

**UNIT-II:**

**Arrays:** one and two dimensional arrays, creating, accessing and manipulating elements of arrays

**Strings:** Introduction to strings, handling strings as array of characters, basic string functions available in C (strlen, strcat, strcpy, strstr etc.), arrays of strings

**Structures:** Defining structures, initializing structures, unions, Array of structures.

**Pointers:** Idea of pointers, Defining pointers, Pointers to Arrays and Structures, Use of pointers in self-referential structures, usage of self referential structures in linked list (no impl)  
Enumeration data type

**1. What is an array? How to declare and initialize arrays? Explain with examples**

**Ans:**

**Array:-**An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

**Declaration of an array:-** We know that all the variables are declared before they are used in the program. Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified. The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

**Syntax:-**        data\_type array\_name[n];

where, n is the number of data items (or) index(or) dimension. 0 to (n-1) is the range of array.

**Ex:**     int a[5];

float x[10];

**Initialization of Arrays:-**

The different types of initializing arrays:

1. At Compile time
  - i. Initializing all specified memory locations.
  - ii. Partial array initialization
  - iii. Initialization without size.
  - iv. String initialization.

2. At Run Time

**1. Compile Time Initialization**

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

The general form of initialization of arrays is

**type array-name[size]={ list of values};**

**i. Initializing all specified memory locations:-** Arrays can be initialized at the time of declaration when their initial values are known in advance. Array elements can be initialized with data items of type int, char etc.

**Ex:-** `int a[5]={10,15,1,3,20};`

During compilation, 5 contiguous memory locations are reserved by the compiler for the variable **a** and all these locations are initialized as shown in figure.

a[0]	a[1]	a[2]	a[3]	a[4]
10	15	1	3	20
1000	1002	1004	1006	1008

Fig: Initialization of int Arrays

**Ex:-** `int a[3]={9,2,4,5,6};` //error: no. of initial vales are more than the size of array.

**ii. Partial array initialization:-** Partial array initialization is possible in c language. If the number of values to be initialized is less than the size of the array, then the elements will be initialized to zero automatically.

**Ex:-** `int a[5]={10,15};`

Even though compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

a[0]	a[1]	a[2]	a[3]	a[4]
10	15	0	0	0
1000	1002	1004	1006	1008

Fig: Partial Array Initialization

**Initialization with all zeros:-**

**Ex:-** `int a[5]={0};`

a[0]	a[1]	a[2]	a[3]	a[4]
0	0	0	0	0
1000	1002	1004	1006	1008

**(i) Initialization without size:-** Consider the declaration along with the initialization.

**Ex:-** `char b[]={'C','O','M','P','U','T','E','R'};`

In this declaration, eventhough we have not specified exact number of elements to be used in array b, the array size will be set of the total number of initial values specified. So, the array size will be set to 8 automatically. The array b is initialized as shown in figure.

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
C	O	M	P	U	T	E	R
1000	1001	1002	1003	1004	1005	1006	1007

Fig: Initialization without size Ex:- `int ch[]={1,0,3,5}` // array size is 4

(ii) **Array initialization with a string:** -Consider the declaration with string initialization.

**Ex:-** char b[]="COMPUTER";

The array b is initialized as shown in figure.

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]
C	O	M	P	U	T	E	R	\0
1000	1001	1002	1003	1004	1005	1006	1007	1008

Fig: Array Initialized with a String

Eventhough the string "COMPUTER" contains 8 characters, because it is a string, it always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

**Ex:-**

char b[9]="COMPUTER"; // correct

char b[8]="COMPUTER"; // wrong

## 2. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:- scanf can be used to initialize an array. int x[3]; scanf("%d%d%d",&x[0],&x[1],&x[2]);

The above statements will initialize array elements with the values entered through the key board.

(Or)

for(i=0;i<100;i=i+1)

```
{
    if(i<50)
        sum[i]=0.0;
    else
        sum[i]=1.0;
}
```

The first 50 elements of the array sum are initialized to 0 while the remaining 50 are initialized to at run time.

---

**2. How can we declare and initialize 2D arrays? Explain with examples.****Ans:**

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns. These are well suited to handle a table of data. In 2-D array we can declare an array as :

**Declaration:-****Syntax:**      `data_type    array_name[row_size][column_size];`**Ex:-** `int arr[3][3];`

where first index value shows the number of the rows and second index value shows the number of the columns in the array.

**Initializing two-dimensional arrays:**

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

**Ex:** `int a[2][3]={0,0,0,1,1,1};` initializes the elements of the first row to zero and the second row to one. The initialization is done row by row.

The above statement can also be written as `int a[2][3] = { { 0,0,0},{1,1,1} };`

by surrounding the elements of each row by braces.

We can also initialize a two-dimensional array in the form of a matrix as shown below

```
int a[2][3]={ { 0,0,0},{1,1,1} };
```

When the array is completely initialized with all values, explicitly we need not specify the size of the first dimension.

**Ex:** `int a[][3]={ {0,2,3},{2,1,2} };`

If the values are missing in an initializer, they are automatically set to zero.

**Ex:** `int a[2][3]={ {1,1},{2} };`

Will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

### 3. Explain how two dimensional arrays can be used to represent matrices. (or)

**Define an array and how the memory is allocated for a 2D array?**

**Ans:** These are stored in the memory as given below.

- i. **Row-Major order Implementation**
- ii. **Column-Major order Implementation**

**In Row-Major Implementation** of the arrays, the arrays are stored in the memory in terms of the row design, i.e. first the first row of the array is stored in the memory then second and so on. Suppose we have an array named **arr** having 3 rows and 3 columns then it can be stored in the memory in the following manner:

int arr[3][3];

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Thus an array of 3\*3 can be declared as follows:

```
arr[3][3] = { 1, 2, 3,
              4, 5, 6,
              7, 8, 9 };
```

and it will be represented in the memory with row major implementation as follows :

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

**In Column-Major Implementation** of the arrays, the arrays are stored in the memory in the term of the column design, i.e. the first column of the array is stored in the memory then the second and so on. By taking above eg. we can show it as follows :

```
arr[3][3] = { 1, 2, 3,
              4, 5, 6,
              7, 8, 9 };
```

and it will be represented in the memory with column major implementation as follows :

1	4	7	2	5	8	3	6	9
---	---	---	---	---	---	---	---	---

## Two-dimensional arrays of variable length

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns,. These are well suited to handle the table of data. In 2-D array we can declare an array as:

### Declaration:-

**Syntax:** data\_type array\_name[row\_size][column\_size];

**Ex:**   int arr[3][3] ;

Where first index value shows the number of the rows and second index value shows the no. of the columns in the array.

These are stored in the memory as given below.

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

### Initialization :-

To initialize values for variable length arrays we can use scanf statement and loop constructs.

### Ex:-

for (i=0 ; i<3; i++)

    for(j=0;j<3;j++)

        scanf(“%d”,&arr[i][j]);

#### 4. Define multi-dimensional arrays? How to declare multi-dimensional arrays?

**Ans: Multidimensional arrays** are often known as array of the arrays. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional arrays we will be discussing mainly about two dimensional arrays and a bit about three dimensional arrays.

**Syntax:**        data\_type array\_name[size1][size2][size3] ----- [sizeN];

In 2-D array we can declare an array as :

```
int arr[3][3] = { 1, 2, 3,  
4, 5, 6,  
7, 8, 9  
};
```

where first index value shows the number of the rows and second index value shows the number of the columns in the array. To access the various elements in 2-D array we can use:

```
printf("%d", a[2][3]);
```

*/\* output will be 6, as a[2][3] means third element of the second row of the array \*/* In 3-D we can

declare the array in the following manner :

```
int arr[3][3][3] =  
{  
  1, 2, 3,  
  4, 5, 6,  
  7, 8, 9,  
  
  10, 11, 12,  
  13, 14, 15,  
  16, 17, 18,  
  
  19, 20, 21,  
  22, 23, 24,  
  25, 26, 27  
};
```

*/\* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns \*/*

If we want to access the element the in 3-D array we can do it as follows: `printf("%d",a[2][2][2]);`

*/\* its output will be 26, as a[2][2][2] means first value in [] corresponds to the grid no. i.e. 3 and the second value in [] means third row in the corresponding grid and last [] means third column  
\*/ Ex:-*

```
int arr[3][5][12];
```

```
float table[5][4][5][3];
```

arr is 3D array declared to contain 180 (3\*5\*12) int type elements. Similarly table is a 4D array containing 300 elements of float type.

