C++ Classes and Objects I

NEW YORK UNIVERSITY
ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

Definition


```
class class_name
{
    access_specifier:
        member;
};
```

```
class Circle
{
    public:
        double radius;    // radius
        double center_x;  // center x position
        double center_y;  // center y position
};
```

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

Member

The parts used to define the class are called members

- Data Member
- Function Member

Member

Data Member

```
class Circle
{
   public:
      double radius;     // radius
      double center_x;   // center x position
      double center_y;   // center y position
      double area;       // area of the circle
      double area();     // calculate the area of a circle
};
```

Function Member

Define Member Function

NEW YORK UNIVERSITY
ABU DHABI

- Member function defined within class definition
- Member function defined outside class definition

Define Member Function

NEW YORK UNIVERSITY
ABU DHABI

```
#define PI 3.14
class Circle
{
   public:
      double radius;     // radius
      double center_x;   // center x position
      double center_y;   // center y position
      double area;       // area of the circle
      double area()    // calculate the area of a circle
      {
        return PI*radius*radius;
      }
};

//Defintion of member function Circle
double Circle::area()
{
    return PI*radius*radius;
}
```

Definition in class

class name :: function name

Definition outside class

Special Member Function

NEW YORK UNIVERSITY
ABU DHABI

- Class Constructor:

A special member function of a class that is executed whenever we create new objects of that class.

- Class Destructor:

A special member function of a class that is called whenever an object passes out of scope or is explicitly deleted.

Special Member Function

NEW YORK UNIVERSITY ABU DHABI

- A constructor is a member function with the same name as its class.
- A destructor is a member function with the same name as its class prefixed by a ~ (tilde).
- The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.

```
class Circle
{
public:
    Circle();  // Constructor for class Circle
    ~Circle(); // Destructor for class Circle
}
```

Will revisit this with detailed examples after introducing C++ Objects

Access Specifier

NEW YORK UNIVERSITY
ABU DHABI

- Public:

  public members are accessible from any function
- Private:

  private members are accessible only from member functions of the same class (or friends of the class).
- Protected:

  protected members are similar to private members but they are accessible from member functions in the derived classes.

Will revisit this with detailed examples after introducing C++ Objects

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

C++ Objects

NEW YORK UNIVERSITY
ABU DHABI

- A class provides blueprint
- An object is an instance of a class

For example:

```
Circle  C_1; // Declare C_1 of type Circle
Circle  C_2; // Declare C_2 of type Circle
```

```cpp
#include <iostream>

class Circle
{
public:
    Circle()
    {
        cout<<"Define a Circle Object"<<endl;
    }
    ~Circle()
    {
        cout<<"Delete a Circle Object"<<endl;
    }
public:
      double radius;     // radius
      double center_x;   // center x position
      double center_y;   // center y position
      double area;       // area of the circle
      double area()    // calculate the area of a circle
      {
        return PI*radius*radius;
      }
      double getRadius()  // Obtain the value of radius
      {
          return radius;
      }
      void setRadius(double v)   // Set the value of radius
      {
          radius = v;
      }

};
// Main function for the program
int main( )
{
    // Define the objects of Circle class
    Circle C_1; // Define the first circle
    Circle C_2; // Define the second circle
    return 0;
}
```

NEW YORK UNIVERSITY

ABU DHABI

Revisit constructor and Destructor

```
class Circle
{
public:
    Circle()
    {
        cout<<"Define a Circle Object"<<endl;
    }
    ~Circle()
    {
        cout<<"Delete a Circle Object"<<endl;
    }
public:
       double radius;     // radius
       double center_x;   // center x position
       double center_y;   // center y position
       double area;       // area of the circle
       double area()     // calculate the area of a circle
       {
         return PI*radius*radius;
       }
       double getRadius()  // Obtain the value of radius
       {
           return radius;
       }
       void setRadius(double v)    // Set the value of radius
       {
           radius = v;
       }

};
// Main function for the program
int main( )
{
    // Define the objects of Circle class
    Circle C_1; // Define the first circle
    Circle C_2; // Define the second circle
    return 0;
}
```

call constructor function

call destructor function

Copy constructor

NEW YORK UNIVERSITY
ABU DHABI

```
classname (const classname &obj)
{
  // body of copy constructor
}
```

