BAOU Education for All

DR.BABASAHEB AMBEDKAR OPEN UNIVERSITY

BCA

BACHELOR OF COMPUTER APPLICATION

BCAR-103

Fundamentals of programming using C language

FUNDAMENTALS OF PROGRAMMING USING 'C' LANGUAGE

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY
AHMEDABAD

Editorial Panel

Author : Dr. Kamesh Raval
Assistant Professor,
Somlalit Institute of Computer Application,
Ahmedabad.

Reviewer : Dr. Himanshu Patel
Assistant Professor (Computer Science),
Dr. Babasaheb Ambedkar Open University,
Ahmedabad.

Language Editor : Dr. Jagdish Vinayakrao Anerao
Associate Professor,
Smt A.P. Patel Arts &
N.P. Patel Commerce College,
Ahmedabad

ISBN 978-81-943679-8-7

Edition : 2020

Acknowledgment

Every attempt has been made to trace the copyright holders of material reproduced in this book. Should an infringement have occurred, we apologize for the same and will be pleased to make necessary correction/amendment in future edition of this book.

ROLE OF SELF–INSTRUCTIONAL MATERIAL IN DISTANCE LEARNING

The need to plan effective instruction is imperative for a successful distance teaching repertoire. This is due to the fact that the instructional designer, the tutor, the author (s) and the student are often separated by distance and may never meet in person. This is an increasingly common scenario in distance education instruction. As much as possible, teaching by distance should stimulate the student's intellectual involvement and contain all the necessary learning instructional activities that are capable of guiding the student through the course objectives. Therefore, the course / self-instructional material is completely equipped with everything that the syllabus prescribes.

To ensure effective instruction, a number of instructional design ideas are used and these help students to acquire knowledge, intellectual skills, motor skills and necessary attitudinal changes. In this respect, students' assessment and course evaluation are incorporated in the text.

The nature of instructional activities used in distance education self-instructional materials depends on the domain of learning that they reinforce in the text, that is, the cognitive, psychomotor and affective. These are further interpreted in the acquisition of knowledge, intellectual skills and motor skills. Students may be encouraged to gain, apply and communicate (orally or in writing) the knowledge acquired. Intellectual-skills objectives may be met by designing instructions that make use of students' prior knowledge and experiences in the discourse as the foundation on which newly acquired knowledge is built.

The provision of exercises in the form of assignments, projects and tutorial feedback is necessary. Instructional activities that teach motor skills need to be graphically demonstrated and the correct practices provided during tutorials. Instructional activities for inculcating change in attitude and behaviour should create interest and demonstrate need and benefits gained by adopting the required change. Information on the adoption and procedures for practice of new attitudes may then be introduced.

Teaching and learning at a distance eliminate interactive communication cues, such as pauses, intonation and gestures, associated with the face-to-face method of teaching. This is

particularly so with the exclusive use of print media. Instructional activities built into the instructional repertoire provide this missing interaction between the student and the teacher. Therefore, the use of instructional activities to affect better distance teaching is not optional, but mandatory.

Our team of successful writers and authors has tried to reduce this.

Divide and to bring this Self-Instructional Material as the best teaching and communication tool. Instructional activities are varied in order to assess the different facets of the domains of learning.

Distance education teaching repertoire involves extensive use of self-instructional materials, be they print or otherwise. These materials are designed to achieve certain pre-determined learning outcomes, namely goals and objectives that are contained in an instructional plan. Since the teaching process is affected over a distance, there is need to ensure that students actively participate in their learning by performing specific tasks that help them to understand the relevant concepts. Therefore, a set of exercises is built into the teaching repertoire in order to link what students and tutors do in the framework of the course outline. These could be in the form of students' assignments, a research project or a science practical exercise. Examples of instructional activities in distance education are too numerous to list. Instructional activities, when used in this context, help to motivate students, guide and measure students' performance (continuous assessment)

PREFACE

We have put in lots of hard work to make this book as user-friendly as possible, but we have not sacrificed quality. Experts were involved in preparing the materials. However, concepts are explained in easy language for you. We have included many tables and examples for easy understanding.

We sincerely hope this book will help you in every way you expect.

All the best for your studies from our team!

FUNDAMENTALS OF PROGRAMMING USING 'C' LANGUAGE

Contents

BLOCK 3 : ARRAYS AND FUNCTIONS

Unit 9 ARRAYS

Introduction, Understanding arrays, One-Dimensional array, Operations on arrays, Two-Dimensional array

Unit 10 HANDLING STRINGS

Introduction, Understanding strings, Displaying strings in different formats, Standard functions of string handling, Table of strings

Unit 11 FUNCTIONS

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

Unit 12 MORE ABOUT FUNCTIONS

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

BLOCK 4 : STRUCTURES, POINTERS AND FILE HANDLING

Unit 13 STRUCTURES AND UNIONS

Introduction, Structures, Unions

Unit 14 POINTERS

Introduction, Understanding Pointers, Pointer Expressions, Pointers and Arrays, Pointers and Character Strings, Pointers and Functions, Pointers and Structures, Points on Pointers

Unit 15 FILE HANDLING

Introduction, Management of Files, Input/Output Operations on Files, Error Handling during I/O Operations

Unit 16 SOLVED PROGRAMS-III

Dr. Babasaheb Ambedkar
Open University Ahmedabad

BCAR-103/
DCAR-103

Fundamentals of Programming Using C Language

BLOCK 1 : BASICS OF C

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 Objectives :

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 2 : Understanding Constants, Data–Types & Variables

Unit 3 : Operators and Expressions

Unit 4 : Input–Output Operators

Unit 01

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 is considered to be one of the most powerful language, which beginners find easy to program. This language is also called the middle–level language, as it is closer to both the machine and the user. In this unit, we will discuss about the basics of C language which are used to develop a C program. After going through this unit, students will definitely be able to write and develop programs on their own.

1.2 Types of Programming Languages :

The programming languages are classified in to 3 main categories : [1] Machine Language [2] Assembly Language and [3] High–Level Language.

1.2.1 Machine Language :

Computer is an electronic device, made by using ICs (Integrated Circuits). These ICs are made by using few millions or billions of electronic devices such as capacitor or transistor. Transistors are two state devices made by silicon or germanium kind of semiconductor material, that can be charged or dis–charged. Based on its two states, charged (1) and dis–charged (0), it can understand only a language of the strings made by using two symbols that are 0 or 1. A language made by using strings of 0 or 1 is called Binary Language.

Binary language is easily understood by the machine, and to execute any program written in this language, machines do not have to use any kind of translator. Machine language or Binary language are also called as a Low–Level language as it directly understood by the machine without any kind of translation. Machine language is good for the programs, as it does not need any kind of translation, and machine can execute it faster. The main drawback of the machine language is, it is difficult to memorize all command of the programming language in the strings of binary that is (0 and 1) like 00100010, 11010011 or 10101101 etc. So, it is very difficult for the programmer to write the programs in the machine language.

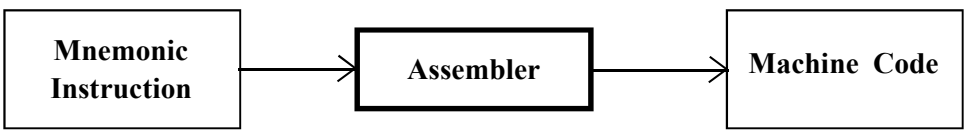
❑ **Check Your Progress – 1 :**

1. Machine Language is also called _____ Language.
   [A] Binary [B] High–Level [C] Mark–up [D] None of these
2. A language which use only 2 symbols that is 0 and 1 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. A language does not require any kind of translator is _____.
   [A] Machine [B] C–Language [C] Java [D] SQL

1.2.2 Assembly Language :

Assembly language is another language, which allows programmer to write the programming instructions in the mnemonic codes rather than string of binaries. In this programming language, programmer can write the instructions in English like language, for example to add two numbers 'ADD' is the instruction, to subtract two numbers 'SUB' is the instruction and so on.

In this language programmer can write the instruction like MOV, ADD, SUB, MUL and so on, but machine cannot understand these instructions. So, before executing these instructions (program), they have to translate it in to the machine language with help of a software called Assembler.



1.2.3 High–Level Language :

High–Level languages are programmer friendly. Programmer can write the instruction in English like language. Most high–level languages use complier or interpreter to translate the source code in the machine language, before executing the program. Compilers allow programmer to write a single instruction, which will be translated into 3 or 4 instruction in the low–level language. This reduces programming efforts. High–level languages are easy to understand and readable.

❑ **Check Your Progress – 2 :**

1. Compare to machine language, _____ language use mnemonic symbols instead of string of binaries.

   [A] Binary [B] Machine [C] Assembly [D] Both A and B

2. _____ translator is required in the assembly language.

   [A] Compiler [B] Interpreter

   [C] Assembler [D] None of the Above

3. C–Language is _____ level language.

   [A] Low [B] High [C] 4th GL [D] Machine

4. High level language is _____ for programmer. (Difficult / Easier)

1.2.4 Assembler, Compiler and Interpreter :

Assembler, Compiler and Interpreter all are translators which translate high–level source code written by the programmer into the low–level (machine language).

1. **Assembler :** Assembler is translator used to translate mnemonic code written in the assembly language in to the machine language. Assembler is a simple translator which translate one mnemonic instruction in to one machine code instruction. Main benefit of using assembler and writing code in the assembly language is, programmer do not have to memorise binary string code of each instruction of the program. Programmer needs to write simple instruction like 'MOV', 'ADD', 'SUB', 'MUL', or 'DIV', instead of their binary strings.

2. **Compiler :** Compiler is a translator, used in high–level language like C, C++, Java and so on. Compiler reads entire program and translate into the machine level code. Once whole program translated in the machine language, exaction of the program gets started. Because before execution all instructions of the program are translated, and machine of each instruction is available at execution time, execution of the program will be faster. Infect, it needs some time to compile entire program. Compiler can translate one line of high–level language into one or more lines of machine code. So, it will reduce programmer's effort. Complier makes high–level code shorter and more readable. In the programming languages, which uses compiler, execution of the program is started if and only if program is error free. If any line has error, compiler will generate compilation error message and execution of the program will be abounded. Execution of the program will be started, when entire program gets translated into the machine code. Compilation process will take some compilation time, but once the execution is started, it will

be faster as all instructions of the program are ready and translated in the machine language.

3. **Interpreter :** Interpreter performs same as compiler (translating source code written in the high–level language into the machine code). But it fetches the first instruction of the program, it translate only one instruction in the machine code and execute. After execution of the first instruction it fetches the second instruction, it will translate it into the machine code and execute it. In short, Interpreter executes a program line by line. Because before executing any instruction it has to translated into machine code, execution of the program will become slower. Infect, it do not take any time for compilation. So, if the program is having syntax error at line 15, in the case of interpreter, it will execute first 14 lines and shows error at line 15.

❑ **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 :

C–Programming language is developed by Dennis Ritchie in 1972 at Bell Laboratories of USA. Some basic idea to develop C–Language is taken from two other programming languages that are B–Language (developed by Ken Thomson in 1970 at Bell Laboratories) and 'Basic Combined Programming Language (BPCL)' (developed by Martin Richards in 1967.

1.3.1 Structure of C–Program :

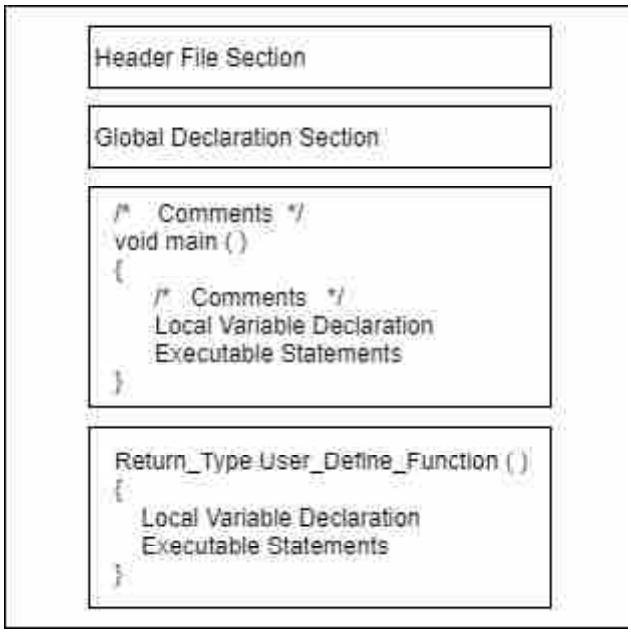
C–Program is divided in to several parts, which are described as given below :

1. **Header File Section :** As we have discussed earlier, C–Language supports functions. C–Language has 30 or more header files, which include more

than 145 library functions. This section of the C–Program allows 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. Header files are stored with '.h' extension. For example, if you have included header file '#include <stdio.h>' (Standard Input Output Header File) then you can use 'printf ()' and 'scanf ()' functions. If you have included header file '#include<conio.h>' (Console Input Output Header File) then you can use 'clrscr () and 'getch ()' functions.

2. **Global Declaration Section :** Variables declared in this section are called global variables. Global variables are created in the memory when program starts its execution and remain into the memory unless the program is not terminated. Global variables can be accessible anywhere in the program.

3. **Main Function :** In C–Programming language, N number of functions you can create, but the function from which you can start execution of the program should have function name 'main ()'. System will always start the execution of the program from the main function. Because of the main () function does not return any value, we have mentioned 'void' return type for the main function. '{'indicate start of the main function and '}' is used to close main () function.



4. **Local Variables :** Variable declared inside the function is called local variable. The life of the local variable is limited. When the function is called and it is loaded into the memory, local variables are created and after the execution when the function is destroyed all the local variables will be deleted from the memory. The scope of the local variable is limited to that function only. You cannot access local variable of one function into other function. Unlike global variable (declared outside of function) local variable cannot be accessible throughout the program.

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

6. **Comments :** Comments are the statements that can be written anywhere in the C–Program. Comment statements are simply skipped by the compiler

and it will not be translated into the machine language, means those statements are not be executed by the system. Comments are the statements, which are generally in the program to increase the readability of the program. There are two types of comments: (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 you are not be able to locate proper library function for particular task, you can define your own function in the C–Language. Such functions are known as user defined function (UDF). You can create as many as UDFs into a program.

❑ **Check Your Progress – 4 :**

1. stdio.h stands for _____.

   [A] Standard Input Output

   [B] String Terminating Operations Input Output

   [C] Store Input Output

   [D] None of the above

2. Function printf() 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 to make single line comment.

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

5. _____ variables are accessible throughout entire program.

   [A] local [B] global

   [C] register [D] None of the above

1.3.2 Compiling and Executing a C–Program :

Now to execute a program, you need to install any of the software like Turbo C++, Borland C++ or you can also use Code Blocks. Open the software you are using for your C–Language program development and write the following code.

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

After writing the program you have to save the program with .c or .cpp extension. Let we save the program written above with name 'hello.c'. Now

we know that the C–Language is a high–level language and we are writing instructions in high–level language like English language. Machine cannot understand this language, so we need to compile (translate the high–level source code into the machine language code) 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 the source code or the program has any error, then compiler will show error message. You need to rectify or correct the error and again you have to compile the program. Once the program compiled successfully (without any 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 code blocks you need to press F9). This will run your program in the Console output 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. A C–program can have multiple functions, but in each program must have 'main ()' function. System always starts the execution of the program by 'main ()' function.
2. Each statement of the C–Program ends with semicolon (;). However conditional statements, like if, for and while loop, and in the function definition, statements do not end with semicolon.
3. Generally, in the C–program all the statements should be written in the lower case. However uppercase strings can be used for symbolic constants.
4. The opening and closing braces should be balanced. i.e. number of opening braces and number of closing braces are same.
5. It is not necessary that each line has single statement in C–program. You can write more than one C–program statements in the same line. 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, which can be used to develop application software and also system software. Popular operating system UNIX is developed in C–Language. The compiler of C–Language is also developed in the same language.
2. C–Language has influential data definition. It supports characters, integers, floating point numbers and strings.
3. C–Language allows programmer to add assembly code.
4. C–Language is highly portable. We can compile and execute the same C–Program written in the windows operating, into other operating systems like UNIX, LINUX or other.

5. C has powerful set of operators.
6. C–Language has 32 keywords, 30 header files and 145 built–in library functions.
7. C is a structure–oriented programming language, allows programmer to develop custom (user–defined) data types.
8. C–Language supports strings in the form of character–based Arrays.
9. C–Language also supports powerful graphics; hence programmer can use it to develop games or applications which has graphical user interface.
10. Programmer can also write a C–Language program to control various 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 that C–Programming Language is general purpose programming language which 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 : It is a 'Console Input Output Header File' contains functions like clrscr(), getch() etc.
2. stdio.h: It is a 'Standard Input Output Header File' 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

❑ **Check Your Progress 4 :**

1. [A] Standard Input Output
2. [A] stdio.h
3. [A] local
4. [B] //
5. [G] Global

❑ **Check Your Progress 5 :**

1. [A] ; (Semi–Colon)
2. [B] 30
3. [C] 32
4. [D] Unix

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 6, 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" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw–Hill Education.

Unit 02

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 :

C is considered to be one of the most powerful language, which beginners find easy to program. This language is also called the middle–level language, as

it is closer to both the machine and the user. In this unit, we will discuss about the basics of C language which are used to develop a C program. After going through this unit, students will definitely be able to write and develop programs on their own.

2.2 Constants :

Constants are those quantities whose value may not change during the execution of a C program. C Supports Integer, Character, String and Floating–point constants. Integer constants represent numbers, floating constants represent decimal numbers combined together they are denoted as numeric type constants. Numeric constants should follow the rules :

- Commas and blank spaces cannot be included within the constant.
- The constant can be preceded by a minus (–) sign.
- The value of constant cannot exceed specified minimum and maximum bounds, which varies from a C compiler to compiler.

2.2.1 Integer Constants :

An integer constant contains digits. There are three types of integers : decimal, octal and hexadecimal.

- A decimal integer constant can consist of any combination of digits from 0 to 9, they can be positive or negative with + or – For example, 156, –562, 3584, +72, 0.
- A hexadecimal integer constant must begin with either 0x or 0X and can be combination of digits between 0 to 9 and a to f (lower or uppercase) which represent the numbers 10 to 15. For example, 0x2, 0xbc5, 0x, 0xf.
- The largest integer value that can be stored, is machine dependent. It is 32767 on 16–bit machines, and 2147483647 on 32–bit machines. You can also store larger integer constants on these machines by appending qualifiers such as U, L and UL to the constants. For example, 762348U (Unsigned Integer), 982653762UL (Unsigned Long Integer), 569289L (Long Integer).

2.2.2 Floating Point Constants :

These constants are base 10 numbers that contain either a decimal point or an exponent (or both). Some of the valid floating–point constants are :

8.5, 0.6, 1.4526E + 5, 425147e15

If an exponent is present, it shifts the location of the decimal point. If the exponent is positive, it shifts the decimal point to the right and if the exponent is negative, it shifts it to the left. They have a greater range than integer constants. The magnitude of a floating–point constant ranges from 3.4E–38 to 3.4E+38. These constants are normally represented as double precision quantities. Each floating–point constant typically occupies 2 words (8 bytes) of memory.

2.2.3 Character Constants :

They are single characters, enclosed in single quotation marks. For example, 'B', 'a', 'O', 'u' etc.

2.2.4 String Constants :

They consist of consecutive characters enclosed in double quotation marks. For example,

"Hello", "Welcome", "Computers", "Object Oriented"

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

❑ **Check Your Progress – 1 :**

1. From the following _____ is not a type of constant.

   [A] Integer [B] String [C] Image [D] Floating–point

2. The largest value of Integer constant in 16–Bit machine is _____.

   [A] 32767 [B] 65536 [C] 255 [D] 2147483647

3. From the given which is not TRUE for constant ?

   [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 :

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.

❖ **Variable Naming Convention :**

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

- The variable name can be of letters, or combination of letters and digits or special character underscore character '_'.
- The first letter of variable must be a character or underscore and rest can be digits.
- Variable names are case sensitive e.g. count and Count refer to two different variables.
- Reserved words cannot be used as variable names. e.g. int, float, char, if, else, include, for, while, switch cannot be name of variables.

2.3.1 Data Types in C :

The C language has a very rich set of data types. Now, we will go through some of the basic data types available in C–Language which will 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 :**

Character data type is used in the C–Language to store or to access a character. To declare the character type of variable we will use keyword "char". A variable declared with "char" datatype can store one character, and variable will occupy 1Byte of storage memory space. For example :

char ch;

ch= 'A';

printf ("%c", ch);

In the above statement, we have declared a variable 'ch' of type "char". Variable "ch" will reserve 1 Byte (=8 Bits) of space in the main memory. In the second statement we are initializing (assigning a value) 'A' to character type variable "ch". In the third statement we are printing the value of ch variable, so we have called a printf function. In the printf statement first we have mention, format specifier that is "%c" (see format required column in the above table for character variable) and then we have mentioned our character variable ch which separated by comma. This printf statement will print the character value stored in the ch variable on the console that is 'A' on the screen.

❖ **Integer :**

Integer variables are used in the C–Language to store numbers without decimal points. To declare a variable of type integer, we will use keyword "int". Any variable declared with type "int" will occupy 2 Bytes (=16 Bits) of space in the memory. For example :

int number;

number=45;

printf ("%d", number);

In the above code we have declared a variable number of type int. We have stored 45 in it, as we have initialized number variable with value 45 in the second

line. Finally, in the third line of code we have printed the value of number variable with the printf statement using format string "%d". To print any variable of type int you can used either "%d" (decimal) or "%i" (integer) format string. You can also print the number in octal with "%o" and hexa–decimal format with "%x" format. Consider the following program :

```
#include<stdio.h>
void main()
{
    int number;
    number=165;
    printf("Number in Decimal format is :%d", number);
    printf("\nNumber in Octal format is :%o", number);
    printf("\nNumber in Hexa–Decimal format is : %x", number);
}
```

❖ **OUTPUT :**

```
Number in Decimal format is :165
Number in Octal format is :245
Number in Hexa–Decimal format is : a5
```

Integer variable can store both positive and negative, and also 0. For example, in the above example, we have initialized number variable with 45 by writing instruction number=45. You can, also store negative number or 0 in the variable by writing statement, number = –45 or number = 0.

Variable of type "int" will occupy 2 Bytes (= 16 Bits) of memory space. From this 16 Bits, 1 bit is used to represent sign. That means to store positive number sign bit will 0 and if sign bit is 1 means that number is negative number. Rest of the 15 bits are used to represent a number. Therefore, maximum number can represent is 215 = 32768. So, we can store –32768 as a smallest number and 32767 as the largest number in the variable of type int. We are using one permutation to represent a number 0 that the reason we cannot represent 32767 (positive) number. So, the range of a variable of type int is : –32768 to 32767.

