According to NEW syllabus of GTU from 2018 onwards

GTU

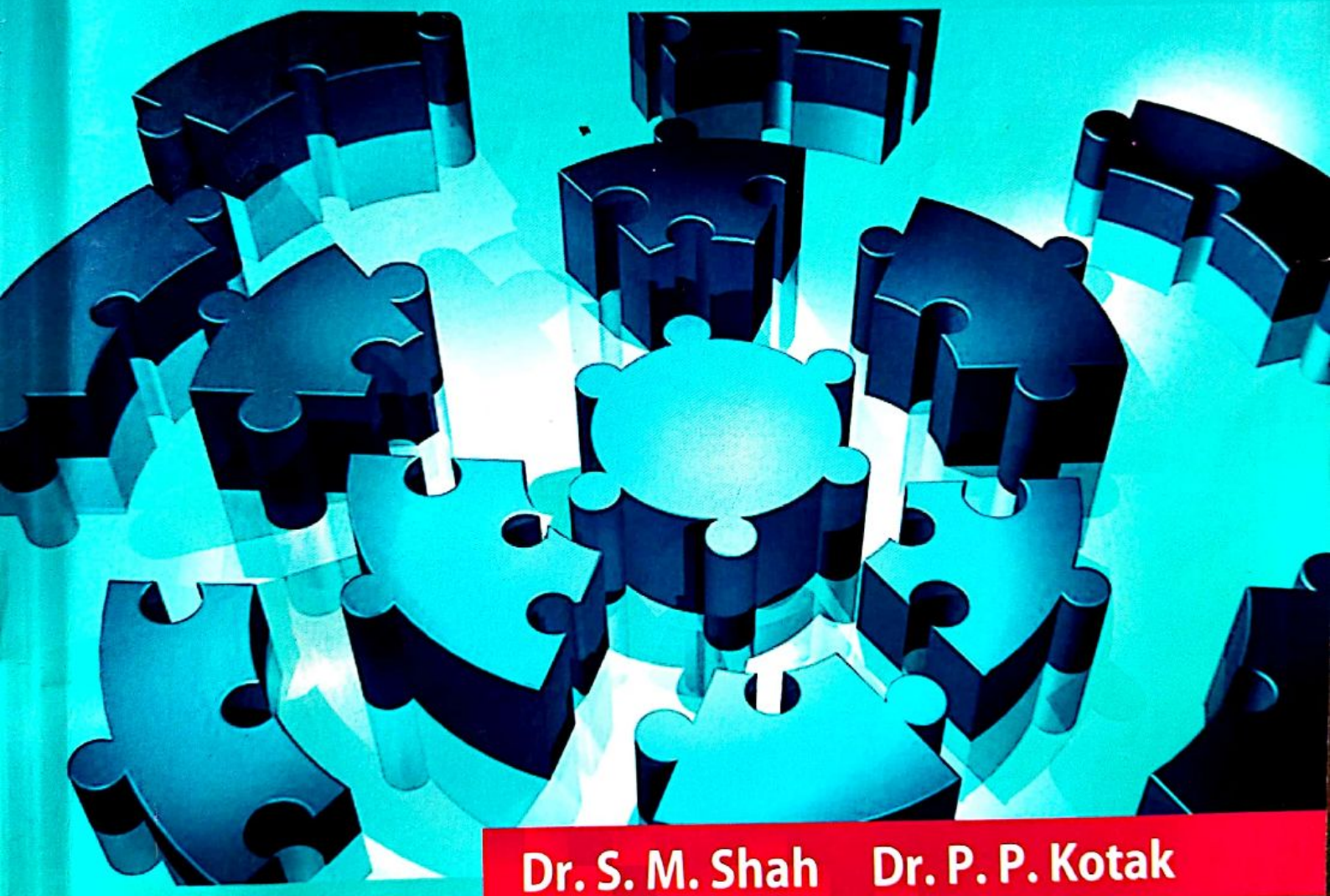
Includes Previous Years GTU paper's solution

2nd Edition

Programming For Problem Solving

B.E. 1st Year

Subject Code : 3110003



Dr. S. M. Shah Dr. P. P. Kotak

Mahajan Publishing House®

According to the syllabus of
Gujarat Technological University (GTU)

Programming for Problem Solving

: Authors :

Dr. S. M. Shah
B.E. (Comp.), M.E. (Comp.)
Prof., Computer Engineering Dept.
Government Engineering College,
Modasa

Dr. P. P. Kotak
B.E. (Comp.), M.E.(Comp.)
Principal,
Government Polytechnic
Rajkot.

MAHAJAN Mahajan Publishing House®
AHMEDABAD

Published by :

Shri Nirav J. Shah
Mahajan Publishing House
Gandhi Road,
Ahmedabad-380001.
Phone : 079-22111123

Price : **₹ 300/–**

Edition : Second Edition (2019-20)

ISBN 978-93-87096-32-5

Type Setting by :

C. J. Patel
106, Karyashilp Tower,
B/h. RTO, Subhashbridge
Ahmedabad

Dedicated to

my Wife, Son, Mother & late Father

Dr. S. M. Shah

Dedicated to

my Family members

Dr. P. P. Kotak

Preface to the Second Edition

We are very happy to present the first edition of the book on Programming for Problem Solving as per syllabus of Gujarat Technological University. The syllabus content has been divided in two parts. First 2 chapters are related with computer fundamentals and basic concept of programming like flowchart and algorithm. Chapter 3 to 13 presents detailed concepts of 'C' Programming language. The book not only contains theory but also contains well structured flow of programs covering different aspects of theory to solve wide range of problems. All programs have sample run with their output. Emphasis has been put on how programming can be used to solve different types of problems. Programs have been well commented to increase the understating of the students. This book is written to meet the needs of beginners as well as advanced learners. The material has been presented in a simple language. At the end of every chapter summary, MCQs, exercises and answers to selected exercise is given.

In this edition of book, new content is added in Chapter 2 and in Chapter 7. We are thankful to all persons, who have directly or indirectly helped us in writing this book. We are also thankful to Nirav Shah of Mahajan Publishing House for giving us an opportunity for writing this book. Suggestions from teaching community are welcome to improve the quality of the book.

—Dr. S. M. Shah

Dr. P. P. Kotak

Preface to the First Edition

We are very happy to present the first edition of the book on **Programming for Problem Solving** as per syllabus of Gujarat Technological University. The syllabus content has been divided in two parts. First 2 chapters are related with computer fundamentals and basic concept of programming like flowchart and algorithm. Chapter 3 to 13 presents detailed concepts of **'C' Programming** language. The book not only contains theory but also contains well structured flow of programs covering different aspects of theory to solve wide range of problems. All programs have sample run with their output. Emphasis has been put on how programming can be used to solve different types of problems. Programs have been well commented to increase the understating of the students. This book is written to meet the needs of beginners as well as advanced learners. The material has been presented in a simple language. At the end of every chapter summary, MCQs, exercise and answers to selected exercise is given.

We are thankful to all persons, who have directly or indirectly helped us in writing this book. We are also thankful to Nirav Shah of Mahajan Publishing House for giving us an opportunity for writing this book. Suggestions from teaching community are welcome to improve the quality of the book.

Dr. S. M. Shah
Dr. P. P. Kotak

Syllabus..

Teaching and Examination Scheme :

Teaching Scheme			Credits	Examination Marks				Total Marks
				Theory Marks		Practical Marks		
L	T	P	C	ESE (E)	PA (M)	ESE (V)	PA (I)	
3	0	2	4	70	30	30	20	150

Content :

Sr. No.	Topics Teaching	Weightage
1	**Introduction to computer and programming:** Introduction, Basic block diagram and functions of various components of computer, Concepts of Hardware and software, Types of software, Compiler and interpreter, Concepts of Machine level, Assembly level and high level programming, Flowcharts and Algorithms	11
2	**Fundamentals of C:** Features of C language, structure of C Program, comments, header files, data types, constants and variables, operators, expressions, evaluation of expressions, type conversion, precedence and associativity, I/O functions	9
3	**Control structure in C:** Simple statements, Decision making statements, Looping statements, Nesting of control structures, break and continue, go to statement	11
4	**Array & String:** Concepts of array, one and two dimensional arrays, declaration and initialization of arrays, string, string storage, Built-in string functions	13
5	**Functions:** Concepts of user defined functions, prototypes, definition of function, parameters, parameter passing, calling a function, recursive function, Macros, Pre-processing	11
6	**Recursion:** Recursion, as a different way of solving problems. Example programs, such as Finding Factorial, Fibonacci series, Ackerman function etc. Quick sort or Merge sort.	9
7	**Pointers:** Basics of pointers, pointer to pointer, pointer and array, pointer to array, array to pointer, function returning pointer	9
8	**Structure:** Basics of structure, structure members, accessing structure members, nested structures, array of structures, structure and functions, structures and pointers	9
9	**Dynamic memory allocation:** Introduction to Dynamic memory allocation, malloc, calloc	9
10	**File management:** Introduction to file management and its functions	9

Contents

Chapter-1 : Introduction to Computer & Programming 1.1 – 1.20

1.1 Introduction, 1.2
1.2 Computer, 1.2
1.3 Computer system characteristics or functions, 1.3
1.4 Classification of computer system, 1.3
1.5 Application of computer, 1.3
1.6 Basic block diagram and functions of computer components, 1.4
1.7 Limitation of computer, 1.6
1.8 Computer software, 1.6
1.9 Computer hardware, 1.8
1.10 Programming languages, 1.9
1.11 Computer generation, 1.11
1.12 Differences, 1.11
❖ Summary, 1.13
❖ MCQs, 1.13
❖ Exercises and Answers to selected Exercises, 1.16
❖ Short Questions, 1.20

Chapter-2 : Flowcharts & Algorithms 2.1 – 2.19

2.1 Introduction, 2.2
2.2 Problem solving techniques, 2.2
2.3 Flowchart, 2.2
2.4 Flowchart examples, 2.3
2.5 Algorithm, 2.11
2.6 Pseudo code, 2.12
2.7 Algorithms examples, 2.12
❖ Summary, 2.16
❖ MCQs, 2.17
❖ Exercises and Answers to selected Exercises, 2.17
❖ Short Questions, 2.19

Chapter-3 : Introduction to 'C' Language 3.1 – 3.14

3.1 Introduction, 3.2
3.2 Classification of computer language, 3.2
3.3 Feature of 'C', 3.3
3.4 Character set, 3.3
3.5 Keywords, 3.3

3.6 Identifiers, 3.4
3.7 Structure of a 'C' program, 3.4
3.8 Constants, 3.6
3.9 Variables, 3.7
3.10 Data types, 3.8
3.11 Variable declaration, 3.8
3.12 Assigning values to variables, 3.8
3.13 Symbolic constants, 3.9
3.14 Enumerated data type, 3.10
3.15 Turbo-C IDE hot keys, 3.11
3.16 Different files in a 'C' programming environment, 3.11
❖ Summary, 3.12
❖ MCQs, 3.12
❖ Exercises and Answers to selected Exercises, 3.13
❖ Short Questions, 3.14

Chapter-4 : Operators and Expressions 4.1 – 4.2

4.1 Introduction, 4.2
4.2 Arithmetic operators, 4.2
4.3 Relational operators, 4.5
4.4 Logical operators, 4.5
4.5 Assignment operators, 4.6
4.6 Increment and decrement operators, 4.7
4.7 Conditional operator (Ternary operator), 4.8
4.8 Bitwise operators, 4.9
4.9 Other special operators, 4.11
4.10 Precedence and associativity of operators, 4.12
4.11 Type conversion, 4.13
4.12 Type casting, 4.14
4.13 Header files, 4.15
4.14 Preprocessor directives, 4.16
4.15 Solved programming examples, 4.16
❖ Summary, 4.19
❖ MCQs, 4.19
❖ Exercises and Answers to selected Exercises, 4.21
❖ Short Questions, 4.23

Chapter-5 : Input-Output 5.1 – 5.16

5.1 Introduction, 5.2
5.2 Input with scanf( ) function, 5.2
5.3 Output with printf( ) function, 5.3

5.4 Character input, 5.4
5.5 Charcter output, 5.6
5.6 String input/output, 5.8
5.7 Formatted input using format specifires, 5.8
5.8 Formatted output, 5.9
5.9 Solved programming examples, 5.11
❖ Summary, 5.13
❖ MCQs, 5.14
❖ Exercises and Answers to selected Exercises, 5.14
❖ Short Questions, 5.16

Chapter-6 : Decision Making Structures **6.1 – 6.28**

6.1 Introduction, 6.2
6.2 if statement, 6.2
6.3 if...else statement, 6.2
6.4 Nested if statement, 6.4
6.5 if...else...if ladder (Multiple if...else) statement, 6.6
6.6 switch statement, 6.9
6.7 break statement, 6.13
6.8 default keyword, 6.15
6.9 goto statement, 6.15
6.10 Solved programming examples, 6.17
❖ Summary, 6.25
❖ MCQs, 6.26
❖ Exercises and Answers to selected Exercises, 6.26
❖ Short Questions, 6.28

Chapter-7 : Looping Control Structures **7.1 – 7.45**

7.1 Introduction, 7.2
7.2 Types of loops, 7.2
7.3 While loop, 7.2
7.4 Do...While loop, 7.11
7.5 For loop, 7.15
7.6 Nesting of loops, 7.20
7.7 break statement, 7.28
7.8 continue statement, 7.29
7.9 Solved programming examples, 7.31
❖ Summary, 7.40
❖ MCQs, 7.41
❖ Exercises and Answers to selected Exercises, 7.41
❖ Short Questions, 7.45

Chapter-8 : Arrays and Strings

8.1 Array, 8.2
8.2 Single dimensional array, 8.2
8.3 Initialization of single dimensional array, 8.12
8.4 Two dimensional array, 8.14
8.5 Initialization of two-dimensional array, 8.19
8.6 Multi-dimensional array, 8.19
8.7 String, 8.20
8.8 Reading strings, 8.20
8.9 Printing strings, 8.21
8.10 String handling built-in functions, 8.24
8.11 Solved programming examples, 8.26
❖ Summary, 8.42
❖ MCQs, 8.43
❖ Exercises and Answers to selected Exercises, 8.44
❖ Short Questions, 8.49

Chapter-9 : Pointers

9.1 Why pointers ?, 9.2
9.2 Concept of pointers, 9.2
9.3 Pointer declaration, 9.3
9.4 Pointer arithmetic, 9.6
9.5 Pointer to pointer, 9.9
9.6 Pointers and arrays, 9.10
9.7 Array of pointers, 9.14
9.8 Pointers and strings, 9.14
❖ Summary, 9.15
❖ MCQs, 9.16
❖ Exercises and Answers to selected Exercises, 9.17
❖ Short Questions, 9.18

