Unit - 3

Constructors, Destructors and Operator Overloading

Constructors

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type at all, not even void.
- Constructors can be very useful for setting initial values for certain member variables.
- The constructor enables an object to initialize itself when it is created. This is known as *automatic initialization of objects*.
- C++ also provides another member function called the **destructor** that destroys the objects when they are no longer required.

- Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator.

```
class A {
private:
        int i;
public:
        A(); //Constructor declared
};
A::A() // Constructor definition
{
        i=1;
}
```

Characteristics of constructor

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void, and therefore they cannot return values.
- They cannot be inherited. However, the derived class can call the base class constructor.
- Constructors cannot be virtual.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body)
- We can declare more than one constructor in a class. These constructor differ in there parameter list.

Types of constructor

- Constructors are of three types :
  - Default Constructor,
  - Parameterized Constructor and
  - Copy Constructor.

Default Constructor

- Default constructor is the constructor which doesn't take any argument.
- If the programmer does not specify the constructor in the program then compiler provides the default constructor automatically.
- Default constructor get called automatically when objects of class get created.
- We don’t need to call default constructor explicitly. Like using object name and **(.)** operator

```
class Rectangle {
private:
        int length, breadth;
public:
        Rectangle() {
                length=5;
                breadth=6;
        }
        void area() {
                int  a=(length*breadth);
                cout<<“area is”<<a;
        }
 };
```

```
int main() {
        Rectangle r1;
        r1.area();
        return 0;
}
```

Parameterized Constructors

- These constructors take parameters. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.
- This constructor cannot be called using object name and **(.)** operator.
- We have to call this constructor when object is created, by specifying appropriate parameters.

```
class Example {
private:
    int  m, n;
public:
    Example(int x, int y); // parameterized constructor
    ..............
    ..............
};
Example : : Example(int x, int y) {
    m = x;
     n = y;
}
```

```
class Rectangle {
private:
    int len, wid;
public:
    Rectangle(int len, int wid) {
        this->len = len;
        this->wid = wid;
    }
    void show() {
        cout << "The sides of rectangle are " << len << " " << wid << endl;
    }
};
int main() {
    Rectangle r1 = Rectangle(10,20); //Explicit call
    Rectangle r2(20,40); //Implicit call
    r1.show();
    r2.show();
    return 0;
}
```

Important Note

- The parameters of a constructor can be of any type except that of the class to which it belongs.
- However, a constructor can accept a reference to its own class as a parameter. In such cases, the constructor is called a copy constructor.

```
class A {
        ....
        ....
public:
        A(A);
};
```

Not legal

```
class A {
        ....
        ....
public:
        A(A &);
};
```

Legal

Copy Constructors

- A copy constructor is used to declare and initialize an object from another object.
- A copy constructor creates a new object using an existing object of the same class and initializes each data member of newly created object with corresponding data member of existing object passed as argument.
- Since it creates a copy of an existing object so it is called copy constructor.

```
class Rectangle {
private:
    int len, wid;
public:
    Rectangle() { }
    Rectangle(int len, int wid) {
        this->len = len;
        this->wid = wid;
    }
    Rectangle(const Rectangle &ob) {
        this->len = ob.len;
        this->wid = ob.wid;
    }
    void show() {
        cout << len << " " << wid;
};
```

```
int main() {
    Rectangle r1 = Rectangle(10,20);
    Rectangle r2 = r1; //Copy Constructor
    Rectangle r3;
    r3 = r2; //Assignment Operator
    r1.show();
    r2.show();
    r3.show();
    return 0;
}
```

Copy constructor Vs. Assignment Operator

- Rectangle r1, r2;
- Rectangle r3 = r1; //Copy Constructor
- r2 = r1; //Assignment Operator
  - The Copy constructor and the assignment operators are used to initializing one object to another object.
  - The main difference between them is that the copy constructor creates a separate memory block for the new object.
  - But the assignment operator does not make new memory space. It uses the reference variable to point to the previous memory block.

