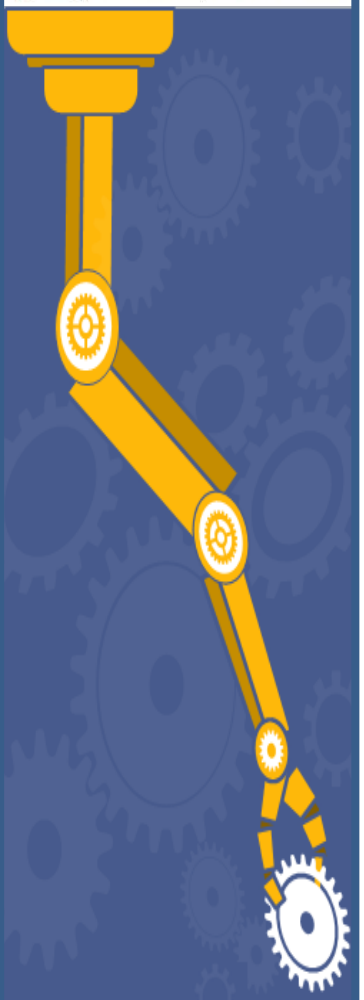
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



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-4: STRUCTURE AND UNION**
  - **Declaration and Access of Structure Variables**
  - **Arrays of Structure, Arrays within Structures**
  - **Structures and Functions**
  - **Pointers to Structure**
  - **Union**

Structure and Functions

- **C supports passing of structure values as arguments to functions.**
- Three methods
  - Pass each member of the structure as an actual argument of the function call. It is treated independently like ordinary variables.
  - Passing of a copy of the entire structure to the **called function**. Any changes to the structure members within the function are not reflected in the original structure(in the **calling function**).

  (Note: Necessary for the function to return the entire structure back to the calling function)
  - Pointers to pass the structure as an argument. In this, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it.

Structure and Functions

- General format of sending a copy of a structure to the called function

  function_name(structure_variable_name);

- Called function is represented in the following form

```
data_type function_name(struct_type    st_name)
        {
                ---------
                ----------
                return(expression);
        }
```

Points to remember

- Called function must be declared for its type, appropriate to the data type it is expected to return.
- Structure variable used as actual argument and the corresponding formal argument in the called function must be of the same **"struct"** type
- Return statement is necessary only when the function is returning some data back to the calling function.
- When a function returns structure, it must be assigned to a structure of identical type in the calling function
- The Called functions must be declared in the calling function appropriately.

Example 1

```
struct student
{
        char *name;
        int age;
        float per;
};
void display(struct student s)
{
        printf("\nName :%s", s.name);
        printf("\nAge : %d", s.age);
        printf("\nPercent : %f", s.per);
}
```

```
int main()
{
struct student s={"Ram",25,7.5};
display(s);
return 0;
}
```

```
Name :Ram
Age : 25
Percent : 7.500000
```

After updating values

```
Name :Ram
Age : 25
Percent : 7.500000
Name :Pradeep
Age : 27
Percent : 8.500000
Name :Ram
Age : 25
Percent : 7.500000
```

Example 2

Example Program: Method of sending an entire structure as a parameter to a function

```
#include<stdio.h>
struct stores
{
        char name[20];
        float price;
        int quantity;
};
struct stores update(struct stores product, float p, int q);
float mul (struct stores stock);// Function declaration
```


```
main()
{
        float p_increment, value;
        int q_increment;
        struct stores item=("XYZ", 26.54, 17);
        printf("\n Input increment values :");
        printf("Price increment and quantity increment\n");
        scanf("%f %d", &p_increment, & q_increment);
        item=update(item, p_increment,q_increment);
        printf("updated values of item\n\n ") ;
        printf("name      : %s \n", item.name);
        printf("price     :%f \n", item.price);
        printf("quantity: %d \n",item.quantity);
        value=mul(item) ;// Function Call
        printf(" \n Value of the item       = %f \n",Value);
}
```

```
struct stores update(struct stores product, float p, int q)
{
        product.price += p;
        product.quantity += q;
        return(product);
}
```

Called function

```
float mul(struct stores stock)
{
        return(stock.price*stock.quantity);
}
```

```
 Input increment values :Price increment and quantity increment
10 12
updated values of item

name     : XYZ
price    :36.540001
quantity: 29
Value of the item= 1059.660034
```