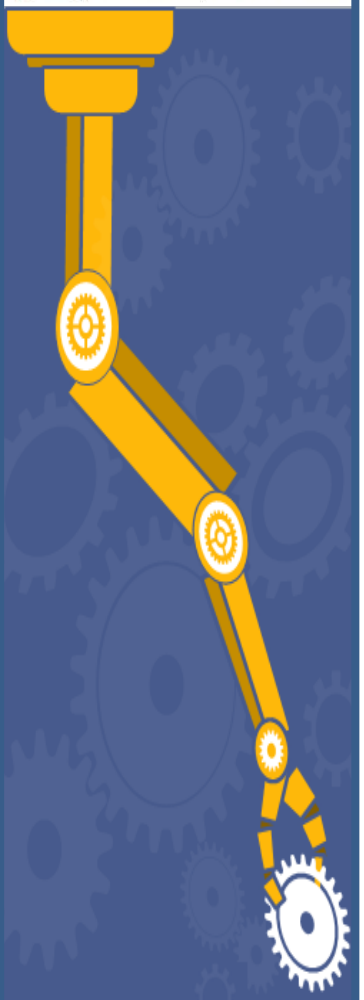
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



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**

Pointers to Structure

- Pointers and structure can be used together in a program.

Points to remember:

- *Name of an array stands for the address of the zeroth element.*
- *Same thing is true for the names of arrays of structure variables*
- Consider the following declaration:

```
struct shop
{
        char name[30];
        int number;
        float price;
} product[2], *ptr;
```

- *This statement declares product as an array of two elements each of the type struct shop and ptr as pointer to data objects of the type struct shop*
- ***ptr=product*** *–> would assign the address of the zeroth element of product to ptr*

Pointers to Structure

```
struct shop
{
        char name[30];
        int number;
        float price;
} product[2], *ptr;
```

To access the members of the structure- alternate way
*(*ptr).number*
**ptr.number // Illegal*

- The pointer ptr will now point to product[0].
- Its members can be used by the following notation.
- arrow operator **(->)** is used. Arrow operator is also called member selection operator.

ptr->name
ptr->number
ptr->price

Note: ptr→ is simply another way of writing product[0]

Pointers to Structure – by using malloc

- To create pointer variable

  struct **shop *ptr;**

  ptr=(shop*)malloc(30+size of(int)+sizeof(float))

It assigns the pointer to a block of memory that stores the structure members.

- It can also be written as,

  ptr = (shop *) malloc (sizeof(shop));

- It is possible to create a block of memory capable of storing a group of structures

  ptr = (shop *) malloc (10*sizeof(shop));

Example – Use of structure pointers

```
struct shop
{
        char *name[20];
        int number;
        float price;
};
main()
{
        struct shop product[3], *ptr;
        printf("INPUT\n\n");
        for(ptr=product; ptr<product+3;ptr++)
        scanf("%s%d%f",& ptr→name, &ptr→number,
        &ptr→price);
        printf("OUTPUT\n\n");
```


```
ptr=product;
        while(ptr<product+3)
        {
        printf("%s %d %f \n", ptr→name, ptr→
        number, ptr→ price);
        ptr++;
        }
}
```

Operator precedence in structure pointers

- Operators such as "->", ".", "()", "[]" has the highest priority among the operators.
- For Example:

```
struct
{
        int count;
        float *p; // pointer inside struct
} ptr; // struct type pointer
```

Consider the following Statements.

