

PROGRAMMING C

BCA

CODE: BCA -04

I SEMESTER

Course writer:

**C. SUBA
VELS UNIVERSITY, ASST. PROFESSOR,
DEP. OF COMPUTER SCIENCE,
GAYATHRI HOMOEOPATHIC CLINIC, AYYAPATHAGAL,
CHENNAI-600056**

CONTENTS

Page No.

MODULE 1

Unit 1	Algorithms and Flowchart	6
Unit 2	Constants Variables and Data Types	29
Unit 3	Operators and Expressions	47
Unit 4	Managing Input and Output Operations	71

MODULE 2

Unit 1	Decision Making and Branching	85
Unit 2	Decision Making and Looping	105
Unit 3	Arrays	122

MODULE 3

Unit 1	User Defined Functions	138
Unit 2	Functions	159
Unit 3	Pointers	171
Unit 4	Structures and Unions	187
Unit 5	File Management in C	205

PROGRAMMING IN C

Subject Code: BCA13

Number of Credit Hours:

MODULE 1

UNIT-1

ALGORITHMS AND FLOWCHART

Algorithms, flowcharts, Divide and conquer strategy, Writing algorithms and flow charts for simple exercises.

UNIT-2

CONSTANTS VARIABLES AND DATA TYPES

Character Set, C tokens, Keywords and identifiers, Constants, Variables, Data types, Declaration of variables.

UNIT-3

OPERATORS AND EXPRESSIONS

Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Increment and Decrement Operators, Conditional Operators, Bitwise Operators, Special Operatrs, Arithmetic Expression, Evaluation of Expression, Precedence of Arithmetic Operators, Type Conversion in Expressions, Operators Precedence and Associativity.

MODULE 2

UNIT-1

DECISION MAKING AND BRANCHING

Decision Making with If Statements, Simple If Statements, The If Else Statements, Nesting of Else Statements, The Else If Ladder, The Switch Statement, The ?: Operator, The Goto Statement.

UNIT-2

DECISION MAKING AND LOOPING

The While Statement, The Do Statement, The For Statement, Jumps in Loops.

UNIT-3

ARRAYS

One Dimensional Arrays, Declaration of One Dimensional Array, Initialization of One Dimensional Array, Two-Dimensional Arrays, Declaration of Two Dimensional Arrays, Initialization of Two Dimensional Arrays, Example Programs.

MODULE 3

UNIT-1

USER DEFINED FUNCTIONS

Need for User-Defined Function, A Multi Function Program, Elements of User Defined Functions, Definition of Functions, Return Values and Their Types, Function Calls, Function Declaration.

UNIT-2

FUNCTIONS

Category of Functions, No Arguments and No Return Values, Arguments but No Return Values, Argument with Return Values, No Argument but Returns a Value, Function that Return Multiple Values.

UNIT-3

POINTERS

Understanding Pointers, Accessing the Address Space of a Variable, Declaring and Initialization Pointer Variables, Accessing a Variable through its Pointers, Pointer Example Programs.

UNIT-4

STRUCTURES AND UNIONS

Definition, Creation, Manipulation.

UNIT-5

FILE MANAGEMENT IN C

Different File Management Operations in C, Programming Examples.

MODULE-1

UNIT-1

UNIT

1

ALGORITHMS AND FLOWCHART

CONTENTS

- 1.1 Aims and Objectives
- 1.2 Introduction
- 1.3 What is Algorithm with Example
- 1.4 Study of Algorithm
- 1.5 Pseudocode for Algorithm
- 1.6 Flowchart
 - 1.6.1 Definition of flowchart
 - 1.6.2 Benefits of using flowchart
 - 1.6.3 Symbols used in flowchart
 - 1.6.4 Rules for creating flowchart
 - 1.6.5 Advantages of flowchart
 - 1.6.6 Limitations of flowchart
- 1.7 Pseudocode Example
- 1.8 Divide and Conquer Strategy
 - 1.8.1 Merge Sort
- 1.9 Writing algorithms and flow charts for simple exercises
- 1.10 Let us Sum up
- 1.11 Lesson and Activity
- 1.12 Keywords
- 1.13 Questions for Discussion
- 1.14 Suggested Readings

1.1 AIMS AND OBJECTIVES

At the end of this chapter you will learn to,

- Explain the concept of Algorithms and Pseudocode
- Define Flowchart, Benefits, and Symbols, Rules for Creating Flowchart, Advantages and Disadvantages of Flowchart
- A Study on Divide and Conquer Strategy with Example
- Writing algorithms and Flowcharts for Exercises

1.2 INTRODUCTION

Algorithm “why do we need to study algorithm”? If you want to be a computer professional, there are both practical and theoretical reasons to study algorithm. From a practical point of view, you should know the standard set of important algorithms from different areas of computing; In addition, you should be able to design new algorithms and analyze their efficiency. From the theoretical standpoint the study of algorithms, sometimes called **algorithms**.

Another reason to study algorithms is the usefulness in developing analytical skills. After that, algorithms can be seen as special kinds of solutions for the problems, but precisely defined procedures for getting answers. Consequently, specific algorithm design techniques can be interpreted and involved course the precision inherently imposed by algorithmic thinking limits the kinds of problems than can be solved with an algorithm.

In the 1930s, before the advent of computers, mathematicians worked very actively to formalize and study the notation of simple instructions were given for solving a problem or computer as a solution. Various formal modes of computation were desired and investigated. Much of the emphasis in the early work in this field computability theory was on describing and characterizing those problems that could be solved algorithmically and on exhibiting some problems that could not be solved. One of the important negative results was insolvability of the “halting problem”. The halting problem is to determine whether an arbitrary given algorithm (or computer program) will eventually halt (rather than, say get into an infinite loop) while working on a given input.

1.3 WHAT IS ALGORITHM WITH EXAMPLE

An algorithm is a sequence of unambiguous instructions for solving a problem that is a sequence of computational steps that transform them input into the output.

An algorithm has the following properties:

1. **Input:** The algorithms get input.
2. **Output:** The algorithms produce output.
3. **Definiteness:** Each instruction to represent with clear & unambiguous.
4. **Finiteness:** The algorithm terminates; that is it terminates after finite number of steps.
5. **Correctness:** The produced output by the algorithm is correct.

NOTATION OF ALGORITHM

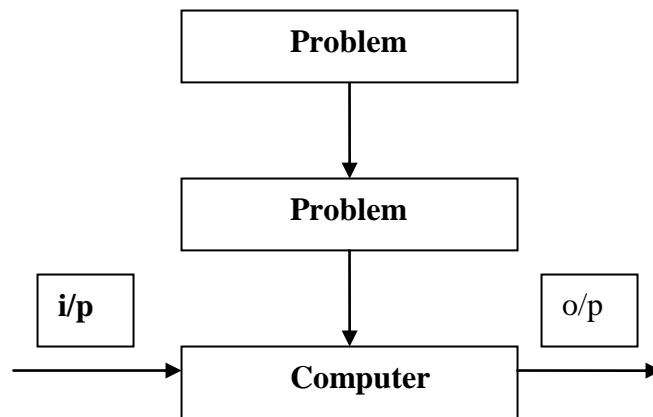


FIGURE 1.1

EXAMPLE

Consider the given algorithm to find the largest of three numbers x , y and z .

- Step 1:** Start the program
Step 2: Read the value x , y and z
Step 3: To compare $((x > y) \text{ and } (x > z))$ then
 print x
 else if $(y > z)$
 print y
 else
 print z
Step 4: Stop the program

Thus in the given above algorithm, consider the first step is to start the program, the second step to read the values of x , y and z and after that compare the first number with second number and also compare the first number with third number, if the condition is true then print the value of first number else compare the second number with third number if the condition is satisfied then display the second number otherwise display the third number.

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING

The given figure shows the sequence of steps in arithmetic problem solving

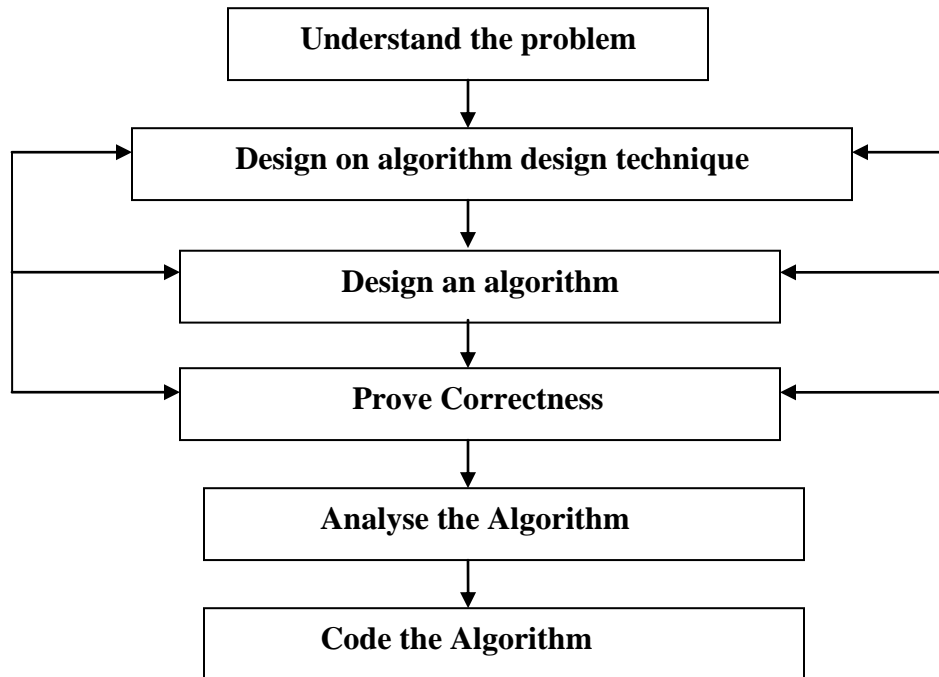


FIGURE 1.2

The first step in algorithm problem solving is to understand the complexity of the problem given. The fundamental importance of both algorithms and data structures for computer programming is very useful. An algorithm design techniques (or “strategy” or “paradigm”) is a general approach for solving problems algorithmically that is applicable to a variety of problems from different areas of computing. There are two methods used for designing an algorithm.

1. Pseudocode
2. Flowchart

A pseudocode is a mixture of a natural language is usually more precise than a natural language and its usage often yields more succinct algorithm description. Pseudocode for the statement used such as for, if and while and also used for \leftarrow assignment operation two slashes for // comments.

In the second approach for specifying algorithms was a flowchart, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithms steps. Once an algorithm has been specified, you have to prove its correctness. That is you have to prove that the algorithm yields a required result for every legitimate input in finite amount of time.

After correctness, by far the most important is efficiency. In fact, there are two kinds of algorithmic efficiency: time efficiency and space efficiency. Time efficiency indicates how fast the algorithm runs; space efficiency indicates how much extra memory the algorithm needs. Another desirable characteristic of an algorithm is simplicity and generality. Two issues have; in generality of the problem the algorithm solves and the range of inputs it accepts. On the first issue, note that it is sometimes easier to design an algorithm for a problem posed in more general terms. Consider for example, the problem for determining whether two integers are relatively prime. Most algorithms are designed to be ultimately implemented as a computer programs.

1.4 STUDY OF ALGORITHM

There are four distinct areas to know about algorithms includes,

1. How to devise algorithms
2. How to validate algorithms
3. How to analyze algorithms
4. How to test a program

1. HOW TO DEVISE ALGORITHMS

To create an algorithm, it may never fully optimized. One of the major goals of an algorithm is to use variety of designing techniques to yield good algorithms. By using dynamic programming techniques, it is used to devise an algorithm (i.e. devise means divided) in a good manner.

2. HOW TO VALIDATE ALGORITHMS

After devising the algorithm, the next field is validating the algorithms. Thus computer will produce correct result for all possible legal inputs.

3. HOW TO ANALYSE ALGORITHMS

Analysis of algorithm is an important part of computer system. It is used to determine the amount of resources such as time and storage necessary to execute. An algorithm can be given in many ways. For example, it can be written down in English or French or any other natural language.

4. HOW TO TEST A PROGRAM

Testing a program consists of two phases: debugging and profiling (or performance measurement). Debugging means to identify errors in a program. Performance measurement is the process of executing a correct program, it measure time and space it takes to compute the results.

1.5 PSEUDOCODE FOR ALGORITHMS

Pseudocode is nothing but the actual code of computer languages such as C, C++ and Java. Let us differentiate between the algorithm and program. Algorithm is a sequence of unambiguous instructions but program means it represents more precise and concise notations called a program.

Another way to represent pseudocode is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. Pseudocode is a generic way of describing an algorithm without use of any specific programming language syntax.

EXAMPLE

To write pseudocode for sum of three numbers using sequence structure

Step 1: Start the program

Step 2: Initialize the variables sum, number1, number2, number3 of type integer

Step 3: Read number1, number2 and number3

Step 4: Calculate $\text{sum} = \text{number1} + \text{number2} + \text{number3}$

Step 5: Print sum

Step 6: End the program

Check your progress 1

1. What is an algorithm? Define its properties.

.....
.....

2. Write an algorithm to find largest of three numbers.

.....
.....
.....

1.6 FLOWCHART

1.6.1 DEFINITIONS OF FLOW CHART

A flow chart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a start description of the process.

1.6.2 BENEFITS OF USING FLOWCHART


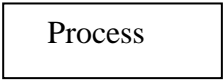
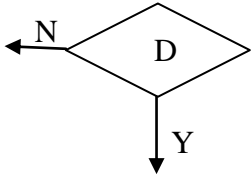
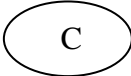
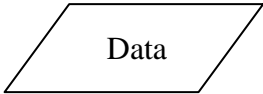
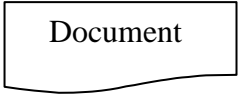
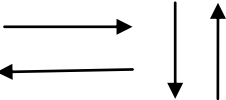
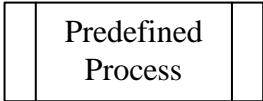
- Correct process understanding
- Provide tools for training
- Identify problem areas and improvement opportunities
- To describe the customer-supplier relationships

1.6.3 SYMBOLS USED IN FLOWCHART

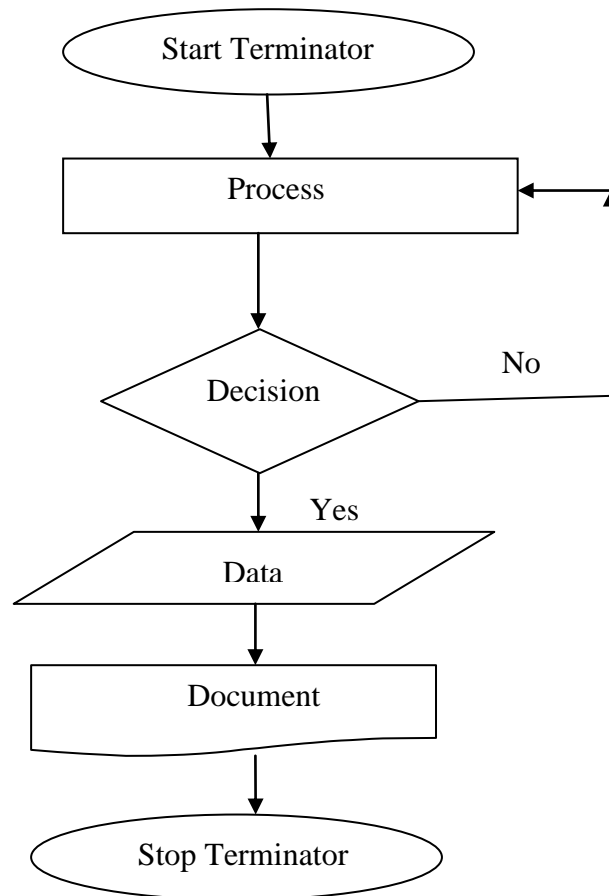
The following symbols that are commonly used in flowcharts they are,

1. **OVAL/TERMINATOR:** Ovals indicates both the start and end of the process.
2. **BOX/PROCESS:** A Rectangular flow shape indicates the activity in the process.
3. **DIAMAND/DECISION:** Diamonds indicates the decision point, such as yes/no or on/off or go/not go.
4. **CIRCLE/CONNECTOR:** A circle indicates the particular step is connected to another page or part of the flowchart.
5. **TRIANGLE/DATA:** A triangle indicates data input or output (I/O) for a process.
6. **DOCUMENT:** A document is used to indicate a document or report.
7. **FLOW LINE/ARROW/CONNECTOR:** An Arrow indicates to show the direction that the process flows.
8. **PREDEFINED PROCESS:** A predefined process is used to indicate a subroutine or interrupt program.

The given table represents symbol, symbol name and function of flowchart.

S.No	SYMBOL	SYMBOL NAME	FUNCTION
1		Oval/Terminator	Start/end
2		Box/Process	Activity in the Process
3		Diamond/Decision	Indicating a Decision Point
4		Circle/Connector	The Particular Step is Connected to Another Page
5		Triangle/Data	Data Input/Output(I/O) for a Process
6		Document	Indicating a Document or Report
7		Arrow/Connector	Direction of Process Flow
8		Predefined Process	Invoke a Subroutine or Interrupt Program

The following flow chart shows the symbolic representation of flow diagram.



1.6.4 RULES FOR CREATING FLOWCHART

1. All boxes in the flowchart are connected with arrows
2. Flowchart symbols have an entry point on the top of the symbol only. The exit point for all flowchart symbols is on the bottom except for decision making
3. Decision symbol have two exit points:
Yes/no **or** true/false **or** on/off
4. Flow chart will flow from top to bottom
5. Connectors are used to connect with 3 manners,
 1. from one page to another page
 2. from the bottom of the page to the top of the same page
 3. Upward flow of more than 3 symbols

1.6.5 ADVANTAGES OF FLOWCHART

The advantages of flowchart are,

1. **COMMUNICATION:** Flowchart is used for better way of communication in all connections.
2. **EFFECTIVE ANALYSIS:** By using flowchart, problems are analyzed in better manner.
3. **PROPER DOCUMENTATION:** Flowchart serves as good program documentation.
4. **EFFICIENT CODING:** The flowcharts act as a guideline during the system analysis and development phase.
5. **PROPER DEBUGGING:** The flowchart helps in debugging process.
6. **EFFICIENT PROGRAM MAINTENANCE:** The maintenance of operating program Is an easy way for drawing the flowchart.

1.6.5 LIMITATIONS OF FLOWCHART

Some of the limitations of flowchart are,

1. **COMPLEX LOGIC:** The program logic is very complicated; in this manner drawing flowchart is difficult
2. **ALTERATIONS AND MODIFICATIONS:** Alteration and modifications cannot be made and hence flowchart is very complex process
3. **REPRODUCTION:** In flowchart, symbols cannot be typed, hence it becomes a problem

1.7 PSEUDOCODE EXAMPLE

To write pseudocode for to prepare a student mark sheet processing using control statement

Step 1: Start the program

Step 2: Initialize the variables rollno, stud_name, m1, m2, total,
Average and result

Step 3: Read the values for rollno, stud_name, m1, m2

Step 4: To calculate total=m1+m2

Step 5: To calculate average=total/2

Step 6: To **calculate** result based on two subjects

```
if(m1>40) and (m2>40) then
    print result="pass"
else
    print result="fail"
end if
```

Step 7: Print rollno,stud_name,m1,m2,total,average and result

Step 8: Stop the program

1.8 DIVIDE AND CONQUER STRATEGY

A divide and conquer strategy process is given below. If the problem is small it's solved directly. If the problem is large, it's divided into two or more parts called subprograms. Each subprogram is solved after solutions to the subproblems are combined into a solution to the original problem. The divide and conquer strategy is used in same process that is to solve the subproblems are further divided into several subproblems and so on. The solutions to the various subproblems are the combined into a solution to the original problem. Recursion is a method it is used to solve a subprogram. An example is array. An array of two or more elements can be sorted by using a divide and conquer strategy. In this manner to use **merge sort** method for the sorted array of elements where it combines into original elements.

DIVIDE: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.

CONQUER: Sort the two subsequences recursively that is by using merge sort again on the Subsequences. Merge sorts on latter.

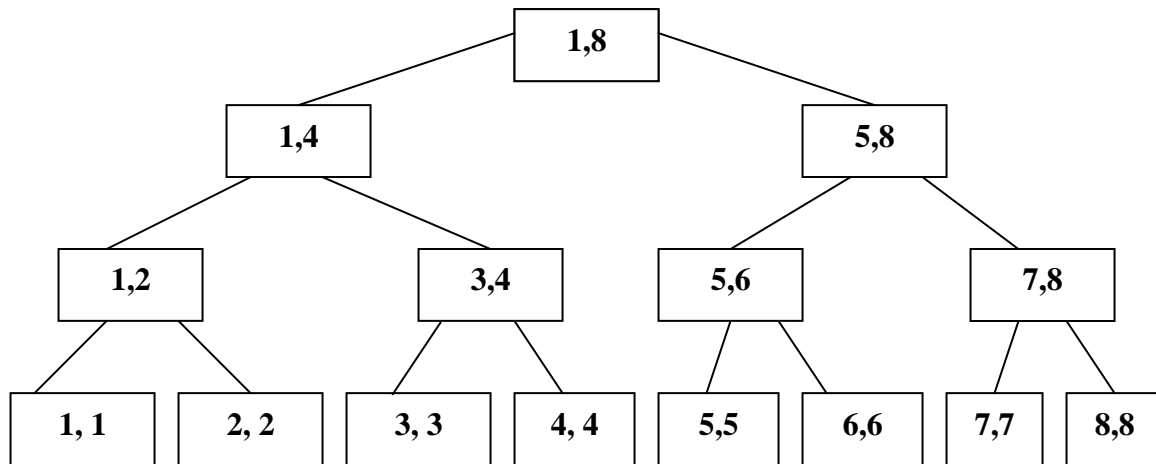
A COMBINE: Merge the two sorted subsequences to produce the sorted answer.

Merge sort follows **divide and conquer strategy**

1.8.1 MERGE SORT

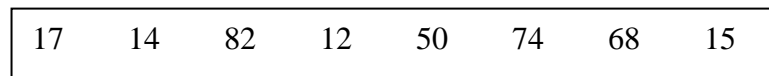
Consider the array of ten element $a[1:10]=(17,14,82,12,50,74,68,15)$. In merge sort method by splitting number of elements into two subarrays each of size is four $a[1:4]$ and $a[5:8]$. The elements in $a[1:4]$ are then split into two subarray of size two ($a[1:2]$) and another ($a[3:4]$). The same manner element split into right on the side split into two subarray of size two ($a[5:6]$) and another ($a[7:8]$). Sort all the sub array of elements after merging into two sub array of elements. Finally, the elements are resulted in sorting manner.

The Structure of Merge Sort:

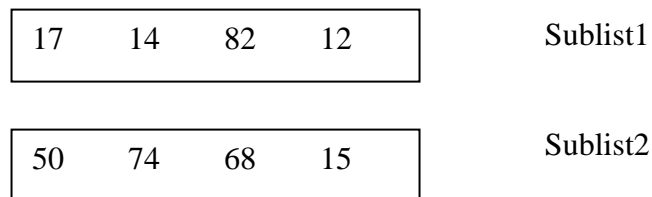


Merge Sort-Visual Example

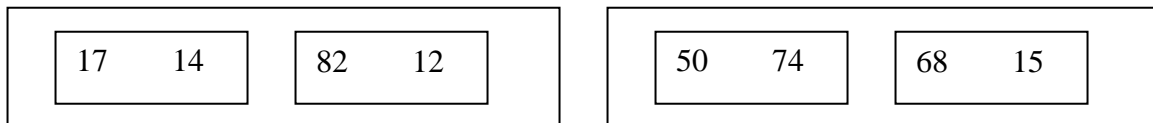
Take eight elements in a list



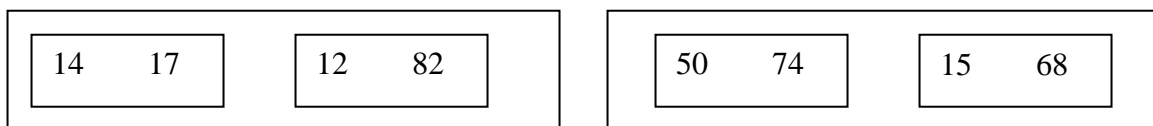
Divide the list into two sublists



Splitting sublists into smaller sublists until we can't split any further,

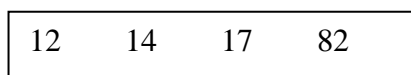


We can swap the element until the sorted order

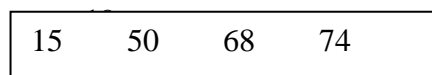


We can repeat the process again until two sublists in sorted order

Sorted sublist-1



Sorted sublist-2



We can merge sorted sub list together so that we can get original list of sorted element

12	14	17	50	68	74	68	82
----	----	----	----	----	----	----	----

Merge Sort Pseudocode

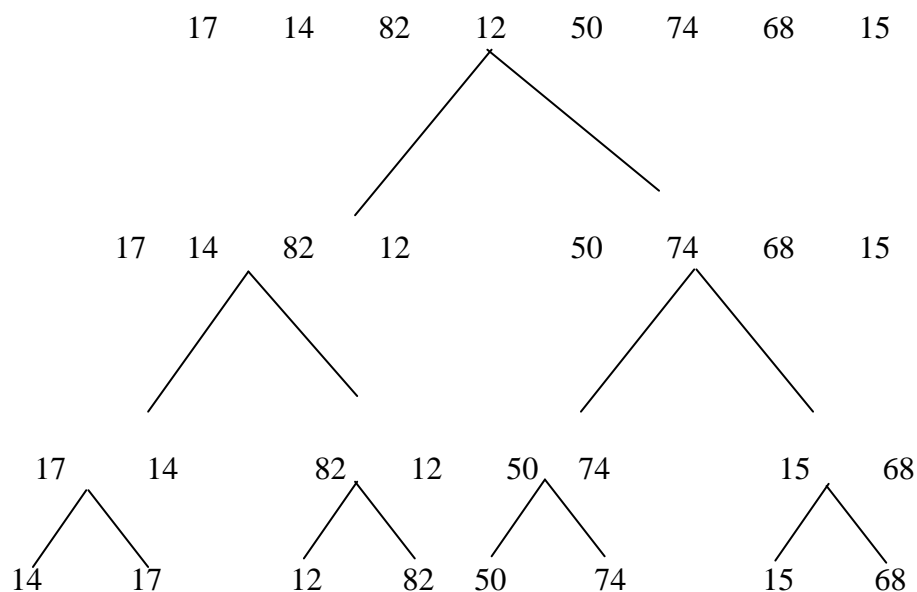
The mergesort algorithm can be written very simply as given below,

```

Mergesort(list[],leftindex,rightindex)
Begin
  If leftindex<rightindex then
    mid=(leftindex+rightindex)/2
    mergesort(list[],leftindex,mid)
    mergesort(list[],mid+1,rightindex)
    merge(list[],leftindex,mid,rightindex)
  end if
End
  
```

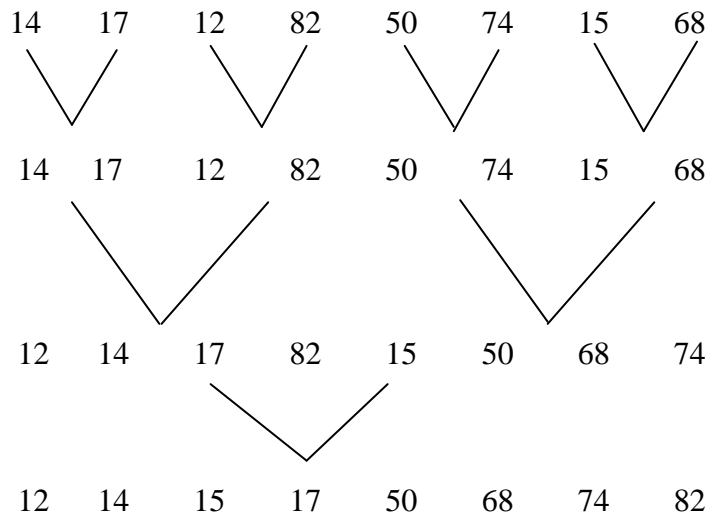
Once the array is recursively split into single elements is recursively split into single element list (14,17,12,82,50,74,15,68) the recursion algorithm with sort the individual arrays and combine them .Here,17,14 are not sorted order.

Graphical representation of above example,



Once the array is recursively split into two list, first list are(17,14,88,12) and the second list is(50,74,68,15) by using recursion algorithm will sort the individual arrays and combine them are(12,14,15,17,50,68,74,82).Here first sorted element are 14 and 17 in the first sublist after sorted elements are 12 and 82 in the sublist, after sorted the element are 50 74 in the second sublist finally 15 and 68 already sorted order in second sublist. Finally combine the sorted elements.

Merging the two sorted elements for graphical representation,



The final array is the array which contains the sorted elements.

SUITABILITY

Merge sort is good for data having big memory at once, because its pattern of storage access is very regular. It uses even fewer comparisons than heap sort, and is especially suited for data stored as linked list.

ADVANTAGES

- Slightly faster than the heap sort for larger sets
- It is suitable sort

DISADVANTAGES

It requires twice the memory of the heap sort because the second array is used to store the sorted list. Like the quick sort, the merge sort is recursive which can makes it as a bad choice for applications that run on machines with limited memory.

COMPLEXITY

The sorting method in merge sort involves sorting of two equal parts of array and combining two sorted lists into one sorted list. Let $T(N/2)$ is the time taken to sort each half part of array and N is time required to combine two sorted lists into a single sorted list. Then

$$T(N) = T(N/2) + T(N/2) + N$$

$$= 2T(N/2) + N$$

$$T(N)/N = T(N/2)/(N/2) + 1$$

Assume the total number of elements N is power of 2, such that $N = 2^M$ (i.e., $M = \log_2 N$), there are M steps to add since the array is divided into exactly half. So,

$$T(N)/N = T(1)/1 + C \log^2 N$$

$$T(N) = N + CN \log_2 N$$

$$= O(N \log_2 N)$$

The complexity of merge sort in worst case is $O(N \log_2 N)$

1.9 WRITING ALGORITHMS AND FLOWCHARTS FOR SIMPLE EXERCISES

EXAMPLE 1

Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

ALGORITHM

Step 1 : Start the program

Step 2 : Input m_1, m_2, m_3, m_4

Step 3 : $\text{Grade} \leftarrow (m_1 + m_2 + m_3 + m_4) / 4$

Step 4 : if (grade < 40) then

Print "FAIL"

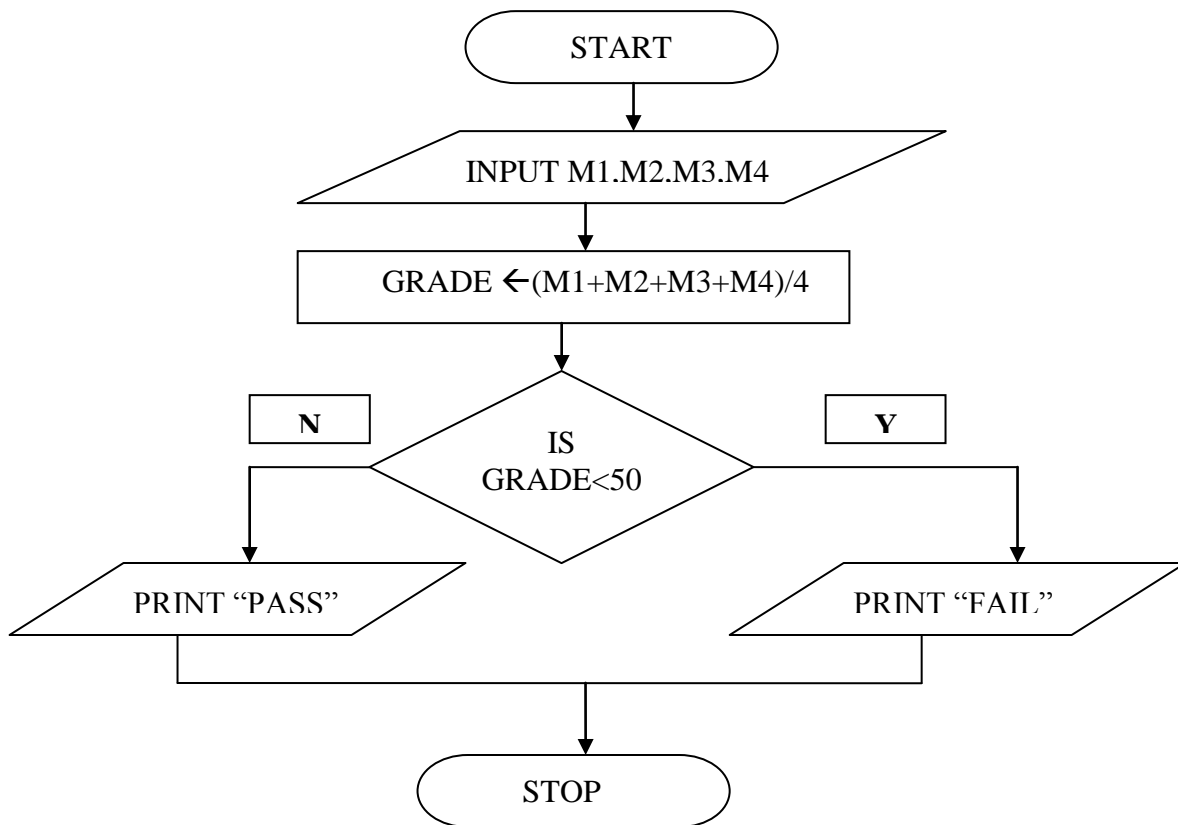
else

Print "PASS"

end if

Step 5 : Stop the program.