### 5. Write a program to perform matrix addition.

**Ans: /\*ADDITION OF TWO MATRICES\*/**

```
#include<stdio.h>
#include<conio.h> #include<process.h> void main()
{
    int a[10][10],b[10][10],c[10][10];
    int i,j,m,n,p,q;
    printf("\n Enter the size of Matrix A:"); scanf("%d%d", &m,&n);
    printf("\n Enter the size of Matrix B:"); scanf("%d%d", &p,&q);
    if(m!=p || n!=q)
    {
        printf("Matrix addition not possible."); exit(0);
    }
    printf(" Enter the Matrix A values:\n");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    printf(" Enter the Matrix B values:\n");
    for(i=0;i<p;i++)
        for(j=0;j<q;j++)
            scanf("%d",&b[i][j]);
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
```



```
        c[i][j]=a[i][j]+b[i][j];
printf("\n The Matrix A is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        printf(" %d",a[i][j]);
    printf("\n");
}
printf("\n The Matrix B is\n");
for(i=0;i<p;i++)
{
    for(j=0;j<q;j++)
        printf(" %d",b[i][j]);
    printf("\n");
}
printf("\n The Output Matrix C is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        printf(" %d",c[i][j]);
    printf("\n");
}
}
```

**OUTPUT:**

Enter the size of Matrix A: 2 3

Enter the size of Matrix B: 2 3

Enter the Matrix A values:

1 2 3

4 5 6

Enter the Matrix B values:

6 5 4

3 2 1

The Matrix A is

1 2 3

4 5 6

The Matrix B is

6 5 4

3 2 1

The Output Matrix C is

7 7 7

7 7 7

**6. Write a program to perform matrix multiplication**

**Ans: /\*MATRIX MULTIPLICATION\*/**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10][10],b[10][10],c[10][10];
    int i,j,k,m,n,p,q;
    printf("\ Enter the size of Matrix A:");
    scanf("%d%d", &m,&n);
    printf("\ Enter the size of Matrix B:");
    scanf("%d%d", &p,&q);
    if(n!=p)
    {
        printf("Matrix Multiplication not possible.");
        exit(0);
    }
    printf(" Enter the Matrix A values:\n");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
```

```
printf(" Enter the Matrix B values:\n");
for(i=0;i<p;i++)
    for(j=0;j<q;j++)
        scanf("%d",&b[i][j]);
for(i=0;i<m;i++)
    for(j=0;j<q;j++)
    {
        c[i][j]=0;
        for(k=0;k<n;k++)
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
printf("\n The Matrix A is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)

        printf(" %d",a[i][j]);

    printf("\n");
}
printf("\n The Matrix B is\n");
for(i=0;i<p;i++)
{
    for(j=0;j<q;j++)

        printf(" %d",b[i][j]);

    printf("\n");
}
printf("\n The Output Matrix C is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)

        printf(" %d",c[i][j]);
```

```
        printf("\n");  
    }  
}
```

**OUTPUT:**

Enter the size of Matrix A: 2 3

Enter the size of Matrix B: 3 2

Enter the Matrix A values:

2 2 2

2 2 2

Enter the Matrix B values:

3 3

3 3

3 3

The Matrix A is

2 2 2

2 2 2

The Matrix B is

3 3

3 3

3 3

The Output Matrix C is

15 15

15 15

## 7. Define C string? How to declare and initialize C strings with an example?

**Ans: C Strings:-**

In C language a string is group of characters (or) array of characters, which is terminated by delimiter \0 (null). Thus, C uses variable-length delimited strings in programs.

### Declaring Strings:-

C does not support string as a data type. It allows us to represent strings as character arrays. In C, a string variable is any valid C variable name and is always declared as an array of characters.

**Syntax:-** `char string_name[size];`

The size determines the number of characters in the string name.

**Ex:-** `char city[10];`  
`char name[30];`

### Initializing strings:-

There are several methods to initialize values for string variables.

**Ex:-** `char str1[6]="HELLO";`

H	E	L	L	O	\0
---	---	---	---	---	----

**Ex:-** `char month[]="JANUARY";`

J	A	N	U	A	R	Y	\0
---	---	---	---	---	---	---	----

**Ex:-** `char city[8]="NEWYORK";`  
`char city[8]={',','N','E','W','Y','O','R','K','\0'};`

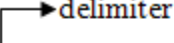
The string city size is 8 but it contains 7 characters and one character space is for NULL terminator.

### Storing strings in memory:-

In C a string is stored in an array of characters and terminated by \0 (null).

**Ex:-**

H	E	L	L	O	\0
---	---	---	---	---	----



A string is stored in array; the name of the string is a pointer to the beginning of the string. The character requires only one memory location.

If we use one-character string it requires two locations. The difference is shown below,

H
---

 a character      

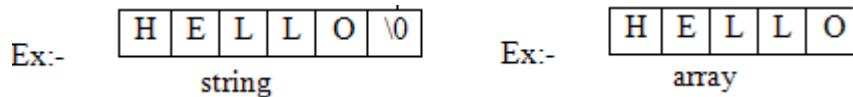
H	\0
---	----

 one-character string      

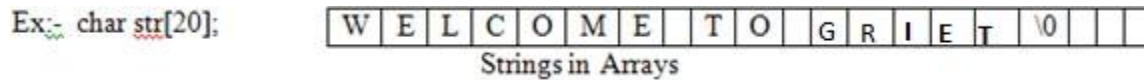
\0
----

 empty string

The difference between array and string is shown below,



Because strings are variable-length structure, we must provide enough room for maximum-length string to store and one byte for delimiter.



### Why do we need null?

A string is not a data type but a data structure. String implementation is logical not physical. The physical structure is array in which the string is stored. The string is variable-length, so we need to identify logical end of data in that physical structure.

### String constant (or) Literal:-

String constant is a sequence of characters enclosed in double quotes. When string constants are used in C program, it automatically initializes a null at end of string.

Ex:- "Hello"  
      "Welcome"  
      "Welcome to C Lab"

### 8. Explain about the string Input/ Output functions with example?

Ans:.

### Reading and Writing strings:-

C language provides several string handling functions for input and output.

### String Input/Output Functions:-

C provides two basic methods to read and write strings. Using formatted input/output functions and using a special set of functions.

### Reading strings from terminal:-

(a) formatted input function:- scanf can be used with %s format specification to read a string.

Ex:- `char name[10];`  
  
`scanf("%s",name);`

Here don't use "&" because name of string is a pointer to array. The problem with scanf is that it terminates its input on the first white space it finds.

Ex:- NEW YORK

Reads only NEW (from above example).

(b) **Unformatted input functions:-**

(1) **getchar()**:- It is used to read a single character from keyboard. Using this function repeatedly we may read entire line of text

Ex:- `char ch="z";`  
`ch=getchar();`

(2) **gets()**:- It is more convenient method of reading a string of text including blank spaces.

Ex:- `char line[100];`  
`gets(line);`

**Writing strings on to the screen:-**

(1) **Using formatted output functions:-** printf with %s format specifier we can print strings in different formats on to screen.

Ex:- `char name[10];`  
`printf("%s",name);`

Ex:- `char name[10];`  
`printf("%0.4",name);`

J	A	N	U
---	---	---	---

/\* If name is JANUARY prints only 4 characters ie., JANU \*/

`Printf("%10.4s",name);`

						J	A	N	U
--	--	--	--	--	--	---	---	---	---

`printf("%-10.4s",name);`

J	A	N	U						
---	---	---	---	--	--	--	--	--	--

(2) **Using unformatted output functions:-**

(a) **putchar()**:- It is used to print a character on the screen.

Ex:- `putchar(ch);`

(b) **puts()**:- It is used to print strings including blank spaces.

Ex:- `char line[15]="Welcome to lab";`  
`puts(line);`

**9. Explain about the following string handling functions with example programs.**

(i) **strlen**      (ii) **strcpy**      (iii) **strcmp**      (iv) **strcat**

**Ans:**

C supports a number of string handling functions. All of these built-in functions are aimed at performing various operations on strings and they are defined in the header file **string.h**.

**(i). strlen( )**

This function is used to find the length of the string excluding the NULL character. In other words, this function is used to count the number of characters in a string. Its syntax is as follows:

**Int strlen(string);**

**Example:**      `char str1[ ] = "WELCOME";`  
                 `int n;`  
                 `n = strlen(str1);`

**/\* A program to calculate length of string by using strlen() function\*/**

```
#include<stdio.h>
#include<string.h>
void main ()
{
    char string1[50];
    int length;
    printf("\n Enter any string:");
    gets(string1);
    length=strlen(string1);
    printf("\n The length of string=%d",length);
}
```

**(ii). strcpy( )**



This function is used to copy one string to the other. Its syntax is as follows:

**strcpy(string1,string2);**

where string1 and string2 are one-dimensional character arrays.

This function copies the content of string2 to string1.

E.g., string1 contains master and string2 contains madam, then string1 holds madam after execution of the strcpy (string1,string2) function.

**Example:**

```
char str1[ ] = "WELCOME";  
char str2[ ] ="HELLO";  
strcpy(str1,str2);
```

**/\* A program to copy one string to another using strcpy() function \*/**

```
{  
  
    char string1[30],string2[30];  
    printf("\n Enter first string:");  
    gets(string1);  
    printf("\n Enter second string:");  
    gets(string2);  
    strcpy(string1,string2);  
    printf("\n First string=%s",string1);  
    printf("\n Second string=%s",string2);  
}
```

### (iii). **strcmp ( )**

This function compares two strings character by character (ASCII comparison) and returns one of three values {-1,0,1}. The numeric difference is „0“ if strings are equal .If it is negative string1 is alphabetically above string2 .If it is positive string2 is alphabetically above string1.

Its syntax is as follows:

**int strcmp(string1,string2);**

**Example:**     char str1[ ] = "ROM";  
                  char str2[ ] ="RAM";  
                  strcmp(str1,str2); (or)  
                  strcmp("ROM","RAM");

**/\* A program to compare two strings using strcmp() function \*/**

