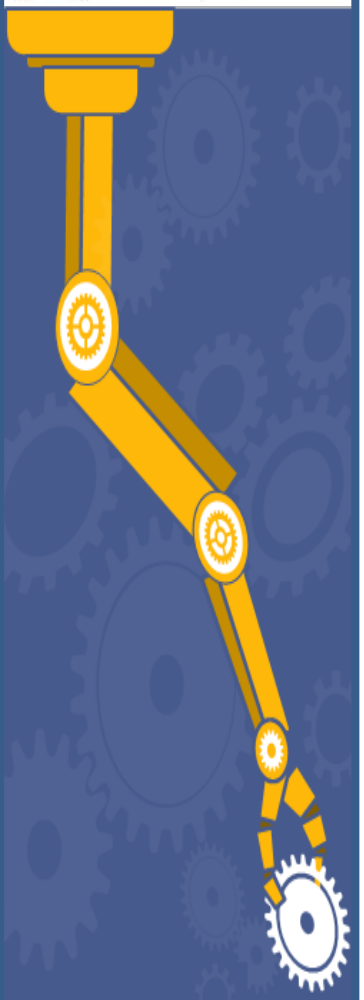
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



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

- **User Defined Data type**
- A convenient data type for handling a **group of logically related data items**
- **Need of Structure:**
  - Allows to join either **homogeneous or heterogeneous data types**, unlike arrays, which allow only homogenous.
  - The right one to model most of the real world collections, which are heterogeneous.
  - Close to real life representation
  - e.g. an employee structure
  - Commonly used to define records to be stored in files
  - Combined with pointers, can create linked lists, stacks, queues and trees.
  - Organizes complex data in a more meaningful manner

Structure

// General form of a Structure

```
   struct <struct-name>
{
        <data-type> element 1;
        <data-type> element 2;
        …
        <data-type> element n;
 };
```

; is a must

- A Structure Prototype does not reserve a space
- Space will be reserved only when **variables of structure type are defined**

Defining a Structure

Keyword- declares a structure that holds the details of data fields

```
struct book_bank
{
    char title[20];
    char author[15];
     int pages;
    float price;
};
```

Name of the structure/Structure Tag

Data Fields/ Structure elements/members

** Note: Above definition has not declared any variables- Like a template to represent information*

Defining a Structure

Keyword- declares a structure that holds the details of data fields

```
struct book_bank
{
    char title[20];
    char author[15];
     int pages;
    float price;
};   Struct book_bank, book1,book2,book3;
```

Name of the structure/Structure Tag

Data Fields/ Structure elements/members

- Above statement declares *book1,book2,book3 as variables of type Struct book_bank*
- *Each variables has 4 members*

Structure: Prototype, Definition & Use

Example: An Employee structure

```
struct Employee       // structure prototype or blue print
  {
        int eno;
        char ename[20];
        char gender[10];
        int  age;
        float salary;
  }; e1,e2,e3;

// Creation or defining structure variable struct Employee e1;
// Accessing elements of structure using the structure variables
```

e1.eno, e1.ename, e1.gender, e1.age & e1.salary

Example: Read information for one person and print the result

```
struct personal
    {
        int idno;
        char name[20];
        char gender[10];
        int  age;
        float salary;
    };
main()
{
     struct personal person;   // Variable declaration
     scanf("%d %s%s%d%f", &person.idno,
    person.name,person.gender, &person.age, &person.salary);
    printf("%d %s%s%d%f", person.idno, person.name,person.gender,
    person.age, person.salary);
}
```

Initializing Structures

```
struct Employee
{
        int eno;
        char ename[20];
        int  age;
};
```

'{' and '}' are must while initialization of structure

// Defining and initializing structure variable

```
struct Employee e1 = {123,"ram", 34};
```

- **Another way to declare and initialize structure variables:**

```
struct Employee
{
        int eno;
        char ename[20];
        int  age;
} e1={123,"laxman",34};
```

Rules to follow in Initializing Structures

- Cannot able to initialize individual members inside the structure template
- Order of values enclosed in braces must watch the order of members in the structure definition.
- Permitted to have a partial initialization. We can initialize only the first few members and leave the remaining. Uninitialized members should be only at the end of the list
- Uninitialized members will be assigned default values.
  - Zero for Integers
  - Zero for Floating Point
  - \0 for Characters and Strings

Copying and Comparing Structure Variables

- Two variables of same structure type can be copied as like ordinary variables.
- For example: employee1 and employee2 belong to the same structure, then following statements are valid.
  - employee1=employee2 & vice versa
- C does not permit **any logical operations** on structure variables like
  - employee1==employee2
  - employee!=employee2
- In order to compare, we can do by comparing the members individually