Copy constructor

NEW YORK UNIVERSITY
ABU DHABI

```
#include <iostream>
class Circle
{
public:
    Circle() // constructor
    {
        cout<<"Define a Circle Object"<<endl;
    }
    Circle(const Circle &obj); // copy constructor

    ~Circle() // destructor
    {
        cout<<"Delete a Circle Object"<<endl;
    }

public:
      double center_x;  // center x position
      double center_y;  // center y position
      double area;      // area of the circle
      double area()    // calculate the area of a circle
      {
        return PI*radius*radius;
      }
      double getRadius()  // Obtain the value of radius
      {
          return radius;
      }
      void setRadius(double v)   // Set the value of radius
      {
          radius = v;

      }
private:
      double radius;    // radius

};
```

```
Circle::Circle(const Circle &obj)
{
    cout<< "Copy constructor called "<<endl;
    center_x = obj.center_x;
    center_y = obj.center_y;
}

// Main function for the program
int main( )
{
    // Define the objects of Circle class
    Circle C_1; // Define the first circle


    //Set the radius
    C_1.setRadius(6.0); //Set the value of radius

    //Copy constructor is called
    Circle C_2(C_1);

    //Copy constructor is called using assignment operator
    Circle C_3 = C_1;

    return 0;
}
```

Revisit Access Specifier

NEW YORK UNIVERSITY ABU DHABI

```
class Circle
{
public:
     Circle()
     {
          cout<<"Define a Circle Object"<<endl;
     }
     ~Circle()
     {
          cout<<"Delete a Circle Object"<<endl;
     }
public:
        double radius;      // radius
        double center_x;    // center x position
        double center_y;    // center y position
        double area;        // area of the circle
        double area()     // calculate the area of a circle
        {
          return PI*radius*radius;
        }
        double getRadius()  // Obtain the value of radius
        {
            return radius;
        }
        void setRadius(double v)    // Set the value of radius
        {
            radius = v;
        }

};
// Main function for the program
int main( )
{
     // Define the objects of Circle class
     Circle C_1; // Define the first circle
     Circle C_2; // Define the second circle

     //Set the radius
     C_1.setRadius(6.0); //Set the value of radius

     //Set the radius without member function
     C_2.radius = 6.0;    //No problem since radius is a public

     return 0;
}
```

Public members can be accessible without from a member function

```
#include <iostream>
class Circle
{
public:
     Circle()
     {
          cout<<"Define a Circle Object"<<endl;
     }
     ~Circle()
     {
          cout<<"Delete a Circle Object"<<endl;
     }
public:
        double center_x;   // center x position
        double center_y;   // center y position
        double area;       // area of the circle
        double area()     // calculate the area of a circle
        {
          return PI*radius*radius;
        }
        double getRadius()  // Obtain the value of radius
        {
            return radius;
        }
        void setRadius(double v)    // Set the value of radius
        {
            radius = v;

        }
private:
        double radius;     // radius

};
// Main function for the program
int main( )
{
     // Define the objects of Circle class
     Circle C_1; // Define the first circle
     Circle C_2; // Define the second circle

     //Set the radius
     C_1.setRadius(6.0); //Set the value of radius

     //Set the radius without member function
     C_2.radius = 6.0;    //Error because radius is a public
     return 0;
}
```

Private members can only be accessible from a member function

Private members cannot be accessible from other functions

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

Pointer to C++ Classes

NEW YORK UNIVERSITY
ABU DHABI

A pointer to a C++ class is done exactly the same way as a pointer to a structure and to access members of a pointer to a class you use the member access operator -> operator, just as you do with pointers to structures. Also as with all pointers, you must initialize the pointer before using it.