```
{  
    char string1[30],string2[15];  
    int x;  
    printf("\n Enter first string:");  
    gets(string1);  
    printf("\n Enter second string:");  
    gets(string2);  
    x=strcmp(string1,string2);  
    if(x==0)  
        printf("\n Both strings are equal");  
    else if(x>0)  
        printf("\n First string is bigger");  
    else  
        printf("\n Second string is bigger");  
}
```

**(iv).     strcat ( )**

This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string. Its syntax as follows:

**strcat(string1,string2);**

where string1 and string2 are one-dimensional character arrays.

This function joins two strings together. In other words, it adds the string2 to string1 and the string1 contains the final concatenated string. E.g., string1 contains **prog** and string2 contains **ram**, then string1 holds **program** after execution of the strcat() function.

**Example:**     char str1[10] = "VERY";  
                  char str2[ 5] ="GOOD";

```
strcat(str1,str2);
```

**/\* A program to concatenate one string with another using strcat() function\*/**

```
#include<stdio.h>
```

```
#include<string.h>
```

```
main()
```

```
{
```

```
    char string1[30],string2[15];
```

```
    printf("\n Enter first string:");
```

```
    gets(string1);
```

```
    printf("\n Enter second string:");
```

```
    gets(string2);
```

```
    strcat(string1,string2);
```

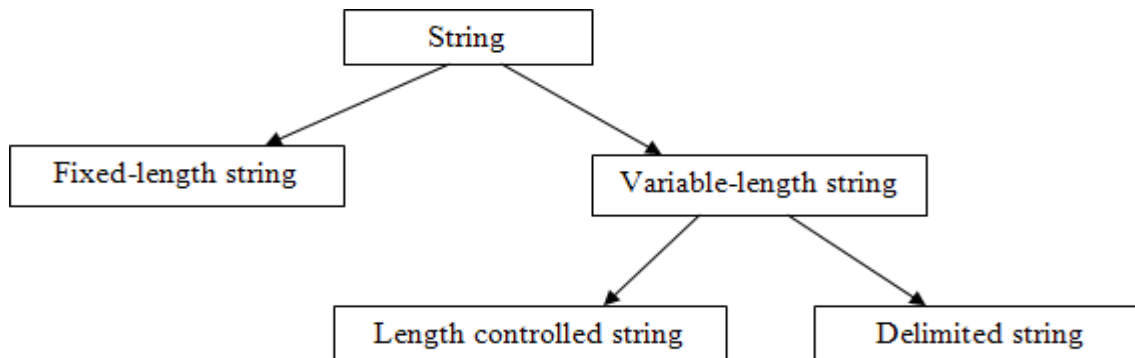
```
    printf("\n Concatenated string=%s",string1);
```

```
}
```

**10. Write about storage representation of fixed and variable length format strings with example?**

**Ans: String concepts:-**

In general a string is a series of characters (or) a group of characters. While implementation of strings, a string created in pascal differs from a string created in C language.



**1. Fixed-length strings:**

When implementing fixed-length strings, the size of the variable is fixed. If we make it too small we can't store, if we make it too big, then waste of memory. And another problem is we can't differentiate data (characters) from non-data (spaces, null etc).

**2. Variable-length string:**

The solution is creating strings in variable size; so that it can expand and contract to accommodate data. Two common techniques used,

**(a) Length controlled strings:**

These strings added a count which specifies the number of characters in the string.

Ex:-

5	H	E	L	L	O
---	---	---	---	---	---

**(b) Delimited strings:**

Another technique is using a delimiter at the end of strings. The most common delimiter is the ASCII null character (\0).

Ex:-

H	E	L	L	O	\0
---	---	---	---	---	----

## 11. How can we declare and initialize Array of strings in C? Write a program to read and display array of strings.

**Ans:** We have array of integers, array of floating point numbers, etc.. Similarly we have array of strings also. Collection of strings is represented using array of strings.

**Declaration:-**

Char arr[row][col]; where,

- arr - name of the array
- row - represents number of strings
- col - represents size of each string

**Initialization:-**

char arr[row][col] = { list of strings };

**Example:-**char city[5][10]={“DELHI”,“CHENNAI”,“BANGALORE”,“HYDERABAD”,“MUMBAI”};

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

In the above storage representation memory is wasted due to the fixed length for all strings.

**Program:**

```
#include<stdio.h>
#include<string.h>
int main()
{
    int i,j,count;
    char str[25][25],temp[25];
    puts("How many strings u are going to enter?: ");
    scanf("%d",&count);
    puts("Enter Strings one by one: ");
    for(i=0;i<=count;i++)
        gets(str[i]);
    for(i=0;i<=count;i++)
        for(j=i+1;j<=count;j++)
        {
            if(strcmp(str[i],str[j])>0)
            {
                strcpy(temp,str[i]);
                strcpy(str[i],str[j]);
                strcpy(str[j],temp);
            }
        }
    printf("Order of Sorted Strings:");
    for(i=0;i<=count;i++)
        puts(str[i]);
    return 0;
}
```

**12. Define a structure? Write the syntax for structure declaration with an example.****Ans:**

**Definition:** A structure is a collection of one or more variables of different data types, grouped together under a single name. By using structures variables, arrays, pointers etc can be grouped together.

Structures can be declared using two methods as follows:

**1. Tagged Structure:**

The structure definition associated with the structure name is referred as tagged structure. It doesn't create an instance of a structure and does not allocate any memory.

The **general form or syntax of tagged structure** definition is as follows,

struct TAG	<b>Ex:-</b>	struct student
{		{
Type variable1;		int htno[10];
Type variable2;		char name[20];
.....		float marks[6];
.....		};
Type variable-n;		
};		

Where,

- struct is the keyword which tells the compiler that a structure is being defined.
- Tag\_name is the name of the structure.
- variable1, variable2 ... are called members of the structure.
- The members are declared within curly braces.
- The closing brace must end with the semicolon.

**2. Type-defined structures:-**

The structure definition associated with the keyword **typedef** is called type-defined structure.

This is the most powerful way of defining the structure.

The **syntax of typedef structure** is

typedef struct	<b>Ex:-</b>	typedef struct
{		{
Type variable1;		int htno[10];
Type variable2;		char name[20];
.....		float marks[6];

```
..... }student;
```

Type variable-n;

```
}Type;
```

where

- typedef is keyword added to the beginning of the definition.
- struct is the keyword which tells the compiler that a structure is being defined.
- variable1, variable2...are called fields of the structure.
- The closing brace must end with type definition name which in turn ends with semicolon.

### Variable declaration:

Memory is not reserved for the structure definition since no variables are associated with the structure definition. The members of the structure do not occupy any memory until they are associated with the structure variables.

After defining the structure, variables can be defined as follows:

For first method,

```
struct TAG v1,v2,v3....vn;
```

For second method, which is most powerful is,

```
Type v1,v2,v3,...vn;
```

### Alternate way:

```
struct TAG
```

```
{
```

```
    Type variable1;
```

```
    Type variable2;
```

```
    .....
```

```
    Type variable-n;
```

```
} v1, v2, v3;
```

### Ex:

```
struct book
```

```
{
```

```
    char name[30];
```

```
    int pages;
```

```
    float price;
```

```
}b1,b2,b3;
```

**13. Declare the C structures for the following scenario:**

**College contains the following fields: College code (2characters), College Name, year of establishment, number of courses. Each course is associated with course name (String), duration, number of students. (A College can offer 1 to 50 such courses)**

**Ans:**

**(i). Structure definition for college :-**

struct **college**

```
{  
    char code[2];  
    char college_name[20];  
    int year;  
    int no_of_courses;  
};
```

**Variable declaration for structure college :-**

```
void main( )  
{  
    struct college col1,col2,col3;  
    ....  
}
```

**(ii). Structure definition for course :-**

struct **course**

```
{  
    char course_name[20];  
    float duration;  
    int no_of_students;  
};
```

**Variable declaration for structure course :-**

```
void main( )  
{  
    struct course c1,c2,c3;  
    ....  
}
```



**14. How to initialize structures in “C”? Write an example.****Ans:**

The rules for structure initialization are similar to the rules for array initialization. The initializers are enclosed in braces and separated by commas. They must match their corresponding types in the structure definition.

The syntax is shown below,

```
struct tag_name variable = {value1, value2,... value-n};
```

Structure initialization can be done in any one of the following ways

**1. Initialization along with Structure definition:-**

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