❖ **Unsigned Integer :**

In the unsigned integer variable, all 16 bits are used to represent a number. There is no provision for sign bit. So, unsigned variable cannot represent negative numbers. Here all 16 bits are used to represent a number so it can represent 216 = 65536 numbers. If we start from 0 as a first number, it can go up to 65535. So, the range of the unsigned variable is 0 to 65535. It can be printed or scanned using format string "%u". Consider the following example :

```
unsigned int num;
num=65000;
printf ("%u", num);
```

❖ **Float :**

To access and to store any real value, float data type is used. C's float data type occupies 4 bytes in the memory.

To declare any float variable :

float per;

float total;

To assign or to store some real value :

per=78.20;

total=123.45

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

❖ **Long :**

To access and to store any integer value long data type is used. C's long data type occupies 4 bytes in the memory.

To declare any long variable :

long account1;

longmobileno;

To assign or to store some long value :

account1=123456;

mobileno=9898998765;

❑ **Check Your Progress – 2 :**

1. The range of Unsigned int variable is _____.

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

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

2. Maximum value can be stored in short integer variable is _____.

   [A] 32767 [B] 65535 [C] 255 [D] 250532247

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

   [A] 1 Bytes [B] 2 Bytes [C] 4 Bits [D] 4 Bytes

2.4 Character Set :

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

The C Character set is given below

- Uppercase Letters (A–Z)
- Lowercase Letters (a–z)
- Digits (0–9)
- Special Characters (#, }, < , / etc.)

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

2.5 C Tokens :

Each word used in a program is called token. There are different types of tokens, which are used while writing program. These tokens are discussed in detail in this unit.

2.5.1 Keywords :

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 :

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 various elements of a program such as variables, functions and arrays. They consist of digits and letters which can be in order but the first character should be a letter, using both uppercase and lowercase letters, but should have different meanings. For example, Mam and mam are not same. The underscore character can also be included and is treated as a letter. Given below are some valid identifiers :

A05, Computer_05, interest_rate, WELCOME

❑ **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 :

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

The syntax for the same can be written as

<data type> <variable name>;

An array variable must be followed by a pair of square brackets with a positive integer specifying the size of the array. For example,

```
int x, y;
float w, l;
int a[10];
char k[5];
```

Thus, x and y are integer variables, w and l are float variables, a is an integer array whose size is 10 and k is a character array with a size of 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 x=5;
char name= 'A';
float z= 5.36;
```

From the above declaration it is clear that variable x is of integer type and initialized with value 5, name is character typed variable initialized with 'A' and z is of type float, initialized with value 5.36.

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

```
char name[]= "Computer";
```

in the above statement we are declaring name as an array of type character, which will store 9 characters 'C', 'o', 'm', 'p', 'u', 't', 'e', 'r', '\0'. The word "Computer" needs 8 characters and at the end computer will store '\0' (null) character which will indicate end of the string.

The above declaration can also be written as :

```
char name[9]= "Computer";
```

Here variable 'name' is an array (collection of same type of data). The size of the array must be specified correctly at the time of declaration. This is so because if the size is smaller, the characters at the end of the string will be lost and if the size is too large then the extra array elements may be assigned a space or may be filled with meaningless characters.

❑ **Check Your Progress – 4 :**

1. From the given below _____ is not a valid variable name.
   [A] abc123 [B] abc+123
   [C] abc_123 [D] All are valid variable names
2. In the declaration, variable names are followed by _____.
   [A] switch [B] continue [C] datatype [D] break
3. From the given below _____ identify valid variable name.
   [A] 123abc [B] int [C] Abc_123 [D] a+b

2.7 Defining Symbolic Constants :

Symbolic constants are also termed as sequence of characters. The characters may represent a numeric constant, a character constant or a string constant. Symbolic constant allows a name to appear in place of a numeric or character or string constant. Each occurrence of a symbolic constant is replaced

by its corresponding character sequence at the time of compilation. They are defined in the beginning of a program. For example :

 # define MAX 10

Where MAX represents a symbolic name and 10 represents the value associated with the symbolic name. Since it is a symbolic constant, definition of the constant, 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.

❑ **Check Your Progress – 5 :**

1. Usually _____ are declared using uppercase letters.

   [A] Constants [B] Varaibles [C] Data Types [D] Keywords

2. In C–Language strings ends with _____.

   [A] $ [B] @ [C] /0 [D] \0

3. Symbolic constants are declared with _____ pre–processive directive.

   [A] #define [B] #include [C] #declare [D] #constant

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

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

Variables : 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" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw–Hill Education.

Understanding Constants, Data types and Variables.

Unit
03

OPERATORS AND EXPRESSIONS

UNIT STRUCTURE

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 data (operands). For example, if we write "a + b" then "a + b" is known as expression, "+" is an operator and "a" and "b" are data (operands).

Operators are used to form expressions by concatenating constants, variables, array elements as studied in previous section. We will study arithmetic operators, unary operators, relational and logical operators, assignment operators and the conditional operators.

Operands are data items on which operators perform arithmetic or logical computations. Binary operators require two operands while unary operators require only one operand. A few operators permit only a single variable as an operand. In this unit, we will discuss about various types of operators and evaluation of expressions.

3.2 Operators and Expressions :

An operator is used to perform operations on operands. There are different operators such as :

(1) Arithmetic (2) Relational

(3) Logical (4) Assignment

(5) Increment/Decrement (6) Conditional

(7) Bitwise (8) Special

3.2.1 Arithmetic Operator :

There are five basic arithmetic operators used in 'C'.

1. For Addition '+' operator
2. For Subtraction '–' operator
3. For Multiplication '*' operator
4. For Division '/' operator, and
5. For Mod (Modulo or Remainder) '%' operator.

Consider a case, where we have two operands A=15 and B=3, If we apply the arithmetic operators as discussed above, we can get the following results.

Expressions	Value	Explanation
A+B	15	A+B = 15+3 = 18
A–B	12	A–B = 15–3 = 12
A*B	45	A*B = 15*3 = 45
A/B	5	A/B = 15/3 =5
A%B	0	A%B = 15%3 =0
		As 15 is divisible by 3 and we get divide 15 by 3 we get 5 as division and 0 as remainder.

3.2.2 Relational Operator :

Relational operators are divided into 6 types such as

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

The associativity of these operators is left–to–right. Relational operators are also known as comparison operators. Relational operators are always producing logical values, such as (TRUE which can be interpret as 1 or FALSE which can be interpreted as 0).

For example, expression 5 > 3 is TRUE, it is interpreted as 1, and expression 9>27 is FALSE will be interpreted as 0.

Suppose, that i, j and k are integer variables whose values are 1, 2 and 3 respectively. Some logical expressions involving these variables are shown below :

Expression	Result	Interpretation Value
i<j	True	1
k<j	False	0
(i+j)<=k	True	1
j<k	False	0
(i+j)>k	False	1
(i+k)>j	True	0
(i+j)>=k	True	1
j>=(i+k)	False	0
j==3	False	0
(i+j) == k	True	1
j!=3	True	1
(i+j)!=k	False	0

3.2.3 Logical Operator :

There are three logical operators are there. That are :

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

These operators are referred to as logical AND, logical OR and logical NOT respectively. Like relational operators, logical operators are also producing Boolean values (Values in terms of TRUE/ FALSE or 1/0)

The following table shows the results of the combined expressions depending on the value of the expression :

Expression		Result			
Expr1	Expr2	Expr1 && Expr2	Expr1 || Expr2	! (Expr1)	! (Expr2)
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 (All values except 0) is considered as 1. For example, if there are 3 variables x=–5, y=0 and z are there, and if we write z=x&&y, then we get z=0 and if we write z=x||y, we get z=1. Because x is non–zero, so it will be considered as 1. Now, in && operation x&&y = 1&&0=0. Similarly, in x||y=1||0=1. Remember we will get x && y = 1, if and only of both x and y are non–zero, and we will get x||y=0, if x and y both are 0.

3.2.4 Assignment Operator :

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

Syntax : Identifier = Expression

i= i+1 It can be written as i+=1 and i=i–1 as i–=1 and so on. For example, x=5 statement indicates, value 5 is assigned to variable x. Now, if we write an expression "x+=4;", that means x = x + 4. If x is initialized with 5 then 5 + 4 =9. And the newer value stored in the variable x is 9.

3.2.5 Increment/Decrement Operator :

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

The operand has to be a variable and not a constant. Thus, the expression a1++ is valid, whereas 61++ is invalid. Operators increments or decrements the value by 1. Expression a1++ increments the value of a1 variable by 1 and the expression a1– – decrements it by 1.

If prefix before it's termed as prefix and if prefixed after it is termed as postfix.

E.g. a++; and ++a; gives same result, it increments value by 1.

However, prefix and postfix operators have different effects when used in association with some other operators in a 'C' statement. For example, if we assume the value of a variable to be 5 and then in the statement b=++a, variable a will be incremented by 1 first (a=6) and the value of a is assigned to variable b. As a result, we will get a=6 and b=6).

On the other hand, in the case of execution of the statement b = a++; the value of variable a is assigned to b first. Therefore, b=5 and then the value of variable a is incremented by 1. So, a will be 6. As a final result, a=6 and b=5. So,

b=++a; means a=a+1 and then b=a (Pre–Increment),

b=a++; means b=a and then a=a+1 (Post–Increment).

[Note : First precedence : R–>L associativity)]

3.2.6 Conditional Operator :

Conditional operators are termed as Ternary Operators (? :). An expression that makes the use of the conditional operator is called a conditional expression.

Syntax :

(Condition) ? True : False

Exp1 ? Exp2 : Exp3

Here, if a condition is evaluated as true then the value of exp2 will return and if condition is evaluated as false then exp3 will return.

For example,

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

If x variable's value is less than 0 then condition is evaluated as true and T1 will be assigned with 0 else the condition is false and 100 will be assigned to variable T1.

For example, to find the greater value from variable x and y we can write :

printf("%d", (x>y) ? x :y);

Similarly, to find greatest value from given three variables x, y and z we can write :

printf("%d", (x>y) ? (x>z) ? x : z : (y>z) ? y : z);

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 logical bitwise operators : bitwise and (&), bitwise exclusive or (^) and bitwise or (|). Each of these operators requires two integer type operands. The operations are carried out independently on each pair of corresponding bits within the two operands. Thus, the least significant bits (i.e. the rightmost bits) within the two operands will be compared and then the next bit and so on.

B1	B2	B1&B2	B1^B2	B1|B2
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=45 & 78 then we will get the value 12 as follows :

Binary of 45 : 0000 0000 0010 1101

Binary of 78 : 0000 0000 0100 1110

Binary of 12 : 0000 0000 0000 1100

Similarly, if we write x =45 | 78 then we get value x=111 as follows :

Binary of 45 : 0000 0000 0010 1101

Binary of 78 : 0000 0000 0100 1110

Binary of 111 : 0000 0000 0110 1111

❖ **Bitwise Shift operators :**

The two bitwise shift operators are shift left (<<) and shift right (>>). Each operator requires two operands. The first is an integer–type operand that represents the bit pattern to be shifted. The second is an unsigned integer that indicates the number of displacements (i.e. whether the bits in the first operand will be shifted by 1–bit position, 2–bit position, 3–bit positions and so on). This value cannot exceed the number of bits associated with the word size of the first operand. The left–shift operator shifts the bits of a binary number to the left by the number of positions indicated by the second operand. The left most bit positions that become vacant will be tilled. For Example, if variable A=28,087.

A = 0110 1101 10110111

After execution of X=A<<6 the value of the variable X is :

X = 0110 1101 1100 0000 = 28,096

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

A = 0110 1101 1011 0111

After execution of X=A>>6 the value of variable X is :

X = 0000 0001 1011 0110 = 438

We, can say that from the variable A, 6 bits from the right side (110111) will be removed and 6 bits (000000) will be added at the left–hand side, which will be stored in the variable X.

❑ **Check Your Progress – 1 :**

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

   [A] 3 [B] 5 [C] 4 [D] 2

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

   [A] 12 [B] 45 [C] 78 [D] 111

3. for two integer x and y, what statement printf ("d", (x>y) ?y :x); 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 size of operator, a compile time operator, it returns number of bytes the operand occupies. The operand may be a variable, a constant or a data type. sizeof() is used to determine the length of array and structures and to allocate memory space dynamically during execution of a program. For Example,

1. sizeof(sum);
2. N1=sizeof(longint);
3. K1=sizeof(253L);

3.3.2 The Comma Operator :

For conjunction in for statement we use comma operator. Using comma operator, we can use two different expressions simultaneously where only one expression would ordinarily be used. For example,

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

Where expression i and j are the two expressions, separated by the comma operator. These two expressions would typically initialize two separate indices that would be used simultaneously within for loop.

For example, one count forward i.e. n1 and other counts backward i.e. m1

for(n1=1,m1=10; n1<=m1; m1++, n1++)

❑ **Check Your Progress – 2 :**

1. if x is a long integer variable, then what will be output of printf("%d",sizeof(x));

   [A] 2 [B] 1 [C] 4 [D] 8

2. if x is a unsigned integer variable, then what will be output of printf("%d",sizeof(x));

   [A] 1 [B] 2 [C] 4 [D] 8

3. _____ operator is used for conjunction of two statements into one.

   [A] , comma [B] % modulo

   [C] . dot operator [D] None of the above

3.4 Arithmetic Expressions :

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

For example,

x1+y1 //Uses addition operator

a1=b1 //Uses assignment operator

p1=q1+r1 //Uses both addition and assignment operator

3.4.1 Evaluation of Expressions :

An assignment statement is used to evaluate arithmetical expressions represented as : ***Variable=expression;***

In the above statement, variable is any valid C variable name. 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 be assigned values before evaluating it. For example,

x1= a1+b1*c1;

y1=b1/c1–d1;

z1=a1+b1/c1*d1;

Now if a parenthesis is given, then the evaluation of the expressions will be done as given below :

Suppose, a1=18, b1=10, c1=4, then the expression,

x1=a1–b1/(2+c1)*(2–1);

will be evaluated as :

Expressions in a parenthesis get 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);

3.4.2 Precedence of Arithmetic Expressions :

An arithmetic expression without a parenthesis can be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C :

High Priority * , / and %

Low Priority + and –

❖ **Rules for evaluating an expression :**

- If in an expression, a parenthesised subexpression is given, it is evaluated from left to right.
- If nested parentheses are given, then the evaluation begins with the innermost subexpression.
- To determine the order of application of operators for evaluating subexpressions the precedence rule is applied.
- The arithmetic expressions are evaluated from left to right using the rules of precedence.
- The associability rule is applied when two or more operators of the same precedence level appear in the subexpression.
- When a parenthesis is used, the expressions within the parenthesis assume the highest priority.

3.4.3 Some Computational Problems :

Precedence rules is used to evaluate an expression without parenthesis it is evaluated from left to right. Two distinct priority levels of arithmetic operators in C :

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

❖ **Rules for evaluating an expression :**

- Parenthesised sub expression is evaluated from Left to Right.
- The evaluation begins with the innermost sub expression if it is nested.

Precedence rule is applied to determine the order of application of operators for evaluating.

- The arithmetic expressions are evaluated from left to right using the rules of precedence.
- The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.

3.4.4 Types of conversions in expressions :

You can mix the types of values in your arithmetic expressions. Char types will be treated as int. Otherwise here types of different sizes are involved, the result will usually be of a larger size, so a float and a double would produce a double result. Where an integer and real types of conversions meet, 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 variable is too small to hold the value. It will be corrupted.
- The variable is an integer type and is being assigned a real value. The value is rounded down. 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 use method called casting which temporarily disguises a value as a different type.

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

```
int i1 = 256;
int root1;
root1 = sqrt( (double) i1);
```

The cast is made by putting the bracketed name of the required type just before the value; (double) in this example. The result of sqrt((double) i1); is also a double, but this is automatically converted to an int on assignment to root1.

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

- Float operands 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 x=5,y=2; then what will be output of printf("%f",x/y);

   [A] 2.000000 [B] 1.5 [C] 2.500000 [D] 2.5

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

   [A] 2.000000 [B] 1.5 [C] 2.500000 [D] 2.5

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

   [A] 1.000000 [B] 1.5 [C] 1.500000 [D] 1.500

3.5 Operator Precedence and Associativity :

Precedence means priority given to the operator for evaluation or execution when there are more than single operators in a single expression or statement. If there are operators of 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=2+3*5–(7–2)+9/3 statement will be evaluated in the following manner

2+3*5–(7–2)+9/3 [First preference will be given to bracket]

2+3*5–5+9/3 [Now, preference will be given to multiplication and division]

1+15–5+3 [Now, + and – has equal priority so as per associative law L to R]

16–5+3

11+3

14

Computer always evaluate expression by looking the priority of the operator. Always those operators will be resolved first which has highest precedence. Precedence means priority of the operator. So, the study of the operator is important. You always have to write the expressions in such a way that it should be evaluated by the computer system correctly. In the following table we have written all the operators, as per their priority in the computer system.

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. From the given operator chose the operator having highest precedence.

   [A] * [B] > [C] & [D] %

2. From the given operator choose 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 x, double y)– Computes x raised to the power y

double sqrt(double x)– Computes the square root of x

double sin(double x)– Computes sine of angle in radians

double sinh(double x)– Computes the hyperbolic sine of x

double tan(double x)– Computes tangent of angle in radians

double tanh(double x)– Computes the hyperbolic tangent of x

double log10(double x)– Computes log to the base 10 of x

double modf(double x, double *intptr)– Breaks x into fractional and integer parts

double log(double x)– Computes log(x)

double exp(double x)– Computes exponential of x

❑ **Check Your Progress – 5 :**

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

   [A] 625 [B] 25 [C] 0 [D] 5

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

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

3. If x=pow(2,3); then output of the statement printf("%d",x); is _____.

   [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

Operators and Expressions

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 :

- Born to Code in C, H. Schildt
- C Programming, Ed. 2, Kerninghan and Ritchie
- C Programming with Problem Solving, Jacqueline A Jones, Keith Harrow

Unit 04

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 :

In a C language program, we provide input values to the program and output the data produced by the program to a standard output device. Values can be assigned to variable using assignment statements such as x1=5 a1= 0;

The basic operation is to read data from the input device e.g. keyboard and output is displayed on the console.

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. There exist several functions in 'C' language that can carry out input and output operations. The functions are stored in standard Input/Output Library. In this unit, we will discuss about the various input/ output operations.

4.2 Managing Input/Output Operations :

The input and output operations in C can be managed by using several functions. These functions are used to accept a single character or a string of characters also we can display a single character or a string of characters.

4.2.1 Reading a Character :

Using function getchar() we can enter a single character using standard input device (typically a keyboard).

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

character variable=getchar( );

E.g. char ch;

ch=getchar( );

Input Output Operations

The first statement declares that ch is a character type variable. The second statement causes a single character to be entered from the standard input device and stores in ch. If an end–of file condition is encountered when reading character with the getchar() function, automatically returns EOF. The getchar function can also be used to read multicharacter strings, e.g. multiple loops.

4.2.2 Writing a Character :

A single character can be displayed (i.e. written out of the computer) using the C library function putchar(). It transmits a single character to a standard output device. The character being transmitted will normally be represented as a character–type variable. It must be expressed 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 (character variable);

A 'C' program contains the following statements :

char ch= 'a';

putchar(ch);

The first statement declares that ch is a character type variable. The second statement causes the current value of ch to be transmitted to the standard output device where it will be displayed. The putchar() function can be used to output a one–dimensional character type array. Each character can then be written separately within a loop.

❑ Check Your Progress – 1 :

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

   [A] putchar() [B] getchar() [C] scanf [D] printf()

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

   [A] putchar() [B] getchar() [C] scanf [D] printf()

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

   [A] putchar() [B] getchar() [C] scanf [D] printf()

4.3 Formatted Input

The scanf( ) function is used to read data from the keyboard and to store that data in the variables. The general syntax for scanf( ) function is as follows :

scanf("Format String",&variable);

Here, format string is used to define which type of data it is taking as an input, this format string can be %c for character, %d for integer variable and %f for float variable. Whereas variable is the name of memory location or name of the variable and the (&) sign is an operator that tells the compiler the address of the variable where we want to store the value. One can take multiple inputs of variables with a single scanf() function but it is recommended that there should be one variable input with one scanf() function.

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

For Integer Variable :

```
int rollno1;
printf("Enter Roll No=");
scanf("%d", &rollno1);
```

Here, in scanf() function %d is a format string for the integer variable and (&) rollno will give the address of variable rollno1, to store the value at variable rollno1 location.

For Float Variable :

```
float per1;
printf("Enter Percentage=");
scanf("%f", &per1);
```

Here, in scanf() function, %f is a format string for the float variable and (&) per1 will give the address of variable per to store the value at variable per location.

For Character Variable :

```
char ans1;
printf("Enter answer=");
scanf("%c",&ans1);
```

Here, in scanf() function, %c is a format string for a character variable and (&) ans1 will give the address of the variable ans1 to store the value at the variable ans1 location.

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

Format String	Meaning
c	Data item is a single character
d or i	Data item is a decimal integer
f	Data item is a floating–point value
e	Data item is a floating–point value
x	Data item is a Hexa integer
o	Data item is an Octal Integer
s	Data item is String
u	Data item is an unsigned integer

Now, consider another example, suppose there are 3 variables char name [15], in emp_id, float salary then the scanf statement for these 3 variables will be

scanf("%s %d %f", name, &emp_id, &salary);

The above statement contains three characters group. %s represents the first parameter, that is, string name, second character group %d represents that the parameter has a decimal 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. Date items may continue onto two or more lines, since the newline character is considered to be a whitespace character.

As we know that an s–type conversion character applies to a string terminated by a whitespace character. Therefore, a string that includes whitespace characters cannot be entered in this manner. To do the same, the s–type conversion character within the control string is replaced by a sequence of characters enclosed in square brackets designated as [….]. Whitespace characters may be included within the brackets, thus accommodating strings that contain such characters.

When the program is executed, successive characters will continue to be read from the standard input device as long as 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.

The string will however, terminate once an input character is encountered that does not match any of the characters within the brackets. A null character \0 will then automatically be added to the end of the string.

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 a1, b1, c1;

scanf ("%d %3d %3d, &a1, &b1, &c1);

Now, if the data input from the keyboard is 1, 2, 3 then the result will be a1=1, b1=2, c1=3 but suppose if the data input is 123, 456, 789 then it will result in x1=123, y1=456, z1=789.

If the input is 1234 5678 9 then x=123 y=4 and z=567. The remaining two digits (8 and 9) would be ignored, unless they were read by a scanf statement.

❑ **Check Your Progress – 2 :**

1. To read or write a string from/on console _____ format string is used.

   [A] %f [B] %d [C] %h [D] %s

2. If str is a character array (string), Then to accept the string from the user following is a valid syntax.

   [A] scanf("%s", str); [B] scanf ("%s", &str);

   [C] scanf("%d", &str) [D] scanf("%d", str);

3. If integer variable y=143 then output of printf("%x",y); is _____.

   [A] 9f [B] 8f [C] 143 [D] 341

4.4 Formatted Output :

The printf() function is used to display on the Console or to display the value of some variable. The general syntax for the printf() function is as follows :

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

To print some message on the screen :

printf("God is great");

This will print the message God is great on the screen or console.

More Examples :	Output
printf("\nIndia is the best");	India is the best
printf("\nVandeMatram");	VandeMatram
printf("\nJai Hind");	Jai Hind

To print the value of some variable on the screen

❖ **Integer Variable :**

int a1=10;

printf("%d", a1);

Here, %d is a format string to print some integer value and a is the integer variable whose value will be printed by the printf() function. This will print the value of a1, "10" on the screen. You can make this output interactive by writing them :

int a1=10;

printf("a1=%d", a1);

This will print a1=10 on the screen. To print multiple variables value, one can use the printf() function in the following way :

int p1=1000, r1=10, n1=5;

printf("amount=%d rate=%d year=%d",p1,r1,n1);

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

❖ **Float Variable :**

float per1=70.20;

printf("Percentage=%f", per1);

Here %f is a format string to print some float(real) value and per1 is the float variable whose value will be printed by the printf() function. This will print the value of a Percentage=70.20 on the screen.

❖ **Character Variable :**

char ans1="Y";

printf("Answer=%c", ans1);

Here %c is a format string to print a single character value and ans1 is the character variable whose value will be printed by the printf() function.

This will print value of ans1, Answer=Y on the screen.

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

char c1='B', c2='A', c3='O', c4='U';

printf("Name = %c %c %c %c",c1,c2,c3,c4);

❑ **Check Your Progress – 3 :**

1. To assign a character to the character variable, value has to be encased within _____.

   [A] double quotes

   [B] single quotes

   [C] parenthesis

   [D] no parenthesis or quotes are required

2. Strings can be represented in _____.

   [A] double quotes

   [B] single quotes

   [C] parenthesis

   [D] no parenthesis or quotes are required

This will print "Name=B A O U" on the screen. You can also use a single printf() function to print different data type variable's value as well.

Example :