Chapter-10 : Functions

10.1 Introduction, 10.2
10.2 User defined functions, 10.2
10.3 Library functions, 10.2
10.4 How the control flow takes place in multi-function program ?, 10.2
10.5 Function declaration, 10.3
10.6 Function definition, 10.3
10.7 Category of functions, 10.5
10.8 Scope of variables, 10.11
10.9 Parameter passing to function, 10.13

10.10 Recursion, 10.16
10.11 Parameters as array, 10.22
10.12 Parameters as string, 10.25
10.13 Function returning a pointer, 10.29
10.14 Storage classes, 10.30
10.15 Solved programming examples, 10.34
❖ Summary, 10.40
❖ MCQs, 10.41
❖ Exercises and Answers to selected Exercises, 10.43
❖ Short Questions, 10.48

Chapter-11 : Structures and Unions **11.1 – 11.30**

11.1 Introduction, 11.2
11.2 What is structure ?, 11.2
11.3 Accessing structure members, 11.3
11.4 Structure initialization, 11.5
11.5 Array of structures, 11.6
11.6 Nested structures, 11.9
11.7 Structure and functions, 11.12
11.8 Pointers and structures, 11.14
11.9 Array of pointers to structures, 11.16
11.10 Self-referential structures, 11.18
11.11 Solved programming examples, 11.21
❖ Summary, 11.25
❖ MCQs, 11.25
❖ Exercises and Answers to selected Exercises, 11.27
❖ Short Questions, 11.29

Chapter-12 : Dynamic Memory Allocation **12.1 – 12.12**

12.1 Introduction, 12.2
12.2 Dynamic memory allocation, 12.2
12.3 Allocating a block of memory – malloc() function, '12.3
12.4 Allocating multiple blocks of memory - calloc() function, 12.6
12.5 Releasing the used space: free() function, 12.7
12.6 Linked lists, 12.7
12.7 Solved Programming examples, 12.8
❖ Summary, 12.11
❖ MCQs, 12.11
❖ Exercises and Answers to selected Exercises, 12.12
❖ Short Questions, 12.12

Chapter-13 : File Management 13.1

13.1 Introduction, 13.2
13.2 Defining, Opening and Closing a file, 13.2
13.3 Input/Output operations on a file, 13.4
13.4 Command line arguments, 13.9
❖ Summary, 13.11
❖ MCQs, 13.11
❖ Exercises and Answers to selected Exercises, 13.12
❖ Short Questions, 13.12

APPENDIX-I : GTU Question Paper P.1

❖ Winter 2018, P.1
❖ Summer 2019, P.1

* * *

CHAPTER

1 Introduction to Computer & Programming

1.1 INTRODUCTION
1.2 COMPUTER
1.3 COMPUTER SYSTEM CHARACTERISTICS OR FUNCTIONS
1.4 CLASSIFICATION OF COMPUTER SYSTEM
1.5 APPLICATION OF COMPUTER
1.6 BASIC BLOCK DIAGRAM AND FUNCTIONS OF COMPUTER COMPONENTS
1.7 LIMITATION OF COMPUTER
1.8 COMPUTER SOFTWARE
1.9 COMPUTER HARDWARE
1.10 PROGRAMMING LANGUAGES
1.11 COMPUTER GENERATION
1.12 DIFFERENCES

- SUMMARY
- MCQS
- EXERCISES AND ANSWERS TO SELECTED EXERCISES
- SHORT QUESTIONS

1.1 INTRODUCTION :

In 21st century, we are crossing the threshold of new information era in which we are developing tools that permit us to amplify human intelligence and acquire the information needed to explore new systems in area of education, research, health care, commercial business and manufacturing etc. This new information era is nothing but the **computer Era**. Here in this chapter basic fundamentals of computer system, software and hardware is described.

1.2 COMPUTER :

It is an electronic device which accept data from out side world (standard input devices) and manipulates or process it at high speed according to instruction given. It has memory to store large amount of data, can process it in accurate form with high speed by using power full processor. Computer is also called data processor because it can store, process and retrieve data when ever required. A computer system consist of six elements.

1. hardware
2. software
3. instructions or procedure or module
4. data/information
5. communication
6. people

Fig 1.1 computer system

A computer can not work on its own, but work is based on some prior instructions given to it that is known as computer programming. According to instruction it perform action. A computer therefore follows a series of instructions, programmed into its memory by its user, here schematic diagram of data processing is shown in fig 1.2.

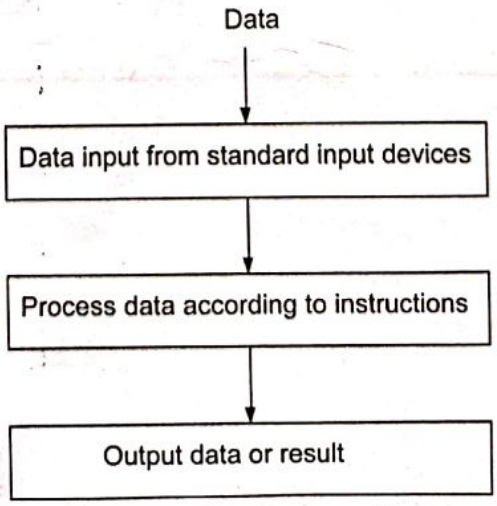


Fig 1.2 data processing

data : fact or raw material used as input

information : process data into meaningful form.

1.3 COMPUTER SYSTEM CHARACTERISTICS OR FUNCTIONS :

1. **Speed and accuracy :** First basic reason of usage of computer is high speed. The speed is measured in clock cycle i.e. Hertz (Hz), now a days desktop are available in speed in GHz.

   In addition to its speed, computers are very accurate. The circuits in a computer having electronic parts, which do not wear and tear. Instruction are carried out without any mistake.

   Computer error caused due to incorrect input data or unreliable procedure or program, which are often referred to as a Garbage – In – Garbage – out (GIGO).

2. **Diligence and maintenance :** It is machine so it works tirelessly around 24 hrs of a day. It is also free from monotony and lack of concentration. It is best tool or machine for performing repetitive jobs and operations.

3. **Vast storage media :** It can store and process vast amount of information in very little space. Computer processing unit break this information in a form which is understandable to it and process it to put in more intelligible form. We can call back these information from its memory and use into calculation or for manipulation thus act as a vast storage media.

4. **Time factor :** Response time is greatly reduced while the total time spent in various decision making activities is reduced to minimum. Most instruction are carried out in fraction of second.

5. **Permanent :** In a computer system a very large amount of data can be stored. Data can be in the form of digits, alphabet, image, audio or video.

6. **Versatility :** Computer is capable of performing any task, by generating finite series of logical steps.

1.4 CLASSIFICATION OF COMPUTER SYSTEM :

It is classified according to principles of operation, size and number of user

- Based on electronic technology
- Based on number of users and working environment
- Based on accuracy and speed
- Based on storage requirement

They are also classified as

- Micro computer
- Mini computer
- Main frame computer
- Super computer.
- Note book computer
- Pam top computer

1.5 APPLICATION OF COMPUTER :

There are various area where computer can be used.

1. **Reservation system** (air, rail way, bus)
2. **Crime detection**
3. **Production system**
4. **Space technology**
5. **Weather forecasting**
6. **Research and science**

7. **Cost analysis**
8. **Banking system**
9. **Hotel management**
10. **Hospital management**
11. **Library management**

1.6 BASIC BLOCK DIAGRAM AND FUNCTIONS OF COMPUTER COMPONENTS : (June, 09)

Basic block diagram of computer system is drawn in fig 1.3. There are basic five components which are

1. Central processing unit (CPU) (ALU and CU)
2. Memory
3. Input device
4. Output device
5. Secondary storage device

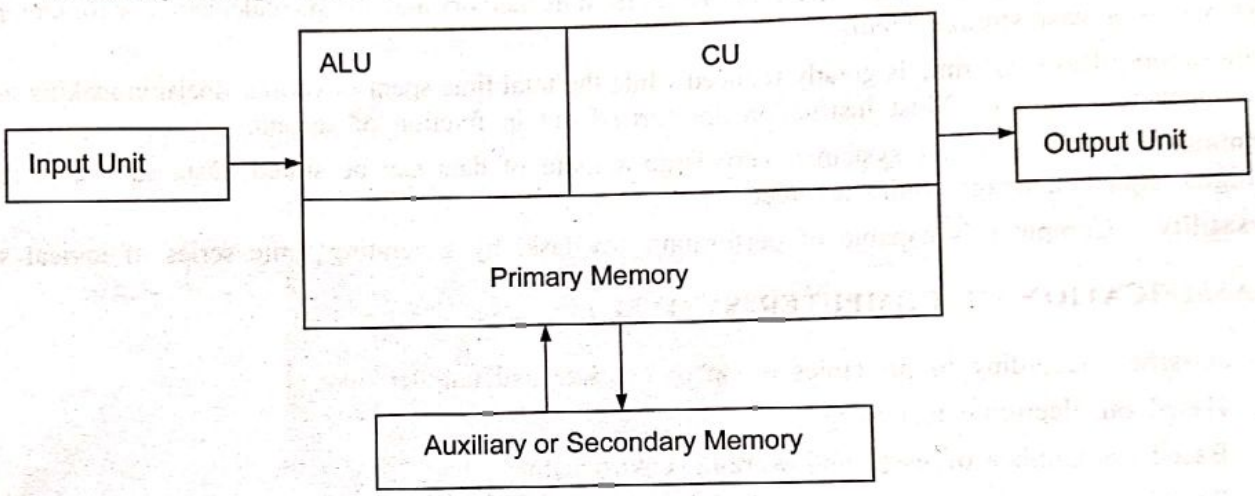


Fig 1.3 block diagram of computer system

1. **Central Processing Unit (CPU) :** The central processing unit (**CPU**), some times referred to as "**brain** of the system, is main part of computer system that contain electronic circuitry that actually process th data. Acting on the instruction it receives, the CPU performs operation on data. It also controls the flo of data through the system, directing the data to enter the system placing data in memory and retrievi them when needed, and directing the output. CPU consists of

   1. ALU (Arithmetic logic unit)
   2. CU (Control Unit)

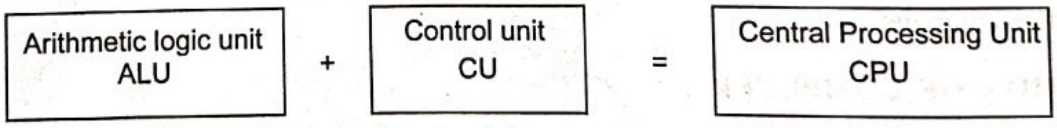


Fig 1.4 Processing outline of CPU

1. **ALU :** Here actual execution instruction take place during processing operation. It performs all arithme calculations and take logical decision. It can compare, count, shift or other logical activities. All such calculat and comparisons are done in this unit. It takes data from memory unit and return information (process da to memory unit if storage require. It calculates very fast.
2. **CU :** It manages and coordinates operations of all other components of computer system. It also perfo following functions.
   - It retrieve instruction or data from memory.

- It decode instruction
- It coordinate time sequence of instruction among various component of system
- It determine requirement of storage and take action according to it.
- It also fetch instruction from main memory.

3. **Memory Unit :** The storage unit of computer system store data for following purposes.
   - Processing data and instructions
   - Storage of temporary result (intermediate data storage)
   - Permanent storage for future requirement (secondary storage)

   There are two types of storage.

   (1) Primary storage

   (2) Secondary or auxiliary storage.

   (1) Primary storage (user RAM) :

   The main features of primary storage are...
   - Store current program or data (running program)
   - It also store temporary data of current program.
   - Less space in comparison of secondary storage.
   - Volatile (data losses on power off)
   - Comparatively more expensive
   - Fast in operation

   (2) Secondary or auxiliary storage :

   The main features of secondary storage are ...
   - Used to store data and program for future usage.
   - Large capacity (GB) in comparison with primary memory
   - Slower than primary memory
   - Retain data without power
   - Cheaper than primary memory

4. **Input /Output Unit :**

Basic aim of I/O devices is to provide communication between computer and user. Input device transmit the data as a series of electrical pulses into the computer memory unit where it will be available for processing. The input devices translate data into a code be read by computer system electronic circuitry. Some input devices, such as the keyboard, enable the user to communicate directly with the machine. Other require data to be first recorded on an input medium such as paper or magnetized material.

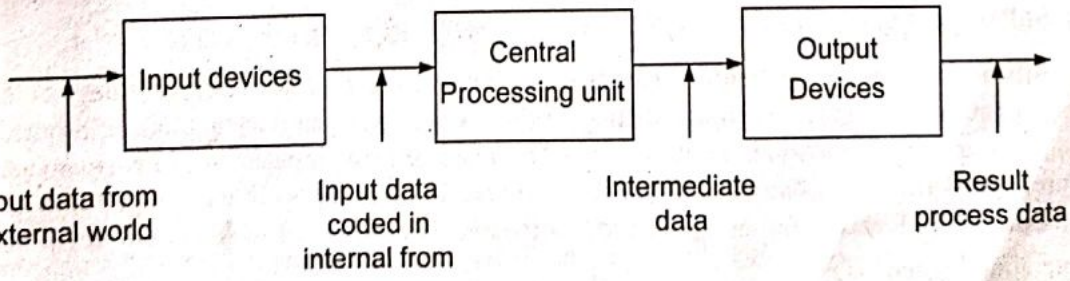


Fig. 1.5 Input/output devices processing diagram

Output devices :

Output devices supply result of processing from primary storage or secondary storage. When a program executed and the results must be made available in a human readable form. The computer system needs an output unit to communicate the processing information to the user. The output device translate processed data from a machine coded form to a form that can be read and used by people. The most common types of output devices are monitor which resembles a screen, printer prints copy from computer on paper, plotter plot diagram or figure on paper. A new type of output device is the speech synthesizer, a mechanism attached to the computer that produces verbal output sounding almost like human speech.

1.7 LIMITATION OF COMPUTER :

- computer cannot identify any input / output by itself for solving problem
- it cannot make any decision related problem, solution formula must be given.
- it can not make any decision on the base of experience.
- It can not interpret the data store in to storage media in different aspects as per situation.
- For every solution rules or procedure or sequence needed.

1.8 COMPUTER SOFTWARE :

Computer software refers to collection of programs. Program is the collection of instructions, which perform particular task and collection of programs which accomplishes application is called software.