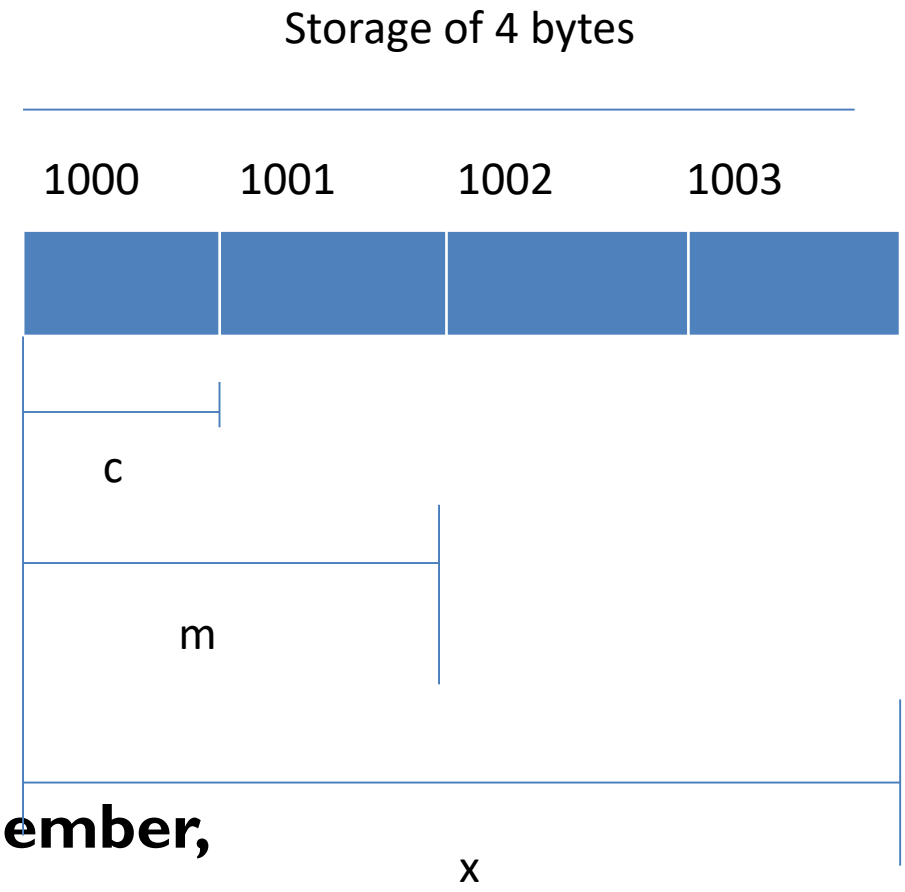
++ptr->count	It increments count not ptr
(++ptr)->count	Increments ptr first and links count
ptr++->count	Increments ptr after accessing count
***ptr->p**	Fetches whatever p points to
***ptr->p++**	Increments p after accessing whatever it points to
(*ptr->p)++	Increments whatever p points to
***ptr++->p**	Increments ptr after accessing whatever it points to

Unions in C

- Similar to Structures - It is a derived data-type to organize a group of related data items of different data types referring to a single entity.
- In structure, each member has its own storage location, whereas all the **members of the union share a same location.**
- The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.
- Because of this kind of storage, union it can handle only one member at a time

Unions in C

```
union item
{
        int m;
        float x;
        char c;
} code;
```



To access a union member,

code.m

code.x

Unions in C

For example,

```
code.m=200;

code.x=22.33;

printf("%d", code.m); // error

printf("%f", code.x); // valid
```

Note: you can access the value that is recently stored. i.e. the new value will supersedes the previous value.

Union can be used in all the places where structures can be used.

Unions in C

For example,

```
code.m=200;

code.x=22.33;

printf("%d", code.m); // error

printf("%f", code.x); // valid
```

Note: you can access the value that is recently stored. i.e. the new value will supersedes the previous value.

Union can be used in all the places where structures can be used.

Unions in C

- Union may be initialized when the variable is declared.
- It can be initialized only with a value of the same type.
- For example,

```
union item abc = {100}; //valid

union item abc = {10.20}; //invalid
```

- Type of the first member is **int.**
- Other members can be initialized by either assigning values or reading from keyboard

Example

```
#include <stdio.h>
union sample
{
int regno;
float mark;
}s1;
int main()
{
        printf("Enter the register number: ");
        scanf("%d",&s1.regno);
        printf("\nEntered register number: %d",s1.regno);
        s1.mark=100;
        printf("\nEntered register number: %d",s1.regno);
        printf("\n Mark: %f",s1.mark);
        return 0;
}
```

```
Enter the register number: 50

Entered register number: 50
Entered register number: 1120403456
 Mark: 100.000000
```

Demonstration of Advantages of using Union

A shop sells two kinds of items

1. Books – Title, author, price etc
2. Shirts - color, size, price, etc

```
struct shop
{
        char *title;
        char *author;
        int color;
        int size;
        double price;
};
```

Price is common property and others are individual

Accessing the members of the structures

```
struct shop
{
        char *title;
        char *author;
        int color;
        int size;
        double price;
};
```

```
int main()
{
        struct shop book;
        book.title=" CP";
        book.author=" Dennis";
        book.price=100;
        return 0;
}
```

Size of shop: 8+8+4+4+8= 32 bytes

Book Variable doesn't possess these properties- wastage of memory

What Next?? – we can save space by using unions

Demonstration of Advantages of using Union

```
struct shop
{
double price; //8
union
{
        struct
        {
        char *title;   //8
        char *author;//8
        } book;
        struct
        {
        int color;  //4
        int size;//4
        } shirt;
} item;
};
```

```
int main()
{
        struct shop s;
        s.item.book.title= " CP";
        s.item.book.author=" Dennis";
        return 0;
}
```

Size: 16

Total space Required for this : MAX(16,8)+ 8= 24