Data Structure Using C

Ch. 02

Advanced Concept of C and Introduction to Data Structure

Introduction to Data Structure

- Data
  - Data is a collection of information.
- Data Structure
  - A data structure is a way of organizing data items that considers not only the items stored, but also their relationship to each other.

Data Types :

- To store data in computer, variable is required.
- To define variable we must specify the data type first.
- C programming language has basic three data types available.
  - Fundamental Data Type (Primary Data Type)
  - Derived Data Type
  - User Define Data Type

Fundamental Data Type : (Primary Data Type)

- Primary data type can be classify into two parts.
  - Integral
  - Floating Point Type
- Integral Type :
  - Signed & Unsigned
  - Int
  - Short Int
  - Long Int
  - Character

Signed & Unsigned :

- Signed Type
  - When we uses the storage capacity of any variable with positive to negative value then this types of data will be signed data.
  - Signed type of data can store positive and negative values. It is default type.
- Unsigned :
  - When we uses the storage capacity of any variable with **ONLY** positive value then this types of data will be **Unsigned data.** This will increase storage range from to negative to positive.
  - Unsigned type of data can store **ONLY** positive values. We must add **unsigned** keyword before data type when declaring the data.

Integral Type :

- Short Integer :
  - Short integer can hold 1 Byte (8 Bits) in memory.
  - A short integer has two parts:
    - Signed short integer
      - Signed short integer can Store integer value in the range of **-128 to 127.**
    - Unsigned short integer.
      - Unsigned short integer can Store integer value in the range of **0 to 255.**
      - We can use format code as **%hd** for short integer.

Integral Type :

- Integer :
  - This is most commonly widely used data type in C language.
  - Any type of integer can hold 2 bytes (16Bits) of memory.
  - It has also two categories :
    - Signed integer
      - Signed integer can store -32768 to 32767
    - Unsigned integer
      - Unsigned integer can store 0 to 65535

Integral Type :

- Long Integer :
  - Long integer can hold 4 bytes (32Bits) of memory.
  - It has also two categories :
    - Signed long integer
      - Signed long integer can store **-2147483648** to **2147473647**
    - Unsigned long integer
      - Unsigned long integer can store **0** to **4294967295**

Integral Type :

- Character :
  - A character can hold 1 byte (8Bits) of memory.
  - To store any character string we have to use character array.
  - It has also two categories :
    - Signed character
      - Signed character can store **-128** to **127**
    - Unsigned character
      - Unsigned character can store **0** to **255**

Floating point :

- A floating point data type is also known as a Real Data Type.
- It can store the value which has fraction parts.
- Floating point is further classified in to three parts as per storage capacity.
  - Float
    - Float can hold 4 bytes (32Bits) of memory with 6 decimal point. Range **3.4E-38** to **3.4E+38**
  - Double
    - Double can hold 8 bytes (64Bits) of memory with 6 decimal point. Range **1.7E-308** to **1.7E+308**

Floating point :

- Long Double
  - Long double can hold 10 bytes (80Bits) of memory. Range **3.4E-4932** to **1.1E+4932**
- Derived Data Type :
  - Derived data types are those which are provided by 'C' language to us.
  - We are using **Array** and **Pointer** which are derived data type of C language.

Floating point :

- User Define Data Type :
  - C language also provides facility to create our own new data type using typedef.

    Syntax :

    typedef <languagedatatype> <newDataType>

    Example : typedef int Num;

    We can use it as

    Num a, b, sum;

Array

- An array is a collection of **similar elements** of the **same data type**.
- These similar elements could be all integers, or all floats, or all characters, etc.
- Individual values of array are called as elements.
- Array can be initialized at a place where it is declared.
- Arrays are helpful to store and access a list of values under a single variable name.

Array

- Direct access to any element value of an array is possible by an integer valued called the subscript.
- The subscript value specifies the relative position of the data values within that array.
- Each array elements is referred to by specifying the array name followed by one or more subscripts.
- Where each subscript enclosed in square brackets in array. Each subscript must be expressed as a non-negative integer value.