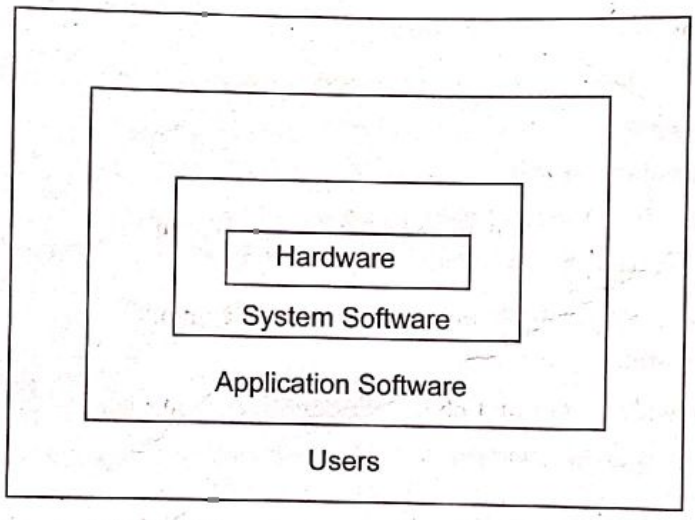


Fig. 1.6 Layer structure of software

There are basically two types of software available

1. **System software**
2. **Application software**

1. System Software :

System software is designed to control operation and extend the processing capabilities of computer system. They control internal computer functions. such as reading data from input device, transmit processing information to output devices, provide various facilities likes file management, Input/Output management and CPU time to user program during execution of program. The help hardware component work together and provide support for development and execution of application software. There are various system software available.

1. operating system (OS)
2. compiler
3. assembler

4. loader
5. linker
6. editor
7. translator
8. macro processor.
9. interpreters

(1) Operating System :

It is a system software which manages hardware as well as interacts with user and provide different services to user program. services like memory management, I/O management, CPU management, process management. Example of OS are Vista, Windows Xp, Linux, D.O.S. etc. It is also classified as...

- single user OS
- multi user OS
- multi tasking OS
- multi programming OS
- multi threading OS
- time sharing OS
- batch processing OS
- distributed OS
- network OS
- real time OS

(2) Compiler :

It translates higher level program to machine level program.

(3) Assembler :

It translates lower level program to machine level program.

(4) Loader :

It loads Operating System part and object program into main memory for execution purpose. e.g. Boot strap loader.

(5) Linker :

Which bind symbolic code of source and library file to make executable program.

(6) Editor :

Used to create a program or software, also used to edit it. It may be a part of a software development package.

(7) Translator :

It converts one language into other. Example is compiler, assembler, interpreter.

(8) Macro processor :

It is used before translator, replace symbolic meaning into their equivalent code. They are also called pre-processor.

(9) Interpreter :

It translates line by line high level programs into low level programs.

2. Application Software :

A software designed for user specific need is called application software or application package e.g. library management, school management, hospital management etc.. They are developed with help of programming

language or packages. Some program direct the computer to perform specific tasks requested by user, su as printing report of customize program, mail, store result etc., such program also called application softwa Application software are available from vendors. There are two types of application software available

(1) general purpose.

(2) specific purpose software.

(1) General purpose :

They are designed for many task and provide many number of tasks and provides many feature e.g. Microsoft Office, PageMaker etc..

(2) Specific purpose software :

The packages pay roll, financial accounting, inventory control etc are some of the specific purpo softwares. The pay roll packages developed for an organization following a set of rules and may n apply to another organization as the pay calculation rules varies. The other packages developed f one organization cannot be applied directly to any other organization.

1.9 COMPUTER HARDWARE :

- It is a physical component of the computer system.
- The term hardware refers to the part of computer user can touch.
- It consists of interconnected electronic devices that control every thing the computer does.
- It consists of a processor, input and output devices, memory, storage devices.
- For maintenance it require special software and some hardware tools.
- Peripherals of computer are also called hardware e.g. printer, plotter, cd, dvd, floppy disk, joy stik key board, mouse etc.

Processor :

The role of processor is to process complex task in to meaning full form as per user requirement. It i like the brain of computer, the part that organizes and carries out instructions that come from either th user or the software. In personal computer, it is also called as microprocessor, which is fitted in to mothe board of system. CPU is divided into two parts ALU and CU which are explained earlier in this chapter.

Memory : memory is classified in to two categories.

(1) primary memory

(2) secondary memory

(1) Primary memory : it is classified as

- Random Access memory (RAM)
- Read only memory (ROM)
- Programmable read only memory (PROM)
- Erasable programmable memory (EPROM)
- Electrically Erasable Programmable (EEPROM)

(2) Secondary memory : it is also called as external memory.

- hard disk (HDD)
- floppy disk
- cd
- dvd
- removable hard disk
- pen drive

1.10 PROGRAMMING LANGUAGES :

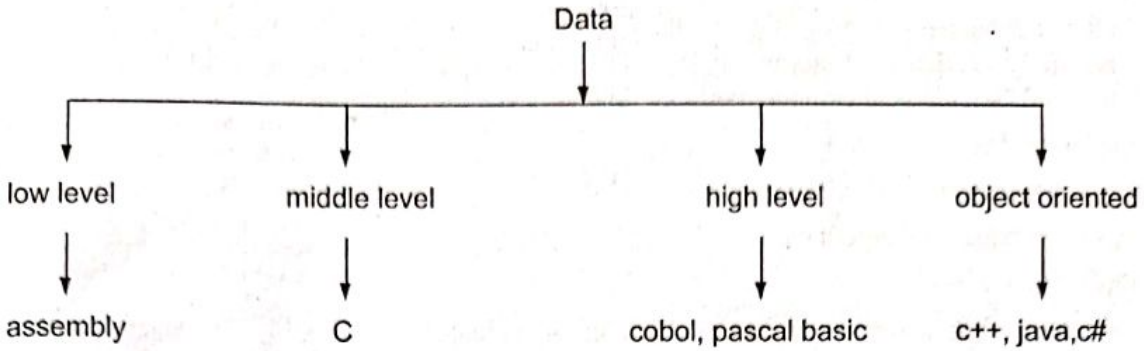


Fig.

Language is system for communicating. Written languages use symbols (characters) to build words. The entire set of words is language's vocabulary. The way in which words can be meaningfully combined are defined by the language's syntax and grammar. The actual meaning of words and combinations of words is defined by the language's semantics.

Programming language is a vocabulary and set of grammatical rules for instructing computer to perform specific tasks. It is classified as

1. Machine language or low level
2. Middle level language
3. High level language
4. Object oriented language

1. Machine or low level language :

In this language programs are written in 0's and 1's forms (binary language). which are directly understandable to computer system. Every CPU has its own machine language. It has following features.

1. Program execution is faster in comparison with others.
2. Due to direct writing in to machine code the size of code is compact and small in comparison with other languages.
3. It occupies less memory

Drawback or disadvantages

1. It require depth technical knowledge to debug program. Some time all program cannot be debug step by step.
2. Due to only two combination (0's and 1's) it is difficult to understand.

2. Assembly language :

- It is also classified as low – level language.
- This language uses mnemonics in place of 0's and 1's to represent codes.
- The word mnemonic refers to a memory aid.
- It uses symbolic addressing capabilities.
- Symbolic presentation makes easy to write program

It has following limitations.

1. Coding in this language is time-consuming.
2. This language is machine-oriented i.e. they are designed for the specific make and model of processor being used.

3. High level language :

These are the languages whose instructions closely resemble human language and mathematical notati
Unlike assembly language, programs made in this language may be used with different makes of compu
with little modification. Other advantages of higher level language are :

1. Easier to learn
2. Requires less time to write
3. Provides better documentation
4. Easier to maintain

As a result, high level languages are used more often than machine or assembly languages. High level languag
must also be translated in to machine language before they can be used by a computer. One of the t
different language-translator program is used to translate high-level language : compiler and interpreter.

4. Object oriented Programming :

- Object oriented programming is a technique for creating common building blocks of a program call
object and assembling different sets of objects to solve specific problems.
- Object oriented programming languages use a structure that defines the objects in a program alo
with their properties or actions. e.g. C++,JAVA.
- An object is pre-defined set of program code which will always behave so that it can be used
other application.
- Instead of writing the entire program line by line, the programmer combines the objects and wri
small amount of code that is necessary finishing the program.
- A class is an abstract concept for a group of related objects. All of the objects in a class inhe
the characteristics of that class. For example, if a class is automobiles, the members of that cl
inherit the class properties of having four wheels, an engine and doors.

Advantages	Disadvantages
1 It is Graphical User Interface (GUI), and therefore easier to use. 2. Larger program produced by the use of OOPs are very slow and use more memory and computer resources.	1. The initial development costs are very hig 2. It enables faster program development an thereby increases the programmer productivit

⇒ **Summary of characteristics of languages :**

Sr. No.	Language	Characteristics
1	Machine language	1. machine dependent. 2. uses special codes and assignment of storage address.
2	Assembly Language	1. machine dependent 2. uses mnemonics 3. 1 to 1 language i.e. for assembly language instruction generated
3	High-level language	1. machine independent 2. uses instructions which seem english-like. 3. 1 to many language i.e. for high level instruction, many machine language generated.

1.11 COMPUTER GENERATION :

No.	Particulars	First	Second	Third	Fourth	Fifth
1.	Year	1949-55	1956-65	1966-75	1976-95	1996 onwards
2.	Size	Room size	Cupboard size	Desk size mini-computers	Type–writer size computers & laptops	Credit card sized computers, palmtops
3.	Density	One component per ckt	100 components per ckt	1000 components per ckt	Hundreds of thousands of components	Million of components per ckt
4.	Technology	Vacuum tubes	Transistors	Ic's	LSI	VLSI
5.	Storage	Magnetic drum, tape	Magnetic tape, disk	Magnetic disk, fdd, optical disk	Optical disk, HDD	Ultra high disk with large storage capacity
6	Operating speed	Milli-seconds 10^{-3}	10^{-6} second	10^{-9} second	10^{-12} second	10^{-15} second
7	Mean time between failure	Minutes	Days	Weeks to month	Months to year	Years

1.12 DIFFERENCES :

1. Distinguish between compiler and interpreter.

No.	Particular	Compiler	Interpreter
1	Scheme	Compiler scan the whole program at a time and lists out errors if any.	Interpreter scans the program line by line and stop scanning whenever error occurs.
2	Manner	Compiler converts the whole source code into object code at a time	Interpreter converts the source program line by line.
3	Source code	After compilation, source code is not required.	For every execution run source is required.
4	Speed	Execution is faster	Execution is slower
5	Object code	The object code is generated when the program is error free.	No object code file is generated.

2. **Give difference between system software and application software.**

No.	Particular	System Software	Application software
1	**Purpose**	System software comprise of those programs, which directs the computer.	Application software is the software developed for solving business problems.
2	**Varying nature**	System software varies from computer to computer	Application software varies from organization to organization.
3	**Language**	System software is written in low-level language/middle level language	Application software are usually written in high level language
4	**Knowledge**	Detail knowledge of hardware is required.	This requires detailed knowledge of organization.
5	**Use**	System software is used to improve the performance and maximum utilization of system resources.	Application software programs are used to improve the speed and quality of a business activity.
6	**Developed by**	The manufactures along with the hardware usually supply system software.	Application software is developed by individuals or supplied by software vendors as generalized application software.

3. **Give difference between Data and Information**

No.	Particulars	Data	Information
1	**Meaning**	**Data** is plural of word **Datum.** It is defined as the raw of facts or observations or assumptions or occurrence about physical phenomenon or business transaction.	It is the data that has been converted in to a meaningful and useful context for specific end users.
2	**Organization**	Data is the collection of facts, which is unorganized.	Information is an organized or classified data having a high value for its users.
3	**Features**	They are the objective measurements of attributes (characteristics) of entities (like people, place, thing and events)	To obtain information, data is subjected to the following : 1. data's form is aggregated, manipulated and organized 2. its content is analyzed and evaluated. 3. it is placed in a proper context for a human user.
4	**Nature**	Data should be accurate but need not be relevant, timely or concise.	Information must be relevant, accurate timely, concise and complete and apply to the current situation. It should be con densed into suitable length.
5	**Form**	It can exist in different forms. E.g. picture, sound, text or all these together	It exists as reports, in a systematic textual format, or as graphics.

: SUMMARY :

- **Software** is a collection of programs. Software is basically of two types: **System software**, and **Application software.**
- **System software** extends the processing capability of computer system and help in development of application software. Some of the examples of system software are: Operating system, Compiler, Assembler, Loader, Linker, Editor etc.
- Software designed for user specific need is called **Application software.** Some of the examples are: Library Management Software, Payroll System, School Management System etc.
- **High level programming languages** are machine independent and use instructions which are English–like language. Some of the examples are: 'C', Pascal, COBOL etc.
- **Compiler** is system software which compiles the source program code and generates an object code file. Once object code file is there, we do not require source code file for execution of a program. Compiler is faster than Interpreter.

: MCQs :

1. Full form of CPU is
   (a) Control Processing Uniform (b) Control Programming Unit
   (c) Central Processing Unit (d) Central Producing Unit
2. Full form ALU is
   (a) Algorithm Legal Unit (b) Arithmetic & Logic Unit
   (c) Array Logic Unit (d) None of above
3. Which memory is volatile?
   (a) Main memory (b) Secondary memory (c) Both a and b (d) None of above
4. Which is a system software?
   (a) MS word (b) Library Management System
   (c) Student Management System (d) Compiler
5. Which language is machine independent?
   (a) Machine level (b) Assembly level (c) High level (d) None of above
6. What is known as brain of computer?
   (a) CU (b) ALU (c) CPU (d) None of above
7. What is full form of BIOS?
   (a) Basic Input Output System (b) Binary Input Output System
   (c) Bus Input Output System (d) None of above
8. BIOS is stored in
   (a) ROM (b) RAM (c) Secondary device (d) None of above
9. COBOL belongs to which generation?
   (a) First (b) Second (c) Third (d) Fourth
10. To Learn Computer, without going to classroom what will be other way
    (a) Distance Learning (b) I–Learning (c) Digital Learning (d) E–Learning
11. How to write e_mail Address ?
    (a) xyz@website.info (b) xyz@website@info (c) xyz.website.info (d) xyzwebsite.info
12. What will be a meaning of 1KB of memory
    (a) 1 Kit Bit (b) 1 Kilo Byte (c) 1 Kernel Boot (d) 1 Key Block

13. Full Form of RAM is
    (a) Random Access Memory (b) Ready Application Module
    (c) Remote Access Machine (d) Read Access Memory

14. What is Computer Virus ?
    (a) Software (b) Hardware (c) a and b (d) None of abo

15. Full form of IP is
    (a) Internet Principle (b) Intercom Protocol (c) Intranet Protocol (d) Internet Prot

16. What will be extension of Web Page's file ?
    (a) .xls (b) .3gp (c) .html (d) .ppt

17. Which of the following you will find on an inkjet printer?
    (a) It has an ink ribbon (b) It has an ink cartridge
    (c) It has high voltage power supply (d) It uses toner powder

18. Why is it important to keep Windows updated using the Windows update website or automatic upd
    (a) To keep your system secure and to install critical updates.
    (b) To keep your system fast.
    (c) To keep your system optimized for the new game.
    (d) To keep your system protected from overheating.

