SLIIT

Discover Your Future

IT1050 - Object Oriented Concepts

Lecture-07

Constructors and Destructors

Learning Outcomes

- At the end of the Lecture students should be able to
  - Use Overloading
  - Use constructors and destructors.
  - Use dynamic objects.

Overloading Functions

```
// Prototype functions
void print();
void print(char msg[]);
void print(char msg[], int no);
```

- In C++, you can have multiple functions with the same name, but having different types of parameters.
- This is called function overloading.
- In overloading the parameters of each function should be different. The return type and the function name have to remain the same.

```
void print (dataType1, dataType2, dataType3);
```

Overloading Functions

```
// Prototype functions
void print();
void print(char msg[]);
void print(char msg[], int no);

// Prototype function implementation
void print() {
    cout << "Hello " << endl;
}
void print(char msg[]) {
    cout << "Hello " << msg << endl;
}
void print(char msg[], int no) {
    cout << msg << " : " << no << endl;
}
```

```
int main() {
    print();
    print("SLIIT");
    print("Age ", 19);
    return 0;
}
```

Output

```
Hello
Hello SLIIT
Age  : 19
```

Employee Class

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            void assignDetails(int pempno, char
                    pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

```
int main() {
        Employee emp;
        emp.printPaySlip();
        return 0;
}
```

Output

```
---------------------------------------
EmpNo               :142
Name                :
Basic Salary        :2.07322e-317
Allowance           :6.95252e-310
Net Salary          :0
---------------------------------------
```

How do we assign values to the emp object when it is created? The output shown above displays random values.

Note : We have not used setters and getters for simplicity

Employee Class – From Lab

```
class Employee {
    private:
        int empno;
        char name[20];
        double basicSal;
        double allowance;
        double salary;
    public:
        void assignDetails(int pempno, char
             pname[], double pbasicSal);
        void setAllowance(double pallowance);
        void calcSalary();
        void printPaySlip();
};
```

```
int main() {
    Employee emp;
    emp.assignDetails(0, "",0, 0);
    emp.printPaySlip();
    return 0;
}
```

Output

```
---------------------------------------
EmpNo               :0
Name                :
Basic Salary        :0
Allowance           :6.9528e-310
Net Salary          :0
---------------------------------------
```

We could call the assignDetails() method before printing,
But we are not handling the allowance in this method.

What if we use another assignDetails() method to set values at the beginning (initialize)

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            void assignDetails();
            void assignDetails(int pempno, char
                    pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

```
void Employee::assignDetails() {
        empno = 0;
        strcpy(name, "");
        basicSal = 0;
        allowance = 0;
}

void Employee::assignDetails(int pempno,
            char pname[], double pbasicSal) {
        empno = pempno;
        strcpy(name, pname);
        basicSal = pbasicSal;
}
```

void assignDetails(int pempno, char pname, pbasicSal) is now an overloaded method.

Using assignDetails() method

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            void assignDetails();
            void assignDetails(int pempno, char
                    pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

```
int main() {
        Employee emp;
        emp.assignDetails();
        emp.printPaySlip();
        return 0;
}
```

Output

```
---------------------------------------
EmpNo               :0
Name                :
Basic Salary        :0
Allowance           :0
Net Salary          :0
---------------------------------------
```

Now it seems to be okay. But what if we want to do this automatically when an object is created.

Initializing the Attributes

- Initialize means setting values to variables (attributes) at the beginning (usually when they are declared).
- Although we add another method to the class called “assignDetails()”, it does not initialize the attributes automatically, instead when called it assigns or overwrites the garbage values that are already in the attributes.
- We could automatically initialize attributes in a class by using a special kind of methods called “Constructors”

Using Constructors

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            Employee();
            Employee(int pempno, char
                  pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

```
// default constructor
Employee::Employee() {
        empno = 0;
        strcpy(name, "");
        basicSal = 0;
        allowance = 0;
}
// Constructor with parameters
Employee::Employee(int pempno,
            char pname[], double pbasicSal) {
        empno = pempno;
        strcpy(name, pname);
        basicSal = pbasicSal;
}
```

Constructors don't have a return type. They have the same name as the Class. **Employee()** is called the **default constructor**
And **Employee(int pempno, char ..)** is the **overloaded constructor**

Constructor

- The constructor is used to initialize the object when it is declared.
- The constructer does not return a value, and has no return type (not even void)
- The constructors has the same name as the class.
- There can be default constructors or constructors with parameters
- When an object is declared the appropriate constructor is executed.

