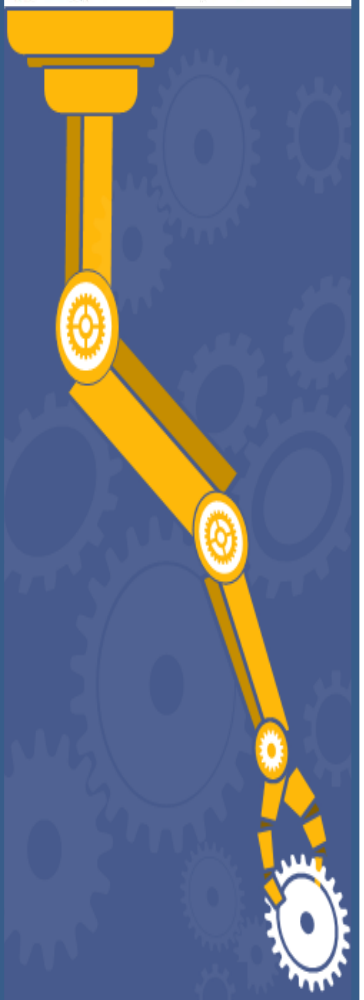
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



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**

Operator Overloading

- Operator overloading refers to overloading of one operator for many different purpose.
- For example: the binary + can be used to add two integer numbers , two float numbers, two structures variables, two union variables or two class objects.
- Use of operator overloading permits us to see no difference between built - in data types and user defined data types.
- It is one of the powerful feature of the C++ which give additional meaning to built-in standard operators like **+, -, *, /, >, <, < =, > = etc.**

Operator Overloading

Consider the Following example:

```
int main()
{
        int a=10, b=20;
        cout<<a+b<< endl;
        string s1= "abcd", s2= "xyz"
        cout<<s1+s2<< endl;
        return 0;
}
```

OUTPUT:

30
abcdxyz

Operator Overloading - Example-1- Normal way

```
class time
{
private :
        int hours;
        int minutes;
public:
        time(int h, int m ) // usage of constructor
        {
        hours=h; minutes=m;
        }
        addminute()
        {
                minutes++;
                if(minutes>=60)
                {
                hours++; minutes-=60;
                }
        }
Void displaytime( )
{
        cout<<hours<<":"<<minutes;
}
};
```

```
int main( )
{
        time t1(6,30);
        t1.addminute();
        t1.displaytime();
        return 0;
}
```

Operator Overloading - Example-1

```
class time
{
private :
        int hours;
        int minutes;
public:
        time(int h, int m )
        {
        hours=h; minutes=m;
        }
        addminute()
        {
                minutes++;
                if(minutes>=60)
                {
                hours++; minutes-=60;
                }
        }
Void displaytime( )
{
        cout<<hours<<":"<<minutes;
}
};
```

```
int main( )
{
        time t1(6,30);
        ++t1;
        t1.displaytime();
        return 0;
}
```

Error: No match for operator++

Is it possible? No

To do this we have to write a function by over loading the operator

Operator Overloading - Example-1

```
class time
{
private :
        int hours;
        int minutes;
public:
        time(int h, int m )
        {
        hours=h; minutes=m;
        }
        void operator ++()
        {
                minutes++;
                if(minutes>=60)
                {
                        hours++; minutes-=60;
                }
        }
Void displaytime( )
{
        cout<<hours<<":"<<minutes;
}
};

int main( )
{
        time t1(6,30);
        ++t1;
        t1.displaytime();
        return 0;
}
```

Alternate way of using
t1.operator ++()

Operator Overloading-Syntax

```
return_type operator operator (arguments)
{
        Statements;
        Statements;
        Statements;
}
```

operator which we want to overload

Keyword and it must be specified

Operator Overloading - Example-2- Adding integer to object

```
class demo
private :
        int num;
public:
        void input( )
        {
        cin>> num;
        }
        void operator +(int x)
        {
        num=num+x;
        }
        void show( )
        {
        cout<<"The num is "<<num<<endl;
        }
};
```

```
int main( )
{
        demo d1;
        d1.input( );
        d1.show( );
        int x;
        cout<<"Enter the value you
        want to add\n";
        cin>>x;
        d1+x;
        d1.show( );
}
```

```
123
The num is 123
Enter the value you want
to add
14
The num is 137
```

Operator Overloading - Example-3

```
class emp
{
private :
        int sal;
public :
        emp(int s)
        {
                sal=s;
        }
void operator>(emp temp)
{
        if(sal>temp.sal)
        cout<<"First employee's salary is higher"<<endl;
        else
        cout<<"Second employee's salary is higher"<<endl;
}
void show( )
{
cout<<"Salary is "<<sal<<endl;
}
};
```

Operator Overloading - Example-3

```
int main( )
{
        emp e1=emp(12545);
        emp e2=emp(13458);
        e1.show( );
        e2.show( );
        e1>e2;
        return 0;
}
```

Salary is 12545
Salary is 13458
Second employee's salary is higher

Rules for Operator Overloading

- Only built-in operators can be overloaded. New operators can not be created.
- Arity of the operators cannot be changed.
  - A unary function takes one argument.
  - A binary function takes two arguments.
  - A ternary function takes three arguments.
- Precedence and associativity of the operators cannot be changed.
- Overloaded operators cannot have default arguments except the function call operator () which can have default arguments.
- Operators cannot be overloaded for built in types only. At least one operand must be defined type.
- Assignment (=), subscript ([]), function call (“()”), and member selection (->) operators must be defined as member functions

Rules for Operator Overloading

- Except the operators specified in previous point, all other operators can be either member functions or a non member functions (Friend).
- Some operators like (assignment)=, (address)& and comma (,) are by default overloaded.
- Friend functions – cannot be used to overload some operators (=,[],(), ->)
- Member functions – cannot be used to overload some operators (<<, >>)
- Should be defined public