2024

Programming Using C

Dr. Babasaheb Ambedkar Open University

Programming Using C

Course Writer:

Dr. Kamesh R. Raval
Assistant Professor,
Som-Lalit Institute of Computer Applications

Content Reviewer and Editor:

Prof. (Dr.) Nilesh K. Modi
Professor & Director School of Computer Science,
Dr. Babasaheb Ambedkar Open University

Programming Using C

Contents

BLOCK3: ARRAYS AND FUNCTIONS

UNIT1   ARRAYS

Objectives, Introduction, Understanding arrays, One-Dimensional array, Operations on arrays, Two-Dimensional array, Let Us Sum Up

UNIT2   HANDLING STRINGS

Objectives, Introduction, Understanding strings, Displaying strings in different formats, Standard functions of string handling, Table of strings, Let Us Sum Up

UNIT3   FUNCTIONS

Objectives, Introduction, Need for User Defined Functions, A Multifunction Program, The Form of C Functions, Return values and their types, Calling of Functions, Category of Functions, Let Us Sum Up

UNIT 4   MORE ABOUT FUNCTIONS

Objectives, Introduction, Handling of non-integer functions, Nesting of Functions, Recursion, Function with Arrays, Scope and Lifetime of Variables in Functions, ANSI C Functions, Let Us Sum Up

---

BLOCK 4: STRUCTURES, POINTERS AND FILE HANDLING

UNIT 1   STRUCTURES AND UNIONS

Objectives, Introduction, Structures, Unions, Let Us Sum Up

UNIT 2   POINTERS

Objectives, Introduction, Understanding Pointers, Pointer Expressions, Pointers and Arrays, Pointers and Character Strings, Pointers and Functions, Pointers and Structures, Points on Pointers, Let Us Sum Up

UNIT 3   FILE HANDLING

Objectives, Introduction, Management of Files, Input/Output Operations on Files, Error Handling during I/O Operations, Let Us Sum Up

UNIT 4   SOLVED PROGRAMS-III

Dr. Babasaheb Ambedkar Open University

BSCITRMA-101

Programming Using C

BLOCK 1: BASICS OF C

Block Introduction

The programming language C was originally developed by Dennis Ritchie of Bell Laboratories and was designed to run on a PDP-11 with a UNIX operating system. It proved to be a powerful, general purpose programming language.

Due to its simple language, expression, compactness of the code and ease of writing a C compiler it is the first high level language used on advance computers, including microcomputers, minicomputers and mainframes.

C is a preferred language among programmers for business and industrial applications because of its features, simple syntax and portability.

In this block, we will study about the basics of C language including its character set, operators and managing input and output operations. This block will be beneficial for beginners who are learning this language for the first time. The basic concepts are explained in a very easy manner.

Once you understand these concepts given in all the units of this block, you will be able to develop programs very easily.

Block Objective

The objective of the block is to aware students, about the programming languages. Student will know, what is programming language, how machine will execute set of instructions called program, and how many different types of programming languages are there. After understanding this block student will learn, what is C-Language? What are the advantages and disadvantages are there of the C-Language?

Main objective of this block is to aware students, about different data types available in the C-Language, various operators and their precedence. Students will learn, how to write IO statements in C-Language?

Finally, the block will clear the concept of program structure of C-Language, so that the student can start making/writing simple programs in C-Language.

Block Structure

UNIT 1 INTRODUCTION TO C PROGRAMMING

Unit Structure

1.0   LEARNING OBJECTIVES

In this unit, we will discuss about the basics of C required for beginners to understand this language.

After working through this unit, you should be able to:

- Comprehend the various types of programming languages
- Understand the basic program structure of C-Language
- Compile and execute your first C-Program
- Describe advantages of C-Language

1.1   INTRODUCTION

C-Language is a most powerful, convenient for the programmers and easy to learn programming language. C-Language is called middle level programming language, which has features of Low-Level (Machine) language as well as it is much simpler as High-Level languages. In this chapter, we will discuss basic and some essential things which you should know, so that you can write a simple and small program in C-Language.

1.2   TYPES OF PROGRAMMING LANGUAGES

Programming languages can be classified into three categories as follows: [1] Machine Language [2] Assembly Language and [3] High-Level Language.

1.2.1 Machine Language

Similar to all electronic devices, Computer is also designed by using IC (Integrated Circuit) technology. ICs are small chips made by Germanium or Silicon kind of semi-conductor materials. Few millions or even more transistors or capacitors kind of electronic components are fitted inside this IC chip. Usually, this type of electronics components can have two states, charged or discharged. Depending upon the component is charged or discharged we assumes 1 or 0. This means computer can understand only one language, and that is the language of 1's and 0's. We can have any type of data like Numbers, Strings (Texts), Images, Videos or Audios but when it stores in the computer, it will be stored in the form of 1's and 0's.

This language in which only two symbols (1 and 0) is allowed is called native language of the machine or Binary language. This language is a Low-Level language and machine can execute it very easy without any kind of translation. So, the execution of the machine language is a faster. But think, how programmer can write all the programming instructions in terms of binary stings like 1100100110.

It is difficult for the programmer to memorize all different instructions in the form different binary strings. Therefore, learning machine language is difficult for the programmer.

Check Your Progress-1

1. Binary language is also called _____ Language.

[A] Machine   [B] High-Level

[C] Mark-up   [D] None of these

2.A language which use only 0 and 1 (two symbols) is _____ language.

[A] Assembly   [B] Binary

[C] C-Language   [D] Java

3._____ language is difficult to learn but easier of the computer to execute.

[A] High-level   [B] Low-level

[C] SQL   [D] C-Language

4. _____ language does not require any kind of translator.

[A] Machine   [B] C-Language

[C] Java   [D] SQL

1.2.2 Assembly Language

Assembly is a programming language, in which instead of memorizing binary strings for different types of instruction, programmer can use mnemonic instructions like ADD for addition, SUB for subtraction, MUL for multiplication and so on. Because, programmer can use mnemonic instruction in English it is more convenient for the programmer.

But how machine can understand ADD, SUB and MUL kind of mnemonic instructions? We know that machine can understand only binary language. Programmer here, use a translator called Assembler after writing the program and before executing the program. Assembler is a software which reads all mnemonic instructions written by the programmer in the program and translate it in to the strings of binaries (machine codes), which will be executed by the CPU of the computer system.

Mnemonic Instruction → Assembler → Machine Code
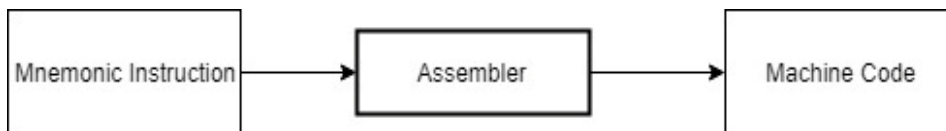
1.2.3 High-Level Language

Most High-Level languages are programmers friendly. Programmer can write the instruction in English words. High-Level languages use complier or interpreter to convert the source code into the machine language, before executing the program. Compilers also allow programmer to write an instruction, which will be translated into several instructions in the Low-Level language. This reduces programming efforts and number of lines in the code. High-Level languages are easy to program, easy to understand and more readable compare to Assembly and Binary languages.

1.2.4 Assembler, Interpreter and Compiler

Assembler, Interpreter and Compilers are all different types of translators which translate high-level source code written by the programmer into machine language codes.

1.   Assembler: Assembler is a kind of translator which translate mnemonic codes written in the assembly language into the machine code. Assembler is a simple translator which translate one line of assembly code into one line of machine code. Because of assembler, programmer do not have to memorize binary strings for different instructions, and programmer can use English words like ADD, SUB, MUL etc.

2.   Interpreter: Interpreter and Compilers do the same thing (translating programming code written in the High-Level language into the Binary code). But it fetches the one instruction of the program, it translate only that instruction in the machine code and execute it. After execution of the one instruction, it fetches another instruction, it will be again translated into the machine code and execute it. That means, execution of Interpreter is line by line. Here before executing any instruction, it has to translated into machine code, program execution will become slower. In fact, it does not take any time for compilation. Because it does not read all instructions of the program and translate it into the machine code, program will respond faster as it translates only the first line of the program and not whole program.

3.   Compiler: Compiler is also a translator, used in high-level languages like C, C++, Java and so on. Compiler reads whole program and translate into the machine level code before it starts execution. Once entire program is translated in the machine code, execution of the program will start. Here, before execution all instructions of the program are translated by a compiler into the machine code, program will be executed faster. In fact, it spends some time to compile (translate) the entire program. Compiler can translate one instruction of high-level language into several instructions of machine code. Therefore, it will reduce programming effort.

Complier enables high-level code to be shorter and more readable. Those programming languages, which is using compiler, execution of the program is started if and only if program source code is error free. If any programming line has error, compiler will generate compilation error message and execution of the program is terminated. Execution of the program will be started, all instructions written in the program is free from error.

Compilation process need some time to translate all programming instruction into the machine code, but when the execution is started, all instructions will be executed faster as all instructions are available in machine language.

Check Your Progress-3

1._____ is a translator which translate mnemonic symbols like 'ADD', 'SUB' etc in to the Machine language.

[A] Compiler   [B] Interpreter

[C] Assembler   [D] All of the above

2.Translator used to translate assembly code into machine language is _____.

[A] Assembler   [B] Compiler

[C] Interpreter   [D] None of the above

3.High-level C-Language use _____ to translate high-level code into machine language.

[A] Compiler   [B] Interpreter

[C] Assembler   [D] None of the above

4. _____ is a translator which translate and execute line by line.

[A] Compiler   [B] Interpreter

[C] Assembler   [D] None of the above

5. _____ is first translating whole program, and then it will start execution of it.

[A] Compiler   [B] Interpreter

[C] Assembler   [D] None of the above

1.3   INTRODUCTION TO C-LANGAUGE

In 1972, C-Programming language is developed by Dennis Ritchie, at AT&T Bell Laboratories (USA). Some basic concepts to develop C-Language is taken from two other programming languages, those are B-Language (designed by Ken Thomson in 1970 at Bell Laboratories) and 'Basic Combined Programming Language (BCPL)' (designed by Martin Richards in 1967.

1.3.1 Structure of C-Program

Program of C-Language is divided in to several parts, which are discussed below:

1.   Header File Section: C-Language supports functions. C-Language has 30 or more header files, which contain more than 145 library functions. First section of the C-Program permits programmer to include header files. Once the header file is included in the program, all library functions specified in that header file can be used in program. Extension for the header file is '.h'. For example, if we include header file '#include <stdio.h>' (Standard Input Output Header File) then we can use the functions like 'printf ()' and 'scanf ()'. If we have included header file '#include<conio.h>' (Console Input Output Header File) then we can use functions like 'clrscr () and 'getch ()'.

2.   Global Declaration Section: Variables declared outside of any functions are called global variables. Global variables reserve memory space in the memory when the execution of the program starts and remain into the memory till the termination of the program. Global variables can be accessible in any function of the program.

3.   Main Function: In C-Programming language, we can define many functions, but the function from which we want to start execution, should have function name 'main ()'. System will always find 'main ()' function and start the execution of the program. Because of the execution of main () function start by the system and main () function do not return any value to its caller (system), we have stated 'void' return type for the main function. '{'sign indicate start of the main function and '}'sign indicates closing of main () function.

```
┌─────────────────────────────────────────┐
│  ┌─────────────────────────────────────┐ │
│  │ Header File Section                 │ │
│  └─────────────────────────────────────┘ │
│  ┌─────────────────────────────────────┐ │
│  │ Global Declaration Section          │ │
│  └─────────────────────────────────────┘ │
│  ┌─────────────────────────────────────┐ │
│  │ /*   Comments  */                   │ │
│  │ void main ( )                       │ │
│  │ {                                   │ │
│  │     /*  Comments  */                │ │
│  │     Local Variable Declaration      │ │
│  │     Executable Statements           │ │
│  │ }                                   │ │
│  └─────────────────────────────────────┘ │
│  ┌─────────────────────────────────────┐ │
│  │ Return_Type User_Define_Function ( ) │ │
│  │ {                                   │ │
│  │     Local Variable Declaration      │ │
│  │     Executable Statements           │ │
│  │ }                                   │ │
│  └─────────────────────────────────────┘ │
└─────────────────────────────────────────┘
```
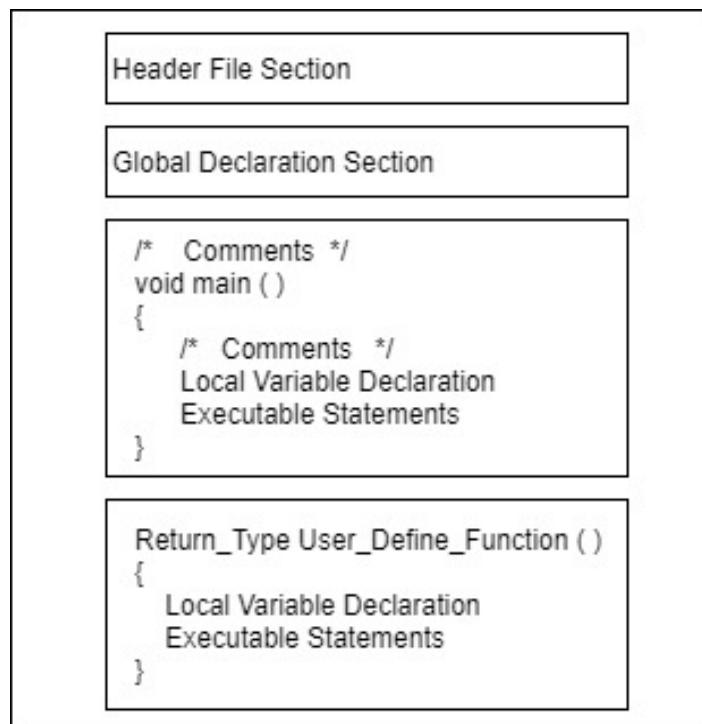
4. **Local Variables:** Those variables which we are declaring inside the function are called Local variables. Local variable has limited life time. When the function starts its execution, local variables declared in that functions are created, and when the function completes its execution at that time all the local variables of that particular functions are deleted automatically from the memory. Scope of the local variable is limited to that function only. We cannot access local variable of one function in other function. Unlike global variable (which are declared outside of any function) local variable can't be accessible throughout the program.

5. **Executable Statements:** Executable statements are those set of statements which are written to solve the specific problem. That can be IO statements, computational statements, conditional statements or looping statements.

6. **Comments:** Comments can be written anywhere in the C-Program. Comment statements are not translated by a compiler into machine codes as well as they will not be executed during execution process. Programmers are adding comments to their programs to increase readability of the program. Comments are of two types: (1) Single line comment which is denoted by // and (2) Multiline comment which is denoted by /*…………. */ For Example:

//This is single line comment

/* This is

      Multiline

      Comment */

7. **User Defined Function:** If we cannot find any particular library function then to solve any specific problem then we can write or design our own function to solve that problem. Such functions are known as user defined function (UDF). Many user defined functions can be created by the user into a program.

Check Your Progress-4

1.Full form of stdio.h is _____.

[A] Standard Input Output   [B] String Terminating Operations Input Output

[C] Store Input Output   [D] None of the above

2.printf() function is available in _____ header file.

[A] stdio.h   [B] conio.h   [C] math.h   [D] string.h

3.Scope of the _____ variable is limited to that function only.

[A] local   [B] global   [C] static   [D] extern

4. _____ is used for single line comment.

[A] /* */   [B] //   [C] %   [D] /?

5. _____ variables are declared outside of any function.

[A] local   [B] global   [C] register   [D] None of the above

1.3.2 Compiling and Executing a C-Program

To execute a c-program you need a software line Turbo-C, Borland-C or CodeBlocks to be installed in your system. Open the software which you have installed in your system and type a C-Language program code as shown below:

#include<stdio.h>
void main ()
{
  / This is my First C-Program */*
          printf ("Hello, World!!!");
}

Once the program is typed or written then save this file with .c or .cpp extension. Suppose we have saved that program with name hello.c. Now we know that the C-Language is a high-level language and we have written instructions in high-level language where we have used English words. Machine cannot understand this high-level code, so we need to compile or translate the high-level source code into the machine language code of the program. To compile the program, you need to press Alt+F9 in Borland C++ or Turbo C++ and if you are using code blocks then you need to press (Ctrl+Shift+F9). At the time of compilation if c- program has any error, then compiler will show error message. You need correct the instructions on which errors are there. Once the program compiled successfully (with no error), your source code will be converted into the object code. Compiler will generate Hello.exe file. Now to run this file you need to press Ctrl+F9 in Turbo C++ or Borland C++ (In case you are using CodeBlocks, you need to press F9). This will run your program in the Console screen window. The following output you can see in the console output windows.

OUTPUT:
Hello, World!!!

1.3.3 Rules of writing a C-Program

1. In C-Language program should have many functions, but in every program there must be a main() function, from where the execution of the program starts. C-Program always starts from main() function.
2. All C-Language statements ends with semi-colon (;). However, conditional statements like if or switch…case, or loops like while or for do not ends with semicolon.
3. Generally, in the C-program all the statements have to be written in the lower case. Upper case strings are generally used for symbolic constants.
4. The opening and closing braces written in the program should be balanced. i.e., number of opening braces and number of closing braces are same.
5. In C-Program multiple statements can be written in a single line. Two statements written in a single line are separated automatically by compiler by semi-colon. For example,
   X = Y + Z;
   P = 5 * X;

Can be written as,

X = Y + Z; P = 5 * X;

1.3.4 Advantages of C-Language

1. C-Language is a general-purpose programming language that can be used to develop application as well as system software. Very popular operating system UNIX developed in C-Language. The compiler used to compile C-Programs are also designed in the C-Language itself.

2. C-Language has significant data definition. It supports different datatypes like characters, integers, floating point numbers and strings.

3. C-Language also allows programmer to add assembly code.

4. C-Programs are highly portable. We can compile and execute the program written on one platform (hardware or operating system) can be execute on any other platform easily.

5. C-Language has powerful set of operators.

6. C-Language has 32 keywords, 30 header files and 145 built-in library functions.

7. C-Language is a structure-oriented programming language, which allows programmer to develop user-defined data types.

8. C-Language supports strings. Strings can be represented as an array of characters in C-Language.

9. C-Language supports powerful graphics, so that programmer can develop games or applications which has graphical user interface in C-Language.

10. C-Language allows program to write a program which controls different types of hardware.

Check Your Progress-5

1.Every C-Statement ends with _____.

[A] ;   [B] @

[C] _   [D] Enter

2.There are _____ header files are there in the C-Language.

[A] 32   [B] 30

[C] 64   [D] 128

3.There are _____ keywords are there in the C-Language.

[A] 48   [B] 30

[C] 32   [D] 125

4. _____ operating system is designed in the C-Language.

[A] Android   [B] Windows

[C] Solaris   [D] Unix

1.4   LET US SUM UP

In this chapter we have seen C-Programming Language is general purpose programming language can be used to develop any type of software that is application software, system software, software which supports graphics, or software which needs to control hardware. C-programming development can be done in any operating system, such as, windows, UNIX, LINUX or any other operating system. We have also discussed programming rules, how can we write a program, compile and execute the C program. We assume that after reading this chapter students will know everything about C-Language and know how to write simple C-Program and execute it. We hope the information served in this chapter will increase the interest of learning C-Programming Language in details.

1.5   GLOSSARY

1. conio.h : is a 'Console Input Output Header File', which contains functions like clrscr(), getch() etc.
2. stdio.h: is a 'Standard Input Output Header File', which contains functions like printf(), scanf() etc.

1.6   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1. [A] Binary
2. [B] Binary
3. [B] Low-level
4. [A] Machine

Check Your Progress-2

1. [C] Assembly
2. [C] Assembler
3. [B] High
4. Easier

Check Your Progress-3

1. [C] Assembler
2. [A] Assembler
3. [A] Compiler
4. [B] Interpreter
5. [A] Compiler

1.7   Assignment

1. Explain program structure of C-Language.
2. Discuss Machine language, Assembly language and C- Language.
3. List and discuss rules for C-Programming.

1.8   Activity

Write the following program, in the C-Language and write the output of it. Make the changes in the line 5, as given in the table (first column) below and write output of the changes into second column:

1. *#include<stdio.h>*
2. *void main ()*
3. *{*
4.   */* This is my First C-Program */*
5.   *printf ("Hello, World!!!");*
6. *}*

Change in line:6	Output
printf("Hello \n World);	
printf("Hello \n\n World);	
printf("Hello \t World);	
printf("Hello\b\b\b World);	

1.8   Case Study

List any 5 High-level programing languages and write at least 5 points about each programming language.

1.9   Further Reading

- "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
- "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
- BAOU Self-Learning Material, BCA Program. URL: https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

UNIT 2: UNDERSTANDING CONSTANTS, DATATYPES AND VARAIBLES

Unit Structure

2.0   LEARNING OBJECTIVES

In this unit, we will discuss about the basics of C required for beginners to understand this language.

After working through this unit, you should be able to:

• Comprehend the concept of constants and variables

• Elaborate on data types used to make a program

• Interpret variables

• Describe Constants and Keywords

2.1   INTRODUCTION

Programming language C is a most powerful programming language, which learners find easy to program. C-Language is also called the middle-level language, as it is closer to both, machine and user. In this unit, we will learn about the Constants, Variables and Datatypes which are essential to write any program. After reading and understanding this unit, you will definitely be able to write and develop your own small programs.

2.2   CONSTANTS

Constants are those values which may not be changed during the execution of a C-Program. C-Language supports Integer, Character, String and Floating-point constants. Integer constants represent numbers without decimal point, floating constants represent decimal numbers. Integer and Float together are denoted as numeric type constants. Numeric constants are following these rules:

• Spaces and commas cannot be included within the constant.
• The constant can be preceded by a minus (-) sign.
• The value of constant cannot exceed specified maximum and minimum bounds, which is varies from compiler to compiler.

2.2.1 Integer Constants

An integer constant contains only digits. Integer constants are of three types: Octal, Decimal and Hexa-Decimal.

• A decimal integer constant should have any combination of digits from 0 to 9, they can be negative or positive with - or + For example, 157, -563, 9584, +79, 0 etc.
• Hexa-Decimal integer constant must begin with either 0X or 0x and can be mixture of digits between 0 to 9 and A to F (upper or lower case) which represent the numbers 10 (A) to 15 (F). For example, 0x2, 0xBC5, etc.

- The largest integer value that can be stored in constant, is machine dependent. For 16-bit of machines it is 32767, and for 32-bit machines it is 2147483647. You can also store larger integer constants on these machines by adding qualifiers such as U, L and UL to the constants. For example, 762548U (Unsigned integer), 982453762UL (Unsigned Long integer), 569189L (Long integer).

2.2.2 Floating Point Constants

These constants are decimal (base=10) numbers that contain either a decimal point, an exponent or both. Example of some valid floating-point constants are:

8.7, 0.625, 1.3526E + 6, 325247e13

If an exponent is present, it shifts the position of the decimal point. If the exponent is positive, it shifts the decimal point to the right-hand side and if the exponent is negative, it is shifts to the left. They have a larger range than integer constants. The magnitude range of the floating-point constant is from 3.4E-38 to 3.4E+38. These constants are normally denoted as double precision quantities. Each floating-point constant typically occupies 8 bytes of memory.

2.2.3 Character Constants

Character means any one symbol from your keyboard, which is enclosed in single quotation marks. For example, 'A', 'e', 'I', 'o', 'U' etc.

2.2.4 String Constants

String Constants are consisting of successive characters enclosed in double quotation marks. For example,

"Welcome", "Hello", "Computers", "B.Sc-IT" etc.

The compiler automatically places a null '\0' (which is NULL character) at the end of every string constant, as a last character within the string. This character is not visible when the string is displayed. Character constant 'P' and string constant "P" are not equal.

Check your progress 1

1.From the given options _____ is not a type of constant.

[A] Integer   [B] String

[C] Image   [D] Floating-point

2.In 16-Bit machine, _____ is the largest value can be stored in Integer constant.

[A] 32767   [B] 65536

[C] 255   [C] 2147483647

3.Identify a valid statement from the given below:

[A] It doesn't store in Memory   [B] Replaced physically at compilation time

[C] It never changes its value   [D] We can overwrite the value of it

2.3   VARIABLES AND DATA TYPES

Variable is a kind of storage, into the computer's memory to store the data. We can refer the data using name of the variable.

Variable Naming Convention

To use variables in your C programs it must follow rules:

- The variable name should be combination of characters, digits and underscore ('_').

- Variable name must start with (first letter of variable name) must be either letter or underscore.

- Name used for variables are case sensitive, that means variable names counter, Counter and COUNTER will be treated as separate variables.

- Keywords (They are the reserved words for programming language) like, char, int, float, long, if, switch, case, for, include, while etc. cannot be name of variables.

2.3.1 Data Types in C

The C language supports very rich set of data types. In the table given below, we have listed some of the basic data types available in the 'C' programming language, which will be useful to declare variables.

VARIABLE TYPE	KEYWORDS	BYTES	RANGE	FORMAT REQUIRED
Character (signed)	Char	1	-128 to +127	%c
Integer (signed)	Int	2	-32768 to +32767	%d
Float (signed)	Float	4	-3.4e38 to +3.4e38	%f
Double	Double	8	-1.7e3.8 to +1.7e3.8	%1f
LongInteger (signed)	Long	4	2,147,483,648 to 2,147,438,647	%1d
Character (unsigned)	Unsigned char	1	0 to 255	%c
Integer (unsigned)	Unsigned int	2	0 to 65535	%u
Unsignedlong	unsignedlong	4	0 to 4,294,,967,295	%1u
Longdouble	longdouble	10	-1.7e932 to +1.7e932	%Lf

Character

The data type, Character is used in the C-Language to store or to access one character. "char" keyword is used to declare, any variable of type character. A variable declared with "char" datatype can store one symbol available on the keyboard, and variable of this type reserves 1 Byte of memory. For example:

char x;

x = 'B';

printf ("%c", x);

In the above source code, first statement declares variable 'x' of type character which reserves 1 Byte (8 bits) of memory space. Second statement initialize (assign a value) character value 'B' to variable 'x'. Third statement of the source code will read the value of variable 'x' and print it on the Console screen using printf() statement. In the printf() function first we have mention, format specifier "%c" to denote, the value of variable 'x' to be printed in character format (make sure all data available in the main memory in the form of binaries). After format sting "%c", we need to mention the name of the variable 'x', so that computer will read the value from variable 'x', and printf() function can print the value on the Console screen.

Integer

Datatype "int" is used to store any Integer value into main memory, in C-Language. Variable declared with type "int" will occupy 2 Bytes (=16 Bits) of memory space. For example:

int num;

num = 51;

printf ("%d", num);

In the above source code, we have declared a variable 'num' of type int. We have stored value 51 in it, as we have initialized (assign) num variable with value 51 in the 2nd line. In the third line of source code we have printed the value of num variable using printf() statement by using format string "%d". To print any variable of type int you can use either "%d" (decimal) or "%i" (integer) format string. To print the value of num variable in octal number system %o and to print the value of num variable in Hexa-Decimal number system, %x format string is used:

```
#include<stdio.h>
void main()
{
    int num;
    num = 240;
    printf("Number with Decimal format is:%d", num);
    printf("\nNumber with Octal format is:%o", num);
    printf("\nNumber with Hexa-Decimal format is: %x", num);
}
```

OUTPUT:

Number with Decimal format is:240
Number with Octal format is:360
Number with Hexa-Decimal format is: f0

Integer variable can be used to store positive numbers, negative numbers and 0. For example, in the above example, if we initialize number variable with 35 by writing instruction num = 35. We can, also store negative number or 0 in the variable by writing following statements, num = -35 or num = 0.

Variable of datatype "int" will reserves 2 Bytes (= 16 Bits) of space in the memory. From total 16 Bits, 1 bit will represent positive or negative sign. Which means to store positive number sign bit will 0 and if sign bit is 0 that means the number is negative. Remaining 15 bits are used to represent a value (magnitude) of that number. That means maximum value can be 2^{15} = 32768. So that, we can say -32768 as a smallest number and 32767 as the largest number can be stored in the variable of type 'int'. We are using one combination to represent a number 0 that is the reason we cannot represent 32768 (positive) number. So, the range of a variable of datatype 'int' is: -32768 to 32767.

Unsigned Integer

Unsigned Integer datatype is used to represent only positive numbers. Therefore, we don't need sign bit. Unsigned variable cannot represent negative numbers. In the case of 'unsigned int' all 16 bits are used to represent a number so variable can represent 2^{16} = 65536 numbers. If we start from 0 as a first number, it can go up to 65535. Therefore, the range of the unsigned variable is 0 to 65535. It is printed or scanned using format string %u. Consider the following example:

unsigned int n;

n = 65000;

printf ("%u", n);

Float

To access(read) and to store(write) any real (floating-point) value, float data type is used. C-Language's float data type occupies 4 bytes (32 bits) in the memory.

To declare any float variable:

float percentage;

float average;

To assign or to store some real value:

percentage = 78.20;

average = 123.45

The range for float variable is -231 to 231 -1, means -3.4e38 to +3.4e38

Long

To access(read) and to store(write) any larger integer value (more than 32767), long data type is used. C-Language's long data type occupies 4 bytes in the memory.

To declare any long variable:

long int account1;

long mobil_no;

To assign or to store some long value:

account1 = 565232;

mobile_no = 1234567890;

Check your progress 2

1.varaible of Unsigned int variable can store a value ranging from _____.

[A] -128 to 127   [B] -32768 to 32767

[C] 0 to 65535   [D] 0 to 255

2. Largest value can be accommodated in short integer variable is _____.

[A] 32767   [B] 65535

[C] 255   [D] 250532247

3.Variable declared with datatype float will occupy _____ space in memory.

[A] 1 Bytes   [B] 2 Bytes

[C] 4 Bits   [D] 4 Bytes

2.4   CHARACTER SET

Set of characters which can be used to write a C-program is called character set for C-Language. In C-Language we can use upper case letters from A to Z, the lowercase letters from a to z, digits from 0 to 9 and some special characters to form basic program elements.

The C Character set is given below

- Uppercase Letters (A-Z)
- Lowercase Letters (a-z)
- Digits (0-9)
- Special characters like, #, (,), {, }, < , +, *, / etc.

C-Language also uses some combinations of characters such as \n, \t, \b to represent special conditions such as newline, horizontal tab and back space characters respectively. These character combinations are known as escape sequences. Each escape sequence represents a single character.

2.5   C TOKENS

Each word used in a C-program is called token. Different types of tokens are used in the C-Program. These tokens are discussed as follows:

2.5.1 Keywords

Words having specific programming meaning are called keywords. Keyword must not be used as variable names. There are 32 keywords in the programming language:

auto	break	case	char
const	continue	default	do
double	else	extern	float
for	float	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

2.5.2 Identifiers

Identifiers are names, given to different elements of a program such as variables, constants, functions, arrays and union. They may consist of letters and digits which can be in order but they must start with a letter, using both uppercase and lowercase letters, but should have different meanings. For example, Madam and madam are not same (C-Language is case sensitive). The underscore character ('_') can also be included and is treated as a letter. Example of some valid identifier are given below:

B05, Computer_07, rate_of_int, BAOU etc.

Check your progress 3

1. From the given below _____ is not a keyword.

[A] int   [B] string

[C] float   [D] break

2. C-language character set includes _____.

[A] Alphabets   [B] Numbers

[C] Special symbols   [D] All of the above

3. From the given below _____ is not a valid identifier.

[A] 123abc   [B] Abc123

[C] Abc_123   [D] baou

2.6   DECLARATION OF VARIABLES

To declare the variable, datatypes are used, which indicates what type of data is being stored in the variable. In the C-language all variables must be declared before they seem in the executable statements.

The syntax for the same can be written as

<data type> <variable name>;

An array variable (which can store multiple elements of same datatype) must be followed by a pair of square brackets with a positive integer specifying the size of the array. For example,

int cntr, num, i;
float avg, per;
int array [10];
char string [5];

Thus, cntr, num and i are integer variables, avg and per are float variables, array is an integer array whose size is 10 and string is a character array with size 5.

2.6.1 Assigning Values to Variables

If the values of the variables are known, then you can present the declarations as given below:

int cntr = 5;
char ch = 'A';
float avg = 51.36;

From the above declaration it is clear that variable cntr is of integer type and initialized with value 5, ch is character type variable initialized with 'A' and avg is of type float, initialized with value 51.36.

A character array also known as string, may also be initialized within a declaration as shown below:

char university [ ] = "BAOU";
in the above statement we are declaring university as an array of type character (String), which will store 5 characters 'B', 'A', 'O', 'U', '\0'. The word "BAOU" needs 4 characters and at the end BAOU, system will store '\0' (NULL) character which will specify end of the string.

The above declaration can also be written as:

char university [4] = "BAOU";
Here variable 'university' is a string (array of char datatype). The size of the array i.e., 4 must be specified in square brackets correctly, at the time of its declaration. Because of the small size, the NULL character cannot be stored, and that is the reason, when you print this array, you will get some extra characters(junk) will be printed after BAOU.

2.7   DEFINING SYMBOLIC CONSTANTS

Symbolic constants are also identifiers, termed as sequence of characters. These identifiers may represent as numeric constant, character constant or string constant. Symbolic constant allows a name to appear in place of a numeric or character or string constant. Each occurrence of symbolic constant is replaced by its corresponding character sequence or value at the time of compilation. They are defined in the beginning of a program. For example:

 # define MAXIMUM 20

Here, MAXIMUM represents a symbolic constant name and 20 represents the value associated with the constant. Since it is a declaration of the symbolic constant, it does not end with a semicolon. If you include a semicolon at the end, this semicolon would be treated as a part of the numeric character or string constant that is substituted for the symbolic name.

2.8 LET US SUM UP

In this unit, we have:

- Explained about the character set of a C program
- Elaborated on tokens, keywords and identifiers
- Described the various types of operators
- Talked about evaluating expressions
- Discussed about managing input and output operators

2.9   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1.   [C] Image
2.   [A] 32767
3.   [D] we can overwrite the value of it

Check Your Progress-2

1.   [C] 0 to 65535
2.   [A] 32767
3.   [D] 4 Bytes

Check Your Progress-3

1.   [B] string
2.   [D] All of the above
3.   [A] 123abc

Check Your Progress-4

1.   [B] abc+123
2.   [C] datatypes
3.   [C] Abc_123

Check Your Progress-5

1.   [A] Constants
2.   [D] \0
3.   [A] #define

2.10   GLOSSARY

Constants are identifier, which are usually initialized at the time of declaration. Once initialized user cannot change the value of it.

Keywords are the reserved words, having specific meaning in the programming languages.

Variables are identifiers, which allows user to store their data. User can change the value of the variable.

2.11   Assignment

1. What is constant? How it differs from the variables.
2. List and explain all datatypes available in the C-Language.
3. Elaborate on the variable initialization methods.

2.12   Activity

Write the following program, in the C-Language and check what happen when we run this program. Comment line 7, run the program again and see what happen. Write and justify outcome in both the cases.

```
1.  #include<stdio.h>
2.  #define MAX 10
3.  void main ()
4.  {
5.    int x=10;
6.        x = x + 5;
7.        MAX = MAX +5;
8.        printf("\n MAX is: %d, and X is: %d", MAX,x);
9.  }
```

2.13   Case Study

- Make a table, having all datatypes, their size, range, and format strings. Write a C-Program in which all variables of different types are present. Initialize and print all variables.

2.14   Further Reading

- "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
- "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
- BAOU Self-Learning Material, BCA Program. URL: https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

UNIT 3   OPERATORS AND EXPRESSIONS

3.0   LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Understand the types of operators

- List special operators

- Understand arithmetic expressions

- Interpret operator precedence and associativity

- List mathematical functions

3.1   INTRODUCTION

Operators are special symbols, which act upon operands (data values). For example, if we write "a * b" then "a * b" is known as expression, "*" is an operator and "a" and "b" are operands (data).

Operators are used to form expressions by joining variables, constants, elements of an array as studied earlier. We will study Unary and Binary operators, Arithmetic, Relational, Logical, Bitwise, Assignment and Conditional operators.

Operands are data values on which operators perform arithmetic or logical operations. Unary operation is performed on single data element, whereas Binary operation can be performed on two data values. Unary operators permit only a single variable as an operand. In this chapter, we will discuss about various types of operators and their evaluation process in the expressions.

3.2   OPERATORS AND EXPRESSIONS

Operator are used to perform operations on operands. List of different types of operators are:

(1)	Arithmetic	(2)	Relational	(3)	Logical
(4)	Assignment	(5)	Increment/Decrement	(6)	Conditional
(7)	Bitwise	(8)	Special		

3.2.1 Arithmetic Operator

In C-Language there are five operators are there which are in the category of Arithmetic operators.

1. Operator '+' for Addition
2. Operator '-' for Subtraction
3. Operator '*' for Multiplication
4. Operator '/' for Division
5. Operator '%' for Modulo or Remainder

Consider a case, where we have two operands or variables X=17 and Y=3, If we apply the arithmetic operators as discussed above, we can get the following results.

Expressions	Value	Explanation
X+Y	18	X+Y = 17+3 = 20
X-Y	12	X-Y = 17-3 = 14
X*Y	45	X*Y = 17*3 = 51
X/Y	5	X/Y = 17/3 =5
X%Y	0	X%Y = 17%3 =2 As 17 is NOT divisible by 3 and 3*5 =15, Therefore, 2 will be a Remainder.

