Day 5

C++ Constructors and Operators Overloading

```
// C++ Default Constructor
/*
A constructor with no parameters is known as a default constructor.
C++ program to demonstrate the use of default constructor
*/

#include <iostream>
using namespace std;

// declare a class
class Wall{
  private:
    double length;

  public:
    // create a constructor
    Wall(){
      // initialize private variables
      length = 5.5;

      cout << "Create a wall." << endl;
      cout << "Length = " << length << endl;
    }
};

int main(){
  // create an object
  Wall wall1;

  return 0;
}


/*
Create a wall.
Length = 5.5

--------------------------------
Process exited after 0.1771 seconds with return value 0
*/
```

C++ Objects & Class

C++ Constructors

A constructor is a special type of member function that is called automatically when an object is created.
In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class  Wall {
  public:
   // create a constructor
   Wall() {
      // code
   }
};
```

Here, the function Wall() is a constructor of the class Wall. Notice that the constructor

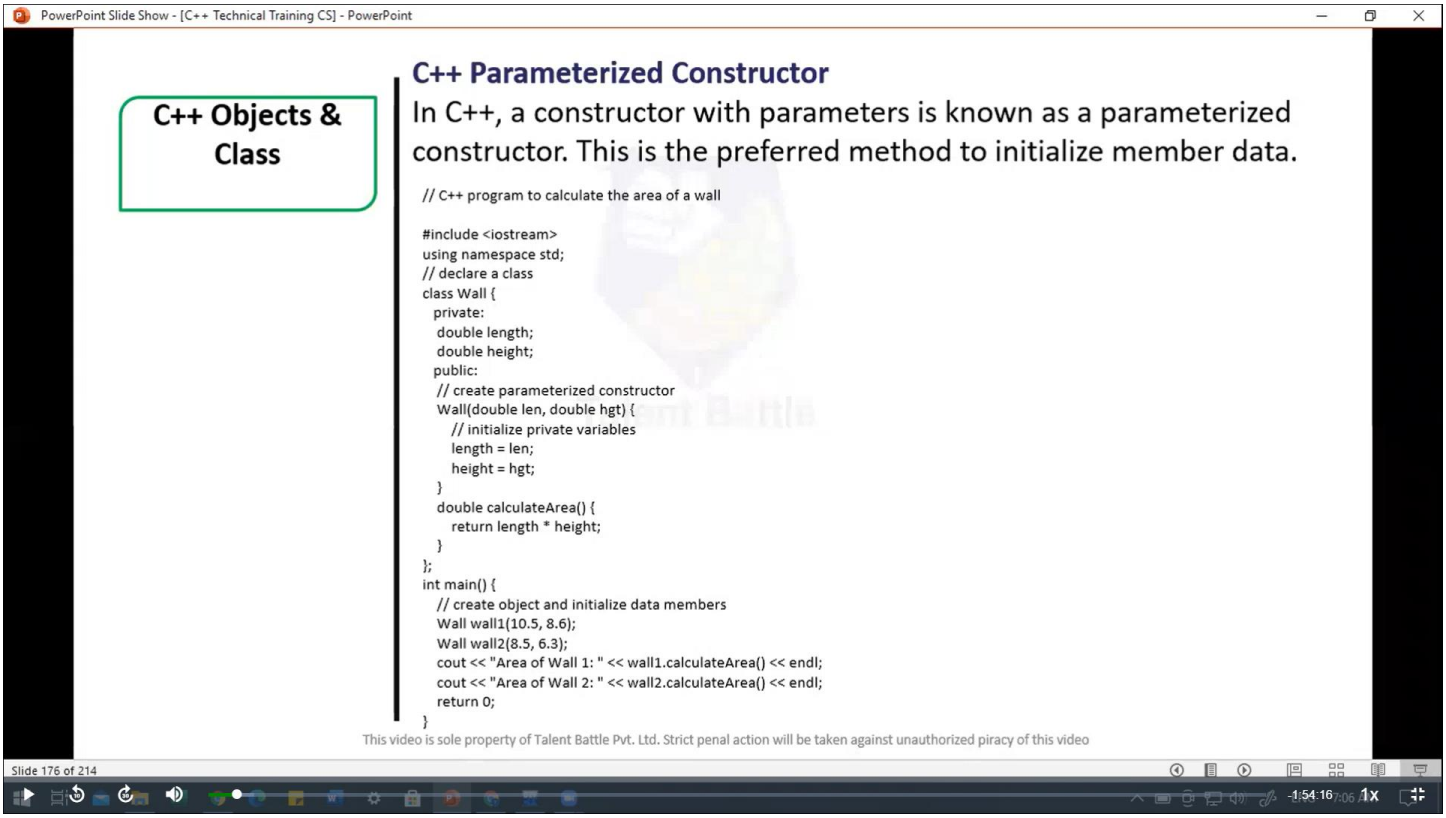
- has the same name as the class,
- does not have a return type, and
- is public

C++ Objects & Class

C++ Default Constructor

A constructor with no parameters is known as a default constructor.