```
struct student
{
    char name [5];

    int roll_number;

    float avg;
} s1= {"Ravi", 10, 67.8};
```

The various members of the structure have the following values.

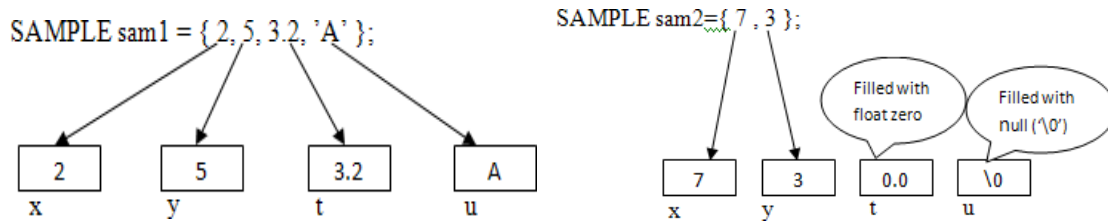
←--- name ----->					← roll_number ->			←----- avg ----->		
<b>R</b>	<b>a</b>	<b>v</b>	<b>i</b>	<b>\0</b>	<b>1</b>	<b>0</b>	<b>6</b>	<b>7</b>	<b>.</b>	<b>8</b>

Figure 5.2 Initial Value of S1

**2. Initialization during Structure declaration:-**

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

```
typedef struct
{
    int x;
    int y;
    float t;
    char u;
} SAMPLE;
```



**Figure** Initializing Structures

The figure shows two examples of structure in sequence. The first example demonstrates what happens when not all fields are initialized. As we saw with arrays, when one or more initializers are missing, the structure elements will be assigned null values, zero for integers and floating-point numbers, and null („\0“) for characters and strings.

**15. Define a structure type *personal*, that would contain person name, date of joining and salary. Write a program to initialize one person data and display the same.**

**Ans:**

```
struct personal
```

```
{
```

```
    char name[20];
```

```
    int day;
```

```
    char month[10];
```

```
    int year;
```

```
    float salary;
```

```
};
```

```
void main( )
```

```
{
```

```
    struct personal person = { "RAMU", 10, "JUNE", 1998, 20000.00};
```

```
    printf("Output values are:\n");
```

```
    printf("%s%d%s%d%f", person.name, person.day, person.month, person.year, person.salary );
```

```
}
```

**Output:-** RAMU 10 JUNE 1998 20000

**16. How to access the data for structure variables using member operator(“.”)? Explain with an example.**

**Ans:**

We know that variables can be accessed and manipulated using expressions and operators. On the similar lines, the structure members can be accessed and manipulated. The members of a structure can be accessed by using dot(.) operator.

**dot (.) operator**

Structures use a **dot (.) operator** (also called **period operator** or **member operator**) to refer its elements. Before dot, there must always be a structure variable. After the dot, there must always be a structure element.

The **syntax** to access the structure members as follows,

structure\_variable\_name . structure\_member\_name

Consider the example as shown below,

struct student

```
{  
    char name [5];  
    int roll_number;  
    float avg;  
};
```

struct student s1= {"Ravi", 10, 67.8};

The members can be accessed using the variables as shown below,

s1.name --> refers the string "ravi"

s1.roll\_number --> refers the roll\_number 10

s1.avg --> refers avg 67.8

**17. Define a structure type *book*, that would contain book name, author, pages and price. Write a program to read this data using member operator (“.”) and display the same.**

**Ans:**

```
struct book
{
    char name[20];
    int day;
    char month[10];
    int year;
    float salary;
};

void main( )
{
    struct book b1;
    printf("Input values are:\n");
    scanf("%s %s %d %f", b1.title, b1.author, b1.pages, b1.price);
    printf("Output values are:\n");
    printf("%s\n %s\n %d\n %f\n", b1.title, b1.author, b1.pages, b1.price);
}
```

**Output:-**

Input values are: C& DATA STRUCTURES KAMTHANE 609 350.00

Output values are:

C& DATA STRUCTURES

KAMTHANE

609

350.00

**18. How to pass a structure member as an argument of a function? Write a program to explain it.****Ans:**

Structures are more useful if we are able to pass them to functions and return them.

**By passing individual members of structure**

This method is to pass each member of the structure as an actual argument of the function call. The actual arguments are treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.

**Program:**

```
#include<stdio.h>
typedef struct emp
{
    char name[15];
    int emp_no;
    float salary;
}record;

float arriers(char *s, int n, float m);
main ( )
{
    record e1 = {"smith",2,20000.25};
    e1.salary = arriers(e1.name,e1.emp_no,e1.salary);
}
float arriers(char *s, int n, float m)
{
    m = m + 2000;
    printf("\n%s %d %f ",s, n, m);
    return m;
}
```

**Output**

```
smith
2
22000.250000
```

**19. How to pass an entire structure as an argument of a function?(or) Write a program to pass entire structure as an argument of a function.**

**Ans:** Structures are more useful if we are able to pass them to functions and return them.

**Passing Whole Structure:** This method involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in the original structure. It is therefore, necessary for the function to return the entire structure back to the calling function. The general format of sending a copy of a structure to the called function is:

```
return_type function_name (structure_variable_name);
```

The called function takes the following form:

```
data_type function_name(struct_type tag_name)
{
    .....
    ..... return(expression);
}
```

The called function must be declared for its type, appropriate to the data type it is expected to return.

The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.

The return statement is necessary only when the function is returning some data back to the calling function. The expression may be any simple variable or structure variable or an expression using simple variables. When a function returns a 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.

**Program:**

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float percentage;
};
void func(struct student record);
int main()
{
```

```
struct student record;
record.id=1;
strcpy(record.name, "Raju");
record.percentage = 86.5;
return 0;
}
void func(struct student record)
{
    printf(" Id is: %d \n", record.id);
    printf(" Name is: %s \n", record.name);
    printf(" Percentage is: %f \n", record.percentage);
}
```

**Output:** Id is: 1

Name is: Raju Percentage is: 86.500000

**20. Write a program for illustrating a function returning a structure.**

**Ans:**

```
typedef struct
{
    char name [15];
    int emp_no;
    float salary;
} record;
#include<stdio.h>
#include<string.h>
void main ( )
{
    record change (record);
    record e1 = {"Smith", 2, 20000.25};
    printf ("\nBefore Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
    e1 = change(e1);
    printf ("\nAfter Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
}
record change (record e2)
{
    strcpy (e2.name, "Jones");
    e2.emp_no = 16;
    e2.salary = 9999; return e2;
}
```

**Output:** Smith 2 20000.25

Jones 16 9999.99

**21. What is an array of structure? Declare a variable as array of structure and initialize it? (or) When the array of structures is used? Write syntax for array of structure.**

**Ans:**

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

As we have an array of integers, we can have an array of structures also. For example, suppose we want to store the information of class of students, consisting of name, roll\_number and marks, A better approach would be to use an array of structures. Array of structures can be declared as follows,

**struct tag\_name arrayofstructure[size];**

Let's take an example, to store the information of 3 students, we can have the following structure definition and declaration,

```
struct student
{
    char name[10];
    int rno;
    float avg;
};
struct student s[3];
```

Defines an array called s, which contains three elements. Each element is defined to be of type struct student.

For the student details, array of structures can be initialized as follows,

```
struct student s[3]={{"ABC",1,56.7},{"xyz",2,65.8},{"pqr",3,82.4}};
```

**Ex:** An array of structures for structure employee can be declared as struct employee emp[5];

Let's take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
{
    int empid;
    char name[10];
    float salary;
```



```
};
```

```
struct employee emp[5];
```

Defines array called emp, which contains five elements. Each element is defined to be of type struct employee.

For the employee details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,"ramu",25,20000},
                           {2,"ravi",65000},
                           {3,"tarun",82000},
                           {4,"rupa",5000},
                           {5,"deepa",27000}};
```

**22. Write a C program to calculate student-wise total marks for three students using array of structure.**

**Ans:**

```
#include<stdio.h>
```

```
struct student
```

```
{
```

```
    char rollno[10];
```

```
    char name[20];
```

```
    float sub[3];
```

```
    float total;
```

```
};
```

```
void main( )
```

```
{
```

```
    int i, j, total = 0;
```

```
    struct student s[3];
```

```
    printf("\t\t\t Enter 3 students details");
```

```
    for(i=0; i<3; i++)
```

```
    {
```

```
        printf("\n Enter Roll number of %d Student:",i);
```

```
        gets(s[i].rollno);
```

```
        printf(" Enter the name:");
```

```
        gets(s[i].name);
```

```
        printf(" Enter 3 subjects marks of each student:");
```

```
        total=0;
```

```

        for(j=0; j<3; j++)
        {
            scanf("%d",&s[i].sub[j]);
            total = total + s[i].sub[j];
        }
    }
    printf("\n*****");
    printf("\n\t\t\t Student details:");
    printf("\n*****");
    for(i=0; i<n; i++)
    {
        printf ("\n Student %d:",i+1);
        printf ("\n Roll number:%s\n Name:%s",s[i].rollno,s[i].name);
        printf ("\nTotal marks =%f", s[i].total);
    }
}

```

**23. Write a C program using array of structure to create employee records with the following fields: emp-id, name, designation, address, salary and display it.**

**Ans:**

```

#include<stdio.h>
struct employee
{
    int emp_id;
    char name[20];
    char designation[10];
    char address[20];
    float salary;
}emp[3];
void main( )
{
    int i;
    printf("\t\t\t Enter 3 employees details");
    for(i=0; i<3; i++)
    {

```

```
scanf("%d",&emp[i].emp_id);
gets(emp[i].name);
gets(emp[i].designation);
gets(emp[i].address);
scanf("%f",&emp[i].salary);
}
printf("\n\t\t\t Employee details:");
for(i=0; i<3; i++)
{
    printf("%d",emp[i].emp_id);
    puts(emp[i].name);
    puts(emp[i].designation);
    puts(emp[i].address);
    printf("%f",emp[i].salary);
}
}
```

**24. What is structure within structure? Give an example for it.(or) Write a C program to illustrate the concept of structure within structure(or) Explain about nested structures.**

**Ans:**

**Nested Structure (Structure within structure)**

A structure which includes another structure is called nested structure or structure within structure. i.e a structure can be used as a member of another structure. There are two methods for declaration of nested structures.