3.2.2 Relational Operator

In C-Language there are 6 relational operators are there:

1. Less than (<)
2. Less than or Equal (<=)
3. Greater than (>)
4. Greater than or Equal (>=)
5. Equal to (==)
6. Not equal to (!=)

All Relational operators have same precedence (priority). The Associativity of Relational operators are from Left-to-Right. When in any expression two operators are used having same precedence then Associativity rule is used. Relational operators are always producing logical (Boolean) values as a result, such as (TRUE which can be interpret as 1 or FALSE which can be interpreted as 0).

For example, expression 6 > 4 is TRUE, it is interpreted as 1, and expression 19>28 is FALSE will be interpreted as 0.

Suppose, that x, y and z are integer variables whose values are 2, 3 and 5 respectively. Evaluation of different relational operators with these data values are shown in the following table.

Expression	Result	Interpretation Value
x<y z<y	True False	1 0
(x+y)<=z y<=z	True False	1 0
(x+y)>z (x+z)>y	False True	0 1
(x+y)>=z y>=(x+z)	True False	1 0
y==4 (x+y) == z	False True	0 1
y!=4 (x+y)!=z	True False	1 0

3.2.3 Logical Operator

In C-Programming language there are three logical operators are there, those are:

[1] AND (&&) [2] OR (||) and   [3] NOT (!)

These operators are called, logical AND, logical OR and logical NOT respectively. Similar to the relational operators, logical operators are also producing Boolean values (1 or 0, TRUE or FALSE).

The following truth table shows the results of the combined expressions using different logical operators depending on different values of the expressions E1 and E2:

Expressions		Result			
E1	E2	E1 && E2	E1 || E2	! (E1)	! (E2)
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Any positive or negative values (except 0) is considered as 1. For example, if there are 3 variables x=-7, y=0 and z are there, and if we write z=x&&y, we get z=0 and if we write z=x||y, we get z=1. Here, x is non-zero, so it will be considered as 1. Now, in AND operation x && y = 1 && 0 = 0. Similarly, in x||y = 1||0 = 1. Make sure, we will get x && y = 1, if and only x and y both are non-zero, and we will get x || y=0, if and only if x and y both are 0.

3.2.4 Assignment Operator

In C-Language, assignment operators are used to assign the value to an identifier.

Syntax: Identifier = Expression OR Constant

For example, x=5. In this example we are assigning constant value 5 to variable x or we can say we are initializing variable x with value 5. Statement x+=1, means x=x+1. Here, value x will be incremented by 1. Statement x*=5, is same as x=x*5. The other assignment statements can be: x/=y, x%=y and so on.

3.2.5 Increment/Decrement Operator

Unary Operator takes only one operand. Increment (++) and Decrement (--) operators are coming in this category.

The operand has to be a variable and not a constant. Thus, the expression x++ is valid, whereas 51++ is invalid. Operators increments or decrements the value by 1. Expression x++ increments the value of a by 1 and the expression x-- decrements it by 1.

Statements x++ is called post-increment and ++x is called pre-increment. For example: x++; and ++x; gives same result, it increments value of x by 1.

However, prefix and postfix operators have different effects when they are used in association with some other operators. For example, if we assume the value of x variable to be 7 and then in the statement y=++x, variable x will be incremented by 1 first (x=8) and then value of x is assigned to variable y. As a result, we will get x=8 and y=8.

On the other hand, in the case of execution of the statement y = x++; the value of variable x is assigned to y first. Therefore, y=7 and then the value of variable x is incremented by 1. So, x will be 8. As a final result, x=8 and y=7. So,

  y = ++x; means x=x+1 and then y=x (pre-Increment),

  y=x++; means y=x and then x=x+1 (post-Increment).

3.2.6 Conditional Operator

Conditional operators are also known as Ternary Operators (?:). Conditional operator has Condition, True and False parts.

Syntax:
(Condition) ? True: False
Expression1 ? Expression2: Expression3

Here, if a condition is evaluated as true then the value of Expression2 will return and if condition is evaluated as false then Expression3 will return.

For example,

printf("%d", (x<5) ? 10:100);

If value of variable x is less than 5, then condition is evaluated as true and 10 will be return to printf() function. In this case printf() function will print 10 on the console scree. But if, the value of variable x is greater than 5 then condition will becomes false and 100 will be printed by a printf() function.

For example, to find the greater value from variable i and j we can write:

printf("%d", (i>j) ? i:j);

Similarly, to find greatest value from given three variables i, j and k we can write:

printf("%d", (i>j) ? (i>k)? i: k : (j>k)? j: k);

3.2.7 Bitwise Operator

Some applications require the manipulation of individual bits within a word of memory. Assembly language or machine language is normally required for operations of this type. However, C contains several special operators that allow such bitwise operations to be carried out easily and efficiently. These bitwise operators can be divided into three general categories: the complement operator, the logical bitwise operators and the shift operators. „C" also contains several operators that combine bitwise operations with ordinary assignment.

Logical bitwise operators

There are three bitwise operators: bitwise and (&), bitwise exclusive or (^) and bitwise or (|). Bitwise operators are type Binary operators; hence they need two operands. The operations are carried out independently on each pair of corresponding bits within the two operands. Thus, the least significant bits (the rightmost bits) within the two operands will be compared and then the next bit and so on.

X	Y	X&Y	X^Y	X|Y
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

The associativity for each bitwise operator is left-to-right.

For example, if we assume x is an integer variable and if we write the statement, x=61 &
78 then we can get the value 12 as shown below:

Binary of 61: 0000 0000 0011 1101
Binary of 78: 0000 0000 0100 1110
Binary of 12: 0000 0000 0000 1100

Similarly, if we write x =61 | 78 then we get value x=117 as shown below:

Binary of 61:   0000 0000 0011 1101
Binary of 78:   0000 0000 0100 1110
Binary of 127: 0000 0000 0111 1111

Bitwise Shift operators

There are two Bitwise shift operators, shift left (<<) and shift right (>>). Both shift
operators are of type Binary operators (needs two operands). The first is an integer-type operand
that represents the bit pattern to be shifted. The second is a positive or unsigned integer that
indicates the number of displacements (i.e., how many bits to be shifted in the first integer
number). This value should not be exceeding the number of bits associated with the word size
of the first operand. The left-shift operator (<<), shifts the number of bits as the value of second
integer, of a binary of first integer at left side. Here after shifting right most bits will be vacant
as they are shifted to left side, which are filled by 0. For Example, if variable A=35.

A = 0000 0000 0010 0011 [Left most 3-bits will be removed after <<]
After execution of B=A<<3 the value of the variable B is:
B= 0000 0001 0001 1000= 280 [Right side 3 bits with 0 is added]

Make sure, in this example system will remove first 3 bits of variable A from the left-
hand side and add 3 bits from the right-hand side (underlined in value of variable B). Similarly,
in the case of right shift (>>), 3 bits will be removed from right side of variable A and 6-bits
will be added from the left side for variable X.

A= 0000 0000 0010 0011
After execution of B=A>>6 the value of variable B is:
 X=0000 0000 0000 0100= 4

We, can say that from the variable A, 3 bits from the right side, (011) will be removed
and 3 bits (000) will be added at the left-hand side, which will be stored in the variable B.

Check your progress 1

1.if z=17%3 then what will the statement printf("%d",z);

[A] 3   [B] 5

[C] 4   [D] 2

2. if a=78 and b=45 then what will be the value of a&b?

[A] 12   [B] 45

[C] 78   [D] 111

3. for two integer a and b, what statement printf ("%d", (a>b)?b:a); will print?

[A] greater number   [B] smaller number

[C] value of variable x   [D] value of variable y

3.3   SPECIAL OPERATORS

3.3.1 Size of Operator

The sizeof is a compile time operator, which returns the size of an operand, constant or datatype in bytes. For example: printf("%d", sizeof(int)); statement will 2 in Borland C or Turbo C. sizeof() is also used to determine the length of array and structures and to allocate memory space dynamically during execution of a program. For Example,

1. sizeof(abc);
2. N1=sizeof(long int);

3.3.2 The Comma Operator

Comma operator can be used as a conjunction in the For loop. Using comma operator, if we want to initialize two variables in the initialization part of the for loop, then comma operator is used. For example,

for (i=0, j=0; i<n; i++)

Here i and j both variables are initialized by 0. These two statements i=0 and j=0 is separated by the comma operator.

3.4   ARITHMETIC EXPRESSIONS

An expression represents a data item, such as a number or a character. Logical conditions and operators and can be used in the expressions.

For example,

a1+a2 //Addition operator is used

x1=y1 //Assignment operator is used

p1=q1+r1 //Both addition and assignment operator is used

2.4.1   Evaluation of Expressions

To evaluate any arithmetic expression assignment operator is used. For example:

Variable=expression;

In the above syntax, variable is any valid name in as per C-Language naming rules. When the statement to be evaluated is encountered, the expression is evaluated first and then it replaces the previous value of the variable on the left-hand side. All variables used in the expression must declared properly and be assigned values before evaluating it. For example,

d1= a1+b1*c1;

a1=b1/c1-d1;

In the case of parenthesis, expression within the parenthesis will be evaluated first:

Suppose, x=18, y=10, z=4, then the expression,

a=x-y/(2+z) *(2-1); will be evaluated as:

Expressions within parenthesis is evaluated first and then the rest of the arithmetical operations are performed. So, the given expression can be evaluated as: x=18-10/(2+4) * (2-1);

=18 -10/6*1 =18-10/6=18-1.6667=16.33333

3.4.2 Precedence of arithmetic expressions

If the parenthesis is not there then the arithmetic expression is evaluated by following associativity rule from Left to Right. In the Arithmetic operator Multiplication (*) and Division (/) is evaluated first. Then Modulo operation (%) is performed and finally addition (+) and subtraction (-) is performed.

Rules for expression evaluation:

- If parenthesis is given in the expression, then content within the parenthesis is considered as a sub-expression and it will be evaluated first.

- If nested parenthesis is there then inner most parenthesis is evaluated first.

- To determine the order of application of operators for evaluating subexpressions the precedence rule is applied.

- Associativity rule for the arithmetic operators is from left to right.

- If the expression has same type or same priority operators then associativity rule is applied.

3.4.3 Types of conversions in expressions

We can mix the types of values in your arithmetic expressions. Char types will also be treated as int in the expression. Otherwise, here types of different sizes are there, then the result will usually be of a larger size, so if an expression has a float and a double variable then it would produce a double result. Where an integer and real types of conversions is mixed-up, then the result will be a double. There is usually no trouble in assigning a value to a variable of different types. The value will be preserved as expected except where;

- If the storage variable is too small to hold the expression's value. It will be corrupted.
- The float value is assigned to an integer variable. The value is rounded down and fraction point is removed. This is often done deliberately by the programmer.
- Values passed as function arguments must be of the correct type. Because automatic conversion cannot take place which can lead to corrupt results. We can also use method called type-casting which temporarily disguises (consider) a value as a different type.

e.g. The function sqrt() finds the square root of a double.

int num= 256;

int root;

root = sqrt((double) num);

The type-cast is made by putting the bracketed name of the required datatype just before the value, like (double) in this example. The result of sqrt((double) num); is considered to be a double, but this is automatically converted to an int after assignment to root varaible.

C permits different types of conversion in data type. Following are the rules for types of casting conversion:

- Float variables are converted to double.

- Char or short (signed or unsigned) are converted to int (signed or unsigned).

- If anyone operand is a double, the other operand is also converted to a double and that is the type of result; otherwise

- If anyone operand is long, the other operand is treated as long and that is the type of the result.

- If anyone operand is of the type unsigned, the other operand is converted to unsigned and that is the type of the result.

- Otherwise, the only remaining possibility is that both operands must be int and that is also the type of the result**.**

Check your progress 3

1.if int a=5,b=2; then what will be output of printf("%f",a/b);

[A] 2.000000   [B] 1.5

[C] 2.500000   [D] 2.5

2 if int a=5,b=2; then what will be output of printf("%f",(float)a/b);

[A] 2.000000   [B] 1.5

[C] 2.500000   [D] 2.5

3. if int a=3; then what will be output of printf("%.3f",a/2.0);

[A] 1.000000   [B] 1.5

[C] 1.500000   [D] 1.500

3.5   OPERATOR PRECEDENCE AND ASSOCIATIVITY

Precedence is priority, which assigned to different operators. Depending upon priority operators will be chosen and evaluated during execution, when more than one operator is there in the expression. If there are two or more operators in the expression having same precedence then the expression will be evaluated from Left-To-Right or Right-To-Left which is called an associativity of an operator.

For Example:

int x=3+5*7-(9-2)+12/3 statement will be evaluated in the following manner
3+5*7-(9-2)+12/3 [First preference will be given to bracket]
3+5*7-7+12/3 [Now, preference will be given to multiplication and division]
3+35-7+4 [Now, + and – has equal priority so as per associative law L to R]
38-7+4
31+4
35

Computer is always evaluating expression by looking to the priority of the operator. Those operators which has highest precedence, will be evaluated first. Precedence is nothing but the priority of the operator. Therefore, the study of the operator is important. We always need to write the expressions in such a way that it should be evaluated by the computer system correctly. In the following table we have listed all the operators, as per their priority and associativity.

Priority	Operators	Operations	Associativity
1st	()	Function call or parenthesis bracket	Left to right
	[]	Array expression or square bracket	
	->	Structure operation with pointer	
	.	Dot operator of structure	
2nd	+	Unary plus	Right to left
	-	Unary minus (Negation of e.g. -5)	
	++	Increment operator	
	--	Decrement operator	
	!	Not operator	
	~	One's complement	
	*	Pointer operator	
	&	Address operator	
	sizeof	Size of operator	
3rd	*	Multiplication	Left to right
	/	Division	
	%	Modulo operator	
4th	+	Addition (Binary)	Left to right
	-	Subtraction (Binary)	
5th	<<	Left shift	Left to right
	>>	Right shift	
6th	<	Less than	Left to right
	<=	Less than or equal	
	>	Greater than	
	>=	Greater than or equal	
7th	==	Equal	Left to right
	!=	Not equal	
8th	&	Bitwise AND	Left to right
9th	^	Bitwise XOR	Left to right
10th	|	Bitwise OR	Left to right
11th	&&	Logical AND	Left to right
12th	||	Logical OR	Left to right
13th	?:	Conditional operator	Right to left
14th	=, *=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=	Assignment operator	Right to left
15th	,	Comma operator	Left to right

Check your progress 4

1.Choose the operator having highest precedence from the given below.

[A] *   [B] >

[C] &   [D] %

2 From the given operatorS, which is a logical operator?

[A] ++   [B] &

[C] &&   [D] ?:

3.6   MATHEMATICAL FUNCTIONS

The mathematical calculations can be done by including the header file <math.h>. A common source of error usually occurs when the<math.h> file is not included. Given below are some mathematical functions, which can be used in a C program as and when desired.

double pow(double a, double b)- Computes a raised to the power b

double sqrt(double a)- Computes the square root of a

double sin(double a)- Computes sine of angle in radians

double sinh(doublea)- Computes the hyperbolic sine of a

double tan(double a)- Computes tangent of angle in radians

double tanh(double a)- Computes the hyperbolic tangent of a

double log10(double a)- Computes log to the base 10 of a

double modf(double a, double *int ptr)- Breaks a into fractional and integer parts

double log(double a)- Computes log(a)

double exp(double a)- Computes exponential of a

Check your progress 5

1.Output of the statements x=sqrt(25); printf("%d",x);,will be _____.

[A] 625   [B] 25

[C] 0   [D] 5

2 In order to use function sin(), _____ header file has to be included.

[A] stdio.h   [B] conio.h

[C] math.h   [D] string.h

3. What is the output for the statements x=pow(2,3); printf("%d",x); ?

[A] 2   [B] 8

[C] 4   [D] 3

3.7   LET US SUM UP

In this unit, we:

- Elaborated on types of operators

- Studied about expressions

- Talked about expressions

- Studied about operator precedence and associability

- Discussed about mathematical functions

3.8   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1.   [D] 2
2.   [A] 12
3.   [B] smaller number

Check Your Progress-2

1.   [C] 4
2.   [B] 2
3.   [A] , comma

Check Your Progress-3

1.   [A] 2.000000
2.   [C] 2.500000
3.   [D] 1.500

Check Your Progress-4

1.   [A] *
2.   [C] &&

Check Your Progress-5

1.   [D] 5
2.   [C] math.h
3.   [B] 8

3.9 Glossary

FOR Statement	This statement is used to execute the same set of statements a number of times.
Goto Statement	It is used to send the control of program from one part to another.
If Statement	It checks the given condition and executes the statements accordingly.
Looping	It is a process of executing the same set of statements a number of times.
Switch Statement	Depending upon the value specified with it, switches to the specified case statement.
While loop	Also called Entry-Controlled Loop, used to execute the same set of statements a number of times.

3.10   Assignment

1. What is typecasting? Explain it with an example.
2. List all relational operators and explain it with an example of each.
3. What is ternary operator? Explain it with an example.

3.11   Activity

Write the following program, in the C-Language and check the values of x and y variables. Now change the statement 6 from y=x++ to y=++x. Note the output again and justify the difference between previous and current output.

```
1.    #include<stdio.h>
2.    #define MAX 10
3.    void main ()
4.    {
5.            int x=10,y;
6.            y=x++;
7.            printf("\n X is: %d, and Y is: %d", x,y);
8.    }
```

3.12   Case Study

- Write a program to find greatest number from given 3 numbers using conditional operators.

3.13   Further Reading

- "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
- "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
- BAOU   Self-Learning   Material,   BCA   Program.   URL: https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

UNIT 4 INPUT OUTPUT OPERATIONS

Unit Structure

4.0   LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Understand the method of accepting an input
- List the functions to accept a character
- Recall the functions to display the entered character
- Understand the formatted input and output methods

4.1   INTRODUCTION

Generally, programs are design to take input values from the user, program will process on these values and produces information or output. In C-Programming language, we can assign the static value to the variable (i.e., int x=15;), or we can take the value from the user using scanf() function. We can also print the result on the standard console output window using printf() statement.

To take the input values from the user, generally we use keyboard, and to show the output to the user we use standard console output screen.

There are two types or functions available in the C-Programming language to perform Input/Output operations. [1] Some function can be used to perform Input/Output for any kind or data. For example, printf() function can print any type of data like char, int, float, double etc. This type of function is called formatted function. [2] Some functions are designed to perform specific task, like in C-Language function getchar() is used to take single character from the user, which can not be used to accept integer value from the user. This type of function is called unformatted function. In this chapter we will study about formatted and unformatted Input/Output functions in greater detail.

4.2   MANAGING INPUT/OUTPUT OPERATIONS

C-Language provides many functions to perform Input/Output operations. Functions discussed in this section is used to accept single character or to print single character. They can also be used to accept string (group of characters) by combining it with loops.

4.2.1 Reading a character

By using function getchar() we can accept single character using standard input device (like keyboard).

In general terms, a reference to the getchar() function is written as.

character variable = getchar( );

E.g., char choice;

choice=getchar( );

The first statement declares character variable choice. The second statement invoke a function getchar() which takes a input character from the user by standard input device (keyboard) and store it in the choice variable. The getchar() function can also be used to read multicharacter strings.

4.2.2 Writing a character

To display single character, stored in the variable from particular memory location, on the console output screen (i.e. written out of the computer), C library function putchar() is used. It communicates a single character to a standard output device. The character being communicated will normally be represented as a character-type variable. It must be passed as an argument to the function, enclosed in parentheses following the word putchar. In general, a reference to the putchar() function is written as:

putchar (char variable);

If 'C' program contains the following statements, then:

char choice= 'a';

putchar(choice);

The first statement declares that choice is a character type variable. The second statement causes the current value of choice, to be communicated to the standard output device where it will be displayed. The putchar() function can be used to output a one-dimensional array of type character. Each character from an array, can be written separately by using a loop.

Check your progress 1

1.To read a character from the user into any character type variable _____ unformatted function is used.

[A] putchar()   [B] getchar()

[C] scanf   [D] printf()

2 To write a character variable on the console screen, following unformatted function is used.

[A] putchar()   [B] getchar()

[C] scanf   [D] printf()

3. To write a character value on the console screen following formatted function is used.

[A] putchar()   [B] getchar()

[C] scanf   [D] printf()

4.3   FORMATTED INPUT

Function scanf( ) is used to read data from the keyboard and to store that data in the any type of variables. Syntax for scanf( ) function is as follows:

scanf("Format String",&variable);

Here, format string will explain to the function that, which type of data it is taking as an input, this format string may be %c for character type of data, %d for integer type of data and %f for float type of data. We know that different type of data is stored in different type of variables. Variables are memory location, where the data is being stored. Function scanf() needs address of the memory location, where it stores the value entered by the user through keyboard. To specify the address of the variable, we need to specify & (address operator) before variable name. Make sure, we don't have use address operator (&) before array name while taking string from the user. The syntax for scanf () function is shown below:

scanf("format string1,format string2",&variable1,&variable2);

Accepting value for Integer Variable:
int rollno;
printf("Enter Roll No=");
scanf("%d", &rollno);

Here, in scanf() function use %d is a format string for the integer variable and &rollno will provide the address of variable rollno, to store the value at variable rollno location.

Accepting value for Float Variable:
float percentage;
printf("Enter Percentage=");
scanf("%f", &percentage);

Here, in scanf() function use %f is a format string for the float variable and &percentage will give the address of variable percentage to store the value at variable percentage location.

For Character Variable:
char ans;
printf("Enter answer=");
scanf("%c",&ans);

Here, in scanf() function use %c is a format string for a character variable and &ans will provide the address of the variable ans, to store the value at the variable ans location.

scanf() is used to accept data from keyboard. This function can be used to enter any combination of characters, numerical values and strings. The function returns the number of data items. The character specified with % sign indicates what type of data should we accept from keyboard.

Format String	Meaning
%c	To print or scan single character
%d or %i	To print or scan decimal integer
%f	To print or scan a floating-point value
%e	To print or scan a floating-point value
%x	To print or scan a Hexa integer
%o	To print or scan an Octal Integer
%s	To print or scan String
%u	To print or scan unsigned integer

Now, consider an example, suppose there are three variables char name [15], int rollno, float percentage then the scanf statement for these three variables will be:

scanf("%s %d %f", name, &rollno, &percentage);

The above statement contains group of three characters %s, %d and %f. Here, %s represents the first parameter, that is, string name, second character group %d represents that the parameter has an integer value and third character group %f represents, that the parameter has a floating-point value.

If two or more characters are entered, they must be separated by white space characters. Data items may continue onto two or more lines, since the newline character is considered to be a whitespace character.

In the case of string variable, we need to use character array, because string is a group of two or more characters. We have discussed that scanf() function needs address of the variable. So, we start all data elements with & in the case of integer, float and character variables. Variable name started with & like &rollno, provides address of variable rollno. In the case where we take a string, we use array and array name itself provide an address we don't need to mention & (address of operator).

When the program is executed, system will read continuous successive characters from the standard input device, along with that each input character matches one of the characters enclosed within the brackets. The order of the characters within the square brackets need not correspond to the order of the characters being entered.

In the case of reading of string variable, it takes one by one characters from the console screen by the user and stored in the character array in successive location. When user gives new line (Enter key) or space character, scanf() function will end storing characters in the array by storing '\0' as the last character.

Another method to achieve the same is to precede the characters within the square brackets by a circumflex (^). This causes the subsequent characters within the brackets to be interpreted in the opposite manner. Thus, when the program is executed, successive characters will continue to be read from the standard input device as long as each input character does not match one of the characters enclosed within the brackets. If the characters within the brackets are simply the circumflex followed by a new line character, then the string entered from the standard input device can contain any ASCII characters except the newline character.

For Example,

char name1[35];

scanf("%[^\n]", name1);

Through the above statement any string of undetermined length (not more than 35 characters) will be entered from the standard input device and assigned to name.

If you want to limit or restrict the width of the data item, you can define it with the help of an unsigned integer indicating the field width by placing it within the control string, that is, between the % and the conversion character. You cannot exceed the number of characters in the actual data item than the specified field width. Any character that extends beyond the specified field width will not be read.

For example,

int x, y, z;

scanf ("%d %3d %3d, &x, &y, &z);

Now, if the data input from the keyboard is 1, 2, 3 then the result will be x=1, y=2, z=3 but suppose if the data input is 987, 654, 321 then it will result in x1=987, y1=654, z1=321.

If the input is 9876 5432 1 then x=987 y=6 and z=543. The remaining two digits (2 and 1) would be ignored, unless they were read by a scanf statement.

4.4   FORMATTED OUTPUT

The formatted function printf() is used to display the value of some variable on the console screen. The general syntax for the printf() function is shown below:

printf("<format string>",<list of variables>);

To print a message on the screen:

printf("B. Sc. IT Programme");

Consider an example given below to print the value of an integer variable.

Printing Integer Variable:

int x=20;

printf("%d", x);

In the above example, %d is a format string to print some integer value and x is the integer variable, whose value will be printed by the printf() function on the console screen. This will print the value of x, that is "20" on the screen. You can make this output interactive by writing them:

int x=20;

printf("x=%d", x);

This will print "x=20" on the console screen instead of just 20.

To print values of multiple variables, you can use the printf() function in the following way:

int p=1000, r=10, n=5;
printf("Principle amount=%d rate=%d year=%d",p,r,n);

This will print "Principle amount=1000 rate=10 year=5" on the screen.

Printing Float Variable:

float perc=70.20;
printf("Percentage is =%f", perc);

In the example given above, %f is a format string to print some float(real) value and perc is the variable of type float, whose value will be printed by the printf() function. This will print the value of a Percentage=70.200000 on the screen.

Printing Character Variable:

char ans="Y";
printf("Answer=%c", ans);

In this example, %c is a format string to print a character value and ans is a variable of type character, whose value will be printed by the printf() function.

This will print value of ans, Answer=Y on the console screen.

Suppose, we want to print "BAOU" on the screen with a character variable

char ch1='B', ch2='A', ch3='O', ch4='U';
printf("Name = %c %c %c %c",ch1,ch2,ch3,ch4);

Check your progress 3

1._____ is used to assign a character value to the character variable.
[A] double quotes   [B] single quotes
[C] parenthesis   [D] no parenthesis or quotes are required
2 t _____ is used to represent string.
[A] double quotes   [B] single quotes
[C] parenthesis   [D] no parenthesis or quotes are required

This will print "Name=BAOU" on the screen. You can also use a single printf() function to print the values of different types of data as given below.

Example:

int rollno=101;

char ch='A';

float perc=78.70;

printf("Rollno is =%d Character is=%c Percentage is=%.2f",rollno,ch,perc);

This will print message "Rollno is=101 Character is=A Percentage is=78.70" on the screen.

Check your progress 4

1. Numerical variables like Integers, float and double are initialized using _____.

[A] double quotes   [B] single quotes

[C] parenthesis   [D] no parenthesis or quotes are required

2 If printf("RollNo=%i, Result=%c, Percentage=%f", rno, res, per); is a correct statement then types of rno, res and per are:

[A] double, int, float   [B] char, int, float

[C] int, char, float   [D] int, string, float

4.5   MORE ON UNFORMATTED FUNCTIONS

Some unformatted functions are given in the following table, and the behaviour of the function is enlightened in the second column of the table.

Function	Function behaviour
getchar()	User can input multiple characters (string), it stores the only first character into character variable. The characters (string) entered by the user will be displayed on screen.
getch()	User can input only one character from the console and that will be stored into the character variable. Inputted character by will not be displayed on the console screen.
getche()	User can input only one character from the console and that will be stored into the character variable. Inputted character by will be displayed on the console screen.
putchar()	This function is used to print a character on the console screen.
puts()	This function is used to print a string on the console screen. It also allow user to print a string having one or more spaces between word, which printf() function with "%s" can't do.
gets()	To accept the string from the user till user presses return (Enter) key.

Check your progress 5

1.To accept a character without displaying it _____ function is used.

[A] getchar()   [B] getch()

[C] getche()   [D] putchar()

2 which function allows user to input multiple characters (string), from which function accept only first character of the string?

[A] getchar()   [B] getch()

[C] getche()   [D] scanf()

3. To print the string with one or more spaces, _____ function is used.

[A] scanf()   [B] gets()

[C] puts()   [D] printf()

4.6 LET US SUM UP

In this unit, we:

- Discussed about performing input/output operations

- Explained the method of accepting a single character

- Interpreted the method of displaying a single character

- Elaborated on the method of accepting formatted input

- Talked about the method of displaying formatted output

4.7   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1.   [B] getchar()
2.   [A] putchar()
3.   [D] printf()

Check Your Progress-2

1.   [D] %s
2.   [A] scanf("%s",str)
3.   [B] 8f

Check Your Progress-3

1.   [B] single quotes
2.   [A] double quotes

4.8 Glossary

1.   **C tokens:** In a C source program, the basic element recognized by the compiler is the „token". A token is a source- program text that the compiler does not break down into component elements.

2.   **Constants:** Constants are the values, never change in a program.

3.   **Data types:** It identifies data type, such as floating-point, integer or Boolean, it states what type of values to be stored.

4.9   Assignment

1.   What are unformatted IO Functions? Explain it with an example of each.
2.   List and explain all formatted IO functions with an example of each.

4.10   Activity

Write the following program, in the C-Language and observe the output. Change line 5 and replace function getch() and getche() instead of getchar() function. Note your observation.

```
1.      #include<stdio.h>
2.      void main ()
3.      {
4.              char ch;
5.              ch=getchar();
6.              printf("\n Character Entered is:", ch);
7.      }
```

4.11   Case Study

•   Write a program to accept the string having spaces between words and print it on the console.

•   Write a program to accept string having spaces and new-line (enter) till user is not pressing Ctrl+z and Enter. Print the string in Console.

4.12 Further Readings

- "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
- "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
- BAOU Self-Learning Material, BCA Program. URL: https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

Block Summary

- The programming language C was originally developed by Dennis Ritchie of Bell Laboratories and was designed to run on a PDP-11 with a UNIX operating system.

- C a preferred language among programmers for business and industrial applications because of its features, simple syntax and portability.

- This language is also called middle-level language, as it is closer to both machine and user.

- A function is a collection of one or more statements which performs a specific task.

- A program is divided 4 parts

- In comments section we can write details like program name, programmers name and functionality of the program.

- In Library section the compiler link functions from the system library.

- In Definition it defines all the symbolic constants.

- In Global declaration contains the declaration of variables which are used by more than one function of the program.

- A program with single function must be the main program. An execution of any program starts with main () function.

- To give programming instructions to your computer, we require an editor and a C compiler to compile the program instructions.

- The pre-processor directives control the way your programs should be compiled. The compiler is a translator, which translates your source code file to an executable file.

- to compile the program, you have to use compile menu or short cut key which is Alter + F9 and to run the program you have to use run menu or short cut key which is control + F9.

- Constants are those quantities whose value may not change during execution of C program.

- C Supports Integer, Character, String and Floating-point constants. Integer constants represent numbers, floating constants represent decimal numbers, combined together is denoted as numeric type constants.

- A variable is a storage location in your computer's memory which stores data. Using a variable's name in the program, we refer to the data stored.

- Char data type is used in C to access and to store single character.

- To access and to store any integer value, int data type is used. C's int data type occupies 2 bytes in the memory.

- All those set of characters which are used to write a C program are called C character set.

- C uses the uppercase letters A to Z, the lowercase letters a to z, digits 0 to 9 and some special characters to form basic program elements.

- Each word used in a program is called token.

- Words with a specific meaning are known as keywords. Keyword must not be used as variable names. There are 32 keywords in the programming language

- Identifiers are names given to various elements of a program such as variables, functions and arrays.

- Declaration of a variable involves specification of data type with it. In C language all variables must be declared before they appear in executable statements.

- An operator is used to perform operations on operands.

- five basic arithmetic operators used in „C‟ For Addition '+', For Subtraction ' -', For Division '/', For Multiplication '*',For Mod '%'.

- Relational operators are divided into 4 types such as Less than ($<$), Less than or equal to ($<=$), Greater than ($>$), Greater than or equal to ($>=$), Equals to ($==$), Not equal to ($!=$).

- Logical operators used are AND ($\&\&$), OR($\|$) and NOT($!$).

- Assignment operators are used to assign the value to an identifier.

- Unary Operator operates on only one operand called as increment ($++$) and decrement ($--$) operators.

- Conditional operators are termed as Ternary Operators ($?:$).

- Bitwise operations which can be divided into three general categories: the one's complement operator, the logical bitwise operators and the shift operators.

- sizeof() is used to determine the length of array and structures and to allocate memory space dynamically during execution of a program.

- Using comma operator, we can use two different expressions simultaneously where only one expression would ordinarily be used.

- Precedence rules is used to evaluate an expression without parenthesis it is evaluated from left to right.

- Highest Priority is given to * , / , % and Lowest Priority is set to + and –

- The mathematical calculations can be done by including the header file <math.h>.

- Two functions are used like getchar() is used to read a character from standard input device and scanf() used to read data from a key board.

- For formatting we can use printf() function which displays the output on the console.

- C library function putchar(). It transmits a single character to a standard output device.

- format string is used to define which type of data it is taking as input, this format string can be %c for character, %d for integer variable and %f for float variable.

BLOCK 2: DECISION MAKING AND LOOPING

Block Introduction

In the previous block, you studied about the basics of C language. Here, we will be discussing about the different constructs of this language which will be useful for program development.

The C language programs presented until now followed a sequential form of execution of statements. Many times, it is required to alter the flow of the sequence of instructions. C language provides statements that can alter the flow of a sequence of instructions. These statements are called control statements or decision making. These statements help to jump from one part of the program to another.

Block Objective

"Decision making" is one of the most important concepts of computer programming. Many Programs require testing of some conditions at some point in the program and selecting one of the alternative paths depending upon the result of condition. This is known as Branching.

The control transfer may be unconditional or conditional. Branching Statements are of the following categories:

- ☐ If Statement
- ☐ If else Statement
- ☐ Nested if Statement
- ☐ Switch Statement

Loops are group of instructions executed repeatedly while certain condition remains true. There are two types of loops, counter controlled and sentinel-controlled loops (repetition).

Counter controlled repetitions are the loops in which the number of statements repeated for the loop is known in advance. These loops require control variables to count number of repetitions. So, in Counter controlled repetitions control variable (loop counter) is initialized, an increment (or decrement) statement which changes the value of loop counter and a condition used to terminate the loop (continuation condition). Sentinel loops are executed until some condition is satisfied. Condition can be checked at top or bottom of the loop.

Thus, in this block, we will study about these statements which will make us acquainted with the different statements and will be helpful in writing programs.

Block Structure

UNIT 1 DECISION MAKING AND BRANCHING

Unit Structure

1.0   LEARNING OBJECTIVES

In this unit, we will discuss about the programming constructs for decision making.

After working through this unit, you should be able to:

- Explain decision-making with the if Statement

- Experiment with the switch Statement

- Elaborate on the goto Statement

1.1   INTRODUCTION

In the previous discussion, we have studied about the basics of C language, where we have discussed how to write simple C-programs and how to write C-statements withing the program. We also learn the use of data types, constants and variable. Now in this unit, we will discuss about the decision-making statements, which are very useful while writing C-programs and helps the programmer to use it whenever certain decisions needs to be made.

1.2   DECISION MAKING WITH THE IF STATEMENT

Whenever we need to make decisions, we can use if statement. In C-programming language, 'If condition' comes with several variations. Depending upon problem we are solving, we use specific variant of the 'If statement'. In this section we are discussing all the variants of 'If statement'.

1.2.1 The if-statement:

Generally, if statement is used to evaluate a logical statement or we can say to check the specific condition and then select one of the alternatives from two possible alternatives depending on the outcome of the logical test (i.e., whether the condition is true or false). Thus, in its simplest form, the syntax of If-condition is as follows:

if (condition)

   Statement;

To specify the condition, parenthesis must be used as specify in the syntax. The condition can be specified by using relational operators. Statement mentioned in the if condition will be executed if and only if, the relational operators used in the condition produces non-zero value, which means the condition specified is TRUE. In the case where condition is FALSE, the statement mentioned in the If condition is simply skipped or ignored.

Conditional statements may be either simple or compound. In most cases if statements are compound and it is used with other control statements.

