UNIT – IV

ARRAYS :

An array is a group of related data items that share a common name.

Ex:- Students

The complete set of students are represented using an array name students. A particular value is indicated by writing a number called index number or subscript in brackets after array name. The complete set of value is referred to as an array, the individual values are called elements.

ONE – DIMENSIONAL ARRAYS :

A list of items can be given one variable index is called single subscripted variable or a one-dimensional array.

The subscript value starts from 0. If we want 5 elements the declaration will be

```
int  number[5];
```

The elements will be number[0], number[1], number[2], number[3], number[4]

There will not be number[5]

Declaration of One - Dimensional Arrays :

```
Type variable – name [sizes];
```

Type – data type of all elements Ex: int, float etc.,

Variable – name – is an identifier

Size – is the maximum no of elements that can be stored.

Ex:- float avg[50]

This array is of type float. Its name is avg. and it can contains 50 elements only. The range starting from 0 – 49 elements.

Initialization of Arrays :

Initialization of elements of arrays can be done in same way as ordinary variables are done when they are declared.

Type array name[size] = {List of Value};

Ex:- int number[3]={0,0,0};

If the number of values in the list is less than number of elements then only that elements will be initialized. The remaining elements will be set to zero automatically. Ex:- float total[5]= {0.0,15.75,-10};

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

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

/* Program Showing one dimensional array */

```
#include<stdio.h>
main()

{

i

n

t

i;

float x[10],value,total;

printf(“Enter 10 real

numbers\n”); for(i=0;i<10;i++)

        {

                scanf(“%f”,&value);

x[i]=value;

        }

        total=0;

        for(i=0;i<10;i++)        total=total+x[i]

for(i=0;i<10;i++)

printf(“x*%2d+=%5.2f\n”,I+1,x*I+);

printf(“total=%0.2f”,total);

}
```

TWO – DIMENSIONAL ARRAYS:

To store tables we need two dimensional arrays. Each table consists of rows and columns. Two dimensional arrays are declare as type array name [row-size][col-size];

```
/* Write a program Showing 2-DIMENSIONAL ARRAY */
/* SHOWING MULTIPLICATION TABLE */

#include<stdio.h>

#include<math.h>

#define  ROWS 5

#define  COLS 5

main()

{

 int row,cols,prod[ROWS][COLS];

int i,j;



        printf("Multiplication table");

for(j=1;j< =COLS;j++)

                printf("%d",j);

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

        {

                row = i+1;

                printf("%2d|",row);

for(j=1;j < = COLS;j++)
```

```
                {
                        COLS=j;
                        prod[i][j]= row * cols;
printf("%4d",prod*i+*j+);
                }
        }
}
```

INITIALIZING TWO DIMENSIONAL ARRAYS:

They can be initialized by following their declaration with a list of initial values enclosed in braces.

Ex:- int table[2][3] = {0,0,0,1,1,1};

Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as

int table[2][3] = {{0,0,0},{1,1,1}};

When all elements are to be initialized to zero, following short-cut method may be used.

int m[3][5] = {{0},{0},{0}};

STRINGS(CHARACTER ARRAYS) :

A String is an array of characters. Any group of characters (except double quote sign)defined between double quotes is a constant string.

Ex: “C is a great programming language”.

If we want to include double quotes.

Ex: “\”C is great \” is norm of programmers “.

Declaring and initializing strings :-

A string variable is any valid C variable name and is always declared as an array.

char string name [size];

size determines number of characters in the string name. When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at end of String. Therefore, size should be equal to maximum number of character in String plus one.

String can be initialized when declared as

1. char city*10+= “NEW YORK’;
2. char city[10]= ,‘N’,’E’,’W’,’
   ‘,’Y’,’O’,’R’,’K’,’/0’-; 3.

C also permits us to initializing a String without specifying size.

Ex:- char Strings* += ,‘G’,’O’,’O’,’D’,’\0’-;

READING STRINGS FROM USER:

%s format with scanf can be used for reading String.

char address[15];

scanf(“%s”,address);

The problem with scanf function is that it terminates its input on first white space it finds. So scanf works fine as long as there are no spaces in between the text.

Reading a line of text :

If we are required to read a line of text we use getchar(). which reads a single characters. Repeatedly to read successive single characters from input and place in character array.

```
/* Program to read String using scanf & getchar */

#include<stdio.h>

main()

{

        char line[80],ano_line[80],character;

        int c;

c=0;

        printf("Enter String using scanf to read

\n");    scanf("%s", line);        printf("Using

getchar enter new line\n");

        do

        {

                character = getchar();

ano_line[c] = character;

                c++;

        - while(character !='\n');

        c=c-1;

        ano_line*c+='\0';
```

}

STRING INPUT/OUTPUT FUNCTIONS:

C provides two basic ways to read and write strings.First we can read and write strings with the formatted input/output functions,scanf/fscanf and prinf/fprinf.Second we can use a special set of strin only functions ,get string(gets/fgets)and put string(puts/fputs).