Benefits of Array:

- If you want to specify more then one variable then you have to specify as follow.

  int sub1, sub2, sub3, sub4, sub5;

- In place of normal variable declaration we can declare an array variable as given below.
- It means that array will be helpful to make you work easy to process.

  int sub[5];

Types of Array :

(1) One-Dimensional Array or

Single Dimensional Array

(2) Two-Dimensional Array

(3) Multi-Dimensional Array

(4) String Array

Types of Array :

- **One-Dimensional Array**
  - A list of items can be given a variable name using only one subscript and such a variable is called a ***single dimensional array***.
- **Declaration of Array:**
  - Like any other variables, arrays must be declared before they are used.
  - Array declaration specifies the data type of array elements like,

    int, float etc. and the size of the array.

Types of Array :

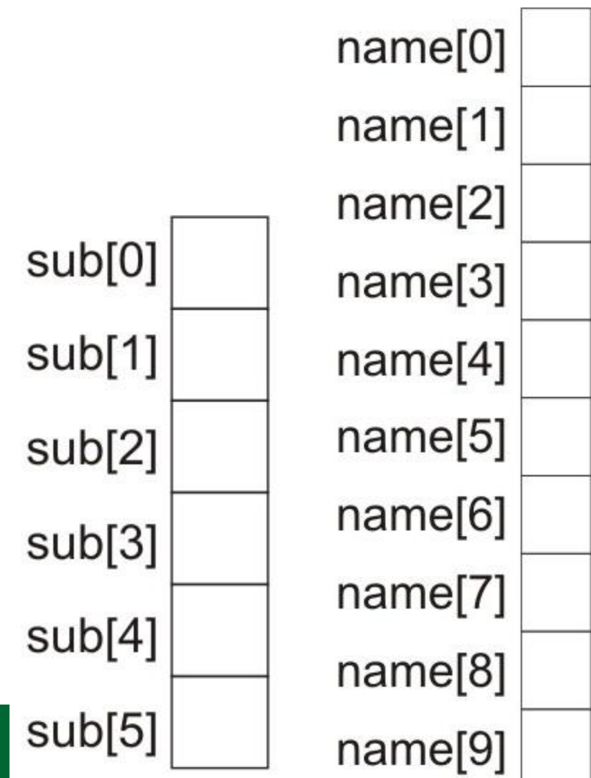
- **Syntax :**

  DataType VariableName[size];

- **Purpose :**
  - To declare single dimensional array we can use above syntax.
- Example :

  int sub[6];

  char name[10];



Def. Write a program to store 5 values in an array and print it.

```
int val[5];
val[0]=10;
val[1]=20;
val[2]=30;
val[3]=40;
val[4]=50;
printf("\n%d",val[0]);
printf("\n%d",val[1]);
printf("\n%d",val[2]);
printf("\n%d",val[3]);
printf("\n%d",val[4]);
```

Output :

10

20

30

40

50

Def. Write a program to store 10 values in an array and print it.

```
int val[10]={10,20,30,40,50,60,70,80,90,100};
printf("\n%d",val[0]);
printf("\n%d",val[1]);
printf("\n%d",val[2]);
printf("\n%d",val[3]);
printf("\n%d",val[4]);
printf("\n%d",val[5]);
printf("\n%d",val[6]);
printf("\n%d",val[7]);
printf("\n%d",val[8]);
printf("\n%d",val[9]);
```

Output :

```
10
20
30
40
50
60
70
80
90
100
```

Def. Write a program to store 10 values in an array and print it using loop

```
int val[10]={1,2,3,4,5,6,7,8,9,10};
int a;
  for(a=0;a<10;a++)
  {   printf("\nValue :",val[a]);
  }
```

Def. Write a program to enter 10 values in an array and print odd and even numbers

```
int val[10],a;
for(a=0;a<10;a++)
{     printf("\nEnter a Value :");
      scanf("%d",&val[a]);
}
printf("\nOdd Numbers :");
for(a=0;a<10;a++)
{     if(val[a]%2!=0)
            printf("\n%d",val[a]);
}
printf("\nEven Numbers :");
for(a=0;a<10;a++)
{     if(val[a]%2==0)
            printf("\n%d",val[a]);
}
```