1.2.2 The if…else statement:

The general form of an If …else statement, which includes the else clause is:

if (condition)
*   Statement 1;*
else
*   Statement 2;*

In If … else, statement condition is evaluated, which is specified in the parenthesis. If the condition is TRUE (non-zero) then statement1 will be executed by the system, but if the condition results in FALSE (zero) then else part of the if…else statement, that is statement2 will be executed. The flow diagram of if … else is shown below:
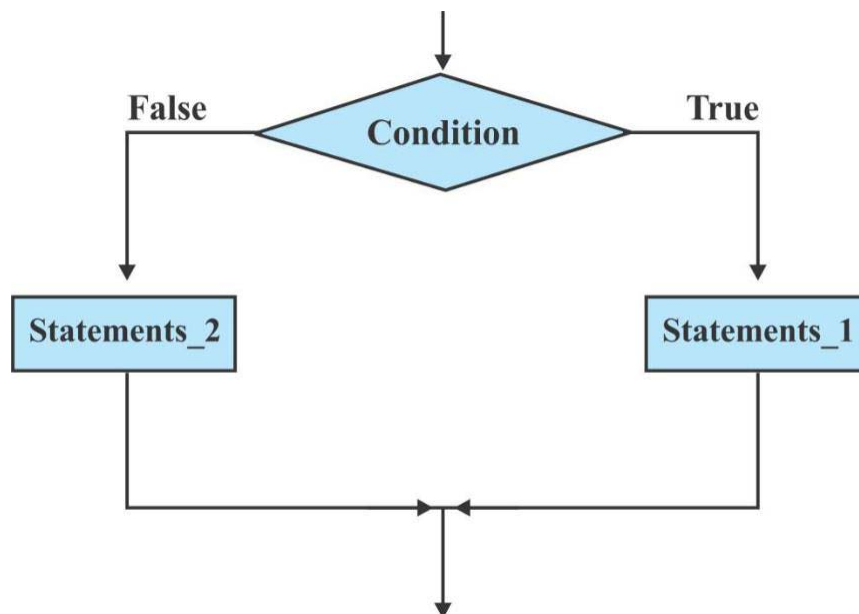
For example, to find a greater number from given two numbers we may write the following C-programming syntax:

#include<stdio.h>

void main()
{
*   int a=10, b=7;*
*   if(a>b)*
*   printf("Greater Number is: %d",a);*
*   else*
*   printf("Greater Number is: %d",b);*
}

1.2.3 Nested If…Else Statements:

As seen earlier, the if clause and else part may contain a compound statement. Moreover, either or both may contain another if or if…. else statement.
This is called nesting of if…else statements. This provides programmer with a lot of flexibility in programming. Nesting could take one of several forms as shown below

Format 1: *if(<conditon1>)* *{* 　　　*statement1;* *}* *else* *{* 　　　*if(<conditon2>)* 　　　*{* 　　　*statement2;* 　　　*}* *}* **Format 2:** *if(<conditon1>)* *{* *if(<conditon2>)* *{* 　　　*Statement1;* *}* *else* *{* 　　　*Statement2;* *}* *}* *else* *{* 　　　*Statement3;* *}* **Format 3:** *if(<Condition1>)* *{* *if(<Conditon2>)* *{* 　　　*Statement1;* *}* *}* *else* *{* 　　　*Statement2;* *}*	 ***Flow diagram of Nested If condition***
Syntax: if Nested If Condition	

Format 4:

if(<Conditon1>)
{
if(<Conditon2>)
{
        Statement1;
}
else
{
      Statement2;
}
}
else
{
if(<Conditon3>)
{
        Statement3;
}
else
{
      Statement4;
}
}

For Example:
#include<stdio.h>
void main()
{
   int num1=10, num2=20, num3=30;
   if(num1>num2)
   {
     if(num1>num3)
        printf("Greatest Number is:%d",num1);
     else
        printf("Greatest Number is:%d",num3);
   }
   else
   {
     if(num2>num3)
        printf("Greatest Number is:%d",num2);
     else
        printf("Greatest Number is:%d",num3);
   }
}

1.2.4 The if…else if...else ladder

When many conditions to be evaluated, we need to use if …else if …else ladder:

if(Condition1)

    Statement1;

else if(Condition2)

    Statement2;

else if(Condition3)

    Statement3;

else if(Condition)

    Statement

n;

else

    Default
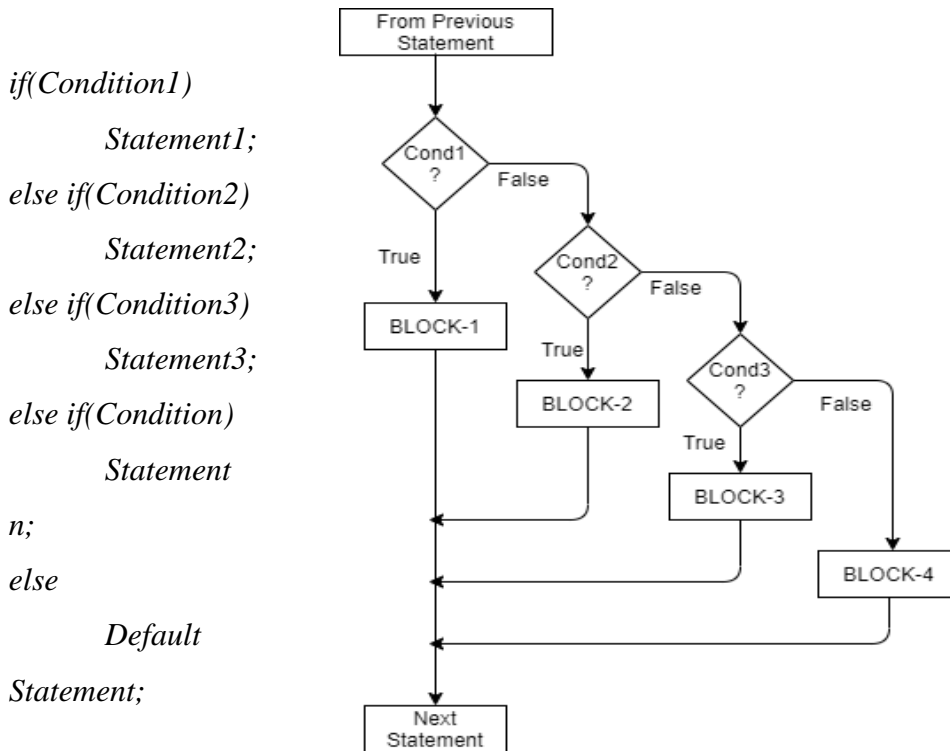
Statement;

Syntax: **If…Else** **Flow Diagram of If…Else If…Else If…Else**

The statement discussed above is known as the if…else if …else ladder. The conditions will be evaluated from top to bottom. When the condition is evaluated to be true, the statement associated with that particular if condition gets executed and the control of the program gets transferred to statement-x skipping the other statements written after it. If no conditions specified with 'if' or 'else if' is true, then the default statement will be executed.

Consider the following example,

Suppose you are asked to generate grade from the obtained marks of the students using the table given below:

Marks	Grade
>80	Distinction
>60 and <80	First Class
>50 and <60	Second Class
>40 and <50	Pass Class
<40	Fail

Then the program for the same using if – else if – else ladder can be written as:

void main( )

{

    int marks=0;

    printf("Enter marks of the Student:\n");

    scanf("%d",& marks);

    if(marks >=80)

    printf("Distinction\n");

    else if(marks >=60)

    printf("First Class\n");

    else if(marks >=50)

    printf("Second Class\n");

    else if(marks >=35)

    printf("Third Class\n");

    else

    printf("Fail");

}

Check Your Progress-1

1.Else keyword can be used with _____.

[A] if statement   [B] switch statement

[C] do … while statement   [D] None of the above

2. if condition is FALSE, _____ block (group of statements) will be executed.

[A] If   [B] Else

[C] Break   [D] None of the Above

3.If we put an if condition, within another if condition is called _____.

[A] simple if   [B] if … else

[C] if … else if … else ladders [D] nested if

1.3 THE SWITCH…CASE STATEMENT

The switch statement is used to execute a particular group of statements to from several available alternatives. Depending upon the value of an expression that is passed within a switch statement, group of statements are executed. The switch…case statement may be similar to an If…Else If…Else statement. The Switch…Case statement is used to test only equality (It cannot be used to test less than, greater than etc). It is more convenient, when we want to match expression with different test cases, and based on equality we select one group of statements.

The general form of switch-case statement is:

switch(expression)

{

   case constant1:

       Statement1;

       break;

   case constant2:

       Statement2;

       break;

   case constant3:

       Statement3;

       break;

}

The expression passed in a switch statement must be integer constant or a character constant. A switch…case statement can have multiple cases. Each case is labelled with some constant value. The expression passed with switch statement will be evaluated and the resultant value is compared with the different case label. When, the value of expression evaluation is matched with particular case the statements written in that particular case will executed.

Example:

switch (option=getchar())

{

case 'a':

case 'A':

  printf ("APPLE");

  break;

case 'b':

case 'B':

  printf ("BLUEBERRY");

  break;

case 'c':

case 'C':

  printf ("CHERRY");

  break;

default:

  printf("Invalid Option Entered:");

}

Thus, APPLE will be displayed if value option represents either a or A. BLUEBERRY will be displayed if option represents either b or B and CHERRY will be displayed if option represents either c or C. Here, each group of statements has two case labels to match with, either upper case or lower case. Note that each of the first three group ends with the break statement. The break statement is used to transfer execution control out of the switch statement after execution of all statements of that particular case, that prevents more than one group of statement from being execution.

The last group is labelled as a default. The group of statements written in the default group will be executed when the value of an expression passed with switch statement do not math with any of the case constant value. Default group can appear anywhere in the switch…case statement. It is not compulsory to place it at the end, if none of the case labels matches the value of the expression and the default group is present and then the statement will take no action.

Here is a variation of switch statement

ch=toupper(getchar());
switch(ch)
{
case 'A':
printf ("APPLE");
break;
case 'B':
printf ("BLUEBERRY");
break;
case 'C':
printf ("CHERRY");
break;
* default:*
*   printf("Invalid Option Entered:");*
}

In the above example, we have taken the choice from the user and stored it in the variable 'ch'. Before storing the value, we have change the user choice in to upper case, by calling a function called toupper(). Here, in the variable 'ch' only upper-case value exists, therefore there is no reason to match the value of variable 'ch' to lower case values. Application of this logic has been reducing number of cases in our switch … case statement.

Function toupper() is a library function and use this function you need to include the header file called 'ctype.h'. Each group has 'break' statement which transfer the control to end of the switch…case statement when any one of the case is executed and the cases written below that particular case will not be executed.

Check Your Progress-2

1.To skip other cases, when one case is executed in the switch…case statement _____ keyword is used.

[A] exit()   [B] continue

[C] break   [D] goto

2. Character constant passed with switch, in switch…case statement automatically converted into _____.

[A] floats   [B] integers

[C] Boolean   [D] None of the Above

3.If we forget to mention break in switch…case statement then _____.

[A] All cases will be executed

[B] No case will be executed

[C] All cases, after first matched case will be executed

[D] Error message will be displayed

1.4   THE CONDITIONAL OPERATOR (?:)

Conditional operators (?:) is ternary operators, that takes three expressions. First expression must be condition(s), which will be evaluated. Base on the condition evaluation, weather it is true or false, out of two anyone expression or statement is executed. Condition and true parts are separated by operator '?', and true and false part is separated by ':'.

Syntax:

(<Condition>)? True: false

Expression1 ? Expression2: Expression3

Here, if condition in exression1 evaluated as true then value of expression2 will return and if condition in expression1 is evaluated as false then expression3 will return.

For example,

Result=(Var1<0) ? 0 : 100

If 'Var1' variable's value is less than 0 then condition is evaluated as true and Result variable will be assigned value 0 but if 'Var1' variable's value is greater than 0 then condition is evaluated as false and value 100 will be assigned to variable Ans. For example,

printf("%d", (var1<0) ? 0:100);

Check Your Progress-3

1. Identify the ternary operator from the given below:

[A] %   [B] &&

[C] ?:   [D] sizeof

2. Statement (a>b)?a:b will return _____.

[A] smaller number from a and b   [B] greater number from a and b

[C] sum of both numbers   [D] Error

1.5   THE GOTO STATEMENT

The 'goto' statement change the natural flow of program execution by transferring control to some other part of the program, unconditionally. The syntax of goto statement is:

goto label;

Here, label is an identifier used to label, the target statement to which control will be transferred.

In the definition of the label, we need to mention label name must be followed by a colon. That means, the target statement will appear as label: statement. The name of the label must be unique in a program. That means, in a program two labels should not have same name.

The common uses of goto are:

- To move from one statement to another statement without evaluating any condition.

- To transfer the control out the loop.

- To repeat the statements (similar to loop), without using any (for, while or do…while) loop.

Branching around statement can be fulfilled with the if-else statement.

For example,

i=1;

loop:

printf("%d", i);
i++;

if (i < 100)

goto loop;

The following program will print from 1 to 100. In this program we have declared and initialized variable 'i' to 1. We have labelled 'loop'. After printing and incrementing the value of I variable, we are evaluating a condition that the value of variable 'i' is less than 100. In this case, we are transferring the control to the label loop and all the statements written below label loop is repeated.

Statement goto is used to exit from the nested loop. We might use the break statement to come out from the loop, but in the case of nesting of the loop we need to write break statement in both inner and outer loop. In such situation goto can solve the purpose by writing only one statement. Statement 'goto' is unnational jump statement, and that why using too much goto statement is not advisable or using too much goto statement is not good programming practice.

Check Your Progress-4

1. Which statement is called unconditional jump?

[A] if condition   [B] for loop

[C] while loop   [D] goto

2. Good programmers avoid _____ statements into their programs, because it degrades performance of the program.

[A] goto   [B] switch…case

[C] if…else   [D] for loop

1.6   USING LOGICAL OPERATORS IN IF

In certain circumstances we need to execute group of statements after evaluating two or more conditions. In such situation, two or more conditions are separated by using logical operators such as OR (| |) and AND (&&). For example, to find a maximum value from given three numbers can be implemented using nested if conditions, as discussed in 1.2.3. Similar thing can be done using logical AND (&&) operators. In the example given below, we have found the maximum value from given three numbers using logical AND (&&) operator. Compare both programs and think which logic is easier and more readable.

```c
#include<stdio.h>
void main()
{
    int a=5, b=7, c=6;
    if(a>b && a>c)
          printf("Greatest Number is:%d", a);
    if(b>a && b>c)
          printf("Greatest Number is:%d", b);
    if(c>a && c>b)
          printf("Greatest Number is:%d", c);
}
```

In the example, we are evaluating that if a>b and a>c then, a is the greatest number, and in the same way we are also checking for b and c. Logical operator && is used in this case, because a will be a greatest number, if and only if a>b AND a>c (It is compulsory that both conditions are TRUE). Consider the next program where we are accepting an alphabet from the user and we are checking, it is Vowel or Consonant.

```c
#include<stdio.h>
void main()
{
    char c;
    printf("\n Enter Alphabet:");
    scanf("%c", &c);
    if(c=='a' || c=='e' || c=='i' || c=='o'||c=='u')
       printf("Entered Character is Vowel");
    else
       printf("Entered Character is Consonant");
}
```

In the program, to print "Entered Character is Vowel", we need to check if the value of the variable 'c' is wither 'a' or 'e' or 'i' or 'o' or 'u'. From the number of cases, if any one is True then character is Vowel.

Similarly, another logical operator NOT (!) is used to negate the condition. For example, if we want to check whether the variable i is not 5 then we can write the following if condition.

```c
if( !(i==5))
     printf ("Value of i is not 5);
else
     printf ("Value of i is 5);
```

Check Your Progress-5

1. In the case where, we need to check, if character entered by the user is Vowel or not, _____ logical operator is used in the if-statement.

[A] &&   [B] ||

[C] !   [D] None of the above

2. Which is not logical operator, from given operators.

[A] !   [B] ||

[C] %   [D] &&

1.7   LET US SUM UP

In this unit, we:

- Discussed about if statements used for decision making

- Elaborated on if …elseif…else statements which provide an alternative of executing a statement when the given condition is not true

- Talked about the switch Statement

- Explained the goto statement used to jump the control of a program from one part to another

1.8   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1.   [A] if statement
2.   [B] Else
3.   [D] Nested if

Check Your Progress-2

1.   [C] break
2.   [B] integers
3.   [C] All cases, after first matched case will be executed

Check Your Progress-3

1.   [C] ?:
2.   [B] greater number from a and b

1.9   GLOSSARY

1.   **goto** is a statement in C-Language which performs jump to some statement without evaluating condition. It is also called unconditional jump.
2.   **switch … case** is a statement in C-Language, which match the expression passed into switch, with all different cases and matched case will be executed. Unlike if condition, it used to check for only equality.

1.10   Assignment

1.   List and explain different types of if conditions with example of each.
2.   Explain switch … case statement with an example.
3.   Explain the use of conditional operators.

1.11   Activity

1.   Write a program to check given character is Vowel or Consonant using switch … case statement.
2.   Write a program to find greatest number from given 3 numbers using conditional operators.

1.12   Case Study

Write a syntax, draw a flow-diagram and example of all different types of if conditions.

1.13   Further Reading

•   "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
•   "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
•   BAOU Self-Learning Material, BCA Program. URL:
https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

UNIT 2   LOOPING

Unit Structure

2.0 LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Interpret Looping

- Comprehend the additional features of for loop

- List break statements

- Write about continue statements

2.1 INTRODUCTION

Sometime, in the programming we need to execute group of statements repeatedly. For example, to check number 7 is prime or not, we need to divide 7 by 2, 3, 4, 5, and 6. If we can not divide 7 by any of these numbers (2 to 7) and not getting remainder 0, then and then we can say 7 is a prime number. Here, we need to repeat some tasks that dividing 7 by another number from 2 to 6. In such situations we need to use loops. Loops are useful, helps us in solving many complex problems and almost available in all high-level programming languages. In this chapter we will discuss different types loops, available in the C-Programming language.

2.2 LOOPING

If certain task we want to perform number of times or we want to execute same statement or a group of statements again and again then, we can use loops. There are three types of loops are there in C-Programming language.

- While loop in the C-language, checks the condition first. If condition is True then it will execute the body part (group of statements associated with while loop). After execution, again control is transferred to while statement to evaluate condition again. This process is repeated till the condition does not becomes False.

- For loop is another loop, which is similar to while loop. It takes initialization, condition and increment/decrement instruction in one line. This makes C-Program to be smaller and more readable.

- In the case of do…while, loop the instructions associated with loop will be executed first and then the condition is evaluated.

2.2.1 The while loop:

While, is a loop structure used to repeat some instructions to be executed several times. The syntax of the while loop is as shown below:

Initialization_statement;

while(Condition)

{

    statement 1;

    statement 2;

    increment/ decrement;

}

The statements written within two braces are called the body part of the loop. The braces are not needed if there is only one statement is there in the body part of the loop. For two or more statements, braces are compulsory. But it is advisable to write statements associated with while loop has to be placed in the opening and closing brace. Body part of the loop will be repeated till condition is True. When the condition is evaluated as False, then program control will come out from the loop and it will stop the repetition. Thus, it is entry-controlled loop in which condition evaluated first and then body part of the loop is executed. Initialization statement will assign some initial value before loop starts. Increment or decrement statements increases or decreases the value of loop variable used in the condition. We must include some features, so that after certain number of iterations the condition will becomes False and loop will stop its repetition.

Logical operators can be used in the loop, if there are two or more conditions are there. In the case of logical && operators, loop will continue to repeat the body part, till any one condition does not becomes False, and in the case of, logical || operators, loop will continue till all condition does not becomes False.

Suppose, if you want to display first 10 natural numbers on each line. You can write the following C-program.

```
#include<stdio.h>
void main ()
{
      int n = 1;
      while(n<=10)
      {
      printf("%d\n", n);
      n++;
      }
}
```

Variable n is initialized with 1. While statement will evaluate the condition that 'n<=10'. Because the value of n variable is 1 and 1<10, body part of the loop will be executed, which will print 1 on the console screen and the value of variable n is increased by 1 and it becomes 2. After executing both statements when '}' statements come, control is transferred to while statement again. Because variable n which 2 is < 10 (condition is True) body part of the loop will be repeated. The same process is repeated till value of n becomes 11 and condition n<=10 becomes False. We will get the following output of this program:

12 3 4 5 6 7 8 9 10

Check Your Progress-1

1. A while loop, which does not have any increment / decrement statement is called _____.

[A] For loop   [B] Null loop

[C] do … while loop   [D] Infinite loop

2. In while loop, Initialisation statement is written _____.

[A] before while loop   [B] in the parenthesis of while

[C] in the body part of the loop [D] after body part of the loop

3. In while loop, condition is written _____.

[A] before while loop   [B] in the parenthesis of while

[C] in the body part of the loop [D] after body part of the loop

---

2.2.2 The Do…while loop:

Do…while loop mostly used in the situation where programmer wants, the loop to be executed at least once. In the case where we want to show some menu options to the user (Menu based program), we need to execute the loop at least once so that we can show the menu options to the user.

The syntax of do-while loop statement is:

do
{
        statement 1;
        statement 2;
        increment/decrement operation;
} while (Condition);

The enclosed statement within { and } braces will be repeated till the condition is True. Note, this loop will be executed at least once, since it is not evaluating the condition when the loop starts its execution. Condition is placed at the bottom part of the loop and it is evaluated after the completion of the first iteration of the loop. In this loop, the condition is evaluated at the end or exit. Therefore, it is also known as Exit-control loop.

Similar to while loop do…while loop also repeat the group of statements associated with the loop till the condition is True. When the condition becomes False, the control comes out of the loop and it stops repeating the statements.

In most cases, we need to evaluate condition prior the execution of the body part. Therefore, there are a smaller number of applications are there, where we need to use do … while loop. But as discussed earlier, in menu-based program do…while loop is used to execute, body part at least once to show the menu options to the user. To print from 1 to 10, do…while loop can be used in the following manner.

```
#include <stdio.h>
void main()
{
    int n = 1;
    do
    {
    printf("%d\n", n);
    n++;
    } while(n<=10);
}
```

In this program variable 'n' is initialized with value 1. The do…while loop displays the current value of variable 'n', increases the value by 1 and then check the condition to see if the current value of variable 'n' exceeds by 10. If it is, loop terminates; otherwise, the loop continues its repetition, using the new value of 'n'. As a result, the loop is repeated for 10 times, resulting in 10 successive numbers from 1 to 10 are printed on the console screen:

1 2   3   4   5   6   7   8   9   10

Check Your Progress-2

1. which loop is known as exit-controlled loop.

[A] For loop   [B] While loop

[C] do … while loop   [D] None of the above

2. Condition is evaluated after execution of body part of the loop in _____.

[A] do … while loop   [B] while loop

[C] for loop   [D] All of the above

3. while loop will be executed at least once.

[A] While loop   [B] Do … while loop

[C] For loop   [D] None of the above

2.2.3 The For loop:

The for loop is similar to while loop. It is an entry control loop and the syntax of the for loop is:

for (initialization; condition; increment)

{

  statement 1;
  statement 2;

}

In this loop 'for' is a keyword indicates that for loop is used. After 'for' keyword we need to specify initialization, conditional statement and increment/ decrement statements in the parenthesis separated by semi-colon (;). Similar to the while loop, all the executable statements needs to specify between { and } braces.

System, will execute the statement or expression written in the initialization part once. It will check the condition. If the condition is True then body part of the loop will be executed. After execution, increment/decrement statement is executed and once again the condition is evaluated. If the condition is True again then loop will execute the body part again in its 2nd iteration, but it the condition is False then control is transferred to the next line of the program outside of the loop.

(a) Consider the following segment of a program:

for (cntr=1; cntr<=10; cnrt++)

{

  printf ("%d\n", cntr);

}

In the above example, for loop is repeated 10 times, which prints the numbers from 1 to 10, each on a new line. The three sections cntr=1, cntr<=10 and cntr++ within parentheses must be separated by a semicolon (;). Note that you do not have write semicolon at the end of the for statement.

In the next example, we are printing the values in descending order from 10 to 1. In this example variable cntr should be initialize with value 10 and we need to repeat the loop for condition cntr >=1. After each iteration, the value of the cntr should be decremented by 1.

```
for(cnt=10; cnt>=1; cnt--)
{
        printf("%d \t",cnt);

}
```

        Output:

        10   9   8   7   6   5   4   3   2   1

Check Your Progress-3

1. Entry-control loop(s) is/are _____.

[A] For loop   [B] While loop

[C] do … while loop   [D] Both A and B

2. Initialization, condition and increment/decrement statements are separated by _____ operator in for loop.

[A] ; (Semi-Colon)   [B] : (Colon)

[C] , (Comma)   [D] . (Dot)

2.2.4 Additional features of for loop

        More than one initialization statement can be mentioned in the For loop. Make sure in the case where multiple variables needs to be initialized in the for loop all initialization statements must be separated by comma (,) operator and after all initialization statement, you need to specify semicolon to make separation between initialization and conditional statements. For example,

for(fact=1,i=1; i<=5; i++,fact=i);*
printf("Factorial of 5 is: %d", fact);

        In the above example for loop does not have any body part (ended with semicolon). Two initialization statements fact=1 and i=1 is written in the initialization part. Condition i<=5 is evaluated after each loop iteration. Value or variable 'i' is incremented and fact=fact*i is computed after each iteration of the for loop. This program will print: Factorial of 5 is: 120.

        For loop also allow multiple conditions in the conditional part of the loop. Make sure in the programming languages, when two or more conditions are written then either AND (&&) or OR (||) logical operator have to be used:

sum=0;

for(num=1; num<=10 && sum<40; num++)
{
        sum+=num;

}
        printf("Sum is: %d", sum);

In the above program we are running a loop from 1 to 10. One obvious condition we to specify that loop variable num<=10. In the for loop, we have added one more condition that if the value of the sum is smaller than 40 then loop should be repeated. Now after 9 iteration value of sum become 45 which not <40, so loop is terminated and Sum 45 will be printed. For loop will not take iteration for num=10.

For loop also allow us to write the expression in the initialization part. The expression will be evaluated and then loop variable will be initialized. For example, a statement of the type:

for(mid=(beg+end)/2; beg < end;mid=(beg+end)/2)

is valid perfectly.

One more important aspect about the for loop is, you can skip any part from initialization, condition and increment/decrement, whenever it is necessary. For example, consider the following statements:

```
num=0;
for(;num!=100;)
{
        printf("%d \n", num);
        num+=5;
}
```

In the above example we have initialize 'num' variable with value 0, outside of the loop. In the 'for' statement, we have skipped and kept initialization part to be empty. Similarly, we have mention increment statement x+=5 in the body of the loop, so we have also skipped that part from the 'for' statemen.

We can also use the for-loop to introduce delay in our program, For Example,

for(i=1000; i>0;i--) { }

If the above loop is executed 1000 times without any output to be produced, it will simply introduce time delay in the execution of the program. We can also write the same statement as given below:

for(i=1000; i>0; i--);

This type of statement is perfectly valid and loop will run for 1000 time, with execution of null statement.

We can write a loop, within one more loop. This type of structure, where we are placing a loop structure inside one more loop is called nesting of the loop or simply nested loop.

Whenever you want to run many iterations of variable 'c' for single iteration of variable 'r', and this type of many iterations you want to run from variable 'r' then you can use nested loop:

```
void main( )
{
int r,c;
for(r=1; r<=5; r++)
{
for(c=1; c<=5; c++)
{
printf(" * ");
}/*End of Loop c*/
printf("\n");

}/*End of loop r*/
}
```

In the above program, the loop for variable 'r' will run for 5 time. For each iteration of 'r', the loop of variable 'c' will take 5 iterations. That means, the loop of 'c' variable will be executed for 25 times.

Check Your Progress-4

1. The correct syntax of the for loop is _____.

[A] for (condition, initialization, increment) { }

[B] for (condition; initialization; increment) { }

[C] for (increment; condition; initialization) { }

[D] for (initialization; condition; increment) { }

2. In a for loop two initialization statement is separated by _____.

[A] ; (Semi-Colon) [B] : (Colon)

[C] , (Comma) [D] . (Dot)

3. If we place a for loop within one more for loop is called _____

[A] Nested for loop [B] Infinite loop

[C] Exit-controlled loop [D] All of the above

2.3 Break and Continue Keywords

To control the execution of loop, that means to terminate premature loop or to skip certain statements within the loop, break and continue keywords are used. In this section we will see, how can we use these keywords into our programs.

2.3.1 Break statement

There 'break' keyword is use to terminate premature loop. When certain condition to be evaluate True and break statement is executed then control is transferred to the next statement after loop and loop will not be repeated from that point. For example,

for(n=1; n<=10; n++)

{

*   if(n==5)*
*   break;*
*   printf("\nN is=%d", n);*

}

The output of the program will be 1,2,3,4 and then break statement will terminate this loop and stop the execution of the for loop.

2.3.2 Continue Statement

This is similar to break except it does not terminate the loop, but rather than sending a control to the end of the loop, continue statement send the program control to the header of the loop.

Like a break statement, continue should also be protected by any if statement.

Statement continue is used to bypass or to skip some step or iteration of loop structure. It is simply written as continue.

for(n=1; n<=10; n++)

{

*   if(n >=5 && n <= 8)*

*   continue;*

*   printf("\nN=%d",n);*

}

The above program will print 1,2,3,4,9,10. For the value 5,6,7, and 8 the 'continue' statement will be executed, which will send the program control to the header of the loop (for statement).

Check Your Progress-5

1. _____ keyword is used to terminate, premature loop.

[A] continue   [B] break

[C] end   [D] stop

2. Predict the output of the following program:

for(n=1; n<=10; n++)

{

if(n>4 && n<8)

*   continue;*

printf("%d,",n);

}

[A] 4,5,6,7,8,   [B] 1,2,3,

[C] 1,2,3,4,8,9,10,   [D] 1,2,3,4,5,6,7,8,9,10,

2.4   LET US SUM UP

In this unit, we:

• Have seen the use of the for statement which is used to execute the same set of statements again and again.

• Have elaborated on break and continue statements which are used to send the control program to the beginning and end of the loop respectively.

2.5   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1.   [D] Infinite loop
2.   [A] Before while loop
3.   [B] In the parenthesis of while

Check Your Progress-2

1.   [C] Do … while loop
2.   [A] Do … while loop
3.   [B] Do … while loop

Check Your Progress-3

1.   [D] Both [A] and [B]
2.   [A] ; (Semi-Colon)

2.6 GLOSSARY

1.   **break** is a keyword in C-Language, which is used to terminate premature loop. It is also used in a switch … case statement to terminate case block.

2.   **Continue** is a statement in C-Language, when executed the control will be transferred to the header part of the loop. The statements written below continue statements within the loop body will not be executed.

3.   **Loop** is used in the programming languages to repeat some task (group of executable statements) again and again till specific condition becomes false.

2.7 Assignment

1.   Write a program to print all prime numbers from 1 to 50.
2.   Write a program to check whether given number is perfect number or not.
3.   Write a program to reverse the given number.
4.   Write a syntax of while, for and do … while loop, also draw the flow diagram of it.
5.   Write a program to generate multiplication table of given number.
6.   Differentiate while loop and do …while loop.

2.8 Activity

1.   Write a program to print the following pattern in the C-Language.

*

* *

* * *

2.   Write a program to print the following pattern in the C-Language.

            *

        *   *

      *   *   *

2.9 Case Study

Write a program to check whether the given number is Magic number or not. To know what is magic number then consider the following example.

Let if user has entered a number 10, then first do sum of all first 10 numbers.
1+2+3+4+5+6+7+8+9+10 = 55
Now do the sum of all digits; that is 5 + 5 =10

Here we are getting 10, which is a number entered by the user so we can say 10 is a magic number. Another magic number is 9. Because (1+2+ … +9=45) and 4+5 is again 9.

2.10 Further Reading

- "Programming in C", From PEARSON Publications, By Ashok N. Kamthane.
- "Programming in ANSI C", From McGraw-Hill Education by E. Balagurusamy.
- BAOU Self-Learning Material, BCA Program. URL: https://baou.edu.in/assets/pdf/BCAR-103_slm.pdf

UNIT 3   SOLVED PROGRAMS - I

After learning "Decision making and Branching" and "Looping" chapter, now it's time to do some programming. So, from now next 2 units we will see few sample C-programs. In the Unit:3 we will discuss programs related to "Decision making and Branching" and in the Unit:4 we will discuss different program related to "Looping". We will start with basic program and expand our knowledge toward complex programs.

```
/* Program:1[A]-Hello World on the Console Screen */
#include<stdio.h>
void main()
{
   /*This is my first C-program */
   printf("Hello World:");
}
```

OUTPUT:

Hello World:

In this program, we need to included header file called 'stdio.h'. Header file 'stdio.h' provides basic functions like 'printf()' and 'scanf()'. So, it is compulsory to include this header file, if you are using 'printf()' or 'scanf()' kind of functions. Header file 'stdio.h' stands for 'Standard Input Output Header file'. In the next line we are creating main() function by writing 'void main()'.As we know every C-Program must have function main(). Function main do not return any value, doe to this reason, the return type of the main() function is 'void'. Main function starts with '{' and ends with '}'. Between start '{'and, end '}' we have written /* This is my first C-program */. We have already discussed that the statement written in /* and */ is a comment and it will not be executed by the system. Comments are non-executable statements, we adding to increase the readability of the program. In the last line of the program, we have called a function 'printf()' and we are passing data "\nHello" to it. Here 'printf()' function execute the code which is written in the 'stdio.h' file and print the string "Hello World:" on the console.

If you are 'Code Block' or 'Borland C' user then you can type this program directly and execute the program after compilation. If you are using 'Turbo C' then in every program you need to include header file 'conio.h', which is 'Console Input Output Header file'.

You need to use function 'clrscr()' to clear the console screen after variable declaration, and finally you need to call a function getch() at the end of every program. This three steps [1] Adding header file 'conio.h' [2] Calling function clrscr() after variable declaration and [3] Before closing main function, you need to call getch(); function. You need to repeat in every C-Program is you are Turbo-C user.

```
/* Program:1[B] Hello World on the Console Screen for Turbo Users */
#include<stdio.h>
#include<conio.h>  //Add conio.h file to program
void main()
{
  /*This is my first program */
  clrscr(); //Call function clrscr() to clear screen and previous output
  printf("Hello World:");
  getch(); //Function getch() will pause the screen so that you can see the output
}
```

We will not add the 'conio.h' header file and function clrscr() and getch() into every program. It actually increases the code. To keep our program shorter, simpler and easy to understand we will avoid these lines. But make sure if you use Turbo-C then you need to include 'conio.h' file and call functions clrscr() and getch() in every program as explained.

```
/* Program:2 Finding Simple Interest */
#include<stdio.h>
void main()
{
  int prin, rate, noyears, int;
  printf("Enter Amount:");
  scanf("%d", & prin);
  printf("Enter Rate of Interest:");
  scanf("%d", & rate);
  printf("Enter Years:");
  scanf("%d", & noyears);
  int = ( prin * rate * noyears)/100;
  printf("Simple-Interest is:%d", int);
}
```

Output:

Enter Amount:1000

Enter Rate of Interest:10

Enter Years:5

Simple Interest is:500

Make sure in the above program if you enter value 1000 for Amount, then it is possible that you will get unexpected result. Because the value of (*prin * rate * noyears*) is going beyond 32767. Integer variable cannot store the value more than 32767, so if you are getting unexpected result then you change the formula to:

i = (p/100) *r *n;

In the above program, we have used integer variable. The following program shows the use of floating-point variables. Normally, we measure temperature in either Celsius or Fahrenheit. The program given below will take, Celsius and convert it into Fahrenheit.

```
/* Program:3 Celsius to Fahrenheit conversion */
#include<stdio.h>
void main()
{
   float cel, fah;
   printf("Enter Temperature in Celsius:");
   scanf("%f", &cel);
   fah=(cel * 9) / 5 + 32;
   printf("Fahrenheit Temperature is:%.2f", fah);
}
```

OUTPUT:

Enter Temperature in Celsius:37

Fahrenheit Temperature is:98.60

In the above program, to accept and store Celsius and Fahrenheit temperature we have taken two variable 'cel' and 'fah' of type float so that we can take the temperature in floating-point numbers. We prompt the user to enter temperature in Celsius, and stored it, in 'cel' variable using function scanf(). Variable 'cel' is of type float, so we need to use format string "%f". We use formula   *fah = (cel * 9) / 5 + 32;* to compute Fahrenheit into variable 'fah'. Finally, we print this using printf() statement. We want to print the value of variable 'fah' in two decimal points. So, we are using format string "%.2f". If you use "%f" format string then, it will print: **Fahrenheit Temperature is : 98.600000.**

```
/*Program:4 Finding Greater Number from given two Numbers */
#include<stdio.h>
void main()
{
    int n1, n2;
    printf("Enter First Number:");
    scanf("%d", &n1);
    printf("Enter Second Number:");
    scanf("%d", &n2);
    (n1>n2)?printf("%d is Greater Number",n1): printf("%d is Greater
Number",n2);
}
```

OUTPUT:

Enter First Number: 15

Enter Second Number: 17

17 is Greater Number

In the program we have taken two variables n1 and n2 to store two numbers given by the user. We accept two numbers and store those numbers in n1 and n2 variables respectively using scanf() statement. Finally, using conditional operator (?:), we check the grater number and display it on the console screen. The same thing we can do with three numbers. Consider the following program to find greatest number from given three number.

```
/* Program:5 Finding greatest number from given three numbers
using Conditional Operators*/
#include<stdio.h>
void main()
{
    int x, y, z, max;
    printf("\nEnter Any three Numbers:");
    scanf("%d%d%d",&x, &y, &z);
    max=(x>y) ? (x>z) ? x : z : (y>z) ? y : z;
    printf("Greater Number is:%d", max);
}
```

This program takes three numbers from the user, using scanf() function into three integer variables x, y and z. In this example, we are nesting condition operators. First, we check 'x>y' if yes, then we again check 'x>z', if again yes then x is the greatest number (x>y>z) and we copy x to another variable max. if 'x>z' is False then (x>y and z>x) variable z is the greatest. In the case, where condition x>y is False then competition will continue with variable y and z.

```
/*Program:6 Checking for given number is Even or Odd */
#include<stdio.h>
void main()
{
  int number, r;
  printf("Enter Number:");
  scanf("%d", &number);
  r = number%2;
  if(r == 0)
    printf("%d  is Even Number", number);
  else
    printf("%d is Odd Number", number);
}
```

OUTPUT:

Enter Number: 28

28 is Even Number:

In the program we accept an integer number into variable 'number' using scanf() function. We divide that number by 2 and store it in the variable 'r'. If the value of variable 'r' is 0, that means the number is completely divisible 2 and that why the number is Even otherwise if remainder in variable r is 1 then that number is Odd. In this program, we have used if…else statement. In the next program we will use if…else if…else statement to check the given number is Positive, Negative or Zero.

```
/* Program:7 Checking that the given number is Positive Negative or Zero*/
#include<stdio.h>
void main()
{
  int n;
  printf("Enter Number:");
  scanf("%d", &n);
  if(n>0)
    printf("You have entered %d, which is Positive number:", n);
  else if(n<0)
    printf("You have entered %d, which is Negative number:", n);
  else
    printf("You have entered 0 (Zero):");
}
```

OUTPUT:

Enter Number: 11

You have entered 11, which is Positive number:

In this program we take a number from the user and store it in the variable 'n'. After that we check the number is Positive by evaluating condition (n>0). If yes we print the appropriate message, if no then we again check for negative number by evaluating condition (n<0). If yes then we print that the number is negative. If both conditions are evaluated to be False means (n is not greater than 0 and n is not less than 0). Which means the number is 0, which we have specified in the else part of the if…else if…else condition.

```c
/* Program:8 Finding greatest number using Nested if conditions*/
#include<stdio.h>
void main()
{
  int x, y, z;
  printf("Enter Any 3 Numbers:\n");
  scanf("%d%d%d", &x ,&y ,&z);
  if(x>y)
  {
    if(x>z)
    {
      printf("The greatest number is: %d", x);
    }
    else
    {
      printf("The greatest number is: %d", z);
    }
  }
  else
  {
    if(y>z)
    {
      printf("The greatest number is: %d", y);
    }
    else
    {
      printf("The greatest number is: %d", z);
    }
  }
}
```

In the above program we have taken three numbers from the user. First, we check whether x>y. If yes then we are comparing x with z (x>z), if again yes then x is the greatest number, otherwise z is the greatest number.

