CSE206, Object Oriented Programming

Week 6

Dept. of CSE, MIST
Nazmun Nahar Khanom

Copy Constructor (1/5)

- In C++, a copy constructor is a special type of constructor that creates a new object by copying an existing object of the same class.
- The copy constructor is used to create a new object that is a copy of an existing object.
- It is called when an object is passed by value, returned by value, or when an object is constructed based on another object of the same class.
- Doesn’t have a return type.

Copy Constructor (2/5)

- Here are three ways in which a copy constructor can be invoked in C++:
- **1. Direct initialization:** The copy constructor can be invoked by directly initializing a new object with an existing object.

  Person p1("Alice", 25);

  Person p2 = p1; // copy p1 to p2 using the copy constructor

- In this example, the ‘Person’ object ‘p2’ is being initialized with the existing ‘Person’ object ‘p1’. This invokes the copy constructor to create a new ‘Person’ object that is a copy of ’p1’.

Copy Constructor (3/5)

- **2. Pass by value:** The copy constructor can be invoked when an object of the class is passed by value to a function.

```
void someFunction(Person p) {
    // function body
}
int main() {
    Person p1("Alice", 25);
    someFunction(p1); // copy p1 to the parameter of someFunction using the
                                                    copy constructor
    return 0;
}
```

- In this example, the **Person** object **p1** is passed by value to the **someFunction** function. This invokes the copy constructor to create a copy of **p1** that is passed to the function as a parameter.

Copy Constructor (4/5)

- **3. Return by value:** The copy constructor can be invoked when a function returns an object of the class by value.

```
Person someFunction() {
    Person p("Alice", 25);
    return p; // copy p to the return value using the copy constructor
}

int main() {
    Person p1 = someFunction(); // copy the return value of someFunction to
                                p1 using the copy constructor
    return 0;
}
```

Copy Constructor (5/5)

- In this example, the **someFunction** function returns a **Person** object by value.
- This invokes the copy constructor to create a copy of the local **Person** object **p** that is returned as the function's result.
- The copy constructor is then invoked again to create a new **Person** object **p1** that is a copy of the returned object.

Why do We Need Copy Constructor? (1/2)

We need a copy constructor in C++ to create a new object that is an exact copy of an existing object. A copy constructor is used to initialize a new object using an existing object's data members.

Here are some reasons why we need a copy constructor:

- **To create a copy of an object:** Sometimes, we need to create a new object that has the same values as an existing object. A copy constructor is used to make a duplicate of an object with the same values for its data members.
- **To pass an object by value:** When we pass an object by value to a function, a copy of the object is created. This copy is made using the copy constructor. The copy constructor is responsible for creating a new object with the same values as the original object.

Why do We Need Copy Constructor? (2/2)

- **To return an object by value:** When a function returns an object by value, a copy of the object is created. The copy constructor is used to create this copy of the object.
- **To initialize an object using another object:** We can use the copy constructor to initialize a new object using an existing object. This is useful when we want to create a new object that is similar to an existing object.
- **In summary**, the copy constructor is a useful tool in C++ for creating a copy of an existing object. It is used in many situations where we need to create a new object with the same values as an existing object, such as passing an object by value or returning an object by value.

Types of Copy Constructor

- In C++, there are two types of copy constructors:
  - *Default Copy Constructor*
  - *User-defined Copy Constructor*
- Other types of copy constructors:
  - *Shallow Copy Constructor*
  - *Deep Copy Constructor*

Default Copy Constructor (1/3)

- A default copy constructor is automatically generated by the compiler if the programmer doesn't provide a copy constructor explicitly.
- It creates a new object that is a copy of the source object with all the member variables copied from the source object.

Default Copy Constructor (2/3)

```
#include <bits/stdc++.h>
using namespace std;
class Person {
private:
    string name;
    int age;
public:
    Person(string n, int a) {
        name = n;
        age = a;
        cout << "Creating a Person object..." <<
endl;
    }

    void print() {
        cout << "Name: " << name << ", Age: "
<< age << endl;
    }
};
```

```
int main() {
    Person p1("Alice", 25);
    Person p2 = p1; // copy p1 to p2 using the default copy
                                constructor
    p1.print(); // prints "Name: Alice, Age: 25"
    p2.print(); // prints "Name: Alice, Age: 25"
    return 0;
}
```

```
Creating a Person object...
Name: Alice, Age: 25
Name: Alice, Age: 25
```