FLOWCHART



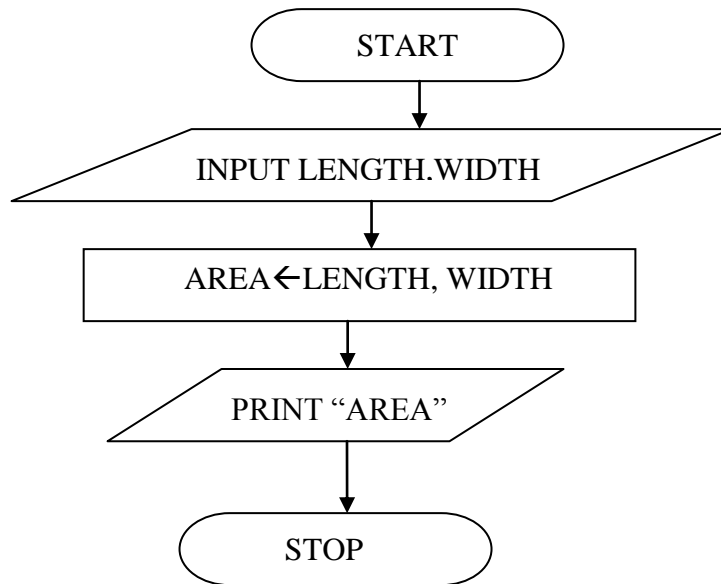
EXAMPLE 2

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate area.

ALGORITHM

- Step 1:** Start the program
- Step 2 :** Input width and length
- Step 3:** Calculate Area- \leftarrow length*width
- Step 4:** Print Area
- Step 5:** Stop the program

FLOWCHART



EXAMPLE 3

Write an algorithm that reads two values, determines the largest value and print message largest value.

ALGORITHM

Step 1: Start the program

Step 2 : Input value1,value2

Step 3: if (value1>value2) then

 max ← value1

else

 max ← value2

end if

Step 4: Print "The Largest Value is:", Max

Step 5: Stop the Program

Check your progress 2

1. Define flowchart? And mention the rules for creating flowchart.

.....

.....

.....

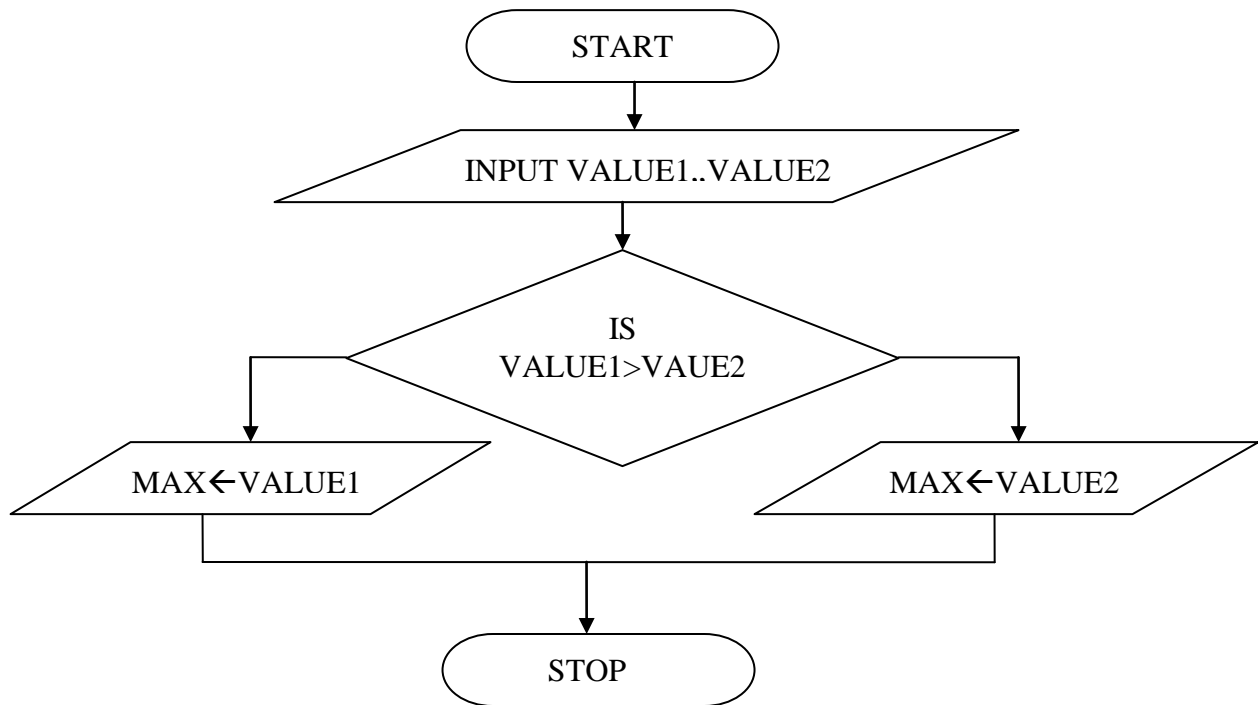
2. Mention the advantages and disadvantages of flowchart.

.....

.....

.....

FLOWCHART



1.10 Let us Sum Up

- An algorithm is a sequence of unambiguous instructions
- Algorithm has some properties such as input,output,definiteness,finiteness,correctness
- Pseudocode is nothing but the actual code of computer languages.
- Flowchart is a graphical representation of process
- Divide and Conquer strategy is a process
- Divide means divide the problem in n-element of sublist

- Conquer means sorting the n-element sequence
- Merge means two sorted sub list can be combined.

1.11 LESSIONS END ACTIVITIES

1. Fill in the Blanks

- A. An algorithm is_____.
- B.Algorithm can be specified in a natural language or a _____.
- C.Flowchart is a _____.
- D. An array of two or more elements can be sorted by using a_____ Strategy.

2. Answer the following Question

- 2.1 Explain different symbols and function of flowchart.
- 2.2 Write an algorithm and draw flow chart that will be area of circle and square.
- 2.3 Explain in detail about divide and conquer strategy with example.
- 2.4 Explain detail about fundamentals of algorithmic problem solving process.

1.12 KEYWORDS

- ❖ **An Algorithm** is a sequence of no ambiguous instructions for solving a problem in a finite amount of time.
- ❖ **Algorithm** can be specified in a natural language or a pseudocode
- ❖ Several ways to classify algorithms
 - Algorithms according to types of problems they solve.
 - Algorithms according to underlying design techniques.
- ❖ **Algorithms Design Techniques** (or design strategies or paradigms) are general approaches for solving problems algorithmically.
- ❖ **A Good Algorithm** is usually a result of repeated efforts and rework.
- ❖ **Algorithms Operate** on data.
- ❖ **A Flowchart** is a graphical or symbolic representation of a process.
- ❖ **Oval** Ovals indicates both the start and end of the process.
- ❖ **Rectangle** A Rectangular flow chart shape indicating activity in the process.
- ❖ **Diamond** A Diamond indicating decision point.

- ❖ **Arrow** indicates show the direction that the processes flow.
- ❖ **Document** A Document is used to indicate a document or report.

Check your Progress: Model Answers

Ans 1

1. An algorithm is a sequence of unambiguous instructions for solving a problem .
Algorithm has the following properties:
 1. Input: The algorithm get input
 2. Output: The algorithm produces output
 3. Definiteness: Each instruction to represent with clear and unambiguous
 4. Finiteness: The algorithm terminates that is it terminate after finite number of steps
 5. Correctness: The produced output by the algorithm is correct
2. Step 1: Read x, y, z
Step 2: To compare (($x > y$) and ($x > z$))


```

      print x
      elseif( $y > z$ )
      print y
      else
      print z
      
```

Ans 2

1. A flowchart is a graphical representation
Rules for creating Flowchart:
 1. All boxes of the flowchart are connected with arrows
 2. Flowchart symbols have an entry point and exit point
 3. Decision symbol have two exit points
 4. Flow chart will flow from top to bottom
 5. Connectors are used to connect with same or other pages of flow
- 2.(i) The advantages of flowchart are,

1. Communication	2. Effective Analysis
3. Proper Communication	4. Effective Coding
4. Proper Debugging	5. Efficient Program
- (ii) Limitations of flow chart are,

1. Complex Logic	2. Alterations and Modifications
3. Reproduction	

1.13 QUESTIONS FOR DISCUSSION

1. What is an algorithm? Give its properties and explain with example.
2. Write short notes on:
 - a. Different distinct areas of algorithm
 - b. Steps for creating and designing algorithm
3. What is flowchart? Briefly explain different symbols used in flowchart with Example.
4. Discuss the following
 - a. Rules for creating flowchart
 - b. Advantages of flowchart
 - c. Limitations of flowchart
5. Briefly explain techniques of divide and conquer strategy.

1.12 SUGGESTED READING

1. Fundamentals of Computer Algorithms-Ellis Horowitz,Sartaj Sahni and Sanguthevar Rajasekaran,Galgotia Publications Pvt Ltd.
2. Computer Algorithms-Sara Baase, Allen Van Gelder,thrid edition-Pearson Education.
3. Algorithms-David Harel, Second Edition-Pearson Education.
4. Introduction to Algorithms, Cormen, Charless E.Leiserson,Ronald L.Rivest,Clifford Stein,PHI.
5. Design and Analysis of Algorithms, Dexter C.Kozen-Springler-Verlag.

UNIT-2

UNIT

2

CONSTANTS VARIABLES AND DATA TYPES

CONTENTS

- 2.1 Aims and Objectives
- 2.2 Introduction
- 2.3 Characteristics of C
- 2.4 Current uses of C
- 2.5 Format of C Program
- 2.6 Features of C
- 2.7 C Character Set
- 2.8 C Tokens and Examples
- 2.9 Identifiers and Keywords
- 2.10 Constants
- 2.11 Variables
- 2.12 Data Types
- 2.13 Declaration of variables
- 2.14 Let us Sum up
- 2.15 Lesson and Activities
- 2.16 Keywords
- 2.17 Questions for Discussion
- 2.18 Suggested Readings

2.1 AIMS AND OBJECTIVES

At the end of this chapter you will learn, how to:

- Describe the characteristics and uses of C
- Understand format of C program with features and examples
- Describe C Character Set and tokens
- Identify constants and data types
- Appreciate the need of declaration of variables.

2.2 INTRODUCTION

The C programming language was developed by Dennis Rictchie in Bell Telephone Laboratories in early 1970's. Usage of this language was largely confined to Bell Laboratories until 1978, when Brian Kernighan and Dennis Ricthie described the C programming language. The original form of the C language is referred to as K & RC.

A programming language is designed to help process certain kinds of data consisting of numbers, characters and string and to provide useful output known as information. The task of processing the data is accomplished by executing a sequence of precise instructions called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules (or grammar). Every program instruction must confirm precisely to the syntax rules of the language.

2

.3 CHARACTERISTICS OF C

C has a variety of language characteristics that contains large consequence of the original constructed development, original constructed development environment and its original applications implementation of an operating system.

Some of the most significant characteristics of the languages are listed below:

1. SIZE OF LANGUAGE

C is an externally small language does not have any built-in input/output capabilities it contains no string handling functions and arithmetic operations beyond those such as basic addition or subtraction provides this functionality through a rich set of function libraries. Most C implementations include standard libraries for input/output, string manipulations and arithmetic operations is characterized by the ability to write concise source programs program use and extensive usage of function calls programs are highly portable.

2. MODERN CONTROL STRUCTURES

C contains all the control structures expected in a modern language. For loops, if-else constructs, case (switch) statements, and while loops are all part of the language.

3. BITWISE CONTROL

To perform systems programming, it is frequently necessary to manipulate the objects at the bit level. C provides variety of operators for bitwise manipulation of data.

2.4 CURRENT USES OF C

C is used for system applications and it is language of choice to most UNIX users. A Systems program forms a portion of the operating system of the computer or its support utilities. System programs include:

1. Operating System
2. Interpreters
3. Editors
4. Assembly Programs
5. Compilers

Because of its portability and efficiency C is also used in a variety of applications running under nearly all operating systems.

The C language is used for developing

- Database Systems
- Graphics Packages
- Spread Sheets
- Word Processors
- Office Automations
- Scientific & Engineering Applications

2.5 FORMAT OF C PROGRAM

A Program written in the C language must have a few basic components. The format of C program given below,

```
void main()
{
    Variable declaration;
    Program statements;
}
```

Variable Declaration

All variable used in C language programs must declared C variable declarations include the name of the variable and its types.

Program Statements

Program statements or executable statements must have variable declarations. An executable statement is an expression followed by semicolon or a control construct such as IF or a WHILE statement.

Most program and supporting functions contain program comments whose purpose is to make the program more understandable. Program comments may contain any message starting with the character sequence “/* and ending with the sequence “*/”.

A SAMPLE C PROGRAM

EXAMPLE 1

```
#include<stdio.h>
void main()
{
    printf(“Welcome to C Programming:\n”);
}
```

Sample Program Output

Welcome to C Programming

EXAMPLE 2

```
#include<stdio.h>
void main()
{
    printf(“Programming in C is easy \n”);
    printf(“and also Programming in C++.”); } }
```

Sample Program Output

Programming in C is easy
and also Programming in C++.

EXAMPLE 3

```
#include<stdio.h>
void main()
{
    printf("Hello...\n..Oh my \n when do I stop?. \n");
}
```

Sample Program Output

Hello...
..Oh my
...when do I stop?

2.6 FEATURES OF C

The following features can be listed for popularity of C.

1. Portability
2. Flexibility
3. Wide acceptability
4. Modern Control and flow structures
5. Rich set of operators

1. PORTABILITY

Program written in these languages can be executed in different computers and operating systems if compilers are available for these systems. The ability of a program is to run in different environments. A different environment refers to different computers, operating systems or different compilers. Machine language or assembly language varies from computer to computer and hence programs written in these languages are not portable can be termed as most portable language.

2. FLEXIBILITY

C combines the convenience and portable nature of a high-level language with the flexibility of a low-level language.

3. WIDE ACCEPTABILITY

This language is suitable for projects both at the system-level (building operating system or compilers) and at the application level (building graphical user interfaces).

4. MODERN CONTROL AND FLOW STRUCTURES

In C, it can support set of control and flow structures such as if...statement, select case statement.

5. RICH SET OF OPERATORS

Programs written in these languages can be executed using set of operators and expressions such as arithmetic operators, logical operators, bitwise and shift operators etc.

2.7 C CHARACTER SET

The C character set consists of upper and lower case alphabets, digits, special characters and white spaces. The alphabets and digits are together called as alphanumeric characters.

The character set in C are divided into following types:

1. Letters
2. Digits
3. Special Characters
4. White Spaces

(1)LETTERS

Uppercase A....Z
Lowercase a....z

(2)DIGITS

All decimal digits 0.....9

(3)SPECIAL CHARACTERS

, Comma	< Opening angle bracket	> Closing angle bracket
. Period	- Underscore	(Left parenthesis
; Semicolon	\$ Dollar sign) Right parenthesis
# Number Sign	% Percent sign	[Left bracket
` Apostrophe	& Ampersand] Right bracket
? Question Mark	^ Caret	(Left bracket
! Exclamation Mark	* Asterisk) Right bracket
Vertical Bar	- Minus Sign	/ Slash
~ Tilde	+ Plus Sign	\ black slash

(4)WHITE SPACES

Blank space new line carriage return form feed horizontal tab and vertical tab.

2.8 C TOKENS AND EXAMPLES

Individual words and punctuation marks are called tokens. In “C Programming” language smallest individual units are called C tokens.

The types of tokens are,

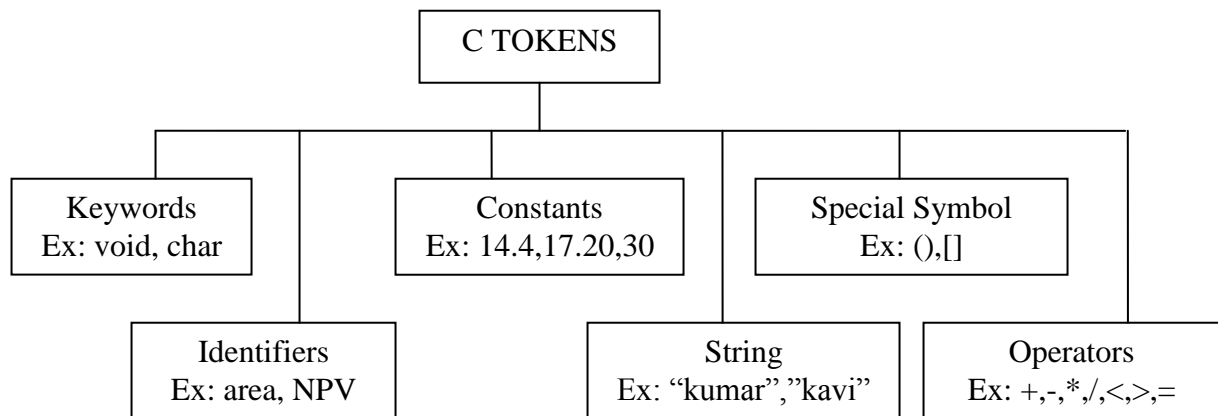


FIGURE 1.3

2.9 IDENTIFIERS AND KEYWORDS

In C programming language every word/string is classified as identifier or keyword. All keywords have fixed meaning and these meanings cannot be changed. There are certain rules regarding identifier name, are listed below.

1. First character must be an alphabet or underscore.
2. Identifiers must consists of letters, digits or underscore
3. Identifier name should not be the keyword(such as int,float,double,if,else, Const, for, goto)
4. Only allowed name of variable as 3 characters, C is case sensitive (i.e., upper and lower case letters are treated separately

2.10 CONSTANTS

In C programming language the fixed values do not change during the execution of a program. In C supports several types of constants are listed below,

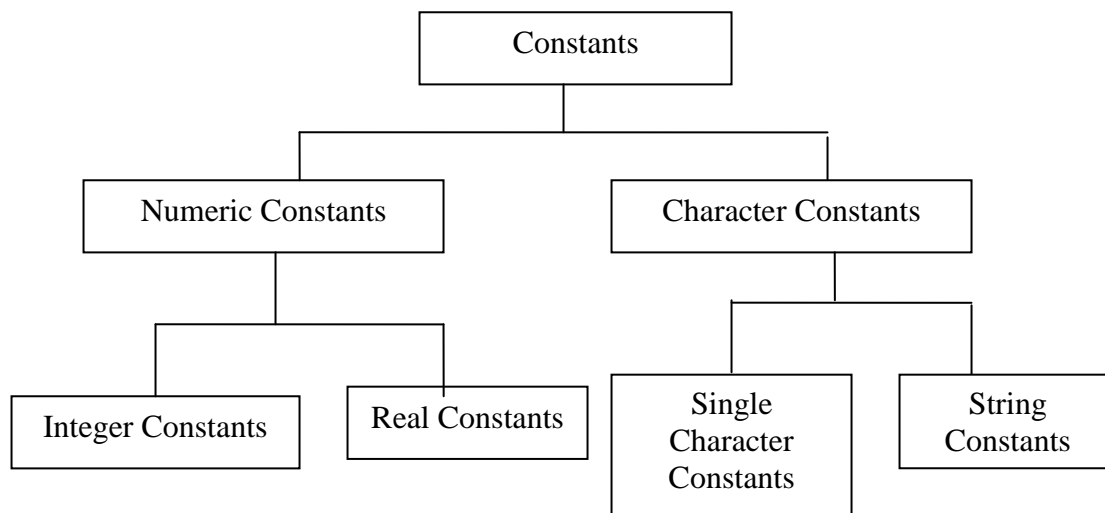


FIGURE 1.4

Integer Constants

Integer constants refer to a sequence of digits. In integer constants can be specified in three ways,

1. Decimal Numbers
2. Octal Numbers
3. Hexadecimal Numbers

For example 25 is a decimal numbers in base 10. In octal numbers are specified with a leading zero, rest of the digits consists of those lying between 0 and 7. For example, 0125. The hexadecimal numbers are specified with 0x or 0X the beginning. The digits that follow 0x must be numbers in the range 0-9 or one of the letters a-f or A-F. For example, 0x125.

An integer constant can be appended at the end of the constant. The suffix u is used for unsigned int constants for long int constants and l for short int constants. Either of the cases can be used.

1. Unsigned integer constants
47424U
47424u
2. Long integer constants
74524L
74524l
3. The suffixes can be combined in any order
647272UL
647272ul
4. Short integer constants
140S
140s
5. Unsigned short integer constants
170S
170s

Real Constant

In real constants have a decimal point or an exponential sign or both.

Decimal Notation

The number is represented as a whole number followed by a decimal point and a fractional part. It is possible to omit digits before or after decimal point.

Valid floating point constants are:

247.0, 0.416, -0.22 and +0.22

Exponential Notation

Exponential Notation is useful in representing numbers whose magnitudes are very large or very small. The exponential notation consists of a mantissa and an exponent. The exponent is positive unless preceded a minus sign. Consider the number 472.45. This can also be written as 0.47245×10^3 representing the number 0.47245×10^3 . The sequence of 47245 after the decimal point of the number 0.47245 is the mantissa, and 3 is the exponent.

Another example, the number 62000000000 can be written as 62×10^9 or 0.62×10^{-9} . Similarly 0.00000000045 can be written as 0.45×10^{-9} .

EXAMPLE

20.04
 4.7×10^{-4}
3E8

Single Character Constant

A single constant is enclosed within single quotes.

EXAMPLE

'a' '4' '\n'

Note

Character constant represents an integer value; it is also possible to perform arithmetic operations.

String Constant

A string constant is a sequence of character enclosed in double quotes

EXAMPLE

"WELCOME" "2009" "XYZ"

2.11 VARIABLES

Variable is a name that can be used to store data value. A variable can have only one value assigned to it any given time during the execution of the program.

EXAMPLE

- **MARKS**
- **AVERAGE**
- **TOTAL**

Rules for declaring variables

- ✚ Identifiers are used for name of variable
- ✚ A variable name consists of a sequence of letters or digits
- ✚ First letter start with alphabet

(i)Valid variable name

- (1)student_name
- (2)emp_name
- (3)mark

(ii)Invalid variable name

- (1)3a
- (2)salary#
- (3)student-name

2.12 DATA TYPES

Data types are used to store various types of data that is processed by program. Data type attaches with variable to determine the number of bytes to allocate the variable and valid operations which can be performed. It supports various data types such as character, integer and floating point types.

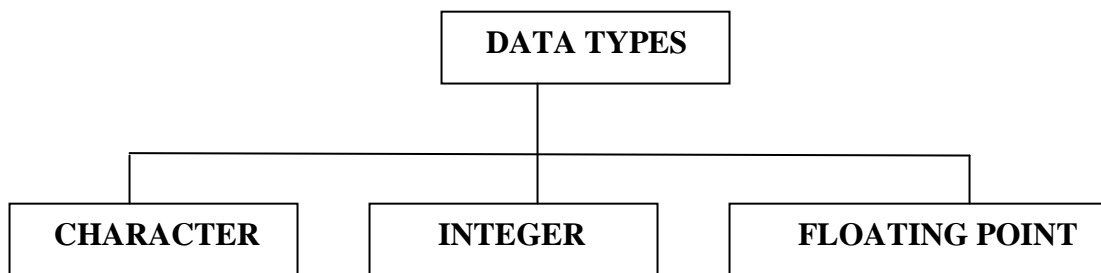


FIGURE 1.5

(i) Character Data Type

C Stores character type internally as an integer. Each character has 8 bits. So we can have 256 different character values 0 to 255.

(ii) Integer Data Type

Integer data types are used to store numbers and characters. Here is the able of integer data type in various forms:

Data Type	Memory Allocation	Range
Signed int	1 byte	$-2^7 - 1$ (-128 to 127)
Unsigned int	1 byte	0 to $2^8 - 1$ (0 to 255)
Short	2 bytes	-2^{15} to $2^{15} - 1$ (-32768 to 32767)
Unsigned short	2 bytes	0 to $2^{16} - 1$ (0 to 65535)
Long int	4 bytes	2^{31} to $2^{31} - 1$ (-2147483648 to 2147423647)
int	2 or 4 bytes	Ranging for 2 or 4 bytes

(iii) Floating point data type

The floating point data types are used to represent floating point numbers. Floating point data types come in three sizes. Float (single precision) double (double precision) and long double (extended precision).

2.13 DECLARATION OF VARIABLES

Declaration of variables in 2 manners,

1. Primary type declaration
2. User-defined type declaration

Primary Type Declaration

A variable can be used to store a value of any data type. That is the name has nothing to do with any data type.

SYNTAX

Data-type variable name1,variable name2...variable name;

EXAMPLE

```
int marks;  
int rollno,average;  
double exam-fees;
```

int and double are keywords to represent integer type and real type data values respectively. In the table given below various data types and their keyword equivalents

Data types and their keywords

Data Type	Keyword Equivalent
Character	Char
Unsigned character	Unsigned
Signed character	Signed char
Signed short integer	Signed short int (or long int or long)
Signed long integer	Signed long int (or long int or long)
Unsigned integer	Unsigned int(or unsigned)
Unsigned short integer	Unsigned short int (or unsigned short)
Unsigned long integer	Unsigned long int (or unsigned long)
Floating point	Float
Double precision floating point	double
Extended double precision floating point	Long double

Declaration

C supports a feature known as “type definition” that permits users to define an identifier represents an existing data type.

SYNTAX

typedef type identifier;

where,

typedef->keyword

type->refers to an data type

identifier->refers to the name of variable name

EXAMPLE

```
typedef float average;
```

```
typedef int block;
```

Here average declared as float and block declared as int. They can be classified into further manner as,

```
block unit1, unit2;
```

The main advantage of typedef is can create meaningful data type for increasing the readability of the program.

Check Your Progress 1

1. What is constant? Explain its types.

.....
.....
.....

2. What is variable? Define rules for creating variable.

.....
.....
.....

2.14 LET US SUM UP

- C Programming language was developed by Dennis Rictchie at Bell Telephone laboratories in early 1970s.
- Various characteristics of C are size of language, modern control structures and bitwise control.
- C Language used for developing such as database systems, graphics packages, spread sheets, word processing, office automation, scientific and engineering application.
- The format of C program are variable declaration and program statements/
- The following popularity features of C are portability,flexibility,wide acceptability, modern control & flow structures, rich set of operators
- The c character set are divided into following types such as letters, digits, special characters and white spaces.
- Individual words and punctuation marks are called tokens such as keywords, identifiers, constants, string, special symbol and operators.
- In C programming every word/string classified into identifier or keyword.
- Constants in C programming fixed values that do not change during the execution of a program support several types of constants such as numeric constants and character constant.
- Variable is a name to store data value assigned to a variable during execution of a program.
- Data types are used to store various types of data that is processed by program. Variable support various data types such as character, integer and floating point types.
- In C programming declaration of variable in two manners such as primary type declaration and user-defined type declaration.

2.15 LESSON END ACTIVITIES

1. Fill in the Blanks

- A.C programming language was developed by_____.
- B.The alphabets and digits are together called_____.
- C.Individual words and punctuation marks are called_____.

2. What is meant by variable? Explain declaration of variables with example.
3. Explain different characters of C
4. What is mean by character set? Explain its types.
5. What is a token? Explain its types with example.

2.16 KEYWORDS

- ❖ **Character set:** Character set consists of upper and lower case alphabets, digits, special characters and white spaces.
- ❖ **C Tokens:** Individual words and punctuation marks are called tokens.
- ❖ **Identifiers:** Every word/string classified as identifier or keyword.
- ❖ **Constants:** Fixed values that do not change during execution of program.
- ❖ **Real Constant:** In real constants have a decimal point or exponential sign or both.
- ❖ **Decimal Notation:** The number is represented followed by decimal point and fractional part.
- ❖ **String Constant:** A string constant is a sequence of character enclosed in double quotes.
- ❖ **Variable:** A variable name that can be used to store data value.
- ❖ **Data types:** Data types are used to store various types of data is processed by the program.
- ❖ **Single Character Constant:** A single constant is enclosed within single quotes.

2.17 QUESTIONS FOR DISCUSSION

1. What are different data types in C?
2. What do you mean by identifier and keywords?
3. What are the applications of C programming?
4. How can we create a C PROGRAM?
5. Write a C program display a message “Programming in C”.

Check your Progress: Model Answers

Ans.1

1. C programming language fixed values that do not change during execution of a program. Several types of constants are
 1. Numeric Constant
 2. Character Constant
2. Variable is a name that can be used to store data value.

Rules for creating value:

- ❖ Identifiers are used to name of variable.
- ❖ Variable name consists of a sequence of letters or digits.
- ❖ First letter start with alphabets

2.18 SUGGESTED READING

1. Mastering C by Venugopal, Prasad – TMH
2. Complete reference with C Tata McGraw Hill
3. C – programming E.Balagurusamy Tata McGray Hill
4. How to solve it by Computer : Dromey, PHI
5. Schaums outline of Theory and Problems of programming with C : Gottfried
6. The C programming language : Kerninghan and Ritchie
7. Programming in ANSI C : Ramkumar Agarwal
8. Mastering C by Venugopal, Prasad – TMH
9. Let Us C by kanetkar
10. An introduction to data structures with applications, Jean-Paul Trembly and Paul Sorenson, (2nd edition), 1884

UNIT-3

UNIT

3

OPERATORS AND EXPRESSIONS

CONTENTS

- 3.1 Aims and Objectives
- 3.2 Introduction
- 3.3 Arithmetic Operators
- 3.4 Unary Operators
- 3.5 Relational Operators
- 3.6 Logical Operators
- 3.7 Assignment Operators
- 3.8 Increment and Decrement Operators
- 3.9 Bitwise Operators
- 3.10 Comma Operators
- 3.11 Special Operators
- 3.12 Arithmetic expression
- 3.13 Evaluation of Expression
- 3.14 Precedence of Arithmetic Operators
- 3.15 Type Conversion in Expression
- 3.16 Operator Precedence and Associativity
- 3.17 Let us Sum up
- 3.18 Lesson and Activities
- 3.19 Keywords
- 3.20 Questions for Discussion
- 3.21 Suggested Readings

3.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to;

- Describe the operators and its types
- Identify the need of arithmetic expressions
- Precedence of arithmetic operators
- Understand the concept of type conversion
- Need of operator precedence and associativity

3.2 INTRODUCTION

C supports different type of operators. In operators is a symbol that operates certain mathematical or logical operations. These operators are being:

1. Arithmetic operators
2. Unary operators
3. Relational operators
4. Logical operators
5. Assignment operators
6. Increment and Decrement operators
7. Conditional operators
8. Bitwise operators
9. Comma operators
10. Special operators

3.3 ARITHMETIC OPERATORS

The arithmetic operators can perform arithmetic operations and can be classified into unary and binary arithmetic operators.

(i) Unary Arithmetic Operators

The use of unary \pm does not serve any purpose by default, numeric constants are assumed to be possible. However it can be used as follows:

$A=+50$

The unary minus operator can be used to negate the value of variable. It is also used specify a negative number. Here a minus sign (-) is prefixed to the number. The unary minus operation has 0 the effect of multiplying its operand by -1.

EXAMPLE

1.int x=10

int y=-x

The value of y=10

2.int x=7

int sum=-x

The value of sum=-7

(ii) Binary Arithmetic Operators

There are five arithmetic operators in C are given below,

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus operators (Remainder after integer division)

The operator acted upon by arithmetic operators must represent numeric values. Thus; the operator can be integer values, floating point values or characters. The modulus operator (%) requires both the operands to be integers and the second operand as a non zero. If one of the operand represents negative values, then the addition, subtraction, multiplication and division operators will result in values whose signs are determined by the rules of algebra.

EXAMPLE

If a and b two integer variables whose values 5 and 10 respectively, then

Expression	Value
a+b	15
a-b	7
a*b	50

a/b	2
a%b	1

If x and y are two floating point variables whose values are 10.7 and 5.5 then

Expression	Value
x+y	16.2
x-y	5.2
x*y	58.85
x/y	1.9

If c1 and c2 are two characters variables that represent the characters P and T respectively, then

Expression	Value
C1	80
C1+C2	164
C1+C2+5	169
C1+C2+'5'	217

3.4 UNARY OPERATORS

Unary operators are another type of operators that act upon a single operand to produce a value.

Operator	Meaning
-	Unary minus
++	Increment operator
--	Decrement operator
size of	Size of the operand

The unary minus is minus sign which precedes as a numerical constant a value or an expression. Unary minus, negates the value of the number. Examples are -5,-(a+b),-0.5,-root2.

If a=-2,b=7,c=-3 then the values of a,b and c would be 2,=7 and 3 respectively.

The increment operator ++ causes its operand to be increment by one where as the decrement operator -- causes its operand to be decrement by one. These operators can each be utilized in two different ways, depending on whether the operator is written before the operand. If the operator precedes the operand (++i), then the operand will change in value before it is utilized for its intended purpose within the program. These operators are called pre increment and decrement operators.

3.5 RELATIONAL OPERATORS

The relational operators are used to compare arithmetic, logical and character expressions. We can compare two similar values and depending on their relation take some decision. These comparisons can be done with the help of relational operators. Each of these operators compares their left hand side with their right hand side. The relational operator then evaluates to an integer. It evaluates to zero if the condition is false, and one if it is true.

There are six relational operators in C

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to

If i, j and k integer variables having the values 1, 2 and 3 respectively, then

Expression	Interpretation	Value
i < j	True	1
(i+j) >= k	True	1
(i+k) > ((i+5))	False	0
K != 3	False	0
J == 2	True	1

3.6 LOGICAL OPERATORS

The C language logical operators allow a programmer to combine simple relational expressions to form complex expressions by using AND, OR and NOT. In C there are the following three logical operators.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

(i) Logical AND

Taking logical AND when two expressions are added the resulting expression will be true only if both sub expressions are true. Example such as (x > 15) && (y < 10) will be true only if x is greater than 15 and if y less than 10. The second expression is not evaluated if the first expression is false.