If the first condition (x>y) is false, we are comparing y and z variables. If y>z, then y is the greatest number, otherwise z is the greatest number.

In the program we have written if…else condition in another if…else statement, which is called nesting of if-condition. Therefore, the previous program is a good example of nested-If.

```
/* Program:9 Finding greatest number from given 3 numbers using Logical operator
AND (&&)*/
#include<stdio.h>
void main()
{
    int x, y, z;
    printf("Enter Any 3 Numbers:\n");
    scanf("%d%d%d", &x, &y, &z);
    if(x>y && x > z)
        printf("%d is the greatest number", x);
    else if(y>x && y>z)
        printf("%d is the greatest number", y);
    else
        printf("%d is the greatest number", z);
}
```

In this program we have accepted three numbers and stored then in the variable x, y and z respectively. Now using logical AND (&&) operator, we are checking if x>y and x>z then x is the greatest number, otherwise (else if) we check if y>x and y>z then number y is the greatest, in the else part (when x and y both are not greatest then) we print number z is the greatest.

In the 10th program, we have demonstrated goto statement. We know that the goto is unconditional jump, which send the program control on particular labelled statement. Program will start its natural execution and print all the statement in top-down order. It prints the words Apple and Banana. Now before it prints Cherry and Dragonfruit the goto statement sends the program control to the lable end. So rather that it prints Cherry and Dragonfruit, control jump to the label end: and start the execution from that label and will print Emblica.

```
/*Program:10 Example of goto statement */
#include<stdio.h>
void main()
{
   printf("Apple");
   printf("\t Banana");
   goto end;
      printf("\t Cherry");
      ptintf("\t Dragonfruit");
      end:
         printf("\t Emblica");
}
```

OUTPUT:

Apple   Banana   Emblica

```
/*Program:11 Program to demonstrate switch...case statement */
#include<stdio.h>
void main()
{
   int n;
   printf("Enter number from 1 to 4:");
   scanf("%d", &n);
   switch(n)
   {
   case 1:
      printf("1: One");
      break;
   case 2:
      printf("2: Two");
      break;
   case 3:
      printf("3: Three");
      break;
   case 4:
      printf("4: Four");
      break;
   default:
      printf("\nInvalid number entered:");
   }
}
```

OUTPUT:

Enter any number from 1 to 4: 3

3: Three

In the 11th program, we have stored the value which inputted by the user in the variable 'n'. We pass variable 'n' as an expression into switch statement which, match the value with different cases like case 1, case 2 and so on. When the value of 'n' variable is matched with particular case then, that case will be executed. We can see in the output that when we enter 3, which will match with case 3. In this case, statement printf("3: Three"); will be executed. Next statement is break; which will bring program control to the end of the switch…case statement. If user is entering any number which not ranging from 1 to 4, then no case is executed, in such scenario, default: case will be executed and it will print the message that "Invalid number entered".

```
/*Program:12 Checking that the character is Vowel or not */
#include<stdio.h>
void main()
{
   char alph;
   printf("Enter any Lower-Case Letter:");
   scanf("%c", &alph);
   switch(alph)
   {
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u':
      printf("Vowel");
      break;
   default:
      printf("Consonant");
   }
```

OUTPUT:

Enter any Lower-Case Letter: e

Vowel

In the above program if user enters any character from the set {a, e, i, o, u)} then it will print "Vowel". if any other character is entered then the system will execute case default, which will print "Consonant".

```
/* Program:13 Swapping of two variables */
#include<stdio.h>
void main()
{
  int num1, num2, tmp;
  printf("Enter Value for Num1:");
  scanf("%d", &num1);
  printf("Enter Value for Num2:");
  scanf("%d", &num2);
  tmp = num1;
  num1 = num2;
  num2 = tmp;
  printf("After swapping Num1 is: %d and Num2 is: %d", x, y);
}
```

OUTPUT:

Enter Value for X: 28

Enter Value for Y: 11

After swap Num1 is: 11 and Num2 is: 28

In this program, we have taken three variables num1, num2 and tmp. We have taken two values from the user and stored it in the variables num1 and num2. In the first step we copy value of num1 into an extra variable tmp. Then we copy the value of num2 into the variable num1. Finally, we copy the value of tmp into num2 variable. Here, we can get the values of num1 and num2 are swapped or interchanged. To understand, you think a problem where you have 3 glasses. 1st glass has milk and 2nd glass has water in it. 3rd glass is empty. What you will do if you want water to be in 1st glass and milk to be in 2nd glass?

3.1   CHECK YOUR PROGRESS

1.   Write a program to check the given year is a Leap year or not.

2.   Write a program to swap two variables, without taking additional (tmp) variable.

3.   Write a program, which will accept the marks from the user and print the grade accordingly. [From 75 to 100 – 'Distinction', From 60 to 74 – Grade First, from 50 to 59 – Grade Second, from 35 to 49 – Pass and below 35 – Fail]

4.   Write a program to find area and perimeter of rectangle.

5.   Write a program to find square and cube of a given number.

6.   Write a program to find maximum from given 3 values [Do not use conditional operator, logical operator or nested if condition]

3.2 SUGGESTED ANSWERS TO CHECK YOUR PROGRESS

1.   To determine whether the year entered by the user is a leap year or not you need to check, whether the year is a century year or not. If the year is completely disable by 100 (y%100==0) then it is century year. If it is then dividing the year by 400, if it is completely divisible (y%400==0) then the year is leap year otherwise it is not leap year.

In the case, where year is not leap year, you need to divide the year by 4. If it is completely divisible by 4 (y%4==0) then it is leap year, else it is not.

2.   Let two variables are num1=28 and num2=11 then you need to write following instructions:

Initially num1=28 and num2=11

num1=num1+num2 [After Execution of this line num1=39 and num2=11]

num2=num1-num2 [After Execution of this line num1=39 and num2=28]

num1=num1-num2 [After Execution of this line num1=11 and num2=28]

Here, we have swapped values of two variables without taking 3rd variable.

3.   This program can be done by using if…else if…. else ladder statement, or by using logical operators AND (&&):

Method:1 if (m >=75)

      printf("Distinction:");

   else if (m >= 60)

     printf ("Grade: First");

   else if (m >= 50)

     printf ("Grade: Second");

   else if (m >= 35)

     printf("Pass");

   else

     printf("Fail");

Method:2 if (m<=100 && m >=75)

      Printf("Distinction:");

   if (m < 75 && m >=60)

     printf("Grade: First");

if (m < 60 && m >=50)
    printf("Grade: Second");
if (m < 50 && m >=35)
    printf("Pass");
if (m < 35 && m >=0)
    printf("Fail");

4. Declare four variables length, breadth, area and perimeter. Accept the values of variable length and breadth from the user. Compute area=length * breadth and perimeter = 2 * (length + breadth); and display area and perimeter both on the console screen using printf() function.

5. Declare a variable 'num' and accept the value for variable 'num' from the user. Then execute following statement to print square and cube:

printf("\nSquare is: %d and Cube is:%d", num*num, num*num*num);

6. Declare four variable num1, num2, num3 and maximum from the user. Accept three numbers from the user and stored it in the num1, num2, and num3 variable. Write the following code to print max value.

maximum=num1;
if (num2> maximum)
    maximum=num2;
if(num3>maximum)
    maximum=num3;
printf("\n Greatest Number is: %d", maximum);

UNIT 4   SOLVED PROGRAMS - II

In this unit we will try to discuss some programs which will clear your concept related to different types of loops, which available in the C-Language and we have discussed the syntax earlier. After doing all these programs, you will understand the concept of the loops better and you can design any program to solve any complex problem, which needs a loop to be solved.

Syntax of while loop:

```
<initialization>
while (<condition>)
{
  Statement: 1;
  Statement: 2;
        :
        :
  Statement: N;
  <increment / Decrement>
}
```

Let us start with the while loop and initially will make a very simple program. Consider the following program and try to predict its output:

```
/*Program:1 Infinite Loop */
#include<stdio.h>
void main()
{
  int i =1;
  while(i <= 10)
  {
    printf("%d\n", i);
  }
}
```

You might thing the output of this program will be 1 to 10. But think, how the program is executed in the system. We have taken a variable 'i' and initialized it value 1. We have stated the while loop which will check the condition that i<=10. Because the value of variable 'i' is 1 and 1<=10, condition will be evaluated to be True and body part of the loop will be executed, which will print 1 on the console screen. After printing, program counter will be shifted to the header of the loop that the while statement and check the condition again. Because there is no change in the value 'i' the same thing will be repeated that is 1<=10 and it will print again. You can see, the program will print 1, 1, 1, many times. We forgot to increase the value of 'i' variable, and that is the reason we are not getting the desired output that is (1,2,3…10). This type of loop, which runs infinite times is called Infinite loop.

"A loop which never ends, is called an Infinite loop".

Now we will correct the error by writing statement i=i+1 or i++. This statement we will write after printf statement within the loop. This will increase the value of variable 'i' by 1, in each iteration of the loop. See the program given below this will print natural numbers from 1 to 10 on the console screen.

```
/*Program:2 Printing 1 to 10 */
#include<stdio.h>
void main()
{
  int i = 1;
  while(i <= 10)
  {
    printf("%d\t", i);
  i++;
  }
}
```

OUTPUT:

1   2   3   4   5   6   7   8   9   10

In 3ʳᵈ program we will do sum of first 10 natural numbers. Means we will compute 1+2+3+…+10 and print that sum on the console screen. To do this we will take a variable 'i' to run loop from 1 to 10. Another variable sum, which will be initialized with value 0. In each iteration of the loop, we will add the value of variable 'i' to sum. When control will come out of the loop, we will print the value of sum variable on the console screen

```
/*Program:3 Program to compute Sum of first 10 natural numbers */
#include<stdio.h>
void main()
{
  int i, sum;
 i=1;
 sum=0;
  while(i<=10)
  {
    sum=sum+i; //you can also write sum+=i;
    i++;
  }
  printf("Sum of first 10 natural integer numbers is:%d", sum)
}
```

OUTPUT:

Sum of first 10 natural integer numbers is:55

```
/*Program:4 Write a program which print odd numbers from 20 to 40*/
#include<stdio.h>
void main()
{
   int i=20;
   while(i<=40)
   {
     if(i%2==1)
        printf("%d\t", i);
     i++;
   }
}
```

OUTPUT:

21   23   25   27   29   31   33   35   37   39

        In this program, we start the loop by initializing loop variable 'i' with value 20. We run the loop till i<=40 and in each iteration value of 'i' is incremented by 1. Make sure, this loop will run for 20, 21, 22, …40. But we don't have to print all the values, we need to print only those numbers which are odd. To do this we have placed printf statement in the if condition which will check if variable 'i' is not divisible by 2 (i%2==1) then and then value of variable is printed on the console screen.

```
/*Program: 5 Write a program to reverse the given number */
void main()
{
   int num, rnum=0, tmp;
   printf("\nEnter Number:");
   scanf("%d", &num);
   while(num > 0)
   {
     tmp=num%10;
     num=num/10;
     rnum=rnum*10+tmp;
   }
   printf("The Reverse Number is:%d", rnum);
}
```

OUTPUT:

Enter Any Number: 2345

Reverse Number is: 5432

        In the 5th program, we have declared three variables num, rnum and tmp. We have initialized the rnum variable to 0 and taken a value for the num variable

from the user. Now, we are running a loop till variable does not becomes 0 that means rnum >0. In each iteration we are dividing a number by 10 and storing it in the tmp variable. we are dividing a number by 10, and finally we are computing rnum=rnum*10+tmp. Finally, when the loop completes it's all iterations, we are printing the value of rnum variable on the screen. To understand the execution process of the loop let us take an example, where user has entered a number 1234. So, our num variable is 2345 and rnum variable is 0.

```
Iteration: 1      while (num > 0 ) // TRUE 2345 > 0
             tmp=num % 10  // tmp is know 2345%10 =5
             num=num/10 // num=2345/10=234
             rnum=rnum*10 + tmp //rnum=0*10+5=5
Iteration: 2      while (num > 0 ) // TRUE 234 > 0
             tmp=num % 10  // tmp is know 234%10 =4
             num=num/10 // num=234/10=23
             rnum=rnum*10 + tmp //rnum=5*10+5=54
Iteration: 3      while (num > 0 ) // TRUE 23 > 0
             tmp=num % 10  // tmp is know 23%10 =3
             num=num/10 // num=23/10=2
             rnum=rnum*10 + tmp //rnum=54*10+3=543
Iteration: 4      while (num > 0 ) // TRUE 2 > 0
             tmp=num % 10  // tmp is know 2%10 =2
             num=num/10 // num=2/10=0
             rnum=rnum*10 + tmp //rnum=543*10+2=5432
Iteration: 5      while (num > 0) //FLASE num=0 and 0 is not > 0.
Print the value of rnum variable that 5432 on the console screen.
```

```c
/*Program: 6 Program to check the given number is Prefect number of Not */
     #include<stdio.h>
     void main()
     {
        int n, i=1, fs=0;
        printf("Enter Number:");
        scanf("%d", &n);
        while(i<n)
        {
           if(n%i==0)
              fs=fs+i;
           i++;
        }
        if(fs==n)
           printf("\nGiven Number is Perfect Number");
        else
           printf("Given Number is not Perfect number:");
     }
```

OUTPUT
Enter Number:28
Given Number is Perfect Number

        Sum of all the factors of a particular number is equal to that number is called perfect number. For example, 28 and 6 are perfect numbers. Factors of 6 are 1, 2 and 3. Sum of factors 1+2+3=6. Similarly, factors of 28 are 1, 2, 4, 7, and 14. If we do sum of all factors, we will get 28. In the program we have taken 3 variables n, i and fs. Variable i initialize with 1 and fs initialize with 0. We are running a loop from 1 to less than of that number (n), using variable i. Whenever we get any i from which number n is divisible (n%i==0), we are adding the value of i to fs variable. Finally, we are comparing if fs and n are equal then given number is Perfect otherwise not perfect.

```
/*Program: 7 To check the given number is Prime number of Not */
#include<stdio.h>
void main()
{
   int n, i=2, boolean=1;
   printf("Enter Number:");
   scanf("%d", &n);
   while(i<n)
   {
      if(n%i==0)
      {
         boolean=0;
         break;
      }
   i++;
   }
   if(boolean==1)
      printf("Given Number is Prime");
   else
      printf("Given Number is Composite");
}
```

OUTPUT:
Enter Any Number: 31
Given Number is Prime

In 7th program, we are inspecting whether the number entered by user is prime or composite. A number which is divisible by 1 or itself, and not divisible by any other number (factor or that number) is called prime number. We have taken three variables n, i and boolean. We are initializing loop variable i with 2 because we will try to divide number n by 2, 3, 4 and up to n-1. We set boolean to 1 (it is an assumption that number n is prime). Now, will try to divide that number n by 2 to n -1 using while loop. If for any value of i, the number n is divisible (n%i==0), then we set boolean to 0 (that means the number is divisible and we made a wrong assumption). If n is divisible by any value of I then, we do not have to continue loop, so here we have used keyword 'break'. When program control come out of loop then, if logic is 1, the n is not divisible by any value of i, so it is prime, otherwise n is divisible by some value of i, and hence it is composite.

```
/*Program: 8 Write a program to print prime numbers from 1 to 50 */
#include<stdio.h>
void main()
{
   int n=2, i, boolean;
   printf("Prime numbers from 1 to 50:\n");
   while(n<=50)
   {
     i=2;
     boolean=1;
      while(i<n)
     {
        if(n%i==0)
        {
           logic=0;
           break;
        }
        i++;
     }
   if(boolean==1)
     printf("%d\t", n);
   n++;
   }
}
```

OUTPUT:
All Prime numbers from 1 to 50:
2   3   5   7   11   13   17   19   23   29   31   37   41   43   47

In the 7th program, we check the number given by the user is prime or composite. To do this, obviously we have used a while loop. In this program, rather than accepting the value of the variable n from the user, we need to generate numbers from 2 to 50 using another while loop. For each value of n (2 to 50), we check that value is prime or composite, if the value is prime then, it will be printed on the console. In this program, we need to run one while loop to generate numbers from 2 to 50, and for each value of variable n, we need a second while loop (of variable i) to test that value is prime or composite. In short, we have placed a while loop inside another while loop. This is called a **nesting of while loop**.

Now, we made sufficient examples of while loop. So, now we will do some programs of 'for' loop. For-loop is a popular loop as it reduces, number of programming or coding lines. The syntax for writing for loop is as follows:

For loop Syntax:

```
for (<initialization> ; <condition> ; <increment/decrement>)
{
        Statement: 1;
        Statement: 2;
                :
                :
        Statement: N;
}
```

In the next program we will print all even numbers from 1 to n. Here n is the number which is entered by the user.

```
/*Program:9 Program to print even numbers from 1 to N*/
#include<stdio.h>
void main()
{
   int num, i;
   printf("Enter Number:");
   scanf("%d", &num);
   for(i=1;i<=num;i++)
   {
     if(i%2==0)
        printf("\t%d", i);
   }
}
```

OUTPUT:

Enter Number: 20

2   4   6   8   10   12   14   16   18   20

In the following program, we take an integer number form the user and compute the factorial of that number.

```
/*Program:10 Program to find factorial of given number*/
#include<stdio.h>
void main()
{
    int num, i, fact;
    printf("Enter Number:");
    scanf("%d", &num);
    for(i=1, fact=1;i<=num ;i++)
    {
        fact*=i;
    }
    printf("Factorial of Number %d is = %d", n, fact);
}
```

OUTPUT:

Enter Number: 5

Factorial of Number 5 is = 120

In this program we have started for loop by initializing i=1 and fact=1. We have accepted a number from the user, and stored it in the variable num. We run a for loop till loop variable i<=num. Here, we multiply fact and i variable and stored it in the fact variable itself (fact*=i;). When for completes it's all iteration we put the value of the fact variable on the console screen.

In the next program we are taking an integer value from the user and will check that integer number is Palindrome or not. Palindrome number are those number whose reverse is equal to that number. For, example reverse of 23432 is 23432, or reverse of 121 is 121.

In this program, we have taken variable n to accept a number from the user. We have taken another variable on to copy variable n (on=n). One variable tmp to store remainder after dividing n by 10, and rn variable, which will be initialized with 0, and it will help to find reverse number.

After taking a number from the user will store it into variable n. We will copy this value into another variable on (original number) and then we will find reverse of number n into rn (reverse number) variable. Once we get reverse number, we will match it with variable 'on'. If the value of variable 'on' and 'rn' are same then, that number is Palindrome otherwise it is not Palindrome.

```
/*Program:10 Program to check given number is Palindrome or Not*/
#include<stdio.h>
void main()
{
  int n, on, tmp, rn=0;
  printf("Enter Number:");
  scanf("%d",&n);
  for (on=n ; n>0;n=n/10)
  {
    tmp=n%10;
    rn=rn*10+tmp;
  }
  if(on==rn)
    printf("Palindrome Number");
  else
    printf("Not Palindrome Number");
}
```

OUTPUT:
Enter Any Number:12321
Palindrome Number

In the 11th program, we accept a number from the user and will check whether it is Armstrong number or not. Armstrong number is that number, whose sum of cube of each digit is equal to that number. For example, 153 is Armstrong number because $1^3+5^3+3^3=1+125+27=153$.

To implement this, we have taken four variables n, on, tmp and sum. We will take the value of n variable from the user and copy it to variable 'on'. We will start the for loop by initializing sum=0 and on=n. We will check the condition $n > 0$. Means when n variable turns to 0, we will stop the execution of loop. In each iteration n variable will be divided by 10 (n=n/10). In the loop first we will computer remainder after dividing n by 10 (tmp=n%10), and cube of variable tmp will be added to the variable sum. sum=sum+(tmp*tmp*tmp). When all iteration of for loops are completed then, we will match the value of sum variable is equal to 'on' variable? if it is then the given number is Armstrong number, otherwise it is not.

```
/*Program:11 Program to check given number is Armstrong or Not*/
#include<stdio.h>
void main()
{
  int n, on, tmp, sum;
  printf("Enter Number:");
  scanf("%d", &n);
  for(on=n, sum=0; n>0; n=n/10)
  {
    tmp=n%10;
    sum=sum+(tmp*tmp*tmp);
  }
  if(on==sum)
    printf("Armstrong Number");
  else
    printf("Not Armstrong Number");
}
```

OUTPUT:

Enter Number:153

Armstrong Number

Another program is using a nested for loop to print a square of character *. In this program depending upon value entered by the user, we print a square of 2*2, 3*3 using character '*'.

```
/*Program:12 Program to draw square shape using star character*/
#include<stdio.h>
void main()
{
  int n, row, col;
  printf("Enter Number:");
  scanf("%d", &n);
  for(row=1;row<=n; row++)
  {
    for(col=1; col<=n; col++)
    {
      printf("* ");
    }
    printf("\n");
  }
}
```

In this program, we have taken three variables n, row, and col. We take the value from the user to initialize variable n. Now we, are starting a for loop for variable row. Loop 'for(row=1; row<=n; row++)' will print number of rows, as the number by the use (variable n). to print row new line character is needed, so we have mentioned 'printf("\n"); statement in the for loop of variable 'row'. In each row we need to print 'n' columns. For that we have written another for loop of variable 'col' that is: 'for(col=1; c<=n; col++)'. The program produces the output as shown below:
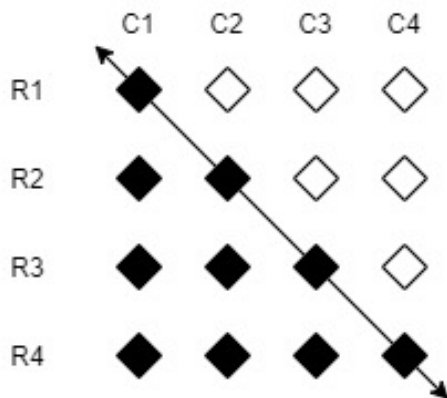
OUTPUT:
Enter Number:3
✳ ✳ ✳
✳ ✳ ✳
✳ ✳ ✳

In the next program, depending upon user's input value, we are printing right angle triangle using character '*'. To do this we will make some changes in the above program.

In the previous program we have learn how to draw square. The same program we will use to draw right-angle triangle shape with some modification. By looking to the figure given as right, it is clear that to print rectangle we need row= n and col=n. Now to draw right angle triangle, we need to draw 1 col (1 star) in the 1st row, 2 col (2 stars) in the second row and so on.

So, it is clear that if we modify a loop of variable col from: 'for (col=1; col<=**n**; col++)' to 'for (col=1; col<=**row**; col++)'. So, make this change in the program to get the desired output.

```
/*Program:13 To draw square shape*/
#include<stdio.h>
void main()
{
   int n, row, col;
   printf("Enter Number:");
   scanf("%d", &n);
   for(row=1; row<=n; row++)
   {
      for(col=1; col<=row; col++)
      {
         printf("* ");
      }
      printf("\n");
   }
}
```

OUTPUT:
Enter Number: 3
*
* *
* * *

       Now in the next program, we need to print triangle by doing some changes into the above program. To draw the triangle, we have to give spaces before printing of each row. But how many spaces we have to give for each row? Well, that is exactly (n-row) spaces. That means, if user want to draw total 5 rows, so value of n=5. For 3rd row (row=3) we need to print (n-row=5-3=2 space) before we start printing columns on row 3. In each row, we have to run a loop which will print (n -row) spaces, before we are executing a loop of col variable, which is printing '*' character. In each row, if we are giving (n -row) spaces, and then we are running a loop of col variable, which will print, row number of '*' symbols. Make sure to achieve this task, we have to add a space in the printf statement after *. That means we need to write 'printf("* ");'. In the printf statement we have written a start and a space. If we compile these modifications in the above program then, it will print triangle shape of the symbol '*' instead of right-angle triangle. In this program we will take a loop for variable row which will run for n time. In that loop we will pace another for loop which runs for n-row times to print spaces, and finally we will place another loop for variable col, which will print * and a space, which will run from 1 to row times.

```
/*Program:14 Program to draw triangle shape*/
#include<stdio.h>
void main()
{
    int n, row, col, space;
    printf("Enter Number:");
    scanf("%d", &n);
    for(row=1; row<=n; row++)
    {
        for(space=1; space<=n-row; space++)
            printf(" ");
        for(col=1; col<=row; col++)
        {
            printf("* ");
        }
        printf("\n");
    }
}
```

OUTPUT:

Enter Number: 3
```
   *
  * *
 * * *
```

Now, in the next program (Program:15) we will do one more modification in the last program. To do your program number 15, remove the space you have given in the previous program in the printf() statement. Instead of writing printf("* <<Space>>"); write printf("*"); and see the output.

By making a small change in the printf () statement of the program:14 you can make different types of patterns. Please follow the following table in which program number, change in printf() statement and its output is shown.

Program	Change in printf() Statement	Output
16	printf("%d ", row);	Enter Number: 3 1 2 2 3 3 3
17	printf("%d ",col);	Enter Number: 3 1 1 2 1 2 3

Program	Change in printf() Statement	Output
18	printf("%c ", row+64);	Enter Number: 3 A B B C C C
17	printf("%c ",col+64);	Enter Number: 3 A A B A B C
18	printf("%c ", row+96);	Enter Number: 3 a b b c c c
19	printf("%c ", col+96);	Enter Number: 3 a a b a b c
20	printf("%d ", row%2);	Enter Number: 3 1 0 0 1 1 1
21	printf("%d ", col%2);	Enter Number: 3 1 1 0 1 0 1
22	printf("%d ", (row+col)%2);	Enter Number: 4 0 1 0 0 1 0 1 0 1 0

Number of different types of patterns can be generated by doing a small change as directed by the table given above in the Program:13, Program:14 and Program:15. Now, we have made sufficient practice for the "for loop". We hope after doing these examples you have clear idea about how to use "for loop" in the C-Programming language. Let us, resume our discussion and will see some programs of do…while loop.

DO…WHILE LOOP:

As we have discussed that the do…while loop is an Exit-Control loop. Usually, the loop will be repeated if the condition specified is True. When condition specified with while becomes False, do…while loop stop the repetitions. In this loop condition is evaluate at the end, that means the loop will be executed at least once even though its initial condition is False. Make sure, while using do…while loop we need to specify semi-colon (;) at the end of the loop.

The syntax of do…while loop is as follows:

<Initialization>
do
{
   Statement:1
   Statement:2
   Increment statement
} while (<condition>);

Consider the following example, where we need to take numbers from the user until, user does not enter 999. When user enters 999, program has to show, number of positive values, number of negative values and number of zeros entered by the user. In this type of situation, where we do not know exactly how many times, we need to run a loop and we need to execute a loop at least once, do … while loop is suitable.

```
/* Program:23 Count No of Positive, Negative and Zeros */
#include<stdio.h>
void main()
{
   int num, positive=0, negative=0, zero=0;
   do
   {
     printf("Enter Number [Enter 999 to Exit]:");
     scanf("%d", &num);
     if(num!=999)
     {
       if(num>0)
         positive++;
       else if(num<0)
         negative++;
       else
         zero++;
     }
   } while(num!=999);
   printf("You have entered %d positive, %d negative,%d zero",positive, negative,zero);
}
```

OUTPUT:
Enter Number [Enter 999 to Exit]:5
Enter Number [Enter 999 to Exit]:10
Enter Number [Enter 999 to Exit]: -60
Enter Number [Enter 999 to Exit]:20
Enter Number [Enter 999 to Exit]:0
Enter Number [Enter 999 to Exit]:23
Enter Number [Enter 999 to Exit]:999
You have entered 4 positive, 1 negative,1 zero

In the previous programs of this Unit, we have focused on while, for and do…while loop. We have also seen nesting of while and for loop. Now we try to focus on two keywords, which can be used to control loop. These keywords are [1] Break and [2] Continue

[1] Break keyword:

Keyword 'break' is used to terminate the premature loop. For example, if we are writing a program to check given number is prime or composite, we start dividing number by 2, 3, 4… up to that number-1. If we get a single number by using, we can divide the number entered by the user then we have to exit the loop. When the system will execute the 'break' statement flow control will terminate the loop and comes out of the loop.

```
/*Program:24 Understanding break keyword*/
#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        if(i==3)
            break;
        printf("%d\t",i);
    }
}
```

OUTPUT:

1   2

In the above example, loop has to print from 1 to 10. But when i=3, if condition becomes true and break statement gets executed. Once the break statement gets executed, loop will be terminated, and 3 to 10 numbers will not be printed. This is called termination of premature termination of loop.

Recall the program to, checking the number is prime or not. In that case we are trying to divide a number (num variable) by 2, 3, … num-1. For example, if we want to test 35 is prime or not, we will divide 35/2, 35/3,35/4,35/5… now here 35 is divisible by 5, and will come to know that 35 is not a prime number. In this case we do not have to check whether 35 is divisible by 6 or 7 or 8…and so on. Here we have to break, premature loop. In this type of case break statement is useful.

[2] Continue keyword:
When the 'continue' statement will be executed then flow of the control is transferred to the header of loop. In this case statements written below 'continue' will not be executed.

```
/*Program: 25 Understanding continue keyword */
#include<stdio.h>
void main()
{
  int i;
  for(i=1;i<=10;i++)
  {
    if( i>=4 && i<=8)
      continue;
    printf("\n%d", i);
  }
}
```

OUTPUT:
1
2
3
9
10

Now, we will see few more programs, so that you can make your programming practice much stronger. I think by doing previous programs, you have now sufficient development skill that you can understand the source code easily. So, for rest of the program we are just providing source code and not the discussion.

/* Program: 26 WRITE A C PROGRAM TO FIND WHETHER THE CHARACTER ENTERED BY THE USER IS A CAPITAL LETTER OR A SMALL LETTER OR A DIGIT OR ANY SPECIAL SYMBOL. */
```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    char ch;
    printf("Enter the character:");
```

```c
    ch=getchar();
    if(isdigit(ch))
    {
        printf("CHARACTER ENTERED IS A DIGIT.");
    }
    else if(isupper(ch))
    {
        printf("CHARACTER ENTERED IS A CAPITAL LETTER.");
    }
    else if(islower(ch))
    {
        printf("CHARACTER ENTERED IS A SMALL LETTER.");
    }
    else if(ispunct(ch))
    {
        printf("CHARACTER ENTERED IS A SPECIAL SYMBOL.");
    }
    else
    {
    exit(0);
    }
}
```

OUTPUT:

Enter the character: A

CHARACTER ENTERED IS A CAPITAL LETTER.

/*Program: 28 WRITE A C PROGRAM TO PRINT A CONVERSION TABLE OF FARENHEITES TO CENTIGRADES. */

```c
#include<stdio.h>
void main()
{
    float i,C=0,F,no=1;
    printf("Enter the value of Fahrenheit:");
    scanf("%f",&F);
    printf("\nFARENHITE\t\CENTIGRADE\n");
    for (i=1; i<=F;i++)
    {
        printf("\n%.2f\t" ,i);
        C=(i-32)/1.8;
        printf("\t%.4f", C);
    }
}
```

OUTPUT:
Enter the value of farenhite:5s

FARENHITE	CENTIGRADE
1.00	-17.2222
2.00	-16.6667
3.00	-16.1111
4.00	-15.5556
5.00	-15.0000

/*Program: 29 WRITE A C-PROGRAM TO GENERATE THE MULTIPLICATION TABLE FOR ANY GIVEN NUMBER. */

```c
#include<stdio.h>
void main()
{
    int x,y=1,mul;
    printf("Enter the number:");
    scanf("%d",&x);
    do
    {
        mul=x*y;
        printf("%d * %d = %d\n",x,y,mul);
        y++;
    } while(y<=10);
}
```

OUTPUT:

```
Enter the number:5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

/* Program: 30 WRITE A C PROGRAM TO ADD THE FIRST N TERMS OF THE FOLLOWING SERIES USING FOR LOOP: 1/1!+1/2!+1/3!+.....*/

```c
#include<stdio.h>
void main()
{
   float n=0, ans=1, i, j, temp;
   printf("\n Enter the Value of N:");
   scanf("%f",&n);
   for(i=1,temp=1;i<=n;i++)
   {
     for(j=1;j<=i;j++)
     {
        temp=temp*j;
     }
     ans=ans+(1/temp);
   }
   printf("\nThe Answer is %.2f ",ans);
 }
```

OUTPUT:

Enter the value of N: 4

The Answer is 2.59

/* Program: 31 WRITE A PROGRAM OF PYRAMID
4 4 4 4
3 3 3
22
1
*/

```c
#include<stdio.h>
void main()
{
   int i,j,n;
   printf("Enter N:");
   scanf("%d",&n);
   for(i=n;i>=1;i--)
   {
     for(j=1;j<=i;j++)
         { printf("%d ",i); }
     printf("\n");
   }
}
```

```
/* Program:32 WRITE A PROGRAM OF PYRAMID
      *
    * *
   * * *
  * * * *
   * * *
    * *
     *                                    */
#include<stdio.h>
void main()
{
   int i, j, k, n;
   printf("Enter N:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      for(k=1;k<=n-i;k++)
      {
         printf(" ");
      }
      for(j=1;j<=i;j++)
      {
         printf("* ");
      }
      printf("\n");
   }
   for(i=n-1;i>=1;i--)
   {
      for(k=1;k<=n-i;k++)
      {
         printf(" ");
      }
      for(j=1;j<=i;j++)
      {
         printf("* ");
      }
      printf("\n");
   }
}
```

```c
/* Program: 33 WRITE A C PROGRAM TO PRINT THE PYRAMID AS
FOLLOWS.
    1
   121
  12321
 1234321
123454321                              */
#include<stdio.h>
void main()
{
  int i, j;
  for(i=1;i<=5;i++)
  {
    printf("\n");
    for(j=1;j<=5-i;j++)
    {
      printf(" ");
    }
    for(j=1;j<=i;j++)
    {
      printf("%d",j);
    }
    for(j=i-1;j>=1;j--)
    {
      printf("%d",j);
    }
  }
}
// Program:34 WRITE A PROGRAM TO PRINT A FIBONACCI SERIES
#include<stdio.h>
void main()
{
  int a=1,b=0,c=1,n,i;
  printf("Enter The range : ");
  scanf("%d",&n);
  printf("%d\t%d\t",b,c);
  for(i=1;i<n-1;i++)
  {
```

```c
      a=b+c;
      b=c;
      c=a;
      printf("%d\t", a);
   }
}
```

OUTPUT:

Enter The range: 7

0   1   1   2   3   5   8

/*Program: 35 WRITE A PROGRAM TO PRINT A SERIES LIKE SUM 1+ ¼ + 1/9 +1/16+…….+N */

```c
#include<stdio.h>
#include<math.h>
void main()
{
   int n;
   float b,c=0,denom,d=0;
   printf("enter the no for series:");
   scanf("%d",&n);
   denom=1;
   while(denom<=n)
   {
      d=pow(denom,2);
      b=(1/d);
      c=c+b;
      denom++;
   }
   printf("the sum of the series is:%f",c);
}
```

OUTPUT:

enter the no for series:2

the sum of the series is:1.250000

/* Program:37 WRITE A PROGRAM TO PRINT SUM= 1+4-9+16-25+….+N */

