The fundamental data types, namely *char, int, float,* and *double* are used to store only one value at any given time. Hence these fundamental data types can handle limited amounts of data.

In some cases we need to handle large volume of data in terms of reading, processing and printing. To process such large amounts of data, we need a powerful data type that would facilitate efficient storing, accessing and manipulation of data items. C supports a derived data type known as ***array*** that can be used for such applications.

Arrays

1. An array is a fixed size sequenced collection of elements of the same data type.
2. It is simply grouping of like type data such as list of numbers, list of names etc.

Types of Arrays

We can use arrays to represent not only simple lists of values but also tables of data in two or three or more dimensions.

1. One – dimensional arrays(1-D)
2. Two – dimensional arrays (2-D)
3. Multidimensional arrays (n-D)

Some examples where arrays can be used are

- List of temperatures recorded every hour in a day, or a month, or a year.
- List of employees in an organization.
- List of products and their cost sold by a store.
- Test scores of a class of students
- List of customers and their telephone numbers.

Array

An array is collection of homogeneous elements that are represented under a single variable name.

(Or)

An array is collection of same data type elements in a single entity.

- It allocates sequential memory locations.
- Individual values are called as elements.

Declaration of One-Dimensional Array

Like any other variables, arrays must be declared before they are used. The general form of array declaration is

Syntax

datatype array_name[size];

- The *data type* specifies the type of element that will be contained in the array, such as *int*, *float*, or *char* or any valid *data type*.
- The *array name* specifies the name of the array.
- The *size* indicates the maximum number of elements that can be stored inside the array.

The *size* of array should be a *constant* value.

For example: *int marks[10];*

The above statement declares a *marks* variable to be an array containing 10 elements. In C, the array *index* (also known as *subscript*) start from *zero.* i.e. The first element will be stored.

in *marks[0]*, second element in *marks[1]*, and so on. Therefore, the last element, that is the 10th element, will be stored in *marks[9]*.

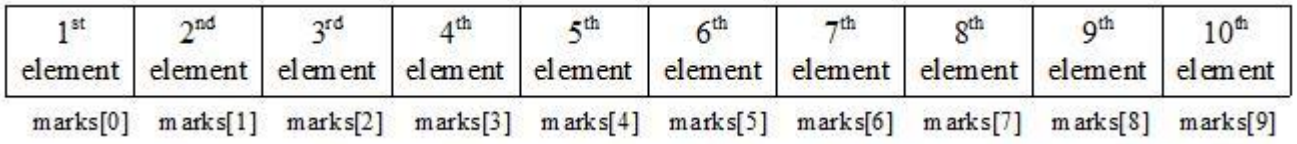


Fig. memory representation of an array of elements **Examples**

float temp[24]; //floating-point array

Declare the group as an array to contain a maximum of 24 real constants.

char name[10]; //character array

Declare the name as a character array (string) variable that can hold a maximum of 10 characters.

C array indices start from 0. So for an array with N elements, the index that last element is N-1

Valid Statements

```
a = marks[0] + 10
marks[4] = marks[0] + marks[2];
marks[2] = x[5] + y[10]
value[6] = marks[i] * 3;
```

Example 1: A C program that prints bytes reserved for various types of data and space required for storing them in memory using array.

```
#include<stdio.h>
int main()
{
   char c[10];
   int i[10];
   float f[10];
   double d[10];
   printf("\nThe type char requires %d byte",sizeof(char)); printf("\nThe type int requires %d
   bytes",sizeof(int)); printf("\nThe type float requires %d bytes",sizeof(float)); printf("\nThe
   type double requires %d bytes",sizeof(double));
   printf("\n%d memory locations are reserved for 10 character elements", sizeof(c));
   printf("\n%d memory locations are reserved for 10 integer elements", sizeof(i)); printf("\n%d
   memory locations are reserved for 10 float elements", sizeof(f)); printf("\n%d memory
   locations are reserved for 10 double elements", sizeof(d)); return 0;
}
```

Output

The type *char* requires 1 byte

The type *int* requires 2 bytes
The type *float* requires 4 bytes
The type *double* requires 8 bytes
10 memory locations are reserved for 10 character elements
20 memory locations are reserved for 10 integer elements
40 memory locations are reserved for 10 float elements
80 memory locations are reserved for 10 double elements

Accessing elements the array

To access all the elements from an array, we must use a loop. That is, we can access all the elements of an array by varying the value of the subscript into the array. But note that the subscript must be an integral value or an expression that evaluates to an integral value.