Odd Numbers :

1

3

5

7

9

Even Numbers:

2

4

6

8

10

Def. Write a program to store a name and print it as given...

```
char name1[20]={"MONARCH"};
int a;
for(a=0;a<=20;a++)
    printf("\n%c",name1[a]);
```

Output :

M
O
N
A
R
C
H

Def. Write a program to enter 5 values and print **min** and **max** number for them.

```
int val[5],max,min,a,b;
for(a=0;a<5;a++)
{     printf("\nEnter Value : ");
      scanf("%d",&val[a]);
}
max=val[0];
min=val[0];
for(b=0;b<5;b++)
{     if(max<val[b])
              max=val[b];
      if(min>val[b])
              min=val[b];
}
printf("\nMax Num : %d",max);
printf("\nMin Num : %d",min);
```

: Output :

Max Num **: 59**

Min Num **: -1**

Types of Array :

- **Two-Dimensional Array**
  - A list of items can be store in table format by **row** and **column** is known as Two-Dimensional Array.

[0][0] 10	[0][1] 15	[0][2] 20	[0][3] 25
[1][0] 30	[1][1] 35	[1][2] 40	[1][3] 45
[2][0] 50	[2][1] 55	[2][2] 60	[2][3] 65

Now [0][1]=15 Now [2][3]=?

Types of Array :

- **Syntax :**

  DataType VariableName[rows][columns];

- **Purpose :**
  - To declare tow dimensional array we can use above syntax.

- Example :

  int std[3][4]; or

  int std[3][4]={10,15,20,25,30,35,40,45,50,55,60,65};

[0][0] 10	[0][1] 15	[0][2] 20	[0][3] 25
[1][0] 30	[1][1] 35	[1][2] 40	[1][3] 45
[2][0] 50	[2][1] 55	[2][2] 60	[2][3] 65

Def. Write a program to store 12 values using **two dimensional** array and print it using nested loop

```
int std[3][4]={10,15,20,25,30,35,40,45,50,55,60,65};
int x,y;
for(x=0;x<3;x++)
{
   for(y=0;y<4;y++)
   printf("\t[%d][%d]=%d",x,y,std[x][y]);
   printf("\n\n");
}
```

Types of Array :

- **Multi-Dimensional Array**
  - A list of items can be store in table format by **row** and **column** is with more then two dimension is known as **Multi-Dimensional Array.**

[0][0][0] 10	[0][0][1] 15
[0][1][0] 20	[0][1][1] 25
[1][0][0] 30	[1][0][1] 35
[1][1][0] 40	[1][1][1] 45

Now [0][1][0]=20
Now [1][1][0]=?

Types of Array :

- **Syntax :**

DataType VariableName[rows][col1][Col2]...;

- **Purpose :**
  - To declare multi dimensional array we can use above syntax.
- Example :

int std[2][2][2]; or

int std[2][2][2]={10,15,20,25,30,35,40,45};

Array Element memory Allocation

Data Type	Memory occupied	Example		Total Size (in Bytes)
int	2 bytes	int arr[10];	(10*2)	20 bytes
float	4 bytes	float a[5];	(5*4)	20 bytes
char	1 bytes	char ch[10];	(10*1)	10 bytes
double	8 bytes	double a[5];	(5 * 8)	40 bytes
long double	10 bytes	long double a[5];	(5*10)	50 bytes

Def. WAP to accept 10 numbers from user in an array and sort it in descending order as well display it.

```
void main()
{ int arr[10],j,k,temp;
  for(j=0;j<10;j++)
  {   printf("\nEnter Value %d :",j+1);
      scanf("%d",&arr[j]); }
```

```
//Sort Descending :
for(j=0;j<10;j++)
{   for(k=j;k<10;k++)
    {       if(arr[j]<arr[k+1])
            {       temp=arr[j];
                    arr[j]=arr[k+1];
                    arr[k+1]=temp;
            }
    }
}
for(j=0;j<10;j++)
    printf("%d\n",arr[j]);
getch();
}
```