```c
#include<stdio.h>
void main()
{
    int n, sum=0, num=0, i;
    printf("Enter the no for series:");
    scanf("%d", &n);
    for(i=2; i<=n; i++)
    {
        num=i*i;
        if(i%2==0)
        {
            sum=sum + num;
        }
        else
        {
            sum=sum+(-num);
        }
    }
    sum=sum+1;
    printf("The sum of the series is:%d", sum);
}
```

OUTPUT:
Enter the number of series: 2
The sum of the series is: 5

Block Summary

- "Decision making" is one of the most important concepts of computer programming.
- Many Programs require testing of some conditions at some point in the program and selecting one of the alternative paths depending upon the result of condition. This is known as Branching.
- The branching may be unconditional or conditional.
- C language provides statements that can alter the flow of a sequence of instructions.
- The "if" statement is two way decision statement used to control the flow of execution.
- The if-else statement is extension of simple if statement.
- When series of decisions are involved, then nested if-else statement can be used. For this, well known construct called else-if ladder is used.
- The switch statement may be thought of as an alternative or replacement to the use of nested if-else statements.
- The switch statement is used to select a particular group of statements to from several available alternatives.
- Depending upon the current value of an expression that is included within a switch statement, group of statements are selected.
- The conditional operator (?:) can be used instead of simple if-else statement.
- The goto statement is used to change the normal flow of program execution by transferring control unconditionally to some other part of the program.
- Loops can be used to perform certain task for a number of times or execute same statement or a group of statements repeatedly.
- There are two types of loops, counter controlled and sentinel controlled loops.
- Counter controlled repetitions are the loops in which the number of statements repeated for the loop is known in advance.
- Sentinel loops are executed until some condition is satisfied. Condition can be checked at top or bottom of the loop.
- C language provides three different types of loop – while loop, do-while loop, for loop.
- The while loop is Sentinel loops or top-tested or entry controlled loop.
- While loop executes group of statements until test condition is true. The condition is checked at the top of the loop.
- The do while loop is similar to while loop, but the test occurs at the end of loop.
- "do while loop" that the loop body is executed at least once.

- The for-statement is another flexible entry controller loop.
- The "for loop" is mostly used, when we know in advance that the loop will be executed a fixed number of times.
- A loop within a loop is called nested loop.
- The inner and outer loops need not be generated by the same type of control structure.
- It is essential that one loop should be completely embedded within the other.
- The break statement is used to terminate loops or to exit a switch.
- If break statement is in the innermost loop, then inner loop will be terminated immediately.
- The continue statement is used to skip or to bypass some step or iteration of looping structure.
- It only works within loops where its effect is to force an immediate transfer to the loop control statement.

BLOCK ASSIGNMENT

Short Questions:

1) List the control statements that support branching.
2) What is syntax of if-else statement?
3) What do you mean by Counter controlled and Sentinel loop?
4) What is use of break and continue statement?
5) What is use of goto statement?
6) What is syntax of for loop?

Long Questions:

1) Explain switch statement in detail with example.
2) Explain in brief different types of loop constructs in C.
3) Explain different types of jump statements in C.

BLOCK 3: ARRAYS AND FUNCTIONS

Block Introduction

An Array is a collection of same type of elements under the same variable identifier referenced by index number. Arrays are widely used within programming for different purposes such as sorting, searching and etc. Arrays allow you to store a group of data of a single type.

These are efficient and useful for performing operations. You can use them to store a set of high scores in a video game, a 2-dimensional map layout, or store the coordinates of a multi-dimensional matrix for linear algebra calculations.

There are two types arrays single dimension array and multi-dimension array. Each of these array types can be of either static array or dynamic array. Static arrays have their sizes declared from the start and the size cannot be changed after declaration. Dynamic arrays that allow you to dynamically change their size at runtime, but they require more advanced techniques such as pointers and memory allocation.

A single dimension array is represented by a single column, whereas a multiple dimensional array would span out n columns by n rows. In this block, you will learn how to declare, initialize and access single and multi-dimensional arrays.

These arrays are discussed in the 1st and 2nd units. The usages of arrays help in tackling with less no. of variables.

In 3rd unit, we have discussed about functions, which are also helpful in reducing the no. of statements in a program. The different methods of defining functions and return types are well explained in the unit which will definitely help the learners to understand these concepts easily.

Recursion is a function that calls itself. In recursive function the instructions are repeated. It is similar like loop which repeats the same code. Recursion makes it easy and result of recursive call is necessary to complete the task. This concept is also well-explained in the 3rd unit which will help you in solving problems recursively.

The 4th unitcontains some solved programs based on the statements/ concepts explained in the first 3 units. This will help the learners to understand those concepts in details as the problems are practically solved.

Block Objective

Main objective of designing this Block is to teach, what is arrays? And How can we implement the array in to the C-Programming language. At the end of the 1st chapter student will learn about declaration and initialization of an array. Students will also be able to handle 2-dimenssional array and can represent matrices in the C-Programming language. Student will learn how to handle strings? Using character types of arrays, various string functions etc. in 2nd chapter.

Another important aspect of designing this block is to provide knowledge about functions. C-Language is a function-oriented language. How students can make their own User Defined function? What is basic structure of User Defined function? And how can we call it. After learning this Block student will able to learn modular programming approach and can design modular and readable program with very less complexity.

Block Structure

BLOCK 3: ARRAYS AND FUNCTIONS

UNIT1   ARRAYS

Objectives, Introduction, Understanding arrays, One-Dimensional array, Operations on arrays, Two-Dimensional array, Let Us Sum Up

UNIT 2   HANDLING STRINGS

Objectives, Introduction, Understanding strings, Displaying strings in different formats, Standard functions of string handling, Table of strings, Let Us Sum Up

UNIT 3   FUNCTIONS

Objectives, Introduction, Need for User Defined Functions, A Multifunction Program, The Form of C Functions, Return values and their types, Calling of Functions, Category of Functions, Let Us Sum Up

UNIT 4   MORE ABOUT FUNCTIONS

Objectives, Introduction, Handling of non-integer functions, Nesting of Functions, Recursion, Function with Arrays, Scope and Lifetime of Variables in Functions, ANSI C Functions, Let Us Sum Up

UNIT 1   ARRAYS

Unit Structure

1.0   LEARNING OBJECTIVES

Up to here, many programs we have made to practice conditional statements and loops. In most programs, we have taken two to three variables to solve the specific programs. But consider a program, in which you need to take 100s of values or character. In such situation Arrays are used. In this chapter we will try to understand Array as well as we will learn how to deal with multi-dimensional arrays.

After working through this unit, you should be able to:

- Learn, how to store large number of values into the single unit called array?

- Understand, how to declare and initialized arrays?

- Know different terms related to array.

- Learn how to perform different operations related to array.

- Know, how 2-dimenssional array is used to represent matrices.

1.1   ARRAY INTRODUCTION:

The term Array can be defined as finite and ordered collection of homogeneous data. The term homogeneous means similar type of data. Array is a collection of same type of data, which are stored under common name, and stored in the consecutive memory locations.

1.2   UNDERSTANDING ARRAY

1.2.1 Array definition

In the C-Language, if we write '***int a [20];***' in the declaration section of the program then we have a single array variable named 'a' which can store 20 elements of same type integer.

1.2.2 Array initialization

Initialization of an array can be done in three different ways:

1. Array can be initialized at the time of its declaration

int a [10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

char b [5] = {'a', 'e', 'i', 'o', 'u'};

In the above code, 'a' is an integer array with size 10, and it is initialized by data elements 2, 4, 6, 8, 12, 14, 16, 18, and 20. First data-element 2 will be stored on the 0th position of an array 'a'. That means, a [0] is 5. Similarly, 4 is stored on 1st position, 6 is stored on 2nd position and so on. Finally, data-element 20 will be stored on 9th position. In C-Language index of an array starts from 0. So, last element we can find on the position Size – 1. In this case, the size of an array 'a' is 10. Therefore, last data element '20' will be stored on position 9.

In second declarative statement, we have declared an array 'b' of type character, in this array we have stored five character type data-elements like, 'a', 'e', 'i', 'o' and 'u'. Similar to array 'a', first data-element 'a' will be stored on 0th position and character 'u' will be stored on 4th position. Make sure, character values should be encased in single quotation mark like: 'k'.

2. Declaration and static value assignment to array elements:

int a [5];

char b [5];

a [0] =5;

a [1] = 10;

a [2] = 15;

and so on. Similarly, array 'b' can be initialized as,

b [0] = 'a';

b [1] = 'e';

and so on.

3. Array initialization by user input

int a [10], i ;

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

{

printf ("Enter Number:");

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

}

1.2.3 Terminologies of an Array

4. **Type:** The term type of an array represents type of data, that can be stored in the array. For example, array can be of type char, int, float, double etc.

5. **Size:** The term size of an array represents maximum number of data elements that can be stored (accommodated) in an array. For example, if we declare 'int a [10]' then array 'a' can store maximum 10 elements.

6. **Index:** The term index refers to a position of particular data element in the array. In C-Language array start from index 0, that means the first data-element of an array is stored on position 0 of an array.

7. **Range:** The term range refers to the range of index numbers. We know that the index number starts from 0. It called the lower bound of an array. The index number of the last element is called upper bound of an array. For example, if we declare: int arr[10], then the range of an array arr is 0-9. Lower bound of arr is 0 and upper bound of arr is 9.

8. **Base:** The term Base or Base address refers to the starting memory location of an array. It refers the memory location of first element of an array. In array, name of the array is used to refer base address (To find base address we don't have to use & operator).
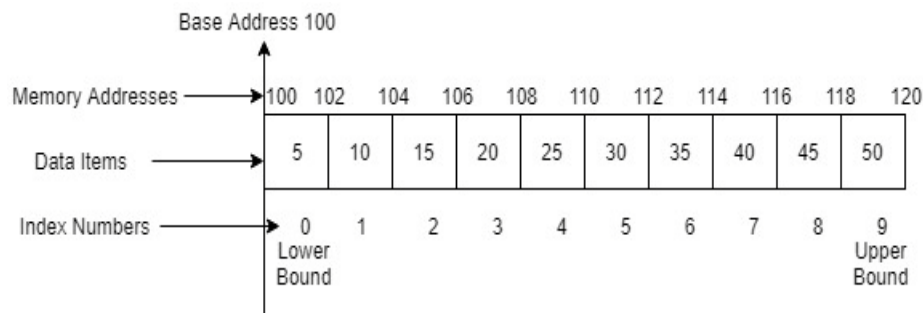
In the figure given above, 100, 102, … 120 are the addresses of memory locations. Because our array starts from memory address 100, it is the base address of an array. In the figure, 0, 1, …9 are the index numbers, which represents position of a data-element in the array. As we know, array starts from index number 0, that means it is lower bound of an array and the index of the last data-element is 9, which is known as upper bound of an array. Declaration of an array is: int arr[10]; Therefore type of the array is int, and size of the array is 10. Which means this array 'arr' can store, 10 integer number in it. As we know int variable reserves 2 bytes of space in the memory, therefore first data-element 5 will be stored from memory location 100 to 102, second data-element 10 will be stored from 102 to 104 and so on.

If we have an array having LB as Lower bound and UB as upper bound. Then the index number of an ith element will be always: Index $(Xi) = LB + i -1$. For example, in C-Language index number of 10th element will be $0 + 10 -1 =9$.

Similarly, array size can be measured as: Size $= UB – LB + 1$. In C-Language if upper bound UB of and array is 9, then the size $= 9 – 0 + 1 =10$.

If you want to know the memory address of ith data-element then, the formula is: Address of ith data-element $=$ Base address $+$ (Index - 1) * size of element. than means from the previous figure: 5th data Element is stored at address: $100 + (5-1) * 2=100+4*2 = 108$. Here, 100 is the base address, 5th data-element is stored on index 5-1=4 and size of data element is 2 as the type of an array is int (2 Bytes).

<div style="border:1px solid black; padding:10px">

Check Your Progress-2

1.Size on an array, is mentioned in _____.

[A] [ ] Square bracket   [B] { } Curly bracket

[C] ( ) Parenthesis   [D] None of the above

2.The address of 5th element, in the array with 750 base address of long integer type of size 10 is _____.

[A] 770   [B] 760

[C] 758   [D] 766

3. Total memory size required to store integer type array of size 10 is _____.

[A] 10 Bytes   [B] 20 Bytes

[C] 40 Bytes   [D] 5 Bytes

</div>

1.3 ONE-DIMESSIONAL ARRAY

Array can be one-dimensional (linear) or two-dimensional (matrix). One-dimensional array is that, in which each element of an array can be represented with single subscript or index number. Array shown in the previous figure, is one-dimensional array. Value and Address of any data-element having index 'i', in the array 'arr' can be found by following equations:

Value of element can be accessed as: arr [i]   (1)

Address of arr [i] = Base address of an array + i * size of element   (2)

For example, an array with base address 750 of type integer, address of element located on index number 5 is: 750 + 5 * 2=760. Here 750 is the base address of an array, 5 is the index number of an element and 2 is the size of an element as array is declared of integer type. In the character array you need to consider size of an element 1 and in the case of long int or float size of each data-element will be 4.

1.4 ARRAY OPERATIONS

We can perform different types of operations on an array. Which includes Traversal, Insertion, Deletion, searching for a data-element, sorting an array and merging of arrays.

1.4.1 Traversing

Traversal is the process of accessing each data-element of an array. In the upcoming program we have declared an array and stored 10 data-elements in it. Using for loop if we print all 10 data-elements on the console screen is called traversal of an array.

```
#include<stdio.h>
void main()
{
  int arr[10]={2,19,11,47,34,45,28,78,84,92};
  int i;
  printf("\nArray contains: ");
  for(i=0; i<10; i++)
  {
    printf("%d\t",arr[i]);
  }
}
```
Output:

Array contains: 2   19   11   47   34   45   28   78   84   92

In the above program we have declared an integer array 'arr' having 10 integer values. Using 'i' loop variable and for loop, we have displayed each element of an array. Reading (accessing) all the elements and printing on the console or any other calculation purpose is called traversing the array.

1.4.2 Inserting value in the Array

Now, think of an array having some values are stored and other positions are empty (filled with 0). Now suppose, user want to insert a new data-element at position 4. In this case first we need to copy 4th element into some variable (temp). The number to be inserted, will be placed on 4th position. Finally, we copy the value of temp variable to variable n. The same process will continue till end of the array. Which means data-element on 4th position is shifted to 5th position, data-element on 5th position is shifted to 6th position and so on. Here, all data-elements from 4th position are shifted in the right direction.

```c
#include<stdio.h>
void main( )
{
  int x[10]={3, 6, 9, 12, 15, 18, 0, 0, 0, 0};
  int n, i, temp, position;
  printf("Enter Position where the new value to be Inserted:");
  scanf("%d", &position);
  printf("Enter New Element:");
  scanf("%d",&n);
  printf("Array Before Insertion:\n");
  for(i=0;i<10;i++)
                printf("%d\t", x[i]);
  for(i=0;i<10;i++)
        {
                if(i>=position -1)
                {
                        temp=x[i];
                        x[i]=n;
                        n=temp;
                }
        }
         printf("\nArray After Insertion:\n");
        for(i=0;i<10;i++)
                printf("%d\t", x[i]);
}
```

OUTPUT:

Enter Position where the new value to be Inserted: 4

Enter New Element:10

Array Before Insertion:

3   6   9   12   15   18   0   0   0   0

Array After Insertion:

3   6   9   10   12   15   18   0   0   0

1.4.3 Deletion in the Array

In the deletion of an element from the array, we need to shift all elements of array, from the element to deleted to last element are shifted in the left side. Therefore, it is reverse process than insertion (In the insertion, we have shifted elements towards the right side). The following program has to code, for deletion of an element in the array.

```c
#include<stdio.h>
void main()
{
        int x[10]={3, 6, 9, 12, 15, 18, 21, 0, 0, 0 };
        int  i, temp, position;
        printf("Enter Position:");
        scanf("%d", &position);
        printf("Elements in Array Before Deletion:\n");
        for(i=0;i<10;i++)
                printf("%d\t", x[i]);
        for(i=0;i<9;i++)
    {
         if(i>=position-1)
                {
                        x[i]=x[i+1];
                }
    }
    x[9]=0;
    printf("\nElements in Array After Deletion:\n");
     for(i=0;i<10;i++)
                printf("%d\t", x[i]);
}
```

OUTPUT:

Enter Position:3

Elements in Array Before Deletion:

3, 6, 9, 12, 15, 18, 21, 0, 0, 0

Elements in Array After Deletion:

3, 6, 12, 15, 18, 21, 0, 0, 0, 0

1.4.4 Searching in the Array

In the case of searching of an element from the array, there are two logics are there: [1] Linear search and [2] Binary search. In the Linear search, we need to compare search element with all elements of the array, starting from position 0. When we get the search element in the array, we need to print the position of that element in the array (i.e., index +1). In the case when search element is do not match with any element print suitable message. The draw back of the linear search is, it slow. Binary search is faster than Linear search but it can be implemented only on sorted array. The Linear search can be implemented as given below:

```
#include<stdio.h>
void main( )
{
        int x[10]={3, 6, 9, 12, 15, 18, 21, 24, 27, 30 };
        int  i, n;
        printf("Enter Search Element:");
        scanf("%d", &n);
        printf("Array:\n");
        for(i=0;i<10;i++)
                printf("%d\t", x[i]);
        for(i=0;i<10;i++)
        {
                if(x[i]==n)
                 {
                                printf("\nElement Found on: %d positions", i+1);
                                break;
                 }
        }
         if(i==10)
        {
                printf("\nSearch element not found:");
        }
}
```

OUTPUT:
Enter Search Element: 12
Array:
3,   6,   9,   12,   15,   18,   21,   24,   27,   30
Element Found on: 4 positions

1.4.5 Sorting an Array

Arranging data elements of an array in specific (either ascending or descending) is called sorting of an array. Many logics are available to sort an array. The program, which presented below is called selection sort. In this logic we are comparing first element of an array, with all other elements. If we get any smaller element than first element then we swap that smaller element with the first element of an array. The same process is repeated for second, third and all elements.

```
#include<stdio.h>
void main()
{
        int x[10]={25, 53, 28, 90, 30, 66, 31, 81, 48, 2 };
        int  i, j, temp;
        printf("Array Before Sorting:\n");
        for(i=0; i<10; i++)
                printf("%d\t", x[i]);
        for(i=0; i<10; i++)
        {
                for(j=i; j<10; j++)
                {
                        if (x[j]<x[i])
                        {
                                temp=x[i];
                                x[i]=x[j];
                                x[j]=temp;
                        }
                }
        }
        printf("\nArray After Sorting:\n");
        for(i=0; i<10; i++)
                printf("%d\t", x[i]);
}
```

OUTPUT:

Array Before Sorting:
25   53   28   90   30   66   31   81   48   2
Array After Sorting:
2   25   28   30   31   48   53   66   81   90

In the program we have used algorithm called selection sort. Other algorithms like bubble sort (where instead of I and j, we are comparing j and j+1 elements), insertion sort, quick sort, merge sort etc are there.

In the program we have sorted the array in the ascending order. If you wish to sort an array in the descending order then change the condition we have placed in the program from: *if (arr[j]<arr[i]) to if (arr[j]>arr[i]).*

1.5 TWO-DIMENSSIONAL ARRAY

In the examples discussed, we have declared array like: 'int arr[10];'. When we are declaring array like this, then array 'arr' has 10 rows only (no columns), we can consider that array is of one-dimensional (rows only). But if we declare array like: 'int arr[3][3];' then array 'arr' has three rows and in each row, three columns are there (that means total nine elements can be accommodated in the arr).

This type of array, which has rows and some columns in each row is called 2-dimensional array. To represent matrix, we need a 2-dimension array. 2-dimesional array can be represented as follows:

As shown in the figure, we have stored 1 to 9 in the array. In the array 1 can be accessible by arr[0][0], 2 can be accessible by arr[0][1], 5 can be accessible by arr[1][1] and 8 can be accessible by arr[2][1]. Consider the following program, which will do the addition of 2 matrices (2-D Arrays) of size 3*3.

1.5.1 Addition of two Matrices

The process of adding two matrices, is quite simple. First data-element of the first row, has to be added in the first data-element of the first row of second matrix, and placed it in the first data-element of the first row of third matrix. The process continues for all elements, which programmatically shown below:

```c
/* Program to Add to Matrices of 3*3 */
#include<stdio.h>
void main()
{
    int x[3][3], y[3][3],z[3][3];
    int i,j;

    printf("Enter Elements for First Matrix:\n");

    for(i=0;i<3;i++)
    {
            for(j=0;j<3;j++)
                {
                        printf("Enter value for X[%d][%d]:", i, j);
                scanf("%d", &x[i][j]);
                }
    }
     printf("Enter Elements for Second Matrix:\n");

    for(i=0;i<3;i++)
    {
                for(j=0;j<3;j++)
                {
                        printf("Enter value for Y[%d][%d]:", i, j);
                scanf("%d",&y[i][j]);
}
 }

for(i=0;i<3;i++)
    {
                for(j=0;j<3;j++)
            {
                z[i][j] = x[i][j] + y[i][j];
            }
    }
printf("Matrix X:\n");

 for(i=0;i<3;i++)
    {
            for(j=0;j<3;j++)
             {
                    printf("%d\t", x[i][j]);
            }
            printf("\n");
  }
```

```c
printf("Matrix Y:\n");
    for(i=0; i<3; i++)
    {
            for(j=0; j<3; j++)
                {
                printf("%d\t", y[i][j]);
                }
                printf("\n");
    }
     printf("Matrix Z:\n");
     for(i=0; i<3; i++)
     {
                for(j=0; j<3; j++)
                {
                        printf("%d\t", z[i][j]);
                }
                printf("\n");
    }
}
```

OUTPUT:

Matrix X:

21   22   23

24   25   26

27   28   29

Matrix Y:

11   12   13

14   15   16

17   18   19

Matrix Z:

32   34   38   40   42

36   44   46   48

Check Your Progress-4

1. To represent a matrix, we need to take _____ array.

[A] 1-Dimenssional   [B] 2-Dimenssional

[C] 3-Dimenssional   [D] None of the above

2.If we have declared int x[3][3]; then x is _____ array .

[A] 1-Dimenssional   [B] 2-Dimenssional

[C] 3-Dimenssional   [D] None of the above

3 Array int x[3][3] will occupies _____ memory and can store ____ elements.

[A] 18, 9   [B] 9, 9

[C] 12, 6   [D] 6, 6

1.5.2 Multiplication of two Matrices

Multiplication of matrices is a complex process. To do this all data-elements of first row matrix of first matrix, has to be multiplied with all data-elements of first column of the second matrix, and the sum of it will be placed a as first data-element of first row of the resultant matrix.

That means,

z[0][0] =x[0][0]*y[0][0] + x[0][1]*y[1][0] + x[0][2] * y[2][0]

z[0][1] =x[0][0]*y[0][1] + x[0][1]*y[1][1] + x[0][2] * y[2][1]

z[0][2] =x[0][0]*y[0][2] + x[0][1]*y[1][2] + x[0][2] * y[2][2]

z[1][0] =x[1][0]*y[0][0] + x[1][1]*y[1][0] + x[1][2] * y[2][0] and so on.

Programmatically, implementation of the multiplication of two matrices of size 3*3 is given below:

```
/* Program to Multiply to Matrices of 3*3 */
#include<stdio.h>
void main( )
{
    int x[3][3], y[3][3],z[3][3];
    int i,j,k;
    printf("Enter Elements for Matrix1:\n");
    for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
       printf("Enter value for X[%d][%d]:",i,j);
       scanf("%d",&x[i][j]);
    }
  }
printf("Enter Elements for Matrix2:\n");
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
       printf("Enter value for Y[%d][%d]:",i,j);
       scanf("%d",&y[i][j]);
       z[i][j]=0;
    }
  }
```

```c
    for(i=0;i<3;i++)
        {
        for(j=0;j<3;j++)
        {
            for(k=0;k<3;k++)
            {
                z[i][j]+=x[i][k]*y[k][j];
            }
        }
        }
    printf("Matrix X:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",x[i][j]);
        }
        printf("\n");
        }
    printf("Matrix Y:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",y[i][j]);
        }
        printf("\n");
    }
        printf("Matrix Z:\n");
        for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",z[i][j]);
        }
        printf("\n");
    }
}
```

OUTPUT

Matrix X:
1   2   3
4   5   6
7   8   9

Matrix Y:
11   12   13
14   15   16
17   18   19

Matrix Z:

```
90     96     102
216    231    246
342    366    390
```

1.5.3 Multi-dimensional arrays:

In Multidimensional array we defined same as single dimensional array; the size is given in separate pair of square brackets. Thus, a three-dimensional array will require three pairs of square brackets and so on

In general terms, a multidimensional array definition can be written as <storage class data type> array [exp1][exp2]…[exp n];

Where storage class refers to the storage class of the array, data type is what data is stored, Array name i the array and exp1, exp2… exp n are positive valued expressions that indicate the number of array elements associated with each subscript. The storage class is optional; the default values are automatic for arrays that are defined inside of a function and external for arrays defined outside of a function

For example,

float table [50][50];

char page [24][80];

static double records [100][66][255];

static double records[L][M][N];

The first line defines the table as a floating-point array having 5 rows and 5 columns (hence 5 x 5 = 25 elements) and the second line establishes the page as a character array with 24 rows and 80 columns (24 x 80 =1920 elements), the third array can be thought of as a set of double precision 100 tables, each having 66 lines and 255 columns (hence 100 x 66 x 255 = 1,683,000 elements).

The last definition is similar to the preceding definition except that the symbolic constant L, M, N defines the array size. Thus, the values assigned to these symbolic constants will determine the actual size of the array.

If a multidimensional array definition includes the assignment of initial values, the order is maintained in which the initial values are assigned to the array elements (remember only external and static arrays can be initialized).

The rule is that the last (right most) subscript increases most rapidly and the first (left most) subscript increases least rapidly. Thus, the elements of a two-dimensional array will be assigned by a row that is the element of the first row will be assigned, then the element of the second row and so on.

For example, consider the following two-dimensional array definition:

int values [3][4] = {1,2,3,4,5,6,7,8,9,10,11.13};

Note, that values can be thought of as a table having three rows and four columns (four elements per row.) Since the initial values are assigned by rows, which results into the initial assignment as follows:

value [0][0] = 1	value [0][1] = 2	value [0][2] = 3	value [0][3] = 4
value [1][0] = 5	value [1][2] = 7	value [1][3] = 8	value [1][1] = 6
value [2][0] = 9	value [2][1] = 10	value [2][2] = 11	value [2][3] =12

There are 3 rows and 4 columns, the row is from 0 to 2 and the column is from 0 to 3. This example can be written as:

int values [4][3]

{

    {1, 2, 3},

    {4, 5, 6},

    {7, 8, 9},

    {10, 11, 12}

 };

The natural order in which the initial values are assigned can be altered by forming groups of initial values enclosed in braces. The values within each innermost pair of braces will be assigned to those array elements whose last subscript changes most rapidly. In a two-dimensional array, for example, the value within the inner pair of braces will be assigned to the element of row, since the second subscript increases most rapidly. On the other hand, the number of values within each pair of braces cannot exceed the defined row size. Multi-dimensional arrays are processed in the same manner as one - dimensional arrays, an element-by-element basis.

However, some care is required when passing multidimensional arrays to a function. In particular, the formal argument declarations within a function definition must include explicit size specifications in all of the subscript positions except the first. These size specifications must be consistent with the corresponding size specifications in the calling program. The first subscript position may be written as an empty pair of square brackets as with a one-dimensional array.

An individual array element that is not assigned, its values will automatically be set to zero. This includes the remaining elements of an array in which certain elements have been assigned non zero values.

The array size need not be specified explicitly when initial values are included as a part of an array definition, with a numerical array, the size will automatically be set equal to the number of initial values included within the definition.

Check Your Progress-5

1. Array int y[3][4][5] can accommodate _____ elements.

[A] 24   [B] 60

[C] 120   [D] 12

2.How much memory space (bytes) is required for, float x[3][4][5].

[A] 60   [B] 120

[C] 24   [D] 240

1.6 LET US SUM UP

In this unit, we:

- Have studied about storing homogeneous elements in a single variable using arrays.

- Have discussed about the method of processing an array.

- Have discussed about handling of two-dimensional arrays.

- Have studied multi-dimensional arrays.

1.7 SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

4. [A] 0
5. [B] Homogeneous
6. [D] 9

Check Your Progress-2

4. [A] [ ] Square brackets
5. [D] 766
6. [B] 20 Bytes
7. [B] UB-LB+1

Check Your Progress-3

3. [B] Sorting
4. [A] Searching
5. [C] Insertion

Check Your Progress-4

3. [B] 2-Dimentional
4. [B] 2-Dimentional
5. [A] 18, 9

Check Your Progress-5

3. [B] 60
4. [D] 240

1.8   GLOSSARY

3. **Array** is a homogeneous (same type of) collection of data.
4. **Base Address** is a starting memory location of an array.
5. **Lower Bound** is an index number where array stores its first element. Usually in C-Programming it is 0.
6. **Upper bound** is an index number where last element of the array is stored. Usually, it is size -1.

1.9   Assignment

4. What is Array? How can we declare and initialized it?
5. Discussed the array operation in details.
6. Discuss, how can we represent a matrix in the C-Language?

1.10   Activity

• Write a program which takes 10 numbers from the user and store it in the array. Inspect each element of the array and make separate list of Even and Odd numbers.

1.11   Case Study

• Write a program to find transpose of given 3*3 matrix.

1.12   Further Reading

• "Let Us C" by Yashwant Kanetkar.
• "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
• "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.

UNIT 2   HANDLING STRINGS

Unit Structure

2.0 LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Learn What is string and how it stores in the memory?

- Know How to declare and initialize string variable?

- Use various IO function, to handle string.

- Know How to handle strings.

- Understand How to manipulate string using string functions.

2.1 INTRODUCTION

To communicate, we use sentences in our natural language. Similar to that in programming language we communicate with the computer system, by giving it instructions (statements). Instructions or sentences are formed words. These words, which are collection of different symbols (characters) are known as strings in the programming languages. In C-Language, a sequence of characters is known as string. Many programming languages like VB.NET, C#.NET, Java etc. have string datatype, but C-Language does not have any built-in datatype to represent strings. In C-Language, string can be stored in an array of type character. It means, in C-Programming language, strings are stored in character array. Each array element stores one character of the string and we know that array elements are stored in the consecutive memory locations.

In C-Language, we use character typed array to store each character of the string, and at the end of the string, one special character '\0' is placed. Special character '\0' is called NULL character, which indicates compiler to end of the string. For example, if we want to store, the name of the country in the string then, we need to declare character array 'country' as follows:

char country [ ] = {'I', 'N', 'D', 'I', 'A', '\0'};

Characters of a string are stored as a character in the separate memory location (see table given below), in contiguous manner, and the last character is a NULL character, which is special character '\0', which indicates end of the string.

Character Stored	I	N	D	I	A	\0
Memory Address	7869	7870	7871	7872	7873	7874

We can also specify size at the time of declaration of string. For example,

char university [5] = "BAOU";

This is same as:

char university [5] = {'B', 'A', 'O', 'U', '\0'};

2.2 UNDERSTANDING STRINGS

In the previous section of this chapter, we have discussed that, in the C-Programming language string is represented as character-based array and ends with special character '\0' which is called NULL character.

2.2.1 Declaration and Initialization of String variable

To declare string variable, we need to declare character array in C-Language and to initialize it, we may use assignment operator =, followed by group of characters (string value) encased in double quotation mark as shown below:

char state [ ] = "Gujarat";

In such type of declaration, Compiler of C-Language automatically append ('\0') called end of the string mark, as a last character of the string. It automatically calculates the size, and declare the array which can accommodate all characters of the string including ('\0') NULL character.

Now, consider the next C-Program, in which we have declared two string variable string1 and string2. We are storing string "BSC-IT" in both the variables. We are keeping the size of string1 array is 6 and string2 array to 7.

```
#include<stdio.h>
void main()
{
   char string1[6]={'B','S','C','-','I','T'};
   char string2[7]={'B','S','C','-','I','T'};
   printf("String1: %s",string1);
   printf("\nString2: %s",string2);
}
```
OUTPUT:
String1: BSC-IT*
String2: BSC-IT

In this program, string1 array having size 6 and the string we have stored in the is "BSC-IT", also having 6 characters. In this case, we do not have space for storing '\0' (NULL) character at the end of the string, as a result, when we print string1 variable, we are getting some extra character(s) which is called junk character(s) will be printed on the screen. In the case of string2 array, the size is 8 and number of characters, we have stored in it are 7. Here complier can have space in the array to place, end of the string i.e., '\0' (NULL) mark. So, when we are printing, string2 using printf() function, it will be printed properly without any extra (junk) characters.

2.2.2 Use of printf() and scanf() functions to print/scan strings

We know that the string can be printed on the console screen, using formatted IO function printf(). To print the string variable, we need to pass "%s" format string. Read the next program, in which we have printed message "HelloWorld" which stored in the message character type array by a printf() function.

```
#include<stdio.h>
void main()
{
   char message [] ="HelloWorld";
   printf("%s", message);
}
```
OUTPUT:

HelloWorld

In this program we have declared, character array message and initialized it with string value "HelloWorld". In the program when, we print the string variable using printf() function and with "%s" format string, we are getting proper output. Now, look at the next program, in which we have tried to store string "Hello How Are You?" (a string with some spaces).

#include<stdio.h>
void main( )
{
  char message [] = "Hello How Are You?";
  printf("%s", message);
}

OUTPUT:

Hello How Are You?

When you execute this program, you will get a string "Hello How Are You?", to be printed on the console screen. That means that printf() function can able to print space characters on the console screen. Now, consider the next program in which, rather taking static string we take, strings from the user, using scanf() function.

#include<stdio.h>
void main( )
{
  char string1[20], string2[20];
  printf("Enter First String:");
  scanf("%s", string1);
  printf("Enter Second String:");
  scanf("%s", string2);
  printf("String1: %s", string1);
  printf("\nString2: %s", string2);
}

OUTPUT:

Enter First String: Gujarat
Enter Second String: God is Great
String1: Gujarat
String2: God

The important thing we want to represent from the previous program is that, if we have two string variable and we accept the values for both variable from the user, and suppose user enters "Gujarat" in the first string and "God is Great" in another string the we will get output as "Gujarat" and "God". The reason behind this is first string do not have space character, while in the string "God is Great" we have used two spaced. Function scanf() do not accept the characters from space character. So in the string2 variable only "God" word is stored, which is printed by the printf() statement. Function gets() has to be used if you have string with some spaces in it.

Check Your Progress-2

1. Which functions are used to accept string?

[A] scanf()   [B] gets()

[C] Both A and B   [D] puts()

2.Which function accept string from the user, but do not allow user to have spaces in the string?

[A] scanf()   [B] printf()

[C] gets()   [D] puts()

2.2.3 Use of puts() and gets() functions:

From the previous program we have learn that, scanf() function can accept the string from the user with "%s" format string, but it is not be able to take space character. If we want to accept a string value from the user, in which space character might be there, we need to use gets() function. Function gets() is used to accept string from the user and it function properly even if that particular string has space characters.

```
#include<stdio.h>
void main()
{
    char str[20];
    printf("Enter String:");
    gets(str);
    printf("Your String is:\n");
    puts(str);
}
```
OUTPUT:

Enter String: Hello, How Are You?
Your String is:
Hello, How Are You?

As we know that the function gets() is unformatted function, specifically designed to accept string from the user. Because it is special function, can be used to accept only string from the user, we don't have to pass any kind of format string with this function (unformatted function). In the program, we have accept the string using gets() function, so to print the string value, we have used a function puts(). Function puts() is also an unformatted function (do not have to pass format string "%s"), specifically designed to print a string value on the console screen.

2.2.4 Use of EOF character:

In the previous section, we have learn that scanf() function can't takes those strings, which include spaces. Therefore, scanf() function is just suitable, to scan words only, and not the sentences. To scan entire sentence (multiple words, separated by spaces), we need to apply gets() function. But gets() function cannot accept New Line (Enter) character. Means, gets() is just suitable to take single line. To take a string with two or more line, we can't use gets() function. To solve this problem, we are accepting one by one character from user and storing it in the array of type character, using loop, this will also allow user to input spaces and New Line characters. When user enters Ctrl+Z and then Enter, the loop will stop, taking characters from the user. This special character Ctrl+Z is known and End of File in short EOF character. Consider the program given below, in which we have taken relatively large size of character array called string. With the help of while loop, we are accepting one by one character from the user and store it in the string variable. At the End user will eneter EOF character by pressing Ctrl+Z keys and then Enter.

```
#include<stdio.h>
void main()
{
    char string[250],ch;
    int i=0;
    printf("Enter Multiline String (press Cntrl+z at the End):\n");
    while(((ch=getchar())!=EOF))
    {
        string[i] = ch;
        i++;
    }
```

```
    str[i]=EOF;
    i=0;
    printf("\nThe Multiline String you have Entered is:\n");
    while((ch=str[i])!=EOF)
    {
       printf("%c",ch);
       i++;
    }
}
```

OUTPUT:

Enter Multiline String (press Cntrl+z at the End):

This first line of the string,

This is second line,

This third line of the string and now we will press ctrl+Z and then Enter

^Z

The Multiline String you have Entered is:

Enter Multiline String (press Cntrl+z at the End):

This first line of the string,

This is second line,

This third line of the string and now we will press ctrl+Z and then Enter

---

Check Your Progress-3

1. _____ function is used to accept a string having spaces.

[A] scanf()   [B] gets()

[C] Both A and B   [D] puts()

2. Ctrl+Z,is called _____ character.

[A] EOS   [B] EOE

[C] SOE   [D] EOF

3. _____ is used to accept string which can have spaces and new lines.

[A] gets()   [B] scanf()

[C] loop till EOF   [D] None of the above

4. EOF stands for _____.

[A] End of function   [B] Execution of function

[C] Enable open file   [D] End of file

2.3 DISPLAYING STRING IN DIFFERENT FORMATS

As you know that, function printf() is a formatted IO function. Formatted IO functions, format the data, based on the format string given by the user and represent the data in different formats. In the following table we have given some examples, which provides you the knowledge of how different format strings can be used with printf() function., Suppose if we have taken an array: *char string[15]="UNIVERSITY".*

Sr. No	printf() statement	Output
1	printf("%s", string);	UNIVERSITY
2	printf("%.5s", string);	UNIVE
3	printf("%.8s", string);	UNIVERSI
4	printf("%.15s", string);	UNIVERSITY
5	printf("%-10.6s", string);	UNIVER
6	printf("%15s", string);	UNIVERSITY

1. First statement use "%s" format string with printf() function. This will print entire string "UNIVERSITY" on the screen.

2. Second statement use "%.5s" format string with printf() function. In this statement we specify precision (the number of characters to be displayed) after the decimal point. As a result, it will only print first five letters of the string.

3. Third statement, with precision 8, will display first eight characters of the string on the console screen.

4. Forth statement, will print entire string on the screen as the precision is greater than number of characters in the screen.

5. Fifth statement use "%-10.6s" format string with printf() function. It means that string will occupies 10 characters on the console, .6 indicates 6 letters to be printed, and rest of the (4) characters will be spaces on the console, aligned left hand side because of – sign. Which means it will print UNIVER followed by 4 spaces. If you write another statement after it, called 'printf("Hi"); immediately after the above printf() statement, then you get the string called "UNIVER   Hi". There are 4 spaces are there between 'UNIVER' and 'Hi'.

6. In the last and sixth statement, format string "%15s" is used. Positive number indicated right alignment of the string. Now, string will be printed with the space of 15 character, number of letters in the string are 12, string is to be aligned at right, which print 3 spaces and then entire string "UNIVERSITY".

Check Your Progress-4

1. Output of statement: printf("%.9s", " UNIVERSITY"); will be _____.
[A] UNIVERSITY   [B] UNI
[C] UNIVER   [D] UNIVERSIT
2. Output of statement: printf("%-7.4s", "BAOU); will be _____.
[A] BAOU<space><space><space>   [B] AOU
[C] <space><space><space>BAOU   [D] BAOU
3. Output of statement: printf("%7.4s", "BAOU); will be _____..
[A] BAOU<space><space><space>   [B] BAO
[C] <space><space><space>BAOU   [D] BAOU

2.4 STANDARD FUNCTIONS FOR STRING HANDLING

Up to here, we have discussed how can we declare and initialize string variable, not only that we have learn, how to use different types of IO functions. Now, this is a good time to discuss how different types of built-in function for string can be used to manipulate the string variable. To use this built-in function for string, you need to include library 'string.h'. For an example the program given below will measure the length of the string and convert the string into the upper case.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char state_name[]="Gujarat";
    int len;
    len=strlen(state_name);
    strupr(state_name);
    printf("Length of the string is: %d",len);
    printf("\nUpper case string is: %s",state_name);
}
```

OUTPUT:

Length of the string is: 7

Upper case string is: GUJARAT

In the above program, we have used 2 string functions. First is strlen() which is used to find the length (number of characters) of string. Second string function used in the program is strupr(), which converts lower-case string into the upper-case string. Consider the following table, which has few more functions you can use for your program to handle strings.