**(i) The syntax for the nesting of the structure is as follows**

```
struct tag_name1
{
    type1 member1;
    .....
};
struct tag_name2
{
    type1 member1;
    .....
    struct tag_name1 var;
    .....
```

```
};
```

The syntax for accessing members of a nested structure as follows,  
 outer\_structure\_variable . inner\_structure\_variable . member\_name

**(ii) The syntax of another method for the nesting of the structure is as follows**

```
struct structure_nm
{
    <data-type> element 1;
    <data-type> element 2;
    -----
    -----
    <data-type> element n;
    struct structure_nm
    {
        <data-type> element 1;
        <data-type> element 2;
        -----
        -----
        <data-type> element n;
    }inner_struct_var;
}outer_struct_var;
```

**Example :**

```
struct stud_Res
{
    int rno;
    char nm[50];
    char std[10];
    struct stud_subj
    {
        char subjnm[30];
        int marks;
    }subj;
}result;
```

In above example, the structure stud\_Res consists of stud\_subj which itself is a structure with two members. Structure stud\_Res is called as 'outer structure' while stud\_subj is called as 'inner structure.'

The members which are inside the inner structure can be accessed as follow:

result.subj.subjnm

result.subj.marks

Program to demonstrate nested structures.

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
struct stud_Res
```

```
{
```

```
    int rno;
```

```
    char std[10];
```

```
    struct stud_Marks
```

```
    {
```

```
        char subj_nm[30];
```

```
        int subj_mark;
```

```
    }marks;
```

```
}result;
```

```
void main()
```

```
{
```

```
    printf("\n\t Enter Roll Number : ");
```

```
    scanf("%d",&result.rno);
```

```
    printf("\n\t Enter Standard : ");
```

```
    scanf("%s",result.std);
```

```
    printf("\n\t Enter Subject Code : ");
```

```
    scanf("%s",result.marks.subj_nm);
```

```
    printf("\n\t Enter Marks : ");
```

```
    scanf("%d",&result.marks.subj_mark);
```

```
    printf("\n\n\t Roll Number : %d",result.rno);
```

```
    printf("\n\n\t Standard : %s",result.std);
```

```
    printf("\nSubject Code : %s",result.marks.subj_nm);
```

```
    printf("\n\n\t Marks : %d",result.marks.subj_mark);
```

```
}
```

**Output:**

Enter roll number : 1 Enter standard :Btech

Enter subject code : GR11002 Enter marks : 63

Roll number :1 Standard :Btech

Subject code : GR11002 Marks : 63

**25.** Write a C program using nested structures to read 3 employees details with the following fields; emp-id, name, designation, address, da ,hra and calculate gross salary of each employee.

**Ans:**

```
#include<stdio.h>

struct employee
{
    int emp_id;
    char name[20];
    char designation[10];
    char address[20];
    struct salary
    {
        float da;
        float hra;
    }sal;
}emp[3];

void main( )
{
    int i;
    printf("\t\t\t Enter 3 employees details");
    grosssalary = 0;
    for(i=0; i<3; i++)
    {
        scanf("%d",&emp[i].emp_id);
        gets(emp[i].name);
        gets(emp[i].designation);
        gets(emp[i].address);
        scanf("%f",&emp[i].sal.da);
        scanf("%f",&emp[i].sal.hra);
        grosssalary = grosssalary + emp[i].sal.da + emp[i].sal.hra;
    }
    printf("\n*****");
```

```
printf("\n\t\t Employee details with gross salary:");
printf("\n*****");
for(i=0; i<3; i++)
{
    printf("%d",emp[i].emp_id);
    puts(emp[i].name);
    puts(emp[i].designation);
    puts(emp[i].address);
    printf("%f",emp[i].sal.da);
    printf("%f",emp[i].sal.hra);
    printf("%f",grosssalary);
}
}
```

**26. Distinguish between Arrays within Structures and Array of Structure with examples.**

**Ans:**

**Arrays within structure:**

It is also possible to declare an array as a member of structure, like declaring ordinary variables. For example to store marks of a student in three subjects then we can have the following definition of a structure.

```
struct student
{
    char name [5];
    int roll_number;
    int marks [3];
    float avg;
};
```

Then the initialization of the array marks done as follows,

```
struct student s1= {"ravi", 34, {60,70,80}};
```

The values of the member marks array are referred as follows,

s1.marks [0] --> will refer the 0<sup>th</sup> element in the marks

s1.marks [1] --> will refer the 1st element in the marks

s1.marks [2] --> will refer the 2nd element in the marks

### Array of structure

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

#### Ex:

An array of structures for structure *employee* can be declared as `struct employee emp[5];`

Let's take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
```

```
{  
    int empid;  
    char name[10];  
    float salary;  
};
```

```
struct employee emp[5];
```

Defines array called `emp`, which contains five elements. Each element is defined to be of type `struct student`.

For the student details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,"ramu",25,20000},  
                           {2,"ravi",65000},  
                           {3,"tarun",82000},  
                           {4,"rupa",5000},  
                           {5,"deepa",27000}};
```



**27. Write a C program using structure to create a library catalogue with the following fields: Access number, author's name, Title of the book, year of publication, publisher's name, and price.**

**Ans:**

```
struct library
{
    int acc_no;
    char author[20];
    char title[10];
    int year_pub;
    char name_pub[20];
    float price;
};

void main( )
{
    struct library lib; printf("Input values are:\n");
    scanf("%d %s %s %d %s%f", &lib.acc_no, lib.author, lib.title, &lib.year_pub, lib.name_pub,
    &lib.price);
    printf("Output values are:\n\n");
    printf("Access number = %d\n Author name = %s\n
        Book Title = %s\n Year of publication = %d \n Name of publication = %s\n
        Price = %f", lib.acc_no, lib.author, lib.title, lib.year_pub, lib.name_pub, lib.price);
}
```

**28. What is self-referential structure? Explain through example.****Ans:****Self-referential structure**

A structure definition which includes at least one member as a pointer to the same structure is known as self-referential structure. It can be linked together to form useful data structures such as lists, queues, stacks and trees. It is terminated with a NULL pointer .

The syntax for using the self referential structure is as follows,

```
struct tag_name
{
    Type1 member1;
    Type2 member2;
    .....
    struct tag_name *next;
};
```

**Ex:-**

```
struct node
{
    int data;
    struct node *next;
} n1, n2;
```

**29. Explain unions in C language? Differentiate structures and unions.****Ans:**

A union is one of the derived data types. Union is a collection of variables referred under a single name. The syntax, declaration and use of union is similar to the structure but its functionality is different.

The general format or syntax of a union definition is as follows,

**Syntax:**

```
union union_name
{
    <data-type> element 1;
    <data-type> element 2;
    .....
}union_variable;
```

**Example:**

