Object-Oriented Programming

Lecture No. 5

Copy constructor, Operator overloading, this pointer, and Array of objects

Copy constructor

- A copy constructor is a member function that initializes an object using another object of the same class.

Need of copy constructor I

In C++, a Copy Constructor may be called in the following cases:
1. When an object of the class is returned by value.
2. When an object of the class is passed (to a function) by value as an argument.
3. When an object is constructed based on another object of the same class.

Copy constructor–Example

```
class Point
{
private:
        int x, y;
public:
        Point(int x1, int y1)
        { x = x1;
          y = y1;
        }
        // Copy constructor
        Point( Point &p1) {x = p1.x; y =
p1.y; }
        int getX() { return x; }
        int getY() { return y; }
};
```

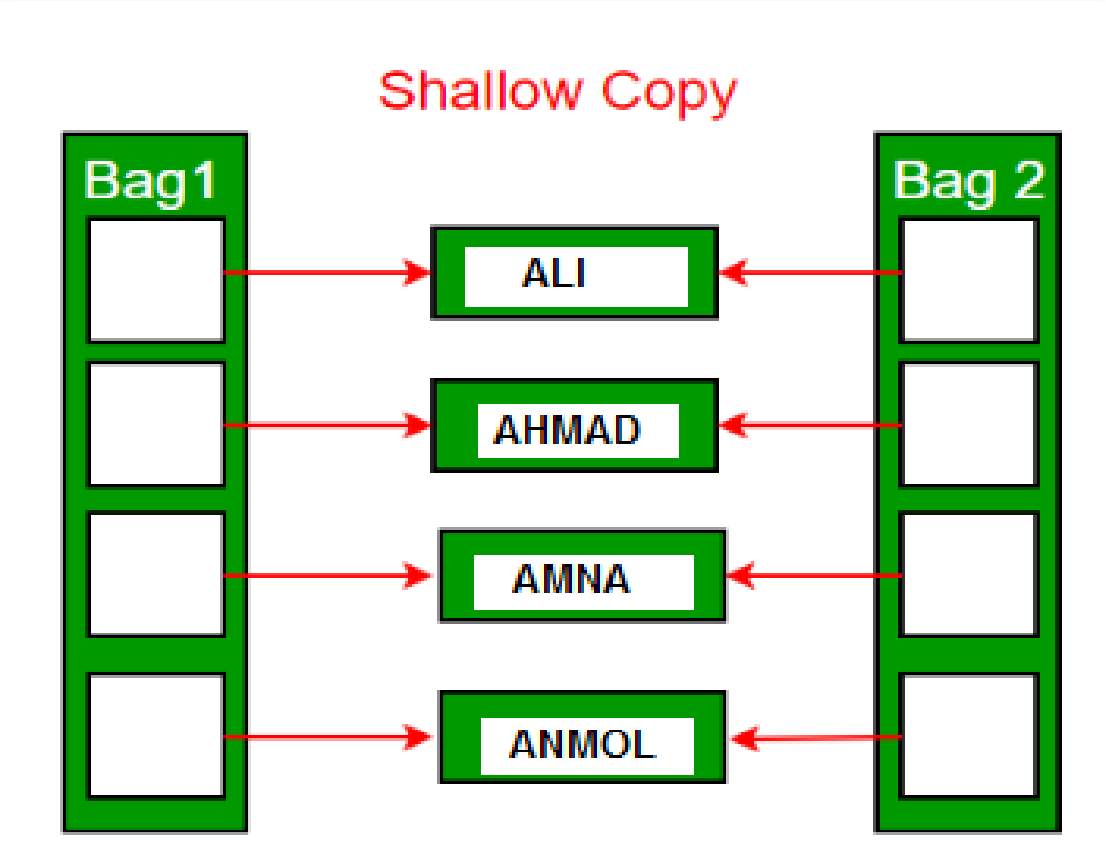
```
main()
{
// Parameterized constructor is called
Point p1(10, 15);
// Copy constructor is called
Point p2 = p1;     //Point p2 (p1);

cout << "p1.x = " << p1.getX() << ", p1.y =
" << p1.getY();
cout << "\n p2.x = " << p2.getX() << ",
p2.y = " << p2.getY();
}
```

Need of user-defined copy constructor

- If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler created copy constructor works fine in general. We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource (recall: dangling pointer).
  The default ***constructor does only shallow copy.***

Shallow copy



Deep copy

- ***Deep copy is possible only with user defined copy constructor.*** In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.

Copy constructor vs. Assignment operator

- Which of the following two statements call copy constructor and which one calls assignment operator?
- MyClass t1, t2;
- MyClass t3 = t1; // ----> (1)
- t2 = t1; // -----> (2)

- Copy constructor is called when a new object is created from an existing object, as a copy of the existing object. Assignment operator is called when an already initialized object is assigned a new value from another existing object. In the above example (1) calls copy constructor and (2) calls assignment operator.

Operator overloading

- In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.
- For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big Integer, etc.