```
int rno=10;
char res='P';
float per=75.70;
printf("Rollno=%d Result=%c Percentage=%f",rno,res,per);
```

This will print message "Rollno=10 Result=P Percentage=75.70" on the screen.

❑ **Check Your Progress – 4 :**

1. Integers, float and double type of numerical variable are initialized using _____.

   [A] double quotes

   [B] single quotes

   [C] parenthesis

   [D] no parenthesis or quotes are required

2. If printf("RollNo=%d, 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 :

Few more unformatted functions are given in the table. And the behaviour of the function is explained in the second column of the table.

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

❑ **Check Your Progress – 5 :**

1. To read a character without displaying it on console _____ function is used.

   [A] getchar() [B] getch() [C] getche() [D] putchar()

2. _____ function allows user to input multiple characters but accept only first character of the string.

   [A] getchar() [B] getch() [C] getche() [D] scanf()

3. To print the string having 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 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.

4.8 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

❑ **Check Your Progress 4 :**

1. [D] no parenthesis or quotes are required
2. [C] int, char, float

❑ **Check Your Progress 5 :**

1. [B] getch() 2. [A] getchar()
3. [C] puts()

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 :

1. Born to Code in C, H. Schildt
2. C Programming, Ed. 2, Kerninghan and Ritchie
3. C Programming with Problem Solving, Jacqueline A Jones, Keith Harrow
4. C Programming, Balaguruswamy
5. Let us C,Yashwant Kanetkar
6. Programming in C, S. Kochan
7. Programming in ANSI C, Agarwal
8. The Art of C, H. Schildt
9. Turbo C/C++ – The Complete Reference, H. Schildt
10. Programming in C, Ashok N. Kamthane by PEARSON

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. A 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.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	1	2	3	4
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	_____
Language and Style	☐	☐	☐	☐	_____
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	_____
Conceptual Clarity	☐	☐	☐	☐	_____
Check your progress Quest	☐	☐	☐	☐	_____
Feed back to CYP Question	☐	☐	☐	☐	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....

Dr. Babasaheb Ambedkar
Open University Ahmedabad

BCAR-103/
DCAR-103

Fundamentals of Programming Using C Language

BLOCK 2 : DECISION MAKING AND LOOPING

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 Objectives :

"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 5 : Decision Making and Branching

Unit 6 : Looping

Unit 7 : Solve Programmes – I

Unit 8 : Solve Programmes – II

Unit 05

DECISION MAKING AND LOOPING

UNIT STRUCTURE

5.0 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

5.1 Introduction :

In the previous unit, you have studied about the basics of C language, that is, the words or statements which are used to write C program. We also saw the use of data types, variables and constants. Now in this unit, we will be discussing about the decision–making statements, which are very helpful while writing programs and helps the programmer to use it whenever certain decisions have to be made.

5.2 Decision Making with the if Statement :

Whenever you are asked to make decisions, you can use if statement for the same purpose. Whenever a problem is given in which you are supposed to make decisions, you have to use If statement with several variations.

5.2.1 Simple if Statement :

The if statement is used to perform a logical test or you can say check the particular condition and then select one of two possible alternatives depending on the outcome of the logical test (i.e. whether the outcome is true or false). Thus, in its simplest general form, the syntax is as follows :

if (condition)

Statement;

The condition must be placed in parenthesis, as shown. The condition is formed by use of relational operators. In this form, the statement will be executed only if the condition has non–zero value (i.e. true). If the condition has a value of zero (i.e. condition is false), then the statement will be ignored or bypassed.

The statement can be either simple or compound. In practice, it is often compound statement, which may contain other control statements.

5.2.2 The if…ELSE statement :

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

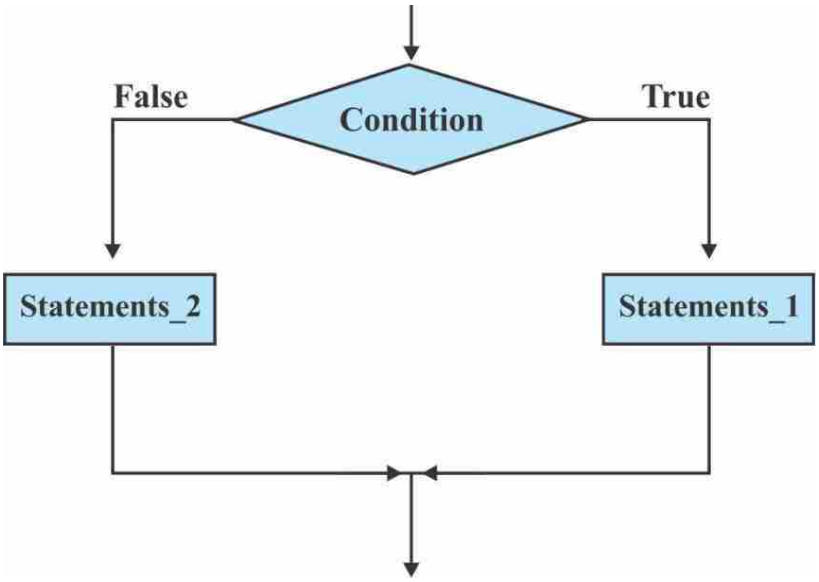
if (condition)

Statement 1;

else

Statement 2;

The condition is evaluated. If it has a non–zero value (i.e. if condition is true), then statement 1 will be executed. Otherwise, (i.e. condition is false), statement 2 will be executed. The flowchart for if–else statement is as follows



For example, program to find maximum of two numbers

```
#include<stdio.h>
void main()
{
        int a=7, b=5;
        if( a>b)
                printf("a is larger")
        else
                printf("b is a larger");
}
```

5.2.3 Nesting of 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;
}
```

Format 4 :

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

For Example :

```
#include<stdio.h>
void main()
{
    int x=5, y=7, z=6;
    if(x>y)
    {
        if(x>z)
            printf("Greatest Number is :%d",x);
        else
            printf("Greatest Number is :%d",z);
    }
    else
    {
        if(y>z)
            printf("Greatest Number is :%d",y);
        else
            printf("Greatest Number is :%d",z);
    }
}
```

5.2.4 The ELSE..if ladder :

Whenever a series of many conditions is checked, then the if … else if … else ladder is used. The general form of else–if ladder can be written as :

```
if(Condition1)
    Statement1;
else if(Condition2)
    Statement2;
else if(Condition3)
    Statement3;
else if(Condition)
    tatement n;
else
    Default Statement;
```

The above construct is known as the if…else construct or ladder. The conditions get evaluated starting from the top towards the bottom. Whenever a condition is found to be true, the statement associated with that particular if statement gets executed and the control of the program gets transferred to statement–x skipping the rest of the statements written after it. If all the conditions specified with if become false, then the default statement gets executed.

Consider an example,

Suppose you are asked to prepare grade cards of students using the table given below :

Marks	Grade
>80	Distinction
>60 and <80	Ist Class
>50 and <60	IInd Class
>40 and <50	IIIrd Class
<40	Fail

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

```
void main( )
{
    int m=0;
    printf("Enter marks\n");
    scanf("%d",&m);
    if(m>=80)
        printf("Distinction\n");
    else if(m>=60)
        printf("Ist Class\n");
    else if(m>=50)
        printf("IInd Class\n");
    else if(m>=35)
        printf("IIIrd Class\n");
    else
        printf("Fail");
}
```

❑ **Check Your Progress – 1 :**

1. The keyword else can be used with _____.

   [A] if statement [B] switch statement

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

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

   [A] If [B] Else

   [C] Break [D] None of the Above

3. If we place an if condition, within one more if condition is called _____.

   [A] simple if [B] if … else

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

5.3 The Switch Statement :

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. In short, the switch statement may be thought of as an alternative or replacement to the use of nested if–else statements, though it can only replace those if else statements that tests for equality. In such cases, switch statement is generally much more convenient.

The general form of switch–case statement is :

```
switch(expression)
{
    case expression1 :
        Statement1;
        Statement2;
    case expression2 :
        Statement1;
        Statement2;
    case expression3 :
        Statement1;
        Statement2;
}
```

Each of these expressions must be either an integer constant or a character constant. Each statement following the case labels may be either simple or compound. Compound statement does not require {} in switch statement. While executing switch statement, the expression is evaluated and control is transferred directly to the group of statements whose case labels value matches the value of the expression. If the case–label value does not match with the value of the expression, then none of the groups within the switch statement will be selected. In this case, the control is transferred directly to the statement that follows the switch statement.

Example :

```
switch (option=getchar())
{
case 'r' :
case 'R' :
    printf ("RASBERRY");
    break;
case 'w' :
case 'W' :
    printf ("WATERMELON");
    break;
```

```
case 'b' :
case 'B' :
        printf ("BLUEBERRY");
        break;
}
```

Thus, RASBERRY will be displayed if value option represents either r or R. WATERMELON will be displayed if option represents either w or W and BLUEBERRY will be displayed if option represents either b or B. Note that each group of statements has two case labels to account for, either upper or lower case. Also, note that each of the first two group ends with the break statement. The break statement is used to transfer control out of the switch statement, so preventing more than one group of statement from being executed.

One of the labelled groups of statements within the switch statement may be labelled default. This group will be selected if none of the case labels matches the value of expression. The default group may appear anywhere within the switch statement. It is not necessary 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

```
switch(option=toupper(getchar())
{
case  'R' :
        printf("RASBERRY");
        break;
case 'B' :
        printf("BLUEBERRY");
        break;
case  'W' :
        printf("WATERMELON");
        break;
default :
        printf("ERROR");
        break;
}
```

The switch statement now contains a default group, which generates an error message if none of the case labels matches the original expression.

The library function toupper() converts all incoming characters to the upper case. Hence, option will always be assigned an upper–case character, so there is no need of multiple cases. Each group of statement ends with a break statement to transfer control out of the switch statement. Break statement is optional in last group, since control will automatically be transferred out of the switch statement. However, as a matter of good programming practice, the break statement is included in last group.

❑ **Check Your Progress – 2 :**

1. Each case statement in switch () is separated by _____.

   [A] exit() [B] continue [C] break [D] goto

2. character constant in switch statement automatically converted into _____.

   [A] floats [B] integers

   [C] Boolean [D] None of the Above

3. If we forget to write 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

5.4 The Conditional Operator (?:) :

Simple conditional operations can be carried out with the conditional operator (?:). An expression that makes the use of the conditional operator is called a conditional expression.

Conditional operators are also known as ternary operators.

Syntax :

(Condition) ? True : false

Exp1 ? Exp2 : Exp3

Here, if condition is evaluated as true then value of exp2 will return and if condition is evaluated as false then exp3 will return.

For example,

T=(I<0) ? 0 : 100

If I variable's value is less than 0 then condition is evaluated as true and T will be assigned with 0 but if I variable's value is greater than 0 then condition is evaluated as false and 100 will be assigned to variable T. For example,

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

❑ **Check Your Progress – 3 :**

1. _____ operator is a ternary operator.

   [A] % [B] && [C] ?: [D] sizeof

2. (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

5.5 The Goto 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. The syntax of goto statement is

goto label;

Where label is a valid identifier used to label the target statement to which control will be transferred.

The label must be followed by a colon. Thus, the target statement will appear as label : statement. Each labelled statement within the program must have a unique label, i.e. no two statements can have same label.

The common uses of goto are :

- Moving around statement or groups of statements under conditions
- Transferring control to the end of a loop under certain conditions, thus bypassing the remainder of the loop, during the loop, during the current pass
- Transferring control completely out of a loop under certain conditions, thus terminating the execution of the loop

Branching around statement can be accomplished with the if–else statement.

For example,

```
m=1;
loop :
m++;
if (m < 100)
    goto loop;
```

Jumping to the end of a loop can be carried out with the continue statement and jumping out of a loop is easily completed using the break statement. The use of these structured features is preferable to the use of the goto statement because the use of goto tends to encourage logic that skips all over the program.

Consider, for example, a situation in which it is necessary to transfer control out of a doubly nested loop if certain conditions are detected. This can be done with two if break statements. One within each loop, though this is awaked. A better solution in this particular situation might make use of the goto statement to transfer out of both loops at once

❑ **Check Your Progress – 4 :**

1. _____ statement is called unconditional jump.

   [A] if condition [B] for loop [C] while loop [D] goto

2. Good programmers try to avoid _____ statements into their programs, because they reduce performance of the program.

   [A] goto [B] switch…case [C] if…else [D] for loop

5.6 Using Logical Operators In If

Sometimes we need to place two or more conditions in a single if statements. In such cases, two or more conditions are separated by logical operators such as AND (&&) and OR (||). For example, to find a greatest number from given 3 numbers can be implemented using nested if conditions, and that is discussed in 1.2.3. The same problem can be sorted out using logical operators. In the following example we have solved the same problem using, logical AND. Compare this program with the program discussed in 1.2.3 and think which logic is easier and more readable.

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

In this example we are evaluating that if x>y and x>z then x will be the greatest number, and in the same way we are checking for y and z. Logical operator && is used here because x will be a greatest number, if x>y AND x>z (It is mandatory that both the conditions have to be TRUE). Consider another program where we are taking an alphabet from the user and we check it is Vowel or Consonant.

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

In this program is entered character is either 'a', or 'e', or 'i', or 'o', or 'u', we need to print it is Vowel, otherwise it is Consonant. Therefore, here multiple conditions are separated by logical operator (| |).

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

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

❑ **Check Your Progress – 5 :**

1. To test whether the character entered by the user is Vowel or Consonant following logical operators are used to separate multiple conditions in if statement.

   [A] && [B] ||

   [C] ! [D] None of the above

2. Identify the operator from the following is not logical operator.

   [A] ! [B] || [C] % [D] &&

5.7 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

5.8 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

❑ **Check Your Progress 4 :**

1. [D] goto 2. [A] goto

❑ **Check Your Progress 5 :**

1. [B] | | 2. [C] %

5.9 Glossary :

1. goto : It is a statement in C–Language which performs jump to some statement without evaluating condition. It is also called unconditional jump.

2. Switch … case : It is a statement in C–Language, which match the expressing passed into switch, with all the cases and matched case will be executed. Unlike if condition, it used to check only equality.

5.10 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.
4. Write a short note on goto statement.
5. Show the use of logical operators in if condition.

5.11 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.

5.12 Case Study :

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

5.13 Further Reading :

- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.
- "Programming in ANSI C" by E Balagurusamy, McGraw–Hill Education.

Unit
06

LOOPING

UNIT STRUCTURE

6.0 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

6.1 Introduction :

Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming. A number of programs or websites that produce extremely complex output (such as a message board) are really only executing a single task many times. They may be executing a small number of tasks, but in principle, to produce a list of messages only requires repeating the operation of reading in some data and displaying it. Thus, a loop lets you write a very simple statement to produce a significantly greater result simply by repetition. In this unit, we will discuss about loops to execute the same set of statements again and again.

6.2 Looping :

If we want to perform certain task for a number of times or we want to execute same statement or a group of statements repeatedly then we can use loops. There are three different types of loop structure in C.

- The while loop executes group of statements until test condition is true. When the programmer does not know in advance how many times the loop will be executed, in that case while loop is useful.
- The do while loop is similar, but the test occurs at the end of loop. This ensures that the loop body is executed at least once.
- The for loop is mostly used, usually where programmer know the loop will be executed a fixed number of times.

6.2.1 The While Statement :

The while statement is used to carry out looping operations. The general form of the while loop is :

```
initialization;
while(exp)
{
        statement 1;
        statement 2;
        increment/ decrement;
}
```

The statements enclosed within two braces are called the body of the loop. The braces are needed only if the body contains two or more statements. However, it is a good practice to use braces even if the body has only one statement. The body of loop will be executed repeatedly as long as the expression evaluated is true. When the expression is evaluated as false or when condition will be false then it will come out from the loop and stop the execution of that loop. Thus, it is top tested or entry–controlled loop in which condition evaluated first. Initialization statement initializes some loop variable with some value. Increment or decrement operator increases or decreases the value of loop variable used in the expression. It must include some features, which eventually stops execution of the loop.

Logical expression can form with help of relational and logical operators which is evaluated either true or false. (Remember that true corresponds to non–zero value and false corresponds to zero value). Thus, the statement will continue to execute as long as the logical expression is true.

Suppose we want to display first 10 numbers on each line. This can be accomplished with the following program.

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

Initially, num is assigned a value of 1. The while loop then displays the current value of num, increases its value by 1 and then repeats the cycle until the value of the num becomes 10, resulting 10 consecutive lines of output. Each line will contain a successive integer value, beginning with 1 and ending with 10. Thus, when the program is executed, the following output will be generated :

1 2 3 4 5 6 7 8 9 10

❑ **Check Your Progress – 1 :**

1. A while loop, without increment / decrement statement is called _____.

   [A] For loop [B] Null loop

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

2. Initialisation statement in a while loop has to 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. Conditional statement in a while loop has to 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

6.2.2 The Do Statement :

Sometimes, however, it is desirable to have a loop with the test for continuation at the end of each pass. This can be fulfilled by means of the do–while statement.

The general form of do–while statement is :

```
do
{
    statement 1;
    statement 2;
    increment/decrement operator;
} while(expression);
```

The enclosed statement within 2 braces will be executed repeatedly as the value of expression is true. Note that statement will always be executed at least once, since the condition is checked at the end of the first pass through the loop. It is bottom tested or exit controlled loop. The statement can be either simple or

compound, though most applications will require it to be a compound statement. It must include some features which eventually alters the value of expression so that loop will be terminated.

In practice, expression is usually a logical expression, which is either true (with a non–zero value) or a false (with a value of 0). The included statement will be repeated (i.e. another pass will be made through the loop) if the logical expression is true.

For most applications it is more natural to test for continuation of a loop at the beginning rather than at the end of the loop. For this reason, the do…while statement is used less frequently than while statement. Suppose that we want to display consecutive numbers 1, 2, 3…10 with one num on each line. This can be shown in the following program.

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

The num is initially assigned a value of 1. The do–while loop displays the current value of num, increases its value by 1 and then check to see if the current value of num exceeds 10. If so, loop terminates; otherwise the loop continues, using the new value of num. The net effect is that the loop is repeated 10 times, resulting in 10 successive lines of output. Thus, when the program is run, the following output is executed :

```
1    2    3    4    5    6    7    8    9    10
```

❑ **Check Your Progress – 2 :**

1. _____ loop is an exit–controlled loop.

   [A] For loop [B] While loop

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

2. In _____ loop, condition will be evaluated after execution of the body part of the loop.

   [A] do … while loop [B] while loop

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

3. _____ loop will be executed at least once.

   [A] While loop [B] Do … while loop

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

6.2.3 The For Statement :

The for statement is another flexible entry controller loop structure similar to while loop. The syntax of the for loop is :

```
for (initialization; test condition; increment)
{
        statement 1;
        statement 2;
}
```

The execution of the for statement is as follows

- The first part i.e. initialisation, initialize loop control variables using assignments statement such as I=0 and count=0.
- The test condition is evaluated before execution of statements in the loop. The test condition is relational expression, such as I>0 or I<10 that determines execution of loop. If outcome of test is true loop will be executed; otherwise the loop will be terminated.
- The control variable is incremented or decremented which affects test condition. If the new value of the control variable is satisfied under test condition, then the body of the loop is executed. This process continues till the value of the control variable fails to satisfy the test–condition.

(a) Consider the following segment of a program :

```
for (cnt=1; cnt<10; cnt++)
{
        printf ("%d\n", cnt);
}
```

This for loop is executed 10 times and prints the numbers 1 to 10 each on new line. The three sections within parentheses must be separated by a semicolon. Note that there is no semicolon at the end of the increment section cnt++.

The for statement also allows negative increments. For example, the loop discussed above can be written as follows :

```
for(cnt=10; cnt>=1; cnt– –)
{
        printf("%d \n",cnt);
}
```

This loop is also executed 10 times. However, the output would be from 10 to 1 instead of 1 to 10.

❑ **Check Your Progress – 3 :**

1. _____ loop is an entry–controlled loop.

   [A] For loop [B] While loop

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

2. In for loop initialization, condition and increment/decrement statements are separated by _____.

   [A] ; (Semi–Colon) [B] : (Colon)

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

6.2.4 Additional Features of for Loop :

The for loop in C has several features that are not found in the other loop constructs. For example, multiple variables can be initialized at a time in for statement.

```
for(n=1,m=50; n<m; n++, m– –)
{
        p=m/n;
        printf("%d%d%d \n",n,m,p);
}
```

This is perfectly valid. The multiple arguments in the increment section are separated by commas.

The third feature is that the test condition may include two or more conditions and the testing need not be limited only to the loop control variable. Consider the example below :

```
sum=0;
for(num=1;num<20 && sum<100; num++)
{
        sum=sum+num;
        printf(("%d \n", sum);
}
```

The loop uses a compound test condition with the control variable and external variable sum. The loop is executed as long as both the conditions num<20 and sum<100 is true. The sum is evaluated inside the loop.

It is also permissible to use expressions in the assignment statements of initialization and increment sections. For example, a statement of the type :

```
for(z=(m+n)/2; x<0;x=x/2)
```

is perfectly valid.

