ARRAYS(II Unit Part II)

Array: An array is a collection of two or more adjacent cells of similar type.

- Each cell in an array is called as array element.
- Each array should be identified with a meaningful name and the number of adjacent cells to be associated for that array should be specified.

Single/One-Dimensional Arrays

The one dimensional arrays are also known as Single dimension array and is a type of Linear Array. In the one dimension array the data type is followed by the array name which is further followed by the size (an integer value) enclosed in square brackets and this represents the dimension either row wise or column wise.

Declaration of Array: To declare an array the following **syntax** should be followed:

<data type> arrayname[SIZE];

- where data type can be any fundamental or basic or instrinsic or implicit or user defined data type.
- Array name should follow the rules of an identifier.
- SIZE takes an integer value that specifies the number of adjacent cells required for that array.

For e.g., **int number[10];**

where **number** is the name of the array, the integer value 1**0** indicates the size of the array i.e., 10 adjacent cells are allocated with each cell storing **integer** element as the data type is **int**.

The memory is allocated with index starting from zero and ends with its size-1. Pictorially the memory allocation and its identification is represented as shown below:

Array name	Number[0]	Number[1]	..	..	..	..	..	..	Number[9]
Element									

Initializing elements to the array:

Elements can be initialized to the array as shown below. We initialize values to the array only if the values are going to be fixed.

General syntax:

<data type> arrayname[SIZE]={*value1,value2,….,valueN*};

For e.g. for the above declarations we will see how to initialize (assigning) values.

```
int number[10]={10,20,30,40,50,60,70,80,90,12};
                    ( or )
number[0]=10;
number[1]=20;
number[2]=30;
      .
      .
number[9]=12;
```

where 10 is assigned to location number[0],20 is assigned to location number[1] and so on.

Referencing the Array:

An array can be referenced in the program in any of the executable statements in either one of the ways given below:

Way 1:

Syntax is given below: **arrayname[index];**

where index takes an integer value, this integer value specifies to the array **location which is index + 1**, this integer value also should not be greater than the size of that array specified in the declaration of the array.

For e.g., int number[15];

to refer 11^{th} location of this array we go for number[10]

Way 2:

Syntax is given below:

Variable=value;
arrayname[variable];

Where an integer value is to be initialized to a variable and that variable is used within the square brackets of that array and this variable is called as **subscript variable** as this variable contains an integer value that refers to the index of that array.

Way 3: Using for loops for sequential access:

In order to read elements into the array sequentially or to access elements of an array sequentially we make use of looping constructs. We will see how this is done.

For reading elements into the array sequentially:

For e.g. for the declaration given below, we will see how elements are read sequentially into the array.

```
int number[10];    /* array declaration  */

/* reading elements into the array sequentially */

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

Explanation:

For loop is used to vary the index of the array from 0 to its size-1.The variable used in for loop is used in the scanf statement along with the array name enclosed in square brackets. This variable is called as subscript variable.

For printing elements of the array sequentially:

For e.g. if we suppose assume that the array is declared and the elements are already initialized, then we will see how elements are accessed sequentially from the array.

```
/* accessing elements from the array sequentially */

for( i=0; i<10 ;i++ )
{
        Printf(" %d ",  number[i]);
}
```

Explanation:

for loop is used to vary the index of the array from 0 to its size-1.

- The variable used in the for loop is used in any of the executable statement along with the array name enclosed in square brackets.
- In the above example each and every element of the array is accessed and assigned to the variable num and that value is immediately printed.

```
/* Sample program to illustrate the declaration, initialization and accessing elements of the array */
#include <stdio.h>
void main()
{
        int number[10]={1,2,3,4,5,6,7,8,9,10};  /* array declaration and intialization*/
        int i; clrscr();
        /* printing the elements of the array */
        printf("elements in the array are:\n");
        for(i=0; i<10; i++)
        printf("%d\n",number[i]);
}
```

```
 /* Sample program to illustrate the declaration, reading and printing elements of the array */
 #include <stdio.h>
 void main()
 {
          int number[10]  /* array declaration */
          int i; clrscr();
          /* reading the elements of the array */
          printf("Enter the elements into the array are:\n");
          for(i=0; i<10; i++)
          scanf(" %d ", &number[i]);
          /* printing the elements of the array */
          printf("elements in the array are:\n");
          for(i=0; i<10; i++)
          printf("%d\n",number[i]);
 }
```

Generating a Pointer to an Array

You can generate a pointer to the first element of an array by simply specifying the array name, without any index.
For example, given

int sample[10];

Element	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Address	1000	1001	1002	1003	1004	1005	1006

A seven-element character array beginning at location 1000

You can generate a pointer to the first element by using the name **sample**. Thus, the following program fragment assigns **p** the address of the first element of **sample**:

```
int *p;
int sample[10];
p = sample;
```

You can also specify the address of the first element of an array by using the **&** operator.
Forexample, **sample** and **&sample[0]** both produce the same results.

However, in professionally written C code, you will almost never see **&sample[0]**.

Passing Single Dimensional Arrays To Functions:

Arrays can be passed as argument in functions. To pass arrays as arguments one should know how to pass the arguments in function call and how to receive the arguments in the function definition.
The general format to pass an array as argument in function call is given below:

functionname (array name);

Where **array name** is the name of the array which is declared in the calling function.
The general format to receive the arguments in the function definition is given below:

<data type> functionname(<data type> arrayname[ ])

Where **array name** is the name that will be referred in the function definition.
This is used in place of the array name passed from the function call and this array should be declared for its type.

For e.g., **int data[10];**

we assume that the elements to the array are either initialized or read from the keyboard , then passing the array as argument in the function call is done as shown below:

printelements(data);

the arguments passed from function call are received in the function definition as shown below:

void printelements(int num[ ])

```
/* sample program illustrating how arrays are passed as arguments in functions */
#include <stdio.h>
void printelements(int [ ]); /* prototype declaration */
void main()
{
        int data[10]; /* array declaration */
          int i;
          clrscr();
          /* reading elements into the array from keyboard */
          printf("enter the elements:");
          for(i=0; i<10; i++)
          scanf(" %d ", &data[i]);
          /* function call */
          printelements(data);   /* array passed as argument in function call */
}
          /* function definition */
void printelements(int num[ ])    /* actual arguments received in function definition */
{
          int i;
          printf("elements are:\n");
          for(i=0; i<10; i++)
          printf("%d\n",num[i]);
}
```

STRINGS

Strings are sequence of characters enclosed in double quotes is considered as string. By default a string gets terminated with null character (\0) marking the end of the string and that character is not visible to the user. Whereas a character is enclosed in single quote and it is not terminated with any default character.

For e.g., printf("welcome to strings"); // "welcome to strings" is string constant.

Declaration of string variables.

The declaration of string variable is very much similar to declaring a character array. The general form is shown below:

char stringname[size];

for e.g., **char name[20];**

where name is the name of the string.

Initialization of string to character array

The general form of initializing string to character array is shown below:

char stringname[size]="value";

where value can be sequence of characters.

For e.g., char name[20]="learn strings";

From the above declaration,"name" is the name of the string or character array and "learn strings" is the value initialized to the character array "name" and by default terminated with null character(\0) marking the end of the string. Below we will see how the string is initialized and also see how memory is allocated.

Name of the array	Name																		
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Value	l	e	a	r	n		s	t	r	i	n	g	s	\0						

Initialization of characters to character array

For e.g.

char name[20]={'l','e','a','r','n',' ','s','t','r','i','n','g','s'};

initialized characters to string

char name[20]={'l','e','a','r','n',' ','s','t','r','i','n','g','s','\0'}; which is equivalent to

char name[20]="learn strings";

Reading a string to the character array:

If we suppose that a character array is declared as shown below:

```
char college_name[25];
```

then a string can be read using the scanf statement as shown below:

```
scanf("%s",college_name);
        or
for(i=0;i<25;i++)
scanf("%c",&college_name[i]);
```

Note: No need to preceed the array name with ampersand(&) in the scanf statement and %s format specifer stands for string, no need to specify the subscript variable.

Printing a string from the character array:

If we suppose that a character array is declared as shown below:

```
char college_name[25];
```

then a string can be read using the scanf statement as shown below:

```
scanf("%s",college_name);
```

and the contents of the character array can be printed as string as shown below:

```
printf("%s",college_name);
                or
for(i=0;i<25;i++)
scanf("%c", &college_name[i]);
          or
for(i=0;i<25;i++)
printf("%s",college_name[i]);
```

Note: No need to preceed the array name with ampersand(&) in the scanf statement and %s format specifer stands for string. no need to specify the subscript variable.

Array of Strings:

String is an array of characters and array of strings is 2-D array of characters in which each row is one string.

For e.g.,