Functions	Description
strlen()	Function is used to find length of the string.
strcpy()	Function is used to copy one string (character array) to another.
strncpy()	Function copy n characters from one string to another string.
strcmp()	Function is used to compare two strings, whether they are identical or not?
stricmp()	Function is used compare two strings by ignoring case (not case-sensitive comparison)
strncmp()	Function is used to compare first n letter of first string to another string.
strnicmp()	Function is used to compare first n letter of first string to another string without considering case-sensitivity.
strlwr()	Function converts all upper-case letters into lower-case.
strupr()	Function converts all lower-case letters into upper-case.
strcat()	Function is used to concatenate (join) two strings.
strrev()	Function is used to reverse the string.
strstr()	Determines the first occurrence of a given string in another string.

Remember, in order to use the functionality listed in the above table, we need to include "string.h" header file. More details and example of some of the important functions are explain below.

Many C compilers include built-in functions that allow strings to be copied, compared or concatenated. Some functions permit operations on individual characters within the strings variable. For example, they allow individual characters to be search within strings and so on. The following example demonstrate the use string functions of "string.h" file:

1. **strlen() :**

   The function is used to count number of characters present in a string. At the time of calling this function, we need to pass the base address of the character array in which we have stored the string. Function strlen() count number of characters by excluding '\0' (NULL) character.

   char msg[] = "Peacock"

   int l;

   l=strlen(msg);

   printf("length of given string is =%d", l);

   The output will be: length of given string is =7.

2. **strcat() :**

   The function is used to concatenate (join) target string into the source string.

   char target [] = "BAOU";

   char source [] = "University";

   strcat(target, source);

   printf("\nSource string is %s", source);

   printf("\nTarget string is %s", target);

   The final output is:
   Source string is University
   Target string is BAOUUniversity

3. *strcpy() :*

   This function copies the contents of one string to another. The base addresses of source and target strings are supplied to the function.

   char source [] = "BAOU"

   char destination [15];

strcpy(destination, source);

printf("Source string is : %s", source);

printf("target string is :%s", destination);

Now string destination has same string as target that is: BAOU.

4. strrev(string):

Function strrev() is used to reverse the string. For Example,

str = "Gujarat";
strrev(str);
printf("%s",str);
This will print - tarajuG

5. strlwr ():

The function strlwr() is used to converts all characters in the string from uppercase to lowercase.

strlwr(string);

For example:

strlwr("BSC-IT") converts to bsc-it

6. strcmp()

The function strcmp() is used to compares two strings to know whether they are the identical or different. The process of character wise comparison continues till a mismatch is occur or till the string ends. If two strings are similar, function will return 0 value, else it returns (1 or -1) depending upon which string is bigger, based on numeric (ASCII) difference between non- matching characters.

Function strcmp() takes addresses of two string and returns an integer value. If both strings are equal then it returns 0 otherwise it returns some other integer value.

#include<stdio.h>
void main()
{
  char str1[]= "BAOU";
  char str2[]= "University";
  int a,b,c,d;

a=strcmp(str1,"BAOU");

b=strcmp(str1,"baou");

   c=strcmp("baou", str1);

   d=strcmp(str1, str2);

   printf("Value A:%d, B is:%d, C is:%d and D is:%d",a,b,c,d);

}

OUTPUT:

Value A:0, B is: -1, C is:1 and D is: -1

In this example, str1 and "BAOU" are identical strings, therefore functions return 0 in the variable a. Because the ASCII value of B is 66 and b is 98 therefore str1 ("BAOU")< "baou" it returns -1 in the variable b. In the reverse case "baou" > str1("BAOU") so it returns 1. Similarly, while comparing "BAOU" and "University", BAOU < University (as the ASCII value of B from BAOU is less than ASCII value of character U of university, it returns -1 into the variable d.

Check Your Progress-5

1. Function strcmp("abcde","ABCDE"); will return _____.
[A] 1   [B] 32
[C] -1   [D] 0

2. Function strcmp("abc", "abc"); will return _____.
[A] 1   [B] 32
[C] -1   [D] 0

3. _____ function is used to convert the string into lower-case.
[A] strlower()   [B] lower()
[C] strlwr()   [D] None of the above

4. _____ function reverse the given string.
[A] reverse()   [B] stringreverse()
[C] strrev()   [D] stringrev()

2.5 TABLE OF STRINGS

We can declare two dimensional arrays of characters. For example, we could write:

char c_name [5][20] = {"India", "USA", "China", "Nepal", "Sri-Lanka"};

The size of the left index (5 in the example) determines the number of strings (number of countries) and the right index specifies the maximum length of each string (number of characters in the country name).

2.6 LET US SUM UP

In this unit, we:

- Have discussed about accepting strings from the user using gets() and scanf() functions.

- Have studied about declaring and initializing string variables.

- Have discussed about various string functions which are used to perform different types of operations on strings.

- Have discussed about EOF, and table of strings.

2.7 SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1
1. [C] characters
2. [D] 8 bits
3. [B] '\0'

Check Your Progress-2
1. [C] Both A and B
2. [A] scanf()

Check Your Progress-3
1. [B] gets()
2. [D] EOF
3. [C] loop till EOF
4. [D] End of File

2.8   GLOSSARY

1. **String** is a group of characters. In C-Language it is represented by array of type character.
2. **'\0'** is an identification mark of string end. It is also known as NULL character.
3. **Concatenation** is a process of appending one string to other string.

1.9   Assignment

1. What is String? How can we represent string in C-Language?
2. List and explain different string functions and explain any 3 of them.

1.10   Activity

- Write a program to reverse the string with using strrev() function.

1.11   Case Study

- Write a program to store name of your friends in the table of string and print them.

1.12   Further Reading

- "Let Us C" by Yashwant Kanetkar.
- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.

UNIT 3 FUNCTIONS

Unit Structure

3.0   LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Understand the necessity of using functions.

- Know the approaches of declaring and using functions.

- Write programs using user defined functions.

- Know about return values from the functions and their types.

- Understand the types of functions.

3.1   INTRODUCTION

In the previous chapter we have used many string functions. In this chapter, we will discuss about functions, it's and types. Functions are basically used to reduce the number of statements in a program. Whenever the same set of instructions are repeated within a program, then we can use functions. Functions have return types, which postulates the type of value returned.

3.2   NEED OF USER DEFINED FUNCTIONS

A function is a block of executable statements that has a name and it has a property that it is reusable i.e., it can be invoked from as many different points in a C-Program as required.

A Function groups number of program-statements into single unit and gives it a name. This unit can be invoked from other parts of a program too. A computer program can't handle all the tasks by itself. Instead, its requests other programs like entities - called functions in C - to get its tasks done. A function is a self-contained block of executable statements that perform a intelligible task of some kind.

A unique name has to be provided to the function in C-Programming. The function can be accessed from any location within a C Program. We pass data to the function which are called arguments stated when the function is called. And the function either returns some value to the point it was called from or sometime it doesn't return any value.

We can divide a long C program into small blocks which makes program to be more readable and manageable.

Why should we use Functions?

The most important reason to use functions is to aid in the conceptual organization of a program.

Another reason to use functions is, it can reduce program size. Any sequence of instructions that appears in a program more than once is a candidate for being made into a function. The function's code is stored in only one place in the memory, even though the function is executed many times in the progression of the program.

- Using functions, we can avoid rewriting the same code over and over. Suppose that there is a section of code in a program that calculates the area of a square. If, later in the program we want to calculate the area of a different square we won't like to write the same instructions again and again. Instead of that, we would prefer to go to a "section of code" that calculates area and then come back to the place from where we left off. This section of code is nothing but a function.

- Using functions, it becomes much easier to write a program and keep track of what they are doing. If the operation of a program can be divided into separate activities and each activity placed is in a different function, then each could be written and checked more or less autonomously. Separating the code in to modular functions also makes the program much easier to design and easy to understand.

Check Your Progress-1

1. Function is called_____.

[A] set of variables   [B] set of data types

[C] set of operators   [D] set of executable statements

2. Functions are readily available into some libraries are called _____.

[A] built-in functions   [B] user defined functions

[C] Both A and B   [D] None of the above

3.If we design any function then it is called _____.

[A] built-in functions   [B] user defined functions

[C] primitive functions   [D] None of the above

4. We need functions, because _____.

[A] to increase readability of program

[B] to reduce program complexity

[C] to save memory

[D] All of the above

3.3 A MULTIFUNCTION PROGRAM

Consider the given example:

```c
#include<stdio.h>
void function2()
{
  printf("\nYou are in Function2:");
}
void function1()
{
  printf("\nYou are in Function1:");
  function2();
  printf("\nYou are back to Function:");
}
void main()
{
  printf("\nYou are in Main:");
  function1();
  printf("\nYou are back to Function1:");
}
```

The output of the above program when executed would be

You are in Main:

You are in Function1:

You are in Function2:

You are back to Function:

You are back to Function1:

From the above program, the following conclusions can be drawn:
- Every C-Program must have at least one function, and if program has one function, then the name of that function must be void.

- If a C-Program have more than one functions, then each function in a program must have unique name, and at least one function must have name 'main ()' from where system will start its execution.

- Function name must start with alphabet or underscore ('_') and cannot start with digit (0 to 9). Function name must not be a keyword, and special symbols like ($, +, -, etc., and space) is not allowed. Only underscore is allowed in the function name.

- There is no limitation on the number of functions present in a C program.

- After execution function will transfer the control to its caller function.

3.4 BASIC STRUCTURE OF UDF:

The general structure of a function is:

Return-type function-name (parameter list)

{

    Statements;

}

       The return-type stipulates the type of data being returned by a function. A function can return any type of data (char, int, float etc.) except an array. The parameter list is separated by a comma, which has number of variables with same or different types to each. It is also possible that function does not have any parameter (parenthesis is null), and function does not return a value (return type is void). If programmer has not specified and return type, then by default function returns an integer value ('int' is a default return type).

Return-type function-name (type variable1, type variable2, ……………, type variableN)

Check Your Progress-2

1. User Defined Function always starts with _____.

[A] body of the function   [B] functions name

[C] return type   [D] argument list

2. Execution of a C-Program must start with _____ function.

[A] begin()   [B] main()

[C] start()   [D] None of the above

3. From the given identify the false statement.

[A] Every C-Program must have main() function.

[B] Function name must not be a keyword.

[C] Definition of a function starts with return type.

[D] In C-program more than one main() functions can exist.

3.5 FUNCTION TYPE AND RETURN VALUE

       Function can return one value, except those function which is define with return type 'void'. All other function, which have return type, they can return the value to the caller function by using 'return' statement.

Usually, the functions can be classified into three types. The first type of function is designed to do simple computational. This type functions are specifically designed to perform on their arguments passed to it and return a value based on the operation performed by it. For example, functions sqrt( ) and sin( ), which computes the square root and sine value of argument passes to it.

The second type of function manipulates data and returns a value that indicates the failure or success of that manipulation. For example, function fclose( ), which is used to close an opened file. If the operation is successful, it returns 0 otherwise returns EOF.

The third and the final type of function is that function, which has no explicit return value. For example, function exit( ) which terminates a program. All those functions which is not returning a value, should be declared with their return type as 'void'.

Points to remember:

- Every C-Program must have at least one 'main()' function. Function name can be duplicated so, no more than one 'main()' functions exists in a one program.

- Definition of the UDFs (User Defined Functions) must start with return type. Function can return only one value. If function doesn't return the value, then it's return type must be void. If no return type is given in the definition of the function, then by default 'int' return type is considered.

- Function name must start with either alphabets or underscore '_', and cannot start with digit. For Example, xyz123 is a valid function name but 123xyz is invalid name for function.

- Function name must not be a keyword. Keywords like (int, include, float, char, if, else, while etc.) are reserved words and cannot be used as a name of the function (identifier).

- Space or special symbols except underscore ('_'), are not allowed in the name of the function.

- Function always returns value or control, to the statement of function who has called it (caller function).

3.6   CALLING OF FUNCTION

Once the function is declared, then it can be called from other functions. We can not call any user defined function, which is not declared. In the following program, user defined function sum takes two arguments (integer numbers) and return sum of both the numbers (integer).

```
#include <stdio.h>
void sum(int, int);      //Function prototype declaration
void main( )
{
      int a, b;
      printf("Enter the values of A and B");
      scanf("%d%d",&a, &b);
            sum(a, b);       //Calling of function
}
void sum (int x, int y)  //Function definition
{
      int z=0;
       z= x + y;
      printf("sum is %d", z);
}
```

175

In the program given above, function main takes two values from the user and store both values in its variable a and b. Then main() function is calling, to the user defined function sum and pass the values of variable a and b as an argument. Variable a and b which are passed to sum function is called actual arguments and parameter values.

Function sum() is defined as, it returns integer value and takes two values from the caller function into variable x and y. In the definition of sum() function, variable x and y are called paraments or formal arguments of the sum function. Function sum() takes (copy) variable a and b of main() function into its x and y variable respectively. Function sum(), now has additional variable z, in with it computes the sum of variable x and y. Finally, function sum() prints the value of variable z on the console screen. Make sure, we have written the definition of the sum() function after main() function, therefore a prototype declaration for sum() function is essential before defining main() function.

Check Your Progress-4

1. Prototype declaration of UDF can be done _____.

[A] before main()   [B] after main()

[C] in another file   [D] None of the above

2. Arguments passed when we call a function is called _____ argument.

[A] formal   [B] normal

[C] actual   [D] dummy

3. Arguments passed at time of definition of function is called_____ argument.

[A] formal   [B] normal

[C] actual   [D] dummy

3.8   TYPES OF FUNCTION

The different categories of functions can be decided by the type of argument value and the value that a particular function will return. Given below are different categories of functions that are categorized on the same basis.

3.7.1 Function with No Argument and No Return Value

It is possible that the function does not returns any value and also it is possible that the function does not accept any value from the caller function. Consider the following program in which 'SayHello()' function is not returning any value as well as it does not take any value from the main() function.

```
#include<stdio.h>
void sayhello();  //Prototype Declaration

void main( )
{
        sayhello(); //Function call
}

void sayhello() //Function defination
{
        printf("Hello, B.Sc. IT Students\n");
}
```

In the above program, we have defined a function called sayhello(). The function is printing a message for the students that is "Hello, B. Sc. IT Students". We have already discussed that program starts it's execution from the main() function. When system begins, execution of main() function, main() function is calling to the function sayhello(). When the main() function is calling to another function, then control will transfer to that particular function, and system will start execution of that function. As a result, we get the message "Hello, B. Sc. IT Students" on the console.

Now, think what happed if we put function calling line in the main() function inside the loop? (just try it) In that case, suppose if loop runs for five times, then main() function will invoke to sayHello() function for five times, and message will be printed for five times.

In this example, our user defined function sayhello(), is not returning any value to its caller function main(). Therefore, the return type of the function is 'void'. In the same way, at the time of calling to sayhello() function, main() function is not passing any actual arguments to the function (parenthesis are empty). Here user defined function, sayhello() is not returning any value and it doesn't have any paraments.

3.7.2 Argument but No Return Value

Now, we will discuss another example which takes arguments but does not return any value.

```
void sum (int, int);
void main( )
{
        int x=5, y=6;
        sum (x, y);
}
void sum (int a, int b)
{
        int tmp=0;
        tmp= x+y;
```

printf("Sum is: %d\n", tmp);

}

In the above program, function sum is a type of function, with no return type (void) and but it has two paraments a and b. Therefore, with sum () function, void data type is specified and it is accepting two integer types of arguments, so we can say that sum( ) is an example of, function with no return type and with arguments.

3.7.3 Arguments with Return Value

The example explained below, which demonstrates the use of function with arguments and with return values.

int sum(int, int);
void main( )
{
*   int x=5, y=7,z=0;*
*   z=sum(x, y);*
*   printf("%d", z);*
}

int sum(int a, int b)
{
*   int tmp=0;*
*   tmp=a+b;*
*   return tmp;*
}

In the above program, the sum function is returning an integer value, that is, tmp.

That is why an integer data type has been specified with it.

3.7.4 No Arguments with Return Value

This is rarely used category. Very smaller number of examples are there of this category. Consider an example, where main function wants to compute the area of circle. Function main() takes the value of radius from the user. Now, to compute the area main() function, is calling a function called pi() and that function is returning a value of PI that is 3.14. Here main() function, doesn't pass any actual argument to the function pi(), but pi() function is returning 3.14 to the main() function. Therefore, this type of function is fall under the category. that is No argument, with return value.

```
#include<stdio.h>
float pi();
void main()
{
  float r, area;
  printf("Enter Radius:");
  scanf("%f", &r);
  area=pi()*r*r;
  printf("Area is: %.2f", area);
}
float pi()
{
    return 3.14;
}
```

Check Your Progress-5

1. Function: void sum (int, int); is of type _____.
[A] No argument with Return [B] Argument with Return
[C] No argument, No Return. [D] Argument with No Return
2. From the given, identify "No argument and No return" type of function.
[A] void printhello(); [B] void sum(int, int);
[C] int sum (int, int); [D] float pivalue();

3.8 LET US SUM UP

In this unit, we:

- In this chapter, we have learnt what is UDF? And what is the basic structure of User Defined Function.

- We have all discussed, Why UDFs are important?

- We have discussed the basic structure of UDF.

- We have seen different types of UDFs.

- Have studied about How to write our own UDFs.

3.9 SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1. [D] set of executable statements
2. [A] built-in functions
3. [B] User Defined Functions (UDFs)
4. [D] All of the above

Check Your Progress-2

1. [C] Return type
2. [B] main()
3. [D] In C-program more than one main() functions can exist.

Check Your Progress-3

1. [C] int
2. [A] void
3. [C] Void is the default return type of the function.

Check Your Progress-4

1. [A] before main()
2. [C] actual
3. [A] formal

Check Your Progress-5

1. [D] Argument with No Return
2. [D] void printhello();

3.10 GLOSSARY

1. **Function** is a set (group) of executable statements.
2. **Return type** is a data type of the value return by a function.
3. **Formal arguments** are list of parameters passed at the time of defining a function.
4. **UDF** is a User Defined Function.

3.11 Assignment

1. What is UDF? List and explain different types of UDFs.
2. Discussed the structure of UDF.
3. What is prototype declaration?
4. Discuss, actual and formal parameters.

3.12 Activity

- Write a program, in which main function has 2 variables locally declared and initialized in the main() function itself. Design a function swap() which will try to swap (exchange) of both variables.

3.13 Case Study

- Write a program in which user defined function isprime(), takes an integer number and return 1 if the number passed to it is a Prime number, else retrun 0 if it is not a Prime number.

3.14 Further Reading

- "Let Us C" by Yashwant Kanetkar.
- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.

UNIT 4   MORE ABOUT FUNCTIONS

Structure

4.0   OBJECTIVES

After working through this unit, you should be able to:

- Know, how to create non-integer functions?

- Know, the scope and lifetime of variables.

- Understand, How the recursion works.

4.1   INTRODUCTION

In this unit, we will do discussion on handling of non-integer functions, nesting of functions and the recursion process.

Recursing is the process, where function calls itself. In future, there are many programs you need to study in which recursion is used. Let us start out discussion on creating non-integer functions.

4.2   CREATING NON-INTEGER FUNCTIONS

In the previous chapter, in most example we have seen, function return an integer value. As we have mentioned that function can return any type of data. It can return character or float value as such. See the following program, which will guide you how can we design a function which can return non-integer value.

```
float sum (float, float);
float average (int, int, int);
void main( )
{
      float x=5.3, y=7.9, s, avg;
      int a=5, b=6, c=8;
      s=sum(x, y);
      avg=average (a, b, c);
      printf("\n Sum is: %.2f and Average is: %.2f", s, avg);
}
float sum(float f1, float f2)
{
      return (f1 + f2);
}
float average (int n1, int n2, int n2)
{
      float ans = (n1+n2+n3)/3.0;
      return ans;
}
```

When passing the value to the function, and returning the result, you need to be careful. If there is a mismatch in the data and datatype in passing the arguments and returning the result, will surprise you by giving unexpected result.

4.3   NESTING OF FUNCTIONS

In C-Programming language, any function can call to any other function. For example, consider the following program. As we know, program start its execution from main() function. Function main() calls function1() and further function1() calls to another function called function2(). This is called nesting of the user defined function.

```c
#include<stdio.h>
void function2()
{
    printf("\nYou are in Function2:");
}
void function1()
{
    printf("\nYou are in Function1:");
    function2();
    printf("\nYou are back to Function:");
}
void main()
{
    printf("\nYou are in Main:");
    function1();
    printf("\nYou are back to Function1:");
}
```

In the program, given above system will start its execution with main() function. System understand that once three lines written in the main() function is executed then its work is over. System, starts with the first line and it prints "You are in Main" on the screen. When system execute second line at that time, it will come to know that the second line is a function call and it need to execute all the lines written in the function1() to complete second statement of the main() function. System will transfer its control to function1() and first line of function1() will be executed which will print the message "You are in Function1" on the screen. When system see the second instruction of the function1() then it come to know that it is function call to function2(). System need to execute function2() to complete second line of function1() and control is transferred to function2(). This function has only one line and system will print message that the "You are in function2", after completion of this line, control will get back to that line from where function2() is called. Now, function1() resumes to execute and its two lines are executed.

System now executes the third line of function1, and it will print "You are back to Function1", once all lines of function1 are executed then control will transferred back from where function1 is called. Yes it is main() function itself. Main() function now resumes its execution and first two line of it is already executed. System will execute third line of the main() function and it will print" You are in main". After execution of all three statements of the main() function, execution of a program gets completed.

Check Your Progress-1

1. User define function cannot return _____ type of data.

[A] int   [B] float

[C] char   [D] All

2. If one function call second, and second function call third, then it an example of _____.

[A] nesting of functions   [B] recursion

[C] multiple functions   [D] All of the above

3. After execution function will return the value or control to _____.

[A] self-function   [B] caller function

[C] main function   [D] system

4.4   RECURSION

Recursion is a process in which a function calls itself repeatedly, until some specified condition met. Suppose if we wish to calculate the factorial of a positive integer number. We would normally express this problem as n! = 1 x 2 x 3 x 4 … x n where n is the stated positive integer. The same problem can be represented in different manner. For example: 5! = 5 * 4!, 4! = 4 * 3! and so on. Here, we can give condition that when function see, n < 1, it stops calling itself and return the result 1 to its caller (itself).

When a recursive program is executed the recursive function, calls are not executed immediately. Rather, they are placed in a stack until the condition that terminates the recursion, is met. The function calls are then executed in a reverse order, as they are popped off the stack.

If a recursive function contains local variables, a different set of local variables will be created (multiple copies) during each call. The name of those variables will always be the same, as declared within the function. However, the variables will represent a different set of values, each time the function is executed. Each set of values will be stored on the stack, so that they will be available to the recursive process.

Consider the following program, which compute, factorial of given number using recursion:

```
#include<stdio.h>
int fact(int);
void main()
{
    int ans, n=5;
    ans =fact(n);
    printf("Factorial is: %d", ans);
}
int fact(int num)
{
  if (num==1)
      return 1;
   else
      return (num * fact(num-1));
}
```

OUTPUT:
Factorial is: 120

In this program, main() function is calling function fact(5), and passed 5 to it and wait to respond it. Function fact(5) takes value 5 into num variable and checks it is 1 or not. The value passed is 5, not 1. So, it will execute the statement 'return (num * fact(num-1)'. Now, to execute this line system needs value of num and value of fact(num-1). System knows that the value of num is 5, but system do not know the value of fact(num-1). Here, system will create the another instance of function fact(4) and passed number 4 to it (as num-1=5-1=4). This process continues and multiple instances of fact function that is fact(5), fact(4), fact(3), fact(2) and fact(1) will be created into the memory.

When fact(1) is responding to fact(2) function by returning value 1 (as num==1 in function fact(1)), fact(2) function will immediately compute 2*1 and return 2 to the fact(3). Once fact(3) will know fact(num-1) is 2, it will return num * fact (num-1) then is 3*2 =6 to fact(4) function. The process will continue and finally fact(5) will respond to main() function by returning 120. Main() function then print the factorial of 5 that is 120 on the console screen.

In this example, fact() function is calling itself (that is fact(5) is calling fact(4), fact(4) is calling to fact(3) and so on), it is called the example of recursion. In recursion multiple instances of the same function, are created in the memory and hence, it takes more memory. The benefit if recursion is, it faster than iterative process (loops). Recursion is complex logic to understand, but can be used to solve complex logic with few numbers of line of code.

4.5   FUNCTIONS WITH STRINGS

We know that the string must ends with special character called '\0' (NULL). Therefore, when we pass the array to function, we don't have to pass the length of the string. If the array is declared withing the function is called locally declared array, and be passed to any function by pointer. We will learn about pointers in great details in the block-4 of this course. Here we will focus on how can we access array declared in a one function into another function. Consider the following program in which we have declared array, and stores name of the user in the lowercase. We will call a function upper() which will change all lowercase letter into uppercase letters.

```
#include<stdio.h>
void upper(char *);
void main()
{
    char n[10];
    printf("Enter name in Lowercase:");
    scanf("%s",n);
    upper(n);
    printf("Your name is: %s", n);
}
void upper(char *name)
{
    int i=0;
    while(name[i]!='\0')
    {    name[i]=name[i] -32;
        i++;
    }
}
```

4.6   STORAGE CLASSES

When we declare any variable in the C-Programming language, then that variable belongs to following storage classes:

Automatic variables

Automatic variables are declared within a function in which they are to be operated. They are formed when the function is invoked and destroyed automatically after execution of that function, hence it named automatic. Automatic variables are basically private to the function in which they are declared. Because of any other function cannot use the variable declared in a function it is also called local variable or internal variable of the function.

When we declare any variable within a function without specifying any storage class then by default it will be of type automatic variable.

Because automatic variables can access or modify by a function in which we have declared it and another function cannot have access to this variable, automatic variables are protected by accidental change. Two functions can have variables with same name in different functions. But make sure both are two separate varaibles. There are two consequences of the scope and longevity of auto variables. First any variable local to main will normally live throughout the whole program, although, it is active only in main. Secondly, during recursion, the nested variables are unique auto variables, a situation similar to function, nested auto variable with identical names.

Automatic variables can also be defined within a set of braces known as "blocks" they are meaningful only inside the block where they are defined.

```
#include<stdio.h>
void main()
{
   auto int x; // Variable x is Automatic variable
   int y; // Variable y is also Automatic varaible
}
```

In the above program, variables x, and y both are of type automatic variables.

External variables

The external storage class defines that the variable has been declared at another place (not in the program, may be in another file). These variables are usually declared before defining function main(). Keyword 'extern' (optional) is used to denote external variable.

For example, create a file myfile.c and write following code:

#include<stdio.h>
int x=7;

Create another file, called test.c in the same directory of myfile.c (otherwise you have to specify the path) and write following code:

#include<stdio.h>
#include "myfile.c"
void main()
{
*   extern int x;*
*   int i;*
*   for(i=1; i<=x;i++)*
*   printf("\nHello");*
}

Compile both the files and run test program. You we get Hello is printed 7 times. The questions is: why 7 times? In the test.c we start running loop from i=1 to x. You may notice that in the entire program of test.c we haven't initialized x=7. So how it takes value to 7 for variable x? The answer is: the value of variable x is taken from the myfile.c file.

Static variables:

The static variables can be either internal or external type depending upon where it is declared. If it is declared outside of function then it will be a static global variable, and if it is declared inside the function then it will be a static local variable. Consider the following example and see the output.

#include<stdio.h>
void main()
{
*   int i;*
*   for(i=1;i<=7;i++)*
*   print();*
}
void print()
{
*   int static x=1;*
*   printf("\n%d", x);*
*   x++;*
}

If you run the above program, you will get output 1, 2, 3, 4, 5, 6, 7 on the screen. Now remove the static word from the print function, for variable x, then compile and run the program again note down the output. You will get 1,1,1,1,1,1,1.

Register Variable

Usually when we declare a variable, it is formed in the main memory (Random Access Memory). When we declare a variable of type register, then it will be formed in the special memory called register. Register is a fastest memory, which is resided in the CPU itself, and hence it will speed up the process. To declare this type of variable, keyword 'register' is used to declare this type of variable. For Example,

register int x;

Check Your Progress-4

1. In the main() function, if we have declared a variable 'int x;', then x will be of type _____.

[A] auto   [B] register

[C] extern   [D] global

2. to access the value of a variable declared in another file, we need to declare it as _____.

[A] auto   [B] register

[C] extern   [D] global

3. Which variables are not declared in the RAM or main memory?

[A] auto   [B] register

[C] extern   [D] global

4.7   ANSI C FUNCTIONS

Some of the commonly used ANSI C functions are:

1. isalnum()- To check whether a character passed to it is alphanumeric (a-z, A-Z, 0-9) or not.

2. isalpha()- To check whether a character passed to it is an alphabet (a-z,A-Z) or not.

3. isdigit()- To check whether a character passed to it is a digit (0-9) or not.

4. islower()- To check whether a character is a lower case letter or not.

5. isupper()- To check whether a character is an uppercase letter or not.

6. isspace()- To check whether a character is a whitespace or not.

7. toupper()- To converts a lowercase character to an uppercase.

8. tolower()- To converts an uppercase character to a lowercase.

Check Your Progress-5

1. _____ function is used, to check whether the character is from 0 to 9.

[A] isnumber()   [B] isalnum()

[C] isdigit()   [D] isapha()

2. ____ function is used to check, weather a character is alphabet or not .

[A] isnumber()   [B] isalnum()

[C] isdigit()   [D] isapha()

4.8 LET US SUM UP

In this unit, we:

- Have learn how to use non-integer numbers with function.

- Have studied nested functions.

- Have understand process of recursion.

- Have studied about the different type of storage classes.

- Have studied about some of ANSI C Functions.

4.9 Suggested Answers for Check Your Progress

Check your progress 1

1. [D] All
2. [A] nesting of function
3. [B] caller function

Check your progress 2

1. [D] stack
2. [B] recursion
3. [B] Recursion is slower process than iterative process

Check Your Progress-3

1. [A] pointer
2. [C] Both A and B

4.10 GLOSSARY

1. **Local variable** is that which is declared inside the function. Its scope is limited to that function only, and we cannot access that variable outside of that function.
2. **Global variable** is that which is declared outside of any function. It can be accessible throughout the entire program. In any function of the program, global variable is accessible.
3. **Pointer** is a special type of variable which holds the address (reference) of some another variable.
4. **Register** is a small sized, fastest memory resided in the CPU itself.

4.11   Assignment

1. Discuss storage classes in details.
2. Discussed the difference between local and global variable.
3. How can we pass array to function? Explain it with an example.

4.12   Activity

- Write program to demonstrate extern variable, register variable and static variable. Write your comments for each type of variable.

3.13 Case Study

- Write a program to have a function called findsubstr(str1, str2). The function will check weather the str2 is a substring of str1. If yes it will return position of str2 in str1, else return -1. Use the following logic to implement it.

```
int result = -1; //
boolean found = false;
for(int i=0; text[i] != '\0' && !found ; i++)
{
Boolean matchsofar = true;
for(int j=0; pattern[j] != '\0' && matchsofar; j++)
    if(text[i+j] != pattern[j])
        matchsofar = false;
if(matchsofar)
{
found = true;
result = i;
}
}
return result;
```

3.14 Further Reading

- "Let Us C" by Yashwant Kanetkar.
- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.

Reference Books

1. The Art of C, H. Schildt

2. Born to Code in C, H. Schildt

3. C Programming, Ed. 2, Kerninghan and Ritchie

4. C Programming with Problem Solving, Jacqueline A Jones, Keith Harrow

5. C Programming, Balagurusamy

6. Let us C, YashwantKanetkar

7. Programming in C, S. Kochan

8. Programming in ANSI C, Agarwal

9. Turbo C/C++ - The Complete Reference, H. Schildt

Block Activities

Activity 1

• Write a program using functions to accept an array of strings counts the number of vowel characters in each string and display the result.

Activity 2

• Write a program to create two matrices and find their product.

Activity 3

• Write a program using functions to concatenate two strings.

Activity 4

1. Write a program to search a substring in a string.

Block Summary

An Array is a collection of same type of elements under the same variable identifier referenced by index number. Arrays are widely used within programming for different purposes such as sorting, searching and etc. Arrays allow you to store a group of data of a single type. There are two types arrays single dimension array and multi-dimension array. Each of these array types can be of either static array or dynamic array. Static arrays have their sizes declared from the start and the size cannot be changed after declaration .

Dynamic arrays that allow you to dynamically change their size at runtime, but they require more advanced techniques such as pointers and memory allocation. Arrays are defined same as variables. Each array name must be followed by size i.e. how many numbers of elements are stored in an array. The size is enclosed in square brackets which is an integer. Single operations on entire arrays is not permitted in C, thus if a and b are similar arrays, the operations are carried out element by element. This is usually done within a loop where each pass of loop will be equal to the number of elements to be passed from and in array.

In Multidimensional array we defined same as single dimensional array; the size is given in separate pair of square brackets. Storage class refers to the storage class of the array, what data is stored. Array name is the array and exp1, exp2… exp n are positive valued expressions that indicate the number of array elements associated with each subscript. The storage class is optional, the default values are automatic for arrays that are defined inside of a function and external for arrays defined outside of a function. The gets() and puts() functions is used to transfer strings between the computer and the standard input/output devices. Each function accepts a single argument. To read in an entire line from the keyboard, or from any other stream, use getline().

You can put two strings together by using concatenation operator, that is, '+' operator.

strlen() This function counts a number of characters present in a string.

strcat() This function concatenates the source string at the end of target sting.

strrev() This function reverse the given string.

ttrupr() This function converts all characters of a string to uppercase.

strlwr () This function converts all characters in a string from uppercase to lowercase.

strcmp() This function compares two strings to find out whether they are same or different.

Function groups a number of program statements into a unit and gives it a name. This unit can be invoked from other parts of a program. A function is a self-contained block of statements that perform a coherent task of some kind. The name of the function is unique in a C Program and is Global. The function can be accessed from any location within a C Program. We pass information to the function called arguments specified when the function is called. And the function either returns some value to the point it was called from or returns nothing using functions avoids rewriting the same code over and over. Using functions, it becomes easier to write programs and keep track of what they are doing. The different category of functions can be decided by the type of argument value and the value that a particular function will return. C functions can be of no argument and no return value type. Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. Automatic variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically. Variables that are both alive and active throughout the entire program are known as external variables. They are also known as global variables.

Once a variable has been declared as global, any function can use it and change its value. ANSI C Functions are:

isalnum()- Checks whether a character is alphanumeric or not isalpha()- Checks whether the

given character is alphabet or not isdigit()- Checks whether a character is a digit or not.

islower()- Checks whether a character is a lower case letter or not. isupper()- Checks whether a

character is an uppercase letter or not. isspace()- Checks whether a character is whitespace or

not.

toupper()- Converts a lowercase character to uppercase.

tolower()- Converts an uppercase character to lowercase

Block Assignment

Short Answer Questions

1.   Explain different types of ANSI C Functions?
2.   What is an array? Explain how can we declare and initialize it?
3.   What is recursion, explain with example?
4.   Explain local and global variables?

Long Answer Questions

1.   Explain 2-dimenssional arrays.
2.   Explain string functions with examples?
3.   Explain storage classes in details.

BLOCK 4: STRUCTURES, POINTERS AND FILE HANDLING

Block Introduction

After discussing about Arrays and Functions in previous blocks, we will now be explaining structure and union which are user defined data types.

A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together like a record.