19. Which protocol is used to display web pages?
    (a) SNMP (b) SMPT (c) Telnet (d) HTTP

20. In Microsoft Excel, you can use horizontal and vertical scroll bar to
    (a) Split a worksheet to two panes (b) edit the contents of a cell
    (c) view different rows and columns (d) view different worksheet

21. The system unit of personal computers typically contains all of the following except
    (a) Microprocessors (b) Disk controller (c) Serial interface (d) Modem

22. Which of the following is a read only memory storage device
    (a) Floppy disk (b) CD–ROM (c) Hard disk (d) None of these

23. Software required to run the hardware is known as
    (a) Task Manager (b) Task Bar (c) Program Manager (d) Device Driver

24. Which the following is application software?
    (a) Compiler (b) Power Point (c) Debugger (d) None of the ab

25. Which of the programming language is said to be machine independent language?
    (a) High Level Language (b) Machine Language
    (c) Assembly Language (d) All the Above

26. Which of the following is smallest Network?
    (a) MAN (b) LAN (c) WAN

27. FTP stands for
    (a) File Transport Protocol (b) File Transfer Protocol
    (c) Folder Transfer Protocol

28. When a key is pressed on keyboard, which standard is used for converting the keystroke into the correspondi bits
    (a) ANSI (b) ASCII (c) EBCDIC (d) ISO

29. Which one is input device?
    (a) Monitor (b) Keyboard (c) CPU (d) Printer

30. Which part of CPU perform calculations and make decision
(a) Alternate Logic Unit (b) Arithmetic Logic Unit
(c) Arithmetic Local Unit (d) Alternate Local Unit

31. What type of memory is volatile ?
(a) Cache (b) ROM (c) RAM (d) Hard Drive

32. With regards to Email Addresses
(a) they must always contain an @ symbol (b) they are case sensitive
(c) they can never contain space (d) All of above

33. Which of the following is a part of primary memory of computer ?
(a) PROM (b) CD–ROM (c) Pen–drive (d) Floppy Disk

34. Which protocol is used to transfer files on internet ?
(a) HTTP (b) SMTP (c) FTP (d) Telnet

35. Microprocessor of computer system is a part of :–
(a) Memory (b) Input Device (c) Output Device (d) Processing Unit

36. Which is not a font style in MS–Word
(a) Bold (b) Superscript (c) italic (d) Regular

37. With which of the following all formulas in excel starts?
(a) / (b) * (c) $ (d) =

38. Which of the following is not one of PowerPoint view?
(a) Slide show view (b) Slide view (c) Presentation view (d) Outline view

39. In the evaluation of a computer language, all of the following characteristics should be considered except?
(a) application oriented features (b) efficiency
(c) readability (d) hardware maintenance costs

40. Keyboard is the type of device
(a) Input (b) Pointing (c) Output (d) Sound

41. A document or image can be scanned in to digital form by using
(a) Marker (b) Printer (c) Light pen (d) Scanner

42. Operating system is installed on
(a) Hard Disk (b) Cache memory (c) Mother Board (d) None of these

43. Recycle bin is used for
(a) To store backup file (b) To restore deleted file/folder
(c) To store most frequently used files (d) All the above

44. The process a user goes through to begin a computer system.
(a) Log out (b) Log in (c) Log off (d) None of these

45. Which among following is not necessary for working of a standalone computer?
(a) RAM (b) Hard Drive (c) Operating System (d) LAN card

46. Which protocol is used to transfer files on Internet?
(a) HTTP (b) FTP (c) SMTP (d) Telnet

47. In Microsoft Excel, you can use the horizontal and vertical scroll bar to
(a) Split a worksheet in two panes (b) edit contents of a cell
(c) view different rows and columns (d) view different worksheet

48. Full form of ROM is
    (a) Random Only Memory
    (b) Read Only Memory
    (c) Read Once Memory
    (d) None of above

: ANSWERS :

1.	(c)	2.	(b)	3.	(a)	4.	(d)	5.	(c)	6.	(c)	7.	(a)
8.	(a)	9.	(c)	10.	(d)	11.	(a)	12.	(b)	13.	(a)	14.	(a)
15.	(d)	16.	(c)	17.	(b)	18.	(a)	19.	(d)	20.	(c)	21.	(d)
22.	(b)	23.	(d)	24.	(b)	25.	(a)	26.	(b)	27.	(b)	28.	(b)
29.	(b)	30.	(b)	31.	(c)	32.	(d)	33.	(a)	34.	(c)	35.	(d)
36.	(d)	37.	(d)	38.	(b)	39.	(d)	40.	(a)	41.	(d)	42.	(a)
43.	(b)	44.	(b)	45.	(d)	46.	(b)	47.	(c)	48.	(b)		

: EXERCISES :

1. What is computer ? explain it.
2. List various applications of computer system.
3. Draw block diagram computer and explain each block.
4. What is software ? explain
5. Give difference between software and hardware.
6. Give difference between compiler and interpreter
7. Describe various types of computer languages and mention its advantages and disadvantages.
8. Give difference between application software and system software
9. Give difference between data and information
10. Explain computer generation.

: ANSWERS TO SELECTED EXERCISES :

3. Draw block diagram computer and explain each block.

Ans. :

Here, Basic block diagram of computer system is drawn in **fig a**. There are five Basic components, which are ...

1. Central processing unit (CPU) (ALU and CU)
2. Memory
3. Input device
4. Output device
5. Secondary storage device

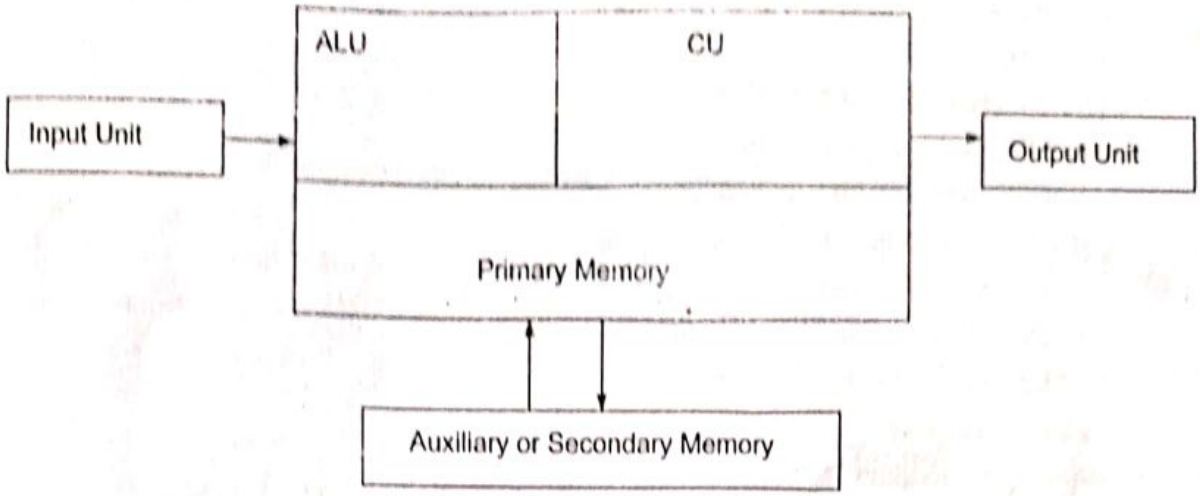


Fig a Block diagram of computer system

1. Central Processing Unit (CPU) :

The central processing unit (**CPU**), some times referred to as "**brain** " of the system, is main part of computer system that contain electronic circuitry that actually process the data. Acting on the instruction it receives, the CPU performs operation on data. It also controls the flow of data through the system, directing the data to enter the system placing data in memory and retrieving them when needed, and directing the output. CPU consists

1. ALU (Arithmetic logic unit)
2. CU (Control Unit)

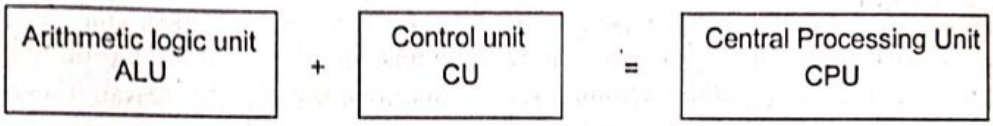


Fig b Processing outline of CPU

(1) ALU :

Here actual execution instruction take place during processing operation. It performs all arithmetic calculations and take logical decision. It can compare, count, shift or other logical activities, all such calculation and comparisons are done in this unit. It takes data from memory unit and return information (process data) to memory unit if storage require.

(2) CU :

It manages and coordinates operations of all other components of computer system. It also perform following functions.

- It retrieve instruction or data from memory.
- It decode instruction
- It coordinate time sequence of instruction among various component of system.
- It determine requirement of storage and take action according to it.
- It also fetch instruction from main memory.

2. Memory Unit :

The storage unit of computer system store data for following purposes.

- Processing data and instructions
- Storage of temp. result (intermediate data storage)
- Permanent storage for future requirement (secondary storage)

There are two types of storage.

1 : primary storage

2 : secondary or auxiliary storage.

(1) Primary Storage (user RAM) :

The main functions of primary storage are...

- Store current program or data (running program)
- It also store temp data of current program.
- Less space in comparison of secondary storage.
- Volatile (data losses on power off)
- Comparatively more expensive
- Fast in operation

(2) Secondary or Auxiliary Storage :

The main functions of secondary storage are...

- Used to store data and program for future usage.
- Large capacity (GB) in comparison with primary memory
- Slower than primary memory
- Retain data without power
- Cheaper than primary memory

3. Input/Output Unit :

Basic aim of computer system to provide communication between computer and user. Input device transmit the data as a series of electrical pulses into the computer memory unit where it will be available for processing. The input devices translate data into a code be read by computer system electronic circuitry. Some input devices, such as the keyboard, enable the user to communicate directly with the machine. Other require data to be first recorded on an input medium such as paper or magnetized material. The travel only in one direction i.e. from input device to the CPU.

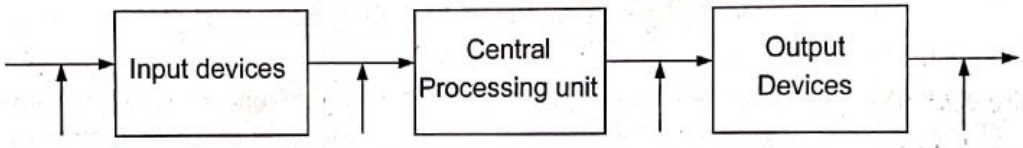


Fig c Input/output devices processing diagram

Output devices :

Output devices supply result of processing from primary storage or secondary storage. When a program executed and the results must be made available in a human readable form. The computer system needs an output unit to communicate the processing information to the user. Te out device translate processed data from a machine coded form to a form that can be read and used by people. The most common types of output devices are monitor which resembles a screen, printer prints copy from computer on paper, plotter plot diagram or fig on paper. A new type of output device being developed now is the speech synthesizer, a mechanism attached to the computer that produces verbal output sounding almost like human speech.

7. Describe various types of computer languages and mention its advantages and disadvantages.

Ans. :

Programming language is a vocabulary and set of grammatical rules for instructing computer to perform specific tasks. It is classified as

1. Machine language or low level
2. Middle level language
3. High level language
4. Object oriented

1. Machine or low level language :

In this language programs are written in 0's and 1's forms (binary language). which are directly under stand able to computer system. Every CPU has its own machine language. It has following features.

1. program execution is faster in comparison with others.
2. due to direct writing in to machine code the size of code is compact and small in comparison with other languages.
3. it occupies less memory

Drawback or disadvantages

1. It require depth technical knowledge to debug program. Some time all program cannot be debug step by step.
2. Due to only two combination (0's and 1's) it is difficult to understand.

1. Assembly language :

- It is also classified as low – level language.
- This language uses mnemonics in place of 0's and 1's to represent codes.
- The word mnemonic referrers to a memory aid.
- It uses symbolic addressing capabilities.
- Symbolic presentation makes easy to write program

It has following limitations

1. Coding in this language is time-consuming
2. This language is machine-oriented i.e. they are designed for the specific make and model of processor being used.

3. High level language :

These are the languages whose instructions closely resemble human language and mathematical notation. Unlike assembly language, programs made in this language may be used with different makes of computers with little modification. Other advantages of higher level language are :

1. easy to learn
2. require less time to write
3. provide better documentation
4. easy to maintain

As a result, high level languages are used more often than machine or assembly languages. High level languages must also translated in to machine language before they can be used by a computer. One of the two different language-translator program is used to translate high-level language : compiler and interpreter.

4. Object oriented Programming :

- Object oriented programming is a technique for creating common building blocks of a program called object and assembling different sets of objects to solve specific problems.
- Object oriented programming languages use a structure that defines the objects in a program along their properties or actions. e.g. C++,JAVA
- Instead of writing the entire program line by line, the programmer combines the objects and writes small amount of codes that is necessary finishing the program.
- A class is an abstract concept for a group of related objects. All of the objects in a class inherit the characteristics of that class. For example, if a class is automobiles, the members of that class inherit the class properties of having four wheels, an engine and doors.

Advantages :

1 It is Graphical User Interface (GUI), and therefore easier to use.

2 It enables faster program development and thereby increases the programmer productivity

Disadvantages :

1. The initial development costs are very high
2. Larger program produced by the use of OOPs are very slow and use more memory and compu resources.

: SHORT QUESTIONS :

1. List the basic components of a computer

⇒ Basic components are CPU, Memory, Input Device, Output Device, Secondary storage device.

2. What is an Operating System?

⇒ Operating System is a system software which manages hardware and interacts with user to provide differ services such as memory management, I/O management, Process management etc.

3. List high level programming languages.

⇒ Pascal, COBOL, Fortran, Java, C, C++ etc.

4. List object oriented programming languages.

⇒ C++, Java, C# etc.

5. What is compiler?

⇒ Compiler is system software which compiles the source program and generates and object code. Obj code only is sufficient to create executable file and run the program.

6. What is application software?

⇒ Software designed for user specific need is called application software. Library management system, Payr management system, School management system, Inventory management system are some of examples application software.

