Week 5: Introduction to C++

CSCI 2100 Data Structures

Fall 2024

Tae-Hyuk (Ted) Ahn

Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

SAINT LOUIS
UNIVERSITY™
EST. 1818

Recap: Full Code

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Public constructor to initialize width and
height
      Rectangle (int a, int b) : width(a), height(b)
{}

      // Destructor
      ~Rectangle() {
         cout << "Destructor called" << endl;
      }

      // Public method to calculate the area
      int area () {
         return (width * height);
      }

   private:
      // Private member variables: these can't be
accessed directly from outside the class
      int width, height;
};
```

```
int main() {
   // Create a Rectangle object without a pointer (on
the stack)
   Rectangle rect1(5, 5);

   // Create a Rectangle object using a pointer (on
the heap)
   Rectangle* rect2 = new Rectangle(3, 4);

   // Access methods using the dot operator for rect1
   cout << "Area of rect1: " << rect1.area() << endl;

   // Access methods using the pointer (*rect2).area()
   cout << "Area of rect2 (using dereference): " <<
(*rect2).area() << endl;

   // Access methods using the arrow operator rect2-
>area()
   cout << "Area of rect2 (using arrow operator): " <<
rect2->area() << endl;

   // Free the dynamically allocated memory for rect2
   delete rect2;

   return 0;
}
```

Introduction to Classes and Objects

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      int width;
      int height;
};

int main() {
   // Create Rectangle objects
   Rectangle rect1, rect2;
   rect1.width = 10;
   rect1.height = 15;
   rect2.width = 20;
   rect2.height = 40;

   cout << "Rect1's width = " << rect1.width << " and
height = " << rect1.height << endl;
   cout << "Rect2's width = " << rect2.width << " and
height = " << rect2.height << endl;

   return 0;
}
```

- Class Definition:
  - A class is a blueprint for creating objects.
  - Objects are instances of a class.
- Public Member Variables:
  - Member variables of a class can be accessed directly if they are declared as `public`.

Code explanation:

- **Class Definition:** We define a simple class `Rectangle` with two public member variables, `width` and `height`.
- **Object Creation:** In `main()`, two objects `rect1` and `rect2` are created from the class Rectangle.
- **Accessing Member Variables:** The width and height of the rectangles are accessed and modified directly since they are public members.

Introduction to Constructors and Destructors

- Constructor:
  - Special member function that initializes an object.
  - Called when an object is created.
- Destructor:
  - Special member function that cleans up when an object is destroyed.
  - Called when an object goes out of scope or is deleted.

Introduction to Classes and Objects

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      int width;
      int height;
};

int main() {
   // Create Rectangle object without defining a
constructor
   Rectangle rect;

   cout << "Rect's width = " << rect.width << " and
height = " << rect.height << endl;

   return 0;
}
```

- Default Constructor:
  - If you do not define your own constructor, the compiler automatically provides a default constructor.
  - This constructor initializes member variables with garbage values (uninitialized values) if no initialization is provided.

```
Rect's width = 1651076199 and height = 779647075
```

Update the code as below and test it.

```
int width = 0;
int height = 0;
```

Parameterized Constructor (most common)

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Public constructor to initialize width and height
      Rectangle (int a, int b) : width(a), height(b) {}

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Create Rectangle object with custom constructor
   Rectangle rect1(5, 5);

   cout << "Area of rect1: " << rect1.area() << endl;
   return 0;
}
```

- Constructor:
  - A special member function that initializes an object when it is created.
  - Can be parameterized to initialize an object with specific values at the time of creation.

Explanation Notes:

•The constructor Rectangle(int a, int b) initializes the width and height when the object is created.

•No need to manually assign values after object creation.

Parameterized Constructor with Default Values

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Constructor with default values for width and height
      Rectangle(int a = 1, int b = 2) : width(a), height(b) {
         cout << "Constructor called with width = " << a << " and height = " << b << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Using constructor without arguments (default values)
   Rectangle rect1;

   // Using constructor with one argument (width is specified, height uses default value)
   Rectangle rect2(5);

   // Using constructor with both arguments
   Rectangle rect3(3, 4);

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;
   cout << "Area of rect3: " << rect3.area() << endl;

   return 0;
}
```

- Case 1: If the caller doesn't provide values for a or b, the default values (1,2) will be used.