Default Copy Constructor (3/3)

- In this example, the **Person** class has a default constructor that takes no arguments. The default constructor initializes the **name** and **age** data members of the object with default values ("", 0 in this case) and prints a message to the console indicating that a default object is being created.
- The purpose of a default constructor is to create an object of the class with default values for its data members. It is used when an object of the class is created without any arguments, such as when the object is declared without initialization or when it is initialized with an empty set of parentheses.
- In this example, the default constructor sets the **name** and **age** data members of the **Person** object to default values, which are empty string and 0 respectively. The **print** function can be used to print the values of the data members of the object, which will show that they have been initialized with the default values.

User-defined Copy Constructor (1/4)

- A user-defined copy constructor allows you to define your own logic for copying objects of a class.
- This is useful when you need to copy non-trivial data members, such as pointers, dynamic arrays, or other objects.

User-defined Copy Constructor (2/4)

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

class Person {
private:
    string name;
    int age;
public:
    Person(string n, int a)
    {
            name=n;
             age=a;
     }
```

```
Creating Person object...
Copying Person object...
Name: Alice, Age: 25
Name: Alice, Age: 25
```

```
// user-defined copy constructor
    Person(const Person& obj) {
            name=obj.name;
            age=obj.age;
            cout << "Copying Person object..." << endl;
    }
    void print() {
        cout << "Name: " << name << ", Age: " << age <<
endl;
    }
};
int main() {
    Person p1("Alice", 25);
    Person p2 = p1; // copy p1 to p2 using the copy
constructor
    p1.print(); // prints "Name: Alice, Age: 25"
    p2.print(); // prints "Name: Alice, Age: 25"
    return 0;
}
```

User-defined Copy Constructor (3/4)

- **Person(const Person& other)** is a user-defined copy constructor for the **Person** class that takes a reference to a constant **Person** object as a parameter. The **const** keyword means that the function cannot modify the object that it takes as a parameter, and the ampersand **&** symbol means that the function takes the object by reference, which avoids making a copy of the object.
- The copy constructor is used when an object of the class is copied, either by being passed by value, returned by value, or initialized with another object of the same type. The copy constructor creates a new object that is a copy of an existing object, and it typically initializes the new object with the same values as the original object.
- In the case of **Person(const Person& other)**, the copy constructor copies the **name** and **age** data members from the **other** object to the corresponding data members of the new **Person** object being constructed. This ensures that the new object is a copy of the original object and has the same values for its data members.

User-defined Copy Constructor (4/4)

- In this version of the code, the user-defined copy constructor initializes the "name" and "age" data members of the current object with the corresponding values from the passed object using a member initialization list.
- The purpose of a copy constructor is to create a new object that is a copy of an existing object. It is used when an object is passed by value, returned by value, or initialized with another object of the same type.
- In this example, the user-defined copy constructor simply copies the "name" and "age" data members of the object. However, in more complex cases where the object contains dynamically allocated memory or other resources, a user-defined copy constructor may need to perform additional operations to properly copy the object.

Shallow Copy Constructor (1/2)

- A shallow copy constructor only copies the address of the dynamically allocated memory from the source object to the new object.
- Both the new object and the source object share the same memory, which can lead to problems if the memory is deleted or modified.

Shallow Copy Constructor (2/2)

```
class MyClass {
public:
  int* arr;
  int size;

  // shallow copy constructor
  MyClass(const MyClass& obj) {
    arr = obj.arr;
    size = obj.size;
  }
};

int main() {
  MyClass obj1;
  obj1.size = 5;
  obj1.arr = new int[obj1.size];
  for(int i = 0; i < obj1.size; i++) {
    obj1.arr[i] = i;
  }
  MyClass obj2 = obj1; // calls the shallow copy constructor

  obj1.arr[0] = 100; // modify obj1's array

  std::cout << obj2.arr[0] << std::endl; // prints 100

  return 0;
}
```

- In this example, the shallow copy constructor is called when creating obj2 as a copy of obj1. The shallow copy constructor only copies the address of the dynamically allocated memory from obj1 to obj2. Both objects share the same memory, which means that modifying the memory of one object affects the other object.

Deep Copy Constructor (1/2)

- A deep copy constructor creates a new object with its own dynamically allocated memory.
- It copies the values of the member variables from the source object to the new object and allocates new memory for the new object's member variables.

Deep Copy Constructor (2/2)