❖

CHAPTER

3

Introduction to 'C' Langua

3.1 INTRODUCTION

3.2 CLASSIFICATION OF COMPUTER LANGUAGES

3.3 FEATURES OF 'C'

3.4 CHARACTER SET

3.5 KEYWORDS

3.6 IDENTIFIERS

3.7 STRUCTURE OF A 'C' PROGRAM

3.8 CONSTANTS

3.9 VARIABLES

3.10 DATA TYPES

3.11 VARIABLE DECLARATION

3.12 ASSIGNING VALUES TO VARIABLES

3.13 SYMBOLIC CONSTANTS

3.14 ENUMERATED DATA TYPE

3.15 TURBO-C IDE HOT KEYS

3.16 DIFFERENT FILES IN A 'C' PROGRAMMING ENVIRONMENT

- ❖ SUMMARY
- ❖ MCQs
- ❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES
- ❖ SHORT QUESTIONS

3.1 INTRODUCTION :

'C' is the language of choice for most of the beginners. 'C' provides almost all the features which the programmers require. Before going into the details of 'C' language, we will understand the types of programming languages.

3.2 CLASSIFICATION OF COMPUTER LANGUAGES :

The computer languages can be classified into following 3 different categories.

1. Machine Language
2. Assembly Language
3. High Level Language

Machine Language :

It is the language which is directly understood by the processor. The processor understands only the binary language (language of 0s and 1s). The person who does the programming at machine level is known as Machine langauge programmer. For writing machine level language, the person must know the hardware level details of processor and the memory.

The instruction consists of two parts :

Op code Operand

Op code specifies the name of operation and the operand specifies the data on which the operation is to be performed. The data may be specified directly or through the register or through memory address.

The advantage in writing machine language is that no translator is required because it is written in the language the processor directly understands. The disadvantage is that the program will run on that type of processor only. Also it is very cumbersome to write the machine language program because one has to deal with sequence of 1s and 0s.

Assembly Language :

It is an improvement over Machine language programming. It makes use of **Mnemonics Operation Code** and uses symbolic addresses, so program looks like written using some English language words. It is easy for the programmer to remember the Mnemonics Operation Code rather than sequence of 1s and 0s.

Instructions in assembly language consist of 3 parts.

Label : Mnemonic OpCode Operand

Label specifies the symbolic address given to the instruction. This field is optional and used only if this instruction is required to be referred by other instructions in the program. **Mnemonic OpCode** specifies the name of operation to be performed. **Operand** specifies on what data that operation is to be performed.

The advantage of using assembly language is it is easy for the programmer to remember Mnemonic Opcode, which reduces errors in programming, program is easy to maintain.

The disadvantage is that the program will run only on that processor for which it is developed; the translator is required to convert assembly language into machine language. The translator program which converts assembly language into machine language is known as **assembler**.

High Level Language :

In the machine language and assembly language programming, the programmer has to understand the architecture of the processor on which the program is to be run. High level languages are the languages which enable the programmer to concentrate more on the logic part and not on the architectural details of the processor. This improves the efficiency of the programmer. High level languages use English words and some mathematical symbols. Examples of high level languages are Basic, COBOL, Pascal,FORTRAN,C,C++,Java etc.

The advantages are it is easy to learn and use, the program is machine independent, and efficiency of programmer is improved, easy maintenance of program.

The disadvantage is that it requires a special converter (known as **compiler**) which will convert high level language into machine language. The programs are less efficient storage wise and time wise in comparison with machine language and assembly language.

For every separate high level language, you require its own separate compiler. Which means C compiler will compile C programs while Java Compiler will compile Java programs. The compiler will generate an error, if the program contains any invalid instruction, which is not recognized by that high level language.

3.3 FEATURES OF 'C' :

Denis Ritchie developed 'C' language. Previous version was known as 'B' language. Important feature of 'C' language is it is **portable**, by portability we mean that the program can be run on any hardware machine. Other features are it is **modular language**, supports **bit-wise operations** and does not provide input-output statements. The bit-wise operations allow the programmer to do the operations on the data bit by bit, which is a feature of machine language. Because of this facility, 'C' is also known as **middle level language**.

The other important feature of 'C' language is that it is **modular**. The large problem is divided into small sub problems (modules). Each module is individually approached as a separate sub problem and solution is found. Then all the modules are put together and the solution of the large problem is derived. Because of the modularity support, 'C' language is also called as **structured programming language**.

3.4 CHARACTER SET :

Every language has its own character set. 'C' language has its own character set. 'C' program basically consists of keywords, identifiers, constants, operators and some special symbols.

The characters that can be used in a 'C' program are **Alphabets** (A-Z and a-z), **Digits** (0-9), **Special Characters** and **White Space Characters**.

Special Characters - ~ ! @ # $ % ^ & * ( ) _ + { } [ ] : ; ' " < > . ? / \ |

White Space Characters – Space, Tab, Return, New Line, Form feed

3.5 KEYWORDS :

Keywords are words whose meaning is fixed and is known by the compiler, you can not use it for other purpose i.e. we can not change the meaning of keyword. Keywords are also known as reserved words. The keywords must be in second alphabet.

The 'C' language keywords as per ANSI standard.

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

3.6 IDENTIFIERS :

These are the words which are defined by the programmer in a program. Identifiers can be the variables, symbolic constants, functions, procedures. The name of identifier should be meaningful so that the maintenance of the program becomes easy. The rules for naming an identifier are

- The first character must be an alphabet.
- Underscore ('_') is valid letter.
- Reserved words i.e. keywords can not be used as identifiers.
- Identifier names are case sensitive. i.e. name and Name are treated as different identifiers.

3.7 STRUCTURE OF A 'C' PROGRAM :

Every 'C' program has to follow specific structure. 'C' program must have at least one function called as main(). The main() function has control over the other part of the program. Execution of the program starts from the main() function. The 'C' program structure is as shown below:

Documentation
Header files
Constants and Global variables

`main()`
`{`
`Statement(s)`
`}`

User defined functions and procedures with their body

Program :

```
/* Write a program to print the message
Hello ! */
#include  <stdio.h>
void  main()
{
    printf("Hello  !");
}
```

When we run this program, the result will be

Output :

```
Hello  !
```

Now, let us relate the above program with the structure of 'C' program. In our program we have not put any documentation regarding the person who developed the program, date on which it was developed, and the purpose of the program. If we want, we can write it as documentation using comments. Comments will be introduced later.

The line **#include <stdio.h>** includes the header file named as stdio.h. This file is necessary to be included in our program whenever we are doing Input-Output operations. In our program, printf () function is for displaying some data on screen (Output operation), so stdio.h must be included.

In our program, we have not used any symbolic constants or global variables.

Then, next is the line **void main()** which indicates the main() function. The void keyword indicates it do not return any value to operating system. Within the brackets { and } the body of main() function is writte The symbol { indicates start of main function, while the symbol } indicates the end of the program.

The statement **printf("Hello !");** actually prints the

Hello !

message on the screen. In 'C' all the statements must be terminated by semicolon ';' symbol. That is t reason why there is a semicolon after printf("Hello !").

Following figure shows the Turbo-C IDE with a program

```
≡ File Edit Search Run Compile Debug Project Options Window Help
=======[■]==== \GTU\CMAT\HELLO.C=================== 5 =[↕]======
|| /* Write a program to print the message
|| Hello ! */
|| #include <stdio.h>
|| void main()
|| {
|| printf("Hello !\n");
|| }
||
====10:26=====◄■::::::::::::::::::::::::::::::::::::::::::::::::::
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```

As shown in status line, various function keys are given on the screen itself. For example, **F2 key sav** your program, **Alt-F9 key compiles** the program and so on. When we compile a program, the source co i.e 'C' program is translated in to **object code**. The program which is having syntax errors can not b compiled. When we press **CTRL-F9 key to run the program**, the object code is linked with the libra functions needed for execution of the program, and **executable code** is generated. If the program is havi no errors , then the output of the program goes on to the user screen. **To see the output of the progra press ALT-F5 key**. To go back to the editor screen, press any key. **To stop Turbo-C program, pre ALT-X key**.

Program :

```
/* Write a program to print the message
Hello
World ! */
/* this program developed by XYZ */
#include <stdio.h>
void main()
{   /* start of main() */
    printf("Hello\n World !"); /* use of '\n' character*/
}   /* end of main() */
```

When we run this program, the result will be

Output :

```
Hello
World  !
```

/* this program developed by XYZ */. This line is a comment and will be ignored by compiler. The comment is written between /* and */. It improves the readability of the program. Similarly,

/* start of main() */ ,

/* use of '\n' character */ and /* end of main() */ are comments.

In the printf() statement the '\n' character is used. It is called as newline character, it forces the string after '\n' to appear on the next line. So Hello and World ! are displayed on separate lines.

3.8 CONSTANTS :

As the name suggests, constant is something whose value does not change throughout the program. The 'C' language supports following types of constants:

- Numeric Constants

Numeric constant is a number type of constant. It can be **Integer** constant or **Real** constant. Integer constant can be positive or negative number. In integer constant the special symbols such as space, comma, currency etc are not allowed.

Following numbers are valid integer constants:

15

–345

0

+7463

While, following numbers are not valid integer constants:

5,000

15 76

$40

'C' language also supports constants in other number system such as octal and hexadecimal. Octal numbers are **preceded by a symbol 0 (zero)** and hexadecimal numbers are **preceded by 0x or 0X.**

Real constant is a number having fraction part written after decimal point. It can also be represented in scientific notation also. Following numbers are valid real constants.

0.004

–0.34

645.9

0.56

0.4e–2 scientific notation of 0.004

–3.4e–1 scientific notation of -0.34

- Non-numeric Constants

These are the constants which represent other than number. They are of two types. **Character** constants and **String** constants.

Character constants are single character and are enclosed in single quotes. Characters are represented in computer memory using a coding scheme known as ASCII code. Every character has its associated ASCII code.

So, valid character constants are: 'B' 'a' '?' '5' '+'

Back slash constants are the special type of character constants which actually consists of two characters. These are known as **escape sequences**. Escape sequences start with backslash '\' character. These escape sequences have their effect at execution time of program.

Escape Sequences and their meaning

Escape Sequence	Meaning
'\0'	End of String – NULL
'\n'	End of line – take the control to next line
'\r'	Carriage return–take the control to next paragraph
'\f'	Form feed - take the control to next logical page
'\t'	Horizontal tab
'\b'	Back Space
'\\'	Prints backslash character \
'\a'	Alert – provide audible alert
'\"'	Double quote – prints double quote

String constant is a sequence of characters which are enclosed in double quotes. Valid character constants are:

"Vansh"
"Computer"
"–345"
"B"

Here, "–345" and –345 are different. The same applies to "B" and 'B'.

3.9 VARIABLES :

In the program, the other type of identifier called variable which is used whose value changes as the program statements are executed. So, variables are the identifiers whose value changes as opposite to constants. A variable is an identifier, all the rules for naming an identifier applies to variables also.

So, following are **invalid** variables

Name	Remark
1abc	The first character is number
int	**int** is a reserve word, it can not be used to declare identifier
double	**double** is a reserve word, it can not be used to declare identifier
sum qty	space not allowed

3.10 DATA TYPES :

The 'C' language supports 4 fundamental data types. These data types are as below

Type	Purpose	Size (Bytes)	Range of values
char	for storing characters and strings	1 Byte	–128 to +127
int	For storing integers	2 Bytes	–32768 to +32767
float	For storing real numbers	4 Bytes	3.4e-38 to 3.4e+38
double	For storing double precision floating numbers	8 Bytes	1.7e-308 to 1.7e+308

In addition to fundamental 4 types of data, 'C' language supports different qualifiers such as **signed, unsigned, short** and **long**. These qualifiers can be put before the fundamental data types to change the range of values supported. These data types with their size and range are as below.

Type	Size (Bytes)	Range of values
unsigned char	1 Byte	0 to 255
unsigned int	2 Bytes	0 to 65535
short int	2 Bytes	–32768 to +32767
unsigned short int	2 Bytes	0 to 65535
long int	4 Bytes	–2147483948 to +2147483647
unsigned long int	8 Bytes	0 to 4294967295
long double	10 Bytes	3.4 e–4932 to 1.1e+4932

As shown in above two tables, when you want to store only positive numbers, then only unsigned qualifier should be used. The long qualifier uses more bytes, so we can store very large number. So, whether to use qualifier or not that depends on your requirement.

3.11 VARIABLE DECLARATION :

In 'C' , the variable must be declared before using it in program. The syntax for variable declaration is :

```
datatype  var1,var2,….,varn;
```

Here, datatype is one of the 4 fundamental data types with or without qualifier. After the data type we have to write the name of variables separated by comma. Here, var1,var2 upto varn are the name of variables. It is not necessary to declare all variables of same type in one line.

Following are valid declaration of variables.