```
// C++ program to demonstrate the use of default constructor

#include <iostream>
using namespace std;

// declare a class
class  Wall {

  private:
     double length;

  public:
   // create a constructor
   Wall() {

      // initialize private variables
      length = 5.5;

      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
   }
};

int main() {

   // create an object
   Wall wall1;

   return 0;
}
```




```
//=============================================
/*
C++ Parameterized Constructor
In C++, a constructor with parameters is known as a parameterized
constructor. This is the preferred method to initialize member data.

C++ Program to calculate the area of a wall
*/


#include <iostream>
using namespace std;

// declare a class
class Wall{
  private:
    double length;
    double height;

  public:
    // creae parameterized constructor
    Wall(double len, double hgt){
      // initialize private variables
      length = len;
      height = hgt;
    }
    double calculateArea(){
```

```
      return length * height;
    }
};

int main(){
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall2: " << wall2.calculateArea() << endl;

  return 0;
}


/*
Area of Wall1: 90.3
Area of Wall2: 53.55

--------------------------------
Process exited after 0.1177 seconds with return value 0
Press any key to continue . . .
*/
```

```cpp
//==================================================
// C++ Copy Constructor
// The copy constructor in C++ is used to copy data of one object to another

#include <iostream>
using namespace std;

// declare a class
class Wall{
  private:
    double length;
    double height;

  public:
    // parameterized constructor
    Wall(double len, double hgt){
      // initialize private variables
      length = len;
      height = hgt;
    }

    // copy constructor with a wall object as parameter
    Wall(Wall &obj){
      // initialize private variables
      length = obj.length;
      height = obj.height;
    }

    double calculateArea(){
      return length * height;
    }
};

int main(){
  // Create an object of Wall class
  Wall wall1(10.5, 8.6);

  // print area of wall1
  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;

  // Copy contents of wall1 to another object wall2
  Wall wall2 = wall1;

  // print area of wall2
  cout << "Area of Wall 2: " << wall2.calculateArea() << endl;

  return 0;
}
```

```
/*
Area of Wall 1: 90.3
Area of Wall 2: 90.3

--------------------------------
Process exited after 0.1467 seconds with return value 0
Press any key to continue . . .
*/
```

C++ Objects & Class

Notice that the parameter of this constructor has the address of an object of the Wall class.

We then assign the values of the variables of the first object to the corresponding variables of the second object. This is how the contents of the object are copied.

In main(), we then create two objects wall1 and wall2 and then copy the contents of the first object to the second with the code

Wall wall2 = wall1;

Note: A constructor is primarily used to initialize objects. They are also used to run a default code when an object is created.

C++ Objects & Class

How to pass and return object from C++ Functions?

In C++ programming, we can pass objects to a function in a similar manner as passing regular arguments.

```
// C++ program to calculate the average marks of two students

#include <iostream>
using namespace std;

class Student {
  public:
   double marks;
   // constructor to initialize marks
   Student(double m) {
      marks = m;
   }
};
// function that has objects as parameters
void calculateAverage(Student s1, Student s2) {
   // calculate the average of marks of s1 and s2
   double average = (s1.marks + s2.marks) / 2;
  cout << "Average Marks = " << average << endl;
}
int main() {
   Student student1(88.0), student2(56.0);
  // pass the objects as arguments
   calculateAverage(student1, student2);
   return 0;
}
```

```
//=========================================================
/*
How to pass and return object from C++ functions?

In C++ programming, we can pass objects to a function in a
similar manner as passing  regular arguments.
*/

// C++ program to calculate the average marks of two students

#include<iostream>
using  namespace std;

class Student{
  public:
    double marks;
    // constructor to initialize marks
    Student(double m){
      marks = m;
    }
};

// function that has object as parameters
void calculateAverage(Student s1, Student s2){
  // calculate the average of marks of s1 and s2
  double average = (s1.marks + s2.marks) / 2;
  cout << "Average Marks = " << average << endl;
}

int main(){
  Student student1(88.0), student2(56.0);
  // pass the objects as arguments
  calculateAverage(student1, student2);
  return 0;
}


/*
Average Marks = 72

--------------------------------
Process exited after 0.1455 seconds with return value 0
Press any key to continue . . .
*/
```

C++ Objects & Class

```
#include<iostream>

class Student {...};

void calculateAverage(Student s1, Student s2) {
    // code
}

int main() {
    ... ...
    calculateAverage(student1, student2);
    ... ...
}
```


C++ Objects & Class

C++ Return Object from a Function

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

class Student {
  public:
   double marks1, marks2;
};

// function that returns object of Student
Student createStudent() {
   Student student;

   // Initialize member variables of Student
   student.marks1 = 96.5;
   student.marks2 = 75.0;

   // print member variables of Student
   cout << "Marks 1 = " << student.marks1 << endl;
   cout << "Marks 2 = " << student.marks2 << endl;

   return student;
}

int main() {
   Student student1;

   // Call function
   student1 = createStudent();

   return 0;
}
```

```cpp
//=============================================
// C++ Return Object from a Function

#include<iostream>
using namespace std;

class Student{
  public:
    double marks1, marks2;
};

// function that returns object of Student
Student createStudent(){
  Student student;

  // initialize member variables of Student
  student.marks1 = 96.5;
  student.marks2 = 75.0;

  // print memeber variables of Student
  cout << "Marks 1 = " << student.marks1 << endl;
  cout << "Marks 2 = " << student.marks2 << endl;

  return student;
}

int main(){
  Student student1;

  // call function
  student1 = createStudent();

  return 0;
}



/*
Marks 1 = 96.5
Marks 2 = 75

--------------------------------
Process exited after 0.06273 seconds with return value 0
*/
```

