MODULE 3

ARRAYS & STRINGS

CO - Students will be able to demonstrate linear data structure implementations in C program using arrays and strings

Prepared By Mr. EBIN PM, AP, IESCE

ARRAYS

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

OR

- **An array is a collection of variables of the same data type that are referenced by a common name.** The array name acts as a pointer to the zeroth element of the array

TYPE OF ARRAYS

1) One dimensional arrays
2) Two- dimensional arrays
3) Multi- dimensional arrays

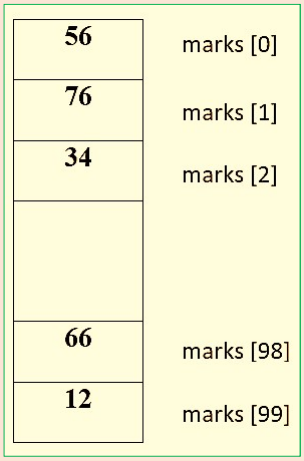
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2

❖ONE DIMENSIONAL ARRAY

- To begin with, like other variables an array needs to be declared so that the compiler will know what kind of an array and how large an array we want.
- The general **syntax** is

  type array_ name [size];

  Eg: int marks [100];
- The computer reserves 100 contiguous storage locations as shown



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 3

- An array is a collection of similar elements.
- The first element in the array is numbered 0, so the last element is 1 less than the size of the array.
- An array is also known as a subscripted variable. The element numbers in [ ] are called **index** or **subscript**.
- Before using an array its type and dimension must be declared. However big an array its elements are always stored in contiguous memory locations
- The **C** language treats character string, simply as array of characters. The size in the character string represents the maximum number of characters that the string can hold. For example

  char **name[10];**
- declares the name as a character array (string) variable that can hold a maximum of 10 characters.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 4

❖A Simple Program Using Array

- Let us try to write a program to find average marks obtained by a class of 50 students in a test.

```
#include<stdio.h>
#include<conio.h>
void main( )
 {
    int avg, sum = 0 ;
     int i ;
     int marks[50] ; /* array declaration */
     for ( i = 0 ; i <= 49 ; i++ )
      {
       printf ( "\n Enter marks " ) ;
       scanf ( "%d", &marks[i] ) ; /* store data in array */
      }

    for ( i = 0 ; i <= 49 ; i++ )
    sum = sum + marks[i] ; /* read data from an array*/
    avg = sum / 50 ;
    printf ( "\n Average marks = %d", avg ) ;
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 5

➢Entering Data into an Array

- Here is the section of code that places data into an array:

```
for ( i = 0 ; i <= 49 ; i++ )
  {
     printf ( "\n Enter marks " ) ;
     scanf ( "%d", &marks[i] ) ;
  }
```

- The first time through the loop, **i** has a value 0, so the **scanf ( )** function will cause the value typed to be stored in the array element **marks [0],** the first element of the array. This process will be repeated until **i** become 49

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 6

➢Reading Data from an Array

- The **for loop** is much the same, but now the body of the loop causes each student's marks to be added to a running total stored in a variable called **sum**. When all the marks have been added up, the result is divided by 50, the number of students, to get the average.

```
for ( i = 0 ; i <= 49 ; i++ )
    sum = sum + marks[i] ;
avg = sum / 50 ;
printf ( "\n Average marks = %d", avg ) ;
```


INITIALIZATION OF 1D ARRAY

An array can be initialized at either of the following stages.

- At compile time.
- At run time.

1) Compile time initialization:

Syntax

```
type array_name[size]= {list of values};
```

Eg: int number [3] = {9, 5, 2};

float total [5] = {0.0, 15.75,-10.9};

- The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements.

Eg: int counter [ ] = {1, 1, 1, 1};

Character arrays may be initialized in a similar manner.

Eg: char name [ ] = {'j','o','h','n','\0'};

- If we have more initializers than the declared size, the compiler will produce an error. That is the statement

• Int number [3] = {10, 20, 30, 40}; will not work. It is illegal in **C.**

2) Run time initialization:

• We can use **scanf ( )** to initialize an array.

```
int x [25],i;
for (i=0;i<25;i++)
   scanf ("%d",&x[i]);
```

TWO DIMENSIONAL ARRAYS

➢The two-dimensional array is also called a **matrix.** Two dimensional arrays are declared as follows.

type array_name [row-size] [column-size];

How do we initialize a two-dimensional array? As simple as this

```
int stud[4][2] = {
                    { 1334, 18 },
                    { 1812, 44 },
                    { 1004, 99 },
                    { 1112, 10 }
                  } ;
```

1334	18
1812	44
1004	99
1112	10

Or even this would work...