```
union techno
{
    int comp_id;
    char nm;
    float sal;
}tch;
```

A union variable can be declared in the same way as structure variable. union tag\_name var1, var2...;

A union definition and variable declaration can be done by using any one of the following

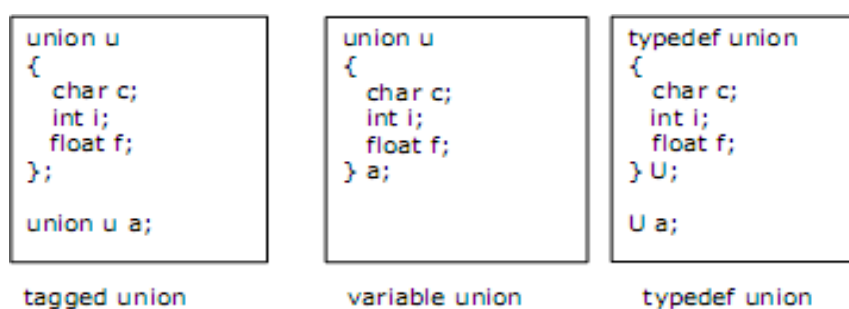


Figure 5.7 Types of Union Definitions

We can access various members of the union as mentioned: a.c a.i a.f and memory organization is shown below,

In the above declaration, the member **f** requires 4 bytes which is the largest among all the members.

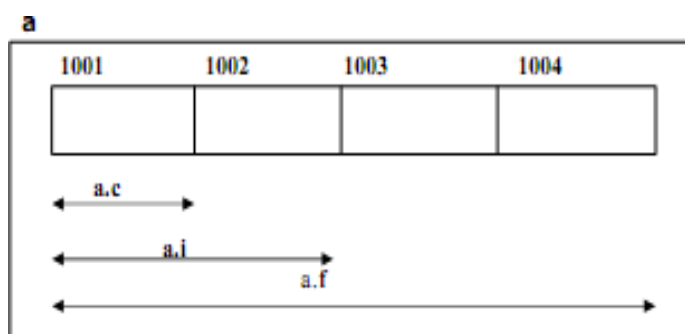


Figure 5.8: Memory Organization Union

Figure 5.8 shows how all the three variables share the same address. The size of the union here is 4 bytes.

A union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supersedes the previous members' value.

**Difference between structure and union:-**

	<b>Structure</b>	<b>Union</b>
(i) Keyword	Struct	Union
(ii) Definition	A structure is a collection of logically related elements, possibly of different types, having a single name.	A union is a collection of logically related elements, possibly of different types, having a single name, shares single memory location.
(iii) Declaration	<pre>struct tag_name{     type1 member1; type1 member2;     ..... }; struct tag_name var;</pre>	<pre>union tag_name{     type1 member1; type1 member2;     ..... }; union tag_name var;</pre>
(iv) Initialization	Same.	Same.
(v) Accessing	Accessed by specifying structure_ variable_name.member_name	Accessed by specifying union_ variable_name.member_name
(vi) Memory Allocation	Each member of the structure occupies unique location, stored in contiguous locations.	Memory is allocated by considering the size of the largest member. All the members share the common location
(vii) Size	Size of the structure depends on the type of members, adding size of all the members. sizeof (st_var);	Size is given by the size of the largest member sizeof(un_variable)
(viii) Using pointers	Structure members can be accessed by using dereferencing operator dot and selection operator(->)	same as structure.
	We can have arrays as a member of structures. All members can be accessed at a time.	We can have array as a member of union Only one member can be accessed at a time.
	Nesting of structures is possible.	same.
	It is possible structure may contain union as a member.	It is possible union may contain structure as a member

**30. Explain in brief about pointers and structures.****Ans:****Pointers to structures:**

We have pointers pointing to int, float, arrays etc., We also have pointers pointing to structures. They are called as structure pointers.

To access the members of a structure using pointers we need to perform The following operations:

- Declare the structure variable
- Declare a pointer to a structure
- Assign address of structure variable to pointer
- Access members of structure using ( . ) operator or using (->)member selection operator or arrow operator.

**Syntax:**

```
struct tagname
{
    datatype member1;
    datatype member2;
    ...
};
struct tagname var;
struct tagname *ptr;
ptr=*var;
```

**to access the members**

(\*ptr).member1; or ptr->member1;

The parentheses around \*ptr are necessary because the member operator “.” has a higher precedence than the operator “\*”

**Example:**

```
struct student
{
    int rno;
    char name[20];
};
struct student s1;
struct student *ptr;
ptr=&s1;
```

to access (\*ptr).rno;

(\*ptr).name; (or)

ptr->rno;

ptr->name;

### **Program to read and display student details using pointers to structures**

struct student

{

int HTNO;

char NAME[20];

float AVG;

};

void main()

{

struct student s1;

struct student \*ptr;

ptr=&s1;

printf("Enter student details");

scanf("%d%s%f",&ptr->HTNO,ptr->NAME,&ptr->AVG);

printf("HTNO=%d",ptr->HTNO);

printf("NAME=%s",ptr->NAME);

printf("AVERAGE MARKS=%f",ptr->AVG);

}

### **31. What is pointer in c programming? What are its benefits?**

**Ans:**

Pointer is a user defined data type that creates special types of variables which can hold the address of primitive data type like char, int, float, double or user defined data type like function, pointer etc. or derived data type like array, structure, union, enum.

Examples:

**int** \*ptr;

In c programming every variable keeps two types of value.

1. Value of variable.
2. Address of variable where it has stored in the memory.

(1) Meaning of following simple pointer declaration and definition: int a=5;

int\* ptr;

```
ptr=&a;
```

Explanation:

#### About variable —a: :

1. Name of variable: a
2. Value of variable which it keeps: 5
3. Address where it has stored in memory: 1025 (assume)

#### About variable —ptr: :

4. Name of variable: ptr
5. Value of variable which it keeps: 1025
6. Address where it has stored in memory: 5000 (assume) Pictorial representation:



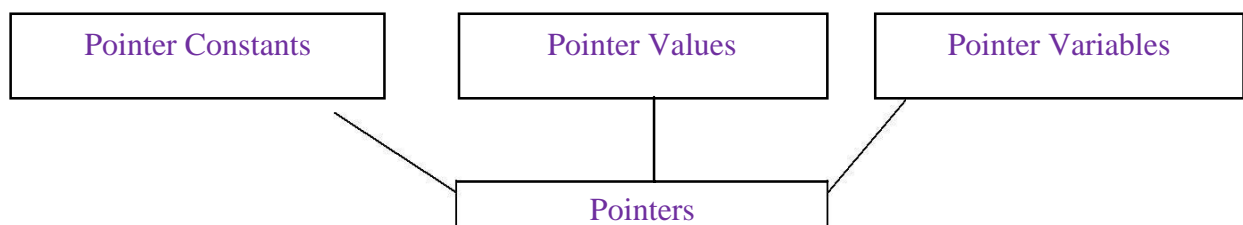
Note: A variable where it will be stored in memory is decided by operating system. We cannot guess at which location a particular variable will be stored in memory.

#### Pointers are built on three underlying concepts which are illustrated below:-

Memory addresses within a computer are referred to as *pointer constants*. We cannot change them. We can only use them to store data values. They are like house numbers.

We cannot save the value of a memory address directly. We can only obtain the value through the variable stored there using the address operator (&). The value thus obtained is known as *pointer value*. The pointer value (i.e. the address of a variable) may change from one run of the program to another.

Once we have a pointer value, it can be stored into another variable. The variable that contains a pointer value is called a *pointer variable*.



Benefits of using pointers are:-

- i. Pointers are more efficient in handling arrays and data tables.
- ii. Pointers can be used to return multiple values from a function via function arguments.

- iii. The use of pointer arrays to character strings results in saving of data storage space in memory.
- iv. Pointers allow C to support dynamic memory management.
- v. Pointers provide an efficient tool for manipulating dynamic data structures such as structures , linked lists , queues , stacks and trees.
- vi. Pointers reduce length and complexity of programs.
- vii. They increase the execution speed and thus reduce the program execution time.

**32. Explain the process of declaring and initializing pointers. Give an example**

**Ans:**

In C, every variable must be declared for its type. Since pointer variable contain addresses that belong to a separate data type ,they must be declared as pointers before we use them.

**a) Declaration of a pointer variable:**

The declaration of a pointer variable takes the following form:

**data\_type \*pt\_name;**

This tells the compiler three things about the variable pt\_name:

- 1) The \* tells that the variable pt\_name is a pointer variable
- 2) pt\_name needs a memory location
- 3) pt\_name points to a variable of type data\_type Ex: int \*p;

Declares the variable p as a pointer variable that points to an integer data type.

**b) Initialization of pointer variables:**

The process of assigning the address of a variable to a pointer variable is known as *initialization*.

Once a pointer variable has been declared we can use assignment operator to initialize the variable.

**Ex:**

```
int quantity ;
```

```
int *p;           //declaration
```

```
p=&quantity;     //initialization
```

We can also combine the initialization with the declaration:

```
int *p=&quantity;
```

Always ensure that a pointer variable points to the corresponding type of data.

It is also possible to combine the declaration of data variable, the declaration of pointer variable and the initialization of the pointer variable in one step.

```
int x, *p=&x;
```



**33. Explain in brief about & and \* operators? Explain how to access the value of a variable using pointer Explain how to access the address of a variable.**

**Ans:** The actual location of a variable in the memory is system dependent and therefore the address of a variable is not known to us immediately. In order to determine the address of a variable we use & operator in c. This is also called as the **address operator**. The operator & immediately preceding a variable returns the address of the variable associated with it. P=&x; would assign the address 5000 to the variable p. The & operator can be remembered as **address of**.

**Example program:**

```
main( )
{
    int a = 5 ;
    printf ( "\nAddress of a = %u", &a );
    printf ( "\nValue of a = %d", a );
}
```

**Output:** address of a=1444

value of a=5

The expression &a returns the address of the variable a, which in this case happens to be 1444. Hence it is printed out using %u, which is a format specified for printing an unsigned integer.

**Accessing a variable through its pointer:**

Once a pointer has been assigned the address of a variable, to access the value of the variable using pointer we use the operator **'\*'**, called **'value at address' operator**. It gives the value stored at a particular address. The „value at address“ operator is also called **'indirection' operator (or dereferencing operator)**.

**Ex:** void main()

```
{
    int a = 5 ;
    printf ( "\nAddress of a = %u", &a );
    printf ( "\nValue of a = %d", a );
    printf ( "\nValue of a = %d", *( &a ) );
}
```

**Output:** The output of the above program would be:

Address of a = 1444, Value of a = 5, Value of a = 5

**34. Explain how pointers are used as function arguments.**

**Ans:** When we pass addresses to a function, the parameter receiving the addresses should be pointers. The process of calling a function using pointers to pass the address of variables is known as “call by reference”. The function which is called by reference can change the value of the variable used in the call.

Example :-