NEW YORK UNIVERSITY

ABU DHABI

```
#include <iostream>
class Circle
{
public:
    Circle() // constructor
    {
        cout<<"Define a Circle Object"<<endl;
    }
    Circle(const Circle &obj); // copy constructor

    ~Circle() // destructor
    {
        cout<<"Delete a Circle Object"<<endl;
    }

public:
      double center_x;  // center x position
      double center_y;  // center y position
      double area;      // area of the circle
      double area()    // calculate the area of a circle
      {
        return PI*radius*radius;
      }
      double getRadius()  // Obtain the value of radius
      {
          return radius;
      }
      void setRadius(double v)   // Set the value of radius
      {
          radius = v;

      }
private:
      double radius;    // radius

};
```

```
// Main function for the program
int main( )
{
    // Define the objects of Circle class
    Circle C_1; // Define the first circle

    Circle *ptrCircle; // Declare pointer to a class
    ptrCircle = &C_1;  // Assign the address of C_1 objec to pointer ptrCircle

    //Set the radius
    C_1.setRadius(6.0); //Set the value of radius

    //Access to member using pointer
    cout<<"The radius of circle is: "<< ptrCircle->getRadius() <<endl;

    return 0;
}
```

“This” Pointer

NEW YORK UNIVERSITY
ABU DHABI

This pointer: The own address of a C++ object (the memory address of every object).
This pointer is an implicit parameter to all member functions.

“This” Pointer


```
class Box
{
   public:
      int comparelen(Box box)
      {
         return this->length > box.length;
      }
   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};
```

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

C++ Overloading

NEW YORK UNIVERSITY
ABU DHABI

- Function Overloading in C++
- Operators Overloading in C++

Function Overloading

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

class ioData
{
public:
      int input1;
      double input2;
public:
      void setData(int i)
      {
        input1 = i;
       }
      void setData(double  f)
      {
        input2 = f;
      }
      void setData(int i, double f)
      {
          input1 = i;
          input2 = f;
      }
};

int main(void)
{
    ioData pd;
    // Call setData to set integer
    pd.setData(5);
    // Call setData to set float
    pd.setData(600.23);
    // Call setData to set integer and float
    pd.setData(5,600.2) ;
    return 0;
}
```

Operator Overloading

NEW YORK UNIVERSITY
ABU DHABI

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined.

Circle operator+(const Circle& other);

Operator Overloading

NEW YORK UNIVERSITY
ABU DHABI

For example, if we want to add two circle we defined above together, how to overload the operator +?

```
#include <iostream>
class Circle
{
public:
    Circle() // constructor
    {
        cout<<"Define a Circle Object"<<endl;
    }
    Circle(const Circle &obj); // copy constructor

    ~Circle() // destructor
    {
        cout<<"Delete a Circle Object"<<endl;
    }

public:
      double center_x;  // center x position
      double center_y;  // center y position
      double area;      // area of the circle
      double radius;    // radius
      double area()    // calculate the area of a circle
      {
        return PI*radius*radius;
      }
      double getRadius()  // Obtain the value of radius
      {
          return radius;
      }
      void setRadius(double v)   // Set the value of radius
      {
          radius = v;
      }

public:
    // Overload + operator to add two Circle objects.
      Circle operator+(const Circle& C)
      {
         Circle C1;
         C1.center_x = this->center_x + C.center_x ;
         C1.center_y = this->center_y + C.center_y ;
         C1.radius =  this->radius + C.radius;
         return C1;
      }
};
```

```
Circle::Circle(const Circle &obj)
{
    cout<< "Copy constructor called "<<endl;
    center_x = obj.center_x;
    center_y = obj.center_y;
    radius = obj.radius;
}

// Main function for the program
int main( )
{
    // Define the objects of Circle class
    Circle C_1; // Define the first circle
    Circle C_2; // Define the second cirlce

    //Set the parameters for C_1
    C_1.setRadius(6.0); //Set the value of radius
    C_1.center_x = 1.0;
    C_1.center_y = 2.0;

    //Set the parameters for C_2
    C_2.setRadius(4.0); //Set the value of radius
    C_2.center_x = 3.0;
    C_2.center_y = 4.0;

