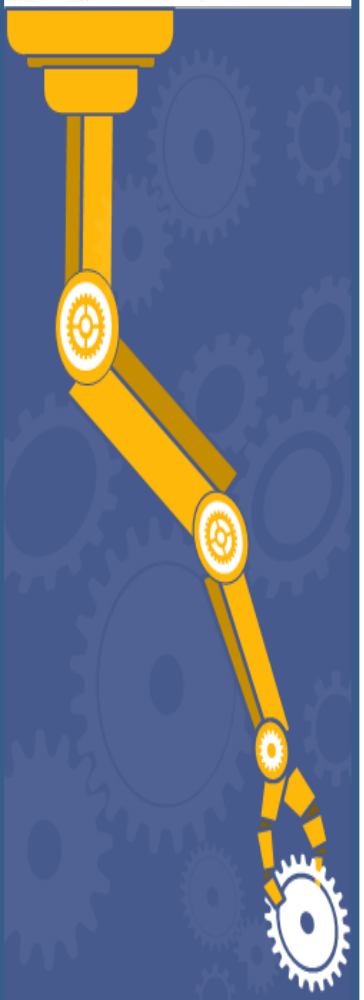
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



BCSE102L- Structured and object-oriented programming

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102P- Structured and object-oriented programming Laboratory

	Indicative Experiments
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-5: Overview of Object Oriented Programming**
  - **Features of OOP- Classes and Objects**
  - **Constructors and Destructors**
  - **Static Data Members and Member Functions**
  - **Inline Functions**
  - **Functions with Default Arguments**
  - **Functions with Objects as Arguments**
  - **Friend Functions and classes**

Constructor

- **Why Constructor??**
- In implementation of class, member functions are used to provide initial values to private member variables.
- For Example:
  - **X.getdata(100,75.45); passes initial values as arguments to the function getdata()**
  - **This values are assigned to private variables of object x.**
- All these function call statements are used with appropriate objects that already created.
- Observation??
- **These functions cannot be used to initialize the member variables at the time of creation of their objects.**

Constructor

- **Main philosophy of C++?**
- Creating any user defined data type such as **class** – behave like similar to basic built in data types.
- **We can able to initialize a class type variable(object) when it is declared as like same initialization of ordinary variable.**
- **For example:**
  - **int x=10;**
  - **float y=22.14;**
- C++ provides a **special member function called constructor** which enables an object to initialize itself when it is created.
- Automatic initialization of objects.

Constructor

- It is a special member function having same name as it's class.
- It is used to initialize the objects of that class type with a legal initial value.
- Constructor is automatically called/invoked when object is created.
- They should be declared in the public section.
- They do not have any return type.
- It is called as " Constructor" because it constructs the value of data members of the class.

Constructor- Basic Example

```
Class integer
{
        int a,b;
        public:
                integer(void);  // Constructor declared
                ---------------
 };

 integer::integer(void)        // Constructor defined outside
 {
        a=0;
        b=0;
 }
```

Types of Constructors

- Four types of constructors in C++:
  1. Default Constructors
  2. Parameterized Constructors
  3. Copy Constructors
  4. Dynamic Constructors

Default Constructors

- A constructor which does not receive any parameters is called a Default Constructor or a Zero Argument Constructor.
- Every class object is initialized with the same set of values in the default constructor.
- Even if a constructor is not defined explicitly, the compiler will provide a default constructor implicitly.

Declared implicitly

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

class Employee {
  public:
    int age;

    // Default constructor.
    Employee() {
    }
};

int main() {
  // Object of class Employee declared.
  Employee e1;
  // Prints value assigned by default constructor.
  cout << e1.age;
  return 0;
}
```

Output: 0

Ways to use Default Constructors

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

class Employee {
  public:
    int age;

    // Default constructor.
    Employee() {
      /* Data member is defined with the help of the
      default constructor.*/
      age = 50;
    }
};

int main() {
  // Object of class Employee declared.
  Employee e1;
  // Prints value assigned by default constructor.
  cout << e1.age;
  return 0;
}
```

Output: 50

Parameterized Constructor

```
Class integer
{
 int a,b;
 public:
 integer(int x, int y); // parametrized Constructor declared
                ----------
 };

 integer::integer(int x, int y)// parametrized Constructor defined
 {
        a=x;
        b=y;
 }
```

Parametrized Constructors

- Using the default constructor, it is impossible to initialize different objects with different initial values.
- A Parameterized Constructor is a constructor that can accept one or more arguments.
- This helps programmers assign varied initial values to an object at the creation time.