Def. Write a program to find maximum element from array.

```
int num[10],a,max=0,n;
printf("\n How many numbers  you want to enter: ");
scanf("%d",&n);
printf("\n Enter Elements: \n");
for(a=0;a<n;a++)
{           scanf("%d",&num[a]);
    if(num[a]>max)
    {
            max=num[a];
    }
}
printf("\nMaximum : %d",max);
```

Pointers :

- In **C** programming, a pointer is a most important topic.
- It can hold the memory address of another variable.
- We know that memory addresses are the locations in the computer memory where data are stored. So we can say that we are using pointer to access and manipulate data stored in the memory.

Pointer :

- Pointers are very useful in many applications.
  - Pointers are used to create dynamic data structures; it built up from blocks of memory allocated from the heap at run-time.
  - A pointer can hold the address of any valid data item, including an array, a singular variable, a structure and a union.
  - A pointer can hold the address of a function.
  - Pointer also permits references to other function to be specified as arguments to a given function. This has the effect of passing function as arguments to the given functions.

Pointer :

- Pointers are also strongly associated with array and therefore provide an alternative way to access individual array elements.
- With the use of pointer, we can also return multiple data from a function via arguments.
- A pointer cannot hold the address of a constant, with one possible exception: A string constant has an address, which can be stored in a pointer variable indirectly.

Pointer Basics :

- To understand pointer, we have some knowledge about memory which is given below.
  - Computer's memory is made up of a sequential collection of storage cells called bytes.
  - Each byte has a number called an address associated with it.
  - In general, the addresses are numbered one by one (સતત), starting from zero. Here, the last address depends on the memory size. The last address as 65535

Pointer Basics :

- When we declare a variable in our program, the compiler immediately assigns a specific block of memory to store the value of the variable.
- Every memory cell has a unique address, this block of memory will have a unique starting address.

Accessing the Address of Variable :

- As we know that in computer, the actual location of a variable in the memory is system dependent.
- That's why the address of a variable is not known to us immediately.
- With the use of **&** operator we can find the address of any ordinary variable.
- For Example to Find Address

```
int a;
printf("%u",&a);
```

Def. WAP to define 5 variables and display it's memory address using "&" operator.

```
int a,b,c,d,e;
printf("\nAddress of A = %u",&a);
printf("\nAddress of B = %u",&b);
printf("\nAddress of C = %u",&c);
printf("\nAddress of D = %u",&d);
printf("\nAddress of E = %u",&e);
```

How to define and use POINTER variable.

- We can define pointer variable as same as regular variable.

  Syntax :

  DataType *<pointerVariable>;

- We can not store any value in pointer variable.
- In pointer we can store only address of another variable.
- Example :

```
int *p;
```

Pointer Initializing …

- **Syntax**
- int a=10,*p;
  - Displays the declaration of pointer as *p.
- p=&a;
  - Displays the storing address of variable a in pointer p;
- *p
  - Here is value of a return from pointer *p.

Def. WAP to define a pointer variable and display its use with normal variable.

```
int a=10,*p;

p=&a;

printf("\n%d",*p);

*p=*p+10;

printf("\n%d",*p);

printf("\n%d",a);
```

Advantages and Disadvantages of pointers.

- Pointer are used to create dynamic data structures.
- It is uses heap memory area to allocate memory blocks at run time.
- A pointer can hold the address of any valid data item, array, single variable, a function, a structure and a union.
- Pointer is also used to transfer value to function as argument with multiple return data values.
- Pointer are also strongly associated with arrays and therefore provide an alternative way to access individual array elements.

Multiple Pointers for Same Variable

- **We can define more then one pointer for a same variable.**

```
int a=10,*x,*y,*z;
x=&a;
y=&a;
z=&a;
printf("\n*x=%d",*x);
printf("\n*y=%d",*y);
printf("\n*z=%d",*z);
```

Pointer Arithmetic…

- It is also known as **pointer expressions** or **operations on pointer**
- Pointer variables can also used in arithmetic expressions addition, subscriptions, multiplications, division.
- Example : int a=10, *p;

  ```
  p=&a;
  *p=*p+2;
  ```
- The new value of pointer variable is 12

Array of pointers...