Another unique aspect of the for loop is that one or more sections can be omitted, if necessary. Consider the following statements :

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

Both the initialization and increment sections are omitted in the for statement. The initialization has been done before the for statement and the control variable is incremented inside the loop. In such cases, the sections are left blank. However, the semicolons separating the sections must remain. If the test condition is absent, then there will be an infinite loop. Such loops can be broken using break or go–to statements in the loop. We can set up time delay loop using the null statement as follows :

```
for(j=1000; j>0;j– –) { }
```

If the loop is executed 1000 times without producing any output, it simply causes a time delay. Notice that the body of the loop contains only a semicolon, known as null statement. This can also be written as :

```
for(i=1000; i>0; i– –);
```

It is perfectly valid. The C compiler will not give any error message. If programmer place a semicolon by mistake at the end of a for statement, the semicolon will be considered as a null statement and the program may produce meaningless output.

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 be completely embedded within the other. There can be no overlap. Also, each loop must be controlled by a different controlled variable. Nesting can be done up to any levels.

Whenever you are asked to perform certain set of operations on a certain range of data, then in that case you can use the concept of nested loops. For example, if you are asked to display table of all numbers within a certain range say from 2 to 100. The program for the same can be written as :

```
void main( )
{
    int i=0,j=0,k=0;
    for(i=2;i<=100;i++)
    {
        for(j=1;j<=10;j++)
        {
            k=i*j;
            printf("\t",k);
        }/*End of Loop j*/
        printf("\n");
        k=0;
    }/*End of loop i*/
} /*End of main function*/
```

In the above program, the outer loop is of i and inside that loop there is another loop of j. So, as we have seen that j's loop is present inside loop of i, hence it is an example of nested loop.

❑ Check Your Progress – 4 :

1. 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. Two initialization statements in the for loop separated by _____.

   [A] ; (Semi–Colon) [B] : (Colon)

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

3. If For loop exists in one more for loop is called _____.

[A] Nested for loop [B] Infinite loop

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

6.3 Jumps in Loops :

We can skip from one point of the program to another by using break and continue statements. The break statement sends the control of program from a particular point to the end of the loop, whereas, the continue statement sends the control of program to the beginning of the loop.

6.3.1 Break Statement :

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. It can be used within a while, a do–while, for or a switch statement. The break statement is written simply as break; without any embedded expressions or statement. For example,

```
for(cnt=1; cnt<=10; cnt++)
{
        if(cnt==5)
        break;
        printf("\ncnt=%d",cnt);
}
```

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

6.3.2 Continue Statement :

This is similar to break except it does not cause the loop to terminate. It only works within loops where its effect is to force an immediate transfer to the loop control statement.

- In a while loop, jump to the test statement
- In a do while loop, jump to the test statement
- In a for loop, jump to the test and perform the iteration

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

The continue statement is used to skip or to bypass some step or iteration of looping structure. It is simply written as continue.

```
for(cnt=1; cnt<=10; cnt++)
{
        if(cnt==5)
                continue;
        printf("\ncnt=%d",cnt);
}
```

The output of the above program will be 1,2,3,4,6,7,8,9,10. The 5thiteration of the loop will be skipped as we have defined the continue for that iteration. So, it will not print the value '5'.

❑ **Check Your Progress – 5 :**

1. To terminate premature loop _____ keyword is used.
   [A] continue [B] break [C] end [D] stop
2. Predict the output of the following program :

```
for(cnt=1; cnt<=10; cnt++)
{
    if(cnt>4 && cnt<8)
        continue;
    printf("%d,",cnt);
}
```

   [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,

6.4 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.

6.5 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)

❑ **Check Your Progress 4 :**

1. [D] for (initialization; condition; increment) { }
2. [C] , (Comma) 3. [B] Infinite loop

❑ **Check Your Progress 5 :**

1. [B] break 2. [C] 1,2,3,4,8,9,10,

6.6 Glossary :

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

6.7 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.

6.8 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.

```
    *
  *   *
*   *   *
```

6.9 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.

6.10 Further Reading :

1. Born to Code in C, H. Schildt
2. C Programming, Ed. 2, Kerninghan and Ritchie
3. C Programming with Problem Solving, Jacqueline A Jones, Keith Harrow
4. C Programming, Balaguruswamy
5. Let us C, Yashwant Kanetkar
6. Programming in C, S. Kochan
7. Programming in ANSI C, Agarwal
8. The Art of C, H. Schildt
9. Turbo C/C++ – The Complete Reference, H. Schildt
10. Programming in C, Ashok N. Kamthane by PEARSON

Unit

07

SOLVED PROGRAMS – I

In this unit we will discuss, some solved programs. We will start our journey from very basic programs. Slowly and gradually we will move up to the programs of conditional statements (Chapter : 1 of Block–2). Programs of looping statements we will discuss in the next unit that is Unit : 4. Each program will teach you the how to use different concepts we have discussed in earlier chapters programmatically. At the end of every program, we have tried to explain the logic. We hope this will help you a lot, to learn C–Programming.

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

❖ **Output :**

Hello World :

In this program, we have included header file called 'stdio.h'. 'stdio.h' is a header file which has basic functions like 'printf()' and 'scanf()'. So, it is mandatory to include this header file, if you are calling (using) 'printf()' or 'scanf()' kind of functions. The full form of stdio.h is 'Standard Input Output Header file'. In the next line we have return 'void main()'. Here, we are creating a 'main()' function. As we know that every C–Program must have 'main()' function. Because this function does not return any value, we have return 'void'. We are starting the function with '{' and ends the functions with '}'. Between start '{'and, end '}' we have written /* This is my first program */. We know that the sentence written between /* and */ will be treated as a comment. Comment is just to increase the readability of the program. Compiler will simply skip this line, and it will not be executed. In the last, we have called a function 'printf()' and we have passed the data "\nHello". Here 'printf()' function execute the code written in the 'stdio.h' file and print the string "Hello World :" on the console.

We are using Code Blocks software to execute the program. We recommend you to use the same as it is freely available. If you are using 'Turbo C++' then in every program you have include header file 'conio.h', that is 'Console Input Output Header file'.

You have to use function 'clrscr()' to clear the console screen after declaring variables, and finally you need to call getch() function at the end of every program. In the following program we have explained what changes you have to in every program if you are a 'Turbo C++' user.

```
/* Program:1–B Printing Hello World on the Console */
#include<stdio.h>
#include<conio.h> //Turbo user needs to add this header file in every program
void main()
{
    /*This is my first program */
    clrscr(); //Turbo user need to call function clrscr in every program
    printf("Hello World :");
    getch(); //Turbo user need to call getch function in every program
}
```

From the next program onwards, we will not include the header file 'conio.h' and we will not call function 'clrscr();' and 'getch();' , with the hope that turbo user will do this in every program. If you are using code blocks software, you don't have to do all these things. Program 1–A will be perfectly run in your Code Blocks software.

```
/* Program:2 Program to Find Simple Interest */
#include<stdio.h>
void main()
{
    int p,r,n, i;
    printf("Enter Principle Amount :");
    scanf("%d", &p);
    printf("Enter Rate of Interest :");
    scanf("%d", &r);
    printf("Enter Number of years :");
    scanf("%d", &n);
    i=(p*r*n)/100;
    printf("\nSimple Interest is :%d",i);
}
```

❖ **Output :**

Enter Principle Amount: 100

Enter Rate of Interest:10

Enter Number of years:5

Simple Interest is:50

Make sure in the above program it you enter value 1000 for principal amount, then it is possible that you will get unexpected result. Because the value of (p * r * n) is going beyond 32000. We know that the integer can store maximum 32,767. In this case you can compute the value i using following formula :

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

Now, consider the following program, in which we are going to use float variables to accept and print values with decimal points, rather than integer variable.

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

❖ **Output :**

Enter Celsius:37

Temperature in Fahrenheit is:98.60

In this program we have taken 2 float variables c for accepting and storing Celsius temperature and f, to compute and store temperature in Fahrenheit. We prompt to the user to enter Celsius temperature, and stored it in variable c using scanf() statement. Variable c is of type float, so we have used format string "%f". We use formula ***f=(c*9)/5+32;*** to compute Fahrenheit from the variable c. Finally we print it using printf() statement. We wish to print the value of variable f in 2 decimal points. So, we have used format string "%.2f". If you use only "%f" format string then it will print : **Temperature in Fahrenheit is : 98.600000.**

```
/*Program :4 Program to Find Greater number from given 2 numbers */
#include<stdio.h>
void main()
{
    int num1, num2;
    printf("Enter First Number :");
    scanf("%d", &num1);
    printf("Enter Second Number :");
    scanf("%d", &num2);
    (num1>num2) ?printf("\nGreater Number is :%d",num1) :
    printf("\nGreater Number is :%d",num2);
}
```

❖ **Output :**

Enter First Number:5

Enter Second Number:7

Greater Number is:7

In this program, we have taken two integer variables num1 and num2. We are accepting the values for our variables from the user. Then using condition operator (?:) we are printing which number is greater.

```
/* Program:5 Program to Find greatest number from given 3 numbers
Using Conditional Operators*/
#include<stdio.h>
void main()
{
    int n1, n2, n3, max;
    printf("\nEnter Any 3 Numbers :");
    scanf("%d%d%d",&n1,&n2,&n3);
    max=(n1>n2) ? (n1>n3) ?n1 :n3 : (n2>n3) ?n2 :n3;
    printf("\nGreater Number is :%d",max);
}
```

In this program we find the greatest number from given 3 numbers using conditional operator. Also, in this program we have accepted 3 value for our integer variables n1, n2 and n3 from the user, using single scanf() statement. The output of the conditional operator is stored in the extra variable called max. And finally, the value of the variable max, printed on the console screen.

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

❖ **Output :**

Enter Any Number:57

Given number 57 is Odd :

In the above program, we have taken 2 integer variables num and r. We are taking the value for the num variable from the user. We are dividing num variable by 2 and store the remainder into variable r, using modulo operator (%). Finally, using if condition we are checking if the value of r is 0 then that number is Even otherwise given number is Odd. This is an example if If…Else statement.

```
/* Program:7 Program to Check given number is Positive Negative or Zero*/
#include<stdio.h>
void main()
{
    int num;
    printf("\nEnter Any Number:");
    scanf("%d", &num);
    if(num>0)
        printf("Given number %d is Positive:",num);
    else if(num<0)
        printf("Given number %d is Negative:",num);
    else
        printf("Given number is 0 (Zero):");
}
```

❖ **Output :**

Enter Any Number:45

Given number 45 is Positive :

In the above program, we are accepting the value of variable num, from the user. We will check if the value of num is greater than 0, then it is positive. If not, then again, we are evaluating second condition, that the value is smaller than 0 ? If it is then that number is negative. If both the condition 'num>0' and 'num<0' fails, then will print number is 0. In this example we have used If…Else If…. Else statement.

```
/* Program :8 Program to find greatest number
Using Nested if conditions*/
#include<stdio.h>
void main()
{
    int num1, num2, num3;
    printf("\nEnter Any 3 Numbers:\n");
    scanf("%d%d%d",&num1,&num2,&num3);
    if(num1>num2)
    {
        if(num1>num3)
        {
            printf("\n%d is greatest number",num1);
        }
        else
        {
            printf("\n%d is greatest number",num3);
```

```
        }
    }
    else
    {
        if(num2>num3)
        {
            printf("\n%d is greatest number",num2);
        }
        else
        {
            printf("\n%d is greatest number",num3);
        }
    }
}
```

In the above program we have taken 3 numbers from the user. First, we check whether num1>num2. If yes then we are comparing num1 with num3 (num1>num3), if again yes then num1 is the greatest number, otherwise num3 is the greatest number.

If the first condition (num1>num2) fails, we are comparing num2 and num3 variables. If num2>num3, then num2 is the greatest number, otherwise num3 is the greatest number.

In this program we have written an if condition, inside one more if condition. Hence, we can say that this is an example of nested if conditions.

```
/* Program:9 Program to Find greatest number from given 3 numbers
Using Logical AND operator*/
#include<stdio.h>
void main()
{
    int num1, num2, num3;
    printf("\nEnter Any 3 Numbers :\n");
    scanf("%d%d%d", &num1,&num2,&num3);
    if(num1>num2 && num1 > num3)
        printf("\n%d is greatest number",num1);
    else if(num2>num1 && num2>num3)
        printf("\n%d is greatest number",num2);
    else
        printf("\n%d is greatest number",num3);
}
```

In the above program, we do the same thing. We are finding the greatest number from given 3 numbers. We have accepted 3 numbers from the user. In the if condition we evaluating, if num1>num2 and num1>num3 then we can say

num1 is the greatest number. To evaluate 2 conditions in the one if statement, we have used logical AND (&&).

Now consider the following program that is program :10, which is an example of goto statement. Statement 'goto' is called an unconditional jump, which set the execution control on a specific statement (label). In the program, initially sequential execution will be started and first 2 printf statements, gets executed, which will print 'A' and 'B'. After execution first 2 statements, goto statement will send the program control to the label 'end :'. In short, the statements written between goto statement and label 'end :' will be skipped and statement written after 'end :' label will be executed.

```
/*Program :10 Example of goto statement */
#include<stdio.h>
void main()
{
    printf("A");
    printf("\tB");
    goto end;
        printf("\tC");
        ptintf("\tD");
        end :
            printf("\tE");
}
```

❖ **Output :**

A B E

```
/*Program :11 Example of switch...case statement */
#include<stdio.h>
void main()
{
    int num;
    printf("Enter any number from 1 to 4 :");
    scanf("%d",&num);
    switch(num)
    {
    case 1 :
        printf("One");
        break;
    case 2 :
        printf("Two");
        break;
```

```
    case 3 :
        printf("Three");
        break;
    case 4 :
        printf("Four");
        break;
    default :
        printf("\nInvalid Number :");
    }
}
```

❖ **Output :**

Enter any number from 1 to 5:3

Three

In the Program : 11 we have taken the value from the user, and stored it in the variable num. We are passing this value in the switch and trying to match the values with different case like case 1, case 2 and so on. When the value is matched with particular case then that case will be executed. You can see in the output we have passed 3, which will match with case 3. In this case, statement printf("Three"); gets executed. Next break statement will bring program control to the end of the switch…case statement. If user is entering any number which do not match with any case then the default : case will get executed and it will print the message that "Invalid Number".

```
/*Program :12 Check the character is Vowel or not */
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter any Alphabet :");
    scanf("%c",&ch);
    switch(ch)
    {
    case 'a' :
    case 'e' :
    case 'i' :
    case 'o' :
    case 'u' :
        printf("\nVowel");
        break;
    default :
        printf("Consonant");
    }
```

❖ **Output :**

Enter any Alphabet : e

Vowel

In the above example if user is entering any character from either (a, e, I, o, u) then it will print "Vowel". if any other character is entered then its system will execute default case and it will print "Consonant".

```
/* Program :13 Program to swap two variables */
#include<stdio.h>
void main()
{
    int x, y, tmp;
    printf("Enter Value for X :");
    scanf("%d",&x);
    printf("Enter Value for Y :");
    scanf("%d",&y);
    tmp=x;
    x=y;
    y=tmp;
    printf("After swap X is : %d and Y is : %d",x,y);
}
```

❖ **Output :**

Enter Value for X : 5

Enter Value for Y : 7

After swap X is : 7 and Y is : 5

In the above program, we have taken 3 variables x, y and tmp. We have taken values for variable x and y from the user. After taking the values from the user we are copying the value of x in the tmp variable, by statement "tmp=x;". So, if the variable x is 5 and y is 7 then tmp=5. Now, we are executing a statement called x=y. So, value of variable will be copy in the variable x. Therefore, after execution of statement "x=y;" tmp is 5 and x=7. Finally, we are copying the value of the tmp variable, that is 5 to the variable y, by the statement "y=tmp;". Because tmp is 5 after execution of this statement y will be 5. Variable x is already 7. So, we can say that we have swapped the values of the variable x and y.

7.1 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 A, From 50 to 59 – Grade B, 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]

7.2 Suggested Answers to Check Your Progress

1. To check the year is Leap year or not, you need a variable year of type int. Prompt user to Enter a year and scan that value in the variable year. Now in the if condition you have to check whether the year is a century year or not by writing if(year%100==0) if it is TRUE then you have to check another condition if(year%400==0). If this is TRUE then this century year is a leap year otherwise it is not a Leap year. If(year%100==0) condition is FALSE then this is ordinary year. If it is then you have to check if(year%4==0) if it is TRUE then this ordinary year is Leap year otherwise not Leap year. With the help of nested if conditions you can do this program.

2. Let of 2 variables x=5 and y=6 then you need to write following instructions :

   Initially X=5 and Y=7

   x=x+y [After Execution of this line x=12 and y=7]

   y=x–y [After Execution of this line x=12 and y=5]

   x=x–y [After Execution of this line x=7 and y=5]

   We have swapped the values of both variables.

3. This program can be done by using if…else if…. else statement, or by using logical operators :

   Method : 1

```
if (marks >=75)
    printf("Distinction :");
else if (marks >= 60)
    printf ("Grade :A");
else if (marks >= 50)
    printf ("Grade :B");
else if (marks >= 35)
    printf("Pass :");
else
    printf("Fail :");
```

   Method : 2

```
if (marks <=100 && marks >=75)
    Printf("Distinction :");
if (marks < 75 && marks >=60)
    printf("Grade :A");
if (marks < 60 && marks >=50)
    printf("Grade :B");
if (marks < 50 && marks >=35)
    printf("Pass");
if (marks < 35 && marks >=0)
    printf("Fail");
```

4. Declare 4 variables l, b, area and perimeter. Accept the values of variable l and b from the user. Compute area=l*b and perimeter = 2 * (l + b). and display it on the screen.

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

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

6. Declare 4 variable x, y, z and max from the user. Accept 3 numbers from the user and stored it in the x, y, and z variable. Write the following code to print max value.

```
max=x;
if (y> max)
        max=y;
if(z>max)
        max=z;
printf("\n Greatest Number is : %d", max);
```

Unit

08 SOLVED PROGRAMS – II

In this unit, we will try to solve number of programs of loops [Unit : 2 Block : 2]. We will try cover the example of while, for and do … while loop. We will also focus on 'break' and 'continue' keyword and finally we will discuss number of examples of nesting of loop. So, let us start with the first example.

❖ **Syntax of while loop :**

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

Now, we will start programming examples of while loop. We will start from a simple program, and will increase the complexity program to program.

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

In the above program, we have declared a variable I and initialized it with 1. In the while statement we are checking whether, the value of variable 'i' is smaller than 10. If yes then we are printing the value of variable 'i' on the console that is 1. In this program, we are not changing the value of 'i'. Because while is loop statement the control will be transfer to condition again the condition is TRUE as i that is 1 <=10, again it will print 1. And program will print 1 1 1 and so on. Because we are not increasing the value of variable 'i', condition i<=10 becomes true every time, and system will repeat the body of the loop again and again. Make sure loop statement will be completed when the condition will become false.

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

Now, we will try to complete the program, by adding a statement i++ in the body of the loop. Which will increase the value of the variable 'i' by 1, on every iteration of the loop. So, on the first iteration i will be 1, in the second iteration i will becomes 2 and so on. When the value of the variable i will becomes 11 at that time 11 is not smaller or equal to 10 (condition will become false), and loop will stop its repetition. The following program will print from 1 to 10.

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

❖ **Output :**

1 2 3 4 5 6 7 8 9 10

In the next program, we are finding sum of first 10 numbers that 1+2+3+…+10 and print it on the screen. For that we take another variable sum with initial value 0, and on every iteration of a while loop (from i=1 to 10), we will add the value of 'i' to the sum variable.

```
/*Program:3 Sum of first 10 numbers */
#include<stdio.h>
void main()
{
    int i=1, sum=0;
    while(i<=10)
    {
        sum=sum+i;
        i++;
    }
    printf("\nSum is:%d", sum)
}
```

❖ **Output :**

Sum is:55

```
/*Program:4 program to print all even numbers from 1 to 20*/
#include<stdio.h>
void main()
{
    int i=1;
    while(i<=20)
    {
        if(i%2==0)
            printf("%d\t", i);
        i++;
    }
}
```

❖ **Output :**

2 4 6 8 10 12 14 16 18 20

In the above program, we have initialized variable i=1, now we are running a loop till i<=20. So, loop will run for i=1,2,3,4 ….20. We just want to print all even numbers. In order to print only even numbers, we have placed an if condition, which will test if(i%2) ==0 then only number will gets printed on the screen.

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

❖ **Output :**

Enter Any Number:1234

Reverse Number is:4321

In the above program, we have declared 3 variables num, rnum and r. We have taken a number from the user and stored it in the num variable.

Initially we have set value to the rnum variable to 0. Now, we are running a loop till variable rnum >0. In each iteration we are dividing a number by 10 and storing it in the variable r. we are dividing a number by 10, and finally we are computing rnum=rnum*10+r. Finally, when we come out of the loop then we are printing the value of rnum variable on the screen. To understand the process let us take an example where user has entered a number 1234. So, our num variable is 1234 and rnum variable is 0.

Iteration : 1 while (num > 0) // TRUE 1234 > 0
r=num % 10 // r is know 1234%10 =4
num=num/10 // num=1234/10=123
rnum=rnum*10 + r //rnum=0*10+4=4

Iteration : 2 while (num > 0) // TRUE 123 > 0
r=num % 10 // r is know 123%10 =3
num=num/10 // num=123/10=12
rnum=rnum*10 + r //rnum=4*10+3=43

Iteration : 3 while (num > 0) // TRUE 12 > 0
r=num % 10 // r is know 12%10 =2
num=num/10 // num=12/10=1
rnum=rnum*10 + r //rnum=43*10+2=432

Iteration : 4 while (num > 0) // TRUE 1 > 0
r=num % 10 // r is know 1%10 =1
num=num/10 // num=1/10=0
rnum=rnum*10 + r //rnum=432*10+1=4321

Iteration : 5 while (num > 0) //FLASE num is 0 and 0 is not > 0

Print the value of rnum variable that 4321 on the console screen.

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

❖ **Output :**

Enter Any Number:28

Given Number is Perfect Number

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

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

❖ **Output :**

Enter Any Number:47

Given Number is Prime

In this program, we are checking whether the number given by user is Prime or composite. That number which is divisible by 1 or itself, and not divisible by any other number is Prime number. We have taken 3 variables num, i and logic. We are initializing variable I with 2 because we will try to divide that number by 2,3, 4 and up to num–1. We set logic to 1 (assuming that the number entered by the user is Prime). Now, will try to divide that number from 2 to that number –1 using while loop. If we get any number i, by using it we can divide the number

(num%i==0), then we set logic to 0 (means our assumption is wrong, number is divisible and hence it is not prime). In this case we do not have to continue our loop, so we will use keyword 'break'. At the end, if logic is 1, the number is not divisible by any number so it is Prime, otherwise number is divisible by some value of i, and hence it is not Prime.