EXAMPLE

```
if(number<0 && number>100)
```

(ii) Logical OR

A logical expression is evaluated from any one of the sub expressions are true. Example (x>15) || (y<10) will be true any first expressions is true or second expressions is true. The entire expression will be true or either any expression is true.

EXAMPLE

```
if(number<0 || number>100)
```

(iii) Logical NOT

The logical !(NOT) operator takes single expression and evaluates to true(1) if the expression is false(zero) and evaluates to false(zero) if the expression is true.

EXAMPLE

```
!(x>=y)
```

The expression after the! Operator in this case is x>=y.The not expression evaluates to true only if the value of x is neither greater than nor equal to y that is only if x is less than y.

3.7 ASSIGNMENT OPERATORS

In C language assignment operator = (Equal Sign).In this operator evaluates the expression on the right side and assigns the resulting value to the variable of left. Assignment expressions are written in the form

SYNTAX

Identifier=Expression

The arithmetic operators are =, +=, -=, *=, /=, %=.

EXAMPLE

1. x+=y is equal to x=x+y
2. x-=y is equal to x=x-y
3. x*=y ix equal to x=x*y
4. x/y=y is equal to x=x/y

3.8 INCREMENT AND DECREMENT OPERATORS

C provides two operators for incrementing and decrementing variables. The increment operator ++ add 1 to its operand, while the decrement operator -- subtracts.

The increment (++) and decrement (--) operator may be used either as prefix operators (before the variable, as in ++x), or postfix operators (after the variable ++). In both cases, the effect is to be increment the value of x. But the expression ++x increments x before its value is used, while x++ increments x after its value has been used.

EXAMPLE

x=5 then

x=x+1, sets x to 5

But x=++x, sets x to 6

In both cases x becomes 6. The increment and decrement operators can only be applied to variables.

3.9 CONDITIONAL OPERATORS

Conditional operator (? :) is ternary operator to construct conditional expressions, replace string if-else construct.

SYNTAX

Expression1? Expression2:Expression3

The conditional (?) works as done similar to see ternary operator. If expression1 is evaluated first, if the expression1 is true, expression2 evaluated, if the expression is false, expression3 is evaluated.

EXAMPLE

a=20;

b=25;

x= (a>b)? a: b

In this example if check the condition if a>b that is 20>25 condition is false so x will be assigned of b.

3.10 BITWISE OPERATORS

Bitwise operator operates on each bit of data. These operators are used for testing the bits or shifting them either left or right. Generally the bitwise operators are not applicable in the cases of float and double variables.

The list of bitwise operators are given below,

Operator	Meaning
&	Bitwise AND
	Bitwise Or
^	Bitwise XOR
<<	Shift Left
>>	Shift Right
~	Bitwise Complement

EXAMPLE

Assume that a,b and c are integers

int a=10 b=5 and c

According to the my convenience, let us assume that the integer copies 16 bits(2 bytes).

The binary representation a: 0000 0000 0000 1010

The binary representation b: 0000 0000 0000 0101

(i)Bitwise AND

Consider the statement c=a+b

This statement can be executable with the help of bitwise AND operator. After this statement is executed, each bit of c will produce the result 1 only if the corresponding bits in the both the bits a and b are 1.Otherwise it will return to 0.The bitwise AND operator is a binary operator.

a: 0000 0000 0000 1010

b: 0000 0000 0000 0101

a+b:0000 0000 0000 0000

(ii)Bitwise OR

Consider the statement c=a|b

This statement can be executed with the help of bitwise OR operator. After this statement is executed, each bit of c will produce the result 1 whenever at least any one of the bit position as 1.otherwise it will return to 0.The bitwise OR operator is a binary operator.

$$\begin{array}{r}
 \text{a: } 0000 \ 0000 \ 0000 \ 1010 \\
 \text{b: } 0000 \ 0000 \ 0000 \ 0101 \\
 \hline
 \text{a+b: } 0000 \ 0000 \ 0000 \ 1111 \\
 \hline
 \end{array}$$

(iii) Bitwise XOR

Consider the statement $c = a \wedge b$

This statement can be executed of the bitwise XOR operator. After this statement is executed each bit of c will produce the result 1, whenever the corresponding bits in a and b differ. The bitwise XOR operator is also binary operator.

$$\begin{array}{r}
 \text{a: } 0000 \ 0000 \ 0000 \ 1010 \\
 \text{b: } 0000 \ 0000 \ 0000 \ 0101 \\
 \hline
 \text{a+b: } 0000 \ 0000 \ 0000 \ 1111 \\
 \hline
 \end{array}$$

(iv) Left shift operator: The left shift operator to use the symbol \ll is a binary operator.

EXAMPLE

$c = a \ll 5$

The value of c in the integer of a is shifted to the left by one bit position. The result will be assigned to c.

To apply left shift \ll for a,

a: 00000000 0000 0101

After executing the left shift operator one zero are inserted in to right.

(v) Right shift operator: The right shift operator to use the symbol \gg is a binary operator.

EXAMPLE

$c = a \gg 2$

The value of c in the integer of a is shifted to right by one bit position. Finally the result will be assigned to c.

To apply left shift >> for a,

a: 0000 0000 0000 0010

after right shift by 1 place, a>>5

a: 0000 0000 0000 0001

after executing right shift operator one zero are inserted into left.

(vi) Bitwise complement operator

The bitwise complement operator (~) is a unary operator. It gives the value complementing each bit to the operand.

Consider the statement c=~b

Bitwise complements (~) are, b: 0000 0000 0000 0101

after bitwise complement that is ~b c: 1111 1111 1111 1010

3.11 COMMA OPERATORS

The set of expressions separated by using the operator are called comma operator. If any expression are represented are evaluated from left to right

EXAMPLE 1

int i,j;
where,
i and j are declared by the statement are separated by comma operator

EXAMPLE 2

i=(j=2,j+3)

In above statement two expressions are separated by comma. If first expression j=2 is evaluated after the expression j+3 is evaluated. In the above statement value 5 is assumed the variable of i.

Flowchart of the binary operator

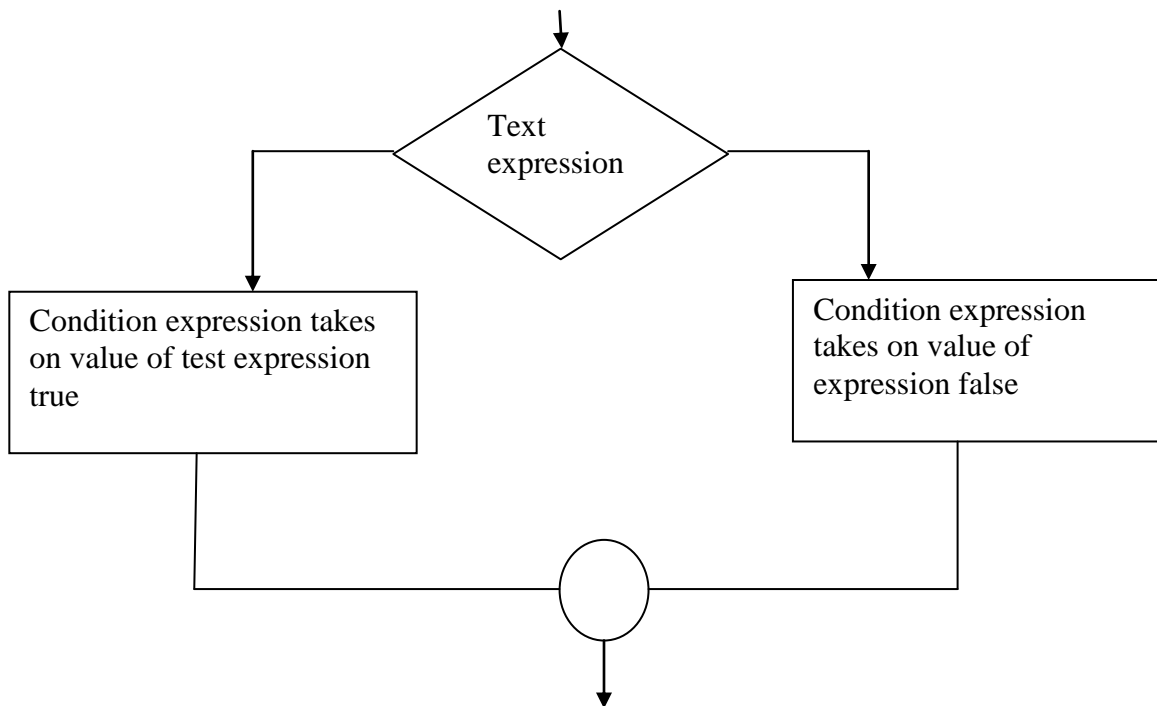


FIGURE 1.6

3.12 SPECIAL OPERATORS

(i)The size of () operator is denoted as special operators when used an operand it returns the number of bytes the operand occupies.

EXAMPLE

`x=sizeof(m1)`

`y=sizeof(float)`

`z=sizeof(#ol)`

(ii)The other special operator are(. and ->) which is used with structures and unions.

3.13 ARITHMETIC EXPRESSION

An Arithmetic expression represents a numeric value. Other types of expressions can represent character or Boolean values. These arithmetic expressions are made up of variable name, values, binary operators and brakets.An expression represents right hand side and results are stored in left hand side of value using assignment operator.

EXAMPLE 1

1. $a+b+c*d$
2. $a*(b-c/d)-e/g$

In another way to represent arithmetic expression is an expression using addition (+), subtraction (-), multiplication (*), division (/) and exponentials (**).

EXAMPLE 2

1. $1+3$ is 4
2. $23-0.45$ is 0.78
3. $3*8$ is 24
4. $-5**2$ is -25
5. $12/4$ is 3

3.14 EVALUATION OF EXPRESSION

In C language the expressions are evaluated using assignment operators. When expression is executed that is evaluated in right side and then the evaluated result is stored in left-side of variable name.

EXAMPLE

```
x=(a*b)/b
y=(a*b)+(b*a)
z=(m/n)/(x+a-b)
```

3.15 PRECEDENCE OF ARITHMETIC OPERATORS

C operators in order of precedence (highest to lowest) is listed given below,

(i) In arithmetic expression is represented without parentheses will be evaluated from left to right. Suppose in expression are represented with parentheses, first evaluated as parentheses of expression, so it is highest priority.

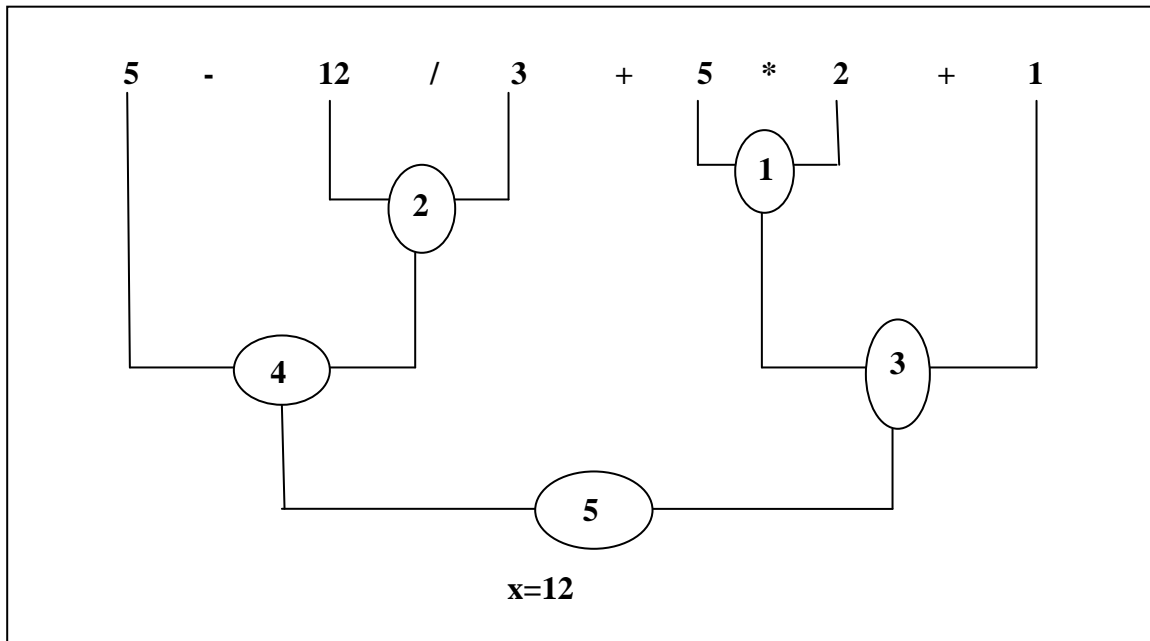
(ii) If expression is represented with two or more parentheses appear the expression is evaluated with left-most parentheses of expression after it evaluated with right-most.

(iii) The next highest priority are *, / and %.

(iv) The last priority that is lowest priority +, -. .

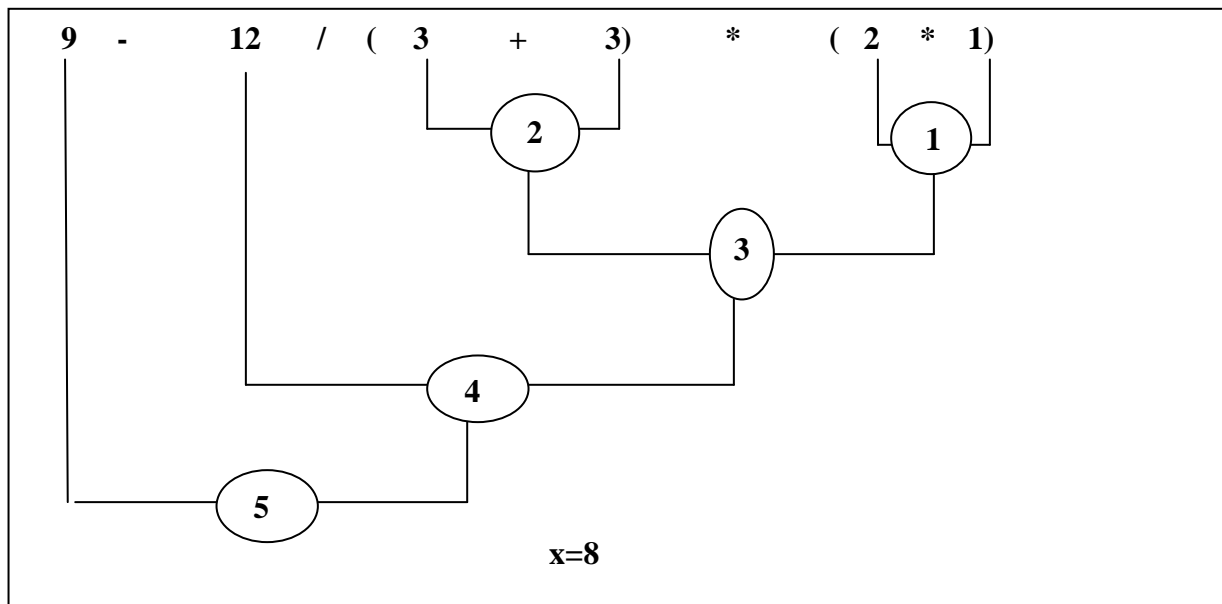
EXAMPLE 1

$$x = 5 - 12 / 3 + 5 * 2 + 1$$



EXAMPLE 2

$$x = 9 - 12 / (3 + 3) * (2 - 1)$$



3.16 TYPE CONVERSIONS IN EXPRESSIONS

In C support two types of conversions expressions they are,

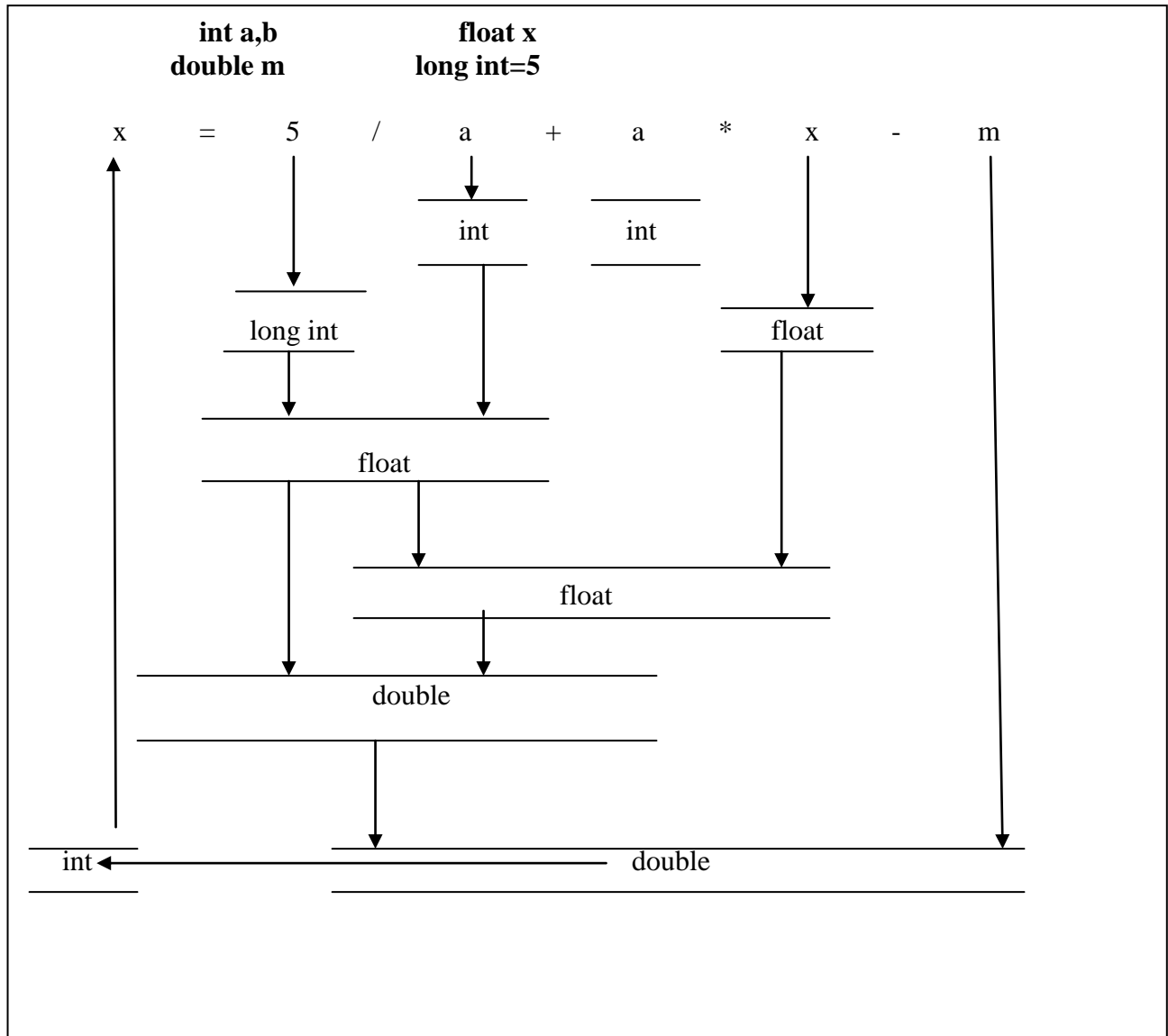
1. Implicit type conversion
2. Explicit type conversion

(i) Implicit Type Conversion

During evaluation automatic type conversion are known as implicit type conversion. In C, it supports automatically type conversion without any intermediate value that is can be evaluated without any losing of expression. Automatic conversion by the compiler.

EXAMPLE

Evaluation of Implicit Type Conversion



The following are the sequence rules for evaluating expressions:

1. If any one of the operands is long double and the other operand also long double finally the result will be long double.
2. If any of the operand is double and the operand also converted to double and final result will be double.
3. If one operand is float and another operand to float and the result will be float.
4. If one of the operands is unsigned long int and the converted operands also unsigned long int and the result will be unsigned long int.
5. If one operand is long int and other will be unsigned int the unsigned int the unsigned int is first converted into long int so final result will be long int.
6. Suppose we have considered both operands will be unsigned long int final result also unsigned long int.
7. If one operand will be long int so result will be long int.
8. If the operand is unsigned int and the other operand also unsigned int and the result will be unsigned int.

Hierarchy of Implicit Type Conversion

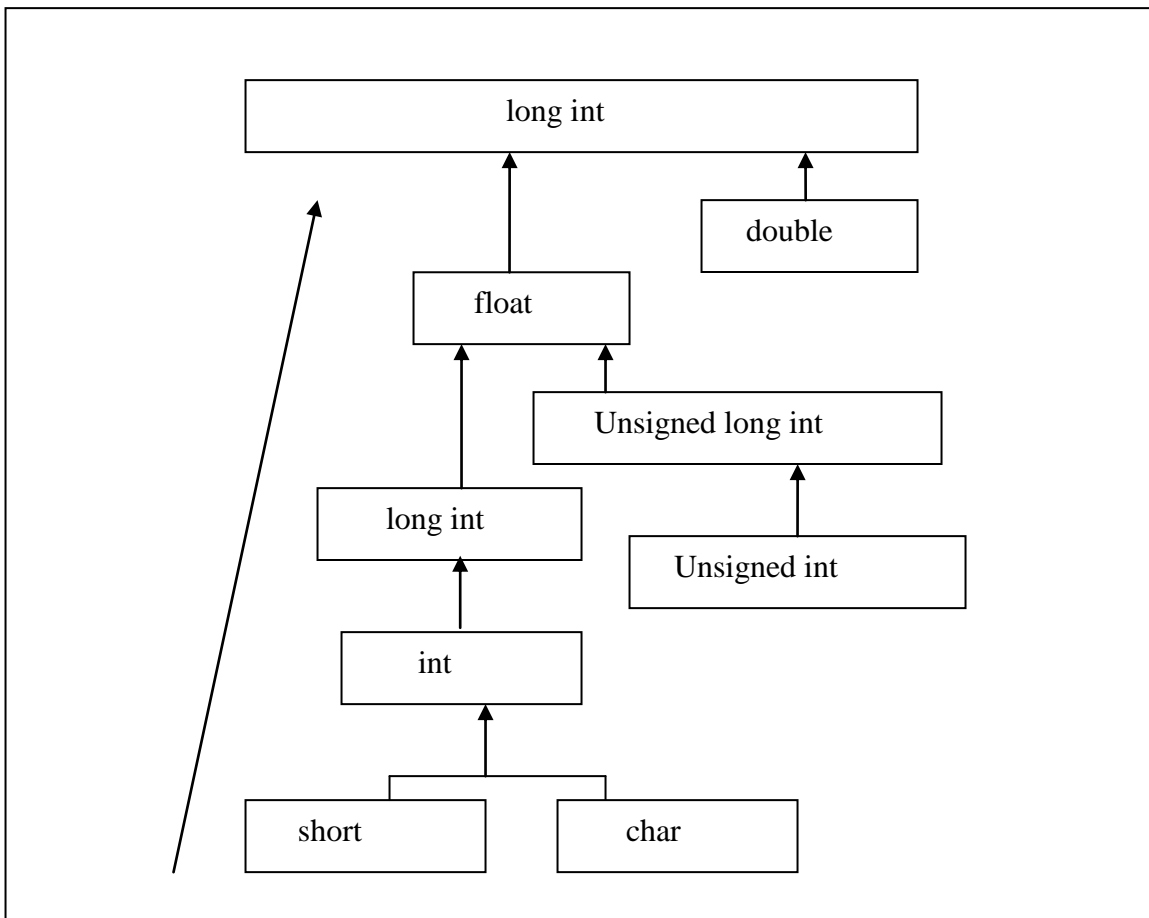


FIGURE 1.7

(i) Explicit Type Conversion

The Explicit type conversion results in explicitly defined with a program (instead of being done by a compiler for implicit type conversion).

SYNTAX

(type-name) expression

where,

type-name->is the data types

expression->it may be constant, variable or expression

EXAMPLE 1

```
x=(int)3.4
```

Here, the float value 3.4 is converted into integer.

EXAMPLE 2

```
x=float(21)
```

Here the integer value 21 is converted into float type.

EXAMPLE 3

```
x=(int)25.7/(int)4.5
```

here the value 25.7 is converted to integer and 4.5 is also converted into integer the expression is evaluated result is 25/4 and the result will be stored as 6.

3.17 OPERATOR PRECEDENCE AND ASSOCIATIVITY

In C contains many operators operator precedence is the order in which compiler group's operands with operators. The C compiler evaluates certain operators and their operands before others. If operands are not grouped using parentheses the compiler group them according to its own rules.

Suppose if an expression has more than one operator it is very important know the order in which they will be applied. There are different distinct levels of precedence and an operator may belong to any one of these levels. The operator are given the higher level of precedence are evaluated first. The operator of the same precedence is evaluated either from 'left to right' or 'right to left' depending on the level. This is known as the associatively property of an operator.

Summary of C operators

(i) Function Operator

Operator	Description	Associativity	Level
()	Function call	Left to Right	1
[]	Array Element Reference		

(ii) Unary Operator

Operator	Description	Associativity	Level
+	Unary plus	Left to Right	2
-	Unary minus		
++	Increment		
--	Decrement		
!	Logical Negation		
~	One's Complement		
Size of(type)	Size of type definition		

(iii) Arithmetic Operators

Operator	Description	Associativity	Level
*	Multiply	Left to Right	3
/	Divide		
%	Modulus		
+	add	Left to Right	4

(iv) Shift Operators

Operator	Description	Associativity	Level
<<	Left Shift	Left to Right	5
>>	Right Shift		

(v) Relational Operators

Operator	Description	Associativity	Level
<	Left than	Left to Right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		

(vi) Equality Operators

Operator	Description	Associativity	Level
==	Equal to	Left to Right	7
!=	Not equal to		

(vii) Bitwise Operators

Operator	Description	Associativity	Level
&	Bitwise AND	Left to Right	8
	Bitwise OR	Left to Right	9
^	Bitwise XOR	Left to Right	10

(viii) Logical Operators

Operator	Description	Associativity	Level
&&	Logical AND	Left to Right	11
^	Logical OR	Left to Right	12

(ix) **Conditional Operators**

Operator	Description	Associativity	Level
?:	Conditional Expression	Right to Left	13

(x) **Assignment Operators**

Operator	Description	Associativity	Level
=	Assign	Right to Left	14
+=	Add		
-=	Subtract		
*=	Multiply		
/=	Divide		
%=	Modules		

(i) **Comma Operator**

Operator	Description	Associativity	Level
,	Comma	Left to Right	15

EXAMPLE

Consider the variable x,y and z are integer variables having the values 2,3 and 4 respectively.

$$x*=-2*(y+z)/3$$

The given expression,

$$x=x*(-2*(y+z)/3)$$

Step 1

$$y+z=3+4=7$$

Step 2

$$-2*(y+z)=-2*7=-14$$

Step 3

$$-2*(y+z)/3=-14/3=-4$$

Step 4

$$2*-4=-8$$

$$x=8$$

Check your Progress

1. What is operator?

.....
.....
.....

2. What is difference between increment and decrement operator?

.....
.....
.....

3.19 LET US SUM UP

- ❖ C support different types of operators. In operator is a symbol that operates certain Mathematic or logical operations.
- ❖ Arithmetic operators can perform arithmetic operations.
- ❖ Unary operators are another type of operators that act upon a single operand to a value.
- ❖ The relational operators are used to compare arithmetic, logical and character expression.
- ❖ The logical operator allows a programmer to combine into single relational expression.
- ❖ In C language assignment operator = (Equal sign).In this operator evaluates the expression on the right side and assigns the resulting value to the variable to left.
- ❖ The reaming type of operator conditional operators, bitwise operators, comma operators and special operators.
- ❖ In C programming language support arithmetic expression and evaluation of expression.
- ❖ It support precedence of arithmetic operators and type conversion

- ❖ In C contains operator precedence. The operator of the same precedence is evaluated either from 'left to right' or 'right to left'.

3.20 LESSON END ACTIVITIES

1. What is output of the following program?

```
void main()
{
    int a=2,b=5,c=8,d;
    d=a+b*c;
    printf("%d",c)
}
```

2. Explain detail about different types of operator with example?
3. What is use logical || operator?
4. Why do you use conditional operator? Why is called ternary operator?
5. What is use of precedence and associativity of relational operator?

3.21 KEYWORDS

Arithmetic Operators: Arithmetic operator can perform arithmetic operations can be classified into unary and binary operator.

Relational Operators: The relational operators are used to compare arithmetic, logical and character expressions.

Conditional Operators (?:): Conditional operator is ternary operator to construct conditional expressions.

Arithmetic Expression: An arithmetic expression that represents a numeric value.

Type Conversion: C support two types of conversions expressions such as implicit type conversion and explicit type conversion.

3.22 QUESTIONS FOR DISCUSSION

1. What is arithmetic expression? Mention different types of arithmetic expressions.
2. What is type conversion? Explain its types
3. What is use of bitwise operators? Explain with example.
4. Explain detail about operator precedence and associativity with example.
5. Difference between prefixing and post fixing increment and decrement operator.

Check your Progress: Model Answers

Ans.1

1. C supports different type of operators. In operators is a symbol that operates certain mathematical or logical operations. These operators are being arithmetic operators, unary operators, relational operators, logical operators, increment and decrement operators, conditional operators, bitwise operators, comma operators, special operators.

Ans.2

2. C provides two operators for incrementing and decrementing variables. The increment operator ++ 1 to its operand while the decrement operator – subtracts 1.

3.23 SUGGESTED READING

1. The C Programming Language, B.W. Kernighan, Dennis M.Ritchie, PHI/Pearson Education
2. C Programming with problem solving, J.A. Jones & K. Harrow, Dreamtech Press
3. Programming in C - Stephen G. Kochan, III Edition, Pearson Education.
4. C – programming E.Balagurusamy Tata McGray Hill
5. How to solve it by Computer: Dromey, PHI
6. Schaums outline of Theory and Problems of programming with C : Gottfried

UNIT-4

UNIT

4

MANAGING INPUT AND OUTPUT OPERATIONS

CONTENTS

- 4.1 Aims and Objectives
- 4.2 Introduction
- 4.3 Reading a Character
- 4.4 Writing a Character
- 4.5 Formatted Data Input Function
- 4.6 Formatted Data Output Function
- 4.7 Let us Sum Up
- 4.8 Lesson End Activities
- 4.9 Keywords
- 4.10 Questions for Discussion
- 4.11 Suggested Readings

4.1 AIMS AND OBJECTIVES

At the end of this chapter you will learn how to:

- Describe the structure of reading a character with example
- Describe the structure of writing a character with example
- Understand the concept of data input function
- Understand the concept of data output function

4.2 INTRODUCTION

Input/Output functions are used to accept values into variables and printing them after the processing. The input and output of data can be done through the standard input/output statement or data. The function interact with the standard input (Eg: keyboard) and standard output (Eg: screen).

A header file is a file containing C declarations and macro definition to be shared between several source file. Each header file contains information in support of a group of related library functions. These files are included in the program by giving the #include statement at the beginning of the program. The header file required by the standard input/output library functions is called stdio.h.

In another header functions are,

```
#include<math.h>
```

Here the library functions used such as cos(x), sin(x), tan(x), sqrt(x) etc, these function used in the header file are math.h.

In C programming all input/output operations are carried out through function as printf and scanf. The scanf function which can read data from keyboard. The printf function which send the result to the monitor or console.

Some input/output functions returns the data items whereas others does not return .The function returning data item appear within expressions these expression output are stored in some of the variable name, for example $c=(x+b)/(x-a)$ or $x=getchar()$. On other hand functions not returning data items may be referred as putchar() statement.

4.3 READING A CHARACTER

By using input/output statement is reading a character from 'standard input' unit (Eg: keyboard) and writing it to the 'standard output' unit (Eg: Monitor). By using getchar () function reading a single character.

The getchar() function representing the following form,

<code>variable-name=getchar()</code>

where,

variable-name is a valid variable name that has been declared as char type. When this standard used to read the single character until anything key pressed and assign this character as a value to getchar() function.

In C program contains the statement are,

```
char  c;  
.  
.  
.  
c=getchar();
```

In C language the statement EOF denotes as End Of File, return on integer value an integer defined in the library file <stdio.h>. Typically EOF will assign value=-1 that is EOF=-1. The value vary from one compiler In C language the statement called EOF () is denoted as end of file is an integer defined in the library to another.

In pressing the keys CONTROL+Z causes <EOT> (End of Terminal) to be send to the program. In this statement CTRL+Z can be used to denote the statement called EOF.

EXAMPLE

```
#include<stdio.h>
void main()
{
    int ch;
    printf("Enter the character");
    c=getchar();
    if(c!=EOF)
        printf("you typed a character not reach end statement");
}
```

4.4 WRITING A CHARACTER

The C library function PUTCHAR() allows the user to display a character by character. The putchar() function is a complement of getchar () function. The PUTCHAR() function to print ASCII value of expression it receives an argument. It receives a single character to a standard output device that is the monitor.

SYNTAX

PUTCHAR(variable_name);

where,

variable-name is a name that contains a character. The characters are printed at the terminal called monitor.

EXAMPLE 1

```
char result;
result='c'
putchar(result);
```

will display the character 'c' on the screen

EXAMPLE 2

The program displays a character entered in the uppercase into lowercase.

```
#include<stdio.h>
void main()
{
    int c;
    printf("Enter an character with uppercase letter:");
    c=getchar();
    printf("Lowercase=");
    putchar(c+32);
}
```

Sample Program Output

Enter an character with uppercase letter: G
Lowercase=g

EXAMPLE 3

The following program given below,

1. Read a character
2. To check while character is not equal to EOF.
3. Display character

```
#include<stdio.h>
void main()
{
    int c;
    printf("Enter character until to press CTRL+Z");
    while(c!=EOF())
    {
        putchar(c);
    }
    printf("\n program terminated");
}
```

4.5 FORMATED DATA INPUT FUNCTION

In C Library function scanf(), is one of the input function. Just like the getchar() function. The scanf() function reads information from the terminal and stores with the particular variable. scanf() can be used to enter any combination of digits, characters and string.