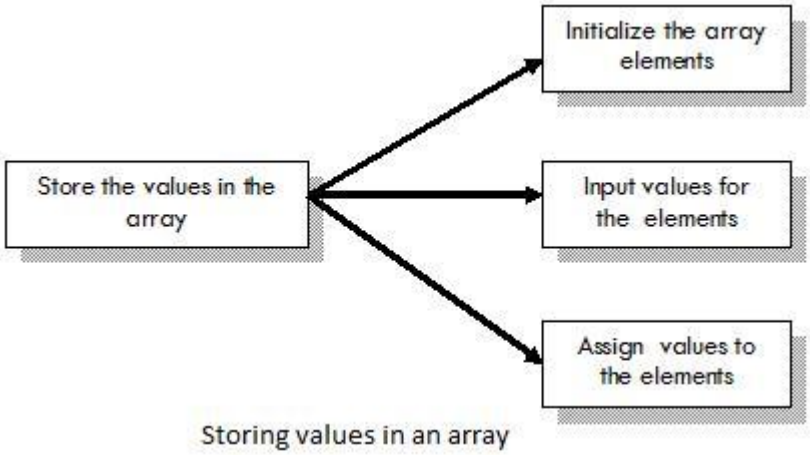
For example

```
int i ,marks[10];
for (i=0; i<10; i++)
        printf("%d",marks[i]);
```

Storing Values in Array

When we declare an array, we are just allocating space for its elements; no values are stored in the array. There are three ways to store values in an array.

- o Initialize the array elements during declaration.
- p Input values for individual elements from the keyboard.
- q Assign values to individual elements using assignment (=) operator.



Storing values in an array

Initializing Arrays during Declaration

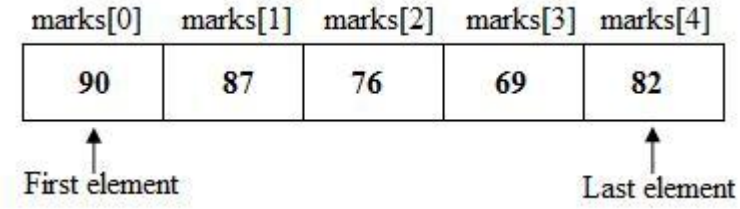
When an array is initialized, we need to provide a value for every element in the array. Arrays are initialized by writing:

type array_name[size]={list_of_values};

The *values* are written within curly brackets and every value is separated by a *comma*.

For example

int marks[5]={90,87,76,69,82};



While initializing the array at the time of declaration, the programmer may omit the size of the array. For example,

int marks[]= {98, 97, 90};

Input values from keyboard

An array can be filled by inputting values from the keyboard. In this method, a *while* or *for* loop is executed to input the value for each element of the array. For example:

```
int i ,marks[10];
for (i=0; i<10; i++)
        scanf("%d",&marks[i]);
```

Assigning Values to Individual Elements

The third way is to assign values to individual elements of the array by using the *assignment operator*. Any value that evaluates to the data type as that of the array can be assigned to the individual array element. A simple assignment statement can be written as

marks[3] = 100;

Here, 100 is assigned to the fourth element of the array which is specified as *marks[3]*.

Example 2: Write a C Program to read and display *N* numbers using an array.

```
#include <stdio.h>
int main()
{
        int i, n, arr[20];
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d",&arr[i]);
        }
        printf("\n The array elements are ");
        for(i=0;i<n;i++)
        printf("\t %d", arr[i]);
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The array elements are 1 2 3 4 5
```

Example 3: Write a C Program to print the maximum and minimum element in the array.

```
#include <stdio.h>
int main()
{
        int i, n, a[20], max, min;
        printf("Enter the number of elements in the array : "); scanf("%d", &n);
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        {
                printf("a[%d] = ",i);
                scanf("%d",&a[i]);
        }
        max = min = a[0];
        for(i=1;i<n;i++)
        {
                if(a[i]<min)
                        min = a[i];
                if(a[i]>max)
                        max = a[i];
        }
        printf("\n The smallest element is : %d\n", min);
        printf("\n The largest element is : %d\n", max);
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The smallest element is : 1
The largest element is : 5
```

Searching the Array elements

Searching means to find whether a particular value is present in an array or not. If the value is present in the array, then searching is said to be *successful* and the searching process gives the location of that value in the array. if the value is not present in the array, the searching is said to be *unsuccessful*. There are two popular methods for searching the array elements:

Linear search
Binary search

Linear Search

Linear search, also called as *sequential search*, is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found. It is mostly used to search an unordered list of elements. For example, if an array *a[]* is declared and initialized as,

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

and the value to be searched is *VAL = 7*, then searching means to find whether the value '7' is present in the array or not. If yes, then it returns the position of its occurrence. Here, POS = 3 (index starting from 0).