Formatted string Input/Output:

Formatted String Input:scanf/fscanf:

Declarations:

int fscanf(FILE ****stream*, const char ****format***, ...);**
int scanf(const char ****format*, ...);**

The ..scanf functions provide a means to input formatted information from a

stream **fscanf** reads formatted input from a stream **scanf** reads formatted input

from stdin

These functions take input in a manner that is specified by the format argument and store each input field into the following arguments in a left to right fashion.

Each input field is specified in the format string with a conversion specifier which specifies how the input is to be stored in the appropriate variable. Other characters in the format string specify characters that must be matched from the input, but are not stored in any of the following arguments. If the input does not match then the function stops scanning and returns. A whitespace character may match with any whitespace character (space, tab, carriage return, new line, vertical tab, or formfeed) or the next incompatible character.

Formatted String Output:printf/fprintf:

Declarations:

int fprintf(FILE *_stream_, const char *_format_, ...);
int printf(const char *_format_, ...);

The ..printf functions provide a means to output formatted information to a stream.

fprintf sends formatted output to a stream

printf sends formatted output to stdout

These functions take the format string specified by the *format* argument and apply each following argument to the format specifiers in the string in a left to right fashion. Each character in the format string is copied to the stream except for conversion characters which specify a format specifier.

String Input/Output

In addition to the Formatted string functions,C has two sets of string functions that read and write strings without reformatting any data.These functions convert text file lines to strings and strings to text file lines. **gets():**

Declaration:

```
char *gets(char *str);
```

Reads a line from **stdin** and stores it into the string pointed to by *str*. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first. The newline character is not copied to the string. A null character is appended to the end of the string.

On success a pointer to the string is returned. On error a null pointer is returned. If the end-of-file occurs before any characters have been read, the string remains unchanged.

puts:

Declaration:

```
int puts(const char *str);
```

Writes a string to **stdout** up to but not including the null character. A newline character is appended to the output.

On success a nonnegative value is returned. On error **EOF** is returned.

STRING HANDLING/MANIPULATION FUNCTIONS:

strcat( ) Concatenates two Strings

strcmp( ) Compares two Strings

strcpy( ) Copies one String Over

another strlen( ) Finds length of

String

strcat() function:

This function adds two strings together.

Syntax: char *strcat(const char *string1, char *string2);

strcat(string1,string2);

string1 = VERY

string2 = FOOLISH

strcat(string1,string2);

string1=VERY FOOLISH

string2 = FOOLISH

Strncat: Append n characters from string2 to stringl.

char *strncat(const char *string1, char *string2, size_t n);

strcmp() function :

This function compares two strings identified by arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the Strings.

Syntax: int strcmp (const char *string1,const char *string2);

strcmp(string1,string2);

Ex:- strcmp(name1,name2);

strcmp(name1,"John");

strcmp("ROM","Ram");

Strncmp: Compare first n characters of two strings.

int strncmp(const char *string1, char *string2, size_t n);

strcpy() function :

It works almost as a string assignment operators. It takes the form

Syntax: char *strcpy(const char *string1,const char *string2);

strcpy(string1,string2); string2

can be array or a constant.

Strncpy: Copy first n characters of string2 to stringl .

char *strncpy(const char *string1,const char *string2, size_t n);

strlen() function :

Counts and returns the number of characters in a string.

Syntax:int strlen(const char *string);

n= strlen(string);

n→ integer variable which receives the value of length of string.

/* Illustration of string-handling */

```
#include<stdio.h>
#include<string.h>
main()
{
 char s1[20],s2[20],s3[20];  int  X,L1,L2,L3;
printf("Enter  two  string  constants\n");
scanf("%s %s",s1,s2);  X=strcmp(s1,s2);  if
(X!=0)
        {
                printf("Strings are not equal\n");
                strcat(s1,s2);
        }
        else
        printf("Strings are equal \n");
strcpy(s3,s1);
        L1=strlen(s1);

        L2=strlen(s2);  L3=strlen(s3);
printf("s1=%s\t length=%d chars \n",s1,L1);
printf("s2=%s\t length=%d chars \n",s2,L2);
printf("s3=%s\t length=%d chars \n",s3,L3);
}
```