Difference between Copy Constructor and Assignment Operator

Copy Constructor	Assignment Operator
The Copy constructor is basically an overloaded constructor	Assignment operator is basically an operator.
This initializes the new object with an already existing object	This assigns the value of one object to another object both of which are already exists.
Copy constructor is used when a new object is created with some existing object	This operator is used when we want to assign existing object to new object.
Both the objects uses separate memory locations.	One memory location is used but different reference variables are pointing to the same location.
If no copy constructor is defined in the class, the compiler provides one.	If the assignment operator is not overloaded then bitwise copy will be made

Constructor Overloading

- C++ permits us to use multiple constructors in the same class.
- When more than one constructor function is defined in the class it is known as Constructor overloading.
- Whenever we define one or more non-default constructors (with parameters) for a class, a default constructor (without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case.

```
class Rectangle
{
 int len, wid;
 public:
 Rectangle();                  // default constructor
 Rectangle(int a);                  //parameterized constructor
 Rectangle(int a,int b);    //parameterized constructor
 Rectangle(Rectangle &);    // copy constructor
};
```

Dynamic Constructors

- The constructors can also be used to allocate memory while creating objects.
- Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.
- The memory is allocated using the new operator.

Dynamic memory allocation in C++

- The new operator is used to allocate memory and the delete operator is used to deallocate memory.
- Syntax:
  - pointer-variable = **new** data-type;
- Examples:
  - int *p; p = new int;
  - int *p = new int(25);
  - float *q = new float(75.25);
  - int *p = new int[10];
- Syntax of delete:
  - **delete** pointer-variable;
- Examples:
  - delete p;
  - delete[] p;

```
class Array {
private:
    int n, *p;
public:
    Array(int n) {
        this->n = n;
        p = new int[n];
    }
    void readArray();
    void findSum();
    void printArray();
};
void Array :: readArray() {
    cout << "Enter " << n << " elements:" << endl;
    for(int i=0; i<n; i++)
        cin >> p[i];
}
void Array :: findSum() {
    int sum = 0;
    for(int i=0; i<n; i++)
        sum += p[i];
    cout << "Sum of array elements is " << sum << endl;
}
```

```
void Array :: printArray() {
	cout << "Array elements are..." << endl;
	for(int i=0; i<n; i++)
		cout << p[i] << " ";
	cout << endl;
}
int main() {
	int n;
	cout << "Enter n:";
	cin >> n;
	Array a(n);
	a.readArray();
	a.findSum();
	a.printArray();
	return 0;
}
```

```
class String {
private:
    int len;
    char *str;
public:
    String() {
        len = 0;
        str = new char[len+1];
    }
    String(char *s) {
        len = strlen(s);
        str = new char[len+1];
        strcpy(str, s);
    }
    void show() {
        cout << str << endl;
    }
    void join(String s1, String s2) {
        delete str;
        str = new char[s1.len+s2.len+1];
        strcpy(str, s1.str);
        strcat(str, s2.str);
    }
};
```

```
int main() {
    String fName("Shrinivas "), mName("Ramesh "), lastName("Mangalwede");
    String partName, fullName;
    partName.join(fName, mName);
    fullName.join(partName, lastName);
    fName.show();
    mName.show();
    lastName.show();
    partName.show();
    fullName.show();
    return 0;
}
```

Destructors

- Destructor is used to destroy the objects that have been created by a constructor.
- Like a constructor, the destructor is a member function whose name is same as the class name but is preceded by a tilde.
  - ~ClassName() {
  - }
- A destructor never takes any argument nor does it return any value.
- We cannot overload destructors.
- It will be invoked automatically by the compiler upon termination of the program (or block or function) to clean up storage that is no longer accessible.

