CHAPTER-2

CONCEPTS OF C PROGRAMMING

2.1 FUNCTIONS

2.1.1 WHAT IS A FUNCTION?

A function is a block of code to perform a specific task.

- Every c program has at least one function main (). Without main () function, there is technically no c program.
- In modular programming, the program is divided into separate small programs called modules. Each module is designed to perform specific function. Modules make our actual program shorter, hence easier to read and understand.

2.1.2 ADVANTAGES OF FUNCTIONS

- Reusability: particular set of instructions can be used repeatedly from several different places within the program.
- Easy Debugging: Since each function is smaller and has a logical clarity its easy to locate and correct errors in it.
- Build library: Functions allow you to build a customized library of frequently used routines pr system-dependent features. Each routine can be programmed as a separate function and stored within a special library file.

2.1.3 TYPES OF C FUNCTIONS

There are two types of functions in c programming:

1. Library function: they are built in functions in c compiler.

   For example: main ()//the execution of every c program stats from this function

   Printf () //used to display output in c

   Scanf () //used to take input in c

   Sqrt(x)// finds the square root of x
2. User defined function: c allows the programmer to define their own function according to their requirement. These are called as user defined functions.

2.1.4 FUNCTION DECLARATION

Every c function should be declared before they are used. Function declaration gives the compiler information about

- Function name
- Type of arguments to be passed and
- Return type

The syntax is shown below:

```
return_type function_name (formal parameters);
```

2.1.5 FUNCTION CALL

A function can be called by specifying its name followed by a list of arguments enclosed in the parentheses and separated by commas.

The syntax is shown below:

```
function_name (actual parameters);
```

2.1.6 FUNCTION DEFINITION

Function definition contains programming codes to perform specific task.

The syntax is shown below:

```
return_type function_name (formal parameters)
{
        Variable declarations;
        Statement 1;
        Statement 2;
        return value;
}
```

Where, return_type is data type the function returns; function_name refers to the name of the function; formal parameters are a comma separated list of variables that receive the values from the main program when the function is called; return statement returns the result of the function.

2.1.7 ACTUAL PARAMETERS AND FORMAL PARAMETERS

ACTUAL PARAMETER	FORMAL PARAMETER
Actual parameters are used in calling function when a function is invoked.	Formal parameters are used in the function header of called function.
They can be constants, variables or expressions.	They can only be variables.
They send values to formal parameters.	They receive values from the actual parameters.
Address of actual parameter is sent to formal parameter.	If formal parameters contain address, they should be declared as pointers.

2.2 STRUCTURES

2.2.1 WHAT ARE STRUCTURES?

- Structure is a collection of elements of different data type, grouped together under a single name.
- Suppose, you want to store the information about person about his name, citizenship number and salary.
- This information can be stored separately but a better approach will be collection of these information under single name because all these information are related to one person.
- For this purpose, a structure can be used.

2.2.2 DECLARATION OF STRUCTURES

The keyword struct is used to declare structure followed by structure tag. The syntax is shown below:

```
struct structure_name
{
        data_type member 1;
        data_type member 2;
        data_type member 3;
};
```

- The variables that are used to store the data are called members of the structure.
- Structure definition is terminated with a semicolon.
- We can create the structure for a person as shown below:

```
struct student
{
        char name[50];
        char usn[10];
        int age;
        float marks;
};
```

2.2.3 DECLARATION OF STRUCTURE VARIBALES

The structure can be declared using following methods:

- when a structure is defined, it creates a user-defined type but, no storage is allocated. The syntax for creating variable can be written as:

```
struct structure_name structure_var1, structure_var2…..;
```

example: struct student s1, s2;

Or

```
struct structure_name
{
        data_type member 1;
        data_type member 2;
        data_type member 3;
} structure_var1, structure_var2….;
```

Example: struct student

```
{
        char name[50];
        int age;
        float marks;
} s1, s2;
```

- here s1 and s2 are variables of type struct student.
- Once the structure variable is declared, the compiler allocates memory for the structure variables.
- The size of the memory allocated is the sum of individual members.

2.2.4 STRUCTURE VARIBLE INITIALIZATION

Initializing a structure means assigning some constant to the members of the structure. The syntax for initializing the structure member variables is:

```
struct structure_name structure_var = {constant1,constant2……..};
```

Example: struct student s1 = {“CANARA”,18, 25.0};

2.2.5 ACCESSING MEMBERS OF A STRUCTURE

The members of a structure can be accessed by using the (.) dot operator. Any member of a structure can be accessed as the following syntax:

```
structure_variable_name.member_name;
```

Example: s1.name = “CANARA”; s1.age=18;

2.3 ARRAYS

2.3.1 WHAT IS AN ARRAY?

- An array is a collection of elements of the same datatype which is stored in consecutive memory locations.
- Each value in an array is indicated by the same name that is array name and an index which indicates the position of value in ana array.
- The datatype of an array can be int, float, double, char and string.

2.3.2 REPRESENTATION OF INDEX

In the below example, ‘a’ represents the name of an array. ‘0’ to ‘4’ represents the index. The index always begins from zero(0) and continues till (size-1) position.

Example: int a[5];

a[0]	a[1]	a[2]	a[3]	a[4]

Depending on the length of the array it is divided into two types. They are

1. Fixed length: it is an array where the length is known in compile time.
   Example: int a[5];
2. Variable length: it is an array where the length is not known at the compile time.
   Example: int a[ ];

2.3.3 ONE DIMENSIONAL ARRAY (or) 1D ARRAY

This is the simplest type of array that contains only one row(linear list) for storing the values of same datatype.