```
Constructor called with width = 1 and height = 2
Constructor called with width = 5 and height = 2
Constructor called with width = 3 and height = 4
Area of rect1: 2
Area of rect2: 10
Area of rect3: 12
```

Parameterized Constructor with Default Values

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Constructor with default values for width and height
      Rectangle(int a = 1, int b = 2) : width(a), height(b) {
         cout << "Constructor called with width = " << a << " and height = " << b << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Using constructor without arguments (default values)
   Rectangle rect1;

   // Using constructor with one argument (width is specified, height uses default value)
   Rectangle rect2(5);

   // Using constructor with both arguments
   Rectangle rect3(3, 4);

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;
   cout << "Area of rect3: " << rect3.area() << endl;

   return 0;
}
```

- Case 1: If the caller doesn't provide values for a or b, the default values (1,2) will be used.
- Case 2: If the caller doesn't provide values for b??

```
Constructor called with width = 1 and height = 2
Constructor called with width = 5 and height = 2
Constructor called with width = 3 and height = 4
Area of rect1: 2
Area of rect2: 10
Area of rect3: 12
```

Parameterized Constructor with Default Values

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Constructor with default values for width and height
      Rectangle(int a = 1, int b = 2) : width(a), height(b) {
         cout << "Constructor called with width = " << a << " and height = " << b << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Using constructor without arguments (default values)
   Rectangle rect1;

   // Using constructor with one argument (width is specified, height uses default value)
   Rectangle rect2(5);

   // Using constructor with both arguments
   Rectangle rect3(3, 4);

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;
   cout << "Area of rect3: " << rect3.area() << endl;

   return 0;
}
```

- Case 1: If the caller doesn't provide values for a or b, the default values (1,2) will be used.
- Case 2: If the caller doesn't provide values for b, the default values of b (2) will be used.

```
Constructor called with width = 1 and height = 2
Constructor called with width = 5 and height = 2
Constructor called with width = 3 and height = 4
Area of rect1: 2
Area of rect2: 10
Area of rect3: 12
```

Parameterized Constructor with Default Values

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Constructor with default values for width and height
      Rectangle(int a = 1, int b = 2) : width(a), height(b) {
         cout << "Constructor called with width = " << a << " and height = " << b << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Using constructor without arguments (default values)
   Rectangle rect1;

   // Using constructor with one argument (width is specified, height uses default value)
   Rectangle rect2(5);

   // Using constructor with both arguments
   Rectangle rect3(3, 4);

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;
   cout << "Area of rect3: " << rect3.area() << endl;

   return 0;
}
```

- Case 1: If the caller doesn't provide values for a or b, the default values (1,2) will be used.
- Case 2: If the caller doesn't provide values for b, the default values of b (2) will be used.
- Case 3: Rectangle rect3(3, 4); uses both specified values width = 3 and height = 4.

```
Constructor called with width = 1 and height = 2
Constructor called with width = 5 and height = 2
Constructor called with width = 3 and height = 4
Area of rect1: 2
Area of rect2: 10
Area of rect3: 12
```

Copy Constructor in C++

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Parameterized constructor
      Rectangle (int a, int b) : width(a), height(b) {}

      // Copy constructor
      Rectangle(const Rectangle &obj) {
         width = obj.width;
         height = obj.height;
         cout << "Copy constructor called." << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   Rectangle rect1(5, 5);
   Rectangle rect2 = rect1; // Copy constructor is called here

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;

   return 0;
}
```

- Copy Constructor:
  - A special constructor used to create a copy of an existing object.
  - Called when an object is passed by value, returned from a function, or explicitly copied.

Understanding the const Keyword in the Copy Constructor

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Parameterized constructor
      Rectangle (int a, int b) : width(a), height(b) {}

      // Copy constructor
      Rectangle(const Rectangle &obj) {
         width = obj.width;
         height = obj.height;
         cout << "Copy constructor called." << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   Rectangle rect1(5, 5);
   Rectangle rect2 = rect1; // Copy constructor is called here

   cout << "Area of rect1: " << rect1.area() << endl;
   cout << "Area of rect2: " << rect2.area() << endl;

   return 0;
}
```

Key Points:

- const in a copy constructor:
  - const ensures that the object being copied is read-only inside the copy constructor.
  - Prevents the original object from being modified when passed by reference.
  - Improves safety by making it clear that the copy constructor does not alter the source object..

Note

- The original object is passed by reference to avoid copying.
- Adding const ensures that obj cannot be accidentally modified inside the constructor.
- It enforces a **read-only** rule, improving code reliability and preventing unintended side effects.

Destructor in C++

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      Rectangle(int a, int b) : width(a), height(b) {}

      // Destructor
      ~Rectangle() {
         cout << "Destructor called" << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   Rectangle rect1(5, 5);
   cout << "Area of rect1: " << rect1.area() << endl;
   // Destructor is called automatically when main ends
   return 0;
}
```

- Destructor:
  - Automatically called when the object goes out of scope or is deleted.
  - Used for cleanup, such as releasing resources or memory.

```
Area of rect1: 25
Destructor called
```

Constructor Overloading, but Careful!

```
class Rectangle {
   public:

      // Public constructor with default width and height
      Rectangle () {
         width = 10;
         height = 10;
         cout << "Constructor #1 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      // Parameterized constructor
      Rectangle (int a, int b) : width(a), height(b) {
         cout << "Constructor #2 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Create Rectangle object with custom constructor
   Rectangle rect1;
   cout << "Area of rect1: " << rect1.area() << endl;

   Rectangle rect2(5, 5);
   cout << "Area of rect2: " << rect2.area() << endl;
   return 0;
}
```

```
Constructor #1 called
Rect's width = 10 and height = 10
Area of rect1: 100
Constructor #2 called
Rect's width = 5 and height = 5
Area of rect2: 25
```

Constructor Overloading, but Careful!