```
void main()
{
    int x;
    x=50;
    change(&x); /* call by reference or address */
    printf("%d\n",x);
}

change(int *p)
{
    *p=*p+10;
}
```

When the function change () is called, the address of the variable x, not its value, is passed into the function change (). Inside change (), the variable **p** is declared as a pointer and therefore **p** is the address of the variable x.

The statement,

$$*p = *p + 10;$$

Means —add 10 to the value stored at the address **p**. Since **p** represents the address of **x**, the value of **x** is changed from 20 to 30. Thus the output of the program will be 30, not 20.

Thus, call by reference provides a mechanism by which the function can change the stored values in the calling function.

**35. Write a “C” function using pointers to exchange the values stored in two locations in the memory. (Or) Write a C program for exchanging of two numbers using call by reference mechanism.**

**Ans:**

Using pointers to exchange the values

- Pointers can be used to pass addresses of variables to called functions, thus allowing the called function to alter the values stored there.
- Passing only the copy of values to the called function is known as "**call by value**".
- Instead of passing the values of the variables to the called function, we pass their addresses, so that the called function can change the values stored in the calling routine. This is known as "**call by reference**", since we are referencing the variables.
- Here the addresses of actual arguments in the calling function are copied into formal arguments of the called function. Here the formal parameters should be declared as pointer variables to store the address.

**Program**

```
#include<stdio.h>
void swap (int, int);
void main( )
{
    int a=10, b=15;
    swap(&a, &b);
    printf("a=%d,b=%d", a,b);
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

**Output:**      **a = 15 b=10**

**36. Explain how functions return pointer with an example program.****Ans:**

Functions return multiple values using pointers. The return type of function can be a pointer of type int , float ,char etc.

Example :

```
#include<stdio.h>
int * smallest(int * , int *);
void main()
{
    int a,b,*s;
    printf("Enter a,b values ");
    scanf("%d%d",&a,&b);
    s=smallest(&a,&b);
    printf("smallest no. is %d",*s);
}
int * smallest(int *a, int *b)
{
    if(*a<*b)
        return a;
    else
        return b;
}
```

In this example, "return a" implies the address of "a" is returned, not value .So in order to hold the address, the function return type should be pointer.

**37. Explain about various arithmetic operations that can be performed on pointers.****Ans:**

- Like normal variables, pointer variables can be used in expressions.  
Ex:  $x = (*p1) + (*p2);$
- C language allows us to add integers to pointers and to subtract integers from pointers  
**Ex:** If  $p1, p2$  are two pointer variables then operations such as  $p1+4, p2-2, p1-p2$  can be performed.
- Pointers can also be compared using relational operators.  
**Ex:**  $p1 > p2, p1 == p2, p1 != p2$  are valid operations.
- We should not use pointer constants in division or multiplication. Also, two pointers cannot be added.  $p1/p2, p1*p2, p1/3, p1+p2$  are invalid operations.

**Pointer increments and scale factor:-**

Let us assume the address of  $p1$  is 1002. After using  $p1 = p1 + 1$ , the value becomes 1004 but not 1003. Thus when we increment a pointer, its value is increased by length of data type that points to. This length is called scale factor.

**38. Explain in detail how to access a one dimensional array using pointers with an example program?****Ans:****Pointers and Arrays :-**

When an array is declared the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations. The base address is the location of the first element (index 0) of the array. The compiler also defines the array name as a constant pointer to the first element.

Ex:- `static int x[5] = {1,2,3,4,5};`

Suppose the base address of  $x$  is 1000 and assuming that each integer requires two bytes. The five elements will be stored as follows.

elements	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$
value	1	2	3	4	5
address	1000	1002	1004	1006	1008

the name  $x$  is defined as a constant pointer pointing to the first element,  $x[0]$ , and therefore the value  $x$  is 1000, the location where  $x[0]$  is stored. That is

$x = \&x[0] = 1000;$

If we declare p as an integer pointer, then we can make the pointer p to the array x by the following assignment

p=x;      which is equivalent to p=&x[0];

Now we can access every value of x using p++ to move from one element to another. The relationship between p and x is shown below

P+0=&x[0]=1000

p+1=&x[1]=1002

p+2=&x[2]=1004

p+3=&x[3]=1006

p+4=&x[4]=1008

**Note:-** address of an element in an array is calculated by its index and scale factor of the datatype

Address of x[n] = base address + (n\*scale factor of type of x).

Eg:- int x[5]; x=1000;

Address of x[3]= base address of x+ (3\*scale factor of int)

= 1000+(3\*2)

= 1000+6

=1006

Exg:- float avg[20];

avg=2000;

Address of avg [6]=2000+(6\*scale factor of float)

=2000+6\*4

=2000+24

=2024.

Ex:- char str [20];

str =2050;

Address of str[10]=2050+(10\*1)=2050+10=2060.

Note2:- when handling arrays, of using array indexing we can use pointers to access elements.

Like \*(p+3) given the value of x[3]

The pointer accessing method is faster than the array indexing.

### Accessing elements of an array:-

/\*Program to access elements of a one dimensional array\*/

#include<stdio.h>

void main()

{

```

int arr[5]={ 10,20,30,40,50};
int p=0;
printf("\n value@ arr[i] is arr[p] | *(arr+p)| *(p+arr) | p[arr] | located @address \n");
for(p=0;p<5;p++)
{
    printf("\n value of arr[%d] is:",p);
    printf(" %d | ",arr[p]);
    printf(" %d | ",*(arr+p));
    printf(" %d | ",*(p+arr));
    printf(" %d | ",p[arr]);
    printf("address of arr[%d]=%u\n",p,arr[p]);
}
}

```

**output:**

```

value@ arr[i] is arr[p] | *(arr+p)| *(p+arr) | p[arr] | located @address
value of arr[0] is: 10      | 10      | 10      | 10      | address of arr[0]=10
value of arr[1] is: 20      | 20      | 20      | 20      | address of arr[1]=20
value of arr[2] is: 30      | 30      | 30      | 30      | address of arr[2]=30
value of arr[3] is: 40      | 40      | 40      | 40      | address of arr[3]=40
value of arr[4] is: 50      | 50      | 50      | 50      | address of arr[4]=50

```

**39. Explain in detail how to access a two dimensional array using pointers with an example program?****Ans:****Pointers to two-dimensional arrays:-**

Pointer can also be used to manipulate two-dimensional arrays. Just like how  $x[i]$  is represented by  $*(x+i)$  or  $*(p+i)$ , similar, any element in a 2-d array can be represented by the pointer as follows.

$$*((a+i)+j) \text{ or } *((p+i)+j)$$

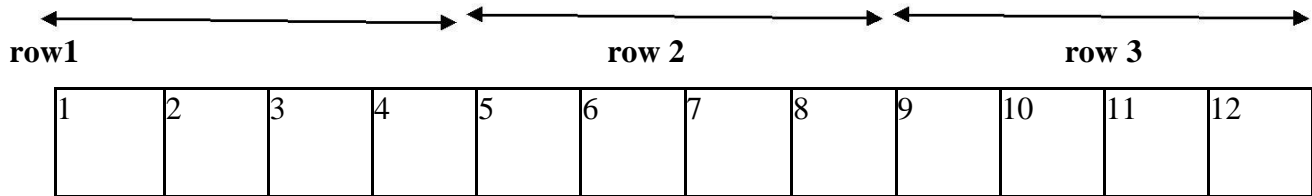
The base address of the array  $a$  is  $\&a[0][0]$  and starting from this address the compiler allocates contiguous space for all the element row – wise .

i.e the first element of the second row is placed immediately after the last element of the first row and so on

Ex:- suppose we declare an array as follows.

```
int a[3][4]={{ 1,2,3,4},{ 5,6,7,8},{ 9,10,11,12}};
```

The elements of a will be stored as shown below.



Base address = &a[0][0]

If we declare p as int pointer with the initial address &a[0][0] then a[i][j] is equivalent to  $*(p + 4 \times i + j)$ . You may notice if we increment „i“ by 1, the p is incremented by 4, the size of each row. Then the element a[2][3] is given by  $*(p + 2 \times 4 + 3) = *(p + 11)$ .

This is the reason why when a two –dimensional array is declared we must specify the size of the each row so that the compiler can determine the correct storage at mapping

### Program:

```
/* accessing elements of 2 d array using pointer*/
#include<stdio.h>
void main()
{
    int z[3][3]={{ 1,2,3},{ 4,5,6},{ 7,8,9}};
    int i, j;
    int *p;
    p=&z[0][0];
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n %d\ti=%d j=%d\t%d",*(p+i*3+j),i,j,*(z[i][j]));
        }
    }
    for(i=0;i<9;i++)
    printf("\n\t%d",*(p+i));
}
```

**output:**



1	i=0	j=0	1
2	i=0	j=1	2
3	i=0	j=2	3
4	i=1	j=0	4
5	i=1	j=1	5
6	i=1	j=2	6
7	i=2	j=0	7
8	i=2	j=1	8
9	i=2	j=2	9

1
2
3
4
5
6
7
8
9_

**40. Explain in detail pointers to character strings with an example program?**

**Ans:**

String is an array of characters terminated with a null character. We can also use pointer to access the individual characters in a string .this is illustrated as below.

