace std;
class MyClass{
    int val;
public:
    MyClass(int i){
    val=i;
    cout<<"Inside constructor"<<endl;
    }
    ~MyClass(){cout<<"Destructing"<<endl;}
    int getval(){return val;}
    MyClass mkBigger(){
        MyClass o(val*2);
        return o;
    }
};
```

```
void display(MyClass ob){cout<<ob.getval() <<endl;}
    int main(){
        cout<<"Before constructing."<<endl;
        MyClass a(10);
        cout<<"After constructing."<<endl;
        cout<<"Before Display."<<endl;
        display(a);
        cout<<"After Display."<<endl;
        cout<<"Before mkbigger"<<endl;
        a = a.mkBigger();
        cout<<"After mkbigger"<<endl;
        cout<<"Before 2nd Display."<<endl;
        display(a);
        cout<<"After 2nd Display."<<endl;
        return 0;
    }
```

THE COPY CONSTRUCTOR

When a copy of an argument is made during a function call the normal constructor is not called

- Instead the object's copy constructor is called.
- It defines how a copy of the object is made.
- So if the constructor initialises variables you have the chance to write a copy constructor so they are not re-initialised.
- In any case, the destructor is always called (more on this in a few slides)

THE COPY CONSTRUCTOR

```
#include <iostream>
using namespace std;
class MyClass{
    int val;
    int copynumber;
public:
    MyClass(int i){
    val=i;
    copynumber=0;
    cout<<"Inside constructor"<<endl;
    }
    MyClass(const MyClass &o){
        val = o.val;
        copynumber = o.copynumber+1;
        cout<<"Inside copy constructor."<<endl;
    }
    ~MyClass() {
        if (copynumber==0)
            cout<<"Destructing original"<<endl;
        else
            cout<<"Destructing copy "<<copynumber<<endl;
        }
    int getval(){return val;}
};
```

```
int main(){
    MyClass a(10);
    MyClass b=a;
    cout<<a.getval()<<endl;
    return 0;
}
```

Output:

Inside constructor

Inside copy constructor.

10

Destructing copy 1

Destructing original

PROBLEM 12.2

Update your 2D point class so it has a copy constructor. Write it in such way it is possible to know if an object is the original object, or a copy.

FRIEND FUNCTIONS

```
#include <iostream>
using namespace std;
class MyClass{
    int a,b;
public:
    MyClass(int i, int j){a=i;b=j;}
    friend int comDenom(MyClass x);
};
int comDenom (MyClass x){
    int max=x.a < x.b ?x.a : x.b;
    for (int i=2; i<=max;i++)
        if((x.a%i)==0 && (x.b%i)==0)
            return i;
        return 0;
}
int main()
{
    MyClass n(18,9);
    if(comDenom(n))
        cout<<"The common denominator is: "<<comDenom(n)<<endl;
    else
        cout<<"No common denominator"<<endl;
    return 0;
}
```

Allows a nonmember function access to the private members of a class.

KEYWORD: THIS

```
#include <iostream>
using namespace std;
class Test{
    int i;
public:
    Test(int val){i = val;}
    int get_i(){return i;}
    void increment_i_once(){i++;}
    void increment_i(int n){
        for (int j=0; j<n; j++)
            this->increment_i_once();
    }
};
int main(){
    Test obj(5);
    cout<<obj.get_i()<<endl;
    obj.increment_i(5);
    cout<<obj.get_i()<<endl;
    return 0;
}
```

This is a trivial example just to illustrate the use of This.

OPERATOR OVERLOADING

```
#include <iostream>
using namespace std;
class ThreeD{
    int x,y,z;
public:
    ThreeD(){x=y=z=0;}
    ThreeD(int i, int j, int k){x=i;y=j;z=k;}
    ThreeD operator+(ThreeD op2);
    ThreeD operator++(){x++;y++;z++; return *this;} //prefix!
    ThreeD operator++(int ignore){
            x+=2;y+=2;z+=2; return *this;}//postfix
    void show(){cout<<"x: "<<x<<" y: "<<y<<" z: "<<z<<endl;}
};
```

```
ThreeD ThreeD::operator+(ThreeD op2){
    ThreeD temp(x+op2.x, y+op2.y, z+op2.z);
    return temp;
}
int main(){
    ThreeD a(1,2,3), b(10,10,10), c;
    a.show();
    c=a+b;
    c.show();
    (a+b).show();
    (++a).show();
    (a++).show();
    return 0;
}
```

PROBLEM 12.3

Update your 2D point class so it overloads the following operators:

++, -- (pre and postfix), +, -, *, /.

OVERLOADING WITH FRIEND

```
#include <iostream>
using namespace std;
class ThreeD{
    int x,y,z;
public:
    ThreeD(){x=y=z=0;}
    ThreeD(int i, int j, int k){x=i;y=j;z=k;}
    friend ThreeD operator-(ThreeD op1,ThreeD op2);
    void show(){cout<<"x: "<<x<<" y: "<<y<<" z: "<<z<<endl;}
};
```

```
ThreeD operator-(ThreeD op1, ThreeD op2)
{
    ThreeD temp(op1.x - op2.x, op1.y - op2.y, op1.z-op2.z);
    return temp;
}
int main(){
    ThreeD a(1,2,3), b(10,10,10);
    (a-b).show();
    return 0;
}
```

OVERLOADING UNARY OPERATORS

Order matters

- A – B is not the same as B – A

The overloaded operator may not have any relationship with its default usage.

- However, you should aim to have some relationship (for readability)