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-7: POLYMORPHISM**
  - **Function Overloading**
  - **Operator Overloading**
  - **Dynamic Polymorphism**
  - **Virtual functions**
  - **Pure Virtual Functions**
  - **Abstract Classes**

Pure Virtual Functions

- Pure virtual functions are used
  - If a function doesn't have any use in the base class
  - But the function must be implemented by all its derived classes
- A pure virtual function doesn't have the function body and it must end with = 0
- The = 0 syntax doesn't mean we are assigning 0 to the function. **It's just the way we define pure virtual functions.**

Abstract class

- A class that contains a pure virtual function is known as an abstract class.
- We cannot create objects of an abstract class. However, we can derive classes from them, and use their data members and member functions (except pure virtual functions).

Pure Virtual Functions & Abstract Class - Example

```
class shape // Abstract Class
{
        public:
        double base,height;
        shape(double a, double b)
        {
        base=a;
        height=b;
        }
        virtual double area()
        {
                cout<<" base class area";
                return 0;
        }
};
```

virtual double area()=0;
// creating a pure virtual function

Pure Virtual Functions & Abstract Class - Example

```
class triangle: public shape
{
        triangle(double a, double b) : shape (a,b){ }
        double area()
        {
                cout<<" area of triangle";
                return (base *height)/2;
        }
};
class rectangle: public shape
{
        rectangle(double a, double b) : shape (a,b){ }
        double area()
        {
                cout<<" area of rectangle";
                return (base *height);
        }
};
```

Try what happens when commenting this area

Pure Virtual Functions & Abstract Class - Example

```
int main()
{
        shape s1(10.0,20.0);
        triangle t(10.0,20.0);
        cout<<t.area(); // static binding or early binding
        rectangle r(10.0,20.0);
        cout<<r.area(); // static binding or early binding
}
```

Try what happens when creating object to the base class

OUTPUT

Area of Triangle
100
Area of rectangle
200

Pure Virtual Functions & Abstract Class - Example

```
int main()
{
        shape *s;
        triangle t(10.0,20.0);
        s=&t;
        cout<<s->area(); // dynamic binding or late binding
        rectangle r(10.0,20.0);
        s=&r;
        cout<<s->area();
}
```

OUTPUT

Area of Triangle
100
Area of rectangle
200

Why Abstract class & pure Virtual Functions??

- Extendability
- Base classes that make the code reusable and extendable
- Must have at least one virtual function
- Cannot create objects for abstract classes
- Derived class must provide definition for the pure virtual function, otherwise it also becomes an abstract class.

Try Yourself

- Create an abstract class MathSymbol may provide a pure virtual function doOperation(), and create two more classes Plus and Minus implement doOperation() to provide concrete implementations of addition in Plus class and subtraction in Minus class.

Example Code

```
class Mathsymbol //Abstract class
{
     public :
          virtual void doOperation() = 0;// pure virtual function
};

class Plus : public Mathsymbol
{              //No doOperation() - Abstract
    public :void print()
    {
        cout << "Doing Addition" << endl;
    }
};
```

Example Code

```
class Minus : public Mathsymbol
{
    public :
    void doOperation()
    {    // Override Mathsymbol:: doOperation()
         int a=10,b=5;
         cout << "Subraction is = " <<a-b<< endl;
    }
};
int main()
{
    Minus m;
    Plus p;
}
```

Try Yourself

- Different types of employees based on the way in which salary is calculated .All employees has salary but that cannot be determined in base class. Force any derived class to implement it and Make it as pure virtual functions.