Example 4: Write a C Program to search an element in an array using the linear search technique.

```
#include <stdio.h>
int main()
{
  int a[100],n,i,key,flag=0;
  printf("Enter number of elements:");
  scanf("%d", &n);
  printf("Enter elements\n");
  /* Read array elements */
  for(i=0;i<n;i++)
  {
        printf("Enter a[%d]=",i);
        scanf("%d", &a[i]);
  }
  printf("Enter an element to be searched:");
  scanf("%d", &key);
  /* linear search starts here */
  for(i=0;i<n;i++)
  {
    if(key==a[i])
    {
      printf("%d is found at position %d\n", key, i);
          flag=1;
      break;
    }
  }
  if(flag==0)
        printf("%d is not found\n",key);
  return 0;
}
```

Output

Enter number of elements: 5

```
Enter elements
Enter a[0] = 14
Enter a[1] = 5
Enter a[2] = 23
Enter a[3] = 9
Enter a[4] = 15
Enter an element to be searched: 9
9 is found at position 3
```

Two Dimensional (2D) Arrays

There could be situations where a table of values will have to be stored. Consider a student table with marks in 3 subjects.

Student	Maths	Physics	Chemistry
Student #1	89	77	84
Student #2	98	89	80
Student #3	75	70	82

- The above table contains a total of 9 values.
- We can think this table as a matrix consisting of 3 rows and 3 columns.
- Each row represents marks of student # 1 in all (different) subjects.
- Each column represents the subject wise marks of all students.
- In mathematics, we represent a particular value in a matrix by using two subscripts such as *Vij*.
- Here *V* denotes the entire matrix, *Vij* refers to the value in i^{th} row and j^{th} column.

In the above table *V23* refers to the value "80". C allows us to define such tables of items by using two-dimensional arrays.

Definition

A list of items can be represented with one variable name using two subscripts and such a variable is called a two – subscripted variable or a two – dimensional *(2D)* array.

5.4.1 Declaring 2-D Array

A two-dimensional array is declared as:

data_type array_name[row_size][column_size];

For example, if we want to store the marks obtained by three students in three different subjects, we can declare a two dimensional array as:

int marks[3][3];

The pictorial form of a two-dimensional array in memory is shown in Figure.

	Col 0	Col 1	Col 2
	[0] [0]	[0] [1]	[0] [2]
Row 0	89	77	84
	[1] [0]	[1] [1]	[1] [2]
Row 1	98	89	80
	[2][0]	[2] [1]	[2] [2]
Row 2	75	70	82

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 curly braces.

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

This initializes the elements of first row to zero and the second row to one.

- This initialization can also be done row by row. The above statement can be equivalently written as

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

  Commas are required after each curly brace that closes of a **row**, except in case of last **row**.
- If the values are missing in an initializer, they are automatically set to zero.

  Ex: *int table [2] [3] = {*

  {1,1}, \\ 1 1 0

  {2} \\ 2 0 0

  };
- When all the elements are to be initialized to zero, the following short-cut method may be used.

  int m[3][5] = { {0}, {0}, {0}};

- The first element of each row is explicitly initialized to zero while the other elements are automatically initialized to zero.
- The following statement will also achieve the same result

  int m[3][5] = { 0 };

Accessing the Elements of Two-dimensional Arrays

Since the *2D* array contains two subscripts, we will use two for loops to scan the elements. The first *for* loop will scan each row in the 2D array and the second *for* loop will scan individual columns for every row in the array.

Example : Write a program to print the elements of a 2D array.

```
#include <stdio.h>
int main()
{
        int a[3][3] = {12, 34, 56,32,89,23,44,67,99};
        int i, j;
        for(i=0;i<3;i++)
        {
                                        for(j=0;j<3;j++)
                        printf("%d\t", a[i][j]);
                printf("\n");
        }
        return 0;
```

Output

```
12      34      56
32      89      23

44      67      99
```

```
}
```