```
char weekdays[7][12]={"Mon","Tues","Wed","Thurs","Fri","Sat","Sun"};
```

In the above example a character array by name "weekdays" is declared as 2-D array which can store 7 rows with each row of 12 characters initialized with strings.

String Library Functions:

C supports a wide range of functions that manipulate strings. The most common are listed here:

Name	**Function**
strcpy(*s1, s2*)	Copies *s2* into *s1*
strcat(*s1, s2*)	Concatenates *s2* onto the end of *s1*
strlen(*s1*)	Returns the length of *s1*
strcmp(*s1,s2*)	Returns 0 if *s1* and *s2* are the same; less than 0 if *s1<s2*; greater than 0 if *s1>s2*

strchr(*s1, ch*) Returns a pointer to the first occurrence of *ch* in *s1*
strstr(*s1, s2*) Returns a pointer to the first occurrence of *s2* in *s1*

These functions use the standard header **<string.h>**. The following program illustrates the use of these string functions:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
char s1[80], s2[80];
gets(s1);
gets (s2);
printf("lengths: %d %d\n", strlen(s1), strlen(s2));
if(!strcmp(s1, s2)) printf("The strings are equal\n");
strcat(s1, s2);
printf ("%s\n", s1);
strcpy(s1, "This is a test.\n");
printf(s1);
if(strchr("hello", 'e')) printf("e is in hello\n");
if(strstr("hi there", "hi")) printf("found hi");
return 0;
}
```

If you run this program and enter the strings "**hello**" and "**hello**", the output is

```
lengths: 5 5
The strings are equal
hellohello
This is a test.
e is in hello
found hi
```

Remember, **strcmp( )** returns false if the strings are equal. Be sure to use the logical **!** operator to reverse the condition, as just shown, if you are testing for equality.

Multi-dimensional arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

```
datatype name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

```
int threedim[5][10][4];
```

Two-Dimensional Arrays:

The simplest form of the multidimensional array is the two-dimensional array.
A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y you would write something as follows:

```
type arrayName [size1][ size2 ];
```

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier.

A two-dimensional array can be think as a table which will have x number of rows and y number of columns.

A 2-dimensional array **a**, which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form **a[i][j]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initializing Two-Dimensional Arrays:

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
                {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
                {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
                {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
            };
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements*:*

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

```
int val = a[2][3];
```

The above statement will take 4th element from the 3rd row of the array.

You can verify it in the above diagram. Let us check below program where we have used nested loop to handle a two dimensional array:

```
#include <stdio.h>

int main ()
{
  /* an array with 5 rows and 2 columns*/
  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
  int i, j;
```

```
   /* output each array element's value */
   for ( i = 0; i < 5; i++ )
   {
      for ( j = 0; j < 2; j++ )
      {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

Indexing Pointers

Pointers and arrays are closely related. As you know, an array name without an index is a pointer to the first element in the array.

For example, consider the following array:

```
char p[10];
```

The following statements are identical:

```
p          &p[0]
```

Put another way,

```
p == &p[0]
```

evaluates to true because the address of the first element of an array is the same as the address of the array.

As stated, an array name without an index generates a pointer.

Conversely, a pointer can be indexed as if it were declared to be an array. For example, consider this program fragment:

```
int *p, i[10];
```

```
p = i;

p[5] = 100; /* assign using index */

*(p+5) = 100; /* assign using pointer arithmetic */
```

Both assignment statements place the value 100 in the sixth element of **i**.

The first statement indexes **p**; the second uses pointer arithmetic. Either way, the result is the same

Variable Length Arrays

Thus, in C89 the size of an array is fixed at compile time.
However, this is not the case for C99, which adds a powerful new feature to arrays: variable length.
In C99, you can declare an array whose dimensions are specified by any valid expression, including those whose value is known only at run time.
This is called a ***variable-length array***. However, only local arrays (that is, those with block scope or prototype scope) can be of variable length.
Here is an example of a variable-length array:

```
void f(int dim)
{
char str[dim]; /* a variable-length character array */
/* . . . */
}
```

Here, the size of **str** is determined by the value passed to **f( )** in **dim**. Thus, each call to **f( )** can result in **str** being created with a different length.
One major reason for the addition of variable-length arrays to C99 is to support numeric processing. Of course, it is a feature that has widespread applicability. But remember, variable-length arrays are not supported by C89 (or by C++).