SYNTAX

scanf("format string",arg 1,arg 2...arg n)

where,

format string->string containing certain required formatting information arg 1,arg 2,
arg n are the addresses the data items in the memory of the computer.

The format string comprises individual group of characters with one character group to begin with a percent sign (%), followed by a conversion character which indicates the type of the corresponding data item to be read from the keyboard. The multiple character groups in the format string can be contiguous or separated by white space characters. If white space characters are used to separate multiple characters groups in the format string, then all consecutive white space characters in the input data will be read but ignored.

Conversion character code	Meaning
%c	Data item read a single character
%d	Data item read a decimal number
%f	Data item read a floating point value
%h	Data item is a decimal, hexadecimal or octal integer
%i	Data item is a decimal, hexadecimal or octal integer
%o	Data item is an octal integer
%s	Data item is a string followed by space character(the null character '\0' will automatically added at the end)
%u	Data item is an hexadecimal number
%x	Data item is an hexadecimal number
[...]	Data item is a string which may indicate white space character

The arguments are written as variables or string, whose types match the corresponding character group in the control string. Each variable name must be preceded by an ampersand (&) sign. The '&' operator give the address of the corresponding variable. The string name should not begin with the ampersand.

EXAMPLE

```
#include<stdio.h>
void main()
{
    char name[20];
    int rollno,marks;
    printf("Enter the data for name,roll_no and marks:\n");
    scanf("%s %d %d",&name,&roll_no,&marks); }
```

OUTPUT

Enter the data for name, roll_no and marks:

Rahul 1001 77

In the given example of scanf statement

```
scanf("%s %d %d",&name,&roll_no,&marks);
```

It can be divided into two manners

1. Character groups -> %s %d %d
2. Address of the variables-> &name,&roll_no,&marks

In the scanf function the format string is %s %d %d. It can have three character groups.

%s->indicates that the first argument item which represents a string.

%d->indicates that the second argument item which represents an integer.

%d->indicates that the third argument item which represents also an integer.

scanf() cannot be used to enter a string that includes white space characters because it considers the character as the string terminator. Therefore, that part of the string until the first white space is uncounted it will be assigned to the array.

EXAMPLE

```
#include<stdio.h>
void main()
{
    char name[20];
    printf("Enter your name:");
    scanf("%s",name);
    printf("your name is %s",name);
}
```

Sample program output

Your name is Siva

The input strings using the scanf functions-type conversion character within the control string replaced by a sequence of characters enclosed in []. The white space character may be included within these brackets.

EXAMPLE

```
#include<stdio.h>
void main()
{
    char line[80];
    printf("Enter the phrase:");
    scanf("%d[ABCDEFGHIJKLMNOPQRSTUVWXYZ]",line);
    printf("\n The phrase of string has",line);
}
```

Sample Program Output

Enter the phrase: I LOVE INDIA
The phrase of string has I LOVE INDIA

In C programming c support the number of characters in a data item can be limited by specifying a maximum field width for that data item. An unsigned integer indicating the field width is placed within the format string, between the % sign and the conversion character. The data item may consist of fewer characters than specified. However; it cannot exceed the specified field width.

EXAMPLE

```
#include<stdio.h>
void main()
{
    int a,b,c;
    scanf("%3d %3d %3d",&a,&b,&c);
}
```

Given the above program if the data would have been entered as:21 14 17 assignment would be a=11,b=14,c=31.If data entered has been 12345678,then the assignment would be a=123 b=456 and c=789.In further had the data entered as 123456789 then the assignment as a=123,b=4,c=567.

Most version of C allows certain conversion character within the control string to be preceded by a single letter prefix. For example, an used to indicate either signed or unsigned long integer argument or double precision argument. Similarly is used to indicate signed or unsigned short integer.

scanf() is a function that reads the data with specified format from a given string stream source.

```
#include<stdio.h>
void main()
{
    short ix,iy;
    long lx,ly;
    double dx,dy;
    scanf("%d %d %lf",&ix,&ix,&dx);
}
```

4.6 FORMATED DATA OUTPUT FUNCTION

The function can be used to show output any combination of numerical values, characters and string.

SYNTAX

printf("control string",arg1,arg2,arg3...argn)

where,

Control string->consists of three types of items.

1. Character that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item
3. Escape sequence character used such as \n,\t and \b.

arg 1,arg 2,arg 3...arg n are arguments with represent the individual data items to be displayed.

The simple format function has following.

EXAMPLE

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("%8.4f\n",123.1234567);
    printf("%3.8d\n",1000);
    printf("right-justified:%8d\n",100);
    printf("left-justified: %8d\n",100);
    printf("10.15 s\n",programming in c");
    getch();
}
```


Sample program output

```
123.1235
00001000
right-justified:    100
left-justified: 100
programming
```

The argument can be written as constants, single, variable or array names, complex expression and function references.

Given the list below represent a conversion characters to be used in conjunction with the printf() function.

Conversion character conjunction function

Conversion character	Meaning
c	Data item is displayed as a single character
d	Data item is displayed as a signed decimal integer
e	Data item is displayed as a floating-point value with an exponent
f	Data item is a floating point value without an exponent
i	Data item is displayed as single integer
o	Data item is displayed as a string
s	Data item is displayed as a string
u	Data item is displayed as an unsigned decimal integer
x	Data item is displayed as a hexadecimal integer, without leading zero.

The following program indicates several different data types can be displayed using the printf function.

EXAMPLE

```
#include<stdio.h>
void main()
{
    char item[20];
    int empno;
    char empname[20];
    float salary;
    scanf("%d %s %f",&empno,&empname,&salary);
    printf("Employee details are %d %s %f",empno,empname,salary);
}
```

within the printf function the control string is

“Employee details are: %d %s %f”.It contains three character groups.

The first character group “Employee details:” is a string constant which will be displayed on the screen as it. The second character %d indicates that the first argument group is a decimal. The third character group %s indicates that the second argument group is a string. The fourth character %f indicates that the third argument group is a float. The printf() function can also contain escape sequences like ‘\n’, ‘\t’, ‘\b’ etc.

In C language, it can supports the displayed number in hexadecimal format also.Hence,a,b,c,d,e,f in hexa decimal systems represent the number 10 through15 these letters can displayed in either uppercase or decrease using %X or %x format specifies respectively.

EXAMPLE

```
#include<stdio.h>
void main()
{
    unsigned int no;
    scanf(“%u”&a);
    printf(“The input integer(in decimal)is %u \n”,no);
    printf(“The input integer(in hexademimal)is %x\n”,no);
}
```

Sample Output

The input integer (in decimal) is 44

The input integer (in hexadecimal) is 4c.

Some of the most commonly used placeholders follow,

- %d->Scan an integer as a signed decimal number
- %i->Scan an integer as a signed number
- %f->Scan floating-point number notation
- %s->Scan a character string
- %c->Scan a character(char).No null character is added
- ‘(space)’->space scans for white space characters.

The format specifies of printf statement are,

- %i->int(Same as %d)
- %lf->double
- %c->char
- %s->string
- %x->hexadecimal

Check your Progress

1. Explain printf() function.

.....
.....
.....

2. Explain scanf() function.

.....
.....
.....

4.7 LET US SUM UP

- Input/output functions are used to accept values into variables and printing them after the processing
- By using getchar() function reading a single character representing the following manner variable-name=getchar().
- In putchar() function it is used to display a character by character. It receives a single character to a standard output devices is called monitor.
 - In C library function scanf() it is one of the input function. Just like the getchar() function. The scanf() function reads information from the terminal stores with the particular variable.
 - The formatted data output function can be used to show the output any combination of numerical values such as character and string.

4.8 KEYWORDS

Input/Output Operations: Accepting the required inputs from input devices and displaying produced results on output devices are referred to as input/output operations.

Getchar() and Putchar() Functions: getchar() and putchar() are the simplest I/O functions and they are used to perform character input and output respectively.

Scanf() and Printf() Functions: The scanf() is used to accept mixed or same types of data through the keyboard.printf() function is used to display data on the screen.

Formatting of outputs: Formatting of output is to displays the output in more readable and comprehensible manner.

4.9 QUESTIONS FOR DISCUSSION

1. Write the syntax of using getchar() and putchar() function.
2. What is the output of the following statements:

```
int i=0;  
printf(“%d %d \n”,i,i++);
```
3. Describe the different format specifies of scanf().
4. Write a program to accept length and breadth of a rectangle and find its area.
- 5.Explain in detail about formatted input and output functions.

Check your Progress: Model Answers

Ans.1

The scanf() is used to accept mixed or same types of data through the keyboard. The syntax of its usage is as follows:

```
scanf(“control string”,arg 1,arg 2,...arg n)
```

Here,arg 1,arg 2,...arg n are the address of the variables into which data are to be accepted.

Ans.2

The printf() is used to display data on the screen.The syntax of its usage is similar to that of scanf()

```
printf(“control string”,arg 1,arg 2,...arg n)
```

Here arg 1,arg 2,arg n are the variables,the values of which are to be displayed on the screen.

4.10 SUGGESTED READING

1. Mastering C by Venugopal, Prasad – TMH
2. Complete reference with C Tata McGraw Hill
3. ANSI C, by Dennis Ritchie
4. Programming in C, by Lipschutz, SCHAUM SERIES OUTLINES

MODULE-2

UNIT-1

UNIT

1

DECISION MAKING AND BRANCHING

CONTENTS

- 1.1 Aims and Objectives
- 1.2 Introduction
- 1.3 Decision Making with If Statement
- 1.4 Simple If Statement with Example
- 1.5 If...Else Statement with Example
- 1.6 Nesting of If...else Statement with Example
- 1.7 Else...if Ladder Statement with Example
- 1.8 Switch...Case Statement with Example
- 1.9 The Ternary Operator `?=` with Example
- 1.10 Goto Statement with Example
- 1.11 Let Us Sum Up
- 1.12 Keywords
- 1.13 Questions for Discussion
- 1.14 Suggested Readings

1.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to:

- Describe the structure of decision making with if statement
- Identify Switch Case statement
- Explain the Ternary `?:` Operator
- Describe the Structure of goto Statement

1.2 INTRODUCTION

In a C program, it is a set of statements which are normally executed sequentially in the Order in which they appear. There are several instances where we have to make decisions. Similarly, when writing a program code, one may come to a point when a decision has to be made by evaluating a given expression as a true or false. Such an expression involves the use of logical or relational operators. Depending on the outcome of the program, execution accordingly branches out.

In C language support the following decision-making capabilities, they are,

1. If statement
2. Switch statement
3. Conditional operator statement
4. Goto statement

These statements are popularly known as decision-making statements. Since these statements 'control' the flow of execution, they are also known as control statements.

1.3 DECISION MAKING WITH IF STATEMENT

The if statement allows the programmer to execute a statement or series of statements Conditionally. If the condition supplied to the statement is logically true, then the statement that follows it is executed. Else it simply transfers control the next statement.

The condition may be any valid C language expression including constants, variables, logical comparisons etc. The condition must be placed with parentheses.

Some examples of decision making using if statements are,

1. if (student average greater than 60)
 grade first class
2. if (age is more than 58)
 person is retired
3. if (employee salary greater than 250000)
 pay with tax

The if statement may be implemented of conditions to be tested. The control statements are four types:

1. Simple if statement
2. if...else statement
3. Nested if...else statement
4. else if ladder

1.4 SIMPLE IF STATEMENT

The syntax of a simple if statement is,

SYNTAX

```
if(condition)
{
    Statement-block;
}
Statement-x;
```

The 'statement-block' may be a single statement or a group of statements. If the condition is true, the statement-block is executed; otherwise statement-block will be jumped and execution will get to the statement-x. When the condition is true both the statement block and statement-x also executed in the sequence. The control flow of the "simple if statement" can be represented using flow chart as follows.

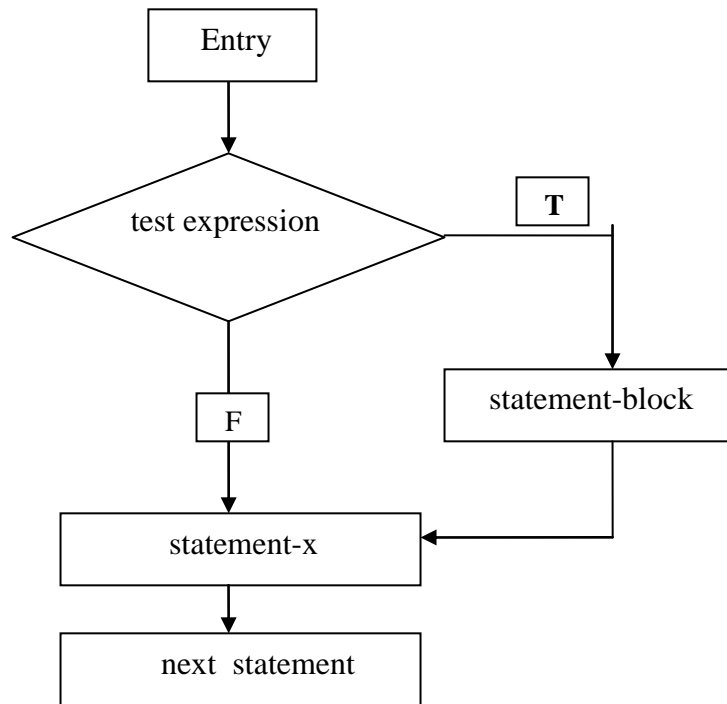


FIGURE 1.8

Consider the following statement of program that is written for calculating marks obtained in a university examination.

EXAMPLE 1

Write a program whether the given number is positive or not.

```

#include<stdio.h>
void main()
{
    float x;
    printf("\n Enter a positive number:");
    scanf("%f",&x);
    {
        print("\n Positive Number:%f",x);
    }
}

```

EXAMPLE 2

Write a program to enter integer number between 1...10.

```
#include<stdio.h>
void main()
{
    int number=0;
    printf("\n Enter an integer between 1 and 10:");
    scanf("%d",&number);
    if(number>7)
        printf("you enter %d which is greater than 7 \n",number);
    if(number<3)
        printf("you entered %d which is less than 3 \n",number);
}
```

1.5 IF...ELSE STATEMENT

When if...else statement taking a decision there are always two faces that is to do or not to do. Similarly, when programming there are two faces to a condition it may evaluate as TRUE or FALSE. To implement such a decision has provided you with the IF...ELSE construct. This construct carries out a logical test and then takes one of the two possible cases of actions, depending on the outcome of the condition.

SYNTAX

```
if(condition)
{
    Statement 1;
}
else
{
    Statement 2;
}
Statement-x;
```

If the condition within the parentheses evaluates to TRUE, then the statement immediately following it is executed; otherwise the statement following the ELSE clause is executed. Given below is flowchart description of the “if...else” construct.

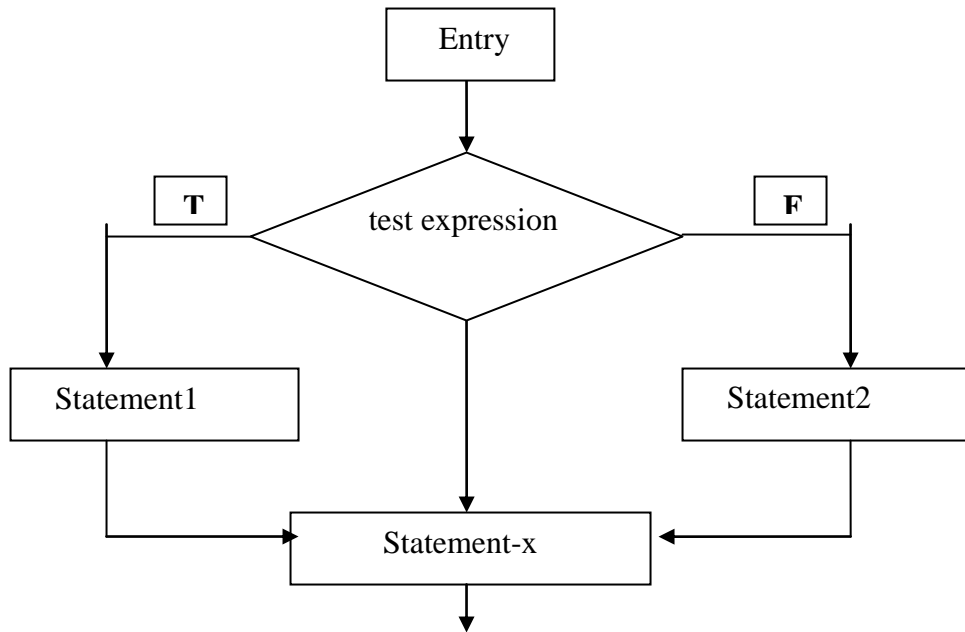


FIGURE 1.9

The program given below illustrate the use of the IF...ELSE construct

EXAMPLE 1

Write a program given below illustrate the use of the IF...ELSE construct.

```

#include<stdio.h>
void main()
{
    int flg;
    float side,area;
    printf("\n enter for 1 for circle and 0 for square:");
    scanf("%d",&flg);
    if(flg)
    {
        printf("\n Enter Radius:");
        scanf("%f",&rad);
        area=3.14*rad*rad;
        printf("\n Area of Circle=%f",area);
    }
    else
    {
        printf("\n Enter Side:");
        scanf("%f",&side);
        area=side*side;
    }
}
  
```

```

        printf("\n Area of the square=%f",area);
    }
}

```

EXAMPLE 2

Write a program to generate magic(random) number.

```

#include<stdio.h>
#include<stdlib.h>
void main()
{
    int magic;
    int guess;
    magic=rand();
    printf("Guess the magic number:");
    scanf(guess=magic)
    {
        printf("***** Right ***** ");
    }
    else
    {
        printf("*****Wrong*****");  }}

```

1.6 NESTING OF IF...ELSE STATEMENT

The else clause like the IF clause can contain compound statements. Moreover, a clause of the statement may contain another if statement. This feature is known as nesting of if statements. There are several forms that nested if-else statement can take.

```

if(Condition1)
{
    if(Condition2)
    {
        Statement 1;
    }
    else
    {
        Statement 2;
    }
    else
    {
        Statement 3;
    }
    Statement-x;
}

```

If the condition within the parentheses evaluates to TRUE, if the statement immediately following another condition is evaluated if the condition is true is evaluated with the clause else, if evaluated else part if the first part of the condition is else executed statement. otherwise executed statement-x.

Given below is a flowchart description of nested if construct

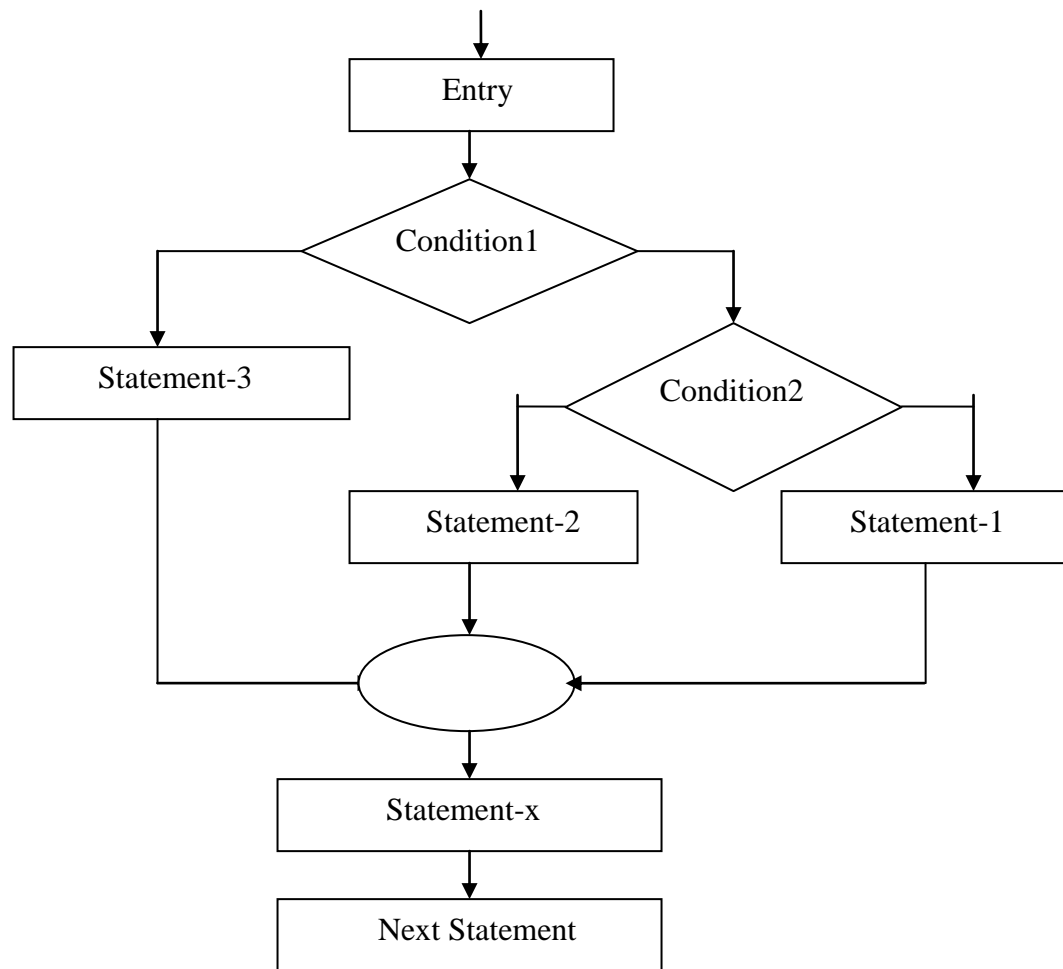


FIGURE 1.10

EXAMPLE 1

Write a program to check whether the integer number is positive or negative.

```
#include<stdio.h>
void main()
{
    int num;
    printf("Enter an integer:");
```

```

scanf("%d",&num);
if(num<0)
    printf("number is negative");
if(num>-1)
    printf("number is positive")
}

```

EXAMPLE 2

Write a program to find the largest of three numbers.

```

#include<stdio.h>
void main()
{
    float a,b,c,large;
    printf("Enter the value for a,b and c \n");
    scanf("%f %f %f",&a,&b,&c);
    large=a;
    if(b>big) large=b;
    if(c>big) large=c;
    printf("The largest of three number=%7.2 \n",large);
}

```

1.7 ELSE IF LADDER STATEMENT

They are another way of writing nested if statement using elseif ladder statement. A multipath decision is a chain associated with each else is an if.

SYNTAX

```

if(condition 1)
    statement 1;
else if(condition 2)
    statement 2;
else
    statement 3;
statement-x;

```

This type of statements is known as “else if ladder”. It checks the condition, if the condition is true the control is transferred to a corresponding n number of statement otherwise control transferred to the else part statement.

Given be is a flowchart description of the else if ladder construct.

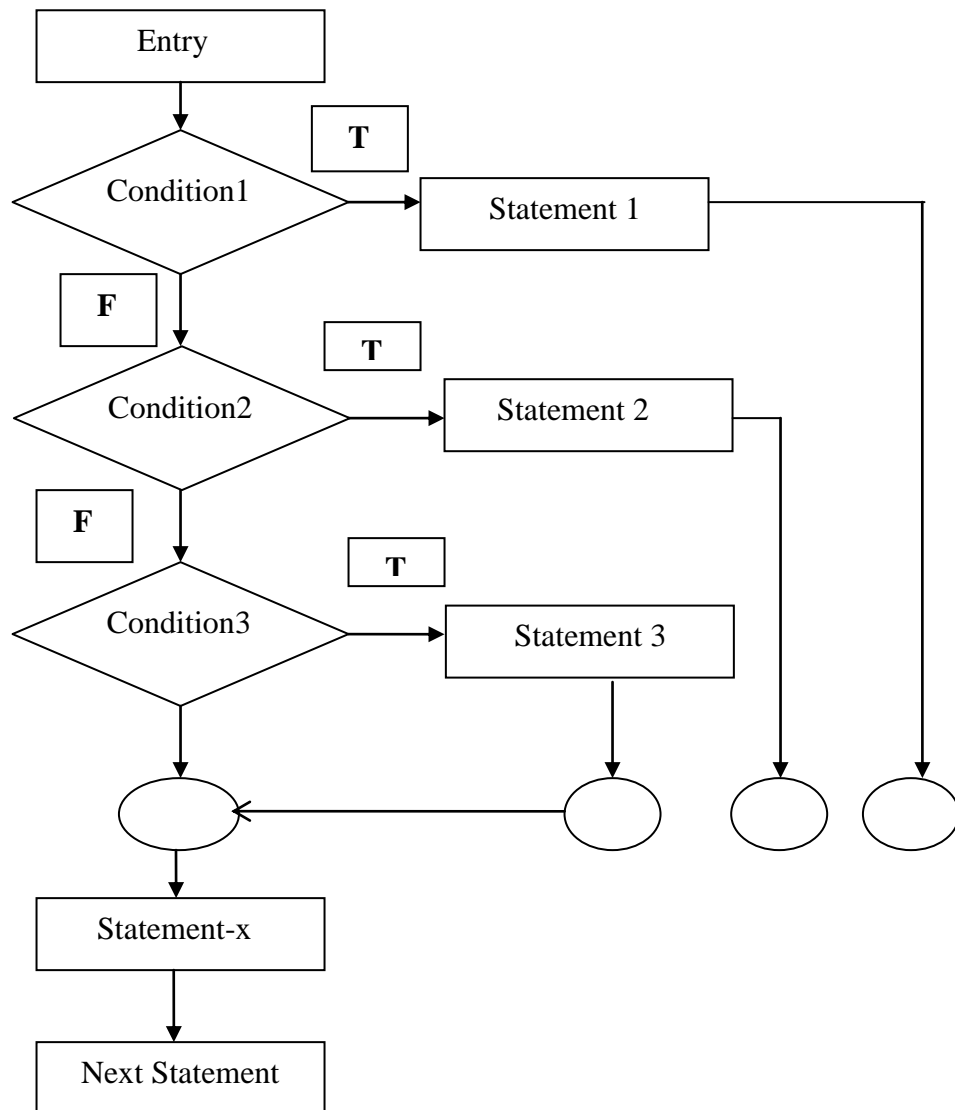


FIGURE 1.11

EXAMPLE 1

Write a program to check whether the given character is upper case or lower case.

```
#include<stdio.h>
void main()
{
    int char;
    printf("Enter Character");
    char=getchar();
    if(char!=EOF)
    {
```



```

    printf("End of file uncounted \n");
}
else if (char>'a' && char<='z')
{
    printf("lower case character \n");
}
else if(char>='A' && char<='Z')
{
    printf("upper case character \n");
}
else if(char>='0' && char<='9')
{
    printf("number \n");
}
else
{
    printf("Alpha Numeric");
}
}

```

EXAMPLE 2

Write a program to check find a grade of students in a university examinations.

```

#include<stdio.h>
void main()
{
    int marks;
    scanf("%d",&marks);
    if(marks>=75)
        printf("Honours \n");
    else if(marks>=60 && marks<75)
        printf("First Class");
    else if(marks>=50 && marks<60)
        printf("Second Class");
    else if(marks>=40 && marks<50)
        printf("third class");
    else
        printf("fail \n");
    printf("\n\n Grade of university Examinations");
}

```

1.8 SWITCH CASE STATEMENT

The Switch Case Statement causes a particular group of statements to be selected from a group of options. The selection is based upon the current value of the expression which is specified with the SWITCH statement.

SYNTAX

```
Switch(expression)
{
Case Label 1
    Statements;
    break;
Case Label 2
    Statements;
    break;
default:
    Statements;
    break;
}
```

The expression whose value is being compared may be valid expression, including the value of a variable as an arithmetic expression in a logical comparison etc, but not a floating-point expression.

The expression value is checked against each of the specified cases and when a match occur the statements following that case is executed. When a break statement is encountered, control proceeds to the end of the switch case statement.

The break statement should be included in each case. If the break statement is omitted then the statement for subsequent cases will also be executed. Even though a match has already take place.

The values that follow the keyword 'case' can only be labels; they cannot be expressions. Case labels cannot be repeated within a switch statement. The last case default is selected and the statement following this case is executed if none of cases mentioned earlier are matched. The default case anywhere may or may not be included depending on the program's needs. The default group may appear anywhere within the switch statement.

Given below is a flow chart description of a Switch Case Statement

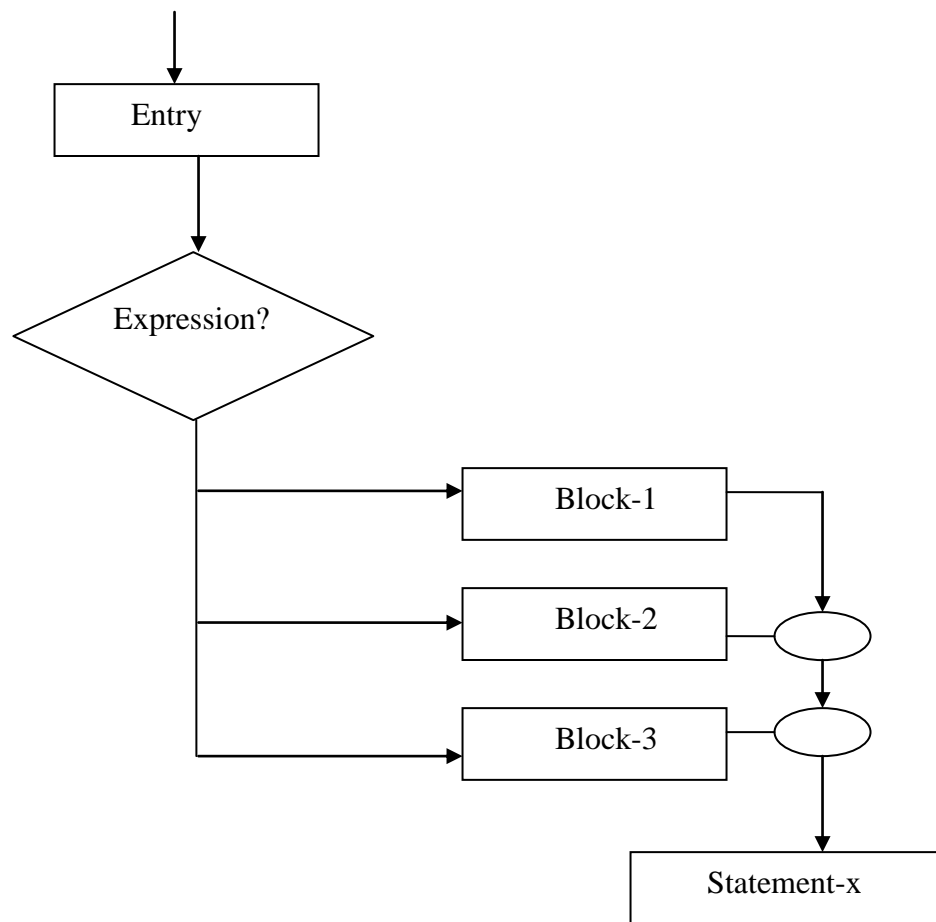


FIGURE 1.12

Write a program to find a grade of students in a university examinations using switch case statement.

```
#include<stdio.h>
void main()
{
    int marks;
    string grade;
    case 10:
    case 9:
    case 8:
        grade="Honours"
        break;
    case 7:
    case 6:
        grade="First Class";
        break;
```

```

    case 5:
        grade="Second Class"
    case 4:
        grade="Thrid Class";
        break;
    case else
        grade="fail";
        break;
}

```

1.9 THE TERNARY ?: OPERATOR

An expression that evaluates conditions is called as conditional expressions. The operator used to evaluate conditional expressions is called as a conditional operator (?:). It is a simple form of an if-else statement. The conditional operator also referred to as a ternary operator.

SYNTAX

Expression 1? Expression 2:Expression 3;

The expression1 is the condition for evaluation when evaluating a conditional operator, expression1 is evaluated first. If expression is true, expression 2 is alone evaluated. If expression is false, expression 3 is alone evaluated.

result=number<5?10:6;

EXAMPLE 1

Write a program to find the largest of three numbers using a conditional operator.

```

#include<stdio.h>
void main()
{
    int number1,number2,result;
    printf("Enter the value of number1 and number2");
    scanf("%d %d",&number1,&number2);
    result=number1>number2?:number1:number2;
    printf("The largest number is %d",result);
}

```

1.10 THE GOTO STATEMENT

The goto statement is used to alter the normal sequence of program execution by unconditionally transferring control to some other part of the program

SYNTAX

```
goto Label;  
.  
.  
.  
Label Statement;
```

where,

Label1 is an identifier used to transfer the control whether label occurs. The Label must followed a colon (:).

EXAMPLE 1

Write a program using goto statement

```
#include<stdio.h>  
void main()  
{  
    int x,y;  
    x=16;  
    y=12;  
    if(x==y)  
        x++;  
    else  
        goto error;  
    error:  
        printf("Fatal Error, exiting \n");  
}
```

EXAMPLE 1

Write a program to find square of given number.