Example : Write a C program to perform addition or subtraction of two matrices

```
#include <stdio.h>
int main()
{
   int a[10][10],b[10][10],c[10][10];
   int rows,cols,i,j;
   printf("Enter number of rows and columns (between 1 and 10):");
   scanf("%d%d",&rows,&cols);
   /* read the elements of first matrix */
   printf("Enter elements of first matrix\n");
   for(i=0;i<rows;i++)
   {
       for(j=0;j<cols;j++)
            scanf("%d",&a[i][j]);
   }
/* read the elements of second matrix */
   printf("Enter elements of second matrix\n");
   for(i=0;i<rows;i++)
   {
          for(j=0;j<cols;j++)
              scanf("%d",&b[i][j]);
     }
  /* Display A and B matrices */
   printf("The Matrix A\n");
   for(i=0;i<rows;i++)
   {
     for(j=0;j<cols;j++)
         printf("%3d",a[i][j]);
     printf("\n");
   }
   printf("The Matrix B\n");
   for(i=0;i<rows;i++)
   {
     for(j=0;j<cols;j++)
         printf("%3d",b[i][j]);
     printf("\n");
   }
          /*Add two matrices and print resultant matrix */
  printf("The resultant matrix is\n");
  for(i=0;i<rows;i++)
  {
     for(j=0;j<cols;j++)
     {
             c[i][j]=a[i][j]+b[i][j];
             printf("%3d",c[i][j]);
```

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

  }
  return 0;
}
```

Output

```
Enter number of rows and columns (between 1 and 10): 3 3
Enter elements of first matrix: 1 2 3 4 5 6 7 8 9
Enter elements of second matrix: 1 2 3 4 5 6 7 8 9
The matrix A
1 2 3
4 5 6
7 8 9
The matrix B
1 2 3
4 5 6
7 8 9
The resultant matrix is
2    4    6
8    10 12
14   16  18
```

Multi – Dimensional Array

A list of items can be represented with one variable name using more than two subscripts and such a variable is called Multi – dimensional array.

Three Dimensional (3D) Array

A list of items can be represented with one variable name using three subscripts and such a variable is called Three – dimensional (3D) array.

Syntax

data type array_name [size_of_2d_matrix][row_ size][column_ size];

Initializing 3D Array

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

int table[2][2][3] = {0,0,0,1,1,1,6,6,6,7,7,7};

This initializes the elements of first two dimensional (matrix) first row to zero's and the second row to one's and second matrix elements are first row to six's and the second row to seven's.

- This initialization is done row by row.
- The above statement can be equivalently written as
  int a[2][3] = {{{0,0,0},{1,1,1}},{{0,0,0},{1,1,1}}}

we can also initialize a two – dimensional array in the form of a matrix as shown. *int a[2][3] = {*

```
{
        {0,0,0},
        {1,1,1}
},
{
        {6,6,6},
        {7,7,7}
}
};
```

Strings

The string in C programming language is actually a one-dimensional array of characters which is terminated by a null character '\0'. Since string is an array, the declaration of a string is the same as declaring a char array.

char string1[30];
char str2[7] = "String";

The following declaration creates string named *"str2"* and initialized with value *"String".* To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word**.**

Declaration and Initialization of Strings

The following declaration and initialization create a string consisting of the word "Hello".

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

Another way of Initialization is (shortcut for initializing string)

char greeting[] = "Hello";

Note: The C compiler automatically places the '\0' at the end of the string when it initializes the array.

The terminating null ('\0') is important, because it is the only way the functions that work with a string can know where the string ends.

Memory Representation of String

char greeting[] = "Hello";

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	5000	5001	5002	5003	5004	5005

/* Program to demonstrate printing of a string */

```
int main( )
{
        char name[ ] = “Vijayanand" ;
        int i = 0 ;
        while ( i <= 9 )
        {
                printf ( "%c", name[i] ) ;
                i++ ;
        }
   return 0;
}
```

And here is the output...

Vijayanand

Following program illustrates printing string using ‘\0’.

```
int main( )
{
        char name[ ] = “Vijayanand" ;
        int i = 0 ;
        while ( name[i] != ‘\0’ )
        {
                printf ( "%c", name[i] ) ;
                i++ ;
        }
         return 0;
}
```

And here is the output...

Vijayanand

Standard Library String Functions

With every C compiler a large set of useful string handling library functions are provided. For using these functions, we need to include the header file ***string.h***

Function	**Use**

strlen()	Finds length of a string
strlwr()	Converts a string to lowercase
strupr()	Converts a string to uppercase
strcat()	Appends one string at the end of another
strcpy()	Copies a string into another
strcmp()	Compares two strings
strchr()	Finds first occurrence of a given character in a string
strstr()	Finds first occurrence of a given string in another string
strrev()	Reverses the given string

strlen() function

This function counts the number of characters present in a string. Syntax for strlen() function is given below:

```
size_t strlen(const char *str);
```

The function takes a single argument, i.e, the string variable whose length is to be found, and returns the length of the string passed

While calculating the length it doesn't count '\0'.

Example : C program that illustrates the usage of *strlen()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char str[ ] = "Henry" ;
        int len1, len2 ;
        len1 = strlen ( str ) ;
        len2 = strlen ( "Humpty Dumpty" ) ;
        printf ( "\nThe string %s length is %d", str, len1 ) ;
        printf ( "\nThe string %s length is %d\n", "Humpty Dumpty", len2 ) ; return 0;
}
```

Output

The string *Henry* length is 5
The string *Humpty Dumpty* length is 13

strcpy( ) function

This function copies the contents of one string into another. Syntax for strcpy() function is given below.

```
char * strcpy ( char * destination, const char * source );
```

- **Example**

  strcpy (str1, str2) – It copies contents of str2 into str1. strcpy (str2, str1) – It copies contents of str1 into str2.
- If destination string length is less than source string, entire source string value won't be copied into destination string.
- For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won't be copied and will be truncated..

Example : C program that illustrates the usage of *strcpy()* function.

```
#include<stdio.h>
#include<string.h>
int main()
{
        char source[ ] = "Sayonara" ;
        char target[20]= "" ;
        strcpy (destination, source ) ;
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", destination ) ; return 0;
}
```

Output

Source string = Sayonara

Destinnation string = Sayonara

strcat( ) function

It combines two strings. It concatenates the source string at the end of the destination string. Syntax for strcat( ) function is given below.

```
char * strcat ( char * destination, const char * source );
```

For example, "Bombay" and "Nagpur" on concatenation would result a new string "BombayNagpur".

Example: C program that illustrates the usage of *strcat()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char source[ ] ="Students!" ;
        char target[30] = "Hello" ;
        strcat ( target, source ) ;
```

```
        printf ( "\nSource string = %s", source ) ;
        printf ( "\nDestination string = %s", target ) ;
        return 0;
}
```

Output
Source string = Students!
Destination string = HelloStudents!

strcmp( ) function

It compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first.

If the two strings are identical, **strcmp( )** returns a value zero. If they're not identical, it returns the *numeric* difference between the ASCII values of the first non-matching pairs of characters.
Syntax for strcmp( ) function is given below.

```
int strcmp ( const char * str1, const char * str2
```

Return Value from strcmp()

Return Value	Description
0	if both strings are identical (equal)
<0	if the ASCII value of first unmatched character is less than second.
>0	if the ASCII value of first unmatched character is greater than second.

Note: *strcmp( )* function is case sensitive. i.e, "A" and "a" are treated as different characters.
Example 15: C program that illustrates the usage of *strcmp()* function.

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char string1[ ] = "Jerry" ;
        char string2[ ] = "Ferry" ; int i, j, k ;
        i = strcmp ( string1, "Jerry" ) ;
        j = strcmp ( string1, string2 ) ;
        k = strcmp ( string1, "Jerry boy" ) ;
        printf ( "\n%d %d %d", i, j, k ) ;

}
```

Output
0 4 -32

Example 16: Program for checking string's palindrome property.

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

```
int main()
{
   char str1[25],str2[25];
   int d=0;
   printf("\nEnter a string:");
   gets(str1);
   strcpy(str2,str1);
   strrev(str1);
   d= strcmp(str1,str2);
   if(d==0)
      printf("\n%s is pallindrome",str2); else
printf("\n%s is not a pallindrome",str2); return 0;

}
```

Output:
Enter a string: madam
madam is palindrome

Some other palindrome strings are:
civic, dad, malayalam, mom,wow etc.

getchar() and putchar() functions

The *getchar()* function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one characters.

The *putchar()* function prints the character passed to it on the screen and returns the same character. This function puts only single character at a time. In case you want to display more than one characters, use putchar() method in the loop. #include <stdio.h>

```
int main( )
{
        int ch;
        printf("Enter a character:");
        ch=getchar();
        putchar(ch);
        return 0;
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

5.9 puts() and gets() functions

The *gets()* function reads a line or multiword string from stdin into the buffer pointed to by *s* until either a terminating newline or EOF (end of file).

The *puts()* function writes the string *s* and a trailing newline to stdout.