Constructor/Destructor Execution

```
int obNum = 0;
class Test {
public:
    Test() {
        obNum++;
        cout << "Created an object with obNum: " << obNum << endl;
    }
    ~Test() {
        cout << "Destroyed object with obNum: " << obNum << endl;
        obNum--;
    }
};
```

Constructor/Destructor Execution

```
int main() {
    cout << "Inside main" << endl;
    Test t1, t2, t3;
    {
        cout << "Inside block 1" << endl;
        Test t4;
    }
    {
        cout << "Inside block 2" << endl;
        Test t5;
    }
    cout << "Inside main again" << endl;
    return 0;
}
```

Term Work 6

Term Work 6(a)

- Create a class called Employee with id, name, designation, salary as the data members. Write the following constructors/functions with the main function.
  - Default constructor to initialize the values for employee
  - Parameterized constructor to initialize the values employees with other designations.
  - display_detail() function to display details of the employee.
  - Destructor.

```
class Employee {
private:
    int id;
    string name, desg;
    int salary;
public:
    Employee() {
        id = 0;
        name = "";
        desg=  "";
        salary = 0;
    }
    Employee(int id, string name, string desg, int salary) {
        this->id = id;
        this->name = name;
        this->desg = desg;
        this->salary = salary;
    }
```

```
    void dispDetails() {
        cout << "Emplyee ID:" << id << endl;
        cout << "Emplyee Name:" << name << endl;
        cout << "Designation:" << desg << endl;
        cout << "Salary:" << salary << endl;
    }
    ~Employee() {
        cout << "Employee object destroyed" << endl;
    }
};
int main() {
    Employee e(1010, "SRM", "Professor",34650);
    e.dispDetails();
    return 0;
}
```

Term Work 6(b)

- Create a class called List that uses constructor to create a list of n elements. Write member functions to read and display the List object. Write destructor to free up the memory allocated.

```
class MyList {
private:
     int length;
     int *p;
public:
     MyList(int n) {
               length = n;
               p = new int[length];
    }
     void read();
     void print();
    ~MyList() {
               delete [] p;
               cout << endl << "List deleted from memory" << endl;
    }
};
void MyList :: read() {
     cout << "Enter elements:";
     for(int i=0; i<length; i++)
               cin >> p[i];
}
```

```
void MyList :: print() {
    cout << "Entered elements are:";
    for(int i=0; i<length; i++)
        cout << " " << p[i];
}
int main() {
    int n;
    cout << "Enter list size:";
    cin >> n;
    MyList list(n);
    list.read();
    list.print();
    return 0;
}
```

Term Work 6(c)

- Create a C++ class for football player object with the following attributes: player no., name, number of matches and number of goals scored in each match.
- The number of matches varies for each player.
- Write parameterized constructor which initializes player no., name, number of matches and creates an array for number of goals in each match dynamically.
- Write a destructor for the class player.

```
class Player {
private:
    int pNo, numMatches, *numGoals;
    string pName;
public:
    Player(int, string, int);
    void printInfo();
    ~Player() {
        cout << endl << "Player object deleted.";
        delete [] numGoals;
   }
};
void Player :: printInfo() {
        cout << "Player Name: " << pName << endl;
        cout << "Number of matches: " << numMatches << endl;
        cout << "Goals scored in each match..." << endl;
        for(int i=0; i<numMatches; i++)
                cout << " " << numGoals[i];
}
```

```
Player :: Player(int num, string name, int mat) {
      pNo = num;
      pName = name;
      numMatches =mat;
      numGoals = new int[numMatches];
      cout << "Enter goals scored in" << endl;
      for(int i=0; i<numMatches; i++) {
            cout << "Match " << i+1 << ":";
            cin >> numGoals[i];
   }
}
int main() {
      string name;
      int playerNum, matches;
      cout << "Enter player name:";          getline(cin, name);
      cin.sync();
      cout << "Enter player number:";        cin >> playerNum;
      cout << "Number of matches played:";   cin >> matches;
      Player p(playerNum, name, matches);
      p.printInfo();
      return 0;
}
```

Term Work 6(d)