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

❖ **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 program 7, we have taken a value from the user and then we have checked that value is prime or not. Obviously, to do this we have taken a while loop. Now, in the program 8, rather than taking the value of the variable num from the user, we are generating value from 2 to 50 using another while loop. For each value of num (2 to 50), we are checking that value is prime or not, if the value is prime then, that value will be printed. Here, we need to run a while loop to generate numbers from 2 to 50, and for each value we need a another while loop to test that value is prime or not. In short, we need to place a while loop inside one more while loop. This is called a **nesting of while loop.**

Now, I think sufficient examples we have done of while loop. So, now we will move towards a 'for' loop. For–loop is a popular loop and it reduces, number of lines in the program. The syntax for writing for loop is as follows:

Syntax of for loop :

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

Now, consider a program, which will print all odd number from 1 to N using for loop.

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

❖ **Output :**

Enter Any Number:20

1 3 5 7 9 11 13 15 17 19

In the next program, we are taking a number from the user and try to find a factorial of that number.

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

❖ **Output :**

Enter Any Number:5

Factorial of 5 is = 120

In the above program, we have initialized two variable i=1 and fact=1 separated by comma (,) operator. We will check the condition i<=n and each time variable I gets incremented. So, let if n is 5, I will be 1,2,3,4,5 and each value of multiplied with variable fact. Finally, at the end of the, we are printing the value of fact variable.

Now, in the next program we are taking a number from the user and we are evaluating that the number is Palindrome or not. Palindrome number is that number in which, number and its reverse number is same. From example 12321 is Palindrome number because if you reverse the given number, you will get the same number. Whereas, 1234 is not Palindrome because number 1234 and it's reverse 4321 are not same.

In the program, we have taken variable num to accept a number from the user. We have taken another variable onum to copy num variable (onum=num). One variable r to compute remainder after dividing number by 10, and rnum variable, which is initialized with 0.

See, in the program we have already initialized variable rnum=0 and onum=num. So, in the for loop there is no initialization statement.

```
/*Program:10 To check given number is Palindrome or Not*/
#include<stdio.h>
void main()
{
    int num, onum, r, rnum=0;
    printf("Enter Any Number:");
    scanf("%d",&num);
    onum=num;
    for ( ; num>0;num=num/10)
```

```
    {
        r=num%10;
        rnum=rnum*10+r;
    }
    if(onum==rnum)
        printf("Given number is Palindrome");
    else
        printf("Given number is Not Palindrome");
}
```

❖ **Output :**

Enter Any Number:12321

Given number is Palindrome

In the next program, we take a number from the user and will check whether the given number is Armstrong number or not. Armstrong number is that, whose sum of cube of each digit is equal to that number. For example, 153 is Armstrong number because 13+53+33=1+125+27=153.

To implement this, we have taken 4 variables num, onum, r and sum. We will take the value of num variable from the user and copy it to variable onum. We will start the for loop by initializing sum=0 and onum=num. We will check the condition num > 0. Means when num variable turn to 0, we will stop the loop. In each iteration num variable will be divided by 10 (num=num/10). In the loop first we will computer remainder after dividing a num by 10 (r=num%10), and cube of variable r will be added to the sum variable. sum=sum+(r*r*r). At the end of the loop, we will match the value of sum variable is equal to onum variable ? if it is then the given number is Armstrong number, otherwise it is not Armstrong number.

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

❖ **Output :**

Enter Any Number:153

Given number is Armstrong

Consider the next program in which we are drawing a square shape, using symbol '*'. The number of rows and columns are same as the number entered by the user.

```
/*Program:12 To draw square shape*/
#include<stdio.h>
void main()
{
    int num, r, c;
    printf("Enter Any Number:");
    scanf("%d",&num);
    for(r=1;r<=num;r++)
    {
        for(c=1;c<=num;c++)
        {
            printf("* ");
        }
        printf("\n");
    }
}
```

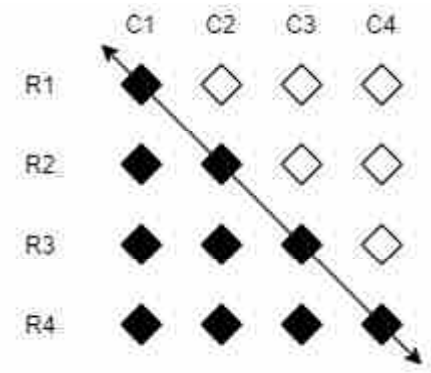
In the above program, we have taken 3 variables num, r, and c. We are initializing variable num with the value entered by the user. Now we, are starting a for loop for variable r (row). Loop 'for(r=1;r<=num; r++)' will print number of rows, equal to the number entered by the user (num). For each row new line is needed, so we have mentioned 'printf("\n");' statement in the for loop of variable 'r'. In each row we need 'num' columns. For that we have writing one more for loop of variable 'c' that is:'for(c=1;c<=num;c++)'. The output of the above program is as follows:

❖ **Output :**

Enter Any Number:4

```
* * * *
* * * *
* * * *
* * * *
```

Now, we need to write a program to print right angle triangle, using symbol *, which will have 'num' number of rows (as the value of 'num' variable is entered by the user). To do this we will consider the same program, we have discussed above. We will do small modification in it. Look at the following figure:



We have already discussed a program to draw a square. The same logic we will use to draw right angle triangle shape after some modification. From the figure, it is clear that to draw rectangle we need r= num and c=num. Now to draw right angle triangle, we need to draw 1 column (1 star) in the 1st row, 2 columns (2 stars) in the second row and so on.

So, it is clear that if we modify a loop of variable c from: 'for (c=1; c<=**num**; c++)' to 'for (c=1; c<=**r**; c++)'. So, modify the program as shown below to get the desire output.

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

❖ **Output :**

Enter Any Number:4

*

* *

* * *

* * * *

Now consider another program, which can be done by doing small change in the above program. We are taking an addition variable to adjust the spaces, that is variable sp. In each row, we have to run a loop which will print (num – r) spaces, before we are executing a loop which is printing *. In each row, if we are giving (num –r) spaces, and then we are running a loop which will print, r number of * symbols. Here, one more small change we have to do, that is, in the printf statement which actually print * we have to print a space after *. Means we need to write a 'printf("* ");'. This is a * and one space. If we are doing these

modifications in the above program, we can print triangle instead of right–angled triangle. So, to write a program, which will draw triangle, using symbol *, first we need to add a loop which will draw (num –r) spaces, and second is we have to give one space in the printf statement which is printing * (after *). Consider the following program, which is making triangle using * symbol.

```
/*Program:14 To draw square shape*/
#include<stdio.h>
void main()
{
    int num,r,c,sp;
    printf("Enter Any Number:");
    scanf("%d",&num);
    for(r=1;r<=num;r++)
    {
        for(sp=1;sp<=num–r; sp++)
            printf(" ");
        for(c=1;c<=r;c++)
        {
            printf("* ");
        }
        printf("\n");
    }
}
```

❖ **Output :**

```
Enter Any Number:4
   *
  * *
 * * *
* * * *
```

Now, in Program:15 one more modification, you have done. That is in the (Program:14) printf statement we have given * and space. Remove the space we have given after *, save compile and execute the program.

Several patterns we can print from the above Program:14. Change the printf("* "); statement as suggested in the column2 of the table given below and compare the output with column:3 of the table.

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

17	printf("%d ",c);	Enter Any Number:2 1 1 2
18	printf("%c ", r+64);	Enter Any Number:3 A B B C C C
17	printf("%c ",c+64);	Enter Any Number:3 A B B C C C
18	printf("%c ", r+96);	Enter Any Number:3 a b b c c c
19	printf("%c ", c+64);	Enter Any Number:3 a b b c c c
20	printf("%d ", r%2);	Enter Any Number:3 1 0 0 1 1 1
21	printf("%d ", c%2);	Enter Any Number:3 1 1 0 1 0 1
22	printf("%d ", (r+c)%2);	Enter Any Number:4 0 1 0 0 1 0 1 0 1 0

So, number of patterns we can generate by doing a small change in the program:14. I think, these examples are sufficient to do practice of nested for loop. Now we will do some discussion about do … while loop, and will see some examples of it.

❖ DO…WHILE LOOP :

Do…while loop is exit control loop. Here program flow control enters into the loop without checking any condition. After the execution of the body part of the loop condition is evaluated. If condition is 'TRUE' then second iteration of the loop will start, but if condition is 'FALSE' then body part of the loop will not be executed. That means, in this loop body part of the will be executed at least once. Another difference between while loop, for loop and do…while loop is do…while loop ends with semicolon (;). The syntax, flow diagram and example of the do…while loop is given below:

Syntax :

```
<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:");
    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. */**

```
#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. */**

```
#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!+.....*/**

```
#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
□ 2
1
*/
#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");
        }
}
```

/ 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++)
    {
```

```
            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 */**

```
#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 */**

```
#include<stdio.h>
void main()
{
    int n,sum=0,num,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 no for series:2
the sum of the series is:5
```

Some useful websites are given below from where above programs are referred :

http://www.comp–psyche.com/2014/04/write–c–program–to–find–fibonacci–series.html

http://wiki.answers.com/Q/C_program_to_print_factorial_of_a_number http://ecomputernotes.com/c–program/find–the–factorial–of–n–number http://www.santoshkabirsir.com/FE_SPA.html

http://www.pinterest.com/codeplusform/0–1–2–3–4–5–6–7–8–9–10–11–12–13–14–15–16–17–18–19–

http://csinfobit.com/category/java/branching–and–looping http://www.mathpages.com/home/kmath025.htm http://taranets.org/cgi/ts/1.37/ts.ws.pl?w=319;b=78 http://www2.its.strath.ac.uk/courses/c/subsection3_8_8.html http://www.asic–world.com/scripting/control_flow_c.html

BLOCK ACTIVITIES :

Activity 1

- Install C Language on your computer and practice all the above units to make your basic C programming perfect.

Activity 2

- Explain any four control statements with appropriate examples.

Activity 3

- Explain difference between while and do–while loop.

REFERENCE BOOKS :

Born to Code in C, H. Schildt

1. C Programming, Ed. 2, Kerninghan and Ritchie
2. C Programming, Balagurusamy
3. Let us C,YashwantKanetkar
4. Programming in C, S. Kochan
5. Programming in ANSI C, Agarwal
6. The Art of C, H. Schildt
7. Turbo C/C++ – The Complete Reference, H. Schildt

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.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	5	6	7	8
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	_____
Language and Style	☐	☐	☐	☐	_____
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	_____
Conceptual Clarity	☐	☐	☐	☐	_____
Check your progress Quest	☐	☐	☐	☐	_____
Feed back to CYP Question	☐	☐	☐	☐	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-103/ DCAR-103

Fundamentals of Programming Using C Language

BLOCK 3 : ARRAYS & FUNCTIONS

ARRAYS & 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 9th and 10th units. The usages of arrays help in tackling with less no. of variables.

In 11th 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 11th unit which will help you in solving problems recursively.

The 12th unit contains some solved programs based on the statements/ concepts explained in the first 11 units. This will help the learners to understand those concepts in details as the problems are practically solved.

Block Objectives :

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 :

Unit 9 : Arrays

Unit 10 : Handaling Strings

Unit 11 : Functions

Unit 12 : More About Functions

Unit
09

ARRAYS

UNIT STRUCTURE

9.0 Learning Objectives :

In this unit, we will discuss about the arrays, which are used to store large number of values in a single variable. The handling of multidimensional arrays and array operations are also explained in this unit

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

- Store a large no. of values in a single variable.
- Create (Declare) and initialize arrays.
- Know various terms related to the array.
- Performing various operations on array.
- Use Two–Dimensional arrays.

9.1 Introduction to Arrays :

Arrays are finite and ordered collection of homogeneous data. Homogeneous means same type of data. As arrays are collection of the data, we can store more than one element in the array under one common name, but make sure all the elements are of same type.

9.2 Understanding Arrays :

9.2.1 Defining Array :

If in the C–Language, you write ***'int x [10];'*** in the declarative section of the program then you have a single variable (array) named 'x', which can store 10 elements of same type ('int' in this case) for you.

9.2.2 Initializing Array :

We can initialize arrays in three different ways :

1. Initializing array at the time of its declaration :

```
int x [10] = {5,10,15,20,25,30,35,40,45,50};
char y [5] = {'A', 'E', 'I', 'O', 'U'};
```

in the above examples, 'x' is an integer array which is initialized by 10 data elements i.e. 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50. First data element '5' will be stored on the first 0th position of the array x. That means x [0] is '5'. Similarly, 10 is stored on 1st position, 15 is stored on 2nd position and so on. Finally, data element 50 will be stored on 9th position. In C–Language array index starts from 0. So, last element we can find on the position Size – 1. Here, the size of array 'x' is 10. So, last data element '50' will be stored on position 9.

In another example, we have declared an array 'y' of type character, which is initialized by five, character typed data elements 'A', 'E', 'I', 'O' and 'U'. First data element 'A' will be stored on 0th position and 'U' will be stored on 4th position

2. Declaring array and initializing it with some static values :

```
int x [5];
char y [5];
x [0] =5;
x [1] = 10;
x [2] = 15;
```

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

```
y [0] = 'A';
y [1] = 'E';
```

and so on.

3. Declaring and initializing array by user input :

```
int x [10] , i ;
for (i=0; i< 10; i++)
{
        printf ("Enter Any Number:");
        scanf("%d", & x[i]);
}
```

❑ **Check Your Progress – 1 :**

1. Array Index always starts from _____.

   [A] 0 [B] 1 [C] 10 [D] 5

2. Array is a _____ collection of data.

   [A] Heterogeneous [B] Homogeneous

   [C] Different types of [D] None of the above

3. If int x[10] is declared. The last element of an array x will be on index number _____.

   [A] 0 [B] 1 [C] 10 [D] 9

9.2.3 Terminologies of an Array :

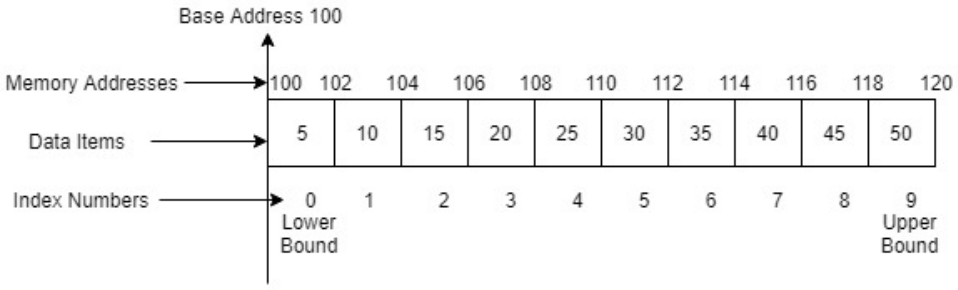
1. **Type :** Type of an array represents type of data can be stored in the array. For example, array can be of type int, float, char, double etc.

2. **Size :** Size of an array represents maximum number of data elements that can be accommodated in the array. For example, if we declare 'int x [10]' then array 'x' can accommodate maximum 10 elements. So, we can say size of array x is 10.

3. **Index :** Each element stored in the array has unique reference number is called index number. It is denoted in the square brackets like [ ]. As we have discussed in the above example x [0] = 5, means data element 5 is stored in at the index number 0 of the array 'x'.

4. **Range :** Range is a set of all valid index numbers. We know that in C–Language array index starts from 0, it is called Lower bound of the array. If we declare array 'int x [10];' then last element we can be placed on 9th position. It is called upper bound of an array. Set of all possible integers from lower bound to upper bound is called range of an array. So, the range of array x is 0 to 9.

5. **Base :** Starting memory location of an array is called base address of an array. In C–Language array name itself refers base address of an array.



In the above figure memory representation of an integer single dimension array is shown. Array of 10 integer values are stored in the consecutive memory location starting 100 to 120. 100 is the starting address of the array is called base address. Because type of an array is integer and we know that, each integer occupies 2 bytes of space in the memory, first data element of the array i.e. '5' will be stored from memory address 100 to 102. Second value 10 will be stored from 102 to 104 and so on. Index number of first data element 5 is 0 is called lower bound of the array and similarly last element is stored on position 9 is called upper bound of an array.

Let we have an array having Lower bound LB and upper bound UB. Then the index number of ith element will be always: Index (Xi) = LB + i –1. For example, in C–Language index number of 5th element will be 0 + 5 –1 =4.

Similarly, size of an array is: Size = UB – LB + 1. In C–Language if UB of and array is 9 the size = 9 – 0 + 1 =10.

The given figure on previous page, will explain all the terms discussed above. Array can be One – dimensional (Linear), Two – dimensional (Matrix) or Multi – dimensional. One – dimensional and Two – dimensional arrays are discussed below in details.

❑ **Check Your Progress – 2 :**

1. Array is represented in C–Language with _____.

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

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

2. For a long integer array of 10 elements, its base address is 750, then address of its 5th element will be _____.

   [A] 770 [B] 760 [C] 758 [D] 766

3. If int x[10] is declared. Then memory size required for array x is _____.

   [A] 10 Bytes [B] 20 Bytes [C] 40 Bytes [D] 5 Bytes

4. Size of the array is: _____ if LB ≠ 0.

   [A] UB–LB–1 [B] UB–LB+1 [C] UB+LB–1 [D]UB+LB+1

9.3 One–Dimessional Array :

One–dimensional array is an array, which requires only one index or subscript number to reference any element stored in the array. Array presented in the figure in the previous page, is one–dimensional array. Value and Address of any element having index 'i', in the array 'X' can be found by following equations:

Value of element can be accessed X [i] (1)

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

For example, in the integer array whose base address is 100, address of element located on index number 4 is: 100 + 4 * 2=108. Here 100 is the base address of an array, 4 is the index number of an element and because of the type on an array is integer size of element will be 2.

9.4 Operations on Arrays :

Different types of operations can be performed on the array such as, Traversing, Insertion, Deletion, Searching, Sorting, Merging etc.

9.4.1 Traversing :

Accessing each element of an array is called traversing operation. Consider the following program in which we are printing all the values stored by the user.

```
#include<stdio.h>
void main()
{
    int x[10]={2,9,11,28,34,45,57,78,83,90};
    int i;
    printf("\nArray X contains: ");
    for(i=0;i<10;i++)
    {
        printf("%d\t",x[i]);
    }
}
```

❖ **Output :**

Array X contains : 2 9 11 28 34 45 57 78 83 90

In the above program we have declare an integer array 'x' having 10 integer values. Using variable 'i' and a for loop we have printed each element of the array. Accessing all the elements of the array for printing or any other calculation purpose is called traversing the array.

9.4.2 Insertion in the Array :

Consider an array having some values are inserted and other cells are empty (having value 0). Now suppose user want to insert a new data element on potion 4. In this case first we copy 4th element into some variable (tmp). On the 4th position we will place, the number user wants to insert. Finally, we copy the value of tmp variable to num variable. The same process is repeated till end of the array. Mean on the 4th position, new elements will be inserted, 4th element will be placed on 5th position, 5th element will be placed on 6th position and so on. So, insertion, requires all elements to be shifted at right hand side.

```
#include<stdio.h>
void main()
{
    int arr[10]={5, 10, 15, 20, 25, 30, 0, 0, 0, 0};
    int num, i, tmp, pos;
    printf("Enter Position:");
    scanf("%d", &pos);
    printf("Enter Element:");
    scanf("%d",&num);
    printf("Array Before Insertion:\n");
    for(i=0;i<10;i++)
        printf("%d\t", arr[i]);
    for(i=0;i<10;i++)
    {
```

```
            if(i>=pos –1)
            {
                tmp=arr[i];
                arr[i]=num;
                num=tmp;
            }
        }
         printf("\nArray After Insertion:\n");
        for(i=0;i<10;i++)
            printf("%d\t", arr[i]);
    }
```

❖ **Output :**

Enter Position:4

Enter Element:18

Array Before Insertion :

5 10 15 20 25 30 0 0 0 0

Array After Insertion :

5 10 15 18 20 25 30 0 0 0

9.4.3 Deletion in the Array :

Deletion is the reverse process than insertion. When any value is deleted from the array then all the right–hand side elements are shifted to the left–hand side by 1 position. The program of deletion from an array is given in the following example.

```
#include<stdio.h>
void main()
{
    int arr[10]={5, 10, 15, 18, 20, 25, 30, 0, 0, 0 };
    int i, tmp, pos;
    printf("Enter Position:");
    scanf("%d", &pos);
    printf("Array Before Deletion:\n");
    for(i=0;i<10;i++)
        printf("%d\t", arr[i]);
    for(i=0;i<9;i++)
    {
         if(i>=pos–1 )
        {
            arr[i]=arr[i+1];
        }
```

```
    }
    arr[9]=0;
    printf("\nArray After Deletion:\n");
     for(i=0;i<10;i++)
          printf("%d\t", arr[i]);
}
```

❖ **Output :**

Enter Position:4

Array Before Deletion :

5 10 15 18 20 25 30 0 0 0

Array After Deletion :

5 10 15 20 25 30 0 0 0 0

9.4.4 Searching in the Array

In the case of searching, user will input the element and we have to compare each element of an array (stating from the position 0) with the element entered by the user. Whenever, we find the exact match we have to stop the process and the position of that element into the array is displayed to the user. The process of searching an element in the array by comparing each element of the array (starting from 0) is called Linear search.

```
#include<stdio.h>
void main()
{
    int arr[10]={5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };
    int i, num;
    printf("Enter Search Element:");
    scanf("%d", &num);
    printf("Array:\n");
    for(i=0;i<10;i++)
          printf("%d\t", arr[i]);
    for(i=0;i<10;i++)
    {
          if(arr[i]==num )
          {
                printf("\nElement Fond on: %d position", i+1);
                break;
          }
    }
     if(i==10)
    {
          printf("\nSearch element do not exists in the Array ");
    }
}
```

❖ **Output :**

Enter Search Element:25

Array :

5 10 15 20 25 30 35 40 45 50

Element Fond on: 5 position

9.4.5 Sorting an Array :

Sorting is the process of arranging array elements either in the ascending (from lower to higher) or descending (from higher to lower). Consider the following program, to sort elements of an array in the ascending order:

```
#include<stdio.h>
void main()
{
    int arr[10]={5, 23, 78, 90, 30, 76, 28, 20, 48, 2 };
    int i, j, tmp;
    printf("Array Before Sorting:\n");
    for(i=0;i<10;i++)
        printf("%d\t", arr[i]);
    for(i=0;i<10;i++)
    {
        for(j=i;j<10;j++)
        {
            if (arr[j]<arr[i])
            {
                tmp=arr[i];
                arr[i]=arr[j];
                arr[j]=tmp;
            }
        }
    }
    printf("\nArray After Sorting:\n");
    for(i=0;i<10;i++)
        printf("%d\t", arr[i]);
}
```

❖ **Output :**

Array Before Sorting:

5 23 78 90 30 76 28 20 48 2

Array After Sorting:

2 5 20 23 28 30 48 76 78 90

The algorithm we have used to sort an array is called selection sort. Other sorting algorithms are also there like bubble sort, insertion sort etc.

Here we are sorting elements of an array in the ascending order. If you wish to sort them in the descending order then change the if condition we have written in the program from: *if (arr[j]<arr[i]) to if (arr[j]>arr[i]).*

❑ **Check Your Progress – 3 :**

1. _____ is a process of arranging array elements into some order.

