6: Arrays

6.1 One, two, multidimensional arrays

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.
- Array subscripts must be of integer type. (int, long int, char, etc.)
- **VERY IMPORTANT:** Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four. This is because the index in C is actually an offset from the beginning of the array. (The first element is at the beginning of the array, and hence has zero offset.)
- **IMP :** The most common mistake when working with arrays in C is forgetting that indices start at zero and stop one less than the array size.
- Arrays are commonly used in conjunction with loops, in order to perform the same calculations on all (or some part) of the data items in the array.

Define arrays

- An array is a collection of data items, all of the same type, accessed using a common name.
- A one-dimensional array is like a list; A two dimensional array is like a table; The C language places no limits on the number of dimensions in an array, though specific implementations may

SYNTAX AND INITILIZATION ONE -DIMENSIONAL ARRAYS

Declaring Arrays

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Uninitialized arrays must have the dimensions of their rows, columns, etc. listed within the square brackets.
- Dimensions used when declaring arrays in C must be positive integral constants or constant expressions
- Examples:

```
int i, j, intArray[ 10 ], number;
float floatArray[ 1000 ];
int tableArray[ 3 ][ 5 ];      /* 3 rows by 5 columns */
```

Initializing Arrays

- Arrays may be initialized when they are declared, just as any other variables.
- Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below.
- An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.
- If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data. (Variation: Multidimensional arrays - see below.)
- Examples:

```
int i =  5, intArray[ 6 ] = { 1, 2, 3, 4, 5, 6 }, k;
float sum  = 0.0f, floatArray[ 100 ] = { 1.0f, 5.0f, 20.0f };
double  piFractions[ ] = { 3.141592654, 1.570796327,0.785398163 };
```

SYNTAX AND INITILIZATION TWO-DIMENSIONAL ARRAYS

The simplest form of 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 [ x ][ y ];
```

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −

	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 the 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 the previous example −

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

Declare and initialize the one dimensional array with TEN elements. Explain how the elements in an array can be accessed.

To declare and initialize a one dimensional array with 10 elements:

int arr[10] = {10, 20, 5, 3, 55, 45, 15, 7, 30, 52};

The elements of an array can be accessed by using indices. The first element in an array will be represented by arr[0], the second element arr[1], third element arr[2], fourth element arr[3], fifth element arr[4] so on. The 10th element will be represented by arr[9].

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]
10	20	5	3	55	45	15	7	30	52

Elements of an array can also be accessed using loop.
Eg:

```
#include<stdio.h>
#include<conio.h>
void main() {
int arr[] = {10, 20, 5, 3, 55, 45, 15, 7, 30, 52};

int i;
clrscr();
for(i = 0; i < 10; i++) {
printf("%d\t",arr[i]);
}
getch();
}
```

Declare and initialize the one dimensional array with 10 elements. Explain how the elements in an array can be accessed.

Declaration of Array:

Syntax: datatype variable-name[size]

Declaration of 10 array element is : Int a[10];

Where a is variable name or array name, 10 is size of an array, int is datatype

Initialize the one dimensional array:

Syntax: datatype array-name[size] ={list of values};

Initialization of 10 array elements:

Int a[10]={ 10,20,30,40,50,60,71,70,80,90};

Accessing Element of an Array:

Once an array is declared, its individual element in the array can be referred. This is done with the subscript number in the brackets following the array name . this number specifies the element‟s position in the array. All the array element are number starting with „0‟. Thus marks[2] is not the second element of array but the third.

Define array with its need and how elements are allocated with space in memory for one dimensional array.

Ans: An array is a fixed-size sequenced collection of elements of the same data type.

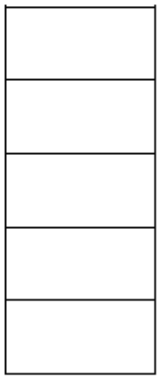
Need of an array:

A variable can store only one value at a given time. Therefore they can be used only to handle limited amount of data. To process such large amount of data, we need a powerful data type that facilitates efficient storing, accessing and manipulation of data items. Array can be used for such kinds of applications.

For one dimensional array elements are allocated by following way:

int number[5]={35,40,56,32,12};

The computer reserves five storage locations as below:



number[0]
number[1]
number[2]
number[3]
number[4]

The values to the array elements can be assigned as follows:

```
number[0]=35;
number[1]=40;
number[2]=56;
number[3]=32;
number[4]=12;
```

This would cause the array number to store the values as shown below:

number[0]	35
number[1]	40
number[2]	56
number[3]	32
number[4]	12

Explain how to initialize two dimensional array with example.

Ans: Following is an array with 3 rows and each row has 4 columns. The one pair of brackets represent

value for a rows. Now in below example there are three rows initialized.

```
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 */
};
```

Example:

```
#include <stdio.h>
#include<conio.h>
void 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] );
}
```

```
}
}
```

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
```

Find error in following program and justify it:

```
#include<stdio.h>
void main ()
{
Int i, a[5]={7, 5, 2,1,9,14};
For (i=0; i<5; i++)
Printf("%f", a[2]);
getch();
}
```

Errors :

1) array size is 5 and number of variables initialized is 6.