    //Add two cirlces together
    Circle C_3;
    C_3 = C_1 + C_2;
    return 0;
}
```

NEW YORK UNIVERSITY

ABU DHABI

How to overload the operator > for class circle?

NEW YORK UNIVERSITY

ABU DHABI

- Class Definitions
- C++ Class Members
- C++ Objects
- Pointers to C++ Classes
- C++ Class Overloading
- C++ Class Template

C++ Class Template

NEW YORK UNIVERSITY
ABU DHABI

we can also define class templates. The general form of a generic class declaration is shown here:

```
template <class type>
class classname
{

}
```

Example:

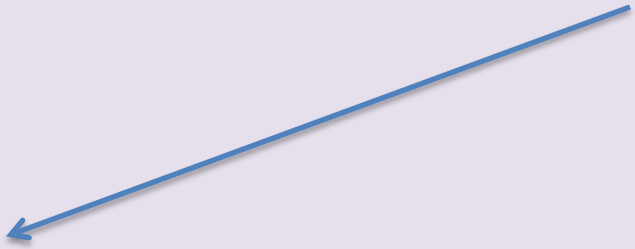
```
template <class T>
class Box
{
  private:
    T height;           // elements
    T weight;
    T length;
  public:
    T calvolume()
    {
        return this->height*this->length*this->weight;
    }
};
```

Example:

NEW YORK UNIVERSITY
ABU DHABI

```
#include <assert.h> // for assert()

class IntArray
{
private:
    int m_nLength;
    int *m_pnData;
 public:
    IntArray()
    {
        m_nLength = 0;
        m_pnData = 0;
    }
    IntArray(int nLength)
    {
        m_pnData = new int[nLength];
        m_nLength = nLength;
    }
    ~IntArray()
    {
        delete[] m_pnData;
    }
    void Erase()
    {
        delete[] m_pnData;
        // We need to make sure we set m_pnData to 0 here, otherwise it will
        // be left pointing at deallocated memory!
        m_pnData = 0;
        m_nLength = 0;
    }
    int& operator[](int nIndex)
    {
        assert(nIndex >= 0 && nIndex < m_nLength);
        return m_pnData[nIndex];
    }
     int GetLength() { return m_nLength; }
};
```



IntArray

DoubleArray

```
class DoubleArray
{
private:
    int m_nLength;
    double *m_pdData;

public:
    DoubleArray()
    {
        m_nLength = 0;
        m_pdData= 0;
    }

    DoubleArray(int nLength)
    {
        m_pdData= new double[nLength];
        m_nLength = nLength;
    }

    ~DoubleArray()
    {
        delete[] m_pdData;
    }

    void Erase()
    {
        delete[] m_pdData;
        // We need to make sure we set m_pnData to 0 here, otherwise it will
        // be left pointing at deallocated memory!
        m_pdData= 0;
        m_nLength = 0;
    }

    double& operator[](int nIndex)
    {
        assert(nIndex >= 0 && nIndex < m_nLength);
        return m_pdData[nIndex];
    }

    // The length of the array is always an integer
    // It does not depend on the data type of the array
    int GetLength() { return m_nLength; }
};
```

NEW YORK UNIVERSITY

ABU DHABI

Template Array

```
template <typename T>
class Array
{
private:
    int m_nLength;
    T *m_ptData;

public:
    Array()
    {
        m_nLength = 0;
        m_ptData = 0;
    }

    Array(int nLength)
    {
        m_ptData= new T[nLength];
        m_nLength = nLength;
    }

    ~Array()
    {
        delete[] m_ptData;
    }

    void Erase()
    {
        delete[] m_ptData;
        // We need to make sure we set m_pnData to 0 here, otherwise it will
        // be left pointing at deallocated memory!
        m_ptData= 0;
        m_nLength = 0;
    }

    T& operator[](int nIndex)
    {
        assert(nIndex >= 0 && nIndex < m_nLength);
        return m_ptData[nIndex];
    }

    // The length of the array is always an integer
    // It does not depend on the data type of the array
    int GetLength(); // templated GetLength() function defined below
};
```