- As we know that we are using arrays of integer, float and character same like as we can define an array of pointer also.
- As we know that pointer contains an address , an array of pointers would be a collection of addresses.
- Syntax:
  - <data-type> *array[indexer];
- In this syntax the data type refers the data type of that dimensional array.

Array of pointers…

- For example that how we can defined the array of pointer.
  - Int *arr[10];

Passing Parameters to the Function…

- The technique which used to pass data from one function to another function is known as parameter passing.
- The two ways for passing parameters are:
  - 1. Pass by value (Call by value)
  - 2. Pass by pointer (Call by Reference)

Passing Parameters to the Function...

1. Pass by value (Call by value)

- In pass by value, the values of the variables are copied in the parameters to the function.
- The called function works on the copy and not on the original values of the parameters.
- This means that the original data in the calling function can not be changed.

Passing Parameters to the Function...

1. Pass by Pointer (Call by Reference)

- The memory addresses of the variables are passed as the function argument.
- In this case, the called function directly works on the data in the calling function.
- Call by Reference method is often used when manipulating arrays and strings.
- This also required when multiple values are return by the function.

Relation between pointers and arrays.

- One dimensional array can also be used with pointer.
- In C programming, there is a strong relationship between pointers and arrays.
- Any operation can be achieved by array done with pointers.
  - Int a[10],*pa;
  - Pa=&a[0];
- Here is we defined array and a pointer.

Scope rules and storage classes…

- The variables can be characterized or classified by their storage classes.
- Data types refers of value represented by a variable whereas storage class refers to the visibility and the scope of the variable within the program.
- A variable can in C can have any one of the four classes:

Scope rules and storage classes…

- **Automatic variables:**
  - Any variable which is declared local to a function or declared inside the function in which they used is known as automatic variable.
  - The scope of the variable done within the function itself.
  - The automatic variable defined in different functions, even if they have same name through they are treated as different.

Scope rules and storage classes…

- We may also use the keyword auto to declare automatic variable.
- The default value stored in this variable is a garbage value.
- So when we not initialized any value on that variable, it take garbage value.
- They are created when functions are called and destroyed automatically when the function exited.
- It also known as a private variable or local variable or internal variable.

- **External variables:**
  - Variable that are both alive and active throughout the entire program are known as external variables.
  - They are also known as global variable.
  - This type of variables are declared outside the main() function and UDF function and they are available to all functions to use them.
  - Once a Global variable is declared all the function can use it and any function can change its value.

Scope rules and storage classes…

- The keyword extern can be used for explicit declarations of external variables.

Scope rules and storage classes…

- **Static variables:**
  - As the name suggest, the value of static variable remain static till the end of the program.
  - A variable can be declared static using the keyword ‘static’.
  - These variables are local to the block to which they are declared.
  - The feature of static variable is, when the function is not active at that time the value of variable is not changed.

Scope rules and storage classes…

- The keyword extern can be used for explicit declarations of external variables.
- By default these variable are initialized to zero.
- This variable is stored in memory.

Scope rules and storage classes…

- **Register variables:**
  - We can tell the compiler that a variable should be kept in one of the machine's registers, instead of keeping in the memory where all the variables are stored.
  - When variables are required very often during the program, they may be declared as register variable.
  - Register variable can be defined by using 'register' keyword.

Scope rules and storage classes…

- It is a special storage areas within a computer's CPU.
- All the arithmetic and logical operations are carried out with this registers.
- Many compilers allow only integer and character variable to keep in register but ANSI should not restrict any.
- Since a register access is much faster then a memory access, keeping the frequency accessed variable in the register will lend to faster execution of program.