   [A] Searching [B] Sorting [C] Inserting [D] Deletion

2. To know the position of specific element in the array, _____ process is used.

   [A] Searching [B] Sorting [C] Inserting [D] Deletion

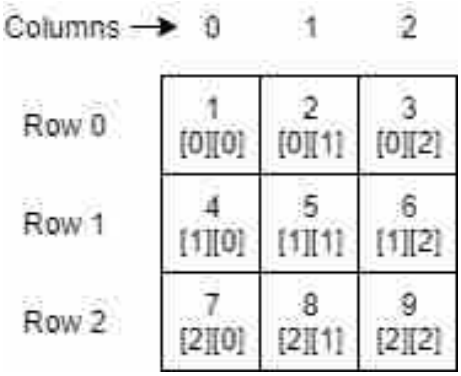
3. _____ process of an array requires shifting of elements.

   [A] Searching [B] Sorting [C] Insertion [D] All of the above

9.5 Two–Dimenssional Array :

In the above examples we have declared array like: 'int x[10];'. When we declare array like this, then array 'x' has 10 rows or we can say, the array is one–dimensional (rows). But if we declare array like: 'int y[3][3];' then array 'y' has 3 rows and in each row, 3 columns are there.

That means array 'y' can store 9 elements. In this case we can say that array 'y' is 2–dimensional array (rows, columns). To represent matrix, we need a 2–dimension array. 2–dimesional array can be represented as follows :



In the above array we have stored the values 1, 2, 3, …9. Value 1 can be accessed by y[0][0]. Similarly 2 can be accessed as y[0][1], 3 can be accessed by y[0][2], 4 can be accessed by y[1][0] and so on. Usually two–loops (nested) are used to access matrix (2–dimensional array). Consider the following program, which will do the addition of 2 matrices (2–D Arrays) of size 3*3.

9.5.1 Addition of Two Matrices :

The process of adding two matrices are quite simple. Here, first element of the first row, has to be added in the first element of the first row of second matrix, and place it in the first element of the first row of third matrix. The process continues for all elements, which is programmatically shown below:

```
/* 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 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]);
                }
}
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");
 }
```

```
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 :

12 14 16

18 20 22

24 26 28

❑ **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

9.5.2 Multiplication of Two Matrices :

Multiplication of matrices is little bit complex process. In this all elements of first row of first matrix, has to be multiplied with all elements of first column of the second matrix, and its sum will be first element of first row of the resultant matrix.

That means,

C[0][0] =A[0][0]*B[0][0] + A[0][1]*B[1][0] + A[0][2] * B[2][0]

C[0][1] =A[0][0]*B[0][1] + A[0][1]*B[1][1] + A[0][2] * B[2][1]

C[0][2] =A[0][0]*B[0][2] + A[0][1]*B[1][2] + A[0][2] * B[2][2]

C[1][0] =A[1][0]*B[0][0] + A[1][1]*B[1][0] + A[1][2] * B[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;
        }
    }
```

```
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

9.5.3 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 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 value is 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. If int x[3][4][5] is declared, it can accommodate _____ elements.

   [A] 24 [B] 60 [C] 120 [D] 12

2. Memory space required to accommodate float x[3][4][5] is _____ Bytes.

   [A] 60 [B] 120 [C] 24 [D] 240

9.6 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.

9.7 Suggested Answers for Check Your Progress :

❑ **Check Your Progress 1 :**

1. [A] 0 2. [B] Homogeneous
3. [D] 9

❑ **Check Your Progress 2 :**

1. [A] [ ] Square brackets 2. [D] 766
3. [B] 20 Bytes 4. [B] UB–LB+1

❑ **Check Your Progress 3 :**

1. [B] Sorting 2. [A] Searching
3. [C] Insertion

❑ **Check Your Progress 4 :**

1. [B] 2–Dimentional 2. [B] 2–Dimentional
3. [A] 18, 9

❑ **Check Your Progress 5 :**

1. [B] 60 2. [D] 240

9.8 Glossary :

1. **Array :** It is a homogeneous (same type of) collection of data.

2. **Base Address :** It is a starting memory location of an array.

3. **Lower Bound :** It is an index number where array stores its first element. Usually in C–Programming it is 0.

4. **Upper bound :** It is an index number where last element of the array is stored. Usually it is size –1.

9.9 Assignment :

1. What is Array ? How can we declare and initialized it ?
2. Discussed the array operation in details.
3. Discuss, how can we represent a matrix in the C–Language ?

9.10 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.

9.11 Case Study

Write a program to find transpose of given 3*3 matrix.

9.12 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 10

HANDLING STRINGS

UNIT STRUCTURE

10.0 Learning Objectives :

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

- Understand string as character–based array.
- Know how to declare and initialize string.
- Use various I/O functions like printf, scanf, puts, gets etc.
- Know how strings can be formatted.
- Understand different functions to process string.

10.1 Introduction :

In any programming language, to communicate with the user, we need to use sentences. Programming language also contains statements. Sentences or statements are formed by words. These words are known as strings in the programming languages. In C–Language, a sequence of characters is known as string. C–Language do not have string data type, where other programming languages like C#.NET, JAVA, VB.NET and many other programming languages has string data type. To handle strings, in C–Language we use array of character datatype. That means in the C–Language, to represent a string we need to declare character array, and as we know elements of the array (different characters in the case of string) are always stored in the contiguous memory locations.

In C–Language, we use character type 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 university in the string then, we need to declare character array 'uni_name' as follows:

char uni_name [ ] = {'B', 'A', 'O', 'U', '\0'};

All the characters of a strings are stored as a character in the separate memory location (as shown in the following table), in contiguous manner, and the last character is NULL, which is special character '\0', which indicate end of the string.

Character Stored	B	A	O	U	\0
Memory Address	3001	3002	3003	3004	3005

You can also specify size at the time of declaration of string. For example,

char state_name [8] = "GUJARAT";

This is same as:

char state_name [8] = {'G', 'U', 'J', 'A', 'R', 'A', 'T', '\0'};

❑ **Check Your Progress – 1 :**

1. Strings are nothing but group of _____.

   [A] floats [B] Booleans

   [C] characters [D] None of the above

2. A character is occupying _____ memory.

   [A] 2 Bytes [B] 4 Bytes [C] 8 Bytes [D] 8 Bits

3. In C–Language string ends with _____.

   [A] @ [B] '\0' [C] \n [D] $

10.2 Understanding Strings :

In the prevision section, we have discussed that the strings are group of characters, and in the C–Language, we can represent it using array of type character. We have also discussed, a special character '\0' which is also known as NULL character is used to indicate end of the string.

10.2.1 Declaring and Initializing String :

To declare the string variable, we need to declare an array of type character. To initialize the string variable, we can use assignment operator =, followed by string value (group of characters) encased in double quotation mark as shown below :

char state_name [ ] = "Gujarat";

In this type of declaration, C–Compiler automatically append end of the string mark, that is NULL ('\0') character at the end of the string. It will automatically calculate the size, and declare the array which can accommodate all string characters including NULL character.

Now, consider the following C–Program, in which we have declared 2 string variable str1 and str2. We are storing string "Gujarat" in both the variable. we are keeping the size of str1 array is of 7 and str2 array is of 8.

```
#include<stdio.h>
void main()
{
    char str1[7]={'G','U','J','A','R','A','T'};
    char str2[8]={'G','U','J','A','R','A','T'};
    printf("String1: %s",str1);
    printf("\nString2: %s",str2);
}
```

❖ **Output :**

String1: GUJARAT<

String2: GUJARAT

In this program, str1 array having size 7 and the string we have stored in the is "Gujarat", also having 7 characters. In this case, we don't have extra space to store NULL character at the of string, as a result, when we print str1 variable, some extra junk character(s) will be printed on the console screen. In the case of str2 array, size is 8 and number of characters are stored 7. Here complier can place NULL character at the end of the string (because of extra space for one character), as a result str2 will be perfectly print string on the console.

10.2.2 Using printf and scanf functions with String :

We have already discussed, strings can be printed using printf() function, using format string "%s". Consider the following program, which will print the given string on the console using printf() statement.

```
#include<stdio.h>
void main()
{
    char msg[]="Hello";
    printf("%s",msg);
}
```

❖ **Output :**

Hello

In this program we have declared, character array msg and initialized it with string value "Hello". In the next statement, we are printing the string using printf() statement and "%s" format string. Here we are getting desire output that is "Hello" on the console screen. Now, consider another program as given below:

```
#include<stdio.h>
void main()
{
    char msg[]="Hello How Are You ?";
    printf("%s",msg);
}
```

❖ **Output :**

Hello How Are You ?

So, by looking to this program, we can say that the printf() functions can also print those strings, which includes spaces. Now, consider the following program, in which we are printing 2 strings str1 and str2, but its values are taken from the user (rather than initializing it with static values) using scanf() function.

```
#include<stdio.h>
void main()
{
    char str1[20],str2[20];
    printf("Enter String1:");
    scanf("%s",str1);
    printf("Enter String2:");
    scanf("%s",str2);
    printf("String1: %s",str1);
    printf("String2: %s",str2);
}
```

❖ **Output :**

Enter String1: BAOU

Enter String2: Hello How Are You ?

String1: BAOU

String2: Hello

Note that in the first–string variable we have entered string "BAOU", which do not include any space. In the second–string variable we have stored "Hello How Are You ?", if we print both strings on the console, we get word "BAOU" will be printed properly, but in the str2 variable, it will print only "Hello" world. That means scanf() function cannot scan those strings, which includes spaces. In the case of str2 variable scanf() function, has scanned only Hello word, and skipped all those characters came after first space in the string.

❑ **Check Your Progress – 1 :**

1. To accept the strings, which functions can be used ?

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

   [C] Both A and B [D] puts()

2. _____ function cannot handle spaces in the string.

   [A] scanf() [B] printf() [C] gets() [D] puts()

10.2.3 Using puts and gets functions with String :

In the previous program, we have seen that scanf() function fails to accept those strings, which includes space. In this case we can use puts() and gets() functions. Consider the following example in which we have an array str. We are accepting string value from the user, using gets() function, and finally we are printing the value of the string on the console screen using puts() function.

```
#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 ?

From the above example, we can say that puts() and gets() functions, can easily takes and print those strings, which has spaces. Functions puts() and gets() are specifically design to handle strings, whereas printf() and scanf() functions can print and scan any type of data for example, int, char, float etc. puts() and gets() are known as unformatted functions, as they do not take any format strings like "%s", "%c", "%d" etc. As in the printf() and scanf() functions, we need to pass, format string, to specify format and type, they are called formatted functions.

10.2.4 Using EOF :

In the previous section, we have discussed that scanf() function cannot accept those strings, which has space. So, scanf() function is suitable, to scan only words, and not sentence. To scan sentence (multiple words, separated by spaces), we need to use gets() function. But gets() function cannot accept Enter (New Line). Means, gets() is suitable to take one line sentence. If you want to store two or more sentences separated by Enter (New Line) then gets() fails. To deal with this problem, we are taking one by one character from the user and putting it in the character–based array using loop, this will allow user to input spaces and also enters (New Lines). When user presses Ctrl+Z and then Enter, the loop stops taking any new character. This special character of Ctrl+Z is known and EOF (End of File) character. Consider the following program, in which we have declared a relatively large sized array str. Using while loop, we are taking one by one character from the user and stored it in the str variable. At the end user will give EOF character by pressing Ctrl+Z keys followed by Enter.

```
#include<stdio.h>
void main()
{
    char str[250],ch;
    int i=0;
    printf("Enter String:\n");
    while(((ch=getchar())!=EOF))
    {
```

```
            str[i]=ch;
            i++;
        }
        str[i]=EOF;
        i=0;
        printf("\nThe String you have Entered is:\n");
        while((ch=str[i])!=EOF)
        {
            printf("%c",ch);
            i++;
        }
    }
```

❖ **Output :**

Enter String:
Hello, How are you ?
We have stored, more than one
lines in the string variable.
^Z

The String you have Entered is:
Hello, How are you ?
We have stored, more than one
lines in the string variable.

❑ **Check Your Progress – 1 :**

1. To accept the strings having spaces, which functions can be used ?
   [A] scanf() [B] gets()
   [C] Both A and B [D] puts()
2. To accept the string till user is not entering Ctrl+Z, _____ is used.
   [A] EOS [B] EOE [C] SOE [D] EOF
3. To accept the sting from the user having spaces and new line _____ is used.
   [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

10.3 Displaying String in Different Formats :

We know that, function printf() is a formatted output function. Formatted output function, format the data based on the format string passed by the user and represent the data in the various formats. Here, are some examples of the

different format strings used with printf() statements, and its output in the table. Consider an array: ***char str[15]="UNIVERSITY".***

Sr. No	printf() statement	Output
1	printf("%s",str);	UNIVERSITY
2	printf("%.6s",str)'	UNIVER
3	printf("%.9s",str);	UNIVERSIT
4	printf("%.15s",str);	UNIVERSITY
5	printf("%–10.5s",str);	UNIVE
6	printf("%14s",str);	UNIVERSITY

1. In the first statement format string "%s" is used with printf() statement. This will print whole string "UNIVERSITY" on the console screen.
2. In the second statement we are using "%.6s" format string. In this statement we are specifying precision (the number of characters to be displayed) after the decimal point. As a result, it will only print first six letters of the string.
3. In the third statement, precision is 9, therefore 9 characters will be printed on the screen.
4. In the fourth statement, whole string gets printed on the console screen as the precision is higher than number of characters stored in it.
5. In the fifth statement, format string "%–10.5s" is used. This means that the string will occupies 10 characters on the console, .5 indicates 5 characters will be printed, and these 5 characters out of 10–characters add spaces on the console, aligned left hand side (because of – sign). That means it will print UNIV followed by 5 spaces. If you write another statement called 'printf("Hello");' immediately after the above printf statement, then you will get "UNIVE Hello". There are 5 spaces are there between 'UNIVE' and 'Hello'.
6. In the sixth statement, format string "%14s" is used. Positive number indicated string will be right hand side aligned, with space of 14 characters on the console. Because of we have only 10 characters, in the string 'str', it will print 4 spaces, and then UNIVERSITY.

❑ **Check Your Progress – 4 :**

1. Statement printf("%.9s", " UNIVERSITY"); will print _____.

   [A] UNIVERSITY [B] UNI

   [C] UNIVER [D] UNIVERSIT

2. Statement printf("%–7.4s", "BAOU); will print _____.

   [A] BAOU<space><space><space> [B] AOU

   [C] <space><space><space>BAOU [D] BAOU

3. Statement printf("%7.4s", "BAOU); will print _____.

   [A] BAOU<space><space><space> [B] BAO

   [C] <space><space><space>BAOU [D] BAOU

10.4 Standard Functions for String Handling :

In this section, we will discuss, some basic functions available in the C–Language to handle strings. These functions are built–in functions (library

function) and you do not need to write any logic. Just you need to include the header file 'string.h' and you can use any of the function which is described in the table given below. For an Example :

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str[]="indian";
    int l;
    l=strlen(str);
    strupr(str);
    printf("Length of the string is: %d",l);
    printf("\nUpper case string is: %s",str);
}
```

❖ **Output :**

Length of the string is: 6

Upper case string is: INDIAN

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 strlen() is used to determine the length of the string.
strcpy()	This function is used to copy a source string to destination.
strncpy()	This function is used to copy first n characters of source to destination.
strcmp()	This function is used to compare two strings. In this comparison lower case and uppercase alphabets are considered to be not same.
stricmp()	This function is used to compare two strings, by ignoring case.
strncmp()	This function will compare first n characters of two strings
strnicmp()	This function will compare first n characters of two strings by ignoring case.
strlwr()	This function will convert the string in lower case.
strupr()	This function will be used to convert the string in the upper case.
strcat()	This function is used to concatenate two strings
strren()	This function will reverse the given string
strstr()	Determines the first occurrence of a given string in another string.

Make sure in order to use the function, described in the table above, you must need to include header file 'string.h' in your program. The detail descriptions of these functions are given as follows:

Most C compilers include library functions that allow strings to be compared, copied or concatenated. Other functions permit operations on individual characters within strings. For example, they allow individual characters to be found within strings and so on. The following example illustrates the use of some of these library functions:

1. strlen():

This function counts a number of characters presents in a string. While giving a call to the function we have to pass the base address of the string. Total numbers of characters of the stings are counted without counting the null character, returning the length of the string.

```
static char msg[] = "KameshRaval"
int n;
n=strlen(msg);
printf("length of string =%d", n);
```

The output will be length of string=11.

2. strcat():

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

```
static char s[] = "BabasahebAmbedkar";
static char t[] = "OpenUniversity";
strcat(t,s);
printf("source string is %s\n", s);
printf("target string is %s\n", t);
```

The final output is source string s is BabasahebAmbedkar

target string t is BabasahebAmbedkarOpenuniversity.

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.

```
static char source[] = "Gujarat"
static char target[15];
strcpy(target, source);
printf("Source string is: %s", source);
printf("target string is:%s",target);
```

Now string target has characters Gujarat.

4. strrev(string):

Function is used to reverse the given string. For Example,

```
str = "abcde";
strrev(str);
printf("%s",str);
```

This will print – edcba

5. strlwr ():

This function converts all characters in a string from uppercase to lowercase.

strlwr(string);

For example:

strlwr("BCA FIRSTYEAR") converts to bca first year

6. strcmp()

This function compares two strings to find out whether they are the same or different. The process of character wise comparison continues till a mismatch is attained or till the string ends. If two strings are identical, it returns a zero value, else it returns a numeric difference between ASCII values of non– matching characters.

This function compares two strings and returns some integer value. If both these strings are equal then it returns 0 otherwise it returns some other integer value.

```
void main()
{
    char s1[]="Divine ";
    char s2[]="Akshar";
    char s3[]="Divine Akshar"
    int i, j, k;
    i = strcmp(s1, "Divine");
    j= strcmp(s1, s2);
    k=strcmp(s1, s3);
    printf("%d %d %d", i,j,k);
}
```

❖ **Output :**

1 1 –1

Because, "Divine" is compared with "Divine". First string is bigger than second so function will return 1. In second comparison, first string "Divine" is compared with "Akshar". Number of characters are same in both strings but first letter 'D' of first string is bigger than first letter 'A' of second string. Therefore, first string is bigger than second and it will produce 1. In third comparison "Divine" with "Divine Akshar". Here, second string is bigger than first, hence strcmp function will return –1.

❑ **Check Your Progress – 5 :**

1. strcmp("abcd","ABCD"); will return _____.

   [A] 1 [B] 32 [C] –1 [D] 0

2. strcmp("abcd","abcd"); will return _____.

   [A] 1 [B] 32 [C] –1 [D] 0

3. To convert the string into lower–case, _____ function is used.

   [A] strlower() [B] lower()

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

4. To reverse the given string, _____ function is used.

   [A] reverse() [B] stringreverse() [C] strrev()
   [D] stringrev()

10.5 Table of Strings :

We can declare two dimensional arrays of characters. For example, we could write :

char countries [6][20] = {"India", "China", "Srilanka", "Nepal", "Bangladesh", "Pakistan"};

The size of the left index determines the number of strings and the right index specifies the maximum length of each string.

10.6 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.
- ave discussed about EOF, and table of strings.

10.7 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

❑ **Check Your Progress 4 :**

1. [D] UNIVERSIT

2. [B] BAOU<space><space><space>

3. [C] <space><space><space>BAOU

❑ **Check Your Progress 5 :**

1. [A] 1 2. [D] 0

3. [C] strlwr() 4. [C] strrev()

10.8 Glossary

1. **String :** It is a group of characters. In C–Language it is represented by array of type character.
2. **'\0' :** It is an identification mark of string end. It is also known as NULL character.
3. **Concatenation :** It is a process of appending one string to other string.

10.9 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.

10.10 Activity :

Write a program to reverse the string with using strrev() function.

10.11 Case Study :

Write a program to store name of your friends in the table of string and print them.

10.12 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
11

FUNCTIONS

UNIT STRUCTURE

11.0 Learning Objectives :

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

- Understand the importance of using functions.
- Know the methods of declaring and using functions.
- Write programs using functions.
- Know about return values and their types.
- Understand the different types of functions.

11.1 Introduction :

In this unit, we will discuss about functions, its need and types. Functions play an important role to reduce the number of statements in a program. Whenever the same set of statements are repeated in a program a number of times, then we use functions. Functions have return types, which specifies the type of value returned.

11.2 Need For User Defined Functions :

A function is a block of code that has a name and it has a property that it is reusable i.e. it can be executed from as many different points in a C Program as required.

A 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 computer program cannot 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 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 which are 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.

We can divide a long C program into small blocks which can perform a certain task.

Why 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 to 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 course of the program.

Two reasons :

- Using functions avoids rewriting the same code over and over. Suppose that there is a section of code in a program that calculates the area of a triangle. If, later in the program we want to calculate the area of a different triangle we won't like to write the same instructions all over again. Instead we would prefer to jump to a "section of code" that calculates area and then jump back to the place from where you left off. This section of code is nothing but a function.
- Using functions, it becomes easier to write programs 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 independently. Separating the code in to modular functions also makes the program easier to design and understand

❑ Check Your Progress – 1 :

1. Functions are _____.

   [A] set of variables [B] set of data types

   [C] set of operators [D] set of executable statements

2. Readily available functions, which we can use by including proper header file is called _____.

   [A] built–in functions [B] user defined functions

   [C] Both A and B [D] None of the above

3. Functions which is designed by programmer is called _____.

   [A] built–in functions [B] user defined functions

   [C] primitive functions [D] None of the above

4. We use functions, because _____.

[A] to increase readability of program

[B] to reduce program complexity

[C] to save memory

[D] All of the above

11.3 A Multifunction Program :

Consider the given example :

