But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all condition are checked.

Lecture Note: 11

ARRAY

Array is the collection of similar data types or collection of similar entity stored in contiguous memory location. Array of character is a string. Each data item of an array is called an element. And each element is unique and located in separated memory location. Each of elements of an array share a variable but each element having different index no. known as subscript.

An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. And number of subscript is always starts with zero. One dimensional array is known as vector and two dimensional arrays are known as matrix.

ADVANTAGES: array variable can store more than one value at a time where other variable can store one value at a time.

Example:

```
int  arr[100];
```

int mark[100];

DECLARATION OF AN ARRAY :

Its syntax is :

Data type array name [size];

int arr[100];

int mark[100];

int a[5]={10,20,30,100,5}

The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space. Like for int data type, it occupies 2 bytes for each element and for float it occupies 4 byte for each element etc. The size of the array operates the number of elements that can be stored in an array and it may be a int constant or constant int expression.

We can represent individual array as :

int ar[5];

ar[0], ar[1], ar[2], ar[3], ar[4];

Symbolic constant can also be used to specify the size of the array as:

#define SIZE 10;

INITIALIZATION OF AN ARRAY:

After declaration element of local array has garbage value. If it is global or static array then it will be automatically initialize with zero. An explicitly it can be initialize that

Data type array name [size] = {value1, value2, value3…}

Example:

in ar[5]={20,60,90, 100,120}

Array subscript always start from zero which is known as lower bound and upper value is known as upper bound and the last subscript value is one less than the size of array. Subscript can be an expression i.e. integer value. It can be any integer, integer constant, integer variable, integer expression or return value from functional call that yield integer value.

So if i & j are not variable then the valid subscript are

ar [i*7],ar[i*i],ar[i++],ar[3];

The array elements are standing in continuous memory locations and the amount of storage required for hold the element depend in its size & type.

Total size in byte for 1D array is:

Total bytes=size of (data type) * size of array.

Example : if an array declared is:

int [20];

Total byte= 2 * 20 =40 byte.

ACCESSING OF ARRAY ELEMENT:

```
/*Write a program to input values into an array and display them*/

#include<stdio.h>

int main()

{

int arr[5],i;

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

{

printf("enter a value for arr[%d] \n",i);

scanf("%d",&arr[i]);

}
printf("the array elements are: \n");
for (i=0;i<5;i++)
{
printf("%d\t",arr[i]);
}
return 0;
}
```

OUTPUT:

```
Enter a value for arr[0]  = 12
Enter a value for arr[1]  =45
Enter a value for arr[2]  =59
Enter a value for arr[3]  =98
Enter a value for arr[4]  =21
The array elements are 12  45  59  98  21
```

Example: From the above example value stored in an array are and occupy its memory addresses 2000, 2002, 2004, 2006, 2008 respectively.

a[0]=12, a[1]=45, a[2]=59, a[3]=98, a[4]=21

ar[0]	ar[1]	ar[2]	ar[3]	ar[4]
12	45	59	98	21
2000	2002	2004	2006	2008

Example 2:

```
  /* Write a program to add 10 array elements */
   #include<stdio.h>
    void main()
{
 int i ;
int arr [10];
int sum=o;
for (i=0; i<=9; i++)
{
printf ("enter the %d element \n", i+1);
scanf ("%d", &arr[i]);
}
for (i=0; i<=9; i++)
{
sum = sum + a[i];
}
printf ("the sum of 10 array elements is %d", sum);
}
```

OUTPUT:

```
Enter a value for arr[0]  =5
Enter a value for arr[1]  =10
Enter a value for arr[2]  =15
Enter a value for arr[3]  =20
Enter a value for arr[4]  =25

Enter a value for arr[5]  =30

Enter a value for arr[6]  =35

Enter a value for arr[7]  =40

Enter a value for arr[8]  =45

Enter a value for arr[9]  =50

Sum = 275
```

while initializing a single dimensional array, it is optional to specify the size of array. If the size is omitted during initialization then the compiler assumes the size of array equal to the number of initializers.

For example:-

```
int  marks[]={99,78,50,45,67,89};
```

If during the initialization of the number the initializers is less then size of array, then all the remaining elements of array are assigned value zero .

For example:-

```
int marks[5]={99,78};
```

Here the size of the array is 5 while there are only two initializers so After this initialization, the value of the rest elements are automatically occupied by zeros such as

Marks[0]=99 , Marks[1]=78 , Marks[2]=0, Marks[3]=0, Marks[4]=0

Again if we initialize an array like

```
int array[100]={0};
```

Then the all the element of the array will be initialized to zero. If the number of initializers is more than the size given in brackets then the compiler will show an error.