Scope rules and storage classes…

- Generally, the register variables are used as loop counters which are used multiple times in the program.
- We must be careful when we are using register variable because they are very few CPU registers and many of the might been busy in other processing.
- If the registers are not available, the variable is treated like the automatic variable.

Dynamic Allocation and De-Allocation of Memory…

- In C language, variables are declare and memory occupied by program.
- There are mainly two types of memory allocation:
- A variable which declared in block with fixed in size and when program will be executed at that time space is allocated which never free is called as Static Allocation.

Dynamic Allocation and De-Allocation of Memory…

- A variable which declared in block and when the block will be ended, the memory will freed is called as Automatic Allocation.

Dynamic Allocation and De-Allocation of Memory…

- **Dynamic Memory Allocation:**
- It means that we create variable or occupy memory allocation of variable at run time.
- That called as Dynamic Memory Allocation.
- The advantages of are given below:
  - 1. When programmer actually needs to use memory space at that time dynamically allocates. So it save memory space.

Dynamic Allocation and De-Allocation of Memory…

2. Programmer not need to know in advance how many variables we needed.

- For example, assume that programmer wants to store BCA student club member name in array but we don’t know how many students will join the club.
- So we just make an array enough large that every student name can store in array but that is a huge waste of memory.

Dynamic Allocation and De-Allocation of Memory…

- A better way is Dynamic Memory Allocation(DMA) to store large amount of data.
- So in other words we can say that, allocating memory at run time is known as dynamic memory allocation.
- The memory allocation functions are:
  - Malloc
  - Calloc
  - free

Dynamic Allocation and De-Allocation of Memory…

- **malloc() function**
  - With the use of malloc() function, we can allocate or reserves a specified amount (size) of memory and returns a pointer variable which is void type.
  - It means that user can assign it to any type of pointer.

Dynamic Allocation and De-Allocation of Memory…

P=(<data type> *) malloc (size * size of (<data type>));

- In this above syntax p is a pointer variable of particular data type.
- The malloc() function returns a pointer of any data type to an area of memory with specified size.
- Example:

  Int *inptr;

  Inptr=(int *) malloc (100 * size of (int));

Dynamic Allocation and De-Allocation of Memory…

calloc() function

- This function is also used to dynamically allocate memory space to derived data type such as arrays and structures.
- As we know that malloc() function is used to allocates a single block of storage space and does not initialize memory to zero but it has garbage value.

Dynamic Allocation and De-Allocation of Memory…

- Whereas calloc() is used to allocates multiple block of storage with same size and sets all bytes to zero.
- Syntax
  - P=(structure *) calloc (num,element_size);
- The first argument num is the number of variables(item) we want to save in the allocated memory space.

Dynamic Allocation and De-Allocation of Memory…

- The second argument element_size is the size of each variable means that number of bytes that each variable(item) takes.
- This function also returns a void pointer.
- If allocation of memory space is successful, all the allocated memory is cleaned (set to 0), and the function returns a pointer to the first byte.

Dynamic Allocation and De-Allocation of Memory…

- If the calloc() function fails to allocate a memory space, it returns a null pointer.
- Example
  - Int *ip;
  - Ip=(int *) calloc(100, sizeof(int));

Dynamic Allocation and De-Allocation of Memory…

- free() function
- The malloc() and calloc() function is used to dynamically allocate memory space to the free memory area which is known as heap area.
- This heap area is finite.
- So when our program finishes using particular block of dynamically allocated memory it is good to de-allocate or free memory space for the future use and also it will be free up resources and improve performance(speed).

Dynamic Allocation and De-Allocation of Memory…

- Syntax:
  - free(ptr_var);
- The free() function de-allocate(release) the memory block which is pointed by ptr_var.
- This memory must have been allocated with the use of malloc(), calloc(), or realloc().
- If ptr_var is NULL then free() does nothing.