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

class Employee {
  public:
    int age;
    // Parameterized constructor
    Employee(int x) {
    /* Age assigned to value passed as an argument
    while object declaration.*/
      age = x;
    }
};

int main() {
  /* Object c1 declared with argument 40, which
  gets assigned to age.*/
  Employee c1(40);
  Employee c2(30);
  Employee c3(50);
  cout << c1.age << "\n";
  cout << c2.age << "\n";
  cout << c3.age << "\n";
  return 0;
}
```

```
40
30
50
```

Parameterized Constructor

- Passing initial values as arguments to the constructor function when an object is declared.
- **Can be done in two ways:**
  - Calling Constructor Explicitly
  - Calling Constructor implicitly

- **Explicit call**

integer x = integer(10,100); // *creates an object x and passes value 10 and 100 .*

- **Implicit call**

integer x(10,100);

Example

```
include<iostream.h>
using namespace std;
Class integer
{
    int a,b;
    public:
    integer(int , int); // parametrized Constructor declared
    Void display(void)
    {
        cout<<a<<"\n";
        cout<<b<<"\n";
    }
};
```

Example

```
integer::integer(int x, int y)// Constructor defined
{
        a=x; b=y;
}
 int main()
{
        integer ob1(10,100); // Implicit Call
        integer ob2 = integer(25,75); // Explicit Call
        ob1.display();
        ob2.display();
        return 0;
}
```

Copy Constructor

- A Copy constructor in C++ is a type of constructor used to create a copy of an already existing object of a class type.
- A copy constructor comes into the picture **whenever there is a need for an object** with the same values for data members as an already existing object. A copy constructor is invoked when an existing object is passed as a parameter.

- **integer(integer &i);**
- **integer I2(I1); - would define object I2 and at the same time initialize it to the values of I1.**
- **integer I2=I1;**

Copy Constructor

```
class Employee {
  private:
    // Data members
    int salary, experience;
  public:
    // Parameterized constructor
    Employee(int x1, int y1) {
      salary = x1;
      experience = y1;
    }
    // Copy constructor
    Employee(Employee &new_employee) {
      salary = new_employee.salary;
      experience = new_employee.experience;
    }
    void display() {
      cout << "Salary: " << salary << endl;
      cout << "Years of experience: " << experience << endl;
    }
};
// main function
int main() {
  // Parameterized constructor
  Employee employee1(34000, 2);
  // Copy constructor
  Employee employee2 = employee1;
  cout << "Employee1 using parameterized constructor : \n";
  employee1.display();
  cout << "Employee2 using copy constructor : \n";
  employee2.display();
  return 0;
}
```

OUTPUT

```
Employee1 using parameterized constructor :
Salary: 34000
Years of experience: 2
Employee2 using copy constructor :
Salary: 34000
Years of experience: 2
```

Dynamic Constructor

- While creating objects, When the allocation of memory is done dynamically using a dynamic memory allocator **"new"** in a constructor, it is known as a **Dynamic constructor.**
- By using this, we can dynamically initialize the objects i.e memory is allocated at run time.

Destructor

- As name implies, it is used to destroy the objects that have been created by a constructor.
- A Destructor is also a member function that is instantaneously called whenever an object is destroyed. The destructor is called automatically by the compiler when the object goes out of scope, i.e., When a function ends, the local objects created within it are also destroyed.
- The destructor has the same name as the class name, but the name is preceded by a tilde(~). A destructor has no return type and receives no parameters.

~ integer() { }

Dynamic Constructor

```
#include <iostream>
using namespace std;
class A
{
        int *value;
public:
A()    //Default constructor
{
        value = new int; //Memory allocation at run time
        *value = 1234;
}
A(int p_value) //Parameterised constructor
{
        value = new int; //Memory allocation at run time
        *value= p_value+3;
}
void display()
{
        cout<< *value <<endl;
}
~A()
{
        delete value ;
}
};
```

SYNTAX:

```
pointer_variable = new datatype;
```

Dynamic Constructor

```
int main()
{
A obj1, obj2(1550);
cout<<"The value of object obj1 is: ";
obj1.display();// Calling default constructor
cout<<"\nThe value of object 0bj2 is: ";
obj2.display();// calling parameterised constructor
return 0;
}
```

OUTPUT:

The value of object obj1 is: 1234
The value of object obj2 is: 1553

Creating Array of objects dynamically

```
class aobexample
 {
        int* count;
public:
        aobexample()
        {
                // allocating memory at run time and initializing the values
                count = new int[5] { 1, 2, 3, 4, 5 };
                for (int i = 0; i < 5; i++)
                {
                        cout << count[i] << " ";
                }
        }
};
int main()
{

        //creating an array of 5 objects
        aobexample* count = new aobexample[5]; //dynamically allocating object
}
```

Practice Exercise

- Write a C++ program to add two complex numbers. **Real, imag** should be considered as class members. Use default constructor for initializing the class members. Also use appropriate destructor.
- Write a C++ program to add two complex numbers. **Real, imag** should be considered as class members. Use parameterized constructor for initializing the class members. Also use appropriate destructor.
- Write a C++ program to calculate the area of rectangle for two different objects. **Len,breadth** should be considered as static class members. Appropriate static member functions should be used.
- Write a C++ program for calculating the Simple Interest through inline function.
- Define a class to represent a bank account, the data members should be name of the depositor, acc number, type of account, balance amount in account. The member functions are assign initial values, deposit an amount, withdraw an amount after checking balance , display name and balance.