2.3.4 DECLARATION OF 1D ARRAY

The syntax for declaring 1D array is:

```
datatype array_name[size];
```

where

- datatype: type of an array
- array_name: name to represent the set of values
- size: the number of values that can be stored.

2.3.5 INITIALIZATION OF 1D ARRAY

Initialization of array can be done in three ways. They are:

1. Initialize in a single statement:
   i. Basic initialization - Ex: int a[5]={10,20,30,40,50};
   ii. Initialization without size - Ex: int a[ ]={10,20,30,40,50};
   iii. Partial initialization – Ex: int a[5]={10,20};
2. Assigning values to each index:
   Initialization of elements can be done by assigning index values one by one.
   Ex: int a[5];
   a[0]=10;
   a[2]=20;
   a[3]=30;

a[4]=40;

a[5]=50;

3. Read input from keyboard:

   Arrays can also be initialised during run time by reading values from user using the scanf function.

2.3.6 TWO-DIMENSIONAL ARRAY (or) 2D ARRAY

the two-dimensional array is the simplest form of multi-dimensional array. A two-dimensional array is that which contains rows and columns for storing the values of same datatype.

2.3.7 DECLARATION OF 2D ARRAY

the syntax for declaration of 2D arrays is as shown below:

```
datatype array_name[size1][size2];
```

where

- datatype: type of an array
- array_name: name to represent the set of values.
- size1: the number of rows that can be stored
- size2: the number of columns that can be stored

ex: int a[2][3]

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]

2.3.8 INITIALIZATION OF 2D ARRAY

Initialization of array can be done in three ways. They are:

1. Initialize in a single statement:

   Ex: int a[3][3]={{10,20,30},{40,50}};

2. Assigning values to each index:

   Initialization of elements can be done by assigning index values one by one.

   Ex: int a[3][3];

   a[0][0]=10;

   a[0][1]=20;

a[1][0]=30;

a[1][1]=40;

3. Read input from keyboard:

Arrays can also be initialised during run time by reading values from user using the scanf function.

2.4 STRINGS

2.4.1 WHAT ARE STRINGS?

- A group of characters together is called String.
- String is always enclosed within double quotes (“ “)
- String always ends with delimiter (NULL – ‘\0’)
- Ex: a = “CBIT”

2.4.2 DECLARATION OF STRINGS

A String is declared like an array of character.

Syntax:

```
data_type string_name[size];
```

Example: char name[20];

String size 20 means it can store up to 19 characters plus the NULL character.

2.4.3 INITIALIZATION OF STRINGS

Syntax:

```
data_type string_name[size] = value;
```

Example: The String can be initialized in two ways:

i. char name[30] = “SUVIKA”;

ii. char name[30] = {‘S’, ‘U’, ‘V’, ‘I’, ‘K’, ‘A’, ‘\0’};

2.4.4 STRING MANIPULATION FUNCTIONS

- C library supports a large number of string handling functions that can be used to carry out many of string manipulation and are stored in header file <string.h>
- There are mainly 6 string handling functions:

2.4.4.1 strcpy( ) – STRING COPY

- It is possible to assign the value to a string variable using strcpy( ).
- It allows us to copy one string from one location to another.
- Syntax:

```
strcpy(destination, source);
```

This function has two parameters:

i. Destination: A string variable whose value is going to be changed.

ii. Source: A string variable which is going to be copied to destination.

Ex:
```
char a[10], b[10];
strcpy(a, "CBIT");
strcpy(b, a);
```

After two calls to strcpy( ) a, b contains CBIT.

2.4.4.2 strlen( ) – STRING LENGTH

- The string length function can be used to find the length of the string in bytes.
- Syntax:

```
length = strlen(str);
```

String length function has one parameter str which is a string. It returns an integer value.

Ex:
```
char str[10] = "CBIT";
int length;
length = strlen(str);
```

2.4.4.3 strcmp( ) – STRING COMPARE

- It is used to compare two strings. It takes the two strings as a parameter and returns an integer value.
- Syntax:

```
result = strcmp(first, second);
```

result = 0 - first = second

result > 0 - first > second

result < 0 - first < second

Example:

```
int res;
res = strcmp("cat", "car"); //res > 0
res = strcmp("pot", "pot"); //res = 0
res = strcmp("big", "small"); //res < 0
```

2.4.4.4 strcat( ) – STRING CONCATENATE

- It is used to concatenate or join two strings together. It has two parameters where the combined string will be stored in the (destination) first parameter.
- Syntax:

```
strcat(destination, source);
```

Example:

```
char first[30] = "Computer";
char last[30] = "Programming";
strcat(first, last);
```

2.4.4.5 strncmp( ) – STRING 'n' COMPARE

- It compares up to specify number of characters from the two strings and returns integer value.
- Syntax:

```
result = strncmp(first, second, numchars);
```

result = 0 - first = second

result > 0 - first > second w.r.t number of characters

result < 0 - first < second

Example:

```
int res;
res = strncmp("string", "stopper", 4); //res > 0
res = strncmp("string", "stopper", 2); //res = 0
res = strncmp("stopper", "string", 4); //res < 0
```

2.4.4.6 strncpy( ) – STRING ‘n’ COPY

- This function allows us to extract a substring from one string and copy it to another location.
- Syntax:

```
strncpy(dest, source, numchars);
```

- The strncpy( ) takes three parameters. It copies the number of characters (numchars) from source string to destination string.
- Since, numchars doesn’t include NULL character we have to specify explicitly.

Ex:

```
char a[10] = “CBIT”;
char b[10];
strncpy(b, a, 2);
b[2] = ‘\0’;
```