A structure can be defined as a new named type, thus extending the built-in data types. It can use other structures, arrays or pointers as some of its members, though this can get complicated unless you are careful.

A union is an object that can hold any one of a set of named members. The members of the named set can be of any data type. Members are overlaid in storage. The storage allocated for a union is the storage required for the largest member of the union, plus any padding required for the union to end at a natural boundary of its strictest member.

The method of accessing addresses of variables using pointers is also well explained.

We frequently use files for storing information which can be processed by our programs. In order to store information permanently and retrieve it we need to use files.

Files are not only used for data. Our programs are also stored in files. The file management techniques along with error handling operations are well explained in this block which will definitely help the learners to understand and develop programs based on file operations.

At the end of the block, in 4th unit, some solved programs are given which are based on the concepts discussed in the earlier units, will help the learners to understand those concepts practically.

The 4th unit contains some solved programs based on the statements/ concepts explained in the first 3 units. This will help the learners to understand those concepts in details as the problems are practically solved.

Block Objective

Main objective of designing this Block is to teach, what is structure? How can we create heterogenous collection of data using structure? After learning Unit-1 student will able to make their own user defined data type using structure and union.

Unit-2 is intended to teach what is pointer? And how can we use pointer to return multiple value from a function. How can we change a local variable of one function into another function using pointer?

To teach, how can we store data into secondary memory using files, we have designed a separate unit-4. Behind this unit our objective is to aware student about handling of text and binary files.

Our final unit, that is Unit-4 have some sample programs, which will give sufficient coding practice to the student, and students will be able to make programs of arrays, strings, functions, pointers, structures and files.

Block Structure

BLOCK 4: STRUCTURES, POINTERS AND FILE HANDLING

UNIT1   STRUCTURES AND UNIONS

Objectives, Introduction, Structures, Unions, Let Us Sum Up

UNIT 2   POINTERS

Objectives, Introduction, Understanding Pointers, Pointer Expressions, Pointers and Arrays, Pointers and Character Strings, Pointers and Functions, Pointers and Structures, Points on Pointers, Let Us Sum Up

UNIT 3   FUNCTIONS

Objectives, Introduction, Management of Files, Input/Output Operations on Files, Error Handling during I/O Operations, Let Us Sum Up

UNIT 4   SOLVED PROGRAMS-III

UNIT 1 STRUCTURES AND UNIONS

Unit Structure

1.0   LEARNING OBJECTIVES

In this unit, we will learn about how can we create our own data types by using structures and unions.

After working through this unit, you should be able to:

- Learn about defining user defined data type

- Understand about Structure declaration and initialisation

- Know about passing variables of structure in the functions

- Gain knowledge about declaration of union and how it differs from structure

1.1   INTRODUCTION

Up to here, we have discussed many different types of C-Programs and in that we have declared many variables of different data types. To declare the variable, we have learnt different data types. These data types (char, int, float, double etc.,) are either built-in datatype or primitive data type. Have you thought about how can we create our own data type? Can we? The answer is yes. But to create our own data types we need to understand structures and unions.

In this chapter, we will be discussed about how can we declare structure to create our own (user-defined) data types, how can create its variable and how can we use those variables into our C-Program.

1.2   STRUCTURES

Structures in C are defined as collection of a sequence of named elements of different types. Member elements are similar to the fields of a record and structure itself behave like a record in the database table. The member elements of a structure are stored in consecutive memory locations, but for space efficiency, the compiler can insert separation byte between or after members. Note that compiler never place a pad byte before the first member. The size of a structure variable is equal to the sum of the sizes of its member elements and the size of the padding bytes.

1.2.1 Structure Initialization

Structure can be defined as group of different types of member elements stored under common name. The main difference between array and structure is that array has same type of data (homogeneous), while structure has different types of data members (heterogeneous). A structure can contain basic data types as well as structured data types like arrays and other structures. Each variable within a structure is called a

data member element of the structure.

struct <structure_name>

{

datatype member1;

datatype member2;

-

-

} instance;

For Example,

struct student

{

     int roll_no;
     char stu_name[10];
     float percent;

};

struct student s1;

In above syntax, the struct is a keyword, which is used to declare structures. The 'struct' keyword identifies the beginning of a structure definition. It is followed by a structure name (identifier name). After structure name, we need to declare different member elements, enclosed in braces. If you define the structure without its variable, then it is just a template that can be used later in a program to declare structure variable. Here, s1 and s2 are student kind of variables which has three elements such as roll_no, stu_name and percentage.

struct student
{
  int roll_no;
  char stu_name[5];
  float percent;
} s1, s2;

These statements define the structure type student and declare two structures variable s1 and s2 of type student. s1 and s2 are each instance or variables of student data type. Each structure variable contains three members roll_no, stu_name and percentage.

We can also assign or initialization of member elements during the declaration of the structure. To initialize member elements of the structure, their values must be given in { }. The values must match with the datatype of the structure members.

struct student

{
*   int roll_no;*
*   char stu_name[15];*
*   float percentage;*
} s1= {51, "Mohan", 80.7};

OR

struct student s1={51,"Monah",80.7 } ;

To access the data from the member element of the structure, we need to mention name of the structure, then dot operator (.) and then name of the member element of the structure.

Syntax:

Structure_variable .Member_Element

For example

s1.roll_no=201;
printf("%d %s %f" , s1.roll_no, s1.stu_name, s1.percentage);

1.2.2 Size of Structure

The total size of a structure variable can be calculated by adding the size of all the member elements used to create the structure. A structure declaration does not reserve any memory space. It simply describes template. Memory is allocated only when variables or instance of a structure is created.

For example,

struct employee
{
      int emp_no;
      float salary;
      char name[5];
};

       Then the size of the above structure can be calculated by adding the size of each data type. As the size of emp_no. variable is 2 bytes, float is 4 bytes and name is 5 bytes. So, the total size becomes 11 bytes.

Check Your Progress-2

1. Compute memory space needed to declare one variable of following structure.

struct student
{   *int roll_no;*
      char name[10];
      float percentage;
};

[A] 7 Bytes   [B] 16 Bytes

[C] 3 Bytes   [D] 17 Bytes

2.Memory space required by a variable of structure type = _____.

[A] sum of memory space required for each data member of a structure.

[B] memory space needed to accommodate largest data member of structure.

[C] memory space needed to accommodate smallest data member of structure.

[D] None of the above.

1.2.3 Comparison of Structure Variables

       Structures, in C-Language supports functions like copying data members, assignment and passing reference to function etc. But the relational operators to compare the values of two variable of type structures is not allowed. Note that, you cannot compare the structures using the standard comparison operators (= =,>,<etc.);

1.2.4 Arrays within structures

       An array can be a member of structure. For example, consider the given structure:

struct emp
{
      int nums[5][5];   *// nums is an array of type integer with size 5 x 5.*
      float x;
} y;

Now, in order to refer element of 3rd row and 5th column you should write the given statement:

y. nums[2][4]

As the numbering starts from 0, so 3rd element can be referred by using index number 2 and 5th element can be referred by using index number 4.

1.2.5 Arrays of structures

If we need many variables then normally, we are taking array. For example 'int x[10];'. This statement will declare array 'x' which can store 10 integer numbers in it. Here, x is an array of primitive data type (int). If we want to create array of structure (user defined data type), then it is also possible. The syntax is as follows:

struct student
{
        int rollno;
        char name[10];
        float height;

};
struct student x[5];

Now, to understand the array of the structure, consider the structure called student, and rollno, name and height are the member elements of it. Suppose if, we want to store the data of 5 students, then we can declare the array with size 5. After structure declaration we have written a statement "struct student x[5]" is declaring the array x, which can accommodate the details of 5 students. To understand the array of structures, consider the figure given below:
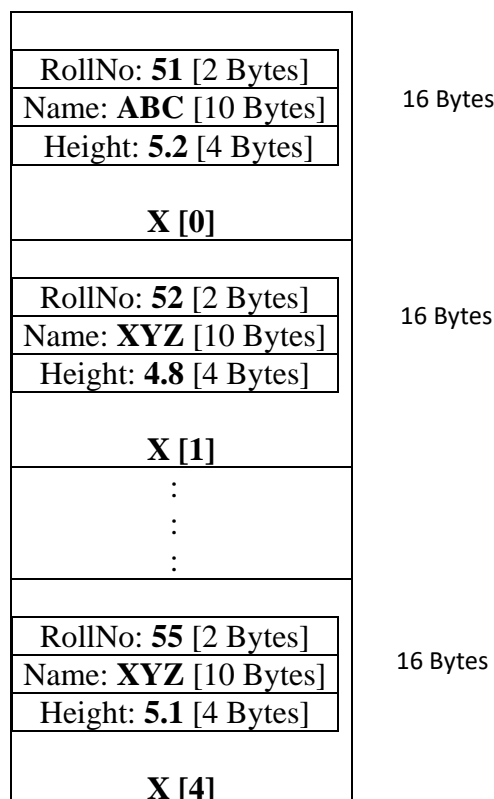
Here we can declared the array of the structure like **'struct student x[5];'**. In this case system will reserve 80 Bytes of space for array x. Each student will occupy 16 bytes of space in the memory (2 Bytes for RollNo, 10 Bytes for Name and 4 Bytes for Height), and our 'x' is an array which is able to store the details of 5 Students [5 * 16 = 80 bytes].

Similar to normal array, here also we can access the data of the students by using index number of the array. For example, first student will be represented like x[0], Second student will be represented like x[1] and so on. Here, you just have to understand that x[0], x[1],…x[4] are variables of type struct students. Now the specific details of the student can be accessible like this:

x[0].roll no=51;   //This will set roll no of the first student by 51
x[0].height=5.2;   //This will set height of the fist student by 5.2
x[1].rollno=52   //This will set roll no of the Second student by 52

Similarly, we can print the data of the first student like:
printf("Roll No:%d",x[0].rollno);
printf("Name of the Student:%s",x[0].name);
printf("Height of the Student:%.2f",x[0].height);

In the case of array, use can use for loop to either access the data of the student or to print the data of the student. Consider the following code segment, suppose if we want to store the details for all 5 students then:

```
void main()
{
        struct student x[5];
        int I;
        for(i=0 ;i<5; i++)
        {
                printf("\nEnter the details for Student-%d", i+1);
                printf("Enter Roll Number:");
                scanf("%d", &x[i].rollno);
                printf("Enter Name:");
                scanf("%s", x[i].name);
                printf("Enter Height:");
                scanf("%f", &x[i].height);
        }
}
```

For loop of variable i will for 5 times and it will take the details of 5 students and store these details in the array x of type students.

1.2.6 Structures within Structures

Structures can be nested, that is, structure templates can contain structures as members. For example, consider two structure types:

struct date
{
int dd, mm, yy;
};
struct employee
{
int empcode;
struct date dob;
float salary;
}x;

In the above code, variable x of type struct employee, will occupy 10 Bytes of space in the memory. To store empcode 2 Bytes, dob 6 Bytes, and salary 4 Bytes. You might be wondering, why dob requires 6 Bytes? The reason is simple it is also structure of type date which has dd, mm and yy integer variables (2Bytes * 3 variables = 6 Bytes). Here, structure employee has dob member element, which again type of date (another structure). This is called nesting of a structure. You can initialize this data elements in following manner:

x.empcode=51;

x.dob.dd=22;

x.dob.mm=8;

x.dob.yy=1976; and so on.

Notice we need to write x.empcode as empcode is member element of structure employee. In the case of date, we need to write x.dob.dd=22, because dob is member element of x and because of dd is a member element of date.

1.2.7 Structures and Functions

Like other data types, a structure can be passed as an argument to a function. Following program uses a function to display data on the screen.

Example:

```
#include <stdio.h>
/* Declare and define a structure to hold the data. */
struct emp_data
 {
      float sal_amount;
      char first_name[30];
      char last_name[30];
} emp_rec;
void print_rec(structemp_datax)
{
      printf("\nDonor %s %s gave $%.2f.\n", x.first_name, x.last_name,x.sal_amount);
 }
/* The function prototypes. The function has no return value, */
      /* and it takes a structure of type data as its one argument. */
void main( )
{
      /* Input the data from the keyboard. */
      printf("Enter the     donor's first    and     last names,\n");
      printf("separated by a space: ");
      scanf("%s %s",emp_rec.first_name, emp_rec.last_name);
      printf("\nEnter the donation amount: ");
      scanf("%f", &emp_rec.sal_amount);
      /* Call the display function. */
      print_rec(emp_ rec );
      return 0;
}
```

In this example, emp_rec is a global variable and hence it is accessible in both the functions (main() and print_rec()). Main function is accepting the values from the user such as donor's first name, last name, amount and encapsulate this information into a single unit of data called emp_rec.

Now, this single unit of data encapsulated under name 'emp_rec' passed to a function print_rec() by a main() function. Function print_rec() will print all the encapsulated details like first name, last name and amount. Here we have passed a structure from main() to print_rec() function. It is also possible if you want to return structure from some function to main() function.

Check Your Progress-4

1. Identify the correct method of initializing data member of the structure.

[A] structure_name. data_memeber_name=value;

[B] structure_varaible.data_member_name = value;

[C] structure_name->data_member_name=value;

[D] structure_varaible->data_member_name = value;

2. Compute memory space requirement for x, for following code consideration.

struct employee

{   int emp_code;

*   char name[10];*

*   struct date { int dd, mm, yy; } dob, doj, doa;*

*   float salary;*

} x;

[A] 16 Bytes   [B] 22 Bytes

[C] 13 Bytes   [D] 34 Bytes

1.3   UNIONS

Like structures, unions also contain member elements whose individual data types may vary from one another. However, all the member elements that compose a union share the same storage area, whereas each member elements within a structure are having its own unique storage area. Thus, Unions provide an effective way of using the same memory location for multi-purpose. Hence, only one of the members will be active at a time. In short, unions are used to conserve memory. The syntax of union can be written as:

union Union_Name

{

*   data type member_element 1;*

*   data type member_element 2;*

*   …………;*

*   data type member_element m;*

};

Where union is a required keyword and the other terms have the similar meaning as in a structure definition. Individual union variables or instances can then be declared as

[storage-class] union Union_Name variable1, variable2, …………, variableN;

where storage class is an optional storage class specifier, union is a required keyword, Union_Name is the name that appeared in the union definition and variable 1, variable 2, variable n are union instances of type Union_Name.

Now let us take an example to illustrate the same:

union employee
{
    char name[20];
    int emp_id;
} emp1, emp2;

Here, we have two union variables, emp1 and emp2 of type employee. Each variable can represent either a 20-character string (name) or an integer quantity (emp_id) of any one time.

A union may be a member of a structure and a structure may be a member of a union. An individual union member can be accessed in the same manner as an individual structure member, using the operators (->) and. (dot). Thus, if a variable is a union variable, then variable.member refers to a member of the union. Similarly, if ptr is a pointer variable that points to a union, then ptr->member refers to a member of that union.

For example, consider the given program:

```
#include<stdio.h>
void main( )
{
        union employee
        {
                char initial_name;
                int emp_id;

        };
        struct emp
         {
                char initial_name;
                int emp_id;

        }
```

printf("\nSize of Structure is: %d" sizeof(struct emp));

printf("\nSize of Union is: %d" sizeof(union employee));

}

In this program, you will get size of structure is:3 and size of union is: 2. That is because of structure variable occupies the memory space that is equal to some of memory spaces occupies by all its members. In the struct emp initial_name is of type character (therefore 1 Byte), and emp_id is integer (therefore 2 Bytes), total 3 bytes.

In the case of union, it occupies memory size is equal to the size occupies by the largest data member. In initial_name (1 Byte) and emp_id (2 Byte) the largest is 2 Bytes. So, union will occupy 2 Bytes of memory space. Make sure in the case of structure we can initialize both data members that is initial_name and emp_id, whereas in the case of union any one data member either initial_name or emp_id can be initialized (not both).

Check Your Progress-5

1._____ occupies more memory. [Union / Structure]

2.The memory space occupies by the variable of _____, is same as memory space needed for its largest data member. [Union / Structure]

3.In _____ all data members can be initialized. [Union / Structure]

4.In _____, we can initialize only one data member [Union / Structure]

5. The memory space occupies by the variable of _____, is sum of all its data members. [Union / Structure]

1.4   LET US SUM UP

In this unit, we:

1. Studied about the method of defining user defined data types, structures and unions

2. Studied about structure initialisation and working with structures

3. Studied about unions and using them in developing programs.

1.5   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

7. [C] Both A and B
8. [C] structure is used to store same type of data**.**

Check Your Progress-2

8. [B] 16 Bytes
9. [A] Sum of memory space required for each data member

Check Your Progress-3

6. [C] 320 Bytes
7. [A] 220 Bytes

Check Your Progress-4

6. [B] structure_varaible.data_member_name=value
7. [D] 34 Bytes

Check Your Progress-5

5. Structure
6. Union
7. Structure
8. Union
9. Structure

1.6   GLOSSARY

7. **Structure** is a user defined data type, which allows us to encapsulate different types of data into a single unit.

1.7   Assignment

7. What is Structure? How can we declare and initialized it?
8. What is Union? How it differs from the Structure?
9. Discuss array of Structures with an example.

1.10   Activity

- Write a program which has a structure, student to store RollNo, Name, Marks of 3 subjects, and Total. Create an array of students to the details of the students such as RollNo, Name and Marks of 3 subjects for 5 students. Compute the total and display RollNo, Name and Total in a table format.

1.11   Case Study

- Write a program to use following structure.

```
struct employee
{   int emp_code;
    char name[10];
    struct date { int dd, mm, yy; } dob, doj, doa;
    float salary;
} x[5];
```

Store the employee details and print in the following manner.

EmpCode	Name	DOB	DOJ	DOA	Salary
1	Ram	22-08-1976	15-07-2000	15-04-2001	48000
2	Shyam	15-08-1975	15-07-2002	12-05-1999	45000

And so on,

1.12   Further Reading

- "Let Us C" by Yashwant Kanetkar.
- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.

UNIT 2 POINTERS

Unit Structure

2.0   LEARNING OBJECTIVES

After working through this unit, you should be able to:

- Know about the concept of pointers

- Know about accessing the address of a variable

- Understand pointer assignments

- Know about pointers and arrays

- Know about Pointers and Structures

2.1   INTRODUCTION

A pointer is feature of C-programming language which refers directly to (or "points to") another variable (value) stored elsewhere in the computer memory using its address. Or in simple words, pointer is special variable which can store the address of another variable.

A pointer identifies or references a memory location, and obtaining the value at the location it refers to is known as dereferencing the pointer. A pointer simply can be considered as the abstract reference data type.

In this unit, we will be discussing about the concepts of pointers and the various operations which can be performed on them.

2.2   UNDERSTANDING POINTERS

A pointer is a variable that stores the memory location of a data item, such as a variable or an array element. Pointers serves many advantages which are listed below:
1. We know that the function can return only one value. With the help of pointer variable, we can make it possible to return multiple values from the function.
2. Pointers are used, if you want to change the value of a local variable declared in one function, and if you want to change the value of it by another function.
3. Pointer can be used in the dynamic memory allocation, to allocate the memory to dynamically created variable.
4. Pointers can also be used to pass the address of an array or a structure from one function to another function.

2.2.1 Accessing the Address of a Variable

*And & Operators

Operator * is used to declare pointer variable. As we know pointer is variable, which is used to store the address of another variable. Then how can we separate it from normal variable which is used to store the values. To separate normal variable and pointer variable, we declare pointer variable with * operator. For example:

int x; // x is normal variable. It is used to store value

int *p; // p is a pointer variable. It is used to store address of another variable

The address operator &, is used to fetch the memory address of any variable. In the above example suppose if we initialize variable x by some value 5 and then we want to print the value and address of variable x using printf() statement then we need to write following code:

x=5;

printf("%d", x); //This will print the value of variable x

printf("%u", &x); //This will print the memory address where x is created

In this program, suppose if we want to store address of variable x into the pointer variable p then we need to write: p = &x; The following program segment will further clear you concept related to * operator and & operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x;  //normal variable
        int *p; //pointer variable
        x=5;    //x stores value 5
        p=&x; //p (pointer stores) address of variable x
        printf("\nValue of X is:%d", x);
        printf("\nAddress of X is:%d", &x);
        printf("\n *p is:%d",*p);
        printf("\n  p is:%d",p);
         printf("\n &p is:%d",&p);
}
```
Output:
Value of X is:5
Address of X is: 12345 (address of x)
*p is: 5
P is: 12345 (address of x)
&p is: 67890 (address of p)

2.2.2 Declaring and Initializing Pointers

Pointer variables, like all other variables, must be declared before; they may be used in C program. When a pointer variable is declared, the variable name must be preceded by an asterisk (*). This identifies the fact that the variable is a pointer. The data type that appears in the declaration refers to the object of the pointer. For example, if pointer variable is declared as integer type, then it stores data item integer type.

Syntax for pointer declaration is as follows

*data_type *ptr;*

Here, *ptr* is the name of the pointer variable and data type refers to the type of the pointer and note that an asterisk must precede ptr.

For example,

float u, v;

*float *p;*

---

p = &v;

The first line declares u and v to be floating point variables. The second line declares p to be a pointer whose object is a floating-point quantity. i.e. p points to a floating-point quantity. Note that p represents an address not a floating-point quantity. An asterisk should not be included in the assignment statement.

A pointer variable can be initialized by assigning in the address of another variable, but you have to declare that another variable earlier in the program. for example,

float u,v;

*float *v=&v;*

v should be declared before pointer v

2.2.3 Accessing a variable through its pointer

A variable can be accessed through its pointer. This can be understood with the help of the given example:

void main( )
{
int x=5;
*int *ptr;*
ptr=&x;
printf("Value of x= %d", x);
printf("Address of x= %d", ptr);
printf("Value of ptr= %d", ptr);
*printf("Value of x= %d",*ptr);*
}
Output:
Value of x= 5;
Address of x =1000
Value of y =1000
Value of x= 5;

Thus, we have seen that variable x's value can be displayed using the pointer variable ptr.

Check Your Progress-2

1. For statement int x, *p; which is a valid statement to initialize pointer variable p?

[A] *p=x;   [B] *p=&x;

[C] p=x;   [D] p=&x;

2.To declare the pointer variable, _____ operator is used.

[A] *   [B] &

[C] ->   [D] ^

2.3 POINTER OPERATIONS

Expressions involving expressions conform to the same rules as other expressions do. Few special aspects of pointer expressions are discussed below.

2.3.1 Pointer Assignments

As we initialize other variables, similarly you may use a pointer at the right-hand side of an assignment statement to assign its value to another pointer. For example,

```
 void main( )
{
      int a;
      int *p1, *p2;
      p1= &a;
      p2=p1;
      printf("%u", p2);      //Print the address of a
}
```

Both p1 and p2 are pointing to variable a.

2.3.2 Pointer Increments and Scale Factor

Only addition and subtraction are the two arithmetic operations, that can be performed on pointers. To understand the same operation, let us consider the given example:

Let p1 be an integer pointer with a value of 7000. Now assuming that integers are 2 bytes long, the given statement:

p1++;

p1 will contain 7002 instead of 7001, as each time when p1 is incremented, it will point to the next integer. Because of in the declaration of p1 pointer we have specified the data type (int) on every increment operation, pointer will add 2 bytes to its address value.

From the above example, it can be pointed out that, each time a pointer is incremented, it points to the memory location of the next element of its base type. With character pointers, it will act in normal manner as characters are always 1 byte long.

2.4   POINTERS AND ARRAYS

If x is a one-dimensional array, then the address of the first array element or base address on an array can be expressed as either &x[0] or simply x, moreover, the address of the second array element can be written as either &x[1] or as (x+1), and so on. In general, the address of $i+1^{th}$ array element is expressed as (x+i). Since x[1] and (x+1) both represent the address of 1 element of x, it would seem reasonable that x[1] and *(x+1) both represent the contents of that address. i.e., the value of the 1st element of x for example.

```
#include <stdio.h>
void main( )
{
        static int x[5] = {10, 11, 12, 13, 14};
        int i;
        for(i=0; i<5;i++)
        printf("\ni=%d x[i]=%d *(x+i)=%d &x[i]=%d (x+i)=%d", i, x[i], *(x+i), &x[i], x+i);
}
```

Executing this program results the following output:

i=0 x[i]=10	*(x+i)=10	&x[i]=72	x+i=72
i=1 x[i]=11	*(x+i)=11	&x[i]=74	x+i=74
i=2 x[i]=12	*(x+i)=12	&x[i]=76	x+i=76
i=3 x[i]=13	*(x+i)=13	&x[i]=78	x+i=78
i=4 x[i]=14	*(x+i)=14	&x[i]=7A	x+i=7A

We can see that the value of the ith array element can be represented either x[i] or x[i+1] and the address of the ith element can be represented by either &x[i] or x+i. While assigning a value to an array element such as x[i], the left side of the assignment statement may be written as either x[i] or as *(x+i). Thus, a value may be assigned directly to an array element or it may be assigned to the memory area whose address is arbitrary address to an array name or to an array element. Thus, expression such as &(x+1) and &x[i] cannot appear on the left side of an assignment statement. For example, these four statements are all equivalent.

line[2] = line[1];

*line[2] = *(line+1);*
(line+2) = line[1];
*(line+2) = *(line+1);*

The address of an array element cannot be assigned to some other array element. We cannot write a statement such as

&line[2] = &line[1]; // Invalid Statement

We can assign the value of one array element to another through a pointer, for example,

pl = &line[1] ;

line[2]=*p1;

pl = line+l;

*(line+2) = *pl;

Numeric array element cannot be assigned initial values if the array is defined as a pointer variable. Therefore, a conventional array definition is required if initial values will be assigned to the element of a numerical array. However, a character type pointer variable can be assigned an entire string as a part of the variable declaration. Thus. a string can conveniently be represented by either a non-dimensional array or a character pointer.

A two-dimensional array, for example is actually a collection of one-dimensional arrays. Therefore, we can define a two-dimensional array as a pointer to a group of contiguous one-dimensional arrays. A two-dimensional array declaration can be written as:

Data-type (*ptvar) expression 2;

This concept can be generalized to higher dimensional arrays that is

Data-type (*ptvar) [expression 2] [expression 3] ... [expression n];

In these declarations data type refers to the data type of the array. ptvar is the name of the pointer variable and expression 1, expression 2 … expression n is positive-valued integer expression that indicates the maximum number of array element associated with each subscript.

The parentheses that surround the array name should normally be evaluated right to left. Suppose that x is a two-dimensional integer array having 10 rows and 20 columns. We can declare x as a

int(*x)[20];

In this declaration, x is defined to be a pointer to a group of contiguous, one-dimensional 20 element integer arrays. Thus, x points to the first 20-element array, which is actually the first row (row 0) of the original two-dimensional array. Similarly, (x+1) points to the second 20 elements of the array, which is the second row (row 1) of the original two-dimensional array and so on.

An individual array element within a multidimensional array can also be accessed by repeatedly using the indirection operator. Usually, however this procedure is more difficult than the conventional method for accessing an array element. The following example illustrates the use of the indirection operator.

Suppose that, x is a two-dimensional integer array having 10 rows and 20 columns, as declared as:

int(*x) [20];

The item in row2, column 5 can be accessed by working either.

*x[2][5] OR *(*(x+2)+5)*

Here, (x+2) is a pointer to row number 2, therefore, the object of this pointer *(x+2), refers to the entire row, since row 2 is a one-dimensional array. *(x+2) is actually a pointer to the first element in row 2 (row starts from 0, i.e., row0, row1…). We now add 5 to the pointer, hence, (*(x+2)+5) is a pointer to element 5 in row2. The object of this pointer, refers to the item in column 5 or row 2 which is x[2][5].

Check Your Progress-3

1. int x[10], *p; statements are given, choose the correct option, to store the address of an array x to pointer variable p.

[A] p=&x;   [B] p=x;

[C] *p=x;   [D] p=&x;

2. int x[10], *p; statement is given, identify the correct statement to store the address of 3rd element of the array into pointe p.

[A] p=x[3];   [B] *p=x[3];

[C] p=&x[3];   [D] *p=&x[3];

3. char x[]="GUJARAT"; printf("%c",*(x+2)); will print _____.

[A] G   [B] U

[C] R   [D] J

2.5   POINTERS AND CHARACTER STRINGS

Now let us understand, how the pointers are reacting with string variables. Consider the following program in which we have stored a string without declaring array and we are printing that screen. Let we can't to store string "Programming is my Passion" then:

char *p = "Programming is my Passion";

This declaration stores the string in the memory and returns the starting address of the string into pointer p. For example,

```
void main( )
{
      char *p = "Programming is my Passion";
      while (*p!='\0')
      {
            printf("%c", *p);
            p++; //In this program address in pointer p gets change
      }
}
```

Because, p++ the address in the pointe will be change and we can not get back the starting address of the string again. To overcome this problem, we can use integer variable i.

Using variable i we can access different characters of string without changing address, which stored in the pointer p.

```
void main( )
{
        char *p = "Programming is my passion";
        int i;
        for(i=0; p[i]!='\0'; i++)
                printf ("%c", p[i]);
}
```

2.6   POINTERS AND FUNCTIONS

In the case of functions, if the user defined function wants to read the value of any variable, which is locally declared in another function then it is possible. For Example, in the program of sum() function, function is reading the data which is passed by main(). Here, the values passed by the main() function is copied into two variables (parameters) of sum(). Function sum() just do the sum and return back answer to main().

But do you think, sum() function can do any change in the variables locally declared in the main() function? Answer is no. That is the reason, we can easily implement user defined function to compute sum(), but suppose if you want to implement a function, which swap the values of local variable declared in main() is little bit tricky. Consider the following program in which we are making function swap(), to swap the values of variables a and b, which are locally declared in the main() function.

In the following program, when the swap() function is called, at that time values passed of variable a and b, are copied in the paraments x and y of the swap(), and then whatever logic we are writing to swap the variables are swapping variables x, y but not variable a and b.

```
void main( )
{
    int a=11, b=28;
    swap(a, b); //Passing values
    printf("Value of A is:%d, and B is:%d", a, b);
}
void swap(int x, int y) //Values are taken in normal variables x and y
{
    int tmp =x;
    x=y;
    y=tmp;
}
```
Output:
Value of A is:11, and B is:28

You can see here, we didn't get swap in the values of variable a and b, even after calling the swap function. Solution to this problem is pointer. Using pointer we can enable to swap()

function, to swap (write) the local variables a and b or main function, see the solution given below:

```
void main( )
{
    int a=11, b=28;
    swap(&a, &b); //Passing addresses of variable a and b
    printf("Value of A is:%d, and B is:%d",a, b);
}
void swap(int *x, int *y) //Addresses are taking in *x and *y
{
    int tmp =*x;
    *x=*y;
    *y=tmp;
}
```

In this program, instead of passing the values to the swap function, we are passing the addresses of variable a and b to the swap() function. Function swap() accept those addresses into the pointer variables x and y. Finally, using addresses swap() function, change the local variables a and b of the main() function. Therefore, by implementing this program we have to understand that the local variable of the function can be changed via another function using pointers. Now, see the following program in which function 'calc' computes sum, subtraction, multiplication and division and returns all these values to main.

```
#include<stdio.h>
void main( )
{
    int a=20, b=5, ans1, ans2, ans3, ans4;
    calc(a,b, &ans1, &ans2, &ans3, &ans4);
    printf("Sum is:%d, Subtraction is:%d, Multiplication:%d, Division is:%d", ans1, ans2, ans3,
ans4);
}
void calc(int x, int y, int *sum, int *sub, int *mul, int *div)
{
    *sum=x+y;
    *sub=x-y;
    *mul=x*y;
    *div=x/y;
}
```

OUTPUT:

Sum is:25, Subtraction is:15, Multiplication:100, Division is:4

Here, calc function returns 4 values to main() function and still it's return type is 'void'.

Check Your Progress-4

1. When the reference of the local variable is to passed to the function, instead of value?

[A] If function is reading the value of the variable.

[B] If function needs to write the value of the variable.

[C] To make those variables global.

[D] None of the above.

2. If x and y are integer variables then, for function declared as int my_function(int *p), which is a correct way to call that function?

[A] y=myfunction(&x);   [B] y=myfunction(*x);

[C] y=myfunction(x);   [D] All are valid options

2.7   POINTERS AND STRUCTURES

As we have pointers pointing to integers, float and character variables. Similarly, we can have pointers pointing to structures also; those pointers are called structure pointers.