```
class Rectangle {
   public:

      // Public constructor with default width and height
      Rectangle () {
         width = 10;
         height = 10;
         cout << "Constructor #1 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      // Parameterized constructor
      Rectangle (int a, int b) : width(a), height(b) {
         cout << "Constructor #2 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Create Rectangle object with custom constructor
   Rectangle rect1;
   cout << "Area of rect1: " << rect1.area() << endl;

   Rectangle rect2(5, 5);
   cout << "Area of rect2: " << rect2.area() << endl;
   return 0;
}
```

```
Constructor #1 called
Rect's width = 10 and height = 10
Area of rect1: 100
Constructor #2 called
Rect's width = 5 and height = 5
Area of rect2: 25
```

Constructor Overloading, but Careful!

```
class Rectangle {
   public:

      // Public constructor with default width and height
      Rectangle () {
         width = 10;
         height = 10;
         cout << "Constructor #1 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      // Parameterized constructor
      Rectangle (int a=10, int b=10) : width(a), height(b) {
         cout << "Constructor #2 called" << endl;
         cout << "Rect's width = " << width << " and height = " << height << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Create Rectangle object with custom constructor
   Rectangle rect1;
   cout << "Area of rect1: " << rect1.area() << endl;

   Rectangle rect2(5, 5);
   cout << "Area of rect2: " << rect2.area() << endl;
   return 0;
}
```

- NO!! Error!!

```
jdoodle.cpp: In function ‘int main()’:
jdoodle.cpp:149:14: error: call of overloaded ‘Rectangle()’ is ambiguous
  149 |    Rectangle rect1;
      |              ^~~~~
jdoodle.cpp:133:7: note: candidate: ‘Rectangle::Rectangle(int, int)’
  133 |       Rectangle (int a=10, int b=10) : width(a), height(b) {
      |       ^~~~~~~~~
jdoodle.cpp:125:7: note: candidate: ‘Rectangle::Rectangle()’
  125 |       Rectangle () {
      |       ^~~~~~~~~
```

Dynamic Memory Allocation (Using Pointers)

```
class Rectangle {
   public:
      Rectangle(int a, int b) : width(a), height(b) {}

      // Destructor
      ~Rectangle() {
         cout << "Destructor called" << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Create a Rectangle object using a pointer (on the heap)
   Rectangle* rect2 = new Rectangle(3, 4);

   // Access methods using the pointer
   cout << "Area of rect2: " << rect2->area() << endl;

   // Free the memory and call the destructor
   delete rect2;

   return 0;
}
```

- Key Points:
  - Objects can also be created dynamically on the heap using `new`.
  - Remember to use `delete` to free memory and call the destructor.

```
Area of rect2: 12
Destructor called
```

Accessing Methods (Dot vs Arrow Operator)

```
class Rectangle {
   public:
      Rectangle(int a, int b) : width(a), height(b) {}

      ~Rectangle() {
         cout << "Destructor called" << endl;
      }

      int area() {
         return width * height;
      }

   private:
      int width;
      int height;
};

int main() {
   // Stack object
   Rectangle rect1(5, 5);
   cout << "Area of rect1: " << rect1.area() << endl;

   // Heap object
   Rectangle* rect2 = new Rectangle(3, 4);
   cout << "Area of rect2 (using arrow operator): " << rect2->area()
<< endl;

   // Free the memory and call the destructor
   delete rect2;

   return 0;
}
```

- Key Points:
  - Use the dot operator for stack objects.
  - Use the arrow operator for heap-allocated objects (pointers).

```
Area of rect2: 12
Destructor called
```

Recap: Full Code

```
#include <iostream>

using namespace std;

class Rectangle {
   public:
      // Public constructor to initialize width and
height
      Rectangle (int a, int b) : width(a), height(b)
{}

      // Destructor
      ~Rectangle() {
         cout << "Destructor called" << endl;
      }

      // Public method to calculate the area
      int area () {
         return (width * height);
      }

   private:
      // Private member variables: these can't be
accessed directly from outside the class
      int width, height;
};
```

```
int main() {
   // Create a Rectangle object without a pointer (on
the stack)
   Rectangle rect1(5, 5);

   // Create a Rectangle object using a pointer (on
the heap)
   Rectangle* rect2 = new Rectangle(3, 4);

   // Access methods using the dot operator for rect1
   cout << "Area of rect1: " << rect1.area() << endl;

   // Access methods using the pointer (*rect2).area()
   cout << "Area of rect2 (using dereference): " <<
(*rect2).area() << endl;

   // Access methods using the arrow operator rect2-
>area()
   cout << "Area of rect2 (using arrow operator): " <<
rect2->area() << endl;

   // Free the dynamically allocated memory for rect2
   delete rect2;

   return 0;
}
```