For example:-

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

we cannot copy all the elements of an array to another array by simply assigning it to the other array like, by initializing or declaring as

```
     int a[5] ={1,2,3,4,5};
   int b[5];
   b=a;//not valid
```

(**note**:-here we will have to copy all the elements of array one by one, using for loop.)

Single dimensional arrays and functions

```
/*program to pass array elements to a function*/
#include<stdio.h>
void main()
{
int arr[10],i;
printf("enter the array elements\n");
for(i=0;i<10;i++)
{
scanf("%d",&arr[i]);
check(arr[i]);
}
}
```

```
void check(int num)

{

if(num%2=0)

{

printf(”%d is even \n”,num);

}

else

{

printf(”%d is odd \n”,num);

}

}
```

Lecture Note: 12

Two dimensional arrays

Two dimensional array is known as matrix. The array declaration in both the array i.e.in single dimensional array single subscript is used and in two dimensional array two subscripts are is used.

Its syntax is

Data-type array name[row][column];

Or we can say 2-d array is a collection of 1-D array placed one below the other.

Total no. of elements in 2-D array is calculated as **row*column**

Example:-

int a[2][3];

Total no of elements=row*column is 2*3 =6

It means the matrix consist of 2 rows and 3 columns

For example:-

```
 20   2    7

8     3    15
```

Positions of 2-D array elements in an array are as below

```
00    01    02

10    11    12
```

a [0][0]	a [0][0]	a [0][0]	a [0][0]	a [0][0]	a [0][0]
20	2	7	8	3	15
2000	2002	2004	2006		2008

Accessing 2-d array /processing 2-d arrays

For processing 2-d array, we use two nested for loops. The outer for loop corresponds to the row and the inner for loop corresponds to the column.

For example

int a[4][5];

for reading value:-

```
        for(i=0;i<4;i++)
        {
                for(j=0;j<5;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
```

For displaying value:-

```
        for(i=0;i<4;i++)
        {
        for(j=0;j<5;j++)
                {
                        printf("%d",a[i][j]);
                }
        }
```

Initialization of 2-d array:

2-D array can be initialized in a way similar to that of 1-D array. for example:-

int mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};

These values are assigned to the elements row wise, so the values of elements after this initialization are

Mat[0][0]=11,	Mat[1][0]=14,	Mat[2][0]=17	Mat[3][0]=20
Mat[0][1]=12,	Mat[1][1]=15,	Mat[2][1]=18	Mat[3][1]=21
Mat[0][2]=13,	Mat[1][2]=16,	Mat[2][2]=19	Mat[3][2]=22

While initializing we can group the elements row wise using inner braces.

for example:-

```
   int mat[4][3]={{11,12,13},{14,15,16},{17,18,19},{20,21,22}};
```

And while initializing , it is necessary to mention the 2nd dimension where 1st dimension is optional.

```
int mat[][3];

int mat[2][3];
```

$$\left.\begin{array}{l}\texttt{int mat[][];}\\ \texttt{int mat[2][];}\end{array}\right\}\text{invalid}$$

If we **initialize an array** as

```
   int mat[4][3]={{11},{12,13},{14,15,16},{17}};
```

Then the compiler will assume its all rest value as 0,which are not defined.

Mat[0][0]=11,	Mat[1][0]=12,	Mat[2][0]=14,	Mat[3][0]=17
Mat[0][1]=0,	Mat[1][1]=13,	Mat[2][1]=15	Mat[3][1]=0
Mat[0][2]=0,	Mat[1][2]=0,	Mat[2][2]=16,	Mat[3][2]=0

In memory map whether it is 1-D or 2-D, elements are stored in one contiguous manner.

We can also give the size of the 2-D array by using symbolic constant

Such as

```
        #define ROW 2;
```

```
#define COLUMN 3;
int mat[ROW][COLUMN];
```

String

Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character.

We can initialize the string as

char name[]={‘j’,’o’,’h’,’n’,’\o’};

Here each character occupies 1 byte of memory and last character is always NULL character. Where ’\o’ and 0 (zero) are not same, where **ASCII** value of ‘\o’ is 0 and ASCII value of 0 is 48. Array elements of character array are also stored in contiguous memory allocation.

From the above we can represent as;

J	o	h	N	‘\o’

The terminating NULL is important because it is only the way that the function that work with string can know, where string end.

String can also be **initialized** as;

char name[]=”John”;

Here the NULL character is not necessary and the compiler will assume it automatically.

String constant (string literal)