```
#include<stdio.h>  
void main()  
{  
    double x,y;  
    read:
```

```

scanf("%f",&x);
if(x<0) goto read;
y=sqrt(x);
}

```

Check your Progress

1. What is mean by sequential execution?

.....

2. Define simple statement and compound statement.

.....

1.10 LET US SUM UP

- In C programs are is a set of statements which normally executed sequentially in the order there are several instance where we have to make decision by changing execution order
- The If statement allows the programmer to execute statement or series of statement sequentially.
- The simple if statement the statement-block may be a single statement or group of statement.
- When if...else statement taking a decision there are always two faces is to do or not do to.
- Nesting of if...else statement the else clause like the “if clause” can contain compound statement.
- The else...if ladder statement there are another way of writing result if statement using else if cases.
- The switch case statement causes a particular group of statements to be selected from a group of option.

- The ternary operator also referred to as a conditional operator, that evaluates conditional is called conditional expression.
- The goto statement is used to alter the normal sequence of program.

1.11 LESSION END ACTIVITIES

1. Define control statements with an example.
2. Explain the used of if statement with an example.
3. Explain the used of if-else statement with an example.
4. Multiple-if statements. Explain with necessary examples.
5. Nested-if statements. Explain with necessary examples.
6. Explain the syntax and use of the switch statement with an example.
7. Explain the various keywords used in a switch statement with an example.
8. Define fall through in a switch statement with an example.
9. Explain the used of conditional operator with an example.
10. State the used of goto and label statements in C.
11. Write a program to find the largest of three numbers using conditional operator.

1.12 KEYWORDS

Branching: One of several possible action will be carried out depending upon the outcome of the logical test is referred as branching.

If: C uses the keyword if to implement decision control statement.

If-Else Statement: The if-else statement is an extension of an ordinary if statement. The “if statement” by default will execute a single statement or a group of statements when the condition is false.

Multiple If-Else Statements: When a series of decisions are involved we have to use more than one “if-else statement” called as multiple ifs.

Nested If-Else Statement: One of more if or if-else statements inside an if or if-else statement(s) is called as nested if statements.

Switch-Case Statement: The switch statement allows us to make a decision from a number of choices. It is usually referred as the switch-case statement.

The Conditional Operator: An expression evaluates conditions is called as conditional expressions. The operator used to evaluate conditional expressions is called as a conditional operator.

Goto Statement: The goto statement is used to transfer the control in a loop or a function from one point to any other portion in that program.

1.13 QUESTIONS FOR DISCUSSION

1. Write a program to read in four numbers and print out the largest number of the four.
2. In an organization employee is paid as given below:
If his basic salary is less than Rs.1500 then HRA=15% of basic salary and DA=25% basic if his salary either equal to or above Rs.1500, then HRA=Rs.1500 and DA=1000 of basic. If the employee's salary is input through the keyword write a program to find the gross salary which is the sum of basic salary and HRA and DA.
3. Write the syntax of simple-if structure.
4. Explain nested if-else structure.
5. What is the need for else-if ladder? Give an example of its requirement.
6. Write the syntax of switch structure.
7. Compare else-if ladder and switch structure.
8. Why is the usage of goto statement?
9. Give the syntax the usage of goto statement.

Check your Progress: Model Answers

Ans.1

All the statements from the first statement till the last statement get executed without fall in a manner. That is, one after the other. This kind of execution of statements in a program is called sequential execution.

Ans.2

In a program given by single statement is called a simple statement and a program can contain a set of statements enclosed within a pair of opening and closing braces is called a compound statement.

1.14 SUGGESTED READING

1. The C Programming Language, B.W. Kernighan, Dennis M.Ritchie, PHI/Pearson Education
2. C Programming with problem solving, J.A. Jones & K. Harrow, Dreamtech Press
3. Programming in C - Stephen G. Kochan, III Edition, Pearson Education
4. C – programming E.Balagurusamy Tata McGray Hill
5. How to solve it by Computer: Dromey, PHI
6. Schaums outline of Theory and Problems of programming with C : Gottfried

UNIT-2

UNIT

2

DECISION MAKING AND LOOPING

CONTENTS

- 2.1 Aims and Objectives
- 2.2 Introduction
- 2.3 While Statement with Example
- 2.4 For...Loop Statement with Example
- 2.5 do...While Loop Statement with Example
- 2.6 Jumps in Loops
 - 2.6.1 Break Statement with Example
 - 2.6.2 Continue Statement with Example
 - 2.6.1 Difference between Break and Continue Statement
- 2.7 Nesting of For Loops with Example
- 2.8 Break Statement in Nested Loops
- 2.9 Continue Statement in Nested Loop with Example
- 2.10 Let Us Sum Up
- 2.11 Lesson End Activities
- 2.12 Keywords
- 2.13 Questions for Discussion
- 2.14 Suggested Readings

2.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to

- Describe the structure of determinate loop
- Describe the structure of indeterminate loop
- Explain the continue statement in nested loops
- Identify break statement
- Identify continue statement

2.2 INTRODUCTION

A loop is a process of the program to be executed repeatedly until condition is satisfied. When condition becomes false, the loop terminates and the control passes on the statement following the loop. A Loop consists of two segments, one is known as the control statement and the other is the body of the loop.

They are three kinds of looping statement in C,

1. The for...loop statement
2. The while statement
3. The do...while statement

2.3 WHILE STATEMENT WITH EXAMPLE

The while statements is used to carry out looping operation. Hence, the loop is executed until the expression evaluates be TRUE.

SYNTAX

```
while(expression)
{
    Statement;
}
```

The condition can be valid C language expression including the value of a variable, a unary or binary expression an arithmetic expression etc. In a “while construct” the condition is evaluated first. The statements given will be continuously executed until the value of the expression is not zero. This cycle continues until the condition evaluates to zero.

The “while statements” may be a simple statement or a compound statements. The group of statements usually contain one such statement which determents the value of the expression and ultimately leads to termination of loop. Once the condition evaluates to zero (false), the control is transferred to the statement following this loop.

The flowchart of the while loop are given below:

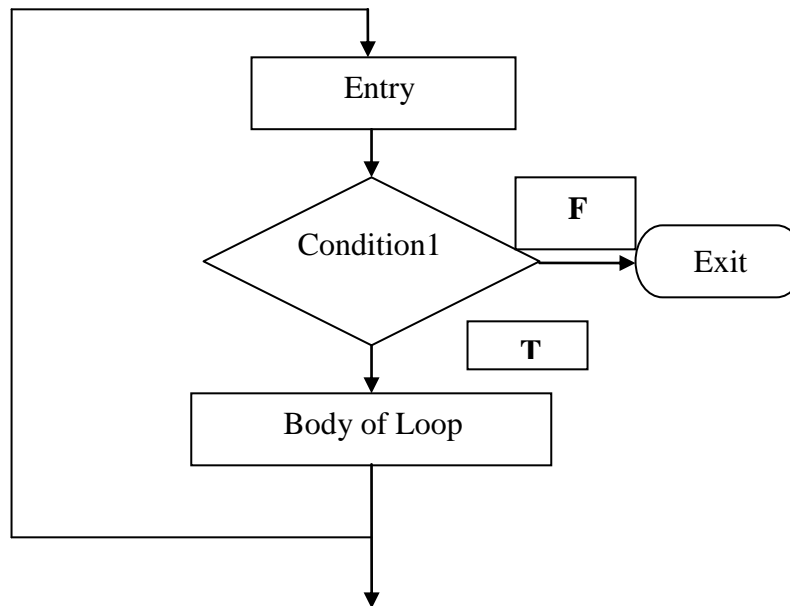


FIGURE 1.13

The condition in the while statement evaluates to false and the statement inside the loop are not executed. The condition in the while statement evaluates to true, and the statement inside the loop are executed.

EXAMPLE 1

Write a program to displays the integer number between 1 to 10 using while loop

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

EXAMPLE 2

Write a program to find the average of the marks.

```
#include<stdio.h>
void main()
{
    int i,num=0;
```

```

float sum=0,average;
printf("Enter the marks of student");
scanf("%d",&i);
{
    sum=sum+1;
    num++;
    scanf("%d",&i);
}
    average=sum/num;
    printf("The average is % 2f",average);
}

```

2.4 FOR...LOOP STATEMENT WITH EXAMPLE

The for...loop allows us to specify three things about the loop in a single line to construct a loop, initialization, test expression and incrementing or decrementing the value, each represented separated by semicolons and enclosed within a pair of parenthesis. The initialization, test-expression and increments or decrementing can be used in different places like the way used in while and do-while loop. The “for loop” allows us to specify three thing about in a single line.

SYNTAX

```

for(initialization;test-expression;increment/decrement)
{
    Statements;
}

```

Executions of statement in for...loop as follows;

1. Initialization statement is executed.
2. Test-Expression is evaluated.
3. If the test-expression evaluates true, the statements in the body of the loop executed.
4. Next, control goes to incremented or decremented value.
5. If it condition is true, again the statement of the body of the loop is executed.
6. Otherwise, it exits from the statements.

The given below flowchart of for loop,

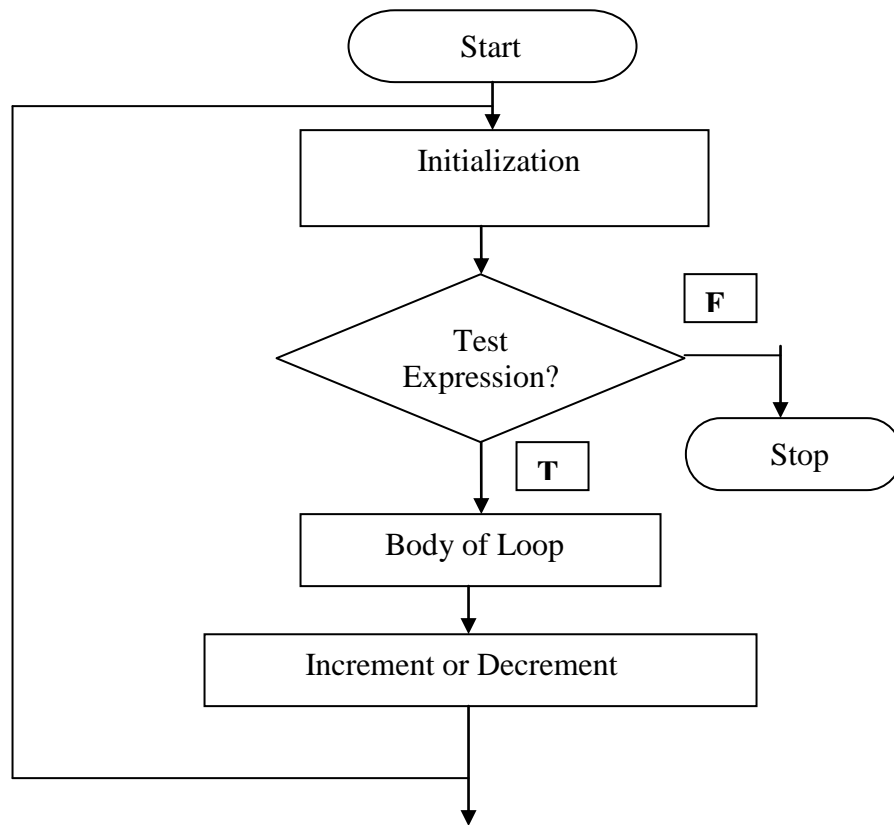


FIGURE 1.14

EXAMPLE 1

Write a program to find sum of first N natural numbers.

```
#include<stdio.h>
void main()
{
    int i,n,sum;
    printf("Enter a Number \n");
    scanf("%d",&n);
    sum=0;
    for(i=1;i<=n;i++)
    {
        sum=sum+i;
    }
    printf("The sum of first n number is %d",sum);
}
```

EXAMPLE 2

Write a program to generate multiplication of a number.

```
#include<stdio.h>
void main()
{
    int number,i,product;
    printf("Enter a number \n");
    scanf("%d",&number);
    for(i=1;i<=10;i++)
    {
        product=number*i;
        printf("%d * %d=%d \n",number,i,product);
    }
}
```

2.5 DO...WHILE STATEMENT WITH EXAMPLE

The do...while loop sometimes referred to as “do loop” differs from its counterpart the while loop in checking the condition. The condition of the loop is not tested until the body of the loop has been executed once. If the condition is false, after the first loop iteration the loop terminates. If the test expression evaluates to true then the body of the loop gets executed. This process repeated as long as the test-expression evaluates to true. Once the test-expression evaluates to false, the loop is exited and the control goes to the first statement following looping construct.

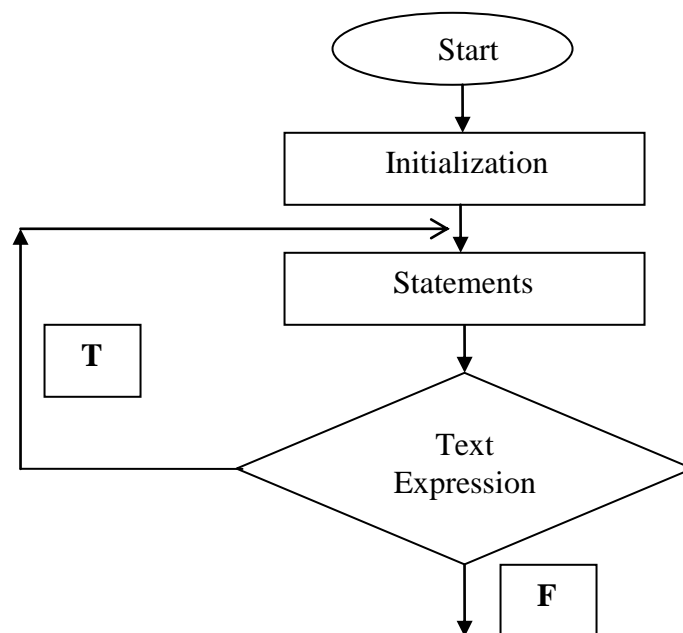


FIGURE 1.15

EXAMPLE 1

Write a program to find the sum of first natural numbers using for-loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,i,sum;
    clrscr();
    printf("Enter a number \n");
    scanf("%d",&n);
    i=1;
    sum=0;
    do
    {
        sum=sum+i;
        i++;
    }
    while(i<=n)
    printf("Sum=%d \n",sum);
}
```

EXAMPLE 2

Write a program to print numbers from 1 to 5.

```
#include<stdio.h>
void main()
{
    int x=5;
    int i=0;
    do
    {
        i++;
        printf("%d \n",i);
        while(i<=x);
    }
}
```

2.6 JUMPS IN LOOPS

Break and Continue statement are represented as jump in looping statement.

2.6.1 Break Statement

A break statement is used to terminate or to exit for, switch, while or do-while statement and the execution continues following the break statement.

SYNTAX

break;

The break statement does not have any embedded expressions or arguments. The break statement usually used at the end of each (in a switch statement) and before the start of the next case statement. The keyword “break” breaks the control only from the loop in which it is placed.

EXAMPLE

Write a program to loop exits only when ‘Q’ is pressed.

```
#include <stdio.h>
{
    int main()
    char c;
    for(;;)
    {
        printf( "\nPress any key, Q to quit: " );
        scanf("%c", &c);
        if (c == 'Q')
            break;
    }
}
```

2.6.2 Continue Statement

The continue statement is used to transfer the control the beginning of loop, there by terminating the current iteration of the loop and starting again form the next iteration of the same loop. The continue statement can be used within a while or a do-while or a for loop.

SYNTAX

continue;

The continue statement does not have any expression or arguments. Unlike break, the loop does not terminate when a continue statement is encountered, but it terminates the current iteration of the loop by skipping the remaining part of the loop and resumes the control to the start of the loop for the next iteration. The continue statement applies only to loops and not for switch statement.

EXAMPLE 1

Write programs to loop continue again when the value of i is 3.

```
#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=5;i++)
    {
        if(i==3)
            continue;
        printf("%d",i);
    }
}
```

EXAMPLE 2

Write a program for sum of positive number using continue statements.

```
#include<stdio.h>
void main()
{
    int n,i,sum=0,number;
    printf("Enter Number \n");
    scanf("%d",&n);
    printf("Enter %d number one by one \n,n);
    for(i=1;i<=n;i++)
    {
        scanf("%d",&number);
        if(number<0)
            continue;
        sum=sum+number;
    }
    printf("Sum of positive number:%d \n",sum);
}
```

2.6.3 Difference between break and continue statement

break statement	continue statement
1.can be used in switch statement	cannot be used in switch statement
2.causes premature exit of the loop	causes skipping of the statements following it in the body of the loop
3.The control is transferred to the statement of the loop	control is transferred back to the loop
4.The loop may not complete the intended number of iterations	The loop completes the intended number of iterations.

2.7 NESTING OF FOR LOOPS WITH EXAMPLE

The for loop can be nested within a while loop or another for loop are represented with another manner called nesting of loops. The enclosing loop is called outer-loop and the enclosed loop is called inner-loop.

EXAMPLE

To write a program to generate multiplication tables

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int m,n,i,j,p;
    clrscr();
    printf("enter lower limit \n");
    scanf("%d",&m);
    printf("enter upper limit \n");
    scanf("%d",&n);
    for(i=m;i<=n;i++)
    {
        for(j=1;j<=10;j++)
        {
            p=i*j;
            printf("%4d",p);
            {
                printf("\n");
            }
        }
        Getch();
    }
}
```

2.8 BREAK STATEMENT IN NESTED LOOPS

Already you know about break statement in single loop. Many programming used break statement with the multiple loops also. In this break statement it affects only the loop in which it is enclosed.

SYNTAX

```
for(initialization; test-expression 1; incre/decre)
{
for(initialization; text-expression 2; incre/decre)
{
if(test-expressions)
{
break;
}
statements;
}
}
```

```
for(initialization; test-expression 1; incre/decre)
{
for(initialization; text-expression 2; incre/decre)
{
statement;
}
if(test-expressions)
break;
}
```

2.9 CONTINUE STATEMENT IN NESTED LOOPS

You already see about the working principle of continue statement in single loop. Many programming used for continue statement when the loops are nested. It is important they continue statement affects only within the loop are enclosed.

SYNTAX

Continue statement enclosed in inner loop

```
for(initialization; test-expression 1; incre/decre)
{
    for(initialization; text-expression 2; incre/decre)
    {
        if(test-expressions)
        {
            Continue;
            Statements;
        }
        Statements;
    }
}
```

Continue statement enclosed in outer loop

```
for(initialization; test-expression 1; incre/decre)
{
    for(initialization; text-expression 2; incre/decre)
    {
        Statement;
    }
    if(test-expressions)
        Continue;
}
Statement;
```

Check your Progress

1. Explain for loop with an example.

.....
.....
.....

2. Explain while loop with an example.

.....
.....
.....

2.10 LET US SUM UP

- A loop is a process the program to be executed repeatedly until condition stratified.
- The while loop executed until the expression evaluates be TRUE.
- The for...loop allows us to specify three things about the loop in a single line.
- The do-while loop sometimes referred to as do...loop the condition of the loop is not tested until the body of the loop has been execute.
- A break statement is used to terminate or to exit a for, switch, while or do-while statement.
- The continue statement is used to transfer the control to the beginning of loop.
- The for...loop can be nested within a while loop or another loop are represented with another manner called nesting of loops.

2.11 LESSON END ACTIVITIES

1. Explain do-while with an example.
2. What is the need for looping?
3. Difference between selection and looping.
4. Difference between break and continue statement.
5. Mention the three necessary things required to construct a loop.
6. Explain for loop with its syntax.
7. Explain while loop with its syntax.
8. Difference between while loop and do-while loop.

2.12 KEYWORDS

Sequence: All the statements from the beginning till the end get executed without fail in the serial order is called sequence.

Selection: Selection was sometimes some statements are executed and some are skipped depending on some condition.

Looping: The repeat is one of a block of statements as long as some condition is true is called looping.

Iteration: Looping is also called iteration.

Valid Loop: To execute a block of statements repeatedly as long as some condition is true.i.e. to construct a valid loop.

Entry-controlled or pre-tested looping construct: The test-expression is first evaluated before entering into the body of the loop, the looping construct is called entry-controlled or pre-tested looping construct.

Break statement: A break statement is used to terminate or to exit a for, switch, while or do-while statement and the execution continues following the break statement.

Continue statement: The continue statement is used to transfer the control to the begging of the loop.

2.13 QUESTIONS FOR DISCUSSION

1. Write a program to print N natural numbers.
2. Write a program to generate Fibonacci Series up to N terms.
3. Write a program to generate first N prime numbers.
4. Write a program to generate Armstrong numbers up to 1000.
5. Write a program to reverse a number.
6. Write a program to print the following outputs using for loops.

a.1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

b. *

* *

* * *

* * * *

* * * * *

Check your Progress: Model Answers

Ans.1

The “for statement” is one of the most commonly used iteration constructs. The statement includes an expression that specifies an initial value for index, another expression that determines whether the loop is to be executed or not and the third which modifies the value of the index at the end of each pass. Like the while statement, testing of condition is done in the beginning of the loop.

Ans.2

The “while statement” or looping is defined as the repeated execution of a group of statements until the expression evaluates to TRUE.

SYNTAX:

```
while(expression)
    statement;
```

2.14 SUGGESTED READING

1. The C Programming Language, B.W. Kernighan, Dennis M. Ritchie, PHI/Pearson Education
2. 1. Turbo C/C++ - The Complete Reference - H. Schidt
3. Programming in C - S. Kochan
4. Born to code in C - H. Schidt
5. The Art of C - H. Schidt
6. C Programming - Kernighan and Ritchie - 2nd Ed.
7. Programming in ANSI C - Agarwal
8. Let us C - Kanitkar
9. Programming in ANSI C - Balguruswamy

UNIT-3

UNIT

3

ARRAYS

CONTENTS

- 3.1 Aims and Objectives
- 3.2 Introduction
- 3.3 One Dimensional Arrays
- 3.4 Declaration of One Dimensional Array
- 3.5 Accessing Array Elements
- 3.6 Initialization of One-Dimensional Arrays
- 3.7 Example Programs using One-Dimensional Arrays
- 3.8 Two Dimensional Arrays
- 3.9 Initializing Two Dimensional Arrays
- 3.10 Declaration of Two Dimensional Arrays
- 3.11 Example Programs Using Two Dimensional Arrays
- 3.12 Let Us Sum Up
- 3.13 Lesson End Activities
- 3.14 Keywords
- 3.15 Questions for Discussion
- 3.16 Suggested Readings

3.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to:

- Define and Describe One Dimensional Arrays
- Example Programs using One Dimensional Arrays
- Define and Describe Two Dimensional Arrays
- Example Programs using Two Dimensional Arrays

3.2 INTRODUCTION

An array is a group of elements (data items) that have common characteristics (Ex: Numeric data, Character data etc.) and share a common name. The elements of an array are differentiated from one another by their positions within an array.

Each array element (i.e., each individual data item) is referred as specifying the array name followed by its subscript enclosed in square brackets. The subscript indicates the position of the particular element with respect to the rest of the element. The subscript must be a non negative number.

For example, in the n element array, x, the array elements are x [1], x [2], x [3], x [4]....x[n] and 1,2,3...n are the subscripts x[i] refers to the ith element in a list of n elements.

Depending on the number of subscripts used, arrays are classified into one-dimensional arrays, two-dimensional arrays and so on.

Definition of Array

An array is defined to be a group of logically related data items of similar type stored in contiguous memory location is called array.

3.3 ONE DIMENSIONAL ARRAYS

Arrays whose elements are specified by a single subscript are called one-dimensional or single dimensional array. It is used to stored a list of values, all of which share a common names and disguisable by subscripts values.

3.4 DECLARATION OF ONE DIMENSIONAL ARRAYS

A single dimensional array is declared as follows:

SYNTAX

`type array-name[n];`

where,

array name->is the name of an array of n elements of the type specified. The size of an array must be an integer constant.

EXAMPLE

```
int i[100];  
char text[80];  
float n[12];
```

The integer array declaration `int x[100]` creates an array that is 100 elements long with the first element being 0 and the last being 99.

The subscript used to declare an array is sometimes called a dimension and the declaration for the array is often referred to as dimensioning. The dimension used to declare an array must always be a positive integer constant, or an expression that can be evaluated to be constant when the program is compiled. It is sometimes convenient to define an array size in terms of the symbolic constant. For example `i[20]=124`.

An individual element in an array can be referred to by means of the subscript, the number in brackets following the array name. A subscript is the number that specifies the elements position in an array. In the C language, subscript begins with zero. Thus the valid subscript value can from 0 to n-1, if n is the dimension of array. The subscript value used to access an array element could a binary expression etc. Thus, `i[2]` is not the second element of the array I but the third.

Properties of an array

1. The type of an array is the data type of its elements.
2. The location of an array is the location of its first element.
3. The length of an array is the number of data elements in an array.
4. The size of the array is the length of the array times the size of the elements, but the size of the array is usually referred as the product of subscript used.

3.5 ACCESSING ARRAY ELEMENTS

Once the array is declared, how individual elements in the array are accessed. This is done with the help of subscripts (i.e., number specified in the square brackets following the array name). The subscript specifies the elements position in the array. The array elements are numbered, starting from zero. The first item stored at the address pointed by the array name itself. This can also be referred as position 0. Thus the first element of the array age is referred as `age[0]`. The fourth element of the array is reoffered as `age[3]`.

3.6 INITIALIZATION OF ONE-DIMENSIONAL ARRAYS

An array can be initialized when declared by specifying the values of some or all of its elements. Array can be initialized at the time of declaration when their initial values are known in advance. The values to initialize an array must be constants never variables or function calls. The array can be initialized as follows:

```
int array[5]={4,6,5,7,2};  
float x[6]={0,0.25,-0.50,0,0};
```

when an integer array is declared as `int array[5]={4,6,5,7,2};` the compiler will reserve ten contiguous bytes in hold the five integer elements as shown in the diagram.

4	6	5	7	2
array[0]	array[1]	array[2]	array[3]	array[4]

The array size need not be specified explicitly. When initial values are included as a part of an array declaration. With a numerical array, the array size will automatically be set equal to the number initial values included within the declaration.

```
int digits[]={ 1,2,3,4,5,6}  
float x[]={0,0.25,0,-0.5}
```

Thus, `digits` will be a six-element integer array, and `x` will be a four-element floating-point array. The individual elements will be assigned the following values. The example given below,

```
digits[0]=1;digits[1]=2;digits[2]=3;  
digits[3]=4;digits[4]=5;digits[5]=6;
```

An array may also be initialized as follows:

```
int xyz[10]={78,23,67,56,87,76}
```

in the above array initialization, although the array size is 10, values where defined only for the first six elements. If fewer then all numbers have values specified, then the rest will have undefined values.

3.7 EXAMPLE PROGRAM USING ONE DIMENSIONAL ARRAY

EXAMPLE 1

Write a program to get five person ages to calculate average of person age.

```
#include<stdio.h>
void main()
{
    int age[5];
    int i,n;
    printf("Number of Persons:");
    scanf("%d",&n);
    if(n<=0||n>5)
        printf("Invalid number of persons entered \n");
    else
    {
        for(i=0;i<n;i++)
        {
            printf("Age:");
            scanf("%d",&age[i]);
            sum=sum+age[i];
        }
        printf("The average of age are: \n")
        for(i=0;i<n;i++)
            printf("%d \n",age[i]);
        printf("The average is %f,sum);
    }
}
```

EXAMPLE 2

Write a program to get 10 elements, to display any one of the element in the given list.

```
#include<stdio.h>
void main()
{
    int array[10]={ 10,20,30,40,50,60,70,80,90,100};
    printf("which element do you want display? \n");
    print("The element value is:%d \n",array[getchar()-'\']);
    printf("\n Do you wish to display another element! (y/n));
    if(getchar()='Y');
    {
        printf("\n Enter element number(1 to 10):");
        printf("The element value is:%d",array[getchar()-'\']);
    } }
```

3.8 TWO DIMENSIONAL ARRAYS

Arrays whose elements are specified by two subscripts are referred as two-dimensional arrays or double dimensional arrays. A two dimensional arrays of size m rows by n columns are declared as follows:

`type array-name[m][n];`

A two dimensional array two dimension of type int, 3 rows and 4 columns is declared as follows:

EXAMPLE

`int twodim[3][4];`

Arrays can be stored in the memory in two orders. Row major order and column major order. Multi-dimensional arrays are stored in memory using the row major order. Organization of the array named twodim with 3 rows and 4 columns in the row major order is shown here:

	Column 1	Column2	Column 3	Column 4
Row 1	[0][0]	[0][1]	[0][2]	[0][3]
Row 2	[1][0]	[1][1]	[1][2]	[1][3]
Row 3	[2][0]	[2][1]	[2][2]	[2][3]

Elements in a two dimensional array can be accessed by means of a row and a column. The row subscript generally is specified before the column subscript.

For example, twodim [1][3] will access the elements in two number 1(the second row) and in a column number (fourth) of the array. In array notation, twodim [1] represents a row and twodim [1][3] represents a column within that row. When initializing a two dimensional array at the time of array declaration, the elements will be assigned by rows, that are the elements of the first row will be assigned, then the elements of the second and so on.

Consider the following array declaration:

`int values[3][4]={ 1,2,3,4,5,6,7,8,9,10,11,12 }`

The result of this initial assignment is as follows:

values[0][0]=1	values[0][1]=2	values[0][2]=3	values[0][3]=4
values[1][0]=5	values[1][1]=6	values[1][2]=7	values[1][3]=8
values[2][0]=9	values[2][1]=10	values[2][2]=11	values[2][3]=12

The values can also be initialized by forming groups of initial values enclosed within braces. The value within an integer pair of braces will be assigned to the elements of a row.

For Example,

```
int values[3][4]={
    {1,2,3,4}
    {5,6,7,8}
    {9,10,11,12}
};
```

NOTE

1. If there are too few values within a pair of braces the remaining elements of that row will be assigned zero.
2. If the number of values in each inner pair of braces exceeds the defined row size, it will result a compilation error.

3.9 INITIALIZING TWO DIMENSIONAL ARRAYS

Like single-dimensional array, a double dimensional array can also be initialized. Arrays can be initialized by the following ways,

- Initializing an array during declaration
- Initializing an array using loops

3.9.1 Initializing an array during declaration

Similar to single dimensional arrays, a double dimensional array can also be initialized with one or more values during its declaration.

Storage-type data type array-name[row size][col size]={list of values};

Where the storage type, may be either auto or register or static or extern. The storage type while declaring an array is optional. The datatype specifies the type of elements that will be stored in the array. The array name is the name used to identify the array. The rules for naming array names are same as identifiers. Note that an array should not have the same name as an ordinary variable in the same block or a function.

For Example

```
int marks[4][2];
marks[4][2]={ 82,44},
             {92,65},
             {73,64},
             {65,100}.
```

The first subscript ranges from 0 to 3 and the second subscript ranges from 0 to 1. In the first case, the first inner pair of braces are assigned to the array elements in the first row, the value of second inner pair of braces are assigned to the array elements in the row and so on.

The array of mark can also be declared as,

```
int marks{82,44,92,65,73,64,65,100};
```

marks[0][0]=82	marks[0][1]=44
marks[1][0]=92	marks[1][1]=65
marks[2][0]=73	marks[2][1]=64
marks[3][1]=65	marks[3][2]=100

Examples for initializing an array using declaration

```
float amount[2][3]={ 70.50,90.74,50.75,75.10}
```

In the example, the size of the array is 6(3*2) and the values 1,2,3,4,5,6 are assigned to the variable amount[0][0],amount[0][1].....amount[2][1],amount[3][1] respectively. In the second example, the row size is missing. While initializing an array, the second dimension (i.e., colsize) is missing, but the first dimension (i.e., rowsize) is optional.

3.9.2 Initializing an array using loops

An array can be initialized by using for loop, a while loop or do-while loop. The given example illustrates the use of initializing an array using for loops.

Examples for initializing an array using a for loop,

```
int i,j,mark[5][5]
for(i=0;i<5;i++)
scanf("%d",&mark[i][j]);
```

In the example, the for loop initializes all the program elements of the array to zero. Similarly, array elements of the index to zero. Similarly, array can also be initialized while loop and do-while loops.

3.10 DECLARATION OF TWO-DIMENSIONAL ARRAYS

The syntax of declaring a two-dimensional array is as follows:

data-type variable-name[row-size][col-size];
--

where,

data-type->refers to any valid c data-type.

variable-name->(A valid C identifier),refers to the name of the array

row-size->indicates the number of rows and column size indicates the number of elements in each row

Row size and column size should be integer consists or convertible into integer values. Total number of locations allocated will be equal to row size* column size. Each element in a 2D array is identified by the array name followed by a pair of square brackets enclosing its row-number, followed by a pair of square brackets enclosing its column-number. Row number ranges from 0 to row-size-1 and column number ranges from 0 to column size-1.

EXAMPLE

```
int a[3][3];
```

Where, a declared to be an array of two dimensions and of data type int.Row size and column size of a 3 and e respectively. Memory gets allocated to accommodate the array are as follows. It is important to note that a is the column name shared by all the elements of the array.

column numbers			
	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]

Each data item in the array a is identifiable by specifying the array name a followed by a pair of square brackets enclosing row number, followed by a pair of square brackets enclosing row number followed by a pair of square brackets enclosing column number. Row-number ranges from 0 to 2 that are first row is identified by row-number 0, second row is identified by row-number 1 and so on.Similarly, column-number ranges from 0 to 2.First column is identified by column-number 0, second column is identified by column-number 1 and so on.

a[0][0] refers to data item in the first row and first column

a[0][1] refers to data item in the first row and second column

...

...

...

a[3][3] refers to data item in the fourth row and fourth column

a[3][4] refers to data item in the fourth row and fifth column

3.11 EXAMPLE PROGRAMS USING TWO-DIMENSIONAL ARRAYS

EXAMPLE 1

Write a program to accept and display a matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][4],i ,j;
    clrscr();
    printf("Enter the elements of matrix a of order 3*4 \n");
    for(i=0;i<3;i++)
    for(j=0;j<4;j++)
        scanf("%d",&a[i][j]);
    printf("Matrix a \n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
            printf("%4d",a[i][j]);
        printf("\n");
    }
    getch();
}
```

EXAMPLE 2

Write a program to calculate the sum of all elements in a matrix.

```
#include<stdio.h>
void main()
{
    int num[5][5],i,j,m,n,sum=0;
    printf("Enter the order of matrix:");
    scanf("%d %d",&m,&n);
    printf("Enter the elements of matrix:");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%d",&num[i][j]);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            sum=sum+num[i][j];
    printf("Sum of elements of the matrix is %d:",sum);
}
```