2) printf statement should include %d as control specifier because data type of array a is int.

Output :

22222

Differentiate between character array and integer array with respect to size and initialization.

Sr.No	integer array	character array
1	An integer array is an array whose elements are all of an integer type which has no fractional component.	character array is an array which contains nothing but character types
2	An integer array is capable of holding values of type int.	A char array is capable of holding values of type char
3	int array take more memory to allocate	char array take less memory to allocate
4	integer array takes 2 bytes for each cell	whereas char takes 1 byte for each cell
5	array 's r not terminated with NULL(\0) character	Strings r terminated with NULL(\0) character

Write a program to sort elements of an array in ascending order.

```
#include <stdio.h>
#include<conio.h>
void main()
{
int i, j, a, n, number[30];
clrscr();
printf("Enter the size of an array \n");
scanf("%d", &n);
printf("Enter the array elements \n");
for (i = 0;i<n; i++)
```

```
scanf("%d", &number[i]);
for (i = 0; i < n; i++)
{
for (j = i + 1; j < n; j++)
{
if (number[i] > number[j])
{
a =number[i];
number[i] = number[j];
number[j] = a;
}
}
}
printf("The numbers arranged in ascending order are \n");
for (i = 0; i < n; i++)
printf("%d\n", number[i]);
getch();
}
```

Write a program to sort element of an array descending order.

```
#include<stdio.h>

#include<conio.h>

void main()

{

int arr[5];

int i,j,temp;

clrscr();

printf("\n Enter elements:");

for(i=0;i<5;i++)

{
```

```
scanf("%d",&arr[i]);
}
for(i=0;i<4;i++)
{
for(j=i+1;j<5;j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
}
}
printf("\nSorted array elements :\n");
for(i=0;i<5;i++)
printf("%d ",arr[i]) ;
getch();
}
```

Write a program for addition of two 3 x 3 matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3],b[3][3],c[3][3],i,j;
```

```
clrscr();
printf("Enter first matrix elements:\n");

for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nEnter second matrix elements:\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&b[i][j]);
}
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
printf("\n\nAddition of two matrices is:");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",c[i][j]);
}
}
getch();
}
```

Multidimensional Arrays

- Multi-dimensional arrays are declared by providing more than one set of square [ ] brackets after the variable name in the declaration statement.

- One dimensional arrays do not require the dimension to be given if the array is to be completely initialized. By analogy, multi-dimensional arrays do not require **the first** dimension to be given if the array is to be completely initialized. All dimensions after the first must be given in any case.

- For two dimensional arrays, the first dimension is commonly considered to be the number of rows, and the second dimension the number of columns.
- Two dimensional arrays are considered by C/C++ to be an array of (single dimensional arrays). For example, "int numbers[5][6]" would refer to a single dimensional array of 5 elements, wherein each element is a single dimensional array of 6 integers. By extension, "int numbers[12][5][6]" would refer to an array of twelve elements, each of which is a two dimensional array, and so on.

Sample Program Using 2-D Arrays

```
/* Sample program Using 2-D Arrays */

    #include <stdlib.h>
    #include <stdio.h>

    int main( void ) {

       /* Program to add two multidimensional arrays */
       /* Written May 1995 by George P. Burdell */

       int a[ 2 ][ 3 ] = { { 5, 6, 7 }, { 10, 20, 30 } };
       int b[ 2 ][ 3 ] = { { 1, 2, 3 }, {  3,  2,  1 } };
       int sum[ 2 ][ 3 ], row, column;

       /* First the addition */

       for( row = 0; row < 2; row++ )
          for( column = 0; column < 3; column++ )
             sum[ row ][ column ] =
                a[ row ][ column ] + b[ row ][ column ];
```

```
    /* Then print the results */

    printf( "The sum is: \n\n" );

    for( row = 0; row < 2; row++ ) {
       for( column = 0; column < 3; column++ )
          printf( "\t%d",  sum[ row ][ column ] );
       printf( '\n' );   /* at end of each row */
    }

    return 0;
}
```

Code to Calculate Average Using Arrays

```
#include <stdio.h>


int main()
{
   int n, i;
   float num[100], sum = 0.0, average;


   printf("Enter the numbers of elements: ");
   scanf("%d", &n);


   while (n > 100 || n <= 0)
   {
      printf("Error! number should in range of (1 to 100).\n");
      printf("Enter the number again: ");
      scanf("%d", &n);
   }
```

```
  for(i = 0; i < n; ++i)
  {
     printf("%d. Enter number: ", i+1);
     scanf("%f", &num[i]);
     sum += num[i];
  }

  average = sum / n;
  printf("Average = %.2f", average);

  return 0;
}
```

Output

```
Enter the numbers of elements: 6

1. Enter number: 45.3

2. Enter number: 67.5

3. Enter number: -45.6

4. Enter number: 20.34

5. Enter number: 33

6. Enter number: 45.6
```

Average = 27.69