Operator overloading example1 –overloading ‘+’ operator

```
class Complex {
private:
        int real, imag;
public:
        Complex(int r = 0, int i = 0) {real = r; imag = i;}

// This is automatically called when '+' is used with
        // between two Complex objects
        Complex operator + (Complex &obj) {
                Complex res;
                res.real = real + obj.real;
                res.imag = imag + obj.imag;
                return res;
        }
        void print() { cout << real << " + i" << imag <<
'\n'; }
};
```

```
int main()
{
        Complex c1(10, 5), c2(2, 4);
        Complex c3 = c1 + c2;
        c3.print();
}
```

Operator overloading example1 –overloading assignment operator

```
class Distance {
   private:
      int feet;
      int inches;
      public:
Distance(int f, int i) {
         feet = f;
         inches = i;
      }
      void operator = (Distance &D ) {
         feet = D.feet;
         inches = D.inches;
      }
      void displayDistance() {
         cout << "F: " << feet <<  " I:" <<  inches
<< endl;     } };
```

```
int main() {
   Distance D1(11, 10), D2(5, 11);

   cout << "First Distance : ";
   D1.displayDistance();
   cout << "Second Distance :";
   D2.displayDistance();

   // use assignment operator
   D1 = D2;
   cout << "First Distance :";
   D1.displayDistance();
```

Static data member

- Static data members are class members that are declared using the static keyword. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. The static data member is always initialized to zero when the first class object is created.

Static data member–Example

```
class Student {
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student() {
    objectCount++;
  }

  void getdata() {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
    cout << "Enter name: "<<endl;
    cin >> name;
    cout << "Enter marks: "<<endl;
    cin >> marks;
  }
```

```
 void putdata() {
     cout<<"Roll Number = "<< rollNo <<endl;
     cout<<"Name = "<< name <<endl;
     cout<<"Marks = "<< marks <<endl;
     cout<<endl;
   }
};
int Student::objectCount = 0;
int main(void) {
  Student s1;
  s1.getdata();
  s1.putdata();
  Student s2;

  s2.getdata();
  s2.putdata();
  Student s3;

  s3.getdata();
  s3.putdata();
  cout << "Total objects created = " <<
Student::objectCount << endl;
  return 0;
}
```

this pointer

- ‘this’ pointer is passed as a hidden argument to all non-static member function calls and is available as a local variable within the body of all non-static functions.
- Some situations where ‘this’ pointer is used:

1) When local variable’s name is same as member’s name
2) To return reference to the calling object

this pointer -examples

```
//Example 1
class Test
{
private:
  int x;
  int y;
public:
  Test(int x , int y) { this->x = x; this->y = y; }
  void fun1() { cout << "Inside fun1()"; }
  void fun2() { cout << "Inside fun2()"; this->fun1(); }
};

int main()
{
  Test obj (2, 4);
  obj.fun2();
  return 0;
}
```

```
//Example 2


/* Reference to the calling object can be
returned */
Test& Test::func ()
{
   // Some processing
   return *this;
}
```

Array of objects

An array of objects is declared in the same way as an array of any built-in data type.
The syntax for declaring an array of objects is

    class_name array_name [size] ;

Array Of Objects – Example 1

```
class Number
{           private:
                    int a;
                    float b;
            public:
                    //parameterless constructor
                    Number(){a=0;b=0.0;}
                    //Parameterized constructor
                    Number(int x,float y)
                    {
                            a=x;
                            b=y;
                    }
                    void printValue()
                    {

            cout<<"a="<<a<<",b="<<b<<endl;
                    }
};
```

```
int main()
{
Number NUM[3]= { Number(10,20.00),
Number(15,30.56), Number(20,45.50) };

            NUM[0].printValue();
          NUM[1].printValue();
          NUM[2].printValue();

          return 0;
}
```

Array of objects –Example 2

```
class Student
{
     string name;
     int marks;
     public:
    void getName()
    {
     getline( cin, name );
    }
   void getMarks()
    {
   cin >> marks;
    }
    void displayInfo()
   {
   cout << "Name : " << name << endl;
    cout << "Marks : " << marks << endl;
    }
};
```

```
int main()
{
        Student st[5];
        for( int i=0; i<5; i++ )
        {
                cout << "Student " << i + 1 << endl;
                cout << "Enter name" << endl;
                st[i].getName();
                cout << "Enter marks" << endl;
                st[i].getMarks();
        }

        for( int i=0; i<5; i++ )
        {
                cout << "Student " << i + 1 << endl;
                st[i].displayInfo();
        }
        return 0;
}
```

Array of objects –Example 3

```
class Employee
    {    int Id;
         char Name[25];
         int Age;
         long Salary;
         public:
         void GetData() {
              cout<<"\n\tEnter Employee Id : ";
              cin>>Id;
              cout<<"\n\tEnter Employee Name : ";
              cin>>Name;
              cout<<"\n\tEnter Employee Age : ";
              cin>>Age;
              cout<<"\n\tEnter Employee Salary : ";
              cin>>Salary;
         }
         void PutData() {
cout<<"\n"<<Id<<"\t"<<Name<<"\t"<<Age<<"\t"<<Salary;
         }
    };
```

```
int main()
    {
         int i;

         Employee E[3];
         for(i=0;i<3;i++)
         {
              cout<<"\nEnter details of "<<i+1<<"
Employee";
              E[i].GetData();
         }

         cout<<"\nDetails of Employees";
         for(i=0;i<3;i++)
         E[i].PutData();

    }
```

Reference

- C++ How to Program
  By Deitel & Deitel

- The C++ Programming Language
  By Bjarne Stroustrup

- Object oriented programming using C++ by Tasleem Mustafa, Imran Saeed, Tariq Mehmood, Ahsan Raza

- https://www.tutorialspoint.com

- https://www.geeksforgeeks.org/copy-constructor-in-cpp/

- https://www.includehelp.com/cpp-programs/