- Create a class called IntArray with
  - Data members:
    - pointer to an integer array and
    - integer to hold the array length
  - Member functions:
    - A zero-arg constructor
    - A parameterized constructor with an array and its length as parameters
    - A copy constructor
    - Display array elements
    - Destructor
- Write the corresponding main()

```
class IntArray {
private:
    int len, *arr;
public:
    IntArray();
    IntArray(int [], int);
    IntArray(const IntArray &);
    void printArray();
};
IntArray :: IntArray() {
    len = 10;
    arr = new int[len];
    for(int i=0; i<len; i++)
            arr[i]=0;
}
```

```
IntArray :: IntArray(int a[], int n) {
   len = n;
   arr = new int[len];
   for(int i=0; i<len; i++)
         arr[i] = a[i];
}
IntArray ::IntArray(const IntArray&a) {
   len = a.len;
   arr = new int[len];
   for(int i=0; i<len; i++)
       arr[i] = a.arr[i];
}
```

```
void IntArray :: printArray() {
    for(int i=0; i<len; i++)
        cout << " " << arr[i];
    cout << endl;
}
int main() {
    cout << "Creating a default IntArray object..." << endl;
    IntArray ob1;
    cout << "Default IntArray object contents are:" << endl;
    ob1.printArray();
    cout << endl<< "Creating parameterized IntArray object..." << endl;
    int a[] = {1,2,3,4,5};
    IntArray ob2(a,5);
    cout << "Parameterized IntArray object contents are:" << endl;
    ob2.printArray();
    cout << endl << "Creating IntArray object using copy  constructor..." << endl;
    IntArray ob3(ob2);
    cout << "Contents of copied IntArray object are:" << endl;
    ob3.printArray();
    return 0;
}
```

Term Work - 4

Term Work4(a)

- Create an array using dynamic memory allocation. Write functions to perform the following:
  - Find the minimum element in the array
  - Find the maximum element in the array
  - Find the mean of the elements in the array

```
void readArray(int a[], int n) {
    cout<< "Enter " << n << " elements:";
    for(int i=0; i<n; i++)
        cin>> a[i];
}
void findMin(int a[], int n) {
    int minimum=a[0];
    for(int i=1; i<n; i++)
        if(a[i]<minimum)
            minimum = a[i];
    cout<< "Minimum element is " << minimum <<endl;
}
void findMax(int a[], int n) {
    int maximum=a[0];
    for(int i=1; i<n; i++)
    if(a[i]>maximum)
    maximum = a[i];
    cout<< "Maximum element is " << maximum <<endl;
}
```

```
void findMean(int a[], int n) {
    int sum = 0;
    for(int i=0; i<n; i++)
        sum += a[i];
    cout<< "Mean of the elements is " << sum/n <<endl;
}
int main() {
    int n;
    cout<< "How many elements?";
    cin>> n;
    int *a = new int[n];
    readArray(a, n);
    findMin(a, n);
    findMax(a, n);
    findMean(a, n);
    return 0;
}
```

Term Work 4(b)

- Develop a C++ program to illustrate dynamic allocation and deallocation of memory using new and delete operators for an array of n integers. Include functions to
  - read the elements
  - find the number of even and odd integers in the array and
  - display the elements

```
void readArray(int a[], int n) {
	cout<< "Enter " << n << " numbers:";
	for(int i=0; i<n; i++)
	cin>> a[i];
}
void findEvenOdd(int a[], int n) {
	int even[50], odd[50], j=0, k=0;
	for(int i=0; i<n; i++)
		if(a[i]%2==0)
				even[j++]=a[i];
		else
				odd[k++] = a[i];
	cout<< "Even numbers are:" <<endl;
	for(int i=0; i<j; i++)
		cout<< " " << even[i];
	cout<<endl<< "Odd numbers are:" <<endl;
	for(int i=0; i<k; i++)
		cout<< " " << odd[i];
}
```

```
void dispAll(int a[], int n) {
    cout<<endl<< "Array has following numbers:" <<endl;
    for(int i=0; i<n; i++)
            cout<< " " << a[i];
    cout << endl;
}
int main() {
    int n;
    cout<< "How many numbers?";
    cin>> n;
    int *a = new int[n];
    readArray(a, n);
    findEvenOdd(a, n);
    dispAll(a, n);
    return 0;
}
```

Term Work 4(c)