```
void main( )
{
        struct emp
        {
                char name[25];
                char designation[30];
                int empid;
        };
        struct emp e= {"Jose", "IT Administrator", 1254};
        struct emp *p;
        p=&e;
        printf("\n%s %s %d", e.name, e.designation,e.empid);
        printf("\n%s %s %d", ptr->name, ptr->designation, ptr->empid);
}
```

Notice the second statement of printf( ), instead of using dot(.) operator an arrow operator (->) is used.

2.8   POINTS TO POINTERS

- If a pointer variable is not initialized with some address and if some assignment is done to that variable, it will cause the value of that particular variable to be written at some unknown memory location. This type of problem is often unnoticed if the program is small. But when the program grows, the probability of p pointing to some wrong address increases and eventually your program stops working.

- Sometimes due to incorrect assignment problem arises. If suppose p is a pointer variable and x is an integer variable and if we have done the assignment as p=x and when printf("%d", *p) will be written it will print some unknown value as the assignment p=x is wrong as that statement assigns the value of x to the pointer p. However, p is supposed to contain an address, not a value. Therefore, the assignment should be p=&x.

- As you never know where your data will be placed in memory so for the same reasons making any comparisons between pointers will yield unexpected results as they do not point to a common object. For example:

char s1[20], s2[20];
char *p1, *p2;
p1=s1;
p2=s2;
if(p1<p2)
will be an invalid concept.

- While using pointers be careful and make sure that you know where each pointer is pointing before it is used in a program.

Check Your Progress-5

1. To access data elements of structure, by pointer of it, _____ operator is used.

[A] -> (Arrow)   [B] . (Dot)

[C] : (Colon)   [D] :: (Scope resolution)

2. Use of pointer to a pointer variable is _____.

[A] To change local variable value, through another function.

[B] To change the value of local structure variable, through another function

[C] To change the reference stored in pointer, through another function.

[D] None of the above

3. From the statements given below, identify false statement.

[A] Pointer is a special variable, used to store reference of another variable.

[B] Using pointer, function can return multiple values.

[C] Name of the array itself is pointer.

[D] Pointer variable is equally important for global variables.

2.9   LET US SUM UP

In this unit, we

- Have studied about the method of accessing address of a variable using pointers

- Have studied about declaring and initialising pointers

- Have discussed about using pointers with functions

Have studied about using pointers with structures

2.10   SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1. [C] Pointer
2. [B] &

Check Your Progress-2

1. [D] p=&x;
2. [A] *

Check Your Progress-3

1. [B] p=x;
2. [C] p=&x[3];
3. [D] J

Check Your Progress-4

1. [B] if function needs to write the value of the variable.
2. [A] y=myfunction(&x);

Check Your Progress-5

1. [A] -> Arrow
2. [C] To change the reference stored in pointer, through another function.
3. [D] Pointer variable is equally important for global variable.

2.11 GLOSSARY

7. **Pointer** is a special variable used to store reference of another variable.

8. **String** is a group of characters. In C-Language it is represented by character type array.

9. **Function** is a group of executable statements which is used to make modular programming. It provides reusability of code, and reduces complexity of the program.

2.12   Assignment

1. What is Pointer? How can we declare and initialized it?
2. List the advantages of pointer variables.
3. Discuss, how pointer can be used with array?
4. How pointer is useful in function? Explain it with an example.

2.13   Activity

- Write a program which main() function has integer variables x, y, sum, sub, mul and div. Accept two numbers from the user and store it in the variable x and y. Call a function 'calc()' with necessary input, so that function will calculate sum, subtraction, multiplication and division of variable x and y.

(Note that we have discussed that, function can return only one value. This is the example where function is returning 4 values that is addition, subtraction, multiplication and division using pointer).

2.14   Case Study

- Write a program to implement singly link list.

2.15   Further Reading

- "Let Us C" by Yashwant Kanetkar.
- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw-Hill Education.
- "Programming in C" by Reema Thareja, Second Edition by Oxford publication.

UNIT 3   FILES HANDLING

Unit Structure

3.0   LEARNING OBJECTIVES

After working through this unit, you should be able to

- Know about management of files

- Understand the concept of file

- Understand performing various operations on files

- Know about performing input/output operations on files

- Know about error handling during I/O operations.

3.1   INTRODUCTION

To store the data, we have created variables in all the programs. Variable usually created in the primary memory called RAM (Random Access Memory). It is a volatile memory, means the data within the memory exists till we provide continuous (uninterrupted) power supply. If we discontinue the power supply then data stored in the primary memory is erased. To store the data permanently, we need to store the data in the files.

Files are sequences of bytes, stored on the permanent memory (secondary memory). Secondary memories are usually a disk. Once we store the data on the disk, it will remain as it is, even if discontinue power supply. In this chapter we will be focusing on how can we read and write the data in the files by writing C-Programs.

3.2   MANAGEMENT OF FILES

Different types of operations can be performed on the file, once it is created. C-Language offers facility to create and work with files for large amount of data, which has to been stored in the form of files. The term File management or File handling refers to managing of files or working with files.

3.2.1 Files

A file is a place on the disk (secondary memory) where a group of related data is stored. Some basic file operations can be performed as:

- Give a name to the file.

- Open a particular file from particular location.

- Reading the data from the opened file.

- Write the data to opened file.

- Close the file, when work is completed.

3.2.2 Defining and Opening a File

When working with a file, a temporary buffer area must be created to store information which will be transferred between the random-access memory and the data file on disk. This buffer area permits data to be read from or written to the data file. The buffer area is created, while we write the data to file. To work with the file, we need to create the instance of the FILE as given below:

*FILE *fp;*

Here FILE (in uppercase) is a special structure type that creates the buffer area and 'fp' is a pointer variable that designates the beginning of the buffer area. The structure type file is defined with a system, by including header file 'stdio.h'.

A data file must be opened or created before it is processed. Here, we need to associates the file name with the buffer area. It also specifies how the data file will be utilized, that means it is a read only file, a write-only file or a read/write file, in which both operations are permitted. The library function fopen() is called to open a file. This function can be written as:

fp = fopen("file_name", "file_type")

Where, file_name and file_type are string arguments, that represents the name of the data file and the mode (read, write or append) in which the data file will be utilized, respectively. The name chosen for the file, must be according to the rules for naming files. The file_type must be one of the strings shown in the following table:

File_type	Meaning
"r"	opens an existing file to read the content from.
"w"	opens a new file to write the content to.
	If the file with specified file name is exists, then it will overwrite (destroyed and new file is created in its place).
"a"	open an existing file to append the data. If the file is not exists, then a new file will be created.
"r+"	opens an existing file to read as well as write.
"w+"	opens a new file for to read as well as write. If a file with the specified file_name currently exists, it will be overwritten (deleted and a new file is created in its place).
"a+"	opens an existing file to read and write. A new file will be created if the file with the file does not exist.

The fopen() function returns a NULL value if it fails (the file cannot be opened or when an existing file cannot be found), otherwise it returns a file pointer to the beginning of the file. At the end of program, we need to close the file, so that all outstanding data related with the file is flushed out from the buffers and all links to the file are destroyed. It also prevents any accidental corruption of the file.

3.2.3 Closing a file

Operating system restricts number of files to be opened. So, in that case closing of unnecessary files might help to open other required files. Sometimes, when we need to reopen the same file in a different mode, then first we need to close a file. This can be able with the library function fclose(). The syntax is as shown below:

fclose(fp);

Example:
#include <stdio.h>
void main( )
{
    *FILE *fp;*
    fp=fopen("myfile.dat","w");
    if(fp= =NULL)
        printf("\n File Can not Open);
    fclose(fp);
}

Check Your Progress-1

1. To create the file pointer _____ function is used.

[A] file   [B] FILE

[C] fopen   [D] fclose

2. Syntax for opening a file is _____.

[A] fpt=fopen("path","mode");   [B] fpt=fopen("mode", "path");

[C] fopen("mode","path");   [D] None of the above

3.To close the file, which is a correct syntax?

[A] exit(fpt);   [B] quit(fpt);

[C] fpt=flose();   [D] flcose(fpt)

3.3   INPUT/OUTPUT OPERATIONS ON FILES

THE fgetc AND fputc FUNCTIONS:

The simplest file i/o functions are fgetc() and fputc(). These functions are used to handle single character at a time.

fgetc() is used to read character from a file that has been opened in read mode("r"). For example, the statement

C = fgetc(fp1);

Would read character from file whose file pointer is fp1. The file pointer moves by one-character position for every read and write operation of fgetc and fputc respectively. Therefore, the reading should be terminated when end-of-file (EOF) is encountered.

Assume that a file is opened with mode "w" with a file pointer fp1. Then, the statement

fputc(c,fp1);

Writes value of variable c to the file associated with FILE pointer fp1.

For example,

```
void main()
{
        FILE *fpt;
        char ch;
        fpt = fopen("input.dat", "w");
        while(ch=getchar()) != EOF)
        {
                fputc(ch,fpt);
        }
        fclose(fpt);
        fpt = fopen("input.dat","r");
        while(ch=fgetc(fpt)) != EOF)
        {
                printf("%c",ch);
        }
        fclose(fpt);
}
```

THE fgets() AND fputs() FUNCTIONS:

Functions fgets() and fputs() are used to read or write a string from /to file. The function fgets() is used to read a string with specified length from a file, which is opened in read mode. For syntax is:

fgets(string, 80, fp);

It would read string of 80 characters from the file whose file pointer is fp store it in the character array string. The function will return '\0' (NULL) character when reached at the end of file.

To write the data into the file, we need to open the file in the write mode and we need to call fputs() function as follows:

fputs(string, fp);

This will write the string contained in the character array string to the file associated with FILE pointer fp.

For example:

```
#include<stdio.h>
#include<stdio.h>
void main( )
{
   FILE *fp;
   char string[80];
   fp = fopen("myfile.txt", "w");
   printf("Enter Your Name:");
   scanf("%s", string);
   fputs(string, fp);
   fclose(fp);
   fp = fopen("myfile.txt", "r");
   fgets(string,80, fp);
   fclose(fp);
   printf("\nYour Name is:%s", string);
}
```

In this program, we have taken a string array of type character, and we are creating a file myfile.txt in write mode with the help of file pointer fp. We take the data from the user into string array and we write it into the file using fputs(). We are closing the file and open it the same in read mode. We are fetching the data from the file using fgets() and store it in the string array. Finally, we print the sting array.

THE fscanf() AND fprintf() FUNCTIONS:

These functions are used to deal with multiple data types. The reading or writing in formatted form from/to the files can be done. The syntax of fprintf() is

fprintf (fp, "control string", list);

For Example,

fprintf (fp1, "%s%d%f", name, age, salary);

Here fp1 is the file pointer, name is an array variable of type character, age is integer variable. The general form of fscanf() is :

fscanf(fp1, "control string", list);

This statement would cause the reading of the items in the list from the file specified by fp1, according to the specifications contained in the control string. For example,

fscanf(fp1, "%s%d%f", &item1, &qty1,&price1);

fscanf() returns the number of items that are successfully read. When the end of the file is reached, it returns the value of EOF.

THE fread() AND fwrite() FUNCTIONS:

Some commercial applications include the use of data files to store blocks of data (in binary) rather than characters, where each block consists of a fixed number of contiguous bytes. Here, block is generally represented a complete data structure, such as a structure or an array. For example, a data file may consist of multiple variables of structure having the same composition or it may contain multiple arrays of same type and size. For such type of applications, it is required to read the entire block (variable of structure) from the data file or write the entire block (variable of structure) to the file. This is used only with binary files.

Each of these functions requires following four arguments: (i) a pointer to the data block, (ii) size of data block, (iii) number of data blocks being transferred and (iv) the file stream pointer. Thus, the fwrite() function can be written as:

fwrite(&struct_varaible, sizeof(struct_varaible),1, file_ptr);
fread(&struct_varaible, sizeof(struct_varaible),1, file_ptr)

Here *struct_varaible* is a structure variable and file_ptr is the file pointer associated with the file that has been opened to read/write. Once an unformatted data file has been created with fwrite(), it can be read with fread() function. The function returns a zero value when end-of-file condition has been detected and non-zero value when end-of-file is not detected. Hence, a program that reads an unformatted data file can be reading file, as long as the value returned by fread() is non-zero value.

An example program to create an unformatted data file containing book information records:

```
#include<stdio.h>
#include<stdlib.h>
struct book
{
        int bookno;
        char title[30];
        float price;
} book;

void main()
{
        int ch, i, n;
        float p;
        FILE *fp;
```

```c
        printf("\n Enter number of records:");
        scanf("%d", &n);
        flushall();
        fp=fopen("book.dat","a+b");
        if(fp == NULL)
        {
                printf("\n File Cannot open");
                exit(0);
        }
        for(i=1;i<=n;i++)
        {
                printf("\n Enter Book Code=");
                scanf("%d", &book.bookno);
                printf("\n Enter Book Title=");
                scanf("%s", &book.title);
                printf("\n Enter Book Price=");
                scanf("%f", &p);
                book.price=p;
                fwrite(&book, sizeof(bk), 1, fp);
        }
        fclose(fp);
        fp=fopen("book.dat","rb");
        if(fp == NULL)
        {
                printf("\n File Cannot open");
                exit(0);
        }
        printf("\n Book no\tTitle\t\tPrice\n");
        while(!feof(fp))
        {
                fread(&book, sizeof(book),1,fp);
                printf("\n %d\t\t %s\t\t %.2f", book.bookno, book.title, book.price);
        }
        fcloseall();
}
```

Output:

Enter Number of records:2

  Enter Book Code=1

Enter Book Title=c-programming

Enter Book Price=300

Enter Book Code =2

Enter Book Title=c++

Enter Book Price=450

Book no	Title	Price
1	cprogramming	300.00
2	c++	450.00

Check Your Progress-4

1. Function fread() and fwrite() is used for _____ type of file.

[A] Text   [B] ASCII

[C] Binary   [D] None of the above

2. To fetch the data byte wise from a file _____ file function is used.

[A] fprintf()   [B] fread()

[C] fputs()   [D] fgets()

3.4 ERROR HANDLING DURING I/O OPERATIONS

The standard library function ferror( ) is used to check whether the file is opened successfully or there is some error in the opening of the file. It returns any error that might came during any read or write operation on a file. For successful read/write operation, it returns a zero value otherwise a non-zero value in case of a failure.

For example,

void main ( )

{

*file *fpt;*

char ch;

fp=fopen("hello, "r");

while(!feof(fpt))

{

ch=fgetc(fpt));

```
if(ferror())
{
        printf("error in reading file");

        break;
}
else
        printf("%c", ch);

}
fclose(fpt);

}
```

Check Your Progress-5

1. _____ function is used to detect any error, in read/write operation of file.

[A] error()   [B] _Error()

[C] ferror()   [D] None of the above

2. To open an existing file, for adding content to it, file has to open with ____ mode.

[A] r   [B] w

[C] a   [D] None of the above

3.5 LET US SUM UP

In this unit, we

- Discussed about solving more complex problems related to files

- Discussed about the different functions required for performing various operations on file

- Discussed about the different functions required to perform input and output operations on files

- Discussed about situations where error can be handled during i/o operations

3.6 SUGGESTED ANSWERS FOR CHECK YOUR PROGRESS

Check Your Progress-1

1. [B] File
2. [A] fpt=fopen("path","mode");
3. [D] fclose(fpt);

3.7 GLOSSARY

1. **File:** File is a representation of data, stored in permeant secondary memory.

3.8 Assignment

1. What is File? How can we can we open it? Discuss various modes of it.
2. Discuss fprintf() and fscanf() functions of it.

3.9 Activity

Write a Program to insert the following contents in a file named "File1".
Customer No. Account Type Balance
101   Savings   2000
102   Current   5000
103   Savings   3000
104   Current   10000
Append the contents of "File1" in another file "File2". Also display the contents of File2 on screen.

3.10 Case Study

- Write a program to copy a text file.

3.11 Further Reading

- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.

UNIT 4   SOLVED PROGRAMS – III

In this unit we will solve some programs, which will clear your concept about the different topics, we have discussed earlier. So, by understanding these programs your programming fundaments related to Array, Strings, Functions, Structure, Pointer and File will become clearer.

PROGRAMS OF ARRAYS

```c
/* Program:1 WRITE APROGRAM TO FIND THE MINIMUM AND MAXIMUM VALUE */
#include<stdio.h>
void main()
{
   int arr[10], i, max, min, num;
   printf("Enter Number of Elements for array:");
   scanf("%d", &num);
   for(i=1; i<=num; i++)
   {
     printf("Enter Number:");
     scanf("%d", &arr[i]);
   }
   for(i=1, max=arr[0]; i<=num; i++)
   {
     if(arr[i]> max)
     {
       max =arr[i];
     }
   }
   for(i=1, max=arr[0]; i<=num; i++)
   {
     if(arr[i]<min)
     {
       min=arr[i];
     }
}
printf("\nMaximum Value is:%d\n",max);
printf("Minimum Value is:%d\n",min);
}
```

Enter Number of Elements for array: 5

Enter Number: 28

Enter Number: 76

Enter Number: 24

Enter Number: 11

Enter Number: 45

Maximum Value is: 76

Minimum Value is: 11

/*Program:2 PROGRAM TO CONVERT THE DECIMAL NUMBER TO

- *BINARY NUMBER*
- *OCTAL NUMBER*
- *HEXA-DECIMAL NUMBER */*

```c
#include<stdio.h>
void main()
{
  void hexadecimal(int);
  int base1,number1,number2,a1[20],b1[20],i,j,k,ch;
  float no1,num2;
  char answer;
  printf(" -: Convert Decimal TO:-\n");
  printf("*******************************************\n");
  printf(" 1 BINARY NUMBER \n");
  printf(" 2 OCTAL NUMBER \n");
  printf(" 3 HEXADECIMAL NUMBER \n");
  do
  {
    printf("ENTER CHOICE = ");
    scanf("%d",&ch);
    printf("\n\nENTER DECIMAL NUMBER:-\t");
    scanf("%f",&no1);
    switch(ch)
    {
      case 1: printf("\nBINARY NUMBER IS = "); base1=2; break;
      case 2: printf("\nOCTAL NUMBER IS = "); base1=8; break;
      case 3: printf("\nHEXA-DECIMAL NUMBER IS = ");
      base1=16;
```

```
            break;
            default: printf("\nWRONG CHOICE IS SELECTED.TRYAGAIN.");
            break;
    }
    i=j=0;
    number2=no1;
    num2=no1-number2;
    while(number2 > 0 || number2 > 1)
    {
        a1[i]=number2 % base1; number2=number2 / base1; i++;
    }
    while(num2 > 0.00)
    {
        num2=num2 * base1;
        number1=num2;
        b1[j]=number1;
        num2=num2-number1;
        j++;
        if(j==4)
        {
            break;
        }
    }
    if(base1==2 || base1==8)
    {
        for(i=i-1;i>=0;i--)
        {
            printf("%d",a1[i]);
        }
        printf(".");
        for(k=0;k<j;k++)
        {
            printf("%d",b1[k]);
        }
    }
```

else

```
    {
       for(i=i-1;i>=0;i--)
       {
          hexadecimal(a1[i]);
       }
       printf(".");
       for(k=0;j>k;k++)
       {
          hexadecimal(b1[k]);
       }
    }
    printf("\nCONTINUE(Y/N)?\n");
    scanf("%s",&answer); }while(answer=='y' || answer=='Y');
}
/* Function Hexadecimal */
void hexadecimal(int c1)
{
   switch(c1)
   {
     case 10: printf("A"); break;
     case 11: printf("B"); break;
     case 12: printf("C"); break;
     case 13: printf("D"); break;
     case 14: printf("E"); break;
     case 15: printf("F"); break;
     default: printf("%d",c1); break;
   }
}
```

OUTPUT:

-: Convert Decimal TO:-

****

1 BINARY NUMBER

2 OCTAL NUMBER

3 HEXADECIMAL NUMBER

ENTER CHOICE = 1

ENTER DECIMAL NUMBER:- 45

BINARY NUMBER IS = 101101.

CONTINUE(Y/N)?

Y

ENTER CHOICE = 2

ENTER DECIMAL NUMBER:- 45

OCTAL NUMBER IS = 55.

CONTINUE(Y/N)?

Y

ENTER CHOICE = 3

ENTER DECIMAL NUMBER:- 45

HEXA-DECIMAL NUMBER IS = 2D.

PROGRAMS OF STRINGS

```c
/*Program:3 WRITE A PROGRAM TO SORT A LIST OF NAMES IN ALPHABETIC
ORDER.*/
#include<stdio.h>
#include<string.h>
void main()
{
  int i,j;
  char n1[5][20],t1[5][20];
  for(i=0;i<5;i++)
  {
    printf("ENTER NAME: ");
    gets(n1[i]);
    strcpy(t1[i],n1[i ]);
  }
  for(i=0;i<5;i++)
  {
    for(j=i+1;j<5;j++)
    {
      if(strcmp(n1[i],n1[j])>0)
      {
        strcpy(t1,n1[i]);
        strcpy(n1[i],n1[j]);
        strcpy(n1[j],t1);
      }
    }
  }
```

```
    printf("\t\t\nOrder are:\n");
    for(i=0;i<5;i++)
    {
        puts(n1[i]);
    }
}
```
OUTPUT:
ENTER NAME: kamesh

ENTER NAME: ramesh

ENTER NAME: nilesh

ENTER NAME: kalpesh

ENTER NAME: kamlesh

Order are:

kalpesh

kamesh

kamlesh

nilesh

ramesh

```
/*Program:4 WRITE APROGRAM TO COUNT AND DISPLAY ALL THE VOWELS IN A
GIVEN LINE OF TEXT. */
#include<stdio.h>
#include<string.h>
void main()
{
    char str1[20];
    int a=0, e=0, i=0, o=0, u=0, j, k, ch=0, total=0;
    printf("Enter the string:");
    gets(str1);
    for(j=0;j<strlen(str1);j++)
    {
        switch(str1[j])
        {
            case 'a':
            case 'A' :
                a++;
                break;
            case 'e':
            case 'E' :
                e++;
                break;
```

```
        case 'i' :
         case 'I' :
            i++;
            break;
        case 'o' :
        case 'O' :
            o++;
            break;
        case 'u' :
        case 'U' :
            u++;
            break;
      }
   }
   printf("\nThe total no of a or A are:%d", a);
   printf("\nThe total no of e or E are:%d", e);
   printf("\nThe total no of i or I are:%d", i);
   printf("\nThe total no of o or O are:%d", o);
   printf("\nThe total no of u or U are:%d", u);
   total=a+e+i+o+u;
   printf("\n The total no of vowels are:%d", total);
}
```

OUTPUT:

Enter the string: C-Programming Language

The total no of a or A are:3

The total no of e or E are:1

The total no of i or I are:1

The total no of o or O are:1

The total no of u or U are:1

The total no of vowels are:7


```
/* Program:5 PROGRAM TO COUNT THE NUMBER OF TIMES THE LETTER IN THE
STRING IS REPEATED AND DISPLAY A LIST OF REPETATION OF EACH LETTERS.
*/
#include<stdio.h>
#include<string.h>
void main()
{
        char str1[20], c, str2[20] = {0};
        int n[20]={0}, len1, i, len2, j, flag=0;
```

```c
    printf("\n\nSTRING = ");
    scanf("%s", str1);
    len1=strlen(str1);
    for(i=0; i<len1 ;i++)
    {
      for(flag=0, j=0; j<len1;)
      {
        if(str1[i]==str2[j])
        {
          n[j]=n[j]+1;
          j=len1;
          flag=1;
        }
        else
          j++;
      }
      if(flag==0)
      {
        str2[strlen(str2)]=str1[i];
        n[strlen(str2)-1]=1;
      }
    }
    printf("\n\tREPETATION OF LETTERS: -\n");
    for(i=0;i<strlen(str2);i++)
    {
      printf("\n\t %c -> %d ",str2[i],n[i]);
    }
}
```

OUTPUT:

STRING = banana

    REPETATION OF LETTERS: -

     b -> 1

     a -> 3

     n -> 2

```c
/* Program:6 Reverse the given string without using built-in function */
#include<stdio.h>
void main()
{
    char str[10], ch;
    int i=0,j=0;
    printf("Enter Any String: ");
    scanf("%s",str);
    while(str[j]!='\0')
       j++;
    j--;
    while(i<j)
    {
       ch=str[i];
       str[i]=str[j];
       str[j]=ch;
       i++;
       j--;
    }
    printf("Reverse String is: %s", str);
}
```

OUTPUT:

Enter Any String: BSCIT

Reverse String is: TICSB

```c
/* Program:7 Check given string is Palindrome or not */
#include<stdio.h>
#include<string.h>
void main()
{
    char str[10], rstr[10];
    int i;
    printf("Enter Any String:");
    scanf("%s", str);
    strcpy(rstr, str);
    strrev(rstr);
    i=strcmp(str, rstr);
```

```c
if(i==0)
    printf("Given String is Palindrome:");
else
    printf("Given string is Not Palindrome");
}
```

OUTPUT:

Enter Any String: madam

Given String is Palindrome:

```c
/* Program:8 check given string is Palindrome or not, without using any built-in function */
#include<stdio.h>
void main()
{
    char str[10], ch;
    int i=0,j=0,logic=1;
    printf("Enter Any String:");
    scanf("%s",str);
    while(str[j]!='\0')
        j++;
    j--;
    while(i<j)
    {
        if(str[i]!=str[j])
        {
            logic=0;
            break;
        }
        i++;
        j--;
    }
    if(logic==1)
        printf("Given String is Palindrome:");
    else
        printf("Given String is Not Palindrome:");
}
```

OUTPUT:

Same output as previous program (Program-7)

PROGRAMS OF STRUCTURES

/*Program: 9 Example of Array of Structures, Having array inside the Structure*/

```c
#include<stdio.h>
struct stu
{
    int rollno;
    char name[10];
    int marks[3];
    int total;
};
void main()
{
    struct stu x[3];
    int i,j;
    for(i=0;i<3;i++)
    {
        printf("Enter RollNo:");
        scanf("%d", &x[i].rollno);
        printf("Enter Name:");
        scanf("%s",x[i].name);
        printf("Enter Marks for 3 Subjects:\n");
        x[i].total=0;
        for(j=0;j<3;j++)
        {
            printf("Student[%d]-Marks[%d]:",i+1,j+1);
            scanf("%d",&x[i].marks[j]);
            x[i].total=x[i].total+x[i].marks[j];
        }
    }
    printf("RollNo  Name  Total");
    printf("\n/--------------------/");
    for(i=0;i<3;i++)
        printf("\n%d\t%s\t%d",x[i].rollno,x[i].name,x[i].total);
}
```

OUTPUT:

Enter RollNo:1

Enter Name:ABC

Enter Marks for 3 Subjects:

Student[1]-Marks[1]:39

Student[1]-Marks[2]:50

Student[1]-Marks[3]:70

Enter RollNo:2

Enter Name:XYZ

Enter Marks for 3 Subjects:

Student[2]-Marks[1]:45

Student[2]-Marks[2]:78

Student[2]-Marks[3]:84

Enter RollNo:3

Enter Name:PQR

Enter Marks for 3 Subjects:

Student[3]-Marks[1]:90

Student[3]-Marks[2]:87

Student[3]-Marks[3]:94

RollNo   Name   Total

|--------------------|

1   ABC   159

2   XYZ   207

3   PQR   271

/*Program: 10 Filter Hotel Details by Price */

```c
/*Program: 10 Filter Hotel Details by Price */
#include<stdio.h>
struct hotel
{
    int h_id;
    char h_name[20];
    int price;
};
void main()
{
    struct hotel h[3];
    int i, fltrprc;
```

```
  for(i=0;i<3;i++)
   {
      printf("Enter Hotel ID:");
      scanf("%d",&h[i].h_id);
      printf("Enter Hotel Name:");
      scanf("%s",h[i].h_name);
      printf("Enter Price:");
      scanf("%d",&h[i].price);
   }
   printf("Enter Filter Price:");
   scanf("%d",&fltrprc);
   printf("Hotel Details After Applying Filter:\n");
   printf("HotelID  Name  Price");
   printf("\n----------------------");
   for(i=0;i<3;i++)
   {
      if(h[i].price <= fltrprc)
         printf("\n%d\t%s\t%d",h[i].h_id,h[i].h_name,h[i].price);
   }
}
```

OUTPUT:

Enter Hotel ID:1

Enter Hotel Name:ABC

Enter Price:5000

Enter Hotel ID:2

Enter Hotel Name:XYZ

Enter Price:6000

Enter Hotel ID:3

Enter Hotel Name:PQR

Enter Price:4000

Enter Filter Price:5000

Hotel Details After Applying Filter:

HotelID Name Price

----

1   ABC   5000

3   PQR   4000

PROGRAMS OF FILES

/* Program: 10 Accept numbers from the user and store them in the Nums.bin file, until user inputs 999. Open Nums.bin file read each number and stored them into Even.bin or Odd.bin based on number is even or odd. Print the content of Even and Odd file */

```c
#include<stdio.h>
void main()
{
  int num;
  FILE *fp, *fp1, *fp2;
  fp=fopen("Nums.bin","w");
  do
  {
    printf("Enter Any Number:");
    scanf("%d", &num);
    if(num==999)
      break;
    else
      putw(num,fp);
  }while(num!=999);
  fclose(fp);
  fp=fopen("Nums.bin","r");
  fp1=fopen("Even.bin","w");
  fp2=fopen("Odd.bin","w");
  while((num=getw(fp))!=EOF)
  {
    if(num%2==0)
      putw(num,fp1);
    else
      putw(num,fp2);
  }
  fclose(fp);
  fclose(fp1);
  fclose(fp2);
```

```c
printf("Content in the Even File:\n");
fp=fopen("Even.bin","r");
while((num=getw(fp))!=EOF)
    printf("%d\t",num);
fclose(fp);
printf("\nContent in the Odd File:\n");
fp=fopen("Odd.bin","r");
while((num=getw(fp))!=EOF)
    printf("%d\t",num);
fclose(fp);
}
```

OUTPUT:

Enter Any Number:75

Enter Any Number:45

Enter Any Number:44

Enter Any Number:87

Enter Any Number:98

Enter Any Number:23

Enter Any Number:28

Enter Any Number:999

Content in the Even File:

44   98   28

Content in the Odd File:

75   45   87   23

/*Program:11 Take a text data from the user and write that data in the test.txt file. Open the test.txt file and show the content */

```c
#include<stdio.h>
void main()
{
    char ch;
    FILE *fp;
    fp=fopen("E:\\test.txt","w");
    printf("\nEnter your text Now:\n");
    while((ch=getchar())!=EOF)
        putc(ch,fp);

    fclose(fp);
```

```
    fp=fopen("E:\\test.txt","r");
    printf("\nText Entered by you is:\n");
    while((ch=getc(fp))!=EOF)
    {
        printf("%c",ch);
    }
}
```

OUTPUT:

Enter your text Now:

This is sample text,

having 3 lines

of data.

^Z

Text Entered by you is:

This is sample text,

having 3 lines

of data.

[Make sure we have used EOF in this program. So, after inputting the text you need to press Ctrl+Z and the Enter key which denoted as ^Z in the OUTPUT]

/*Program: 12 Write student details in the file. Open the file, read and display its content on the console*/

```
#include<stdio.h>
struct stu
{
    int rollno;
    char name[10];
    float age;
};
void main()
{
    struct stu x[10];
    char tmpnm[10];
    int i,tmprno;
    float tmpage;
    FILE *fp;
    printf("Enter Student Details:");
    printf("\n------------------\n");
```

```
for(i=0;i<3;i++)
  {
     printf("Enter Roll No:");
     scanf("%d",&x[i].rollno);
     printf("Enter Name:");
     scanf("%s",x[i].name);
     printf("Enter Age:");
     scanf("%f",&x[i].age);
  }
  printf("\n------------------");
  printf("\nWriting Data to File:");
  fp=fopen("C:\\Users\\kamesh\\Desktop\\MyCProgs\\fprnf.txt","w");
  for(i=0;i<3;i++)
  {
     fprintf(fp,"%d %s %f\n",x[i].rollno,x[i].name,x[i].age);
  }
  fclose(fp);
  printf("\nReading Data from a File:");
  fp=fopen("C:\\Users\\kamesh\\Desktop\\MyCProgs\\fprnf.txt","r");
  printf("\nRollNo Name Age");
  printf("\n---------------");
  for(i=0;i<3;i++)
  {
     fscanf(fp,"%d%s%f",&tmprno,tmpnm,&tmpage);
     printf("\n%d\t%s\t%.2f",tmprno,tmpnm,tmpage);
  }
  fclose(fp);
}
```

OUTPUT:

Enter Student Details:

Enter Roll No:1

Enter Name:AAA

Enter Age:18

Enter Roll No:2

Enter Name:BBB

Enter Age:20

Enter Roll No:3

Enter Name:CCC

Enter Age:19

Writing Data to File:
Reading Data from a File:
RollNo Name   Age
----
1   AAA   18.00
2   BBB   20.00
3   CCC   19.00

```c
/*Program: 13 Program to reduce spaces */
#include<stdio.h>
void main()
{
    FILE *fp, *fp1;
    int logic=0;
    char ch;
    fp=fopen("myfile.txt","r");
    fp1=fopen("newfile.txt","w");
    while((ch=getc(fp))!=EOF)
    {
        if(ch==' ')
        {
            if(logic==0)
            {
                logic=1;
                printf("%c",ch);
                putc(ch,fp1);
            }
        }
        else
        {
            printf("%c",ch);
            putc(ch,fp1);
            logic=0;
        }
    }
    fclose(fp);
    fclose(fp1);
}
```

OUTPUT:

Make a file called myfile.txt and enter the text, which has multiple spaces between words. Run the program, which will create another file newfile.txt, which will have same content as nyfile.txt but, multiple spaces between two words will be squeezed to single space.

/* Program:14 Program to occurrence of word in the file */

```
#include<stdio.h>
#include<string.h>
void main()
{
    FILE *fp;
    int i,cnt=0;
    char ch, str[20], tmpstr[20];
    fp=fopen("myfile.txt","r");
    printf("\nEnter any word:");
    scanf("%s",str);
    printf("\nWords found:");
    while((ch=getc(fp))!=EOF)
    {
        i=0;
        while(ch!=' ')
        {
            tmpstr[i]=ch;
            i++;
            ch=getc(fp);
            if(ch==EOF || ch=='\n')
                break;
        }
        tmpstr[i]='\0';
    //  printf("\n%s",tmpstr);
        if(strcmp(str,tmpstr)==0)
        {
            cnt++;
            strcpy(tmpstr,"");
        }
    }
    printf("\nOccurrence is: %d", cnt);
}
```

OUTPUT:

Create a text file called myfile.txt and type some content in it. Run the program, when it will prompt "Enter any word", type any word, which exists in the file. Program will count how many times, that word is written in the file, and display the occurrences of that word on the console screen.

PROGRAMS OF POINTERS

/*Program:15 Program to swap 2 variables using swap user defined function */
#include<stdio.h>
void main()
{
*   int x=5,y=7;*
*   printf("X is:%d \nY is:%d",x,y);*
*   swap(&x,&y);*
*   printf("\nAfter Swapping:");*
*   printf("\nX is:%d \nY is:%d",x,y);*
}
void swap(int *p,int *q)
{
*   int tmp;*
*   tmp=*p;*
*   *p=*q;*
*   *q=tmp;*
}
OUTPUT:
X is:5
Y is:7
After Swapping:
X is:7
Y is:5

/*Program:16 Design a function which will return the length of the given string */
#include<stdio.h>
void main()
{
    char x[10];
    int l;

```c
  printf("Enter Any String: ");
  scanf("%s",x);
  l=length(x);
  printf("Length of the Given String is:%d",l);
}
int length(char *p)
{
  int tmp=0;
  while(*p!='\0')
  {
    p++;
    tmp++;
  }
    return tmp;
}
```

OUTPUT:

Enter Any String: BAOU

Length of the Given String is:4

/*Program:16 Design a user defined function which will return reverse string of the given
string */

```c
#include<stdio.h>
void main()
{
  char x[10];
  printf("Enter Any String: ");
  scanf("%s",x);
  reverse(x);
  printf("Reverse String is: %s",x);
}
void reverse(char *s)
{
  char *d=s,tmp;
  while(*d!='\0')
  {
    d++;
  }
  d--;
```

```
    while(d>s)
    {
       tmp=*s;
       *s=*d;
       *d=tmp;
       s++;
       d--;
    }
}
```

OUTPUT:

Enter Any String: BAOU

Reverse String is: UOAB

Some useful websites for learning this block are as follows:

http://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp4_PointerReference.html

http://levelstuck.com/100-doors-2013-walkthrough-level-72-73-74-75-76-77-78-79-80

http://cboard.cprogramming.com/c-programming/6660-fwrite.html

http://cprogrammingcodes.blogspot.in/2012/01/palindrome-using-pointer.html

http://learnprogskills.blogspot.com/2010_01_01_archive.html

http://clanguagecorner.blogspot.in

http://santanuthecowboy.blogspot.in/2013/11/c-more-issue-in-input-and-output_5359.html

http://c-programmingbooks.blogspot.in/2011/11/detecting-errors-in-readingwriting-in-c.html

http://bestutors.blogspot.in/2008/05/opening-closing-data-file-data-file_11.html

http://lernc.blogspot.in/2009/12/opening-and-closing-of-data-file-in-c.html http://elite-world.biz/Member/MyCourses/internet/ch10-1.html

http://prajapatirajnikant.weebly.com/uploads/7/6/1/4/7614398/advc.pdf

https://gcc.gnu.org/onlinedocs/gcc-4.8.3/libstdc++/manual/manual/memory.html

http://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp4_PointerReference.ht ml

http://www.cprogrammingexpert.com/C/Tutorial/pointers.aspx

http://tksystem.in/tkdown/uploads/466287736_MSIT-102.pdf

http://gtu1.blogspot.in/2011/07/pointers-in-c.html http://www.mycplus.com/tutorials/c-programming-tutorials/pointers/2

http://learnprogskills.blogspot.com/2010_01_01_archive.html

http://ecomputernotes.com/what-is-c/structure-and-union/what-is-structures-and-unions

http://www.ustudy.in/node/8481 http://www.phy.pmf.unizg.hr/~matko/C21/ch11/ch11.htm

http://gtu1.blogspot.in/2011/07/structures-in-c-language.html

http://www.phy.pmf.unizg.hr/~matko/C21/ch11/ch11.htm

http://www.exforsys.com/tutorials/c-language/c-structures-and-unions.html

http://www.csi.ucd.ie/staff/jcarthy/home/2ndYearUnix/FilesinC%20lecture.doc

http://www.sanfoundry.com/c-program-create-file-store-information

http://www2.its.strath.ac.uk/courses/c/section3_12.html

Activities

Activity 1

- When is it valid to compare the values of two pointers? Explain with your own observation.

Activity 2

2 Create a file in C to store records of students, calculate their percentage and total marks and display each record.

Activity 3

- What is difference between structure and union?

Activity 4

- A file named DATA.dat contains integer. Read this file and copy all even numbers into file named EVEN.dat.

Activity 5

- Is it possible to omit the keyword „struct‟ while declaring structure variables? Justify.

Reference Books

1. The Art of C, H. Schildt

2. Born to Code in C, H. Schildt

3. C Programming, Ed. 2, Kerninghan and Ritchie

4. C Programming with Problem Solving, Jacqueline A Jones, Keith Harrow

5. C Programming, Balagurusamy

6. Let us C,YashwantKanetkar

7. Programming in C, S. Kochan

8. Programming in ANSI C, Agarwal

9. Turbo C/C++ - The Complete Reference, H. Schildt\

Block Summary

- Structure is a collection of one or more variables types grouped under a single name.
- The struct keyword is used to declare structures.
- A structure variable can be assigned values during declaration
- Structure members can be accessed using dot operator(.) also called structure member operator
- The total size of structure can be calculated by adding the size of all the data types used to form the structure.
- We cannot compare the structures using the standard comparison facilities like(= =,>,< etc);
- Arrays of structures are very powerful programming technique to have more than one instance of data.
- Structures can be nested, that is, structure templates can contain structures as members.
- Structure can be passed as an argument to a function.
- Like structures, Unions contain members whose individual data types may differ from one another. However, all the data members that compose a union share the same storage area.
- Unions are used to conserve memory.
- The struct keyword is used to declare structures.
- A union may be a member of a structure and a structure may be a member of a union.
- An individual union member can be accessed in the same manner as an individual structure members, using the operators (->) and . (dot).
- Pointer is variable which stores address of another variable.
- A pointer identifies or **references** a location in memory, and obtaining the value at the location it refers to is known as **dereferencing** the pointer.
- & is a unary operator called the address operator, which evaluates the address of its operand.
- is a unary operator called the indication operator, that operates only on a pointer variable?
- Pointer variables can point to numeric or character variables, arrays, functions or other pointer variables.
- A pointer variable can be initialized by assigning in the address of another variable, but you have to declare that another variable earlier in the program.

- Only addition and subtraction are the two arithmetic operations that can be performed on pointers.
- When handling arrays, instead of using array indexing, we can use pointers to access array elements. X[i] and *(X+i) represent same element.
- Pointers are used with strings. In C, constant character string always represents a pointer to that string.
- Pointers are often passed to a function as arguments. When pointer is passed to a function, the address of an actual data item is passed to the function. The contents of that address can be accessed easily, either within the called function or within the calling routine.
- An array name itself is a pointer to the array. I.e. the array name represents the address of the first element in the array. Therefore, when it is passed to a function, it is treated as pointer.
- It is possible to pass elements of an array, rather than an entire array, to a function.
- A function can also return a pointer to the calling portion of the program.

- We can have pointers pointing to structures also; those pointers are called structure pointers.
- A file is a place on the disk where a group of related data is stored,
- When working with a file, a temporary buffer area must be established to store information which is being transferred between the main memory and the data file.
- Operations on file are creating a file, opening a File, Reading or writing data from or to a File and closing file.
- fgetc() is used to read single character from a file that has been opened in read mode.
- fputc() is used to write single character into a file that has been opened in write mode.
- The function fgets() is used to read a string of specified length from a file opened in read mode.
- fputc() is used to write string into a file that has been opened in write mode.
- **The fscanf() and fprintf()** functions are used to deal with multiple data types. Using these functions, the reading or writing in formatted form from/to the files can be done.
- fread() and fwrite() functions are used to read the entire block from the data file or write the entire block to the file. Each block will generally represent a

complete data structure, such as a structure or an array.

- The standard library function ferror( ) returns any error that might have occurred during a read/write operation on a file.

Block Assignment

Short Answer Questions

1. Define structure. Define structure template for book which contain members like bookno, title, author and price?
2. Define union. What is the main use of union?
3. Define pointer. How array and pointer are related?
4. What is file? List the operations on file?
5. What is use of fgetc() and fputc() functions?

Long Answer Questions

6. What is difference between structure and union?
7. Explain how pointer can be passed as argument to function with example?
8. Explain different modes in which file can be opened?