Dynamic Allocation and De-Allocation of Memory…

- If we are using invalid pointer in the call it may create problems and cause system crash.
- Example :
  - free(ptr);

Dangling Pointer Problem…

- As we know that pointer is very useful in many application.
- The main functionality of pointer is to pointing to the address of any variable.
- If any pointer is pointing the memory address of any variable, however, after some variable has delete from that memory location through pointer is still pointer such memory location.
- Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

Structures…

Introduction :

- As we know that an array is group of similar data items that share a common name and its data type also same.
- A structure is a collection of one or more variables data types grouped under a single name for easy manipulation.
- In array, we can store more than one element under the same name whereas in structure we can store more than one different data type name under the same name.

Structures…

- A structure can contain any of C’s data types, including arrays and other structures.
- Each variable within a structure is called a member of the structure.
- Syntax

```
struct <struct-name>
{
  Structure_member1;
  Structure_member1;
}instance;
```

Structures...

- Example
  - structre student
  - {
    - int rno;
    - char name[10];
    - float per;
  - }s1,s2;

Use of structure member operator (.)

- We can also access the structure member variable same like as ordinary variable.
- Individual structure members can be used like other variables of the same type.
- Structure members are accessed using the structure member operator (.), also called the dot operator, between the structure variable name and the member name.

Structures...

- Syntax:
  - Var-name.member-name;
- Example
  - e1.rno=10;
  - e1.name="Angel"
  - e1.per=99.99;
  - Printf("Roll name is:%d",e1.rno);

Arrays of Structures

- As we know that we can have structures that contain arrays, we also have arrays of structures.
- For example:

```
structure  phone_book
{
  char fname[10];
  char lname[10];
  Float phn_no;
}pb[100];
```

Pointer of Structures

- We can also address or point the structure variable with structure pointer itself.
- Also we can access the numbers of structure variable with the pointer variable.

Enumerated Constants…

- An enumeration is a one type of constant data type which consists of a set of named identifier (values) that represent integer constants.
- An enumeration is also referred to as an enumerated type because you must list (enumerate) each of the values in creating a name for each of them.
- Enumerated constants data type uses to declare symbolic integer constants.
- We can say that in C enumeration constraints have type int.

Enumerated Constants…

- Its integer identifier also known as “enumeration set”, “enumerator constants”, “enumerators” or “members”.
- With the use of enum keyword, we can create a new “type” and specify the values it may have.
- Enumeration provide an alternative to the #define preprocessor diretective.
- The purpose of enumerated type is to enhance the readability of a program.

Enumerated Constants…

- Syntax

```
enum <enumeration-name>
{
    Enumeration-list;
}
```

- Example

```
enum DAY
{
    Sunday,
    Monday
    Etc
}
```

Union...

- Same as struct keyword unions can also with keyword.
- The general form of union is as below:
  - union <union-name>
  - {
    - Union_member1
    - Union_member2
  - }instance;

Union...

- In above syntax, union is the keyword; tagname is the valid identifier name.
- In between curly braces, we have to define numbers (any valid data types) of union.
- For example,

```
union sample
{
    int a;
    float b;
    char c;
} xyz;
```

Union…

- This declares a variable code of type union sample.
- The main difference between union and structure is of storage of memory.
- The compiler allocates a memory that is large enough to hold the largest variable type in the union.
- To access a union member we can use the same syntax that we use for structure members. That is, xyz.a , xyz.b ,xyz.c are all valid member variables.

Union...

- During accessing we should make sure that we are accessing the member whose value is currently stored.
- For example the statement such as
  - xyz.a = 379;
  - xyz.b = 7859.36;
  - printf(“%d”,xyz.a);
- would produce erroneous output.
- In effect, a union creates a storage location that can be used by any one of its members at a time.

Union…

- When a different member is assigned a new Value, the new value supersedes the previous Member’s value.

Union...

<table>
<tr><th>Struct</th><th>Union</th></tr>
<tr><td>In structure each member has its own location</td><td>In union, all the member uses the same memory location and the size of location is depending on the data type which can hold more bytes.</td></tr>
<tr><td>If structure is following:
structure student
{
int no,tot;
char name;
float prec;
}
sizeof() will return 9.</td><td>If union is following:
union student
{
int no,tot;
char name;
float prec;
}
sizeof() will return 4.</td></tr>
</table>