Constructors

- Default Constructors
  - Can be used to initialized attributes to default values

```
Employee::Employee() {
        empno = 0;   strcpy(name, "");
        basicSal = 0;  allowance = 0;
}
```

- Overloaded Constructors (Constructors with Parameters)
  - Can be used to assign values sent by the main program as arguments

```
Employee::Employee(int pempno, char pname[], double pbasicSal) {
        empno = pempno;  strcpy(name, pname);
        basicSal = pbasicSal;
}
```

Using Default Constructor

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            Employee();
            Employee(int pempno, char
                    pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

```
int main() {
        Employee emp;
        emp.printPaySlip();
        return 0;
}
```

Output

```
---------------------------------------
EmpNo           :0
Name            :
Basic Salary    :0
Allowance       :0
Net Salary      :0
---------------------------------------
```

When we create the emp object in the main() program, the default constructor is automatically executed.

Using Overloaded Constructor

```
class Employee {
    private:
            int empno;
            char name[20];
            double basicSal;
            double allowance;
            double salary;
    public:
            Employee();
            Employee(int pempno, char
                    pname[], double pbasicSal);
            void setAllowance(double pallowance);
            void calcSalary();
            void printPaySlip();
};
```

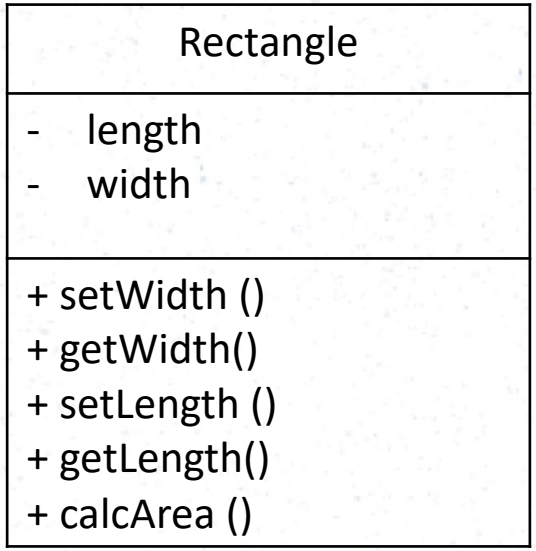
```
int main() {
        Employee emp(100,
                "Niranjan", 5000);
        emp.printPaySlip();
        return 0;
}
```

Output

```
---------------------------------------
EmpNo               :100
Name                :Niranjan
Basic Salary        :5000
Allowance           :0
Net Salary          :0
---------------------------------------
```

When we create the emp object in the main() program, the overloaded constructor is automatically executed.

Exercise 1 - Constructors




```
class Rectangle {
   private:
      int width;
      int length;
   public:
      void setWidth(int w);
      int getWidth();
      void setLength(int l);
      int getLength()
      int calcArea();
};
```

Implement the default and overloaded constructors

Solution

```
// default Constructor
Rectangle::Rectangle () {
  length = 0;
  width = 0;
}

// Constructor with parameters
Rectangle::Rectangle (int l, int w) {
  length = l;
  width = w;
}
```

```
class Rectangle {
   private:
     int width;
     int length;
   public:
     Rectangle();
     Rectangle(int l, int w );
     void setWidth(int w);
     int getWidth();
     void setLength(int l);
     int getLength();
     int calcArea();
};
```

Exercise 2 – Create two objects using the two Constructors.

```
class Rectangle {
   private:
      int width;
      int length;
   public:
      Rectangle();
      Rectangle( int l, int w);
      void setWidth(int w);
      int getWidth();
      void setLength(int l);
      int getLength();
      int calcArea();
};
```

Rectangle Class Solution

```
18  Rectangle::Rectangle()  {
19     length = 0;
20     width = 0;
21  }
22  Rectangle::Rectangle(int l, int w) {
23      length = l;
24      width = w;
25  }
```

```
43  int main() {
44    Rectangle rec1;
45    Rectangle rec2(10, 5);
46
47    cout << "Rectangle 1 = length - "
48         << rec1.getLength()
49         << ", width - "
50         << rec1.getWidth()
51         << endl;
52
53      cout << "Rectangle 2 = length - "
54         << rec2.getLength()
55         << ", width - "
56         << rec2.getWidth()
57         << endl;
58
59    return 0;
60  }
```

```
4   class Rectangle {
5      private:
6         int width;
7         int length;
8      public:
9          Rectangle();
10         Rectangle(int l, int w );
11         void setWidth(int w);
12         int getWidth();
13         void setLength(int l);
14         int getLength();
15         int calcArea();
16  };
```

Output

```
Rectangle 1 = length - 0, width - 0
Rectangle 2 = length - 10, width - 5
```

Destructor

- A class's destructor is called implicitly when a object is destroyed.
- An object gets destroyed when a program execution terminates or leaves the scope in which the object was created.
- A destructor can be used to release memory of attributes that were created dynamically when the object was created.
- The name of the destructors is as same as the class name with a ~ (tilde) at the beginning
- It does not specifies any parameters or a return type.