```
#include<stdio.h>
int main()
```

```
{
        char str[100];
        printf("Enter a string:");
        gets( str );
        puts("Hello!");
        puts(str);
        return 0;
}
```

Output

Enter a string: Vijayanand Ch
Hello! Vijayanand Ch

Difference between scanf() and gets()

S.No	scanf()	puts()
1	It reads single word strings	It reads multi-word strings
2	It stops reading characters when it encounters a whitespace	It stops reading characters when it encounters a newline or EOF
3	Example: Vijay	Example: Vijayanand ch

Structure.

A simple variable can store one value at a time. An array can store a number of variables of the same type. For example, marks obtained by a student in five subjects can be stored in an array marks[5]. marks[0], marks[1], etc, will store student information like the marks obtained in various subjects. Suppose we want to store student information in array name, htno, address, marks obtained. We cannot store this information in an array because name, htno, address are of character type and marks obtained is of integer type. So arrays are used to store homogeneous data. To store heterogeneous data elements in a single group, C language provides a facility called the ***structure***.

Definition of structure

A structure is a collection of variables of different types that are logically grouped together and referred under a single name.

- Structure is a user-defined data type.
- User-defined data type also called as derived data type why because we derived it from the primary/basic data types.

In C language, a structure can be defined as follows:

```
struct structure_name
{
```

```
        data_type member_variable1;
        data_type member_variable2;
            … …
        data_type member_variableN;
    };
```

The variables declared inside the structure are known as *members* of the structure.

For Example

```
struct student
{
    char name[20];
    char ht_no[10];
    char gender;
    float marks;
    long int phone_no;
};
```

Points remember

- The members of the structure may be any of the common data type, pointers, arrays or even the other structures.
- Member names with in a structure must be different.
- Structure definition starts with the open brace(**{**) and ends with closing brace(**}**) followed by a semicolon.
- The compiler does not reserve memory any memory when structure is defined. We have to define the variable to store its information.

Structure variable

As stated earlier, the compiler does not reserve memory any memory when structure is defined. To store members of structures in memory, we have to define the structure variables. The structure variables may be declared in the following ways:

- In the structure declaration
- Using the structure tag

In the structure declaration

The structure variable can be declared after the closing brace. Following example shows this.

```
struct date
{
    int dat;
    int month;
    int year;
}dob, doj;
```

Here date is the structure tag, while *dob* and *doj* are variables type date.

Using the structure tag

The variables of structure may also be declared separately by using the structure tag as shown below:

```
struct date
{
    int dat;
    int month;
    int year;
};

struct date dob,doj;
```

Initialization of structure

We can initialize the values to the members of the structure as:

```
struct structure_name structure_variable={value1, value2, … , valueN};
```

1. There is a one-to-one correspondence between the members and their initializing values.
2. C does not allow the initialization of individual structure members within the structure definition template.

```
struct date
{
    int dat;
    int month;
    int year;
};

struct date dob={25,12,1998};
```

Following example shows how to initialize structure in the above method.

```
#include<stdio.h>
struct student
{
    char name[30];
    int htno;
    char gender;
    int marks;
    char address[30];
};
int main()
{
    struct student st={"XYZ",581,'M',79.5,"Guntur"};
    printf("Student Name:%s\n",st.name); printf("Roll
    No:%d\n",st.htno); printf("Gender:%c\n",st.gender);
    printf("Marks obtained:%d\n",st.marks);
    printf("Address:%s\n",st.address);
    return 0;
}
```

Output

```
Student Name: XYZ
Roll No: 581
```

Gender: M
Marks obtained:79.5
Address: Guntur

Accessing members of structure:

For accessing any member of the structure, we use dot (.) operator or *period* operator.

Syntax:

structure_variable**.**membername

Here, *structure_variable* refers to the name of a *structure* type variable and *member* refers to the name of a member within the structure.

Following example shows how to access structure members

```
#include<stdio.h>
struct date
{
    int day;
    int month;
    int year;
};

int main()
{
        struct date dob;
        printf("Enter your birthday details\n");
        printf("Enter the day:");
        scanf("%d",&dob.day);
        printf("Enter the month:");
        scanf("%d",&dob.month);
        printf("Enter the year:");
        scanf("%d",&dob.year);
        printf("Your date of birth is:");
        printf("%d-%d-%d\n",dob.day,dob.month,dob.year); return
        0;
}
```

Output

Enter your birthday details
Enter the day:12
Enter the month:7
Enter the year:1998
Your date of birth is: 12-7-1998

Nested Structure:

We can take any structure as a member of structure. i.e. called structure with in structure or nested structure.

For example:

```
struct
{
        member 1;
        member 2;
        … …
        … …
        struct
        {
                member 1;
                member 2;
        }s_var2;
        … …
        member m;
}s_var1;
```

For accessing the member 1 of inner structure we write as:

```
s_var1.s_var2.member1;
```

Following example illustrates the nested structure

```
#include<stdio.h>
struct student
{
        char name[30];
        int htno;
        struct date
        {
                int day;
                int month;
                int year;
        }dob;
        char address[10];
};
int main()
{
        struct student st;
        printf("Enter student details\n");
        printf("Enter student name:"); scanf("%s",st.name);
        printf("Enter rollno:"); scanf("%d",&st.htno); printf("Enter
        address:"); scanf("%s",st.address); printf("Enter the day:");
        scanf("%d",&st.dob.day); printf("Enter the month:");
        scanf("%d",&st.dob.month); printf("Enter the year:");
        scanf("%d",&st.dob.year); printf("Student Details\n----------
        ------\n"); printf("Student Name:%s\n",st.name);
        printf("Roll No:%d\n",st.htno);
```

```
        printf("Date of birth:%d-%d-%d\n",st.dob.day,st.dob.month,st.dob.year);
        printf("Address:%s\n",st.address);
        return 0;
}
```

Output

```
Enter student details
Enter student name: Vijayanand
Enter roll no: 511
Enter address: Guntur
Enter the day:12
Enter the month:7
Enter the year:1998
Student Details
----------
Student Name: Vijay
Roll No: 511
Date of birth is: 12-7-1998
Address: Guntur
```

Self-referential structure:

A self-referential is one which contains a pointer to its own type. This type of structure is also known as linked list. For example

```
struct node
{
        int data;
        struct node *nextptr;
};
```

Defines a data type, struct node. A structure type struct node has two members-integer member data and pointer member nextptr. Member nextptr points to a structure of type struct node – a structure of the same type as one being declared here, hence the term “self-referential structure”. Member nextptr referred to as a link i.e. nextptr can be used to to tie a structure of type struct node to another structure of the same type. select referential structures can be linked together to forn a usual data structures such as lists etc.

```
#include<stdio.h>
struct student
{
        char name[30];
        int age;
        char address[20];
        struct student *next;
};
```

```
int main()
{
    struct student st1={"Raja",20,"Guntur"};
    struct student st2={"Raghu",21,"Narasarao pet "};
    struct student st3={"Nagaraju",22,"Sattenapali"};
    st1.next = &st2;
    st2.next = &st3;
    st3.next = NULL;
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st1.name,st1.age,st1.address);
    printf("Student 1 stored at %x \n",st1.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st2.name,st2.age,st2.address);
    printf("Student 2 stored at %x \n",st2.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st3.name,st3.age,st3.address);
    printf("Student 3 stored at %x \n",st3.next); return 0;
}
```

Output

```
Student Name:Raja       Age:20      Address: Guntur
Student 1 stored at 88f0
Student Name:Raghu      Age:21      Address: Narasarao pet
Student 2 stored at 8854
Student Name:Nagaraju   Age:22      Address: Sattenapalli
Student 3 stored at 0
```

typedef in C:

In C programming language, **typedef** is a keyword used to create alias name for the existing datatypes. Using **typedef** keyword we can create a temporary name to the system defined datatypes like int, float, char and double. we use that temporary name to create a variable. The general syntax of typedef is as follows...

Synatax: **typedef <existing-datatype> <alias-name>**

typedef with primitive datatypes

typedef int Number

In the above example, **Number** is defined as alias name for integer datatype. So, we can use **Number** to declare integer variables.

Example Program to illustrate typedef in C.

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

```
typedef int Number;

void main(){

  Number a,b,c;    // Here a,b,&c are integer type of variables.

  clrscr() ;
  printf("Enter any two integer numbers: ") ;
  scanf("%d%d", &a,&b) ;

  c = a + b;

  printf("Sum = %d", c) ;
}
```

typedef with Arrays:

In C programming language, typedef is also used with arrays. Consider the following example program to understand how typedef is used with arrays.

Example Program to illustrate typedef with arrays in C.