```
int stud[4][2] = { 1334, 18, 1812, 44, 1004, 99, 1112, 10 } ;
```

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

Thus the declarations,

```
int arr[2][3] = { 52, 30, 23, 55, 56, 85 } ;
int arr[ ][3] = { 52, 30, 23, 55, 56, 85 } ;
```

are perfectly acceptable,

whereas,

```
int arr[2][ ] = { 52, 30, 23, 55, 56, 85 } ;
int arr[ ][ ] = { 52, 30, 23, 55, 56, 85 } ;
```

would never work.

➢The array elements have been stored row wise and accessed row wise. However, you can access the array elements column wise as well. Traditionally, the array elements are being stored and accessed row wise.

❖Memory Map of a 2-Dimensional Array

- The array arrangement shown below is only conceptually true. This is because memory doesn't contain rows and columns.
- In memory whether it is a one-dimensional or a two-dimensional array the array elements are stored in one continuous chain.
- The arrangement of array elements of a two-dimensional array in memory is shown below:

s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]
1334	18	1812	44	1004	99	1112	10
65508	65510	65512	65514	65516	65518	65520	65522

- We can easily refer to the marks obtained by the third student using the subscript notation as shown below:

```
printf ( "Marks of third student = %d", stud[2][1] ) ;
```

CHARACTER ARRAYS & STRINGS

➢A string is a sequence of characters defined between double quotation marks

Eg: printf ("WELL DONE");

❖**Declaring and initializing**

- Strings are declared as an array of characters. The syntax is:

char string_name[size];

Eg: char city [10];
char name[30];

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 15

- When the compiler assigns a character string to a character array, it automatically supplies a **null character (\0)** at the end of string. Therefore size of a string

Size = maximum number of characters in string + one

- Character arrays can be initialized when they are declared. Initialization can be in either of the following two forms.

char city [9] ="NEW YORK";

char city[9]={'N','E','W',' ','Y','O','R','K','\0'};

- The following format is also valid in **C**

Char string[ ]= {'N','E','W',' ','Y','O','R','K','\0'};

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 16

❖READING STRING FROM KEY BOARD

1. Using scanf( ) function

```
char address[10];
scanf ("%s",address);
```

- In the case of character arrays, the ampersand (**&**) is not required before the variable name. The problem with scanf ( ) function is that it terminates its input on the first white space it finds.
- If we typed in at the terminal NEW YORK then only the string "NEW" will be read in to the array **address.** The **address** array is created in the memory as shown below.

N	E	W	\0	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

- The unused locations are filled with garbage value.
- If we want the entire line "NEW YORK", we use two character arrays of size.

```
char ad1[5],ad2[5];
scanf ("%s%s",ad1,ad2);
```

- Then assign the string "NEW" to ad1 and "YORK" to ad2.

➢**scanf ( )** is not capable of receiving multi-word strings. Therefore names such as 'Sinusha Roy' would be unacceptable.

➢The way to avoid this limitation is by using the function **gets ( )**.It does not skip white space.

➢The usage of functions **gets( )** and its counterpart **puts( )** is shown below.

```
void main ( )
  {
      char name[25] ;
      printf ( "Enter  full name\n " ) ;
      gets ( name ) ;
      puts ( "Hello!" ) ;
      puts ( name ) ;
   }
```

OUTPUT

```
Enter full name
Harun Shan
Hello!
Harun Shan
```


STRING HANDLING FUNCTIONS

➢String conversion functions are stored in the header file **<string.h>**

1. **Strlen - Finds length of a string**
2. **strlwr - Converts a string to lowercase**
3. **strupr - Converts a string to uppercase**
4. **strcat - Appends one string at the end of another**
5. **strcpy - Copies a string into another**
6. **strcmp - Compares two strings**
7. **strdup - Duplicates a string**
8. **strrev - Reverses string**

1) strlen ( )

- This function counts the number of characters present in a string.

```
void main( )
{
    char arr[ ] = "Newyear" ;
    int len1, len2 ;
    len1 = strlen ( arr ) ;
    len2 = strlen ( "Humpty Dumpty" ) ;
    printf ( "\nstring = %s length = %d", arr, len1 ) ;
    printf ( "\nstring = %s length = %d", "Humpty Dumpty", len2 ) ;
}
```

output

string = Newyear length = 7

string = Humpty Dumpty length = 13

2) strcpy( )

- This function copies the contents of one string into another.

```
void main( )
{
    char source[ ] = "Soniya" ;
    char target[20] ;
    strcpy ( target, source ) ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
}
```

output

source string = Soniya

target string = Soniya

3) strcat( )

- This function concatenates the source string at the end of the target string

```
void main( )
{
char source[ ] = "Brother" ;
char target[30] = "Hello" ;
strcat ( target, source ) ;
printf ( "\nsource string = %s", source ) ;
 printf ( "\ntarget string = %s", target ) ;
}
```

output

source string = Brother

target string = HelloBrother

4) strcmp( )

- This function 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, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters.