EXAMPLE 3

Write a to add two matrices

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3],b[3][3],c[3][3],I,j,m,n,p,q;
    printf("Input Row and Column of n Matrix \n");
    scanf("%d %d", &m,&n);
    printf("Input Row and Column of B Matrix \n");
    scanf("%d %d",&p,&q);
    if(n==p) && (m==q)
    {
        printf("Matrices can be added \n");
        printf("Input A-matrix \n");
        for(i=0;i<n;++i)
            for(j=0;j<m;++j)
                scanf("%d",&a[i][j]);
        printf("Input B-matrix \n");
        for(i=0;i<n++i)
            for(j=0;j<m;++j)
                scanf("%d",&b[i][j]);
        for(i=0;i<n;++i)
            for(j=0;j<m;++j)
                c[i][j]=a[i][j]+b[i][j];
        printf("Sum of A and B matrices:\n");
        for(i=0;i<n;++i)
        {
            for(j=0;j<m;++j)
                printf("%5d",c[i][j]);
            printf("\n");
        }
    }
    else
        printf("Matrices Cannot be a Added \n");
}
```

Check your progress

1. How to declare an array?

.....
.....
.....

2. How to accessing array elements?

.....
.....
.....

3.12 LET US SUM UP

- An array is a group of elements (data items) that have a common characteristics (Ex: numeric data, character data etc.,) and share a common name.
- Arrays whose elements are specified by a single subscript are called one dimensional or single dimensional array.
- The Subscript used to declare an array is sometimes called a dimension and declaration for it sometimes called the array referred to as dimensioning.
- An individual element in an array can be referred to by means of the subscript.
- Arrays whose elements are specified by two subscripts are referred as two-dimensional array or double-dimensional arrays.

3.13 LESSON END ACTIVITIES

1. Define an array, element and subscript.
2. Explain single dimensional arrays with examples.
3. Explain double dimensional arrays with examples.
4. State the rules to be followed for during array initialization.
5. Write a program to sort the numbers in ascending order.
6. Write the syntax of declaring a one-dimensional array?
7. Explain the need of array.

8. Find out errors, if any, in the following
- (a) `int at(9)(8);` (b) `int at(9);`
 - (c) `int a(3,4);` (d) `int [3,4];`
9. Explain the syntax of declaring a two-dimensional array with an example.
10. Write a program to generate Fibonacci series using arrays.

3.14 KEYWORDS

A Collective Manipulation: A collective manipulation over a group of values is one, which requires access to each value in the group or in its selected subgroup.

One-dimensional array or 1-D Array: An array with only one subscript is termed as one-dimensional array.

Multidimensional Array: An array with two subscripts is termed as two-dimensional array.

Matrix: A two-dimensional array enables us to store multiple rows of elements is called table of values or a matrix.

Three-dimensional Array: An array with three subscripts is termed as a three-dimensional array.

Elements: The individual values in the array are called as elements.

Subscript: Each array element is referred by specifying the array name, followed by a number. Within square braces referred as an **index or subscript**.

3.15 QUESTIONS FOR DISCUSSION

1. In C, what is the index of the first element in an array?
2. Can array indexes be negative?
3. Illustrate the initialization of a one dimensional array with an example.
4. Illustrate the initialization of a two dimensional array with an example.
5. Illustrate the declaration of a one dimensional array with an example.
6. Illustrate the declaration of a two dimensional array with an example.
7. What is array? Explain.
8. Write a program to add two matrixes.
9. Write a program to multiplication of two matrixes.
10. Write a program to find the inverse of a square matrix.

Check your Progress: Model Answers**Ans.1**

In C, that all variables in C must be declared before use. Hence we must first declare the array. It is done at the beginning of the function main () with the statement. Ex
int a[5]; this statement states that a is an array of 5 integers. Note the use of square braces after the array name (i.e a) enclosing the integer 5 that gives the number of elements in the array.

Ans.2

To access particular element in an array, specify the array name, followed by square braces enclosing an integer, which is called the array index. The array index indicates the particular element of the array which we want to access. The numbering of elements starts from zero. Each element of the array that is each integer occupies two bytes.

SUGGESTED READING

1. Mastering C by Venugopal, Prasad – TMH
2. Complete reference with C Tata McGraw Hill
3. C – programming E.Balagurusamy Tata McGray Hill
4. How to solve it by Computer : Dromey, PHI
5. Schaums outline of Theory and Problems of programming with C : Gottfried
6. The C programming language : Kerningham and Ritchie
7. Programming in ANSI C : Ramkumar Agarwal

MODULE-3

UNIT-1

UNIT

1

USER DEFINED FUNCTIONS

CONTENTS

- 1.1 Aims and Objectives
- 1.2 Introduction
- 1.3 Need for User-Defined Functions
- 1.4 A Multi-Function Program
- 1.5 Elements of User-Defined Functions
 - 1.5.1 Function Definition
 - 1.5.2 Function Call
 - 1.5.3 Function Declaration
- 1.6 Definition of Functions
- 1.7 Return Values and their Types
 - 1.7.1 The Return Statement versus exit
 - 1.7.2 Function Prototype
 - 1.7.3 Types of Function
- 1.8 Function Calls
- 1.9 Function Declaration
- 1.10 Let Us Sum Up
- 1.11 Lesson End Activities
- 1.12 Keywords
- 1.13 Questions for Discussion
- 1.14 Suggested Readings

1.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to:

- Discuss for user-defined functions
- Discuss about multi-function program
- Define and function and function declaration
- Discuss about elements of user-defined function
- Discuss return values and their types

1.2 INTRODUCTION

In the previous units, we used two types of functions such as `printf()` and `scanf()`. These functions have already been written, compiled and placed in libraries are called library functions.

In this unit, you study about user has chosen the function name, return data type and arguments (numbers and type), are called user-defined functions.

In C functions can be classified into two categories,

1. Library functions
2. User-defined functions

main() is the example of user-defined functions. **printf()** and **scanf()** are called the library functions. We already use other library functions such as **sqrt()**, **cos()**, **sin()**, **tan()**, **strcmp()**, **strcat()** etc. This is greatest features in C is that there is no conceptual difference between the user-defined functions and library functions. For example, we can write a series of function to process matrices (such as square of given number, finding cos value, finding sine value, etc). These are user-defined functions they are written by user. After use, they can be compiled into libraries and distributed. To the users of the library, these functions will act as library functions. That is these functions are predefined and precompiled; the users need not worry about the functions work internally.

1.3 NEED FOR USER-DEFINED FUNCTIONS

As discussed earlier, `main ()` is a specially recognized pre-defined function in C. In every program we have to use `main ()` function to indicate where the program has been executed. While in any program, does not utilizing the main function, it has face to the number of problems. In C, to write a program to become too large and complex and the result has been debugging, testing, and maintaining becomes difficult. In this situation, if a program it's large, program is divided into functional parts, then each part may be independently coded and

executed each part of program later combined into single unit. These subprograms called function, this way to manipulate, the easier to understand, debug and test.

In C, some time we need certain types of operation or calculations is repeated again and again in the program. In this situation, we use user-defined functions. This saves both the time and space.

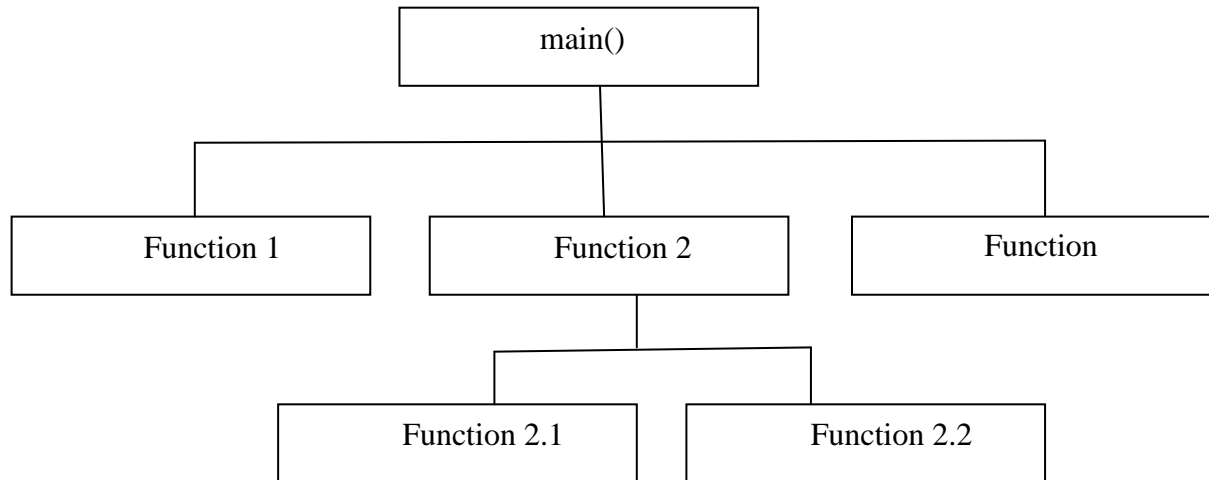


FIGURE 1.16

1.4 A MULTI-FUNCTION PROGRAM

A function is a self-contained block of code that performs a particular task. Once a function has been designed and packed, it can be treated as a ‘block box’ that takes some data from the main program. All that the program knows about a function is what goes in and what comes out. Every C program can be designed using a collection of block boxes known as functions.

Consider the set of statement given below,

```
#include<stdio.h>
void func();
void main()
{
    printf("Inside the Main Function");
    func();
    printf("\n Again Inside Main Function");
}
void func()
{
    printf("\n Inside func Function");
}
```

The above set of statements defined a function called func(), which calls the function main(), its exceeded and print the message is again inside main function.

This program will print the following output

Inside main function

Inside func function

Again inside main function

The above program contains two user-defined functions

(i) Main function

(ii) func () function

The program execution always begins with the main function. During execution of the main, the first statements are printed “inside the main function”. Hence the function is called func(). After it calls the user-defined function to print the message inside func function. After again control passes to the main() function, to print the message is called again the inside the main function.

Any function calls any other function. A ‘called function’ also an another function. A function can be called more than once. This is the main features of using function.

Except the starting point, there are no other predefined relationship, rules of precedence, or hierarchies among the function that make up a complete program. The functions can be placed in any order. A called function can be placed in any order. A called function can be placed either before or after the calling function

The following program shows the flow of control in a multi-function program

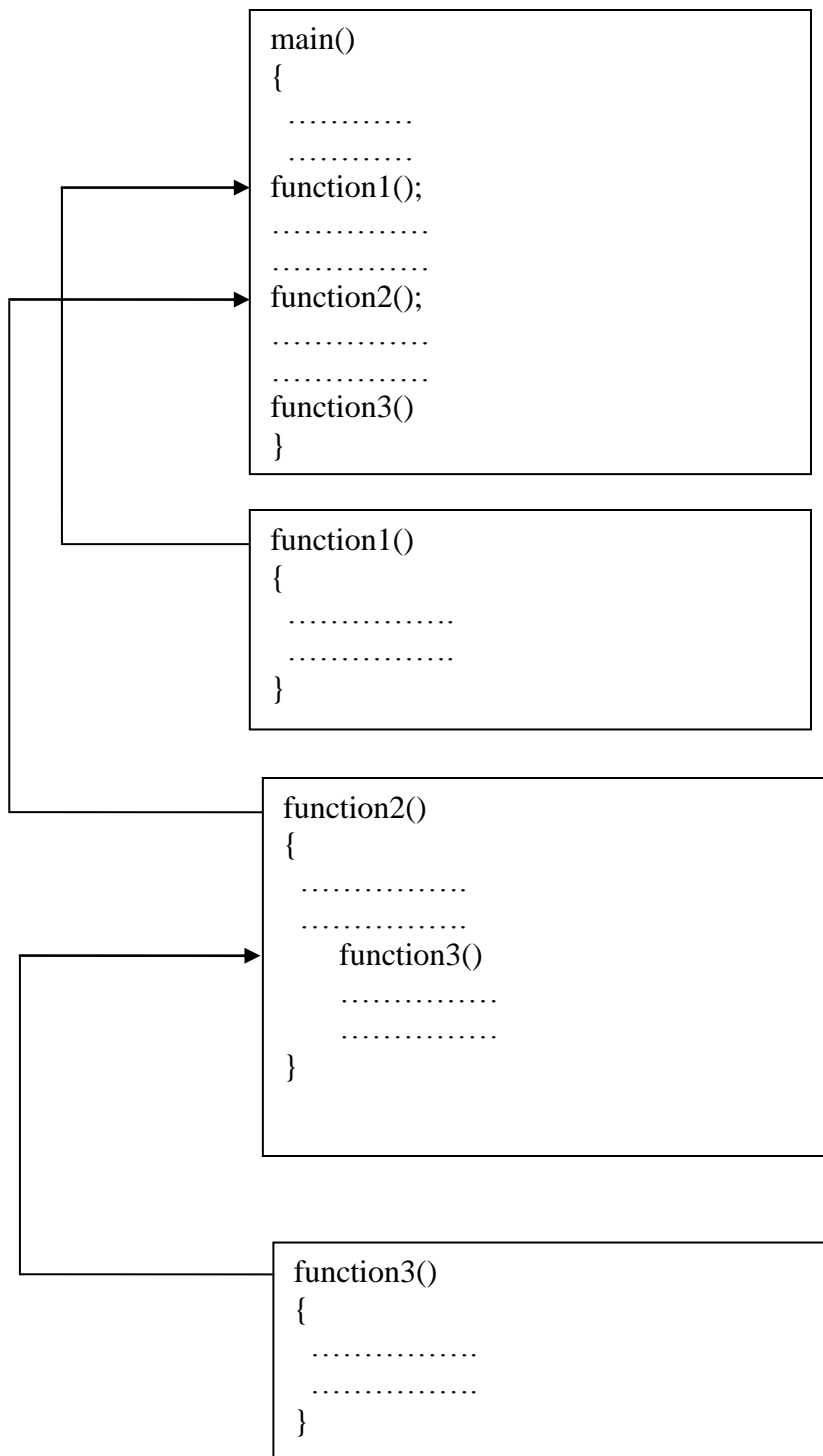


FIGURE 1.17

1.4.1 Modular Programming

Modular programming is a software design technique breaking a large program down into a number of functions, each of which performs a specific, well-defined task. This separate task is called **modules**. These modules are carefully integrated to become a software system that satisfies the system requirements. It is basically a “divide-and-conquer” approach to program solving.

Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Some Benefits of modular programming are:

1. Distributed Development
2. Modular Applications
3. Versioning
4. Secondary Versioning Implementation
5. Dependency Management
6. A modular programming Manifesto
7. Using Net Beans to do the modular programming

1.5 ELEMENTS OF USER-DEFINED FUNCTIONS

A **User-Defined Function**, or **UDF**, is a function provided by the user of a program. C functions can be classified into two categories, namely, library functions and user-defined functions. `main` is an example of user defined function. `printf` and `scanf` belongs to the category of library functions.

The use of functions in C has many advantages:

1. It facilitates top-down modular programming.
2. The length of a source program can be reduced by using functions at appropriate Places.
3. It is easy to locate and isolate a faulty function for further investigations.
4. A function may be used by many other programs..

They are three elements that are related to user defined functions are:

1. Function definition
2. Function call
3. Function declaration

The **function definition** is an independent program module that is specially written to implement the requirements of the function. In order to use this function we need to invoke it at a required place in the program. This is known as the **function call**. The program (or a function) that calls the function is referred to as the **calling program or calling function**. The calling program should declare any function (like declaration of a variable). This is known as function declaration or function prototype.

1.5.1 Function Definition

A function definition, also known as function implementation it's include the following elements.

1. Function name
2. Function type
3. List of parameters
4. Local variable declaration
5. Function statements
6. A return statement

All the six elements are grouped into two parts, namely,

- ❖ Function header
- ❖ Function body

SYNTAX

```
return-type function-name(parameters)
{
    Declarations;
    Statements;
    return value;
}
```

where,

return-type -> type of value returned by function or void if none

function-name-> unique name identifying function

parameters comma-> separated list of types and names of parameters

value-> value returned upon termination (not needed if *return-type* void)

The list of parameters is a declaration on the form

type1 par1, ..., typen par n

And represents external values needed by the function. The list of parameters can be empty.

All declarations and statements that are valid in the main program can be used in the

Function definition, where they make up the function body.

Function Header

The function header consists of three parts: The function type (also known as return type, the function name and the formal parameters list. Note that a semicolon is not used at the end of the function header.

Function Body

The function body contains the declarations and statements necessary for performing the required task. The body enclosed in braces, contains three parts, and is given below:

1. Local declarations that specify the variables needed by the function
2. Function statements that perform the task of the function
3. A return statement that returns the value evaluated by the function

Example, computing averages

```
double average(double x, double y)
{
    return (x+y)/2;
}
```

The return statement forces the function to return immediately, possibly before reaching the end of the function body.

1.5.2 Function Call

A function can be called by simply using the function name followed by a list of actual parameters (or arguments), if any, enclosed in parentheses.

SYNTAX

Variable=function-name (arguments);

Variable is assigned the return-value of the function.

The arguments are values with types corresponding to the function parameters.

EXAMPLE

```
int main(int argc, char **argv)
{
    double a=1;
    double avg;
    avg = average(a, 3.0);
    return 0;
}
```

Implicit type conversion is used when the types of the arguments and parameters do not agree.

When a function is called the value of the argument is copied to the corresponding parameter. Changes made to the parameter inside the function will not affect the argument (**call by value**).

EXAMPLE

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

gives the output 1 0 1.

1.5.3 Function Declaration

Like variables all function in a C program must be declared, before they are invoked. A function declaration (also known as function prototype) consists of four parts.

1. Function type (return type)
2. Function name
3. Parameter list
4. Terminating semicolon

SYNTAX

`return-type function-name(parameter types);`

The function body is replaced by a semi-colon.

Parameters need not be named, it is sufficient to specify their types.

Example declaring average

```
double average(double, double);
```

Declaration is not necessary if the function definition precedes the first call.

1.6 DEFINITION OF FUNCTIONS

A function is defined as a self-contained program which is written for the purpose of accomplishing some task.

A function definition consists of two parts. They are,

1. Argument declaration
2. Body of the function

The first part of the function specification consists of type specification of the value returned by the function followed by the function name, a set of arguments (may or may not be present) separated by commas and enclosed in parenthesis. If the function definition does not include any arguments, an empty pair of parenthesis must follow the function name.

SYNTAX

```
returntype functionname(argument list)
{
    declaration(s);
    statement(s);
    return(expression);
}
```

where

returntype->specifies the data type value to be returned by the function. The return type is assumed to be of type int by default if it is not specified.

function name-> used to identify the function. The rules for naming a function are Same as identifiers.

argument list->specified inside the parenthesis after the function name is optional.

Most functions have list of parameters and a return value that provides means for communication between functions. The arguments in the function reference, which defines the data items that are actually transferred, are referred as **actual arguments** or **actual parameters**. All variables declared in function definitions are local variables. Their scope is visible known only in the function in which they are defined. Functions arguments are also local variables.

EXAMPLE

Write a program for sum of two numbers and return the sum to the main function.

```
#include<stdio.h>
void main()
{
    int addnum(int,int);
    int sum,a,b;
    printf("\n Enter two number to be summed:");
    scanf("%d %d",&a,&b);
    sum=addnum(a,b);
    printf("\n The sum of %d and %d is %d",a,b,sum);
}
int addnum(num1,num2)
{
    int tot;
    tot=num1+num2;
    return(tot);}
}
```

Sample program output:

Enter two numbers to be summed: 15, 17
The of 15 and 17 is 32

The above program consists of two functions,

- The required **main** function
- The programmer defined function **addnum**, which sums the two values.

The function main reads in two integer values, and assigns them to a and b. Then, main calls addnum, transferring the integer values a and b receiving their sum. The sum is then displayed and the program terminates.

The integer values are transferred via the arguments num1 and num2, and their sum tot is returned to the calling portion of the program via the return statement.

1.7 RETURN VALUES AND THEIR TYPES

1.7.1 The return () Statement versus exit()

The **return** statement is used to return the control from the calling function to the next statement of the called portion of the program. The **return** statement also causes program logically to **return** to the point from where the function is accessed (called). The return statement returns one value per call.

The **return** statement can be any one of the types as shown below,

```
return;  
return();  
return(constant);  
return(variable);  
return(expression);  
return(conditional expression);  
return(function);
```

The first and second **return()** statements, does not return any value, and are just equal to the closing brace of the function. If the function reaches the end without using a return statement, the control is simply transferred back to the calling portion of the program without returning any value. The **return()** statement returns a value 1 to the calling function. The third **return()** statements returns a value 1 to the calling function.

The **exit()** is build-in function available as part of C library. The common thing shared by both the return statement and the **exit()** is that both are used to exit the function in which they are used.

EXAMPLE 1

```
if(fact<=1)
return(1);
```

This **return()** statement return a constant q when the condition specified inside the if statement evaluates to true.

EXAMPLE 2

```
return(num1+num2+num3);
```






This **return()** statement returns a value depending upon the result of the conditional expression specified inside the parenthesis.

EXAMPLE 3

```
return(power(5,2));
```

This **return()** statement calls the function specified inside the parenthesis and collects the result obtained from that function, and returns it to the calling function.

NOTE

-  The limitation of a return statement is that it can return only one value from the called function to the calling function.
-  The return statement can be present anywhere in the function, not necessarily at the end of the function.
-  Number of return statements used in a function is not restricted, since the return statement which executes first will return the value from the called function to the calling function and the other return statements are left unexpected.
-  If the called function does not return any value, then the keyword void must be used as the return specifier.
-  Parenthesis used around the expression in a return statement is optional.

1.7.2 Function Prototypes

A C function returns an integer value by default. Whenever a call is made to a function the compiler assumes that this function would return a value of type **int**. If you decide that a function should return a value other than **int**, then it is necessary to mention the first line of the function in the program, before it is used, which is called the **function prototype** also referred as the **function declaration**.

Function prototypes are usually written at the beginning of the program explicitly before all user-defined functions including the `main()` function.

SYNTAX

return type function name(dt1 arg1, dt2 arg2... dtn argn)

where,

return type->represents the data type of the value that is returned by the function

dt 1, dt 2... dt n->represents the data types of the arguments

arg1, arg2... argn->The data types of the arguments should be specified compulsorily in the
Function definition

EXAMPLE

```
int sum(int num)
```

where `sum` is the name of the function, `int` before the function name `sum()` indicates that the function returns a value of type `int`. The variable `num` inside the parenthesis is the parameter passed to the called function. The data type `int` before the parameter `num` indicates that it is type integer.

1.7.3 Types of function

A function depending upon the arguments present or not and whether a value is returned or not, may be classified as

- ❖ Function with no arguments and no return values
- ❖ Function with return values and no arguments
- ❖ Function with arguments and no return values
- ❖ Function with arguments and return values

1.8 FUNCTION CALLS

A function can be accessed (i.e., called) by specifying its name, followed by a list of arguments enclosed in parenthesis and separated by commas. If the function call doesn't require any argument an empty pair of parentheses must follow the function's name.

The arguments appearing in the function call are referred to as actual arguments. The actual arguments may be expressed as constants, single variables, or more complex expressions. The number of actual arguments must be the same as the number of formal arguments. Each actual argument must be the same data type as its corresponding formal argument.

EXAMPLE

To write a program to find maximum of given number using function

```
#include<stdio.h>
void main()
{
    int maximum(int,int);
    int a,b,c,d;
    printf("\n a=");
    scanf("%d",&a);
    printf("\n b=");
    scanf("%d",&b);
    printf("\n c=");
    scanf("%d",&c);
    d=maximum(a,b);
    printf("\n \n maximum=%d",maximum(c,d));
}
int maximum(x,y)
int x,y;
{
    int z;
    z=(x>=y)?x:y;
    return(z);
}
```

Sample program output

```
a=10
b=20
c=30
maximum=30
```

1.9 FUNCTION DECLARATION

Whenever a function return a value than an integer, the calling function must declare the return type of the function. Therefore, a function declaration must be present in the calling portion of a program if a function returns a non integer value and the function call precedes the function definition.

SYNTAX

data-type name arg type1, arg type2...arg type n

Where

data-type-> represents the data types of the value returned by the function

name-> represents the function name and arg type2...arg type n represents the data types of the first argument second argument and so on respectively.

NOTE

The data-types of arguments are optional in the function declaration

when the argument data-types are included in the function declaration the compiler will convert the value of each argument to the declared data-type (if necessary) and then compare each actual data-type with its corresponding formal argument. A compilation error will result if these data-types do not agree. Thus a function declaration will reveal the data-type inconsistencies detected during the compilation process.

so far the functions returning either no value (void) or an (int) have been considered.

EXAMPLE

To write a program to display power of given number using function

```
#include<stdio.h>
void main()
{
    double power(float,int);
    int exponent;
    float number;
    printf("\n Please enter a number");
    scanf("%f",&number);
    printf("\n please enter a non-negative number");
    scanf("%d",&exponent);
    printf("\n \n %f raised to the power %d is %f",number,exponent,power(number,exponent));
```

```

}
double power(float base,int p)
{
    double result;
    for(result=1.0;p>0;--p)
        result=result*base;
    return(result);
}

```

Sample program output

Please enter a non-negative number:2
Please enter a non-negative number:3
2 raised to the power 3 is 8.000000

As mentioned earlier the use of keyword void for the return data-type in the function indicates that the function does not return anything. Function declarations may also include void in argument list in both function definition and declarations to indicate that a function does not require argument .The general form for this types of functions is data-type func_name (void).

Check your progress
<p>1. What is a function?</p> <p>.....</p> <p>.....</p> <p>.....</p> <p>2. What is prototyping? Why it is necessary?</p> <p>.....</p> <p>.....</p> <p>.....</p>

1.10 LET US SUM UP

- A function is a self-contained program segment that performs some specific well defined task.
- Three steps in using a function are declaring a function, defining a function and calling the function.
- A function is called by specifying its name, followed by a pair of parenthesis, which contains parameters if needed.

- The argument that represents the names of data items that are transformed into the function from the calling portion of the program are called as formal arguments or formal parameters.
- The corresponding arguments in the function reference which define the data items that are actually transferred are called as actual arguments or actual parameters.
- The return statement is used to return the information from the function to the calling function of the program
- A function may be declared anywhere as long as its declaration is above all reference to the function. So, a function should be declared before it is called.
- The execution of a function is terminated when it executes the return statement or when the last statement of the function is executed or when it encounters the closing brace of the function.
- A function prototype declares the return type of the function and declares the number type and the sequence of the parameters that a function will receive.

1.11 LESSONS END ACTIVITIES

1. What is difference between function declaration and function definition?
2. What is the **return ()** statement mandatory in a function?
3. State several advantages to the use of functions.
4. Explain the meaning of the following function declaration:

```
double f(double a,int b);
```
5. What is the purpose function prototyping?
6. State whether the following statements are true or false.
 - (a) C functions can return only one value under their function name.
 - (b) A function in C should have at least one argument.
 - (c) A function can be defined and placed before the main function
 - (d) A user-defined function must be called at least once; otherwise a warning message will be used.
 - (e) The return type of a function “float” by default.

1.12 KEYWORDS

main (): `main()` is a specifically recognized function in C. Every Program must have a `main()` function to indicate where the program has to begin its execution.

Function Definition: function definition is an independent program module that is specially written to implement the requirements of the function.

Function Call: The program or a function that calls the function is referred to as the calling program or calling function.

Function Definition or Function Prototype: The calling program should declare any function (like declaration of variable) that is to be used later in the program. This is known as the function declaration or function prototype.

Function Body: The function body contains the declarations and statements necessary for performing the required task. The body enclosed in braces, contains three parts.

return (): A `return()` statement that returns the value evaluated by the function.

Function parameters: Function parameters are the means of communication between the calling and the called functions.

Formal parameters: The formal parameters (commonly called parameters) are the parameters given the function declaration and function definition.

1.13 QUESTIONS FOR DISCUSSION

1. What is the need for functions?
2. What is a function?
3. Explain the general form of defining a function.
4. What are formal arguments?
5. What are actual arguments?
6. What is function header?
7. What do you mean by function call?
8. Difference between build-in functions and user-defined functions. Give examples.
9. Define function prototype.
10. What is calling function?
11. What is called function?
12. Write a program to swap two numbers using functions.

Check your Progress: Model Answers

Ans.1

A function is a self-contained program segment that carries out some specific, well-defined task. Functions break large computing tasks into smaller ones, C has been designed to make a function efficient and easy to use. C programs generally consist of many small functions rather than a few big ones.

Ans.2

Whenever a call is made to a function the compiler assumes that this function would return a value of type **int**. If you decide that a function should return a value other than **int**, then it is necessary to mention the first line of the function in the program, before it is used, which is called the **function prototype** also referred as the **function declaration**.

1.14 SUGGESTED READING

1. Programming in ANSI C - Agarwal
2. Let us C - Kanitkar
3. Programming in ANSI C - Balguruswamy
4. How to solve it by Computer: Dromey, PHI
5. Schaums outline of Theory and Problems of programming with C : Gottfried

UNIT-2

UNIT

2

FUNCTIONS

CONTENTS

- 2.1 Aims and Objectives
- 2.2 Introduction
- 2.3 Category of Functions
- 2.4 No Arguments and no return values
- 2.5 Arguments but no return values
- 2.6 Argument with return values
- 2.7 No argument but returns a value
- 2.8 Function that return multiple values
- 2.9 Let Us Sum Up
- 2.10 Lesson end Activities
- 2.11 Keywords
- 2.12 Questions for Discussion
- 2.13 Suggested Readings

2.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to :

- Discuss for different categories of function
- Understanding different categories of function with example
- Define function that return multiple values
- Discuss arguments with and without values

2.2 INTRODUCTION

A function in C language is a block of code that performs a specific task. It has a name and it is reusable i.e. it can be executed from as many different parts in a C Program as required. It also optionally returns a value to the calling program.

Some of the important properties of function given below,

- ❖ Every function has a unique name. This name is used to call function from “main ()” function. A function can be called from within another function.
- ❖ A function is independent and it can perform its task without intervention from or interfering with other parts of the program.
- ❖ A function performs a specific task. A task is a distinct job that your program must perform as a part of its overall operation, such as adding two or more integer, sorting an array into numerical order, or calculating a cube root etc.
- ❖ A function returns a value to the calling program. This is optional and depends upon the task your function is going to accomplish. Suppose you want to just show few lines through function then it is not necessary to return a value. But if you are calculating area of rectangle and wanted to use result somewhere in program then you have to send back (return) value to the calling function.

C language is collection of various inbuilt functions. If you have written a program in C then it is evident that you have used C's inbuilt functions. `printf`, `scanf`, `exit`, all are C's inbuilt functions. You cannot imagine a C program without function

2.3 CATEGORY OF FUNCTIONS

A function depending on whether arguments are present or not and whether a value is returned or not, hence functions are categorized in the following manner:

:

A function may belong to any one of the following categories:

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.
4. Functions that return multiple values.
5. Functions with no arguments and return values

2.4 NO ARGUMENTS AND NO RETURN VALUES

When a function has no arguments, it does not receive any data from the calling function, similarly when a function has no return values, the calling function does not receive any data from the called function. There is no data transfer between the calling function and the called function. The given diagram describes no arguments and no return values that is, no data communication between functions.

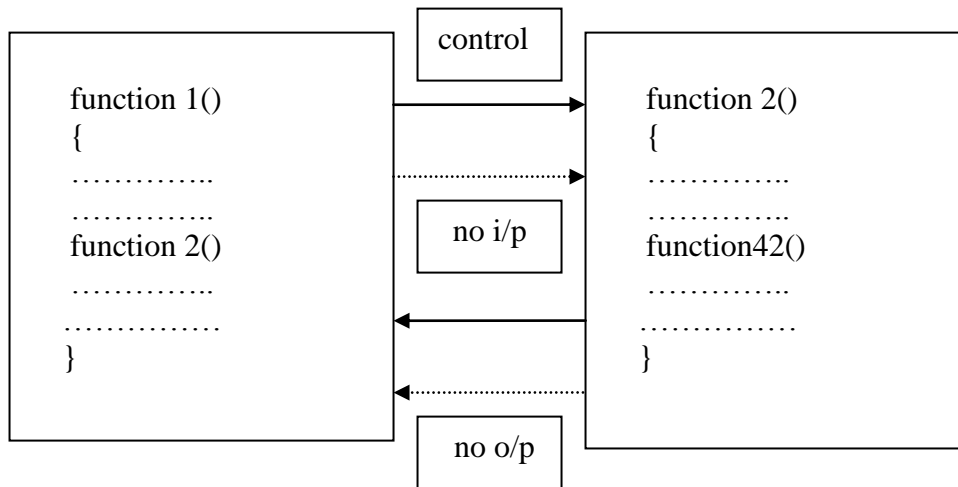


FIGURE 1.18

EXAMPLE

To write a program using function no arguments no return values

```
#include<stdio.h>
#include<conio.h>
Void main()
{
    display (); }
Void display()
{
    char empname[25];
    printf("Enter the Employee Name:");
    scanf("%c",&empname);
    printf("%c Employee Name is:",empname);
}
```

Sample program output

```
Enter the Employee Name: senthil
Employee Name is: senthil
```

main() is the calling function which calls the function **display()**. The function **display()** contains no arguments and hence, there are no argument declarations. Note that the called function that is **display** receives its data that is name of the employee directly from the input terminal that is keyboard and display the contents of a employee name to the output terminal that is screen in the called function itself. No **return** statement is employed since there is nothing to be returned. The closing brace of the function indicates the end of execution of the function, thus returning the control, back to the calling function. The keyword void is used before the function **display()** to indicate that there are no return values.

2.5 ARGUMENTS BUT NO RETURN VALUES

When a function has arguments, it receives data from the calling function. The main () function will not have any control over the way in which the functions receives its input data. We can also make the calling function to read data from the input terminal and pass it to the called function. This approach seems better because the calling function can check the validity data, before it is passed over to the called function.

NOTE

- Function accepts argument but it does not return a value back to the calling program.
- It is single (one-way) type communication.
- Generally output is printed in the called function
- Here area is called function and main is calling function.

EXAMPLE

Write a program to find area of circle using function.