```
int  sum,  count;
float  weight,  average;
double  rho;
char  c;
```

3.12 ASSIGNING VALUES TO VARIABLES :

Once the variable is declared, it can be assigned a value in the program. Assignment operator = is used to assign a value to a variable.

Syntax :

```
variablename  =  value;
```

Example :

weight = 55.5;

sum = 0;

We can also assign a value to a variable at the time of declaration using following syntax :

```
datatype variablename = value;
```

Example :

```
int sum =0;
int length, count=0;
```

3.13 SYMBOLIC CONSTANTS :

When a constant is used at many places in a program, Due to some reason if the value of that co needs to be changed, then we need to change at every statement where that constant occurs in the pr – so modification of program becomes difficult. If the same program is to be understood by other p then understanding becomes difficult. The symbolic constant helps us in solving these problems. Her constant is given a symbolic name and instead of constant value, symbolic name is used in the pro

Symbolic constant is defined as below :

```
#define symbolic_name value
```

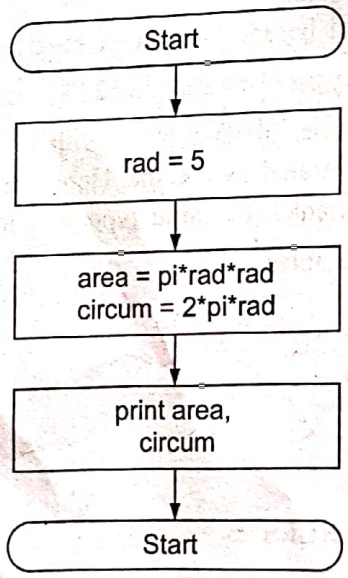
Example :

```
#define FLAG 1
#define PI 3.1415
```

Here, FLAG and PI are symbolic constants. For better readability, it is advisable to use uppercase char in the naming of symbolic constants. See that there is **no semicolon** ';' at the end.

Following program with flowchart explains the concept of declaration and assigning the values to var and using symbolic constants.

Flowchart :



Program :

```
/* Program illustarting use of declartion, assignment of value to variables. Also explains how to use symbolic
constants.
Program to calculate area and circumference of a circle */
#include <stdio.h>
#include <conio.h>
#define PI 3.1415 /* no semicolon here */
main()
{
    float rad = 5; /* declaration and assignment */
    float area, circum; /* declaration of variable */
    clrscr();
    area = PI * rad *rad;
    circum = 2 * PI *rad;
    printf("Area of circle = %f\n",area);
    printf("Circumference of circle =%f\n",circum);
}
```

Output :

```
Area of circle = 78.537498
Circumference of circle =31.415001
```

3.14 ENUMERATED DATA TYPE :

It is a user defined data type. As a programmer we can define our own data types. We can define more than one integer symbolic constants.

Syntax :

```
enum identifier(value1,value2,…,valuen);
```

Example :

```
enum day(sun,mon,tue,wed,thu,fri,sat);
```

Here, enum is a keyword, day is a data type defined & the possible values are as specified in brackets. So, any variable declared of day type can have values which we have specified within brackets.

We can declare variable of enum type as

```
enum day today;
```

We can assign value to variable as

```
today = sun;
```

The compiler automatically assigns integer digits beginning with 0 to all the enumerated constants. For the above example, sun=0, mon=1,… and sat=6.

We can also create the enum type for name of months.

Example:

```
enu mmonth(jan=1,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
```

Here, jan is initialized with value =1, so compiler will assign feb=2,mar =3 and at last dec=12. Now, we can declare variables of month type and use them in the program.

3.15 TURBO-C IDE HOT KEYS :

Following table shows important hot keys in Turbo-C IDE. Remembering the hot keys will help in working efficiently while writing, compiling, debugging and running the program.

Hot Key	Meaning
F1	Activates on-line help
F2	Save the current file
F3	Load a file
F4	Execute program until cursor is reached
F5	Zoom window
F6	Switch between windows
F7	Trace into function call
F8	Trace, but skip function call
F9	Compile and link a program
F10	Toggle between editor and main menu
ALT-F3	Pick a file to load
ALT-F5	Go to user screen
ALT-F9	Compile file and create object file
ALT-C	Activate compile menu
ALT-D	Activate debug menu
ALT-E	Activate editor
ALT-F	Activate File menu
ALT-P	Activate project menu
ALT-R	Activate run menu
ALT-X	Quit Turbo-C IDE
CTRL-F1	Context sensitive help
CTRL-F7	Set a watch expression
CTRL-F8	Set or clear a breakpoint
CTRL-F9	Compile and execute the program

3.16 DIFFERENT FILES IN A 'C' PROGRAMMING ENVIRONMENT:

There are mainly four types of files which a student should know about. They are: source code file, header file, object file and executable file.

Source code file :

It contains the source code of the program. Its extension is '.c'. Normally it contains main () function and may include other header files.

Header file :

These are the files having extension '.h'. It contains definitions of built-in functions. When header file is included, related definitions are automatically available in the program.

Object file :

When we compile a program using a compiler, it generates a compact binary code in a file. The object file has extension '.obj'.

Executable file :

The linker program generates the executable file by linking various object files and creates one executable file. It has an extension '.exe'.

3.16.1 Compilation and execution of a program :

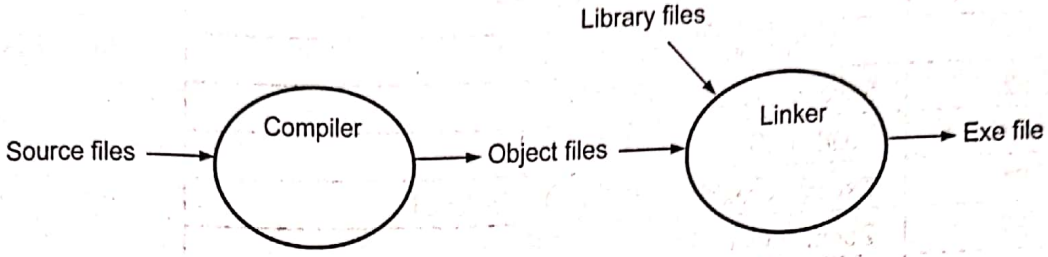


Fig 3.1

As shown in figure 3.1, Source file (.c) file is compiled by compiler and object file is created (.obj) and it is linked by linker with other .obj library files and an executable file is created which can be executed.

: SUMMARY :

- **Denis Ritchie** developed 'C' language. Important features of 'C' language are: It is **portable, Modular structured programming language.**
- **Character set** of 'C' language consist of Alphabets(A–Z and a–z), Digits (0–9), Special characters and white space characters.
- **Keywords** have fixed meaning in a program. They are also known as **reserved words.**
- **Symbolic constant** is a name given to some constant value. Instead of using constant value in a program, we use symbolic constant. Example is:

  #define PI 3.1415
- **Enumerated data type** is a user defined data type. We can define our own more than one symbolic constant. The compiler automatically assigns integer digits beginning with 0.

: MCQs :

1. The translator program which converts assembly language into machine language is known as
   (a) Compiler (b) Assembler (c) Interpreter
2. The translator program which converts high level language into machine language is known as
   (a) Compiler (b) Assembler (c) Loader
3. Which is not the feature of 'C' language?
   (a) Portable (b) Modular (c) Structured (d) None of above
4. Which one is not a keyword of ANSI 'C' ?
   (a) goto (b) int (c) static (d) dynamic
5. Which one is valid constant in 'C'?
   (a) 2,000 (b) $40 (c) 15 76 (d) None of above
6. Which escape sequence will provide horizontal tab ?
   (a) '\0' (b) '\t' (c) '\n' (d) None of above
7. Which one is valid variable?
   (a) sum (b) temp (c) rate_of_int (d) All of above

8. Which qualifier should be used when we want to store only positive numbers ?
   (a) signed (b) unsigned (c) short (d) None of above
9. Which one is correct definition to define symbolic constant FLAG with value 1 ?
   (a) #define FLAG =1 (b) #define FLAG =1; (c) #define FLAG 1 (d) None of above
10. The file having extension .h is called as
   (a) Source file (b) Header file (c) Object file (d) Executable file
11. C is a _____ language.
   (a) High level (b) Middle level (c) Machine level (d) Low level
12. Which of the following is a correct statement
   (a) Variable name must start with underscore
   (b) Variable name must have digit
   (c) Variable name must have white space character
   (d) Keyword cannot be a variable name.
13. A declaration float a, b; occupies _____ of memory ?
   (a) 1 bytes (b) 4bytes (c) 8 bytes (d) 16 bytes
14. In C language a variable can not start with:
   (a) Special character except underscore (b) Number
   (c) Both a & b (d) Alphabet
15. Macro is used to
   (a) save memory (b) fast execution (c) Both a and b (d) None of above

: ANSWERS :

1. (b) **2.** (a) **3.** (d) **4.** (d) **5.** (d) **6.** (b) **7.** (d)
8. (b) **9.** (c) **10.** (b) **11.** (b) **12.** (d) **13.** (c) **14.** (c)
15. (c)

: EXERCISES :

1. List the features of 'C' language.
2. Explain the structure of 'C' program.
3. List and explain the fundamental data types of 'C' language.
4. Explain the statement: " 'C' is a middle level language".
5. What do you mean by constants in C language? Explain types of constants in detail.
6. Compare symbolic constants with enumerated data types.

: ANSWERS TO SELECTED EXERCISES :

5. What do you mean by constants in C language? Explain types of constants in detail.

Ans :

Constant is something whose value does not change throughout the program. C language supports followi types of constants.

- ✦ Numeric Constants
  - ⇒ Integer constant examples
    - → 15
    - → -345

⇒ Real constant examples
- → 0.004
- → -0.45
- → 5453.4

❖ Non-numeric Constants

⇒ Character constants are single character and enclosed in single quotes. Examples
- → 'B'
- → 'a'
- → '5'

⇒ String constant is a sequence of characters which are enclosed in double quotes. Examples
- → "Computer"
- → "Programming"
- → "123"

: SHORT QUESTIONS :

1. What are keywords?

⇒ Keywords are reserved words in a programming language. Meaning of keywords is fixed and can not be changed by the programmer.

2. Why comments are used in a program?

⇒ Comments are used in a program to improve readability of a program. The comment can be written between /* and */.

3. What are qualifiers in 'C'?

⇒ Qualifiers are words which are written before basic data types in 'C' to change the range of values supported by basic data types. Qualifiers in 'C' are : signed, unsigned, short and long.

4. 'C' is which type of language?

⇒ 'C' is middle level language; because it has assembly level as well as high level language features.

5. What is the extension of 'C' program?

⇒ Extension of 'C' program is .c .

6. What are enumerations?

⇒ They are a list of named integer-valued constants.

Example: enum color {black, orange, yellow, green, blue, violet};

The enum actually declares a type, and therefore can be type checked.

7. Is it possible to have more than one main() function in a C program ?

⇒ No, The function main() can appear only once. The program execution starts from main function.

❖

CHAPTER

4 Operators and Expressions

4.1 INTRODUCTION
4.2 ARITHMETIC OPERATORS
4.3 RELATIONAL OPERATORS
4.4 LOGICAL OPERATORS
4.5 ASSIGNMENT OPERATORS
4.6 INCREMENT AND DECREMENT OPERATORS
4.7 CONDITIONAL OPERATOR (TERNARY OPERATOR)
4.8 BITWISE OPERATORS
4.9 OTHER SPECIAL OPERATORS
4.10 PRECEDENCE AND ASSOCIATIVITY OF OPERATORS
4.11 TYPE CONVERSION
4.12 TYPE CASTING
4.13 HEADER FILES
4.14 PREPROCESSOR DIRECTIVES
4.15 SOLVED PROGRAMMING EXAMPLES
❖ SUMMARY
❖ MCQs
❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES
❖ SHORT QUESTIONS

4.1 INTRODUCTION :

Operators help in doing various operations. 'C' language supports rich set of operators. The operators can be divided into different categories as shown below.

- Arithmetic operators
- Logical operators
- Increment and Decrement operators
- Bitwise operators
- Relational operators
- Assignment operators
- Conditional operators
- Other special operators

The operators are used inside expressions. Expression is a formula having one or more operands and may have zero or more operands. The operand can be a variable, constant or a name of a function. For example, following is an expression involving two operands a and b with '+' operator.

```
a + b
```

4.2 ARITHMETIC OPERATORS :

Arithmetic operators are used to perform arithmetic operations. 'C' language supports following arithmetic operators.

Operator name	Meaning
+	Add
–	Subtract
*	Multiply
/	Divide
%	Modulo division (Remainder after division)

The 'C' language does not provide the operator for exponentiation. + and – operators can be used as unary operator also. Except % operator all arithmetic operators can be used with any type of numeric operands, while % operator can only be used with integer data type only.

Following program will clarify how arithmetic operators behave with different data type, particularly the use of / and % operator.

Program :

/* Program illustrating use of arithmetic operators. The numbers x and y are initialized in the program itself with x = 25 and y =4 */

```
#include  <stdio.h>
main()
{
    int  x=25;
    int  y=4;
    printf("%d  +  %d  =  %d\n",x,y,x+y);
    printf("%d  -  %d  =  %d\n",x,y,x-y);
    printf("%d  *  %d  =  %d\n",x,y,x*y);
    printf("%d  /  %d  =  %d\n",x,y,x/y);
    printf("%d  %%  %d  =  %d\n",x,y,x%y);
}
```

Output :

```
25  +  4  =  29
25  -  4  =  21
25  *  4  =  100
25  /  4  =  6
25  %  4  =  1
```

Explanation : First three operations are obvious. The division operation gives the answer 6 because variables x and y are integer variables, when we use / with integer operands the result will be integer num While, % operator produces the remainder after division of 25 by 4.

Following program explains how to get data from user using scanf() function.

Program :

```
/* Program illustarting use of arithmetic operators. This program uses scanf() function to get the number
from keyboard */
#include <stdio.h>
main()
{
    int x,y;
    printf("Enter the values of x and y\n");
    scanf("%d%d",&x,&y);
    printf("%d + %d = %d\n",x,y,x+y);
    printf("%d - %d = %d\n",x,y,x-y);
    printf("%d * %d = %d\n",x,y,x*y);
    printf("%d / %d = %d\n",x,y,x/y);
    printf("%d %% %d = %d\n",x,y,x%y);
}
```

Output :

```
Enter the values of x and y
-25  3
-25  +  3  =  -22
-25  -  3  =  -28
-25  *  3  =  -75
-25  /  3  =  -8
-25  %  3  =  -1
```

Look at the last line of output. Run the program again and try giving positive value of x and negative value for y and check the output.

Following program explains floating-point arithmetic.

Program :

```
/* Write a program to convert Fahrenheit temperature to centigrade.
This program explains floating-point arithmetic. The formula is c = 5/9*(f–32).
In the program, if we write the formula using 5/9 then, it will result into 0.
Because in integer arithmetic 5/9 =0.
To get the correct answer we should write the formula as c= 5*(f–32)/9 or
c= 5.0/9 * (f–32)  or c =5.0/9.0*(f–32)  so that integer gets upgraded to real. */
#include  <stdio.h>
#include  <conio.h>
main()
{
    float  fahr,cent;
    clrscr();
    printf("Give  the  value  of  temperature  in  Fahrenheit\n");
    scanf("%f",&fahr);
    cent  =  5*(fahr-32)/9;
    printf("Fahrenheit  temperature  =  %f\n",fahr);
    printf("Centigrade  temperature  =  %f\n",cent);
}
```

Output :

```
Give  the  value  of  temperature  in  fahrenhit
    100
Fahrenhit  temperature  =  100.000000
Centigrade  temperature  =  37.777779
```

Program :

```
/* Write a program to find sum and average of 4 integer numbers */
#include  <stdio.h>
#include  <conio.h>
main()
{
    int  n1,n2,n3,n4;
    int  sum  =0;
    float  avg=0;
    clrscr();
    printf("Give  four  integer  numbers\n");
    scanf("%d%d%d%d",&n1,&n2,&n3,&n4);
    sum  =  n1+n2+n3+n4;
    avg  =  sum/4.0;  /*  See  4.0  is  used  and  not  4.  We  can  not  write
    avg=sum/4  because  sum  and  4  are  integers.  So  answer  will  be
    truncated,  which  will  be  wrong  */
```

```
    printf("Sum of %d %d %d %d =%d\n",n1,n2,n3,n4,sum);
    printf("Average of %d %d %d %d =%f\n",n1,n2,n3,n4,avg);
}
```

Output :