C++ Objects & Class

```
#include<iostream>

class Student {...};

Student createStudent() {
    Student student;
    ... ...
    return student;
}

int main() {
    ... ...
    student1 = createStudent();
    ... ...
}
```

function call

C++ Objects & Class

C++ Operator Overloading

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading. For example,

Suppose we have created three objects c1, c2 and result from a class named Complex that represents complex numbers.

Since operator overloading allows us to change how operators work, we can redefine how the + operator works and use it to add the complex numbers of c1 and c2 by writing the following code:

result = c1 + c2;
instead of something like

result = c1.addNumbers(c2);
This makes our code intuitive and easy to understand.

Note: We cannot use operator overloading for fundamental data types like int, float, char and so on.

C++ Objects & Class

Syntax for C++ Operator Overloading

To overload an operator, we use a special operator function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className {
   ... .. ...
   public
     returnType operator symbol (arguments) {
        ... .. ...
     }
   ... .. ...
};
```

Here,
returnType is the return type of the function.
operator is a keyword.
symbol is the operator we want to overload. Like: +, <, -, ++, etc.
arguments is the arguments passed to the function.

C++ Objects & Class

Operator Overloading in Unary Operators

Unary operators operate on only one operand. The increment operator ++ and decrement operator -- are examples of unary operators.

```
// Overload ++ when used as prefix

#include <iostream>
using namespace std;
class Count {
  private:
   int value;
  public:
   // Constructor to initialize count to 5
   Count() : value(5) {}
   // Overload ++ when used as prefix
   void operator ++ () {
      ++value;
   }
   void display() {
      cout << "Count: " << value << endl;
   }
};

int main() {
   Count count1;
   // Call the "void operator ++ ()" function
   ++count1;
   count1.display();
   return 0;
}
```

Here, when we use ++count1;, the void operator ++ () is called. This increases the value attribute for the object count1 by 1.

```
//=====================================================
/*
Operator Overloading in Unary Operators
--> Unaryoperatrs operate on only one operand. The increment
operator ++ and decrement operator -- are example of unary
operators.

Here, when we use ++count1;, the void operator ++ () is
called.
This increasees the value attribute for the object
coutn1 by 1.
*/

// Overload ++ when used as prefix

#include <iostream>
using namespace std;

class Count {
  private:
    int value;

  public:
    // Constructor to initialize count to 5
    Count(): value(5){}
    // Overload ++ when used as prefix
    void operator ++ (){
      ++value;
    }

    void display(){
      cout << "Count: " << value << endl;
    }

};

int main(){
  Count count1;
  // Call the "void operator ++ ()" function
  ++count1;
  count1.display();
  return 0;
}


/*
Count: 6

--------------------------------
```

```
Process exited after 0.06413 seconds with return value 0
*/
```

C++ Objects & Class

Operator Overloading in Binary Operators

Binary operators work on two operands.
For example,
result = num + 9;
Here, + is a binary operator that works on the operands num and 9.

When we overload the binary operator for user-defined types by using the code:

obj3 = obj1 + obj2;
The operator function is called using the obj1 object and obj2 is passed as an argument to the function.

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

class Time{
  public:
    int h, m, s;
  public:
    Time(){
      h = 0, m = 0, s = 0;
    }

    void setTime();
    void show(){
      cout << h << ":" << m << ":" << s;
    }
    Time operator +(Time);
};

Time Time::operator +(Time t1){
  Time t;
  int a, b;
  a = s + t1.s;
  t.s = a % 60;

  b = (a / 60) + m + t1.m;
  t.m = b % 60;
  t.h = (b / 60) + h + t1.h;
  t.h = t.h%12;
  return t;
}

void Time::setTime(){
  cout << "\n enter hour ";
  cin >> h;
  cout << "\n enter minute ";
  cin >> m;
  cout << "\n enter seconds ";
  cin >> s;
}

int main(){
  Time t1, t2, t3;
  cout << "\n Enter first time: ";
  t1.setTime();
  cout << "\n Enter second time: ";
  t2.setTime();
  t3 = t1 + t2;
  cout << "\n First time ";
```

```
  t1.show();
  cout << "\n Second time: ";
  t2.show();

  cout << "\n T3 = T1 + T2: ";
  t3.show();

  return 0;
}


/*

 Enter first time:
 enter hour 1

 enter minute 2

 enter seconds 22

 Enter second time:
 enter hour 3

 enter minute 33

 enter seconds 55

 First time 1:2:22
 Second time: 3:33:55
 T3 = T1 + T2: 4:36:17
--------------------------------
Process exited after 19.3 seconds with return value 0
*/
```