Destructor

```
Rectangle::~Rectangle () {
    cout << “Destructor runs” << endl;
}
```

```
class Rectangle {
   private:
     int width;
     int length;
   public:
     Rectangle();
     Rectangle( int w, int l );
     void setWidth(int w);
     int getWidth();
     void setLength(int l);
     int getLength();
     int calcArea();
    ~Rectangle();
};
```

Destructors used in Rectangle Class

```
27  Rectangle::~Rectangle() {
28     cout << "Destructor Runs for Rec with Len = "
29       << length << " and width = " << width <<  endl;
30  }
```

```
43  int main() {
44    Rectangle rec1;
45    Rectangle rec2(10, 5);
46
47    cout << "Rectangle 1 = length - "
48         << rec1.getLength()
49         << ", width - "
50         << rec1.getWidth()
51         << endl;
52
53      cout << "Rectangle 2 = length - "
54         << rec2.getLength()
55         << ", width - "
56         << rec2.getWidth()
57         << endl;
58
59    return 0;
60  }
```

```
4   class Rectangle {
5      private:
6         int width;
7         int length;
8      public:
9          Rectangle();
10         Rectangle(int l, int w );
11         void setWidth(int w);
12         int getWidth();
13         void setLength(int l);
14         int getLength();
15         int calcArea();
16         ~Rectangle();
17  };
```

Output

```
Rectangle 1 = length - 0, width - 0
Rectangle 2 = length - 10, width - 5
Destructor Runs for Rec with Len = 10 and width = 5
Destructor Runs for Rec with Len = 0 and width = 0
```

Static Objects

- Using Default Constructor

```
Rectangle R1;
```

R1
length - 0
width – 0

- Using constructor with Parameters

```
Rectangle R2 ( 100, 50) ;
```

R2
length - 100
width – 50

Static Objects- Accessing methods

```
Rectangle R1;

R1.setLength( 100 );
R1.setWidth(50);
```

The dot (.) operator is used to access the public methods of a static object

Dynamic Objects

```
Rectangle *r;

r = new Rectangle ();

r -> setWidth(100);
r -> setLength(50);
cout<<“Area is : “<< r -> calcArea();

delete r;
```

The arrow (->) is used to access the public methods of a dynamic object

Dynamic Objects

- Most programming languages only support Dynamic Objects.
- You need to delete the allocated memory in C++.
- The new command is used to allocate memory for an object.
- The delete command needs to be used to deallocate memory (release memory) once we have finished using the objects.

Dynamic Objects

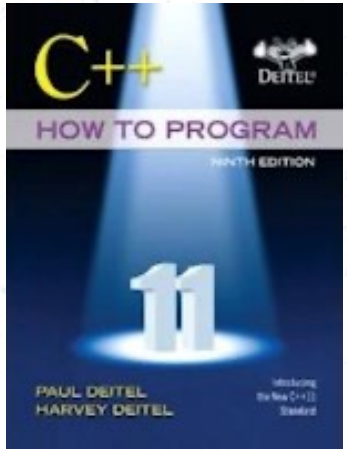
```
64    Rectangle *rec3, *rec4;
65
66    rec3 = new Rectangle();
67    rec4 = new Rectangle(20, 10);
68
69      cout << "Rectangle 3 = length - "
70         << rec3->getLength()
71         << ", width - "
72         << rec3->getWidth()
73         << endl;
74
75      cout << "Rectangle 4 = length - "
76         << rec4->getLength()
77         << ", width - "
78         << rec4->getWidth()
79         << endl;
80
81      delete rec3;
82      delete rec4;
83
```

```
 4  class Rectangle {
 5     private:
 6        int width;
 7        int length;
 8     public:
 9         Rectangle();
10         Rectangle(int l, int w );
11         void setWidth(int w);
12         int getWidth();
13         void setLength(int l);
14         int getLength();
15         int calcArea();
16         ~Rectangle();
17  };
```

Output

```
Rectangle 3 = length - 0, width - 0
Rectangle 4 = length - 20, width - 10
Destructor Runs for Rec with Len = 0 and width = 0
Destructor Runs for Rec with Len = 20 and width = 10
```

Reference



Chapter 09 & 10

Constructors and Destructors

Deitel & Deitel's (2016), C++ How to Program, 9th Edition