Example: Comparison of structure variables

```
struct class
    {
        int number;
        char name[20];
        float marks;
    };
main()
{
    int x;
    struct class student1={1,“ Arun”,95.50};
    struct class student2={2,“ Babu”,7750};
    struct class student3;
    student3=student2;          // Possible
    x=(student3.number==student1.number)?1:0;  // usage of logical operations
    if(x=1)
    {
        printf(“ student 1 and student 3 are same”);
    else
         printf(“ student 1 and student 3 are different”);
    }
}
```

Array of Structures

- For example: **Analyzing marks(various subjects) obtained by a class of students.**
- Declaring an array of structures- each element of the array representing a structure variable

struct student[70];

```
struct marks
{
        int sub1;
        int sub2;
        int sub3;
};
main()
{
        struct marks student[3]={{50,67,89}, {60,77,79},
        {70,97,59}};
}
```

Array of Structures

- An array of structures is stored inside the memory in the same was as a multi-dimensional array.

student[0].sub1	**50**
student[0].sub2	67
student[0].sub3	89
student[1].sub1	60
student[1].sub2	77
student[1].sub3	79
student[2].sub1	70
student[2].sub2	97
student[2].sub3	59

Example: Calculate subject wise and student wise totals

```
struct marks
    {
        int sub1;
        int sub2;
        int sum;
    };
int main()
{   int i;
    struct marks student[2]={{55,90,0}, {65,70,0}};
    struct marks total={0,0,0};
    for(i=0;i<=1;i++)
    {
        student[i].sum=student[i].sub1+student[i].sub2;
        total.sub1= total.sub1+student[i].sub1;
        total.sub2= total.sub2+student[i].sub2;
        //total.sum= total.sum+student[i].sum;
    }
```

Example: Calculate subject wise and student wise totals

```
    printf("Student \t TOTAL \n");
    for(i=0;i<=1;i++)
    {
          printf("student[%d]     %d \n", i+1,student[i].sum);
    }
    printf("Subject \t TOTAL \n");
    printf("Subject 1 =  %d\n",total.sub1);
    printf("Subject 2 =  %d\n",total.sub2);

    printf("\n Final Total= %d\n", total.sum);
    return 0;
}
```

```
Student          TOTAL
student[1]      145
student[2]      135
Subject          TOTAL
Subject 1 =  120
Subject 2 =  160

 Final Total= 280
```

Arrays within Structures

- C Permits use of arrays as structure members.

For Example:

```
struct marks
{
        int number;
        float subject[3];
} student[3];
```

Here subject contains three elements subject[0],subject[1],subject[2].

For example:

student[1].subject[2]; // refers to the marks obtained in the third subject by second student

Structures within Structures

- It means nesting of structures.

```
struct salary
{
        int eno;
        char ename[20];
        char department[10];
        int basic_pay;
        int  da;
        int hra;
        int ca;
} employee;
```

```
struct salary
{
int eno;
char ename[20];
char department[10];
        struct
        {
                int basic_pay;
                int  da;
                int hra;
                int ca;
        }allowance;
}employee;
```

employee.allowance.hra

Structures within Structures

- Inner structure can have more than one variable

```
struct salary
{
int eno;
char ename[20];
char department[10];
        struct
        {
                int basic_pay;
                int  da;
                int hra;
                int ca;
        }
        allowance, arrears;
}employee[50];
```

employee[1].allowance.hra
employee[1].arrears.hra

Structures within Structures

- Another way to define Inner structure

```
struct pay
{
                int basic_pay;
                int  da;
                int hra;
                int ca;
};
struct salary
{
        int eno;
        char ename[20];
        char department[10];
        struct pay allowance,
        struct pay arrears;
};
struct salary employee[50];
```

- Pay template is defined outside the salary template.
- Used to define the structure of *allowance* and *arrears* inside the salary structure

Note: C99 standard Allows 63 levels of nesting

Practice Questions

- Write C program to accept the details of employee and display them using structure. Details consist of Employee ID, Name, Designation, Department, Salary.
- Write C program to read the details of two workers and calculate total payment of workers using structure. Name, no of days, wage for each day should be included as structure members.
- Write a C program to accept details of 'n' employee (emp_no, emp_name, salary) and display the details of employee having highest salary. Use array of structure.
- Write a C program to read information of student such as Name, Roll number, Birthday, admission date. Calculate age of student at the time of admission. Name, roll number and date should be members of the structure. Day, month and year should be members of the structure date. (Use Structure inside another structure)