Note :- In `c` a constant character string always represents a pointer to that string and the following statement is valid

```
char *name;
name = "delhi";
```

these statements will declare name as a pointer to a character and assign to name the constant character string "Delhi"

This type of declarations is not valid for character string Like:- char name [20];

name ="delhi" ; **//invalid**

**Program:**

```
/* pointers and characters strings*/
//length of string using pointers
#include<stdio.h>
void main()
{
    int length=0;
    char *name,*cptr;
    name="sandhya STGY";
    cptr=name; //assigning one pointer to another
```

```

printf("\n ADDRESS OF POINTERS: name=%u\t cptr=%u",name,cptr);
puts("\n entered string is:");
puts(name);
cptr=name;
while(*cptr!='\0') //is true untill end of str is reached
{
    printf("\n\t%c is @ %u",*cptr,cptr);
    cptr++;          //when while loop ends cptr has the address of '\0' in it
}
length=cptr-name; //end of str minus start of string gives no of chars in between //i.e. length
of str;
printf("\n Length of string is:%d",length);
}

```

**Output:**

```

ADDRESS OF POINTERS: name=170   cptr=170
entered string is:
sandhya STGY

s is @ 170
a is @ 171
n is @ 172
d is @ 173
h is @ 174
y is @ 175
a is @ 176
   is @ 177
S is @ 178
T is @ 179
G is @ 180
Y is @ 181
Length of string is:12

```

**Pointer to table of string:-**

One important use of pointers is in handling of a table of strings. Consider the following array string :

```
char name [3][25];
```

Here name containing 3 names, each with a maximum length of 25 characters and total storage requirement for the name table is 75 bytes

We know that rarely the individual string will be of equal length instead of making each row a fixed number of characters we can make it a pointer to a string of varying length.

Eg:- char \*name[3]={“apple”,“banana” ,“pine apple”};

58

This declaration allocates only 21 bytes sufficient to hold all the character

- To print all the 3 names use the following statement ,  
for (i=0; i<=2; i++)  
printf(“%s/n”, name [i]);
- To access the jth character in the ith name we may write ,  
\*(name [i]+j)

Note:- The character arrays with the rows of varying length are called “ragged arrays” and are better handled pointers.

#### 41. What is an array of pointer? How it can be declared? Explain with an example?

**Ans:**

##### **Array of Pointers(Pointer arrays):-**

We have studied array of different primitive data types such as int, float, char etc. Similarly C supports array of pointers i.e. collection of addresses.

Example :-

```
void main()
{
    int *ap[3];
    int al[3]={ 10,20,30};
    int k; for(k=0;k<3;k++) ap[k]=al+k;
    printf(“\n address element\n”);
    for(k=0;k<3;k++)
    {
        printf(“\t %u”,ap[k]);
        printf(“\t %7d\n”,*(ap[k]));
    }
}
```

##### **Output:**

Address	Element
4060	10
4062	20
4064	30

In the above program , the addresses of elements are stored in an array and thus it represents array of pointers.

A two-dimensional array can be represented using pointer to an array. But, a two-dimensional array can be expressed in terms of array of pointers also. The conventional array definition is,

```
data_type array_name [exp1] [exp2];
```

Using array of pointers, a two-dimensional array can be defined as,

```
data_type *array_name [exp1];
```

Where,

- data\_type refers to the data type of the array.
- array\_name is the name of the array.
- exp1 is the maximum number of elements in the row.

Note that exp2 is not used while defining array of pointers. Consider a two-dimensional vector initialized with 3 rows and 4 columns as shown below,

```
int p[3][4]={ { 10,20,30,40},{ 50,60,70,80},{ 25,35,45,55 } };
```

The elements of the matrix p are stored in memory row-wise (can be stored column-wise also) as shown in Fig.

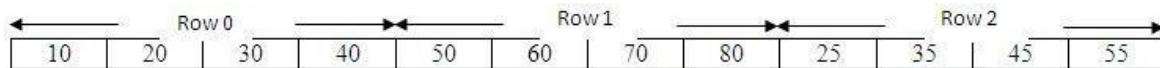


Fig: Two-Dimensional Array

Using array of pointers we can declare p as, `int *p[3];`

Here, p is an array of pointers. p[0] gives the address of the first row, p[1] gives the address of the second row and p[2] gives the address of the third row. Now, p[0]+0 gives the address of the element in 0<sup>th</sup> row and 0<sup>th</sup> column, p[0]+1 gives the address of the elements in 0<sup>th</sup> row and 1<sup>st</sup> column and so on. In general,

- Address of i<sup>th</sup> row is given by a[i]
- Address of an item in i<sup>th</sup> row and j<sup>th</sup> column is given by, p[i]+j.
- The element in i<sup>th</sup> row and j<sup>th</sup> column can be accessed using the indirection operator \* by specifying, \*(p[i]+j)

**42. Write in detail about pointers to functions? Explain with example program.****Ans:****Pointers to functions:-**

A function, like a variable has a type and address location in the memory. It is therefore possible to declare a pointer to a function, which can then be used as an argument in another function.

A pointer to a function can be declared as follows. : `type (*fptr)();`

This tells the compiler that `fptr` is a pointer to a function which returns `type` value the parentheses around `*fptr` is necessary. Because `type *gptr();` would declare `gptr` as a function returning a pointer to `type`. We can make a function pointer to point to a specific function by simply assigning the name of the function to the pointer.

```
Eg: double mul(int,int);  
     double (*p1)();  
     p1=mul();
```

It declares `p1` as a pointer to a function and `mul` as a function and then make `p1` to point to the function `mul`.

To call the function `mul` we now use the pointer `p1` with the list of parameters. i.e `(*p1)(x,y);`  
//function call equivalent to `mul(x,y);`

**Program:**

```
double mul(int ,int); void main()  
{  
    int x,y;  
    double (*p1)();  
    double res;  
    p1=mul;  
    printf("\n enter two numbers:");  
    scanf("%d %d",&x,&y);  
    res=(*p1)(x,y);  
    printf("\n The Product of X=%d and Y=%d is res=%lf",x,y,res);  
}  
double mul(int a,int b)  
{  
    double val; val=a*b; return(val);  
}
```

61

**Output:**

```
using pointers to function  
enter two numbers:22 7
```

```
The Product of X=22 and Y=7 is res=154.000000_
```

**43. What is a pointer to pointer? Write syntax and explain with example program.****Ans:**

It is possible to make a pointer to point to another pointer variable. But the pointer must be of a type that allows it to point to a pointer. A variable which contains the address of a pointer variable is known as pointer to pointer. Its major application is in referring the elements of the two dimensional array.

**Syntax for declaring pointer to pointer,**

```
int *p;           // declaration of a pointer
int **q;          // declaration of pointer to pointer
```

This declaration tells compiler to allocate a memory for the variable „q“ in which address of a pointer variable which points to value of type data type can be stored.

**Syntax for initialization**

```
q = & p;
```

This initialization tells the compiler that now “q” points to the address of a pointer variable “p”.

Accessing the element value we specify, `** q;`

**Example:-**

```
void main( )
{
    int a=10;
    int *p, **ptr; p = &a;
    ptr = &p;
    printf("\n the value of a =%d ", a);
    printf("\n Through pointer p, the value of a =%d ", *p);
    printf("\n Through pointer to pointer q, the value of a =%d ", **ptr);
}
```

**Output:-**

The value of a = 10;

Through pointer p, the value of a = 10;

Through pointer to pointer q, the value of a = 10;

**44. Write a program for pointer to void?**

**Ans:** We can't assign a pointer of one type to other type. But, a pointer of any type can be assigned to pointer to void type and a pointer to void type can be assigned to a pointer of any referenced type. We can use Pointer to void to point either an integer or a real number.

```
void *p;
```

**Program for Pointer to void:-**

```
#include<stdio.h>
int main(void)
{
    void *p;
    int i=7;
    float f=23.5;
    p = &i;
    printf(" i contains %d \n",*((int *)p));
    p=&f;
    printf("f contains %f \n",*((float *)p));
}
```

**Output:** i contains: 7

f contains:23.500000

**45. Write Short notes on enumeration data type**

**Ans:** In C programming, an enumeration type (also called enum) is a data type that consists of integral constants. To define enums, the `enum` keyword is used.

```
enum flag {const1, const2, ..., constN};
```

By default, `const1` is 0, `const2` is 1 and so on. You can change default values of enum elements during declaration (if necessary).

```
// Changing default values of enum constants
enum suit {
    club = 0,
    diamonds = 10,
    hearts = 20,
    spades = 3,};
```

### Enumerated Type Declaration

When you define an enum type, the blueprint for the variable is created. Here's how you can create variables of enum types.

```
enum boolean {false, true};  
enum boolean check; // declaring an enum variable
```

Here, a variable `check` of the type `enum boolean` is created.

You can also declare enum variables like this.

```
enum boolean {false, true} check;
```

Here, the value of `false` is equal to 0 and the value of `true` is equal to 1.

### Example: Enumeration Type

```
#include <stdio.h>  
  
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};  
  
int main()  
{  
    // creating today variable of enum week type  
    enum week today;  
    today = Wednesday;  
    printf("Day %d",today+1);  
    return 0;  
}
```

### Output

Day 4