```
#include<stdio.h>
#include <conio.h>
void area;
(float rad);
void main()
{
    float rad;
    printf("\n Enter the radius:");
    scanf("%f",&rad);
    area(rad);
    getch();
}
void area(float rad)
{
    float area;
    area=3.14*rad*rad;
    printf("Area of circle = %f",area); }
```

Sample program output

Enter the radius : 3
Area of circle=28.2640000

The function receives one arguments (**i.e area ()**) .The argument are of floating type. No **return** statement is employed since there is nothing to be returned. The closing brace of the function signals the end of the function thus returning the control back to the calling function. The keyword void is used before the function name **area()** to indicate that there are no return values.

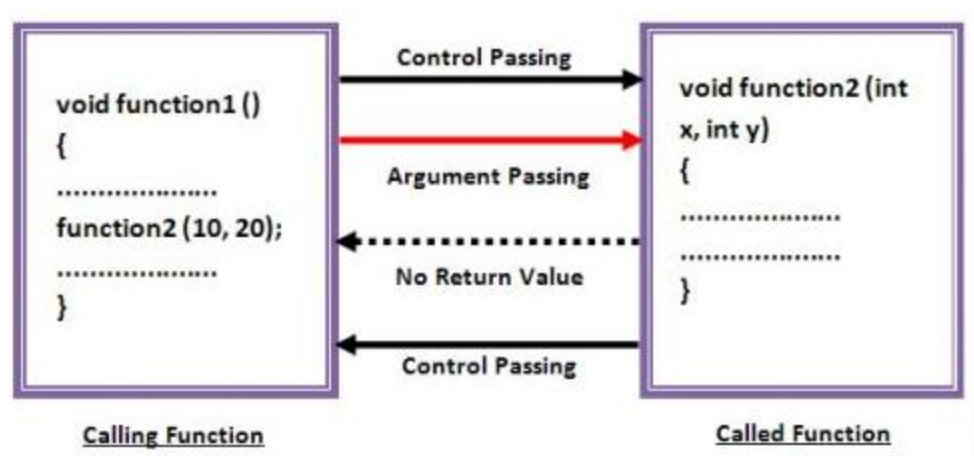


FIGURE 1.19

2.6 ARGUMENTS WITH RETURN VALUES

When a function has arguments it receives data from the calling function and does some process and then returns the result to the called function. In the way the main() function will have control over the function. This approach seems better because the calling function can check the validity of data before it is passed to the called function and to check the validity of the result before it is sent to the standard output even (i.e., screen). Note that when a function is called, a copy of the values of actual arguments is passed to the called function.

EXAMPLE

To write a program to find sum of two numbers.

```
#include<stdio.h>
#include<conio.h>
int add(int x, int y)
{
    int result;
    result = x+y;
    return(result);
}
```

```

void main()
{
    int z;
    clrscr();
    z = add(952,321);
    printf("Result %d.\n\n",add(30,55));
    printf("Result %d.\n\n",z);
    getch();
}

```

Sample program output

Result 85
Result 1273

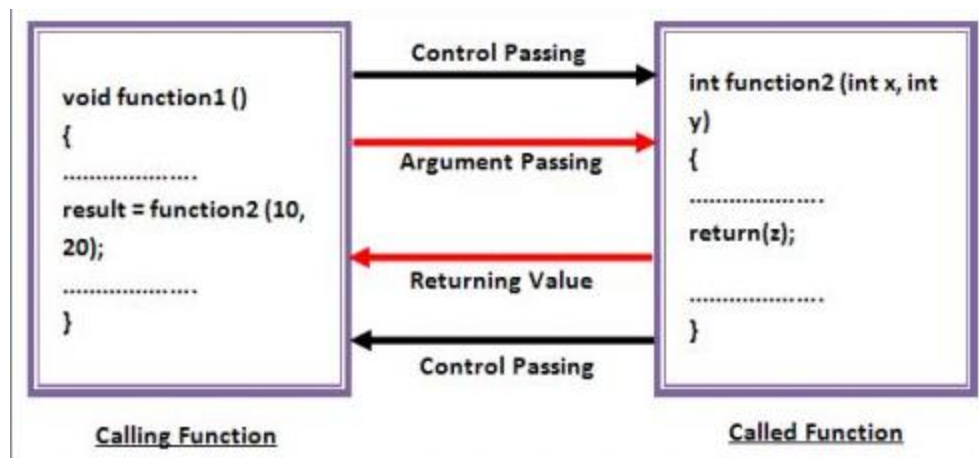


FIGURE 1.20

This program sends two integer values (x and y) to the User Defined Function “add()”, “add()” function adds these two values and sends back the result to the calling function (in this program to “main()” function). Later result is printed on the terminal.

This int is the return type of the function, means it can only return integer type data to the calling function. If you want any function to return character values then you must change this to char type. You can assign any integer value to experiment with this return which ultimately will change its output. Why we are using integer variable “z” here? You know that our User Defined Function “add()” returns an integer value on calling. To store that value we have declared an integer value. We have passed 952, 321 to the “add()” function, which finally return 1273 as result. This value will be stored in “z” integer variable. Now we can use “z” to print its value or to other function.

2.7 NO ARGUMENTS BUT RETURNS A VALUE

When a function has no arguments, it does not receive any data from the calling function, but it can do some process and then return the result to the called function. Hence there is data transfer between the calling function and the called function.

EXAMPLE

To write a program to take one integer from keyboard and display it

```
#include<stdio.h>
#include<conio.h>
int send()
{
    int no1;
    printf("Enter a no : ");
    scanf("%d",&no1);
    return(no1);
}
void main()
{
    int z;
    clrscr();
    z = send();
    printf("\nYou entered : %d.", z);
    getch();
}
```

Sample program output

```
Enter a no: 5
You entered: 5
```

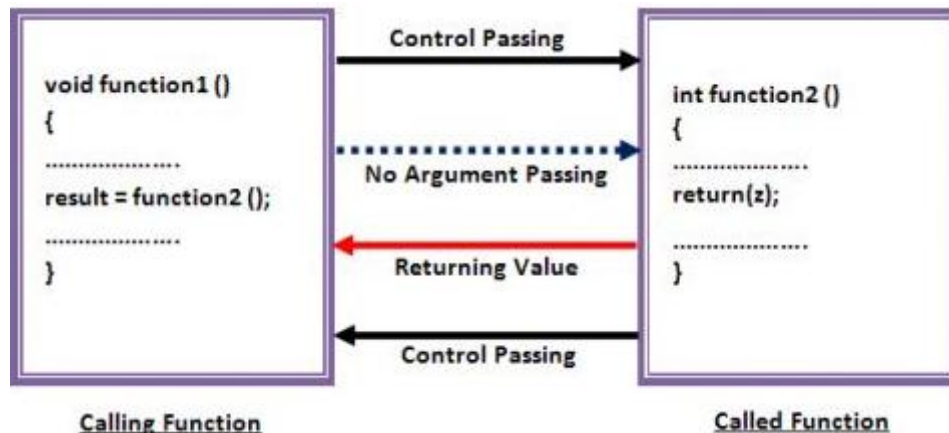


FIGURE 1.21

In this program we have a User Defined Function which takes one integer as input from keyboard and sends back to the calling function.

2.8 FUNCTION THAT RETURN MULTIPLE VALUES

In a function, return statement was able to return only single value. That is because; a return statement can return only one value. But if we want to send back more than one value. We have used arguments to send values to the called function, in the same way we can also use arguments to send back information to the calling function. The arguments that are used to send back data are called **Output Parameters**.

EXAMPLE

To write a program to add, subtract of two numbers using function.

```
#include<stdio.h>
#include<conio.h>
void calc(int x, int y, int *add, int *sub)
{
    *add = x+y;
    *sub = x-y;
}
void main()
{
    int a=20, b=11, p,q;
    clrscr();
    calc(a,b,&p,&q);
    printf("Sum = %d, Sub = %d",p,q);
    getch();
}
```

Sample program output

Sum=31, Sub=9

We call User Defined Function “calc ()” and sends argument then it adds and subtract that two values and store that values in their respective pointers. The “*” is known as indirection operator whereas “&” known as address operator. We can get memory address of any variable by simply placing “&” before variable name. In the same way we get value stored at specific memory location by using “*” just before memory address.

This User Defined Function is different from all above it implements pointer. Pointer can only store address of the value rather than value but when we add * to pointer variable then we can store value of that address. When we call “calc()” function in the line no. 12 then following assignments occurs. Value of variable “a” is assigned to “x”, value of variable “b” is assigned to

“y”, address of “p” and “q” to “add” and “sub” respectively. In line no. 5 and 6 we are adding and subtracting values and storing the result at their respective memory location.

Check your progress

1. Define category of functions.

.....
.....
.....

2. What is mean by no arguments and no return values?

.....
.....
.....

2.9 LET US SUM UP

- A return statement can occur anywhere within the body of a function.
- A function definition may be placed either after or before the main function.
- When the value returned is assigned to a variable, the value will be converted to the type of the variable receiving it.
- A function with void return type cannot be used in the right-hand side of an assignment statement. It can be used only as a stand-alone statement.
- A function that returns a value cannot be used as a stand-alone statement.
- Where more functions are used, they may be placed in any order.
- A global variable used in a function will retain its value for future use.
- A local variable defined inside a function is known only to that function. It is destroyed when the function is exited.

2.10 LESSION END ACTIVITIES

1. Fill in the blanks in the following statements.

- a. The parameters used in a function call are called_____.
- b. A variable declared inside a function is called_____.
- c. By default,_____ is the return type of a C function.
- d. In prototype declaration, specifying _____ is optional.

- e. A function that calls itself is known as a _____ function.
- f. If a local variable has to retain its value between calls to the function, it must be declared as_____.
- g. A_____ the compiler to check the matching between the actual arguments and the formal ones.

2. Find errors in the following function calls:

- a. void xyz ()
- b.xyz (void);
- c.xyz (int x,int y);
- d.xyz () + xyz ();

2.11 KEYWORDS

Category of Functions: A function depending on whether arguments are present or not and whether a valued is returned or not depending on types of function based.

No arguments and no return values: When a function has no arguments, it does not receive any data from the calling function.

Arguments but no return values: The calling function to read data from the terminal and pass it on to the called function.

Arguments with return values: It receives data from the calling function through arguments, but does not send back any value.

No arguments but return a value: We may need to design a function that may not take any arguments but return a value to the calling function.

Functions that return multiple values: A return statement can return only value, but functions that return multiple values.

Output parameters: The arguments than are used to “send out” information are called output parameters.

Check your Progress: Model Answers

Ans.1

A function depending on whether arguments are present or not and whether a value is returned or not based these function categories in the following manner. A function may belong to any one of the following categories: Functions with no arguments and no return values, Functions with arguments and no return values, Functions with arguments and return values, Functions that return multiple values, Functions with no arguments and return values.

Ans.2

When a function has no arguments, it does not receive any data from the calling function, similarly when a function has no return values, the calling function does not receive any data from the called function. There is no data transfer between the calling function and the called function.

S

2.12 QUESTIONS FOR DISCUSSION

1. What is a function? Define its properties.
2. Explain different categories of functions with example.
3. Explain functions with no arguments and no return values.
4. Explain functions with arguments and no return values.
5. Explain functions with arguments and return values.
6. Explain functions that return multiple values.
7. Functions with no arguments and return values.

2.13 SUGGESTED READING

1. Structured Programming Approach C.Behrouz A.Forouuzan and Richard F.Gilberg, 2nd Edition, Thomson 2001.
2. Programming with ANSI and Turbo C, Ashok N.Kamathane, Ist Edition, Pearson Education Asia 2002.
3. Beginning C:From Novice to Professional.Ivor Horton,4th Edition,Springer,India 2006.

UNIT-3

UNIT

3

POINTERS

CONTENTS

- 3.1 Aims and Objectives
- 3.2 Introduction
- 3.3 Understanding Pointers
- 3.4 Accessing the Address of Variable
- 3.5 Declaring a Pointer Variables
- 3.6 Initialization of Pointer Variables
- 3.7 Accessing Variable through its Pointers
- 3.8 Pointer Example Programs
- 3.9 Let Us Sum Up
- 3.10 Lesson End Activities
- 3.11 Keywords
- 3.12 Questions for Discussion
- 3.13 Suggested Readings

3.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to;

- Discuss for Pointers Definition and Initialization of Pointer Variables
- Understanding Accessing Variable through its Pointers
- Define Declaring Pointer Variables
- Discuss Different example Program for Pointers

3.2 INTRODUCTION

Pointers are a powerful concept in C and add to its strength. The pointer enable to

1. Write efficient and concise programs
2. Establish inter-program data communication
3. Dynamically allocate and De-Allocate memory
4. Optimize memory space usage
5. Deal with hardware components
6. Pass variable number of arguments to functions

The memory (RAM) of a computer is organized as an array of consecutively memory cells. Since there usually will be several thousands of memory location in the RAM, the CPU needs some mechanism to identify and request transfer of data from and to a particular location. This memory is called **Addressing**.

Each location in the RAM has been given a unique identification number, called an address of that memory location. Therefore, it is imperative that, every memory location should have only one address and one address should correspond to only one memory location. Every variable and every function has an address and these addresses are distinct for each variable and function.

The identifiers used for variables and functions enable the programmer to refer memory locations by name. A variable name is a symbolic reference given to a memory location. Each variable and character string has an address that describes its location. The compiler translates these names into address. For example, consider the statement `product=var1*var2`.

If `var1` occupies the address 1000 and `var2` occupies the address 1005, and `product` occupies 1025, the above statement will be interpreted by the compiler as move the contents of the memory location 1000 and 1005 into the CPU registers, multiply them and move the resultant product to the memory location 1020.

A pointer is a variable that represents the location (rather than a value) of a data item, such as variable or an array element.

3.3 UNDERSTANDING POINTERS

The computer's memory is a sequential collection of '**Storage Cells**' as shown in following figure. Each cell, commonly known as a **byte**, has a number called **Address Associated** with it. Typically; the addresses are numbered consecutively, starting from zero.

The last address depends on the memory size. A computer system having 64K memory will have its last address as 65535.

Address	Location
0	
1	
2	
:	
:	.
:	.
:	.
:	.
:	.
:	.
:	.
:	.
:	.
:	.
:	.
:	.
1022	
1023	
1024	

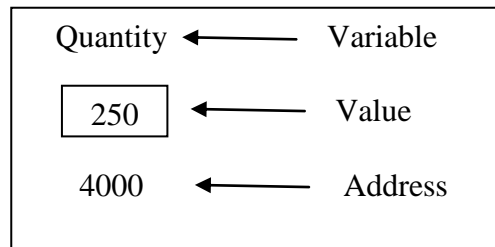
We know that variables are to be declared before they are used in a program. Declaration of a variable conveys two things to the compiler. As a result, the compiler

1. Allocates a location in memory. The number of bytes in the location depends on the data type of the variable.
2. Establishes a mapping between the address of the location and the name of the variable.

Whenever we declare a variable, the system allocates some location in the memory, an appropriate location to hold the value of the variable. Since every byte has a unique address number, this location will have its own address number. Consider the following declaration of statement:

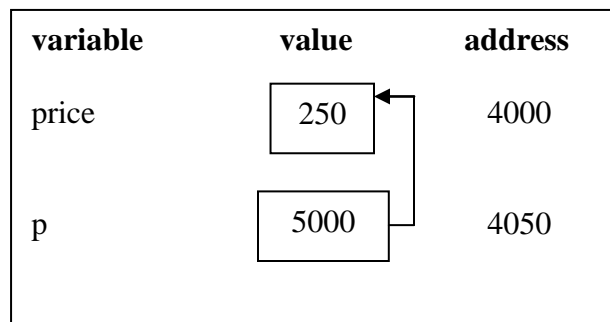
int price=250;

This statement tells the system to find a location for the integer variable **price** and put the value 250 in the location. Let us consider that the system has chosen the address location 4000 for price. Representation of variable in the following figure:



During execution of the program, the system always associates the name **price** with the address 5000. We may access to the value 250 by using either the name price or the address 4000. Since memory address are simply numbers, they can be assigned to some variables, which can be stored in memory, like any other variable. Such variables that hold memory address are called pointer variables. A pointer variable is, therefore, nothing but a variable that contains an address, which is a location of another variable in memory.

Since a pointer is a variable, its value is also stored in the memory in another location. Suppose, we assign the address of **price** to a variable **p**. The link between the variables **p** and **quantity** can be visualized as shown in the following figure. The address of **p** is 4050.



Since the value of the variable **p** is the address of the variable **price**, we may access the value of **price** by using the value of **p** and therefore, the variable **p** 'points' to the variable **price**. Thus **p** gets the name 'pointer'

Pointers are constructing on the three important concepts as shown below:

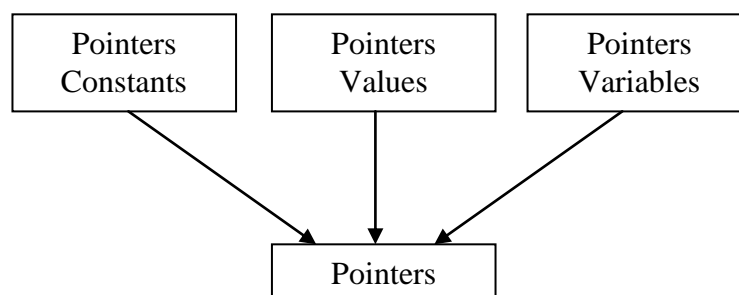


FIGURE 1.22

Memory address within a computer is referred to as pointer constants. We cannot change them; we can only use them to store data values. We cannot save the value of a memory address directly. We can only obtain the value through the variable stored there using the address operator (&). The value thus obtained is known as pointer value. The pointer value may change from one program to another.

Once we have a pointer value, it can be stored into another variable. The variable that contains a pointer value is called a pointer variable.

3.4 ACCESSING THE ADDRESS OF A VARIABLE

We declare a variable; the amount of memory needed is assigned for it at a specific location in memory (its memory address). The address that locates a variable within memory is what we call a **reference** to that variable. This reference to a variable can be obtained by preceding the identifier of a variable with an ampersand sign (&), known as reference operator, and which can be literally translated as "address of".

EXAMPLE

p = &quantity;

This would assign to `p` the address of variable `quantity`, since when preceding the name of the variable `quantity` with the reference operator (&) we are no longer talking about the content of the variable itself, but about its reference (i.e., its address in memory).

From now on we are going to assume that `quantity` is placed during runtime in the memory address 2000. This number (2000) is just an arbitrary assumption we are inventing right now in order to help clarify some concepts in this tutorial, but in reality, we cannot know before runtime the real value of address of a variable will have in memory.

Consider the following code fragment:

```
1.quantity = 25;  
2.p = quantity;  
3.p = &quantity;
```


EXAMPLE

Write a program to print the address of a variable along with its value.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char a;
    int x;
    float p,q;
    a='A';
    x=125;
    p=11,q=35.50;
    printf("%c is stored at address %u \n",a,&a);
    printf("%d is stored at address %u \n,x,&x");
    printf("%f is stored at address %u \n",p,&p);
    printf("%f is stored at address %u",q,&q);
}
```

3.5 DECLARING POINTER VARIABLES

Declaring pointer variable is quite similar to declaring a normal variable but before insert a star '*' operator before it.

SYNTAX

data-type *pt-name;

where

- 1.The asterisk(*)->represents that the variable **pt-name** is a pointer variable
- 2.pt-name->represents memory location
- 3.pt_name->represents points to a variable of type **data-type**

EXAMPLE 1

int *p;

declares the variable **p** as a pointer variable that points to an integer data type. That the data type **int** refers to the data type of the variable being pointer by **p** and not the value of the pointer.


EXAMPLE 2

```
float *x;
```

Declares **x** as a pointer to a floating-point variable.

The above pointer variable declaration cause the compiler to allocate memory locations for the pointer variable **p** and **x**. Since the memory locations have not been assigned any values, these locations may contain some unknown values in them and therefore they point to unknown locations as shown in the following figure given below:

```
int *p;
```



The diagram illustrates a pointer variable 'p' which is represented by a small box. To the right of this box is a larger box containing a question mark '?', representing an unknown memory location that the pointer 'p' points to.

Contains garbage points to unknown location

Can we declare the pointer variable in the following type of the manner,

1. `int *p;`
2. `int *p;`
3. `int *p;`

In the second type has become most popular because the following reason,

1. It is convenient to declare multiple declarations in the same manner

EXAMPLE

```
int *p,x,*q;
```

2. This type matches with the format used for accessing the target values.

EXAMPLE

```
int x,*p,y;  
x=10;  
p=&x;  
y=*p;  
*p=20;
```

3.6 INITIALIZATION OF POINTER VARIABLES

Like ordinary variables, a pointer variable can also be initialized. Static and external (global) pointer variables are initialized with **NULL** by default. When the stat ‘* operator’ is applied to a pointer variable to which any address has not been assigned, the pointer points to an

unknown location. So, applying * to such a pointer is dangerous. Therefore, before a pointer variable is used to access an object, the pointer should be made to point at a valid object.

Once a pointer variable has been declared we can use the assignment operator to initialize the variable.

EXAMPLE

```
int price;  
int *p;  
p=&price;
```

We can also possibly combine the initialization with the declaration. That is,

```
int *p=&price;
```

Pointers are used to be allowed. The only represent here is the variable **price** must be declared before the initialization takes place.

Pointer variables always point to the corresponding type of data.

EXAMPLE

```
float a,b  
int x,*p;  
p=*a;  
b=*p;
```

will result erroneous output because we are trying to assign the address of a **float** variable to an integer pointer. When we declare a pointer to be **int** type, the system assumes that any address that the pointer will hold will point to an integer variable. Since the compiler will not detect such errors, care should be taken to avoid wrong pointer assignments.

```
int x,*p=&x;
```

The above statement is perfectly valid. It declares **x** as an integer variable and **p** as a pointer variable and then initializes **p** to the address of **x** and the following statement is also wrong that the target variable **x** is declared first.

The statement given below,

```
int *p=&x,x;
```

is not valid.

It also defines the pointer variable with an initial value **NULL** or **0(Zero)**. The following statement given below,

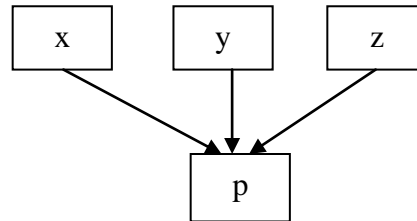
```
int *p=NULL;
int *p=0;
```

with the exception of NULL and 0, no other constant value can be assigned to a pointer variable.

```
int *p=5000;
```

It is also possible that pointers are flexible. We can make the same pointer point to different data variables in different statements.

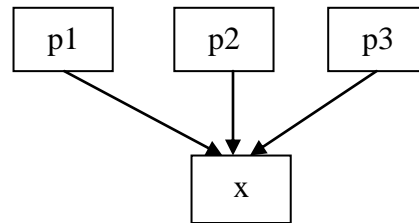
```
int x,y,z,*p;
.....
p=&x;
.....
p=&y;
.....
p=&z;
.....
```



It can also be possible for different pointers to point to the same data variable.

EXAMPLE

```
int x;
int *p1=&x;
int *p2=&x;
int *p3=&x;
```



3.7 ACCESSING VARIABLE THROUGH ITS POINTERS

Pointers are variables that contain address. Suppose **v** is a variable that represents some particular data item. The compiler will automatically assign memory cells for this data item. The data item can be accessed if the location (that is the address of the memory cell) is known. The address of **v**'s memory location can be determined by the expression **&v**, where **&** is a unary operator called an address operator. The address **v** is assigned to another variable **pv** as follows: **pv=&v**. This new variable is called as pointer to **v**, since it points to the location where **v** is stored.

In above explanation, **pv** represents **v**'s address, not its value. Thus, **pv** is referred to as a pointer variable. Pointer variables are declared as follows:

type *variable_name;

For example, a pointer to a character variable is declared as, **char *cpt**; where the variable's name is **cpt** and the value it holds is a character's address. The asterisk preceding the variable's name informs the compiler that this variable does not hold the character's address and not a

character. The * used above is a unary operator and is known as the indirection or **referencing operator**. When applied to a pointer, it accesses the object the pointer points to.

A Variable that contains integer's address is declared as follows: `int *iptr;` where `iptr` is the name of the pointer variable. Several pointers can also be declared in a single declaration as: `int x, y, *iptr, *jptr;` where `x` and `y` is regular integers and `iptr` and `jptr` are pointers to integer type of variables. Each data type in C has its own associated pointer type.

A Pointer is to void the generic pointer and can be used to point any type of object. This implies that a pointer of any type can be assigned to pointers of type void (and vice versa), if appropriate type casts are used. The void pointer is particularly useful when various types of pointers are manipulated by a single routine. The void pointer is declared as follows `void *p.`

To obtain a variable's address, the `&` operator is used. This operator applies only to objects in the memory such as variables and array elements. It cannot apply to expressions or constants.

EXAMPLE

```
#include<stdio.h>
void main()
{
    int x,*intptr;
    intptr=&x;
    printf("Enter an integer");
    scanf("%d",intptr);
    printf("\n The value entered is %d \n",x);
}
```

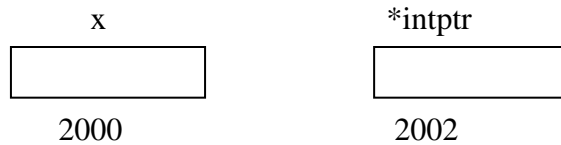
Sample program output

```
Enter an integer: 1234
The value entered is: 1234
```

In the above program, two variables, an integer name `x` and pointer to an integer name `intptr`. The variable `intptr` is set to the address of the variable `x`. The value of the pointer variable, `intptr` is passed to the `scanf ()` function instead of `&x`, the address of `x`.

In the above example, if the variable `x` is at memory location 2000 and `intptr` which is also another variable (pointer) has its own memory location. The value assignments and storage locations in the program `samepoint.c` would be given below:

After the int x,*intptr;



After the line intptr=&x;



After the line of scanf("%d",input)



3.8 POINTER EXAMPLE PROGRAMS

EXAMPLE 1

Write a program to swap two numbers using pointers

```
#include<stdio.h>
void main()
{
    int a,b;
    printf("\n Enter two Numbers...");
    scanf("%d %d",&a,&b);
    clrscr();
    printf("\n Before Exchange...%d \t %d",a,b);
    exchange(&a,&b);
    printf("\n After Exchange...%d \t %d \t",a,b);
    getch();
}
exchange(m,n)
int *m,*n;
{
    int t;
    t=*m;
    *m=*n;
    *n=t;
    return; }
```

EXAMPLE 1

Write a program to find largest of two numbers using pointer

```
include <stdio.h>
int main ()
{
    int data[100];
    int* p1;
    int *p2;
    for (int i = 0; i <100;i++)
    {
        data[i] = i;
    }
    p1 = &data [1];
    p2 = &data [2];
    if (p1 > p2)
    {
        printf ("\n\n p1 is greater than p2");
    }
    else
    {
        printf ("\n\n p2 is greater than p1");
    }
}
```

Check your progress

1. Define a pointer variable.

.....
.....
.....

2. Explain the uses of pointers.

.....
.....
.....

3.9 LET US SUM UP

- A pointer is a variable, which represents the memory location (not the value) of a data items, such as a variable or an array element.

- Like all variables a pointer variable should also be declared before they are used.
- The ampersand operator (&) gives the address of a variable.
- The three values that can be used to initialize the pointer are zero, null and address.
- The pointer that has not been initialized is referred to as the dangling pointer.
- Only an address of a variable can be stored in a pointer variable.
- Do not store the address of a variable of one type into a pointer variable of another type.
- The value of a variable cannot be assigned to a pointer variable.
- A pointer variable contains garbage until it is initialized.

3.10 LESSON END ACTIVITIES

1. What is the output of the following program?

```
void main()
{
    int a[10],*ap,*bp;
    ap=a;
    bp=&a[0];
    printf("%d",ap,bp);
}
```

2. What is the output of the following program?

```
void main()
{
    int a[10]={ 1,2,3,17},*ap,*bp;
    ap=a;
    bp=&a[3];
    printf("%d %d",*ap,*bp));
}
```

3. Fill in the blanks in the following statements:

- A pointer variable contains as its value the_____of another variable.
- The_____operator is used with a pointer to de-reference the address Contained in the pointers.

C.The_____operator returns the value of the variable to which it has operand points.

D.The only integer that can be assigned to a pointer variable is_____.

E.The pointer that is declared as_____cannot be de-referenced.

4. How is a pointer initialized?

5. Explain the uses of pointers.

3.11 KEYWORD

Pointers: A pointer is a variable, which represents the location (not the value) of a data item, such as variable or array element.

Address: A byte is a basic storage and accessible unit in memory.

Indirection operator or deference operator: The indirection operator (*) is also called the dereference operator. When a pointer is dereferenced, the value at the address stored by the pointer is retrieved.

Pointer Initialization: A pointer is a variable that contains the memory location of another variable.

Pointer Arithmetic: The arithmetic operations that can be performed on pointers are addition and subtraction. You can also use increment or decrement operator.

3.12 QUESTIONS FOR DISCUSSION

1. What is the meaning of the following declaration: `int *px;`
2. The indirection operator can be applied to_____type of operand.
3. State the uses of address operator in pointers.
4. State the uses of indirection operator in pointers.
5. State the need for memory allocation in programs.
6. Write a program to find the largest of three number using pointers.
7. Write a program to sort the strings in alphabetical order using pointers.
8. What are the advantages of pointer?
9. Give the significance of & and * operators.

Check your Progress: Model Answers**Ans.1**

Pointers are variable that contain address. Suppose v is a variable that represents some particular data item. The compiler will automatically assign memory cells for this data item. The data item can be accessed if the location that is address of the memory cell is known.

Ans.2

Usage of pointers result in a more compact and efficient code. Pointers can be used to achieve clarity and simplicity. Pointers are used to pass information back and forth between function and its reference point. They provide a way to return multiple area items from a function using its function arguments pointers also provide an alternate way to access an array element. Pointers enable one to access the memory directly.

3.13 SUGGESTED READING

1. Programming with C : Bryon Gottfried
2. Let us C: Yashwant Kanetkar.
3. C programming: Dennis Ritchie
4. Programming in ANCI C: Balgurusamy
5. Graphics under C: Yashwant Kanetkar
6. Pointers in C: Yashwant Kanetkar

UNIT-4

UNIT

4

STRUCTURES AND UNION

CONTENTS

- 4.1 Aims and Objectives
- 4.2 Introduction
- 4.3 Declaring a Structure
- 4.4 Defining a Structure
- 4.4 Accessing Structure Members
- 4.5 Initializing a Structure
- 4.6 Operations on Structures
- 4.7 Arrays and Structures
 - 4.7.1 Arrays of Structures
 - 4.7.2 Arrays within Structures
- 4.8 Union
- 4.9 Difference between Structure and Union
- 4.10 Let Us Sum Up
- 4.11 Lesson end Activities
- 4.12 Keywords
- 4.13 Questions for Discussion
- 4.14 Suggested Readings

4.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to:

- Discuss the structures definition and initialization of structure variables
- Understanding Accessing Structure Members
- Define operations and array of structures
- Discuss about union

4.2 INTRODUCTION

A **Structure** is a derived type usually representing a collection of variables of same or different data types grouped together under a single name. The variables or data items in a structure are called as members of a structure. A structure may contain one or more integer variables, floating-point variables characters variables, arrays, pointers, and even other structures can also be included as members. Structures help to organize data, particularly in large programs, because they allow a group of related variable to be treated as a single unit.

There are two fundamental differences between structures and arrays.1.An array demands a homogeneous data type, i.e., the elements of an array must be of the same data type, where as structure is a heterogeneous data type, since it can have any data type as its member.2.The difference is that elements in an array are referred by their positions, where as members in a structure are referred by their unique name.

4.3 DECLARING A STRUCTURE

A structure within a C program is defined as follows:

```
struct struct-type
{
    member-type1  member-name1;
    member-type2  member-name2;
    .....
    .....
}
```

Where,

struct->is a keyword

struct type->is a name (tag) that identifies structures of composition member-type1

member name1.member-type2 member name2...are individual declaration.

The individual members can be ordinary variables, pointers, arrays, or other structures. The member names within a particular structure must be distinct from one another; though a member name can be the same as the name of a variable defined outside the structure.

A storage class cannot be assigned to an individual member, and individual members cannot be initialized within a structure type declaration. Unlike the declaration of a variable of array, defining a structure causes no storage to be reserved.

By defining a structure the programmers derives a new data type composed of a collection of already known data types and their name. For example, suppose that the information about 10000 business accounts has to be maintained. The information consists of mixed data types, where for each account the following are required.

EXAMPLE

- account number (int)
- account type (short)
- name (30 char array)
- street address (30 char array)
- city/state/zip (30 char array)
- balance long
- last payment (long)
- date

```
struct account
{
    int      acct_no;
    short    acct_type;
    char      name[30];
    char      street[30];
    char      city_state[30];
    long      balance;
    long      last_payment;
};
```

4.4 DEFINING A STRUCTURE

Declaring a structure is just a skeleton it merely describes the template. It does not reserve any memory space rather than the declaration creates a new data type. To use the structure you have to define it. Defining a structure means creating variables to access the members in the structure. Creating a structure variable allocates sufficient memory space to hold all the members of the structure. Structures variables can be created during structure declaration or by explicitly using the structure name.