```
class MyClass {
public:
  int* arr;
  int size;

  // deep copy constructor
  MyClass(const MyClass& obj) {
    size = obj.size;
    arr = new int[size];
    for(int i = 0; i < size; i++) {
      arr[i] = obj.arr[i];
    }
  }

  // destructor
  ~MyClass() {
    delete[] arr;
  }
};
```

```
int main() {
  MyClass obj1;
  obj1.size = 5;
  obj1.arr = new int[obj1.size];
  for(int i = 0; i < obj1.size; i++) {
    obj1.arr[i] = i;
  }

  MyClass obj2 = obj1; // calls the deep copy constructor

  obj1.arr[0] = 100; // modify obj1's array

  std::cout << obj2.arr[0] << std::endl; // prints 0

  return 0;
}
```

Explanation

- It is an example of a class **MyClass** that contains an integer array **arr** and an integer **size**.
- It defines a deep copy constructor that creates a new object with its own dynamically allocated memory, and a destructor that deletes the memory when the object is destroyed.
- In the **main** function, an instance **obj1** of **MyClass** is created and initialized with a size of 5 and values of 0, 1, 2, 3, and 4 in its **arr** array.
- Then, a new instance **obj2** is created and initialized with **obj1** using the deep copy constructor.
- After that, the first element of **obj1**'s **arr** array is modified to 100.
- This does not affect **obj2** because its **arr** array has its own memory allocation and is not linked to **obj1**.
- Therefore, when the program prints out the first element of **obj2**'s **arr** array, it still contains the original value of 0.

Problem-1

- Write a program that defines a **Point** class with **x** and **y** integer coordinates.
- Define a user-defined copy constructor that copies the **x** and **y** coordinates of the source object to the destination object.
- Create two **Point** objects and copy one object to the other.
- Print the **x** and **y** coordinates of both objects.

Solution

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

class Point {
public:
    int x;
    int y;

    // User-defined copy constructor
    Point(const Point& other) {
        x = other.x;
        y = other.y;
    }
};

int main() {
    Point p1 = { 10, 20 };
    Point p2 = p1;

    cout << "p1: (" << p1.x << ", " << p1.y << ")" << endl;
    cout << "p2: (" << p2.x << ", " << p2.y << ")" << endl;

    return 0;
}
```

Output:

```
p1: (10, 20)
p2: (10, 20)
```

Problem-2

- Write a program that defines a **Student** class with a **name** string and a **marks** integer array.
- Define a user-defined copy constructor that allocates memory for the **name** field and copies the contents of the **name** and **marks** fields from the source object to the destination object.
- Create two **Student** objects and copy one object to the other. Modify the **name** and **marks** fields of the second object and print the details of both objects.

Solution (1/2)

```cpp
#include <iostream>
#include <string.h>
using namespace std;

class Student {
public:
    char* name;
    int* marks;

    // User-defined copy constructor
    Student(const Student& other) {
        name = new char[strlen(other.name) + 1];
        strcpy(name, other.name);

        marks = new int[5];
        memcpy(marks, other.marks, sizeof(int) * 5);
    }

    // Destructor
    ~Student() {
        delete[] name;
        delete[] marks;
    }
};

int main() {
    Student s1 = { "John", new int[5]{ 80, 70, 90, 85, 95 } };
    Student s2 = s1;

    s2.name = "Jane";
    s2.marks[0] = 75;

    cout << "s1: " << s1.name << ", marks: ";
    for (int i = 0; i < 5; i++) {
        cout << s1.marks[i] << " ";
    }
    cout << endl;

    cout << "s2: " << s2.name << ", marks: ";
    for (int i = 0; i < 5; i++) {
        cout << s2.marks[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Solution (2/2)

- Output:

```
s1: John, marks: 80 70 90 85 95
s2: Jane, marks: 75 70 90 85 95
```

Problem-3

- Write a program that defines a **Rectangle** class with **width** and **height** integer fields.
- Define a user-defined copy constructor that copies the **width** and **height** fields of the source object to the destination object.
- Define a member function **area** that returns the area of the rectangle.
- Create two **Rectangle** objects and copy one object to the other.
- Modify the **width** and **height** fields of the second object and print the details of both objects.

Solution (1/2)

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

class Rectangle {
public:
    int width;
    int height;

    // User-defined copy constructor
    Rectangle(const Rectangle& other) {
        width = other.width;
        height = other.height;
    }

    // Member function to calculate area
    int area() {
        return width * height;
    }
};
```

```cpp
int main() {
    Rectangle r1 = { 10, 20 };
    Rectangle r2 = r1;

    r2.width = 15;
    r2.height = 25;

    cout << "r1: width = " << r1.width << ", height = " << r1.height << ", area = "
    cout << "r2: width = " << r2.width << ", height = " << r2.height << ", area = "

    return 0;
}
```

Solution (2/2)

- Output:

```
r1: width = 10, height = 20, area = 200
r2: width = 15, height = 25, area = 375
```

Keep in mind!

- A copy constructor is called when a new object is created from an existing object of the same class.
- If you do not define a copy constructor, the compiler will provide a default copy constructor that performs a shallow copy of the object's member variables.
- If your class contains pointers or other dynamically allocated resources, you should define a copy constructor that performs a deep copy of the object's state.
- Copy constructors can also be used in a variety of ways, such as passing objects by value, returning objects from functions, and initializing objects with other objects.
- Copy constructors can also be used for inheritance. When you create a derived class object, the base class constructor and the copy constructor are called in that order.
- Finally, you should always follow the Rule of Three (or Rule of Five in C++11 and later), which states that if you define a copy constructor, you should also define a destructor and an assignment operator.

Problem-4

Write a shallow copy constructor for a class **Person** that contains a string **name** and a pointer to an object of class **Address**.

Solution

```
class Address {
public:
    string street;
    string city;
    string state;
    int zip;
};

class Person {
public:
    string name;
    Address* address;

    // shallow copy constructor
    Person(const Person& other) {
        name = other.name;
        address = other.address;
    }
};
```

Problem-5

Write a deep copy constructor for a class **Matrix** that contains a two-dimensional integer array **data** and two integers **rows** and **columns**.

Solution

```
class Matrix {
public:
    int** data;
    int rows;
    int columns;

    // deep copy constructor
    Matrix(const Matrix& other) {
        rows = other.rows;
        columns = other.columns;
        data = new int*[rows];
        for (int i = 0; i < rows; i++) {
            data[i] = new int[columns];
            for (int j = 0; j < columns; j++) {
                data[i][j] = other.data[i][j];
            }
        }
    }
};
```

Problem-6

Write a class **MyString** that contains a character array **str** and an integer **length**. Write a deep copy constructor for this class.

Solution

```
class MyString {
public:
    char* str;
    int length;

    // deep copy constructor
    MyString(const MyString& other) {
        length = other.length;
        str = new char[length+1];
        strcpy(str, other.str);
    }
};
```

Problem-7

Write a class **Student** that contains a string **name**, an integer **age**, and a pointer to a vector of strings **courses**. Write a deep copy constructor for this class.

Solution

```
class Student {
public:
    string name;
    int age;
    vector<string>* courses;

    // deep copy constructor
    Student(const Student& other) {
        name = other.name;
        age = other.age;
        courses = new vector<string>(*other.courses);
    }
};
```

Problem-8

Write a class **Image** that contains a pointer to a two-dimensional integer array **pixels**, two integers **width** and **height**, and a string **name**. Write a shallow copy constructor for this class.

Solution

```
class Image {
public:
    int** pixels;
    int width;
    int height;
    string name;

    // shallow copy constructor
    Image(const Image& other) {
        pixels = other.pixels;
        width = other.width;
        height = other.height;
        name = other.name;
    }
};
```

Assignment

- Write a program that creates a class **Rectangle** with two member variables: **width** and **height**.
- The class should have a constructor that initializes **width** and **height**, and a copy constructor that creates a new object with the same **width** and **height**.
- Additionally, the class should have a **getArea** method that returns the area of the rectangle (i.e. **width * height**).
- In the **main** function, create an instance of **Rectangle** with a width of 5 and a height of 10.
- Then, create a new instance of **Rectangle** using the copy constructor and modify its **width** to 7.
- Finally, print out the area of both rectangles to verify that the original rectangle was not modified by the change in the copy.
- Once you have completed the program, you can try extending it further by adding more functionality to the **Rectangle** class, such as methods for calculating the perimeter or checking if the rectangle is a square.


thank you