```
void 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

The two strings are identical—"Jerry" and "Jerry"—and the value returned by strcmp ( ) is zero

In the second call, the result is 4, which is the numeric difference between ASCII value of 'J' and ASCII value of 'F'

-32, which is the value of null character minus the ASCII value of space, i.e., '\0' minus ' ', which is equal to -32.

<ctype.h>

- The character function use **ctype.h** header file. It is used for character testing and conversion functions.
- **isalpha (c):** Determine if argument is alphabetic. It return non zero value if true, 0 otherwise. Return type is int.
- **isdigit (c):** Determine if argument is a decimal digit. It return non zero value if true, 0 otherwise. Return type is int.
- **islower (c):** Determine if argument is lower case. It return non zero value if true, 0 otherwise. Return type is int.
- **isupper (c):** Determine if argument is upper case. It return non zero value if true, 0 otherwise. Return type is int.

- **tolower (c):** Convert letter to lower case. Return type is int
- **toupper (c):** Convert letter to upper case. Return type is in

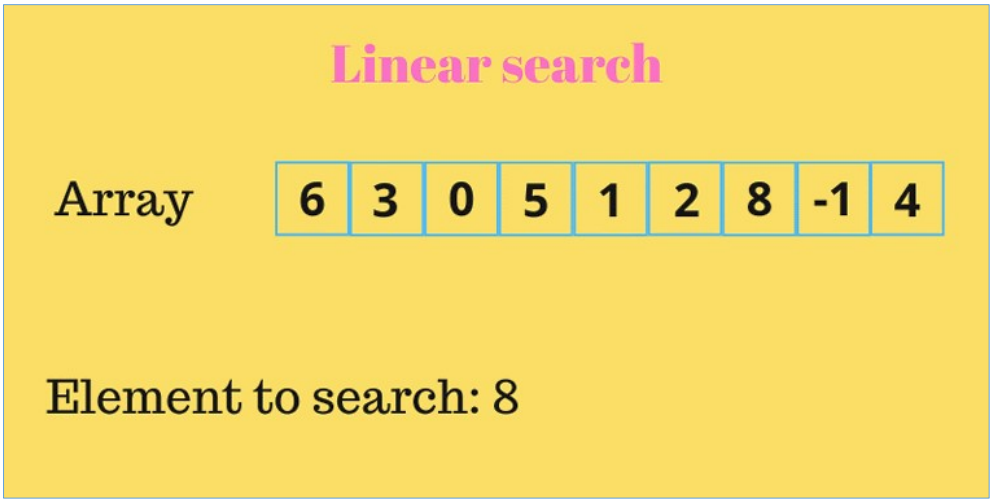
➢Let us consider an example to print the length of a string using (i) strlen() library function and gets (ii) gets( ) and null character

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
  int i;
  char name[30];
 printf(" Enter some string" );
 gets(name);
  i=strlen(name);
  printf("The length of string%d",i);
  getch();
}
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i,count=0;
  char name[30];
  printf(" Enter some string" );
  gets(name);
  for(i=0;name[i]!='\0';i++)
    count++;
 printf("The length of string%d",count);
 getch();
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 27

LINEAR SEARCH PROGRAM

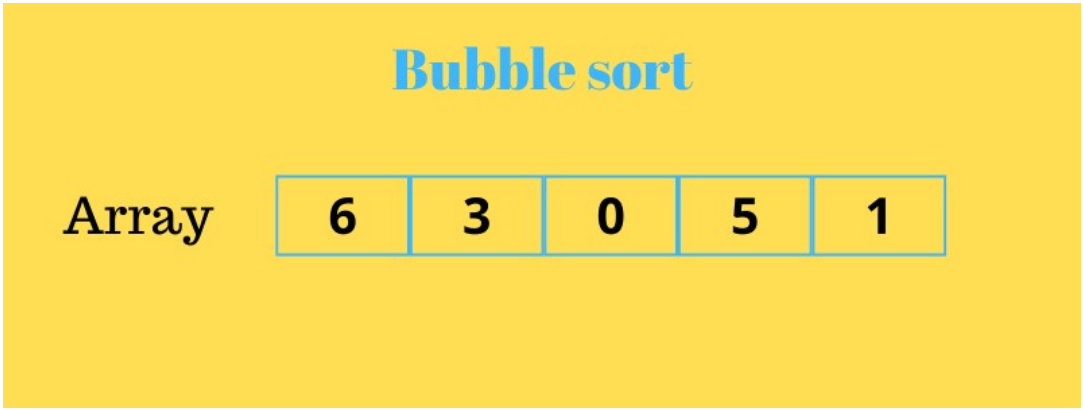


Prepared By Mr.EBIN PM, AP, IESCE EDULINE 28

Program

```c
#include <stdio.h>

int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```


BUBBLE SORT PROGRAM



Program

```c
#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);

  return 0;
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 31