```
#include<stdio.h>
#include<conio.h>
void main(){
  typedef int Array[5];    // Here Array acts like an integer array type of size 5.
  Array list = {10,20,30,40,50};   // List is an array of integer type with size 5.
  int i;
  clrscr() ;
  printf("List elements are : \n") ;
 for(i=0; i<5; i++)
      printf("%d\t", list[i]) ;
}
```

typedef with user defined datatypes like structures, unions etc.,

In C programming language, typedef is also used with structures and unions. Consider the following example program to understand how typedef is used with structures.

Example Program to illustrate typedef with structures in C.

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

typedef struct student
{
   char stud_name[50];
   int stud_rollNo;
}stud;

void main(){
  stud s1;
  clrscr() ;

  printf("Enter the student name: ") ;
  scanf("%s", s1.stud_name);
  printf("Enter the student Roll Number: ");
  scanf("%d", &s1.stud_rollNo);

  printf("\nStudent Information\n");
  printf("Name - %s\nHallticket Number - %d", s1.stud_name, s1.stud_rollNo);
}
```

Enumerated Types (enum) in C:

In C programming language, an enumeration is used to create user-defined datatypes. Using enumeration, integral constants are assigned with names and we use these names in the program. Using names in programming makes it more readable and easy to maintain.

Enumeration is the process of creating user defined datatype by assigning names to integral constants

Example Program for enum with default values

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

enum day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday} ;

void main(){

  enum day today;

  today = tuesday ;

  printf("\ntoday = %d ", today) ;

}
```

Example Program for enum with changed integral constant values

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

enum day { Monday = 1, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday} ;

void main(){

  enum day today;

  today = tuesday ;

  printf("\ntoday = %d ", today) ;

}
```

In the above example program, the integral constant value starts with '1' instead of '0'. Here, tuesday value is displayed as '2'.

Example Program for enum with changed integral constant values

```
#include<stdio.h>
#include<conio.h>
enum day {Monday, Tuesday, Wednesday, Thursday = 10, Friday, Saturday, Sunday} ;
void main(){
  enum day today;
   today = tuesday ;
  printf("\ntoday = %d ", today) ;
    today = saturday ;
    printf("\ntoday = %d ", today) ;
  }
```

In the above example program a user defined datatype "**day** is created seven values, monday as integral constant '0', tuesday as integral constant '1', wednesday as integral constant '2', thursday as integral constant '10', friday as integral constant '11', saturday as integral constant '12' and sunday as integral constant '13'.

Union

A *Union* is a user defined data type like structure. The union groups logically related variables into a single unit.

- In structure each member has its own memory location where as the members of union has the same memory location.
- We can assign values to only one member at a time, so assigning value to another member that time has no meaning.

When a union is declared, compiler allocates memory locations to hold largest data type member in the union. So, union is used to save memory. Union is useful when it is not necessary to assign the values to all the members of the union at a time.

Union can be declared as

```
union
{
        member 1;
        member 2;
        … …
        … …
        member N;
};
```

Following example shows how to declare union and accessing the members of union

```
#include<stdio.h>
union student
{
        char name[30];
        int age;
};
int main()
{
        union student st;
        clrscr();
        printf("Enter student name:");
        scanf("%s",st.name);
        printf("Student Name:%s, age= %d\n",st.name,st.age);
        st.age=20;
        printf("Student Name:%s, age= %d\n",st.name,st.age);
        getch();
        return 0;
}
```

Output

Enter student name: Vijay
Student Name: Vijay, age= 26966
Student Name: ¶ , age= 20

Comparison between structure and union

S.No	Structure	Union
1.	Stores heterogeneous data	Stores heterogeneous data
2.	Members are stored separately in memory locations.	Members are stored in same memory location.
3.	In structures all the members are available.	In union only one member is available at a time.
4.	Occupies more memory when compared to union	Occupies less memory when compared with structure.
5.	Size of the structure is the total memory space required by its members	Size of the union is the size of the largest data type member in the union.

6.9 Comparison between array and structure

S.No	Array	Structure
1.	Stores homogeneous data	Stores heterogeneous data
2.	Two arrays of same type cannot be assigned to one to one	Structures of same type can be assigned.
3.	Arrays can be initialized	Structures cannot be initialized
4.	Array is a combination of elements	Structure is a combination of members
5.	Multidimensional arrays are possible	No in case of structures
6.	No operators are required to access	Operators like „.“ or „->“ are required to

	elements of an array	access members of a structure.
7.	An element in the array referenced by specifying array name and its position in the array. For example: a[5]	Members in a structure will be referenced as structurename.membername
8.	C treats array names as pointers	Structure name is not treated as pointer variable.
9.	When an array name is passed as argument to function, it is call by reference.	When an structure name is passed as argument to function, it is call by value.