- Create a C++ application that receives n test scores from the user. The application computes the average of test scores. It also displays the scores in ascending order. The application does dynamic allocation of memory for storing the scores.

```
void readScores(int a[], int n) {
    cout<< "Enter " << n << " scores:";
    for(int i=0; i<n; i++)
        cin >> a[i];
}
void sortAnddisplay(int a[], int n) {
    for(int i=0; i<n-1; i++)
        for(int j=0; j<n-i-1; j++)
            if(a[j]>a[j+1]) {
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
        }
    cout << endl << "Scores in ascending order:" <<endl;
    for(int i=0; i<n; i++)
        cout<< " " << a[i];
    cout << endl;
}
```

```
void findAverage(int a[], int n) {
	int sum = 0;
	for(int i=0; i<n; i++)
		sum += a[i];
	cout << "Average of all scores is " << sum/n <<endl;
}
int main() {
	int n;
	cout<< "How many scores?";
	cin>> n;
	int *a = new int[n];
	readScores(a, n);
	findAverage(a, n);
	sortAnddisplay(a, n);
	return 0;
}
```

Term Work 4(d)

- Develop a C++ program to read records of n items as per the following structure:

```
struct Item {
  int itemCode;
  string name;
  float price;
};
```

- Create the array of structures using dynamic memory allocation. Write functions to
  - to read the records and
  - display item having highest price.

```
struct Item {
    int itemCode;
    string name;
    float price;
};
void readItems(Item a[], int n) {
    cout<< "Enter " << n << " items:" <<endl;
    for(int i=0; i<n; i++) {
        cout<< "Item Code:";
        cin >> a[i].itemCode;
        cout << "Name of the item:";
        cin >> a[i].name;
        cout << "Price:";
        cin >> a[i].price;
        cout << a[i].itemCode << a[i].name << a[i].price << endl;
   }
}
```

```
void findHighest(Item a[], int n) {
      int itemNum = 0;
      float highest = a[0].price;
      for(int i=1; i<n; i++)
            if(a[i].price > highest) {
                  highest = a[i].price;
                  itemNum = i;
       }
      cout << "Item with highest price:" << endl;
      cout << "Item Code:" << a[itemNum].itemCode << endl;
      cout << "Item Name:" << a[itemNum].name << endl;
      cout << "Price:" << a[itemNum].price << endl;
}
int main() {
      int n;
      cout << "How many items?";
      cin >> n;
    Item *a = new Item[n];
      readItems(a, n);
      findHighest(a, n);
      return 0;
}
```

Operator Overloading

- 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.
- It's a type of polymorphism in which an operator is overloaded to give it the user-defined meaning.
- Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.
- Syntax:

```
return_type <class_name> :: operator op(arg_list) {
    FunctionBody
}
```

Implementing of Operator overloading

- Member function: It is in the scope of the class in which it is declared.
- Friend function: It is a non-member function of a class with permission to access both private and protected members.
- Steps:
  - Create a class that defines the data type that is to be used in the overloading operation.
  - Declare the operator function operator op() in the public section of the class. It may be either a member function of a friend function.
  - Define the operator function to implement the required operations.

Member function Vs. friend function

- **Member function:**
  - The number of parameters to be passed is reduced by one, as the calling object is implicitly supplied as an operand**.**
  - Unary operators take no explicit parameters.
  - Binary operators take only one explicit parameter.
- **Friend Function:**
  - Friend function using operator overloading offers better flexibility to the class.
  - These functions are not a members of the class and they do not have 'this' pointer.
  - When you overload a unary operator you have to pass one argument.
  - When you overload a binary operator you have to pass two arguments.
  - Friend function can access private members of a class directly.