```
void main( )
{
        printf("\n I am in main");
        display( );
        show( );
        putdata( );
}
void display( )
{
        printf("\n Computers"):
}
void show( )
{
        printf("\n I am studying C");
}
void putdata( )
{
        printf("\n We are studying computers");
}
```

The output of the above program when executed would be

I am in main

Computers

I am studying C

We are studying computers

From the above program, the following conclusions can be drawn :

- Any C program can have at least one function.
- If there is only one function, it can be main( )
- If multiple functions are present, then program execution should always begin from main( ).
- There is no limitation on the number of functions present in a C program.
- After the execution of each function, the control of the function returns to the caller function (A function from which other function is called).

3.4 The Form of C Functions :

The general form of a function is :

Return–type function–name(parameter list)

{

Body of the function

}

The return–type specifies the type of data being returned by the function. A function can return any type of data except an array. The parameter list is a comma separated list of variables along with their data types. A function can be without parameters, in that case, the parameter list will be empty. The parameter list if present, can be written as given below:

Return–type function–name(type varname1, type varname2,…,type varnameN)

❑ Check Your Progress – 2 :

1. User Defined Function starts with _____.

   [A] body of the function [B] functions name

   [C] return type [D] argument list

2. C–Program always start its execution with _____ function.

   [A] begin() [B] main()

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

3. From the following statements which is false _____.

   [A] In C–Program main() is compulsory.

   [B] Function name should not be a keyword.

   [C] Function starts with return type.

   [D] In a one program more than one main() functions can exist.

11.5 Return Value and Their Types :

All functions except those of type void return a value. The value to be returned by a function is specified by the return statement. If with a function name, return type is associated then the function should return a value using the return statement.

While writing programs, your functions can be generally of three types. The first type is simply computational. These functions are specifically designed to operate on their arguments and return a value based on that operation. For example, sqrt( ) and sin( ) functions, which computes the square root and sine of their arguments.

The second type manipulates information and returns a value that indicates the success or failure of that manipulation. For example, fclose( ) function which is used to close a file. If the operation is successful, it returns 0 else returns EOF.

The third and the last type of function is that function which has no explicit return value. For example, exit( ) which terminates a program. All those functions which don't return values should be declared with their return type as void.

❖ **Remember following points :**

- Every C–Program has one main() function. More than one main() should not be there. C–Program always starts from the main() function.
- User defined function always start with return type. If no return type is mentioned then default return type int is considered.
- Function must starts with either alphabets or underscore '_'. Function name never starts with digits. For Example, abc123 is a valid function name but 123abc is not a valid name for function.
- Function name should not be a keyword. Keywords are reserved words and can never be used as a function name (identifier).
- Space or special symbols are not allowed in the name of the function.
- Function always return value or control, to the function who has called it (caller function).
- Function can return only one value. To return multiple value either structure or pointer is used.
- If function do not return any value, then its return type has to be 'void'.

❑ **Check Your Progress – 3 :**

1. Default return type of the function is _____.

   [A] void [B] char [C] int [D] double

2. If function is not returning any value, then its return type is _____.

   [A] void [B] char [C] int [D] double

3. From the following statements which is false _____.

   [A] Function name always start with alphabets or underscore.

   [B] Function name should not be a keyword.

   [C] Default return type of the function is void.

   [D] Function can return only one value.

11.6 Calling of Function :

A function can be called after it is declared in a program. As if a function is not declared and is called then the compiler will not return that particular function and will generate an error by not recognizing the function. The program given below will illustrate the same:

```
#include <stdio.h>
void sum(int, int);          //Function prototype declaration
void main( )
{
      int x, y;
      printf("Enter the values of x and y");
      scanf("%d%d",&x,&y);
            sum(x, y);    //Calling of function
}
```

```
void sum(int a, int b)     //Function definition
{
        int c=0;
        c=a+b;
        printf( "sum is %d", c);
}
```

In the above program, in the main ( ) function, sum( ) function is called and the values of x and y are passed. The parameters specified at the time of calling of the function are called actual parameters whereas, while defining the body of the function the arguments / parameters which are specified are called formal parameters, that is, a and b are formal parameters.

Here, in the previous example we have defined the function sum(), after main() function, therefore after including header file, it is necessary to declare the function. This function declaration is called prototype declaration of the function. Prototype function declaration is essential if function is defined after main() function, but suppose if we define the sum() function before main() function in that case prototype declaration is not required.

❑ **Check Your Progress – 4 :**

1. Prototype declaration is needed for the function, which is _____.

   [A] declared before main(). [B] declared after main().

   [C] declared in another file. [D] very large.

2. Arguments passed at time of calling a function is called _____ argument.

   [A] formal [B] normal [C] actual [D] dummy

3. Arguments passed at time of function declaration is called _____ argument.

   [A] formal [B] normal [C] actual [D] dummy

11.7 Category of Functions :

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.

11.7.1 Functions with No Argument And No Return Values :

C functions can be of no argument and no return value types. In order to illustrate the same, let us understand the same type of functions with the help of an example,

```
#include<stdio.h>
void printhello(); //Prototype Declaration
void main( )
{
        printhello(); //Function call
}
```

```
void printhello() //Function defination
{
        printf("Hello, BAOU Students\n");
}
```

In the above program, we have defined a function called printhello(). The function is just having a message for the students that is "Hello, BAOU Students". We know that after compilation, execution of the program start from main() function. When system starts, executing main() function, main() function is calling to function printhello(). When the main() function is calling to some function, control will transfer to that function, and system will start to executing that function. As a result, we get the message "Hello, BAOU Students" on the console.

Now, think what happed if we place function calling line in the main() function in the loop ? (just try it) In that case, suppose if loop runs for 5 times, then main() function will call to printHello() function for 5 times, and message will be printed for 5 times.

In this example, our user defined function printhello(), is not returning any value to the caller function main(). Therefore, the retrun type of the function is 'void'. In the same way, at the time of calling, main() function is not passing any actual paraments to the function (see empty parenthesis at the time of function calling). Here user defined function, printhello() is not returning any value and it doesn't take any argument.

11.7.2 Argument but No Return Value :

Now, we will discuss another example which has arguments but no return values.

```
void sum (int, int);
void main( )
{
        int a=5, b=6;
        sum(a,b);
}

void sum(int x, int y)
{
        int z=0;
        z=x+y;
        printf("Sum is %d\n",z);
}
```

In the above program, sum function is a type of function with no return type and arguments. As with sum ( ), void data type is specified and it is accepting two integer types of arguments, so we can say that sum( ) is a function with no return type and with arguments.

11.7.3 Arguments with Return Value :

Given below is an example, which illustrates the use of functions with arguments and with return values.

```
int sum(int, int);
void main( )
{
        int a=5, b=6,c=0;
        c=sum(a, b);
        printf("%d",c);
}
int sum(int x, int y)
{
        int z=0;
        z=x+y;
        return z;
}
```

In the above program, the sum function is returning an integer value, that is, z.

That is why an integer data type has been specified with it.

11.7.4 No Arguments with Return Value :

This is rarely used category. Very few examples are there of this category. Consider an example, where main function wants to compute the area of circle. Main() function has taken the value of radius from the user. Now, to compute the area main() function is calling a function called pivalue() 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 pivalue(), but pivalue() function is returning 3.14 to the main() function. Therefore, this type of function is coming under the category that is No argument, with return value.

```
#include<stdio.h>
float pivalue();
void main()
{
        float r, area;
        printf("Enter Radius:");
        scanf("%f",&r);
        area=pivalue()*r*r;
        printf("Area is: %.2f", area);
}
float pivalue()
{
        return 3.14;
}
```

❑ **Check Your Progress – 5 :**

1. Identify the category for the function void sum (int, int);

   [A] No argument with Return [B] Argument with Return

   [C] No argument, No Return. [D] Argument with No Return

2. From the following function, which is example of No argument and No return ?

   [A] void printhello(); [B] void sum(int, int);

   [C] int sum (int, int); [D] float pivalue();

11.8 Let Us Sum Up :

In this unit, we :

- Have discussed about the concept of functions, which minimizes the no. of statements used in a program.
- Have discussed whenever the same set of instructions are used again and again in the program, then in spite of writing them several times, the function can be called.
- Have studied about the forms of C functions.
- Have studied about the different types of return values.
- Have studied about the various types of functions.

11.9 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 a one program more that one main() functions can exists

❑ **Check Your Progress 3 :**

1. [C] int 2. [A] void

3. [C] default return type of the function is void

❑ **Check Your Progress 4 :**

1. [B] declared after main() 2. [C] actual

3. [A] formal

❑ **Check Your Progress 5 :**

1. [D] Argument with No Return 2. [D] void printhello();

11.10 Glossary :

1. **Function :** It is a set (group) of executable statements.
2. **Return type :** It is a data type of the value return by a function.
3. **Formal arguments :** It is a list of parameters passed at the time of defining a function.
4. **UDF :** It is a User Defined Function.

11.11 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.

11.12 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.

11.13 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.

11.14 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
12

MORE ABOUT FUNCTIONS

UNIT STRUCTURE

12.0 Learning Objectives :

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

- Know about handling of non–integer functions.
- Know about the scope and lifetime of variables.
- Understand recursion and recursive programs.

12.1 Introduction :

In this unit, we will be discussing about the method of handling non–integer functions. The nesting of functions and the recursion process are also explained which avoid the usage of a loop.

Recursive function is a function that calls itself upon its execution. The functions of different ANSI C functions are also explained which will be helpful in developing programs.

12.2 Handling of Non–Integer Functions :

Sometimes, instead of returning integer value, some functions may return float, double or character type of values. Depending on the type of value returned by them, the specified data type is mentioned before the function name. For example, consider the given program :

```
float calculate(float, float);
double divide(double , double );
void main( )
{
        float x=0;
        double y=0;
        x=calculate(a, b);
        y=divide (m, n);
}
float calculate(float a, float b)
{
        //Body of calculate function
}
double divide (double a, double b)
{
        //Body of Divide function
}
```

If we have a mismatch between the type of data that the called function returns and the type of data that the calling function expects, unpredictable results can be observed.

12.3 Nesting of Functions :

C–Language allows nesting of two functions. There is no limitation in how deeply functions can be nested. Suppose a function x can call function y and function y can call function z and so on. The same can be illustrated more clearly with the help of an example:

```
#include<stdio.h>
int sum4(int, int, int, int);
int sum2 (int, int);
void main()
{
        int result;
        result =sum4(5,10,15,20);
        printf("Result is: %d", result);
}
int sum4(int a, int b, int c, int d)
{
        int x,y,z;
        x=sum2(a,b);
```

```
        y=sum2(c,d);
        z=x+y;
        return z;
}
int sum2(int p, int q)
{
        return (p+q);
}
```

In the above program main() function needs a sum of 4 numbers, those are 5, 10, 15 and 20. To do so, it is calling a function sum4 and passed all 4 numbers to it. Function sum4 will compute the sum of 4 numbers. This function takes 4 numbers passed by main() function into 4 variables, a, b, c, and d, respectively. Function sum4() in terms calling a function sum2, which can do the sum of only 2 numbers. sum4() function, is calling sum2() function and passed a and b and its sum is stored in variable x. Similarly, sum4 function again calling sum2 function and pass variable c and d to sum2 function. Sum returned by sum2 function is being stored in variable y. finally z= x + y will be return to the main function and main function will print the result that is 50 on the console screen. Here, main() function is calling to sum4() function and sum4() function is calling to sum2() function. This is called the nesting of the functions.

❑ **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

12.4 Recursion :

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. The process is used for repetitive computation in which each action is stated on forms of a previous result. Many iterative problems can be written in this form. Suppose we wish to calculate the factorial of a positive integer quantity. We would normally express this problem as n! = 1 x 2 x 3 x 4 … x n where n is the specified positive integer. However, we can also express it in another way, by writing n=1 = n*(n–1). This is a recursive statement of the problem, in which the last expression provides a condition to stop, for the recursion.

When a recursive program is executed the recursive function, calls are not executed immediately. Rather, they are placed on a stack until the condition that terminates the recursion, is encountered. 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 during each call. The name of the local 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.

For example, given below is a program of calculating factorial of a number using recursion :

```
#include<stdio.h>
int fact(int);
void main()
{
        int result,n=5;
        result =fact(n);
        printf("Factorial is: %d", result);
}
int fact(int tmp)
{
        if (tmp==1)
                return 1;
        else
                return (tmp * fact(tmp–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 tmp variable and checks it is 1. The value passed is 5, not 1. So it will execute the statement 'return (tmp * fact(tmp–1)'. Now, to execute this line system needs value of tmp and value of fact(tmp–1). System knows that the value of tmp is 5, but system do not know the value of fact(tmp–1). Here, system will create the another instance of function fact(4) and passed number 4 to it (as tmp–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 1 (as tmp=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(tmp–1) is 2, it will return tmp * fact (tmp–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 and hence, it takes more memory. The benefit if recursion is, it faster than iterative

process. Recursion is complex logic to understand, but can be used to solve complex logic in very a smaller number of lines of code.

❑ **Check Your Progress – 2 :**

1. Recursion process is suitable to _____ data structure.

   [A] queue [B] tree [C] array [D] stack

2. The process of function calling itself is called _____.

   [A] nesting of functions [B] recursion

   [C] multiple functions [D] All of the above

3. Which of the following statement is false ?

   [A] Recursion uses more memory than iterative process.

   [B] Recursion is slower process than iterative process.

   [C] Function calls itself is called recursion.

   [D] Complex logic can be implemented easily with less code using recursion.

12.5 Functions With Arrays :

Since the convention is that char arrays contain text which ends in '\0', it is not necessary to pass the length information to the function. Array declared inside a function is called locally declared array, which can be accessible to another function by pointers. We will do details discussion about pointer in Block–4. Here just will see, how can we access array declared in a one function to another function. Consider 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 name[10];
        printf("Enter your name in Lowercase letters:");
        scanf("%s",name);
        upper(name);
        printf("Your name is: %s");
}
void upper(char *name)
{
        int i=0;
        while(name[i]!='\0')
        {               name[i]=name[i] –32;
                i++;
        }
}
```

❑ **Check Your Progress – 3 :**

1. If we need to pass array declared locally in one function to another function, the formal argument of another function should be _____.

   [A] pointer [B] structure

   [C] normal variable [D] None of the above

2. To pass the character array, formal argument of the function should be _____.

   [A] function_name(char *t) [B] function_name(char t[])

   [C] Both A and B [D] None of the above

4.6 Scope and Lifetime of Variables in Functions :

A variable in C can have any one of the storage classes :

Automatic variables

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, hence the name automatic. Automatic variables are therefore private to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

A variable declared inside a function without storage class specifications, by default, is automatic.

One important feature of an automatic variable is that their value cannot be changed accidentally. This assures that we may declare and use the same variable name in a different function in the same program without causing any confusion to the compiler. 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;
        int y;
}
```

In the program given above, variables x, and y both are of type automatic variables .

❖ **External variables**

The external storage class describes that the variable has been defined at another place (not in the same program, may in another file). These variables are usually declared before defining main() function. Keyword 'extern' (optional) is used to specify external variable.

For example, create a file test.c and write following code :

```
#include<stdio.h>
int j=5;
```

Create another file, called test1.c in the same directory of test.c (so we don't have to specify the path) and write following code :

```
#include<stdio.h>
#include "test.c"
void main()
{
    extern int j;
    int i;
    for(i=1; i<=j;i++)
        printf("\nHello");
}
```

Compile both the files and run test1 program. You we get Hello is printed 5 times. The questions is: why 5 times ? In the test1.c we start running loop from i=1 to j. You can notice that in the entire program of test1.c we haven't initialized j=5. So how it takes value to 5 for variable j ? The answer is: the value of variable j is taken from the test.c file.

❖ **Static variables :**

The static variables may 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 local static variable. Consider the following example and see the output.

```
#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=5;i++)
        print();
}
void print()
{
    int static m=1;
    printf("\n%d",m);
    m++;
}
```

If you run the above program, you will get output 1, 2, 3, 4, 5 on the screen. Now remove the static word from the print function, for variable m, then compile and run the program again note down the output. You will get 1,1,1,1,1.

❖ **Register Variable :**

Usually when we declare a variable, it is created in the main memory (Random Access Memory). When we declare a variable of type register, then it will be created in the special memory called register. Register is a fastest memory, resided in the CPU itself, and hence it will speed up the process. Keyword 'register' is used to declare this type of variable. For Example,

register int x;

❑ **Check Your Progress – 4 :**

1. If we have declared a variable 'int x;', in the main() function, then x will be of type _____.

   [A] auto [B] register [C] extern [D] global

2. To access the variable declare and initialized in another file, we need _____ variable.

   [A] auto [B] register [C] extern [D] global

3. _____ variable is not declared int the primary memory, and they will reside in the special memory in the CPU.

   [A] auto [B] register [C] extern [D] global

12.7 Ansi C Functions :

Some of the commonly used ANSI C functions are :

1. isalnum()– Checks whether a character is alphanumeric or not.
2. isalpha()– Checks whether the given character is an alphabet or not.
3. isdigit()– Checks whether a character is a digit or not.
4. islower()– Checks whether a character is a lower case letter or not.
5. isupper()– Checks whether a character is an uppercase letter or not.
6. isspace()– Checks whether a character is a whitespace or not.
7. toupper()– Converts a lowercase character to an uppercase.
8. tolower()– Converts an uppercase character to a lowercase.

❑ **Check Your Progress – 5 :**

1. To check whether the value in a character variable, is from 0 to 9 _____ function is used.

   [A] isnumber() [B] isalnum() [C] isdigit() [D] isapha()

2. To check whether the value in a character variable, is alphabet or not _____ function is used.

   [A] isnumber() [B] isalnum() [C] isdigit() [D] isapha()

12.8 Let Us Sum Up :

In this unit, we :

- Have studied about making programs using non–integer functions.
- Have studied the method of nesting functions.
- Have seen the process of making recursive functions.
- Have studied about the different types of variables and their lifetime.
- Have studied about the functions of ANSI C Functions.

12.9 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

- **Check Your Progress 4 :**

  1. [A] auto
  2. [C] extern
  3. [A] register

- **Check Your Progress 5 :**

  1. [C] isdigit()
  2. [D] isalpha()

12.10 Glossary :

1. **Local variable :** 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 :** 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 :** It is a special type of variable which holds the address (reference) of some another variable.
4. **Register :** It is a small sized, fastest memory resided in the CPU itself.

12.11 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.

12.12 Activity :

Write program to demonstrate extern variable, register variable and static variable. Write your comments for each type of variable.

12.13 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;
```

12.14 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

- 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.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	9	10	11	12
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	_____
Language and Style	☐	☐	☐	☐	_____
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	_____
Conceptual Clarity	☐	☐	☐	☐	_____
Check your progress Quest	☐	☐	☐	☐	_____
Feed back to CYP Question	☐	☐	☐	☐	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....

Dr. Babasaheb Ambedkar
Open University Ahmedabad

BCAR-103/
DCAR-103

Fundamentals of Programming Using C Language

BLOCK 4 : STRUCTURES, POINTERS AND FILE HANDLING

STRUCTURES, POINTERS AND FILEHANDLING

Block Introduction :

After discussing about Arrays and Functions in previous block, 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 16th 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 16th unit contains some solved programs based on the statements/ concepts explained in the first 15 units. This will help the learners to understand those concepts in details as the problems are practically solved.

Block Objectives :

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-13 student will able to make their own user defined data type using structure and union.

Unit-14 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-16. Behind this unit our objective is to aware student about handling of text and binary files.

Our final unit, that is Unit-16 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 :

Unit 13 : Structures & Unions

Unit 14 : Pointers

Unit 15 : File Handling

Unit 16 : Solved Programs–III

Unit
13

STRUCTURES AND UNIONS

UNIT STRUCTURE

13.0 Learning Objectives :

In this unit, we will learn about user defined datatypes such as structure and union and the various operations to be performed on them.

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

- Know about defining user defined data type
- Know about Structure Initialisation
- Know about passing Structure variables in functions
- Know about Union declaration and its concept

13.1 Introduction :

In this unit, we will discuss about Structure and Union, creating structure variables, assigning values to data members, functions and structures, passing entire structures to functions, use of arrays of structures, structure within a structure (i.e. nested structure and union).

Structures are slightly different from the variable types you have been using till now. Structures are data types by themselves. When you define a structure or union, you are creating a custom data type.

13.2 Structures :

Structures in C are defined as collection of a sequence of named members of different types. They are similar to records in other programming languages like Pascal. The data members of a structure are stored in consecutive locations in memory, but for space efficiency, the compiler can insert padding byte between or after members. Note that compiler never pad byte before the first member. The size of a structure is equal to the sum of the sizes of its data members and the size of the padding bytes.

13.2.1 Structure Initialization :

In simple way, you can define a structure as a collection of one or more variables types grouped under a single name. The difference between array and structure is that array has same data type members, while structure has different types of data members. 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 of the structure.

```
struct <tagname>
{
datatype member1;
datatype member2;
_
_
/* additional statements may go here */
} instance;
For Example,
struct student
{
        int rollno;
        char name[5];
        float percent;
};
struct student S1;
```

In above syntax, the struct keyword is used to declare structures. The keyword struct identifies the beginning of a structure definition. It is followed by a tag name, which is the name of structure. Following the tag are the structure data members of various types, enclosed in braces. If you define the structure without the instance, it's just a template that can be used later in a program to declare structures. Here S1 is student kind of variable which has three member's rollno, name and percentage.

```
struct student
{
        int rollno;
        char 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 instances of type student. Each structure variable contains 3 members no, name and per.

❑ **Check Your Progress – 1 :**

1. Structure is _____.

[A] heterogeneous collection of data

[B] user defined data type

[C] Both A and B

[D] homogeneous collection of data

2. From the following which statement is false :

[A] Structure allows us to create our own custom data types.

[B] Structure can store different types of data.

[C] Structure is used to store same type of data.

[D] Structure should have its instances (variable).

A structure variable can be assigned values during declaration. The initialization values must be given in { }. The values must match type of the structure members.

```
struct student
{
        int no;
        char nm[10];
        float per;
} S1= {101, "Veena", 80.5};
```

OR

```
struct student S1={101,"Veena",80.5 } ;
```

structure members can be accessed using dot operator (.) also called structure member operator.

Syntax :

Structure_variable .member

For example

```
S1.no=201;
printf("%d %s %f" , S1.no,S1.name,S1.per);
```

13.2.2 Size of Structure :

The total size of structure can be calculated by adding the size of all the data types used to form the structure. A structure declaration does not reserve any member space. It merely describes template. Memory is allocated only when instances or variables of structure are 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 1 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 = _____.

[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.

13.2.3 Comparison of Structure Variables :

ANSI C standard supports small operations on structure like copying data members, assignment and passing reference to function etc. One of the operations which are not allowed is comparison of structures. Note that, you cannot compare the structures using the standard comparison facilities (= =,>,<etc.);

13.2.4 Arrays within Structures :

An array can be a member of structure. For example, consider the given structure:

```
struct emp
{
        int a[5][5]; // a 5 x 5 array of integers
        float x;
} y;
```

Now, in order to refer element of 2nd row and 4th column you should write the given statement :

y. a[1][3]

As the numbering starts from 0, so 2nd element can be referred by using index number 1 and 4th element can be referred by using index number 3.

13.2.5 Arrays of Structures :

As you aware that structures that contain arrays, so you also have arrays of structures. In fact, arrays of structures are very powerful programming technique to have more than one instance of data. For example, in a program to maintain a list of phone numbers, you can define a structure to hold each person's name and number :

```
struct phone_entry
{
        char first_name[10];
        char last_name[12];
        char phone_no[8];
};
```

A phone list must hold many entries, however, so a single instance of the entry structure isn't of much use. What you need is an array of structures of type entry. After the structure has been defined, you can declare an array as follows:

struct phone_entry list[100];

This statement declares an array named list that contains 1,00 elements. Each element is a structure of type phone_entry and is identified by subscript like other array element types. Each of these structures has three elements, each of which is an array of type char. When you have declared the array of structures, you can manipulate the data in many ways like copying data. For example, to assign the data in one array element to another array element, you would write

list[1] = list[5];

This statement assigns to each member of the structure list[1] the values contained in the corresponding members of list[5]. You can also move data between individual structure members. The statement,

strcpy(list[1].phone_no, list[5].phone_no);

copies the string in list[5].phone_no to list[1].phone_no. If you want to, you can also move data between individual elements of the structure member arrays :

list[2].phone_no[3]= list[5].phone_no[1];

This statement moves the second character of list[5]'s phone number to the fourth position in list[2]'s phone number. (Do not forget that subscripts start at offset 0.)

