Module 4

Arrays:

1. To perform operation on several inputs, lot of variables need to be declared which becomes a tedious task. This problem can be solved by using derived data structure 'Array'.
2. An array is defined as an ordered set of similar data items that shares common name; due to which, it is also called as homogeneous data structure.
3. All the data items of an array are stored in consecutive locations in memory.
4. The elements of an array can be differentiated from one another by their index or position.
5. Types of Arrays:
   a. Single dimensional Array:
      i. Declaration Syntax: data-type array_name[size];
      ii. Example: int a[10];

      It declares an array, named a, consisting of ten elements, each of type int. These elements are numbered from 0 to 9.

      iii. Array Initialization:
         A. Compile Time Initialization: It is possible to initialize elements of an array when the array is defined. Syntax: data-type array_name[]={list of values…..};

```
Ex:  int a[4] = {10, 20, 30, 40};          int n[ ] = { 2, 4, 12, 5, 45, 5 };
     char b[]={'C','O','M','P','U','T','E','R'};   char b[]="COMPUTER";
```

         B. Run Time Initialization: It is possible to enter data into an array during runtime by accepting from the user.

```
Ex:  for ( i = 0 ; i < 10 ; i++ ){
         scanf ( "%d", &a[i] ) ;
     }
```

      iv. Accessing Elements of an Array/Displaying Data from an Array:

```
Ex:  for ( i = 0 ; i < 10 ; i++ ){
         printf ( "%d", a[i] ) ;
     }
```

   b. Multidimensional arrays/Array of the arrays:
      i. In multidimensional arrays the array is divided into rows and columns. These are well suited to handle a table of data.
      ii. Declaration Syntax: data_type array_name[row_size][column_size];
      iii. Example: int arr[3][3]; int a[4][2];

	col. no. 0	col. no. 1
row no. 0	1234	56
row no. 1	1212	33
row no. 2	1434	80
row no. 3	1312	78

      iv. Array Initialization: int a[2][3] = {{ 0,0,0},{1,1,1}};
      v. Example:

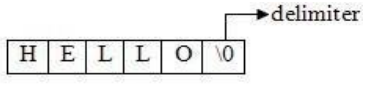
```
for (i=0 ; i<3; i++) {
    for(j=0;j<3;j++) {
        scanf("%d",&arr[i][j]);
        }
}
```

Strings:

1. In C, a string is group of characters which is terminated by delimiter '\0' (null).

2. A string constant is a one-dimensional array of characters terminated by a null ('\0').
3. The terminating null ('\0') is important, because it is the only way the functions that work with a string can know where the string ends.
4. Strings declaration syntax: char string_name[size];
   Where, size determines the number of characters in the string name.
5. Example: char city[10];
6. Strings Initialization: There are several methods to initialize values for string variables.
   a. Example: char city[8]="NEWYORK";
   b. Example: char city[8]={"N","E","W","Y","O","R","K","\0"};

Ex:-



How to read strings from the user?

1. Formatted input function:
   a. scanf() can be used with %s format specifier to read a string.
   b. Example:
   ```
   charname[10];
   scanf("%s",name);
   ```
   c. **Note:** "&" sign is not used because name of string is a pointer to array.
   d. The problem with scanf is that it terminates its input on the first white space it finds like:
   Example: for "NEW YORK", scanf reads only NEW.
2. Unformatted input function:
   a. gets() is more convenient method of reading a string of text including blank spaces.
   b. Example:
   ```
   char line[100];
   gets(line);
   ```

How to write strings to the user?

1. Formatted output function:
   a. printf() can be used with %c format specifier to write a string character by character till the array ends using loop.
   b. Example:
   ```
   while ( name[i] != '\0' ) {
           printf ( "%c", name[i] ) ;
           i++;
   }
   ```
   OR
   c. printf() can be used with %s format specifier to print strings on to the screen.
   d. Example:
   ```
   printf("%s",name);
   ```
2. Formatted output function:
   a. puts() is used to print strings including blank spaces.
   b. Example:
   ```
   char line[15]="Welcome to lab";
   puts(line);
   ```

String Inbuilt Functions in C:

1. strlen():
   a. Defn: to find the length of the string excluding the NULL character. In other words, it is used to count the number of characters in a string. It is defined in string.h header file.
   b. Syntax: int strlen(const char *str);
   c. Example:
   ```
   #include <stdio.h>
   void main(){
   char str1[ ] = "WELCOME";
   int n;
   n = strlen(str1);
   printf ("Length is:%d",n);
   }
   ```
2. strcpy():
   a. Defn: is used to copy one string to another. In C it is present in string.h header file
   b. Syntax: char* strcpy(char* dest, const char* src);
   c. Example:
   ```
   #include<stdio.h>
   ```

```
#include<string.h>
void main() {
   char str1[]="Hello Geeks!";
   char str2[100];
   strcpy(str2, str1);
   printf ("str1: %s\nstr2: %s ", str1, str2);
}
```

3. strcmp():
   a. Defn: compares two strings character by character (ASCII comparison) and returns the result. It is defined in string.h header file. Result is zero when both strings are found to be identical. A value greater than zero is returned when the first not matching character in leftStr have the greater ASCII value than the corresponding character in rightStr and vice versa.
   b. Syntax: int strcmp(const char *leftStr, const char *rightStr);
   c. Example:

```
#include<stdio.h>
#include<string.h>
void main() {
        char leftStr[] = "zfz";
        char rightStr[] = "gfg";
        int res = strcmp(leftStr, rightStr);
        printf("Value of result: %d" , res);
}
```

4. strcat():
   a. Defn: This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string. It is defined in string.h header file.
   b. Syntax: char *strcat(char *dest, const char *src);
   c. Example:

```
#include <stdio.h>
#include <string.h>
void main(){
  char str1[] = "VERY", str2[] = "GOOD";
  strcat(str1,str2);
  printf("Concatenated string is: %s",str1);
}
```

5. strchr():
   a. Defn: Returns a string from the first occurrence of character ch in string s1. It is defined in string.h header file.
   b. Syntax: char *strchr(const char *str, char c);
   c. Example:

```
#include <stdio.h>
#include <string.h>
void main () {
  char str[] = "programming";
  char ch = 'a';
  printf("String is - %s", strchr(str, ch));
}
```

Some other string functions:

Function	Description
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcmpi	Compares two strings ignoring the case
strrchr	Finds last occurrence of a given character in a string
strrev	Reverses string