```
Give four integer numbers
1 2 3 4
Sum of 1 2 3 4 = 10
Average of 1 2 3 4 = 2.500000
```

The precedence of arithmetic operators is

`*` `/` `%` first priority

`+` `-` second priority

If in an expression, the operators having same priority occur, then the evaluation is from left to right. S expressions written in brackets are evaluated first.

4.3 RELATIONAL OPERATORS :

Relational operators are used to compare variables or constants. In programming, relational operators a used to compare whether some variable value is higher, equal or lower with other variable. 'C' langua provides following relational operators.

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

Relational operators can be used in an expression. The operands of relational operators can be constant variables or expressions. The relational expression will have value either True (1) or False (0). Followi are some examples of use of relational operators. For example, if a=2, b =3 and c=5 then,

`a < b` (true)

`5> 3` (true)

`4 < 8` (true)

`a >5` (false)

`b< (a+c)` (true)

4.4 LOGICAL OPERATORS :

Sometimes in programming, we need to take certain action if some conditions are true or false. Logica operators help us to combine more than one condition, and based on the outcome certain steps are taken 'C' language supports following logical operators.

Operator name	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT (Negation)

Logical NOT is an unary operator. In an expression, we can use more than one logical operator. If more than one operator is used, ! (NOT) is evaluated first, then && (AND) and then || (OR). We can use parentheses to change the order of evaluation.

For example, if we have a=2, b=3 and c=5 then,

Expression	Value	Remark
a<b && c==5	true	both expression are true
! (5>3)	false	5>3 is true & negation of true is false
a< b || c>10	true	a<b is true which makes the expression true
(b> a)&&(c!=5)	false	c =5, so second condition false
(b<c||b>a)&&(c==5)	true	Both sub expression are true

4.5 ASSIGNMENT OPERATORS :

We have already used the assignment operator = in previous programs. 'C' language supports = assignment operator. It is used to assign a value to a variable. The syntax is

```
variablename = expression;
```

The expression can be a constant, variable name or any valid expression. In an expression involving arithmetic operators, assignment operator has the least precedence i.e +,-,*,/and % will be given more priority.

'C' language also supports the use of shorthand notation also. For example, the statement a=a+5 can be written using shorthand notation as a += 5. So, shorthand notation can be used for any statement whose form is

```
varname = varname operator expression;
```

into

```
varname operator= expression;
```

Use of shorthand notation makes your statement concise and program writing becomes faster particularly when the variable names are long in size.

Assignment operator	Shorthand
a = a +5;	a +=5;
a = a-5;	a -=5;
a = a*5;	a *=5;
a = a/5;	a /=5;
a = a%5; (assuming a as integer)	a %=5;

Program :

```
/* Program explaining short hand notations */
#include <stdio.h>
#include <conio.h>
main()
{
    int a;
    clrscr();
    printf("Give the value of a\n");
    scanf("%d",&a);
    a += 5;   /* a= a +5 */
```

```
    printf("a=  %d\n",a);
    a  -=  5;    /*  a  =  a-5  */
    printf("a=  %d\n",a);
    a  *=5; /*  a  =  a*  5  */
    printf("a=  %d\n",a);
    a  /=5; /*  a=  a  /5  */
    printf("a=  %d\n",a);
    a  %=  5;    /*  a=  a  %5  */
    printf("a=  %d\n",a);
}
```

Output :

```
Give  the  value  of  a
4
a=  9
a=  4
a=  20
a=  4
a=  4
```

4.6 INCREMENT AND DECREMENT OPERATORS :

Sometimes we need to increase or decrease a variable by one. We can do that by using assignment operator =. For example, the statement a = a +1 increments the value of variable a, same way the statement a= a-1 decrements the value of variable a.

'C' language provides special operators ++ (increment operator) and — (decrement operator) for doing this. These operators are unary i.e it requires only one operand. The operand can be written before or after the operator. If a is a variable, then ++a is called prefix increment while a++ is called postfix increment. Similarly, —a is called prefix decrement while a— is called postfix decrement.

If the ++ or — operator is used in an expression or assignment, then prefix notation and postfix notation give different values. One should use prefix or postfix notation carefully in an assignment or expression involving other variables.

Program :

```
/* Program showing use of increment ++ and decrement — operators */
#include  <stdio.h>
#include  <conio.h>
main()
{
    int  x=10;
    int  y;
    int  z=0;
    clrscr();
    x++;  /*  x  incremented  using  postfix  notation  x=11*/
    ++x;  /*  x  incremented  using  prefix  notation  x=12*/
```

```
    y = ++x; /* x incremented first and then assigned
        to y. y=13 x=13 */
    printf("Value of x=%d y=%d and z=%d\n",x,y,z);
    z= y--;/* y assigned to z first and then decremented
        z=13 y =12*/
    printf("Value of x=%d y=%d and z=%d\n",x,y,z);
}
```

Output :

```
Value of x=13 y=13 and z=0
Value of x=13 y=12 and z=13
```

If we execute following code,

```
int x = 10;
void main()
    {
    int x = 15,y;
    y = x++;
    printf("%d %d",++y, x++);
}
```

In above program,

Statement	Effect
int x = 10;	x=10
int x = 15,y;	x=15 (local value in main()) y =random value
y = x++;	y=15 and x=16
printf("%d %d",++y, x++);	y incremented first so(y=16) , x printed first & then incremented

So, output

16 16

4.7 CONDITIONAL OPERATOR (TERNARY OPERATOR) :

'C' language provides ? operator as a conditional operator. It is also known as a **ternary** operator. It is used as shown below.

```
expr1 ? expr2 :expr3;
```

where expr1 is a logical expression, logical expression can be TRUE or FALSE. expr2 and expr3 are expressions. expr1 is evaluated first, and depending on its value TRUE or FALSE, either expr2 or expr3 is evaluated. If expr1 is TRUE, then expr2 is executed, otherwise expr3 is executed.

Actually, use of ternary operator is an alternative use of if..else statement, which we will study later.

Program :

```
/* Write a program to find maximum and minimum of two numbers using ternary ? operator */
#include <stdio.h>
#include <conio.h>
main()
{
    int x,y,max,min;
    clrscr();
    printf("Give two integer numbers\n");
    scanf("%d%d",&x,&y);
    max = (x>y) ? x: y; /* x>y is a logical expression */
    min = (x<y) ? x: y; /* x<y is a logical expression */
    printf("maximum of %d and %d is = %d\n",x,y,max);
    printf("minimum of %d and %d is = %d\n",x,y,min);
}
```

Output :

```
Give two integer numbers
3 6
maximum of 3 and 6 is = 6
minimum of 3 and 6 is = 3
```

4.8 BITWISE OPERATORS :

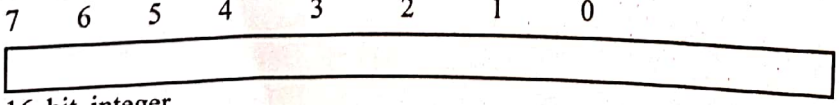
We know that internally, the data is represented in bits 0 and 1. 'C' language supports some operators w can perform at the bit level. These type of operations are normally done in assembly or mac level programming. But, 'C' language supports bit level manipulation also, that is why 'C' language is known as middle-level programming language.

Following table shows bit-wise operators with their meaning.

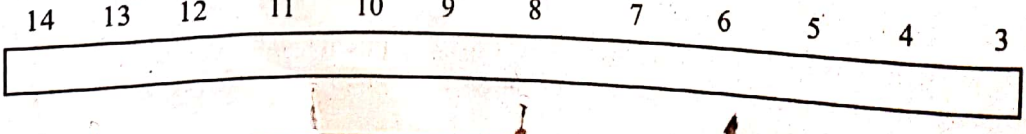
Operator name	Meaning
&	Bit-wise AND
|	Bit-wise OR
^	Bit-wise Exclusive OR (XOR)
<<	Left Shift
>>	Right Shift
~	Bit-wise 1's complement

One character requires 1 byte(8 bits) of storage, while integer requires 2 Bytes(16 bits). The individual are numbered as follows.

8 bit character



16 bit integer



The bit numbering is from right to left.

Following table explains the basic &, | and ^ operations.

Bit1	Bit2	Bit1 & Bit2	Bit1 | Bit2	Bit1 ^ Bit2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Let us understand with an example.

```
char  x=15;
char  y=8;
char  z;
```

x will be represented in memory as bit sequence 0 0 0 0 1 1 1 1

y will be represented in memory as bit sequence 0 0 0 0 1 0 0 0

Following figure explains the various operations on x and y. The numbers written in brackets are the decimal numbers.

```
x      (15)    0 0 0 0 1 1 1 1
y       (8)    0 0 0 0 1 0 0 0
-----------------------------
z=x&y  (8)     0 0 0 0 1 0 0 0
```

```
x      (15)    0 0 0 0 1 1 1 1
y       (8)    0 0 0 0 1 0 0 0
-----------------------------
z=x|y  (15)    0 0 0 0 1 1 1 1
```

```
x      (15)    0 0 0 0 1 1 1 1
y       (8)    0 0 0 0 1 0 0 0
-----------------------------
z = x^y (7)    0 0 0 0 0 1 1 1
```

```
y       (8)    0 0 0 0 1 0 0 0
z= ~y  (245)   1 1 1 1 0 1 1 1
```

The << and >> operators are used to shift the bits in left and right directions respectively. After the << or >> operator some integer number is written.

Example of <<

```
z = y << 1;
```

will shift the bits of y one bit in left side, and 0 is inserted from right side.

```
        ↓
y  (8)    0 0 0 0 1 0 0 0
z = y << 1(16) 0 0 0 1 0 0 0 0 ← inserted
```

So, we can see that the left shift by one bit multiplies the number by 2. If we left shift it by 2 bits, the number is multiplied by 4.

Example of >>

```
z = y >> 2;
```

will shift the bits of operand y by 2 bit positions towards right side, and 0 is inserted from left side.

```
y  (8)          0 0 0 0 1 0 0 0
z = y >>2 (2)   0 0 0 0 0 0 1 0
                ↑
                inserted
```

Each one bit shift on right divides the number by 2. We shifted 2 bits towards right so y divided by 4. Here, the answer may be truncated if the operand is not even number.

Program :

```
/* Write a program to multiply and divide the given number by 2 using bit-wise operators << and >> */
#include <stdio.h>
#include <conio.h>
main()
{
    int x;
    int mul,div;
    clrscr();
    printf("Give one integer number\n");
    scanf("%d",&x);
    mul = x << 1; /* left shift */
    div = x >> 1; /* right shift */
    printf("multiplication of %d by 2 = %d\n",x,mul);
    printf("division of %d by 2 = %d\n",x,div);
}
```

Output-1 :

```
Give one integer number
5
multiplication of 5 by 2 = 10
division of 5 by 2 = 2
```

Output-2 :

```
Give one integer number
8
multiplication of 8 by 2 = 16
division of 8 by 2 = 4
```

Explanation :

In the first output, input is number 5 (odd number), so division operation output is truncated, while in secon output, input is number 8(even number), so division operation produces correct answer 4.

4.9 OTHER SPECIAL OPERATORS :

'C' language provides other special operators. They are: comma operator, sizeof operator, arrow (->) operato dot (.) operator , * operator and & operator. The arrow (->) operator and dot (.) operator are membe selection operators used with a data structure called as structure, while * operator and & operator are normall used with pointer data type. We will see more detail about these 4 operators in detail in later chapters

Comma operator

(,) Comma operator is used to combine multiple statements. It is used to separate multiple expressions. I has the lowest precedence. It is mainly used in for loop. We will study for loop in later chapters.

The expressions which are separated by comma operator are evaluated from left to right i.e. associativity of comma operator is from left to right.

For example, the following statement

```
z = (x=5, x+5);
```

is equivalent to the statement sequence

```
x = 5;
z = x +5;
```

Program :

```
/* Program demonstrating comma operator (,) */
#include <stdio.h>
#include <conio.h>
main()
{
    int x,z;
    clrscr();
    z = (x=5 , x+5);   /* comma operator here */
    printf("Value of z= %d\n",z);
}
```

Output :

```
Value of z= 10
```

sizeof operator :

sizeof operator is used to find out the storage requirement of an operand in memory. It is an unary operator, which returns the size in bytes.

The syntax is sizeof(operand)

For example,

```
sizeof(float) returns the value 4
sizeof(int) returns the value 2
```

The statement sequence,

```
char c;
sizeof (c);
```

will return the value 1, because c is character type variable, and character requires one byte of storage.

4.10 PRECEDENCE AND ASSOCIATIVITY OF OPERATORS :

When arithmetic expressions are evaluated, the answer of that expression depends on the value of operands, the operators used and also on the precedence and associativity of the operators used. We have already seen what we mean by precedence of the operators, when we studied arithmetic and other operators. In 'C', the operators having the same precedence are evaluated from left to right or right to left depending upon the level at which these operators belong.

For arithmetic operators,

Operators	Precedence	Associativity
* / %	first priority	Left to right
+ -	second priority	Left to right

Precedence and associativity of all operators is shown in following table.

Operators	Associativity
( ) [ ] -> :: .	Left to right
! ~ ++ -- + - * sizeof	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
?:	Left to right
= += *= -= %= ^= != <<= >>=	Right to left
,	Left to right

From the associativity table, it is clear that unary operators like (+, -, *, sizeof, ! etc) have right to associativity. Similarly, the assignment operators such as (=, *=, +=, -=, != etc) have right to left associati

Let us consider following example,

Assume a=6, b=4 and c=9

For the statement, `z = b-c *a;`

Will be evaluated as z = 4 – 9 * 6 = 4 – 54 = -50.

In the above expression – and * are two operators. The * operator has higher precedence than – oper so c * a will be evaluated first, then + will be evaluated.

The expression,

5 -3 * 4 +6 /4 will be evaluated as 3 *4 first, then 6/4,then – operation and at last + operation as sh below.

5–3 * 4 + 6 /4

↓

5 – 12 + 6/4

↓

5 – 12 + 1

↓

–7 +1

↓

–6

In an expression, we can use the parenthesis to change the order of evaluation of sub expressions. example, in the expression 4 * (3 +6) – (4 +3) the order of evaluation will be first (3 +6), then (4 then 4 * result of (3+6) and finally the – operation will be done. So the answer will be 29.

4.11 TYPE CONVERSION :

Whenever an expression involves two different types of operands, 'C' language applies the type convers rules to evaluate an expression. At a time only one operator under consideration is taken. If the operat

are of different type, then the operand with a lower type is upgraded to the higher type and then the operation is performed.

Program :