Overloading Unary Operators

- The unary operators operate on a single operand and following are the examples of Unary operators –
  - The increment (++) and decrement (--) operators.
  - The unary minus (-) operator.
  - The logical not (!) operator.
- The unary operators operate on the object for which they were called and normally, the operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

```
class Rectangle {
private:
	int length, width;
public:
	Rectangle(int length, int width) {
		this->length = length;
		this->width = width;
	}
	void operator ++(int) {
		length++;
		width++;
	}
	void operator ++() {
		length++;
		width++;
	}
	void showSides() {
		cout << "Length=" << length << " and Width=" << width << endl;
	}
};
int main() {
	Rectangle r1(4,5);
	r1.showSides();
	++r1;
	r1.showSides();
	r1++;
	r1.showSides();
	return 0;
}
```

```
class Distance {
private:
    int feet, inches;
public:
    Distance(int feet, int inches) {
        this->feet = feet;
        this->inches = inches;
    }
    void operator -() {
        feet = -feet;;
        inches = -inches;
    }
    void showDistance() {
        cout << feet << " Feet and " << inches << " Inches" << endl;
    }
};
int main() {
    Distance d1(4,-5);
    d1.showDistance();
    -d1;
    d1.showDistance();
    return 0;
}
```

```
class Distance {
private:
	int feet, inches;
public:
	Distance() {
		feet = 0;
		inches = 0;
	}
	Distance(int feet, int inches) {
		this->feet = feet;
		this->inches = inches;
	}
	friend void operator -(Distance &);
	void showDistance() {
		cout << feet << " Feet and " << inches << " Inches" << endl;
	}
};
void operator -(Distance &d) {
	d.feet = -d.feet;
	d.inches = -d.inches;
}
int main() {
	Distance d1(4,-5);
	d1.showDistance();
	-d1;
	d1.showDistance();
	return 0;
}
```

Overloading Binary Operators

```
class Currency {
private:
	int rupee, paise;
public:
	Currency() { rupee = paise = 0; }
	Currency(int rupee, int paise) {
		this->rupee = rupee;
		this->paise = paise;
	}
	void operator +(Currency c1) {
		rupee += c1.rupee;
		paise += c1.paise;
		if(paise>99) {
			rupee += 1;
			paise -= 100;
		}
	}
	void show() {
		cout << rupee << " Rupees and " << paise << " Paise" << endl;
	}
};
int main() {
	Currency c1(12,85), c2(3,25), c3;
	c1 + c2;
	c1.show();
	return 0;
}
```

```
class Currency {
private:
	int rupee, paise;
public:
	Currency() { rupee = paise = 0; }
	Currency(int rupee, int paise) {
		this->rupee = rupee;
		this->paise = paise;
	}
	Currency operator +(Currency c1) {
		rupee += c1.rupee;
		paise += c1.paise;
		if(paise>99) {
			rupee += 1;
			paise -= 100;
		}
		return *this;
	}
	void show() {
		cout << rupee << " Rupees and " << paise << " Paise" << endl;
	}
};
int main() {
	Currency c1(12,75), c2(3,50);
	c1 = c1 + c2;
	c1.show();
	return 0;
}
```

```
class Currency {
private:
	int rupee, paise;
public:
	Currency() { rupee = paise = 0; }
	Currency(int rupee, int paise) {
		this->rupee = rupee;
		this->paise = paise;
	}
	Currency operator +(Currency c1) {
		Currency result;
		result.paise = paise + c1.paise;
		result.rupee = rupee + c1.rupee;
		if(result.paise>99) {
			result.rupee += 1;
			result.paise -= 100;
		}
		return result;
	}
	void show() {
		cout << rupee << " Rupees and " << paise << " Paise" << endl;
	}
};
int main() {
	Currency c1(12,75), c2(3,50), c3;
	c3 = c1 + c2;
	c3.show();
	return 0;
}
```

```
class Currency {
private:
	int rupee, paise;
public:
	Currency() { rupee = paise = 0; }
	Currency(int rupee, int paise) {
		this->rupee = rupee;
		this->paise = paise;
	}
	friend Currency operator +(Currency c1, Currency c2) {
		Currency result;
		result.rupee = c1.rupee + c2.rupee;
		result.paise = c1.paise + c2.paise;
		if(result.paise>99) {
			result.rupee += 1;
			result.paise -= 100;
		}
		return result;
	}
	void show() {
		cout << rupee << " Rupees and " << paise << " Paise" << endl;
	}
};
int main() {
	Currency c1(12,85), c2(3,25), c3;
	c3 = c1 + c2;
	c3.show();
	return 0;
}
```