SYNTAX

```
struct <structure-name>
{
    data type member1;
    data type member2;
    data type member3;
    :
    :
    data type member;
}structure variable(s);
```

EXAMPLE

```
struct employee
{
    int empno;
    char empname[15];
    float salary;
}empl
```

Where the structure employee declares a variable empl of its type. The **structure variable(s)** are declared like an ordinary variable is used to access the members of the structure. More than one **structure variable** can also be declared by placing a comma in between the variables.

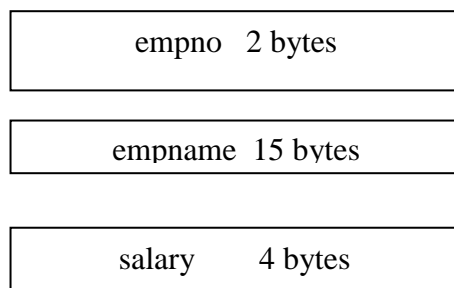
Once the composition of the structures has been defined, individual structures variables can be declared as follows: **storage class struct struct_type variable 1, variable2... variable;**

A variable of the above type id declared like this: struct account. Where the variable name is customer and the data type is struct account. It is also possible to define a structure and declare a variable of that at the same time:

SYNTAX

```
Storage-class struct struct-type
{
    member_type1 memebr_name1;
    member_type2 memebr_name2;
    member_type3 memebr_name3;
    .....
    member_typen memebr_namen;
}
variable1, variable2....variable n;
```

The following figure shows the memory allocation of structure shown in the following figure:



NOTE

The keyword struct is optional when defining the structure using the structure name.

4.5 INITIALIZING A STRUCTURE

Similar to initialization of arrays, we can initialize structure variables also. The initializing a structure variable expressed is expressed as:

SYNTAX

```
struct <structure_name>
{
    data type member1;
    :
    :
    :
    data type member;
}structure_variable={value1,value2,.....valueN};
```

EXAMPLE

```
struct employee
{
    int height;
    float weight;
}emp1={165,60};
```

the above initialization initializes **165** to the structure member height and **60** to the structure member weight. The value to be initialized for the structure members must be enclosed within a pair of braces.

NOTE

The constants to be initialized to the structure members must be in the same order in which the members are declared in the structure.

Write a program to initializing a structure using the structure name.

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int empno;
    char name[20];
    float salary;
};
```



```

void main()
{
    struct emp e1,e2;
    int size;
    clrscr();
    printf("Enter empno,name and salary \n");
    scanf("%d %s %f",&e1.empno,e1.name,&e1.salary);
    size=sizeof(e1);
    printf("\n No. of bytes required for e1=%d \n\n",size);
    e2=e1;
    printf("After assigning e1 to e2 \n\n");
    printf("e2.empno=%d \n",e2.empno);
    printf("e2.name=%s \n",e2.name);
    printf("e2.salry=%8.2f \n\n",e2.salary);
    printf("Address of l= %u \n",&e1);
    getch();
}

```

Sample Program Output

Enter empno, name and salary
123 Nishu 3456

No. of bytes required for e1=26

After assigning e1 to e2

e2.empno=123
e2.name=Nishu
e2.salry=3456.00

Address of e1=65498

e1 and **e2** are declared to be variables of **struct** emp type. Both can accommodate details of an employee. Details of an employee are accepted into the variable **e1**. The number of bytes occupied by a variable of **struct** emp type is found out with the help of the operator `sizeof()` by passing **e1** to it. The value returned by `sizeof()` is then displayed. To illustrate the fact that structure variables assignment is permissible, **e1** is assigned to **e2**. The contents of **e2** are then displayed. The address of operator `&` is used with **e1** to obtain its address and it is then displayed.

4.6 OPERATIONS ON STRUCUTRES

The number of operations which can be performed over structures is limited. Following are the permissible operations:

1. Accessing the individual members of a structure variable with the help of members operator (using dot operator)

EXAMPLE

In the case of a variable of type struct emp,

```
struct emp e;  
e.empno=10;
```

10 are assigned to empno number of e.

```
strcpy (e.name,"Kavi");
```

The string "Kavi" is copied to name number of e.

2. Assigning one structure variable to another of the same type.

```
struct emp e1={12,"Kamal",4000},e1  
e2=e1;
```

e1 has been assigned to e2.

3. Retrieving the size of a structure variable using sizeof() operator.

```
struct emp e  
int e;  
s=sizeof(e);
```

4. Retrieving the address of a structure variable using & (address of) operator.

```
struct emp e;
```

5. Passing and returning a structure variable value to and from a function.

6. Checking whether two structure variables of same type are equal using ==. If s1 and s2 are two variables of the same structure type, s1==s2 returns 1 and if all the members of s1 are equal to the corresponding members of s2, it returns 0.

7. Checking whether two structure variables of same type are not equal using !=. If s1 and s2 are two variables of the same structure type, s1!=s2 returns 1 and if all the members of s1 are not equal to the corresponding members of s2, it returns 0.

EXAMPLE

Write a program to perform operation of structure

```
#include<stdio.h>
struct classroom
{
    char name[15];
    long int reg_no;
}s1;
void main()
{
    struct classroom s1={"sudhakar","5690823"};
    struct classroom s2={"ragul"};
    printf("Name is %s",s1.name);
    printf("\n Reg no is %d",s1.reg_no);
    printf("Name is %s",s2.name);
    printf("\n Reg no is %d",s2.reg_no);

    s3=s1;

    printf("Name is %s",s3.name);
    printf("\n Reg no is %d",s3.reg_no);
}
```

Sample program output

Name is kavitha
Reg no is 5690823

Name is kamal
Reg no is 5690853

Name is martin
Reg no is 5690821

The values to be initialized of the members of a structure must be enclosed within a pair of braces.

NOTE

The constants to be defined to the members of the structure must be in the same order in which the members are declared in the structure. The information contained in one structure variable can also be assigned to another structure variable using a single assignment statement.

s3=s1

to copy of the values in the structure variable s1 to s3. This type of assignment statements will be necessary when many members of the structure are to be declared. The structure variable can be assigned to another only when they both are of the same structure type.

4.7 ARRAYS AND STRUCTURES

4.7.1 Arrays of Structures

An array of structures is simply an array in which each element is a structure of the same type.

```
struct emp e[10];
```

Hence the array elements e[0], e[1]...e[9] are variables of **struct** emp type and thus each can accommodate an employee's details. Since all the variables share a common name e and are distinguishable by subscript values [0-9], collective manipulation over the structure elements becomes easy.

EXAMPLE

Write a program to search for an employee in a list of employees

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int empno;
    char name[20];
    float salary;
};
void main()
{
    struct emp e[10];
    int i,n,flag;
    char sname[20];
    float f,*fp;
    fp=*f;
    clrscr();
    printf("Enter the no. of employees \n");
    scanf("%d",&n);
    printf("Enter %d employees details \n",n);
    scanf("%d %s %f",&e[i].empno,e[i].name,&e[i].salary);
    printf("Enter name of the employee to be searched \n");
    scanf("%s",&name);
    printf("%d employees details \n",n);
    for(i=0;i<n;i++)
```

```

printf("\n %5d %10s %10.2f \n",e[i].empno,e[i].name,e[i].salary);
printf("Name of the employee to be searched \n");
printf("\n %10s \n",sname);

flag=0;
for(i=0;i<n;i++)
if(strcmp(e[i].name,sname)==0)
{
    flag=1;
    break;
}
if(flag==1)
    printf("found");
else
    printf("not found");
getch();
}

```

4.7.2 Arrays within Structures

The array was treated in its entirety. The array name was used to refer to the entire sequence of characters forming a name. An array of **int** type as a member of structure, where we need to deal with each integer value of the array. Suppose we need to maintain a list of students' details (Reg-no, Name, Marks in five subjects)

The structure template definition will be:

```

struct student
{
    int regno;
    char name[20];
    int marks [5];
}

```

EXAMPLE

Write a program to create a list of student's details and display them

```

#include<stdio.h>
#include<conio.h>
struct student
{
    int reg_no;
    char name[20];
    int marks[5];
    int total;
    float percent;
}

```

```

};
void main()
{
    struct student s[10];
    int i,n,j;
    clrscr();
    printf("Enter no. of students \n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter reg_no,name of student -%d \n",i+1);
        scanf("%d",&n);
        for(i=0;i<n;i++)
            scanf("%d",&s[i].marks[j]);
    }
    for(i=0;i<n;i++)
    {
        s[i].total=0;
        for(j=0;j<5;j++)
            s[i].total+=s[i].marks[j];
        s[i].percent=(float)s[i].total/5;
    }
    printf("\n Reg-no Name percentage \n");
    for(i=0;i<n;i++)
        printf("%6d %15s %7.2f \n",s[i].reg_no,s[i].name,s[i].percent);
    getch();
}

```

4.8 UNION

Unions like structure contain members whose individual data types may differ from one another. However the members that compose a union all share the same storage area within the computer's memory where as each member within a structure is assigned its own unique storage area. Thus unions are used to observe memory. They are useful for application involving multiple members. Values need not be assigned to all the members at one time. Like structures union can be declared using the keyword union as follows:

Before instantiating variables of some union type, the data items which are to share a common name should be grouped together. This is done with the help of union template.

SYNTAX

```
union tag_name
{
    data-type member1;
    data-type memembr2;
    :
    :
    data-type member n;
}
```

Where

union->is a keyword.

tag-name->is any user-defined name, which should be valid C identifier.

data-type->is any valid data type supported by C or user-defined type.

member1,member2...member->are the members of the union

The syntax of declaring a variable of union type is:

```
union tag_name variable_name;
```

A memory location gets allocated, the size of which is equal to that of the largest of the members member1,member2,member3,...member n. Accessing the members of a union is similar to accessing the members of a struct. Dot operator is used to access each individual member. Dot operator expects union variable to its left and member name to its right.

EXAMPLE

```
union item
{
    int m;
    float p;
    char c;
}
code;
```

union item makes a group of three data items of type **int**, **float** and **char**.

```
union item t;
```

A variable **t** declared to be of type **union** temp. As a result of this; only one memory location gets allocated. It can be referred to by any one individual member at any point of time. The size of the memory location is four bytes, which happens to be the size of the largest sized data type float in the member list.

EXAMPLE

Write a program to demonstrate initialization of an union

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int roll_no,mark1,mark2,mark3;
}ul;
void main()
{
    ul.roll_no=459;
    ul.mark1=87;
    printf("Roll Number:%d \n",ul.roll_no);
    printf("Mark 1: %d \n",ul.mark1);
    printf("Mark 2:%d\n",ul.mark2);
    printf("Mark 3:%d\n",ul.mark3);
}
```

Sample program output

Roll No:459
Mark 1:87
Mark 2:87
Mark 3:87

Thus the program, even the uninitialized members (that is mark2 and mark3) take the same value as the other members.

4.9 DIFFERENCE BETWEEN STRUCTURE AND UNION

1. Union allocates the memory equal to the maximum memory required by the member of the union but structure allocates the memory equal to the total memory required by the members.
2. In union, one block is used by all the member of the union but in case of structure, each member has their own memory space.
3. Union is best in the environment where memory is less as it shares the memory allocated. But structure cannot implement in shared memory.
4. As memory is shared, ambiguities are more in union, but less in structure.
5. Self referential union cannot be implemented in any data structure, but self referential structure can be implemented.

Check your progress

1. Define Structure?

.....
.....
.....

2. Define Union?

.....
.....
.....

4.10 LET US SUM UP

- A structure is a derived type usually representing a collection of variables of same or different data types grouped together under a single name
- The keyword struct begins the structure declaration followed by the structure name, the structure members are enclosed in braces, followed by the semicolon which ends the structure declaration.
- The variable or data items in a structure are called as members of the structure.
- Usually the structure type declaration appears at the top of the source code file, before any variables or functions are defined.
- Structure variables may be initialized with a structure variable of the same type.
- A structure that contains another structure as its member is called as nesting of structures.
- A union is another compound data type like a structure that may hold the variables of different types and sizes, with the compiler keeping track of the size.
- The keyword union begins the union declaration followed by the tag (i.e., union name), within the braces, union members are declared.
- A union may contain many members of different types, but the memory space reserved for a union is large enough to store its largest number.

4.11 LET US SUM UP

1. State whether the following statements are true or false
 - A. A **struct** type in C is a build-in data type.
 - B. The tag name of a structure is optional.
 - C. Structures may contain members of only one data type.
 - D. A Structure cannot have a union as one of its members
 - E. A member in a structure can itself be a structure.
2. Fill in the blanks in the following statements:
 - a. The name of a structure is referred to as _____.
 - B. A _____ is a collection of data items under one name in which the items share the Same storage.
3. State which of the following declarations are invalid? Why?
 - A. struct abc v1;
 - B. struct abc v2 [10];
 - C. struct ABC v3;
 - D. ABC a, b, c
 - E. ABC a [10];
4. How does a structure differ from an array?
5. Explain the meaning and purpose of the following:
 - a. **struct** keyword
 - b. **sizeof** operator
 - c. **tag name**

4.12 KEYWORDS

Attributes: Each entity in the world is described by a number of characteristics. These descriptive characteristics of an entity are called **attributes**.

Structure: A structure is a derived type usually representing a collection of variables of same or different data types grouped together under a single name.

Tag: A user defined structure name usually referred to as **tag**.

Nested Structures: Nested structures are nothing but a structure with another structure is called nested structures.

4.13 QUESTIONS FOR DISCUSSION

1. Explain the need for the concept of structure.
2. Explain syntax of declaring a structure variable.
3. Can we initialize structure variables while they are declared? If yes, explain the syntax with an example.
4. Differentiate between structure and union.
5. Write a program to create a 'list of books' details. The details of a book include Title, author, and Publishing year, Number of pages and Price.
6. Explain union with an example, and state the uses of union.
7. Explain the concept of initialization of variables in a union.

Check your Progress: Model Answers

Ans.1

Structures are collection of unlike data types just as arrays are collection of like data types. A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. The individual structure elements are referred to as members.

Ans.2

A union is another compound data type like a structure that may hold objects of different types and sizes, with the compiler keeping track of the size. Unions provide a way to manipulate different kinds of data in a single area of storage.

4.14 SUGGESTED READING

1. Turbo C/C++ - The Complete Reference - H.Schidt
2. Programming in C - S.Kochan
4. Born to code in C - H.Schidt
5. The Art of C - H.Schidt
6. C Programming - Keringhan and Ritchie - 2nd Ed.
7. Programming in ANSI C - Agarwal
8. Let us C - Kanitkar
9. Programming in ANSI C - Balguruswamy

UNIT-5

UNIT

5

FILE MANAGEMENT IN C

CONTENTS

- 5.1 Aims and Objectives
- 5.2 Introduction
- 5.3 Operations on Files
- 5.4 Opening and Closing of files
- 5.5 File I/O Functions
 - 5.1 Character Oriented Functions
 - 5.2 String Oriented Functions
 - 5.3 Mixed Data Oriented Functions
 - 5.4 Unformatted Record I/O Functions
- 5.6 Random Accessing of Files
- 5.7 Error Handling During File I/O Operations
- 5.8 Command Line Arguments
- 5.9 Let Us Sum Up
- 5.10 Lesson end Activities
- 5.11 Keywords
- 5.12 Questions for Discussion
- 5.13 Suggested Readings

5.1 AIMS AND OBJECTIVES

At the end of this chapter, you will learn how to:

- Discuss different operations of files
- Understanding different I/O functions
- Discuss about different operations of files

5.2 INTRODUCTION

The data was written to the standard output and data was read from the standard input. As long as only small amount of data are being accessed in the form of simple variables, and character strings this type of I/O is sufficient. However, with large amounts of data. The information has to be written to or read from auxiliary storage device. Such information stored on the device in the form of a data file.

There are two different categories of data files

- Stream oriented data files
- System oriented data files

Stream oriented data files are of two types. In the first category, the data files comprised of consecutive characters. These characters can be interpreted as individual data items or as components of strings of numbers. These are called text files.

The second category of stream-oriented data files, often referred to as unformatted data files, organizes data into blocks containing contiguous bytes of information. These blocks represent more complex data structures, such as arrays and structures. These file are called binary files.

System oriented data files are more closely related to the computer's operating system than are stream oriented data files. To perform the I/O from and to files, an extensive set of library functions are available in C. Access to files generally requires four basic operations.

Access to files generally requires four basic operations:

Open: This allows access to a file and establishes the position, offset, in the file.

Close: This ends access to the file. When access to a file completes, it should be closed. The number of files that a running program can have any time is limited; by closing file properly these limited facilities can be used more intelligently.

Read: This gets information from the file, either in the form of characters strings, or in the form of data (combined integers, characters, floating point numbers, and structures).

Write: This adds information to the file or replaces information already in the file.

5.3 OPERATIONS ON FILES

The commonly performed operations over files are the following:

1. Opening a file
2. Reading from a file
3. Writing a file
4. Appending to a file
5. Updating a file
6. Deleting a file
7. Renaming a file
8. Closing a file

These operations are accomplished by means of standard library functions that are provided by C.

5.4 OPENING AND CLOSING OF FILES

5.5 `fopen ()`

The `fopen ()` is to open a file. Opening a file basically establishes a link between a program and the file being opened.

SYNTAX

`fp=fopen("filename", "mode of opening");`

where,

filename->is the name of the file being opened(which is remembered by the operating system).Mode of opening can be any one of the following.

Mode of opening	Purpose
w	To create a text file. If the file already exists, its contents are destroyed; otherwise it is created, if possible.
r	To open a text file for reading; the file must exist.
a	To open a text file for appending (writing at the end); if the file does not exist, it is created, if possible.

w+	To create a text file both reading and writing; if the file already exists, its contents are destroyed; otherwise it is created, if possible.
r+	To open a text file for both reading and writing; the file must exist.
a+	To open a text file for both reading and appending; if the file already exists, its contents are retained; if the file does not exist, it is created, if possible.
wb	To create a file. If the file already exists, its contents are destroyed; otherwise it is created, if possible.
rb	To open a binary file for reading; the file must exist
ab	To open a binary file for both reading and writing; if the file already exists, its contents are destroyed; otherwise it is created, if possible.
wb+	To create a binary file both reading and writing; if the file already exists, its contents are destroyed; otherwise it is created, if possible.
rb+	To open a binary file for both reading and writing; the file must exist.
ab+	To open a binary file for reading and appending; if the file already exists, its contents are retained; if the file does not exist, it is created if possible.

(ii) fclose ()

The fclose () is the counterpart of fopen (). This is used to close a file. Closing a file name means de-linking the file from the program and saving the contents of the file.

SYNTAX

```
fclose (fp);
```

where

fp->is the pointer to FILE type and represents a file. The file represented by fp is closed.

5.5 FILE I/O FUNCTIONS

When a file opened, we can read data stored in the file or write new data onto it depending on the mode of opening standard library supports a good number of functions which can be used for performing I/O operations. These functions are referred to as file I/O functions.

File I/O functions are broadly classified into two types:

1. High level files I/O functions
2. Low level files I/O functions

high level file I/O functions are basically C standard library functions and are easy to use. Most of the C programs handling files use these because of their simple nature. Low level file I/O functions are file related system calls of the underlying operating system. These are relatively more complex in nature when compared to high level file I/O functions but efficient in nature.

High level file I/O functions can be further classified into the following two types:

1. Unformatted file I/O functions
2. Low level files I/O functions

Unformatted file I/O functions

- `fputc()` and `fgetc()`-Character-oriented file I/O functions
- `fputs()` and `fgets()`-String-oriented file I/O functions

Formatted file I/O functions

`fprint()` and `fscanf()`-Mixed data-oriented file I/O functions

5.5.1 Character Oriented Functions-`fputc()`,`fgetc()`

`fputc()` is to write a character onto a file.

SYNTAX

```
fputc(c,fp);
```

where

`c->` represents a character and `fp`, a pointer to `FILE`, represents a file. The function writes the content of `c` onto the file represented by `fp`.

fgetc() is to read a character from a file. The syntax of its usage is as follows:

```
c=fgetc (fp);
```

c is a variable of **char** type and **fp** is a pointer to **FILE** type. The function reads a character from the file denoted by **fp** and returns the character value, which is collected by the variable **c**.

EXAMPLE

Write a program to create a file consisting of characters

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char c;
    clrscr();
    fp=fopen("text","w");
    printf("Keep typing characters, Type 'q' to terminate \n");
    c=getchar();
    while(c!='q')
    {
        fputc(c,fp);
        c=getchar();
    }
    fclose(fp);
}
```

Sample program output

Keep typing characters. Type q to terminate
Akdiekldld

here **fp** is declared to be a pointer variable to **FILE** type and **c** is declared to be a variable of **char** type. The file pointer variable is to represent the file to be created by the program and the variable **c** of char type is to collect characters one at a time, entered through the standard input device, keyboard. Note that the external file "text" is opened in "w" mode.

5.5.2 String Oriented Functions-fputs(),fgets()

The fputc () is to write a string onto a file.

SYNTAX

```
fputs (buffer,fp);
```

buffer is the name of a character array, **size** is an integer value, **fp** is a pointer to **FILE** type. The function reads a string of maximum size-1 characters from the file pointed to by **fp** and copies it to the memory area denoted by **buffer**.

EXAMPLE

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char name[20];
    clrscr();
    printf("Strings are \n");
    fp=fopen("name.dat","r");
    while(!feof(fp))
    {
        fgets(name,80,fp);
        puts(name);
        printf("\n");
    }
    fclose(fp);
}
```

Sample program output

Strings are

Nisha
Devi
Kumar
Ravichandran

The variable **fp** is declared to be a pointer to **FILE** type. **name** is declared to be an array of **char** type **and** of size 20. The file variable **fp** is to denote the file names.dat to be read by the program; the string variable **name** is to collect a string read from the file.

5.5.3 Mixed Data Oriented Functions-fprintf(),fscanf()

fprintf() is to write multiple data items which may or may not be of different types to a file.

SYNTAX

```
fprintf(fp,"control string", arguments-list);
```

fprintf() is similar to that of printf() except the presence of an extra parameter **fp**, a pointer to **FILE** type. The parameter **fp** represents the file to which data are to be written.

fscanf() is similar to that of scanf() except the presence of an extra parameter fp. A pointer to **FILE** type. The parameter fp represents the file from which data are to be read.

EXAMPLE

Write a program to read a file consisting of employees details

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int empno;
    char name[20];
    float salary;
};
void main()
{
    FILE *fp;
    struct emp e;
    fp=fopen("emp.dat","r");
    while(!feof(fp))
    {
        fscanf(fp,"%d %s %f",&e.empno,&e.name,&e.salary);
        printf("%06d %15s %7.2f\n",e.empno,e.name,e.salary);
    }
    fclose(fp);
}
```

The structure **struct emp** is defined with the fields **empno, name and salary**. In the main () variable **fp** is declared to be a pointer to **FILE** type and it is to denote the file emp.dat to be created by the program. **e** is declared to be a variable of **struct emp** type, which is to collect the employee details accepted through the keyboard.

5.5.4 Unformatted Record I/O Functions-fwrite (), fread ()

The fwrite () is used to write blocks of data (records) to a file. The contents which are written to the secondary devices are nothing but the exact copy of them in memory. The files of this kind are called **binary files**. But the fprintf(), another file output function, formats the memory contents according to the format specifies passed to it and then write them onto the secondary storage device.

fwrite(buffer_address,size,count,file_pointer);

here,buffer_address is the address of the memory area, the contents of which are to be written to the file denoted by the fourth arguments file_pointer. The second argument **size** specifies the number of bytes of a block(record) and count specifies the number of blocks of data written to the file.

EXAMPLE

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int empno;
    char name[20];
    float salary;
};
void main()
{
    struct emp e;
    FILE *p;
    int i,n;
    clrscr();
    fp=fopen("emp.data","wb");
    printf("Enter the number of employees \n");
    scanf("%d",&n);
    printf("Enter empno,name and salary of %d employees \n",n);
    for(i=1;i<=n;i++)
    {
        scanf("%d %s %f",&e.empno,&e.name,&e.salary);
        fwrite(&e,sizeof(e),1,fp);
    }
    fclose(fp);
    getch();
}
```

Sample program output

Enter the No. of Employees

2

Enter empno,name and salary of 2 employees

120 kirupa 83832

378 haish 36374

This program is similar t

5.6 RANDOM ACCESSING OF FILES-fseek(),ftell(),rewind()

Random Access files consist of records that can be accessed in any sequence. This means the data is stored exactly as it appears in memory, thus saving processing time (because no translation is necessary) both in when the file is written and in when it is read. C standard library provides the following build-in functions to support this:

(i)fseek()

The position fssek() repositions the file pointer.

SYNTAX

fseek(fp,offset,position)

where

fp->is a pointer to **FILE** representing a file

offset->is the number of bytes by which the file pointer is to be moved relative to the byte number identified by the third parameter **position**.

position->position can take any one of the following three values 0,1 and 2

Position	Symbolic Constants	Meaning
0	SEEK_SET	Beginning of File
1	SEEK_CUR	Current position of File
2	SEEK_END	End of file

(ii)ftell()

The function ftell() returns the current position of the file pointer

SYNTAX

position=ftell(fp);

where **fp** is a pointer to **FILE** type representing a file, the current position of the file pointer of the file is returned by the function.

(iii)rewind()

The rewind() positions the file pointer to the beginning of a file.

SYNTAX

```
rewind(fp);
```

EXAMPLE

Write a program to count the number of records in employee file

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    int empno;
    char name[20];
    float salary;
};
void main()
{
    FILE *fp;
    int nor,last_byte;
    clrscr();
    fp=fopen("emp_dat","r");
    fseek(fp,0,SEEK_END);
    nor=last_byte/sizeof(struct emp);
    printf("No. of records=%d",nor);
    fclose(fp);
    getch();
}
```

Sample program output

No. of records=2

The structure **struct emp** is defined with the fields **empno,name and salary**. In the main(),variables **fp** is declared to be a pointer to **FILE** type. The integer variables **nor** and

last_byte are used to collect the number of records in the file **emp.dat** and the last byte number of file respectively.

5.7 ERROR HANDLING DURING FILE I/O OPERATIONS

The file I/O operations cannot always be expected to be smooth sailing. During the course of I/O operations, some errors may be encountered. As a consequence of the error conditions, the underlying program may prematurely terminate or it may produce erroneous results.

Following the circumstances under which the file I/O operations fail:

1. Trying to open a file, this does not exist, for reading purpose.
2. Trying to open a file for writing purpose when there is no disk space.
3. Trying to write to a read-only file.
4. Trying to perform an operation over a file when the file has been opened for some other purpose.
5. Trying to read a file beyond its end-of-mark.

When we try to open a file, which does not exist, for reading purpose, the `fopen()` returns `NULL`.

EXAMPLE

```
fp=fopen("student.dat","r")
```

In case, `student.dat` does not exist, the function `fopen()` returns `NULL` value, which is collected by the variable **fp**. The segment is used to handle this error situation.

```
if(fp==NULL)
{
    printf("This file does not exist");
    exit(1);
}
```

when an attempt to open a file for writing purpose fails.

Write a program errors handling during file I/O operations

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
    FILE *fp;
    clrscr();
    fp=fopen("books.dat","r");
    if(fp==NULL)
    {
        printf("The file books.dat does not exist");
        getch();
    }
    fp=fopen("students.dat","w");
    if(fp==NULL)
    {
        printf("The read-only file students.dat cannot be opened in write mode");
        getch();
    }
    fp=fopen("text.dat","r");
    fputc('a',fp);
    if(ferror(fp))
    {
        printf("The file text.dat has been opened in read mode");
        printf("but you are writing to it. \n");
        getch();
    }
    getch();
}
```

Sample program output

The file **Books.dat** does not exist

The read-only file **students.dat** cannot opened in write mode

The file **text.dat** has been opened in read mode but you are writing to it

To open the file **books.dat**, this does not exist. The statement `fp=fopen("books.dat","r");` is executed, the file variable **fp** collects the NULL value. The error is then trapped and the message "the file books.dat does not exist" is displayed.

Secondly, an attempt is made to open the file **students.dat**, a read-only file, in write mode, since opening of the file fails, once again when the statement

`fp=fopen("student.dat","w");`

is executed, the file variable **fp** would not be NULL value.

5.8 COMMAND LINE ARGUMENTS

In C it is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system. To establish the data communication between a calling function and a called function. It is done through arguments; a calling function passes inputs to a called function, which perform required manipulations.

To display the contents of emp.dat, we use the following command:

`C:\>type emp.dat`

here **type** is the program file name (executable) and **emp.dat** is the input file, the contents of which are displayed

To make `main()` of a program take command line arguments, the function header will have the following form:

`void main(int argc, char *argv[])`

Here, **argc** and **argv []** are the formal arguments, which provide mechanism for collecting the arguments given at command line when the program is launched for execution.

EXAMPLE

To write a program using command line arguments-Copying one file to another file.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main(int argc, char *argv[])
{
    FILE *fin,*fout;
```

```

char c;
clrscr();
if(argc!=3)
{
    printf("Invalid number of arguments");
    exit(1);
}
fin=fopen(argv[1],"r");
fout=fopen(argv[2],"w");
while(!feof(fin))
{
    c=fgetc(fin);
    fputc(c,fout);
}
fclose(fin);
fclose(fout);
getch();
}

```

Sample program output

The program is executed as follows:

C:\>copy text.dat text1.dat

As a result, the contents of text.dat are copied to text1.dat. we have thus simulated DOS copy command.

The purpose of the program is to copy the contents of the file text.dat (source file) to the file text1.dat(target file).The source file and the target files are to be passed as the arguments to the main() itself.So,the main() is defined with two arguments **argc**(int) and **argv[]**(char*).The argument argc collects the number of arguments passed to the main() while it is launched for execution. In this case, the value of **argc** would be three. They are: the program file name (executable) (**argv [0]**), source file (**argv [1]**) and the target file (**argv [2]**).

Check your progress

1. What is a file? Why do we need the concept of files?

.....

2. Give the purpose and syntax of fopen() and fclose().

.....

5.9 LET US SUM UP

- Storage of information is read from or written on a auxiliary memory device is stored in the form of a file.
- The data structure of a file is defined in `stdio.h` which creates a buffer area to store data in a file for reading as well as writing.
- Function `fopen()` is used to open a specified file and `fclose()` is used to close a specified file.
- Functions `fscanf()` and `fprintf()` are used to perform input/output operation in files.
- Functions `fgetc()` and `fputc()` are used to perform character input/output operation in files.
- Functions `fgets()` and `fputs()` are used to perform string input/output operation in files.
- Functions `fseek()` is used to index a file and can be used to increment or decrement the file pointer.
- Function `rewind()` is used to reset the file pointer to the beginning of the file.

5.10 LESSONS END ACTIVITIES

1. State whether the following statements are true or false:
 - A. A file must be opened before it can be used.
 - B. All files must be explicitly closed.
 - C. Files are always referred by name in C programs.
 - D. Using **fseek** to position a file beyond the end of the file is an error.
 - E. Function **fseek** may be used to seek from the beginning of the file only.
2. Fill in the blanks in the following statements.
 - a. The mode_____is used for opening a file for updating.
 - b. The function_____may be used to position a file at the beginning.
 - c. The function_____gives the current position in the file.
 - d. The function_____is used to write data randomly to accessed file.

3. What is the significance of EOF?
4. Distinguish between the following functions:
 - (a) getc() and getchar()
 - (b) print() and fprintf()
5. Explain the general format of fseek function?

5.11 KEYWORDS

FILE: The header file stdio.h defines a new data type called **FILE**.

STDIO.H: Each file used in a program must be associated with a pointer of its type. Three types of file pointers are defined in stdio.h. They are stdin, stdout and stderr.

PRINTF () AND SCANF (): printf () and scanf () are used for performing input/output operations in programs (without involving files).

FPRINTF () and FSCANF (): fprintf () and fscanf () are used to perform input/output operation in files.

FGETC () and FPUTC (): Single character input/output from files are fgetc() and fputc().

FGETCHAR () and FPUTCHAR (): Character input/output functions which act on files.

END OF FILE: Most of the programs use EOF when reading input from the user or when displaying the output data.

FEOF(): The feof() function is used to check the end of file condition.

5.12 QUESTIONS FOR DISCUSSION

1. What are standard files that are accessed when a program begins its execution?
2. Explain the functions fopen() and fclose() and with clearly stating its syntax?
3. What are the various file-opening modes in C?
4. Explain the concept of characters input/output in files?
5. Explain the functions fgetc() and fputc() with clearly stating its syntax?
6. Explain the use of rewind() function in files?
7. Explain the concept of characters input/output files.
8. Explain the concept of string input/output in files.
9. Write a program to count the number of characters in a text file.

Check your Progress: Model Answers

Ans.1

A file is defined as a collection of related data stored on secondary storage device like disk. It is a named storage on secondary storage devices. The concept of files enables us to store large amount of data permanently on the secondary storage devices. The ability to store large amount of data and the ability to store them permanently are attributed to the physical characteristics of the devices.

Ans.2

The commonly performed operations over files are the following:

- | | | |
|----------------------|------------------------|--------------------|
| 1. Opening a file | 2. Reading from a file | 3. Writing a file |
| 4. Appending to file | 5. Updating a file | 6. Deleting a file |
| 7. Renaming a file | 8. Closing a file | |

5.13 SUGGESTED READING

1. Let us C-Yashwant Kanetkar.
2. Programming in C- Balguruswamy
3. The C programming Lang., Pearson Ecl – Dennis Ritchie
4. Structured programming approach using C-Forouzah & Ceilberg Thomson learning publication.
5. Pointers in C – Yashwant Kanetkar
6. How to solve it by Computer – R. G. Dromy
7. Introduction to algorithms – Cormen, Leiserson, Rivest, Stein
<http://www.cs.utexas.edu/users/rpriece>
8. Peter Norton's Introduction to Computers – Tata MGHill