Arrays

Arrays

- one of the C's most essential data structures
- Arrays are data structures consisting of related data items of the same type
- it is a group of memory locations related by the fact that they all have the same name and same type

Why Arrays

```
main( )
{
    int x ;
    x = 5 ;
    x = 10 ;
    printf ( "\nx = %d", x ) ;
}
```

- No doubt, this program will print the value of **x** as 10
- Because when a value 10 is assigned to **x**, the earlier value of **x**, i.e. 5, is lost
- ordinary variables (the ones which we have used so far) are capable of holding only one value at a time

Why Arrays

- However, there are situations in which we would want to store more than one value at a time in a single variable
- suppose we wish to arrange the percentage marks obtained by 100 students in ascending order. In such a case we have two options to store these marks in memory:

1. Construct 100 variables, each variable containing one student's marks.
2. Construct one variable capable of storing or holding all the hundred values.

Why Arrays

- the second alternative is better
- it would be much easier to handle one variable than handling 100 different variables
- Moreover, there are certain logics that cannot be dealt with, without the use of an array

One Dimensional Array

- a one dimensional array is a list of variables that are all of the same type and accessed through a common name
- An individual element in an array is called *array element*
- Array is helpful to handle a group of similar types of data

One Dimensional Array

- To declare a one dimensional array, we use:

data_type array_name [size];

- data_type is a valid C data type
- array_name is the name of that array and
- size specifies the number of elements in the array

One Dimensional Array

```
int my_array[20];
```

- Declares an array name my_array of type integer that contains 20 elements

One Dimensional Array

- An array element is accessed by indexing the array using the number of element
- all arrays begin at zero
- if you want to access the first element in an array, use zero for the index
- To index an array, specify the index of the element you want inside square brackets

One Dimensional Array

- The second element of my_array will be:

my_array [1]

One Dimensional Array

- C stores one dimensional array in one contiguous memory location with first element at the lower address
- an array named *a* of 10 elements can occupy the memory as follows-

Index	Value
a [0]	-45
a [1]	10
a [2]	32
a [3]	100
a [4]	9

a [5]	9
a [6]	50
a [7]	100
a [8]	-9
a [9]	12

One Dimensional Array

```
main(){
        int a[10], i;
        for (i=0;i<10;i++)
              a [i] = 0;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
              printf ("%d \t%d", i, a [i]  );
}
```

- a program that declares an array of 10 elements and initializes every element of that array with 0

One Dimensional Array

```
main(){
        int a[10]={1,2,3,4,5,6,7,8,9,10},
        int i;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
                printf ("%d \t%d", i, a [i]  );
}
```

- an array can also be initialized by following the declaration with an equal sign and a comma separated list of values within a pair of curly brace

One Dimensional Array

```
main(){
        int a[10]={1,2,3,4,5,6,7,8,9};
        int i;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
                printf ("%d \t%d", i, a [i]  );
}
```

- If there are fewer values than elements in array, the remaining elements are initialized automatically with zero

One Dimensional Array

```
main(){
        int a[5]={1,2,3,4,5,6,7,8,9,10},
        int i;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
                printf ("%d \t%d", i, a [i]  );
}
```

- If you put more values than the array can hold, there will be a syntax error

One Dimensional Array

```
main(){
        int a[]={1,2,3,4,5},
        int i;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
              printf ("%d \t%d", i, a [i]  );
}
```

- If array size is omitted during declaration, the number of values during array initialization will be the number of elements the array can hold

Simple program using Array

```
main( )
{
      int  avg, sum = 0 ;
      int  i ;
      int  marks[30] ;  /* array declaration */

      for ( i = 0 ; i <= 29 ; i++ )
      {
            printf ( "\nEnter marks " ) ;
            scanf ( "%d", &marks[i] ) ;  /* store data in array */
      }

      for ( i = 0 ; i <= 29 ; i++ )
            sum = sum + marks[i] ;  /* read data from an array*/

      avg = sum / 30 ;
      printf ( "\nAverage marks = %d", avg ) ;
}
```

Character Arrays

- Character arrays have several unique features
- A character array can be initialized using a *string literal*

  char string1[ ] = “first”;

Character Arrays

- “first” string literal contains five characters plus a special string termination character called null character (\0)
- string1 array actually has 6 elements-f, i, r, s, t and \0

Character Arrays

- Character arrays can be initialized as follows as well-

char string1 [ ]= {‘f’, ‘i’, ‘r’, ‘s’, ‘t’, ‘\0’};

- We can access individual characters in a string directly using array subscript notation. For example, string1 [3] is the character ‘s’

Character Arrays

- We can also input a string directly from the keyboard using scanf () function and %s specifier.

```
char string1 [20];
scanf (“%s”, string1);
```

Character Arrays

- the name of the array is passed to scanf () without the preceding & used with other variables
- The & is normally used to provide scanf () with a variable's location in a memory so a value can be stored there
- Array name is the address of the start of the array, therefore, & is not necessary

Character Arrays

```
main(){
        char string1 [20], string2 [ ] = "Hello There";
        int i;
        printf ("Enter a string with white spaces: ");
        scanf ("%s", string1);
        printf ("\nString 2 is %s", string2);
        printf (\n String 1 is );
        for (i=0;string1 [i]!='\0';i++)
              printf ("%c", string1 [i]);
}
```