Example :

```
#include <stdio.h>
/* Define a structure to hold entries. */
struct phone_entry
{
        char first_name[20];
        char last_name[20];
        char phone_no[10];
};
/* Declare an array of structures. */
struct phone_entrylist[4];
int i;
void main( )
{
        /* Loop to input data for four people. */
        for (i = 0; i < 4; i++)
        {
                printf("\nEnter first name: ");
                scanf("%s", list[i].first_name);
                printf("Enter last name: ");
                scanf("%s", list[i].last_name);
                printf("Enter phone in 123–4567 format: ");
                scanf("%s", list[i].phone_no);
        }
        /* Print two blank lines. */
        printf("\n\n");
        /* Loop to display data. */
        for (i = 0; i < 4; i++)
        {
                printf("Name: %s %s", list[i].first_name, list[i].last_name);
                printf("\t\tPhone: %s\n", list[i].phone_no);
        }
        return 0;
}
```

❑ **Check Your Progress – 3 :**

1. Compute memory space needed to declare array x of following structure.

```
struct student
{       int roll_no;
        char name[10];
        float percentage;
} x[20];
```

[A] 16 Bytes [B] 160 Bytes [C] 320 Bytes [D] 340 Bytes

2. Compute memory space needed to declare array x of following structure.

```
struct student
{       int roll_no;
        char name[10];
        int marks[3];
        float percentage;
} x[10];
```

[A] 220 Bytes [B] 22 Bytes [C] 90 Bytes [D] 180 Bytes

13.2.6 Structures within Structures :

Structures can be nested, that is, structure templates can contain structures as members. For example, consider two structure types:

```
struct employee_one
{
        int code;
        float sal;
};
struct employee_two
{
        int offce_code;
        struct employee_one y;
}x;
```

The above two structures are of different types, but the data members of the first structure are included in the second. An instance of the second structure can be initialized by the following initialized assignments as given below:

```
x.offce_code=25;
x.y.code=10;
x.y.salary=2000.0;
```

Notice the way in which the member operator. can be used again and again. Also note that no parenthesis is necessary, because the reference which is calculated by the dot (.) operator is worked from left to right.

13.2.7 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. From the given below select 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 needed to declare variable x of following structure.

```
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

13.3 Unions :

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, whereas each member within a structure is assigned its own unique storage area. Thus, Unions provide an efficient 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 tag
{
      data type member 1;
      data type member 2;
      ……….;
      data type member m;
};
```

Where union is a required keyword and the other terms have the same meaning as in a structure definition. Individual union variables or instances can then be declared as

[storage–class] union tag variable1, variable 2, ……, variable n;

where storage class is an optional storage class specifier, union is a required keyword, tag is the name that appeared in the union definition and variable 1, variable 2, variable n are union instances of type tag.

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 struct 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]

13.4 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.

13.5 Suggested Answers For Check Your Progress :

❑ **Check Your Progress 1 :**

1. [C] Both A and B
2. [C] structure is used to store same type of data.

❑ **Check Your Progress 2 :**

1. [B] 16 Bytes
2. [A] Sum of memory space required for each data member

❑ **Check Your Progress 3 :**

1. [C] 320 Bytes
2. [A] 220 Bytes

❑ **Check Your Progress 4 :**

1. [B] structure_varaible.data_member_name=value
2. [D] 34 Bytes

❑ **Check Your Progress 5 :**

1. Structure
2. Union
3. Structure
4. Union
5. Structure

13.6 Glossary :

1. Structure : It is a user defined data type, which allows us to encapsulate different types of data into a single unit.

13.7 Assignment :

1. What is Structure ? How can we declare and initialized it ?
2. What is Union ? How it differs from the Structure ?
3. Discuss array of Structures with an example.

13.10 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.

13.11 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,

13.12 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
14

POINTERS

UNIT STRUCTURE

14.0 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

14.1 Introduction :

A **pointer** is feature of C programming language who refers directly to (or **"points** to") another value stored elsewhere in the computer memory using its address. Or in simple words, pointer is variable which stores address of another variable. For high–level programming languages, pointers can be used as effectively as general–purpose registers in low–level languages such as assembly language or machine code. It is more useful for machine address arithmetic.

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. 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.

14.2 Understanding Pointers :

A pointer is a variable that represents the location of a data item, such as a variable or an array element. Pointers have a number of useful applications. For example, when we want to return multiple data items from a function, then pointer provides a way to return via arguments. Also, pointer can be used to pass information back and forth between functions. In particular, Pointer also permits references to other functions to be specified as arguments. This has the effect of passing function as arguments to the given functions. Pointers are also closely related with arrays and therefore provide an alternative way to access individual array elements.

14.2.1 Accessing the Address of a Variable :

***And & Operators**

Suppose v is a variable that represents some particular data item. The address location can be determined by the expression &v. where & is a unary operator called the address operator, which evaluates the address of its operand.

Now let us assign the address of v to another variable pv. Thus pv = &v. This new variable is called pointer to v, since it "points to" the location where v is stored in address, not its value. Thus, pv is referred to as a pointer variable.

The data item represented by v (i.e. the data item stored in v's memory cells) can be accessed by the expression *pv where * is a unary operator called the indication operator, that operates only on a pointer variable. Therefore, *pv and v both represent the same data item (i.e. contents of same memory cell). Furthermore, if we write pv = &v and u=*pv then u and v will both represent the same value, i.e. the value of v will indirectly be assigned to u. (it is assumed that u and v are declared to have the same data type).

The unary operators & and * are members of the same precedence group as the other unary operators. This group of operators has a higher precedence than the groups containing the arithmetic operators. The associativity of the unary operators is right to left.

The address operator (&) must act upon operand associated with unique addresses, such as ordinary variables or single array elements. Thus, the address operator cannot act upon arithmetic expressions such as 2*(u+v).

The indirection operator (*) can only act upon operands that are pointers. However, if pv points to v (i.e. pv=&v) then an expression such as *pv can be used interchangeably with its corresponding variable v, thus an indirect reference can also appear on the left side and assignment statement. This provides another method for assigning a value to an array element of an ordinary variable (e.g. v) within a more complicated expression.

Pointer variables can point to numeric or character variables, arrays, functions or other pointer variables. A pointer can also be assigned the value of another pointer variable (i.e. pv=px) however, both pointer variables point to data items of the same type. Moreover, a pointer variable can be assigned a null (zero) value.

❑ **Check Your Progress – 1 :**

1. _____ is used to store address or reference of another variable.

   [A] Structure [B] Union

   [C] Pointer [D] None of the above

2. To fetch the address of any variable, _____ operator is used.

   [A] * [B] & [C] –> [D] referenceof

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;

Where ptr is the name of the pointer variable and data type refers to the data type of the pointer and note that an asterisk must precede ptr.

For example,

```
float u,v;
float *pv;
– – –
pv = &v;
```

The first line declares u and v to be floating point variables. The second line declares pv to be a pointer whose object is a floating–point quantity. i.e. pv points to a floating–point quantity. Note that pv 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 *pv=&v;
```

v should be declared before pointer pv

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 *y;
    y=&x;
    printf("Value of x= %d",x);
```

```
        printf("Address of x= %d",y);
        printf("Value of y= %d",y);
        printf("Value of x= %d",*y);
    }
```

❖ **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 y.

❑ **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] ^

14.3 Pointer Operations :

Expressions involving expressions conform to the same rules as other expressions do. Few special aspects of pointer expressions are discussed below.

14.3.1 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 will now point to 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 5000. Now assuming that integers are 2 bytes long, the given statement:

p1++;

p1 will contain 5002 instead of 5001, as each time when p1 is incremented, it will point to the next integer.

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 behave in normal manner as characters are always 1 byte long.

14.4 Pointers and Arrays :

If x is a one–dimensional array, then the address of the first array element 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+1th 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 Pointers

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 are 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 awkward 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 2, therefore, the object of this pointer *(x+2), refers to the entire row, since row2 is a one–dimensional array. *(x+2) is actually a pointer to the first element in row2.(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; statement is given, choose the correct option, to store the address of an array x to pointer variable.

   [A] p=&x; [B] p=x; [C] *p=x; [D] p=&x;

2. int x[10], *p; statement is given, choose the correct option, to store the address of third element of an array x to pointer variable 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

14.5 Pointers and Character Strings :

Now let us observe the way pointers are used with strings. Consider the given statement:

char *a = "I like Computers";

It can be noted that a is not an array. The reason for the same is the way the compiler operates, as all C/C++ compilers create a table called string table which is used to store the string constants used by the program. Therefore, the above declaration places the address of "I like Computers" as stored in the string table, into the pointer a. throughout the whole program, a can be used like any other string. For example,

```
void main( )
{
        char *a = " I like Computers";
        int l=0, i=0;
        l=strlen(a);
        for(i=0;i<l;i++)
        {
                printf("%c", a[i]);
        }
}
```

14.6 Pointers and Functions :

Pointers are often passed to a function as arguments. This allows actual parameters within the calling portion of the program to be accessed by the called function. Called function modifies these parameters and returns them to the calling portion of the program in modified form. We refer to this use of pointer as passing of arguments by reference (or by address or by location).

When an argument is passed by value, the data item is copied to the function. Thus, any alteration made to the data item within the function is not carried over into the calling routine. When an argument is passed by reference, however (i.e. when a pointer is passed to a function) the address of 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. Moreover, change that is made to the data item (i.e. to the contents of the address) will be accepted in both the called function and the calling routine, thus the use of pointer as a function parameter allows the corresponding data item to be changed globally from within the function. When pointers are used as parameters to a function, the formal arguments that are pointers must be preceded by an asterisk.

Let us see the example given below

Within fun2, the contents of the pointer address are reassigned the values 0.0; since, the addresses are recognized in both fun2 and main, the reassigned values will be recognized within main after the call to fun2, therefore, the integer variable u will be accessed indirectly, by referencing the pu.

```
void main()
{
        int u=5,v=10;
        void fun2(int *pu, int pv);
        printf(" \n before func2: u=%d v=%d " , u, v) ;
        fun2(&u,v);
        printf(" \n after calling func2: u=%d v=%d " , u, v) ;
}
void fun2(int *pu, int pv)
{
        *pu=0;
        pv=0 ;
        printf(\n within fun2 pu=%d        pv=%d", *pu,pv);
        return;
}
```

❖ **Output :**

```
before func2: u=5 v=10
within fun2 pu=0 pv=0
after calling func2: u=0 v=10
```

thus, v is not changed because we pass the value to pv, not address of v.

In the declaration of the formal argument within the first line of fun2, that is void func2 (int *pv, int pv), the formal arguments pu and pv are consistent with the arguments in the function prototype in main function.

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. However, it is not necessary to precede the array name with an &(ampersand) within the function call. An array name that appears as a formal argument within a function definition can be declared either as a pointer or as an array of unspecified size.

It is possible to pass elements of an array, rather than an entire array, to a Pointers function. To do so, the address of the first array element must be passed as an argument. The remainder of the array, starting with specified array element, will then be passed to the function. A function can also return a pointer to the

calling portion of the program. The function definition and any corresponding function declaration must indicate that the function will return a pointer; this is accomplished by preceding the function name with an asterisk. The asterisk must appear in both the function definition and the function declaration.

For example :

*datatype *function_name(argument list);*

*int *fun3(int a)*

fun3 is function accepting integer a and returning a pointer to an integer.

❑ **Check Your Progress – 4 :**

1. When the reference of the local variable is to passed to the function, instead of value ?

   [A] If function needs to read the value of the variable.

   [B] If function needs to write the value of the variable.

   [C] To make that variable global.

   [D] None of the above.

2. For function int my_function(int *p), which is a correct way to call that function ? Assume x and y are integer variables and we need to pass variable x.

   [A] y=myfunction(&x); [B] y=myfunction(*x);

   [C] y=myfunction(x); [D] All are valid options

14.7 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.

14.8 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.

14.9 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

14.10 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.

14.11 Glossary :

1. **Pointer :** Pointer is a special variable used to store reference of another variable.
2. **String :** String is a group of characters. In C–Language it is represented by character type array.
3. **Function :** 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.

14.12 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.

14.13 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).

14.14 Case Study :

Write a program to implement singly link list.

14.15 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
15

FILE HANDLING

UNIT STRUCTURE

15.0 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.

15.1 Introduction :

A file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image. The meaning attached to a particular file is determined entirely by the data structures and operations used by a program to process the file.

In this unit, we will be discussing about the various operations to be performed on files, performing input/ output operations on them and error handling operations, which will help the learners to make programs on files.

15.2 Management of Files :

You can perform variety of operations on a file once it is created. C offers facility of creating and working with files for large amount of data or when the

data has to be stored in the form of files. Managing of files and working with them will come under the category of file management or file handling.

15.2.1 Files :

A file is a place on the disk where a group of related data is stored. Some basic file operations can be given as :

- Naming a File
- Opening a File
- Reading Data from a File
- Writing Data to a File
- Closing a File

15.2.2 Defining and Opening a File :

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. This buffer area permits information to be read from or written to the data file more quickly. The buffer area is created by writing

*FILE *fp;*

Where FILE (uppercase required) is a special structure type that establishes the buffer area and fp is a pointer variable that indicates the beginning of the buffer area. The structure type file is defined with a system include file: typically, <stdio.h>

A data file must be opened or created before it can be created or processed. Thus, associates the file name with the buffer area. It also specifies how the data file will be utilized, that 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 used to open a file. This function is typically written as:

fp = fopen("fname", "file_type")

Where fname and file_type are strings that represents the name of the data file and the mode in which the data file will be utilized, respectively. The name chosen for the file name must be according the rules for naming files. The file_type must be one of the strings shown in the following table:

File_type	Meaning
"r"	open an existing file for reading only.
"w"	open a new file for writing only. If the file with specified File name currently exists, it will be destroyed and new file is created in its place.
"a"	open an existing file for appending. A new file will be created if the file with the specific file_name does not exist.
"r+"	opens an existing file for both reading and writing.
"w+"	opens a new file for both reading and writing. If a file with the specified file_name currently exists, it will be destroyed and a new file is created in its place.
"a+"	opens an existing file for reading and writing. A new file will be created if the file with the specified file_name does not exist.

The fopen() function returns a NULL value if the file cannot be opened or when an existing file cannot be found, otherwise it returns a pointer to the beginning of the file. At the end of program, data file must be closed so that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. It also prevents any unintentional mishandling of the file.

15.2.3 Closing a File :

Operating system put limits on number of files to be opened. So, in that case closing of unwanted file might help to open the required files. Sometimes, when we want to reopen the same file in a different mode, then first we have to close a file. This can be able with the library function fclose(). The syntax is simply:

fclose(fp);

Example :

```
#include <stdio.h>
void main()
{
        FILE *fpt;
        fpt=fopen("s.dat","w");
        if(fpt= =NULL)
        printf("\n File Can ?t Open);
        fclose(fpt);
}
```

❑ **Check Your Progress – 1 :**

1. To create the file pointer _____ keyword is used.
   [A] file [B] FILE [C] fopen [D] fclose
2. Syntax to open 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, _____ is a correct syntax.
   [A] exit(fpt); [B] quit(fpt); [C] fpt=flose(); [D] flcose(fpt)

15.3 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 :

These functions are used to handle a string at a time. The function fgets() is used to read a string of specified length from a file opened in read mode. For example,

fgets(str,79,fpt);

Would read string of 79 characters from the file whose file pointer is fpstore it in array str. The function will return (\0) NULL character when reached at the end of file.

Assume that a file is opened in the write mode and file pointer is fpt, then the statement

fputs(str,fpt);

Writes the string contained in the character array str to the file associated with FILE pointer fpt.

For example :

```
#include<stdio.h>
void main()
{
    FILE *fp1;
    char ch[80];
    fp1 = fopen("INPUT.txt", "w");
    while(strlen(getc(ch)>0)
    {
        fputs(ch,fp1);
        fputc("\n", ch);
    }
    fclose(fp1) ;
    fp1 = fopen("INPUT,txt", " r") ;
    while(fgets(ch,79,fp1) = NULL)
        printf("%s \n",ch);
    fclose(fp1);
}
```

❑ **Check Your Progress – 1 :**

1. To read a character from a file _____ function is used.
   [A] putc() [B] putchar() [C] getc() [D] getchar()
2. To write a string into the file, _____ function is used.
   [A] fgetstr() [B] fgets() [C] fputstr() [D] fputs()
3. Identify the correct syntax:
   [A] fputs(char, file_ptr); [B] fputc(char, file_ptr);
   [C] fgets(char,file_ptr); [D] Both A and B

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.

❑ **Check Your Progress – 3 :**

1. To write the different types of data to file, _____ function is used.
   [A] fprintf() [B] fscanf() [C] fputs() [D] fgets()
2. To write the different types of data to file, _____ function is used.
   [A] fprintf() [B] fscanf() [C] fputs() [D] fgets()

THE fread() AND fwrite() FUNCTIONS :

Some commercial applications involve the use of data files to store blocks of data instead of character by character, where each block consists of a fixed number of contiguous bytes. Each block will generally represent a complete data structure, such as a structure or an array. For example, a data file may consist of multiple structures 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 from the data file or write the entire block to the file. This is used only with binary – mode files.

Each of these functions requires four arguments: a pointer to the data block, size of data block, number of data blocks being transferred and the stream pointer. Thus, the fwrite() function can be written as:

fwrite(&inventory, sizeof(inventory),1,fptr);

fread(&inventory, sizeof(inventory),1,fptr)

Where inventory is a structure variable and fptr is the file pointer associated with the data file that has been opened for input/output. Once an unformatted data file has been created with fwrite(), it can be read with fread() function. The function returns a zero value if an end–of–file condition has been detected and non–zero value if an 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>
void main()
{
    struct book
    {
        int bookno;
        char title[30];
        float price;
    } bk;
    int ch,i,n;
    float p;
    FILE *fp;
```

```
        printf("\n how many 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 no=");
                scanf("%d",&bk.bookno);
                printf("\n Enter book Title=");
                scanf("%s",&bk.title);
                printf("\n Enter Book Price=");
                scanf("%f",&p);
                bk.price=p;
                fwrite(&bk,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(&bk,sizeof(bk),1,fp);
                printf("\n %d\t\t %s\t\t %.2f",bk.bookno,bk.title,bk.price);
        }
        fcloseall();
}
```

❖ **Output :**

```
how many records:2
Enter book no=1
Enter book Title=cprogramming
```

Enter Book Price=200

Enter book no=2

Enter book Title=c++

Enter Book Price=350

Book no	Title	Price
1	cprogramming	200.00
2	c++	350.00

❑ **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()

15.4 Error Handling During I/O Operations :

The standard library function ferror( ) returns any error that might have occurred during a read/write operation on a file. For successful the 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. _____ standard library function is used to detect any error, during 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

15.5 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

15.6 Suggested Answers For Check Your Progress :

❑ **Check Your Progress 1 :**

1. [B] File 2. [A] fpt=fopen("path","mode");

3. [D] fclose(fpt);

❑ **Check Your Progress 2 :**

1. [C] getc(); 2. [D] fputs();

3. [D] Both A and B

❑ **Check Your Progress 3 :**

1. [A] fprintf() 2. [B] fscanf()

❑ **Check Your Progress 4 :**

1. [C] Binary 2. [B] fread()

❑ **Check Your Progress 5 :**

1. [C] ferror() 2. [C] a

15.7 Glossary :

1. File : File is a representation of data, stored in permeant secondary memory.

15.8 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.

15.9 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.

15.10 Case Study :

Write a program to copy a text file.

15.11 Further Reading :

- "Programming in C" by Ashok N. Kamthane, PEARSON Publications.

Unit
16

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

/* Program:1 WRITE APROGRAM TO FIND THE MINIMUM AND MAXIMUM VALUE */

```
#include<stdio.h>
void main()
{
   int arr[10],i,maximum,minimum,num;
   printf("Enter Number of Elements for array:–");
   scanf("%d",&num);
   for(i=1;i<=num;i++)
   {
      printf("Enter The value:–>");
      scanf("%d",&arr[i]);
   }
   for(i=1;i<=num;i++)
   {
      if(arr[i]>= maximum)
      {
         maximum =arr[i];
      }
   }
    for(i=1;i<=num;i++)
   {
      if(arr[i]<=minimum)
      {
         minimum=arr[i];
      }
   }
   printf("\nMaximum Value is:–>%d\n",maximum);
   printf("Minimum Value is:–>%d\n",minimum);
}
```

❖ **Output :**

Enter Number of Elements for array:–5
Enter The value:–>24
Enter The value:–>76
Enter The value:–>28
Enter The value:–>2
Enter The value:–>45
Maximum Value is:–>76
Minimum Value is:–>2

```
/*Program:2 PROGRAM TO CONVERT THE DECIMAL NUMBER TO
• BINARY NUMBER
• OCTAL NUMBER
• HEXA–DECIMAL NUMBER */
#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

```
/*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: CProgrammingLanguage

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;
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

```
/* 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: BAOU

Reverse String is: UOAB

```
/* 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);
    if(i==0)
        printf("Given String is Palindrome:");
    else
        printf("Given string is Not Palindrome");
}
```

❖ **Output :**

Enter Any String: madam

Given String is Palindrome:

```
/* 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*/
#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 */
#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 */

```
#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);
    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 */
#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:ABC
Enter Age:18
Enter Roll No:2
Enter Name:XYZ
Enter Age:20
Enter Roll No:3
Enter Name:PQR
Enter Age:19

– – – – – – – – – – – – –
Writing Data to File:
Reading Data from a File:
RollNo      Name      Age
– – – – – – – – – – – – –
1           ABC       18.00
2           XYZ       20.00
3           PQR       19.00
```

```
/*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;
      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 */
#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_Pointer Reference.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_Pointer Reference.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

- 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 :**

1. What is difference between structure and union ?
2. Explain how pointer can be passed as argument to function with example ?
3. Explain different modes in which file can be opened ?

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

Unit No.	13	14	15	16
No. of Hrs.				

2. Please give your reactions to the following items based on your reading of the block :

Items	Excellent	Very Good	Good	Poor	Give specific example if any
Presentation Quality	☐	☐	☐	☐	_____
Language and Style	☐	☐	☐	☐	_____
Illustration used (Diagram, tables etc)	☐	☐	☐	☐	_____
Conceptual Clarity	☐	☐	☐	☐	_____
Check your progress Quest	☐	☐	☐	☐	_____
Feed back to CYP Question	☐	☐	☐	☐	_____

3. Any other Comments

.....

.....

.....

.....

.....

.....

.....

.....

DR.BABASAHEB AMBEDKAR
OPEN UNIVERSITY

'Jyotirmay' Parisar,
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481.
Website : www.baou.edu.in