```
/* Program demonstrating type conversion */
#include <stdio.h>
#include <conio.h>
main()
{
    int a,b;
    float c;
    double d;
    a=5;
    c= 5.5;
    d = 4.0;
    clrscr();
    b = a *c +d /10;
    printf("value of b = %d\n",b);
}
```

Output :

```
value of b = 27
```

Explanation :

In the statement b = a *c +d /10,

a*c will be done first, here **a** will be upgraded to float because other operand **c** is float.

So, a*c will evaluate to 27.5. Then, d/10 will be evaluated, 10 will be converted into 10.0 (double) because **d** is double. So, d/10 will evaluate to 0.4. Then 27.5 + 0.4 evaluates to 27.9. This value is assigned to variable **b**, which is integer, so truncated value of 27.9 will be the value of b i.e 27 will be assigned to **b**.

In the case of assignment,

- If float is assigned to integer, then fractional part is truncated as shown in above program.
- If double is assigned to float, then rounding takes place.
- If long int is assigned to integer, then additional bits are dropped, recall that long int requires 4 bytes while int requires only 2 bytes.

4.12 TYPE CASTING :

When user needs the type conversion explicitly, then type casting is used. Remember that type conversion is automatic while type casting is explicitly specified by the programmer in the program.

The type casting can be specified in following form

```
(typename) expression;
```

Here, typename is the name of data type we want to convert the expression to. The converted value is used during evaluation of expression only, it does not change the basic data type of operand(s) of an expression.

For example, float(21) converts 21 integer to float 21.0

Program :

```
/* Program demonstrating type casting */
#include <stdio.h>
#include <conio.h>
main()
{
    int sum =47;
    int n = 10;
    float avg;
    clrscr();
    avg = sum/n;              /*avg without type cast */
    printf("avg=sum/n      = %f\n",avg);
    avg = (float)sum/n; /*avg with type cast on sum */
    printf("avg=(float)sum/n    = %f\n", avg);
    avg = (float)(sum/n);
                /*avg with type cast on (sum/n) */
    printf("avg=(float)(sum/n)  = %f\n", avg);
}
```

Output :

```
avg=sum/n              = 4.000000
avg=(float)sum/n       = 4.700000
avg=(float)(sum/n)     = 4.000000
```

Explanation :

In the first line of output, integer arithmetic takes place, while in second sum which is 47 is converted 47.0, so automatically i.e type conversion takes place and 10 becomes 10.0. So, floating-point arithmetic t place. While in the last line of output, float type casting takes place on sum/n which is 4, converted float becomes 4.0.

4.13 HEADER FILES :

The files included at the beginning of 'C' program and having an extension .h are called as header f The content of these files go at the head of the program, in which the header file is included. In 'C' langu the prototypes of all the library functions are categorized according to their function in header files. example, stdio.h header file contains all the library functions for standard Input- Output functionality, is why the name stdio.h where,

std stands for standard, **i** stands for Input and **o** stands for Output.

Same way, math.h header file contains prototypes of all mathematical library functions.

For example, **sqrt()** function for finding the square root of a number, **pow()** function for finding power a number.

Following is the list of mostly used header files

- stdio.h contains standard input-output functions.
- math.h contains mathematical functions.
- conio.h contains console input-output functions.

- string.h contains string processing functions.
- ctype.h contains character type checking functions.

The advantage of using the header files is that we do not have to write the code again and again in each program. Whatever library functions we are using, we have to just include the name of the header file in which the prototype of the function is defined. If we forget to include the header file in the program, and try to use the library function defined in it, then compiler will generate an error message. Detail of header files can be found in the reference manual.

4.14 PREPROCESSOR DIRECTIVES :

Preprocessor is a program that processes the source program before the control is given to the compiler. Preprocessor understands some commands known as preprocessor directives. The directive begins with symbol #. There are many directives which the preprocessor understands. We have already used the directives **include** and **define**.

For example,

```
#include  <stdio.h>
#  define  PI  3.1415
```

Include directive includes the file mentioned after it, while define directive defines a symbolic constant. Following program uses symbolic constants.

Program :

```
/* Write a program that demonstrates symbolic constants */
#include  <stdio.h>
#include  <conio.h>
#define  cube(x)  (x*x*x)
void  main()
{
    int  ans,a;
    clrscr();
    a=3;
    ans  =  cube(a);
    printf("Answer  =  %d\n",ans);
}
```

Output :

```
Answer  =  27
```

4.15 SOLVED PROGRAMMING EXAMPLES :

Program :

```
/* Write a program to exchange two variables. */
#include  <stdio.h>
#include  <conio.h>
void  main()
{
```

```
    int  x,y,temp;
    clrscr();
    x=5;
    y=3;
        printf("Before  exchange  x  =%d  and  y  =%d\n",x,y);
    temp  =x;
    x  =  y;
    y  =  temp;
    printf("After  exchange    x  =%d  and  y  =%d\n",x,y);
}
```

Output :

```
Before  exchange  x=5  and  y=3
After  exchange    x=3  and  y=5
```

Program :

```
/* Write a program to exchange two variables without use of third variable */
#include  <stdio.h>
#include  <conio.h>
void  main()
{
    int  x,y;
    clrscr();
    x=5;
    y=3;
    printf("Before  exchange  x  =%d  and  y  =%d\n",x,y);
    x=x+y;
    y=x-y;
    x=x-y;
    printf("After  exchange    x  =%d  and  y  =%d\n",x,y);
}
```

Output :

```
Before  exchange  x  =5  and  y  =3
After  exchange    x  =3  and  y  =5
```

Program :

```
/* Write a program to calculate area, perimeter and diagonal length of rectangle of length l and width
b. Area = l *b  Perimeter = 2*l + 2*b
Diagonal Length = sqrt(l*l +b*b) */
#include  <stdio.h>
#include  <conio.h>
#include  <math.h>
```

```
void main()
{
    int l,b;
    int area, peri;
    float d;
    clrscr();
    printf("Give length and width of rectangle\n");
    scanf("%d%d",&l,&b);
    area = l*b;
    peri = 2*l + 2*b;
    d = sqrt(l*l + b*b);
        /* find square root of number in brackets */
    printf("Area = %d Perimeter = %d Diagonal Length = %f\n",area,
    peri,d);
}
```

Output :

```
Give length and width of rectangle
3 5
Area = 15 Perimeter = 16 Diagonal Length = 5.830952
```

Program :

/* Write a program to evaluate the polynomial 3x^3 - 4x +9 */

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    float x, ans=0;
    clrscr();
    printf("Give value of x\n");
    scanf("%f",&x);
    ans = ans + 3 * x*x*x;
    ans = ans - 4*x;
    ans = ans +9;
    printf("Value of polynomial is = %6.2f\n",ans);
}
```

Output :

```
Give value of x
2
Value of polynomial is =    25.00
```

: SUMMARY :

- 'C' language supports rich set of **operators which can be categorized as**: Arithmetic operators, Relation operators, Logical operators, Assignment operators, Increment and Decrement operators, Conditional operato Bitwise operators, other special operators.
- **% is a modulo division** operator. It gives remainder after division.
- **Ternary operator ?** is a conditional operator. It is an alternative to if..else statement.
- Result of arithmetic expression evaluation depends on precedence and associativity of operators in an expressio **Precedence** explains priority of operators and **associativity** explains evaluation from left to right or rig to left.
- In an expression, if operands are of different types, the lower type is upgraded to higher type. This is call as **Type conversion** and is automatic.
- Type conversion is automatic, while **Type Casting** is explicitly specified by the programmer. It is specified a

  (typename) expression;
- **Header files** have .h extension. Header file contain the definition of various library functions, which a readily available to programmer for use.
- **Preprocessor** is a program which pre–processes the directives such as **include** and **define** in source co file. The directives begin with # symbol..
- **include** preprocessor directive is used to include header file and other files in a program.
- **define** preprocessor directive is used to define symbolic constants in a program.

: MCQs :

1. Which operator can be used only with integer numbers?
   (a) + (b) – (c) % (d) *
2. Which operator is used to find remainder after division?
   (a) * (b) / (c) – (d) %
3. What is the value of expression 5/9 in 'C'?
   (a) 0.556 (b) 0 (c) 1 (d) None of above
4. Which operator has higher priority while evaluating arithmetic expression?
   (a) + (b) –
   (c) % (d) All have same priority
5. What is the symbol for logical OR operator?
   (a) != (b) & (c) && (d) ||
6. Which operator is unary?
   (a) –– (b) * (c) ++ (d) Both a and c
7. What is the output of following code?

```
void main()
        {
        int x = 15,y;
        y = x++;
        printf("%d %d",++y, x++);
    }
```

   (a) 15 16 (b) 16 16 (c) 16 15 (d) None of above

8. Which one is correct use of ternary operator?
   (a) expr1 ? expr2 : expr3;
   (b) expr1 ? expr2::expr3;
   (c) expr1 = expr2 :expr3;
   (d) expr1 : expr2 ? expr3;

9. What is value of z for following code?

```
int  y=8;
z=  y  >>  2;
```

   (a) 8 (b) 2 (c) 4 (d) None of above

10. Which operators have right to left associativity?
   (a) Arithmetic (b) Relational (c) Assignment (d) None of above

11. 5 –3 * 4 +6 /4 evaluates to
   (a) 5 (b) –6 (c) 3.5 (d) None of above

12. 4 * (3 +6) – (4 +3) evaluates to
   (a) 8 (b) 27 (c) 11 (d) None of above

13. Type conversion is
   (a) Automatic (b) Manual (c) Both a and b (d) None of above

14. To round off x, which is a float, to an int value, Which one is correct way?
   (a) y = (int)(x + 0.5)
   (b) y = int(x + 0.5)
   (c) y = (int)x + 0.5
   (d) y = (int)((int)x + 0.5)

15. Which bitwise operator is suitable for checking whether a particular bit is on or off?
   (a) && (b) & (c) | (d) ||

16. Ternary operator operates on how many operators?
   (a) 2 (b) 1 (c) 3 (d) 4

17. Which conversion character is associated with short integer?
   (a) %c (b) %hd (c) %d (d) %f

18. What will be printed if we type the statement printf("%d\n",'d');
   (a) 0 (b) 100 (c) error (d) d

19. Which header file is essential for using printf() function ?
   (a) text.h (b) strings.h (c) stdio.h d) strcmp.h

20. Which of the following is a symbol for AND operator?
   (a) || (b) & (c) && (d) $$

21. What will be the output of the following code

```
{
int  x=10,  y=15;
x  =  x++;
y  =  ++y;
printf("%d  %d\n",  x,y);
}
```

   (a) 10,15 (b) 10,16 (c) 11,16 (d) 11,15

22. printf() belongs to which library of c
   (a) stdlib.h (b) stdio.h (c) stdout.h (d) stdoutput.h

23. Which of the following is ternary operator?
   (a) ?? (b) ?: (c) :? (d) :

24. Which header file is essential for using scanf() function?
(a) ctype.h (b) string.h (c) conio.h (d) stdio.h

: ANSWERS :

1.	(c)	2.	(d)	3.	(b)	4.	(d)	5.	(d)	6.	(d)	7.	(b)
8.	(a)	9.	(b)	10.	(c)	11.	(b)	12.	(b)	13.	(a)	14.	(a)
15.	(b)	16.	(c)	17.	(b)	18.	(b)	19.	(c)	20.	(c)	21.	(c)
22.	(b)	23.	(a)	24.	(d)								

: EXERCISES :

1. List the categories (various types) of operators in 'C' language.
2. What is conditional operator ? Write a small program using conditional operator.
3. Explain with example working of >> and << operators.
4. What is sizeof operator? How it is useful?
5. What is type conversion? Explain implicit type conversion and explicit type conversion with example.
6. What is header file? How they are useful? Write the name of any 4 header files.
7. What is a preprocessor directive? Explain the use of preprocessor directives in a simple program to calculate the area of circle.
8. Explain operator precedence and associativity.
9. What will be the output of following program segment?

```
int  i=2,  j=4,k;
float  a,b;
k  =  i/j*j;
a  =  i/j*j;
b  =  j/i  *i;
printf("%d  %d  %f",  k,a,b);
```

10. Find out incorrect variable declaration form following:
(i) num1 (ii) Area of circle (iii) 6PC (iv) num_2
11. Find out errors, if any in following code statements.

```
int  a,b=0;
float  x;
int  float;
double  p,q;
```

12. What will be the output of following code?

```
int  b,  a=5;
a++;
b=  a++;
b++;
b  =  ++a;
printf("%d%d",a,b);
```

13. Write output of following expressions:
(i) 21 / (int) 3.5 (ii) 45 %10 -4 +4 (iii) 25 % 4 /4

: ANSWERS TO SELECTED EXERCISES :

1. List the categories (various types) of operators in 'C' language.

Ans. : 'C' language supports rich set of operators. The operators can be divided into different categories as shown below.

- Arithmetic operators
  + - */%
- Relational operators
  = =(equality) != < > <= >=
- Logical operators
  &&(Logical AND) | | (Logical OR) !(Logical NOT)
- Assignment operators
  = (assignment)
- Increment and Decrement operators
  ++ (increment) — (decrement)
- Conditional operators (Ternary operator)
  ?
  expr1 ? expr2 :expr3;

where expr1 is a logical expression, logical expression can be TRUE or FALSE. expr2 and expr3 are expressions. expr1 is evaluated first, and depending on its value TRUE or FALSE, either expr2 or expr3 is evaluated. If expr1 is TRUE, then expr2 is executed, otherwise expr3 is executed.

- Bitwise operators
  & | ^ << >> ~
- Other special operators
  , (comma) . (dot) * (indirection) & (address of)

2. What is conditional operator? Write a small program using conditional operator.

Ans. : Conditional operators is also known as ternary operator ?

For example, the statement,

expr1 ? expr2 :expr3;

where expr1 is a logical expression, logical expression can be TRUE or FALSE. expr2 and expr3 are expressions. expr1 is evaluated first, and depending on its value TRUE or FALSE, either expr2 or expr3 is evaluated. If expr1 is TRUE, then expr2 is executed, otherwise expr3 is executed.

Following program finds maximum of two numbers using ? operator

```
#include <stdio.h>
main()
{
    int x,y,max;
    clrscr();
    printf("Give two integer numbers\n");
    scanf("%d%d",&x,&y);
    max = (x>y) ? x: y; /* x>y is a logical expression */
    printf("maximum of %d and %d is = %d\n",x,y,max);
}
```

5. What is type conversion? Explain implicit type conversion and explicit type conversion with exampl

Ans : Whenever an expression involves two different types of operands, 'C' language applies the type conversio rules to evaluate an expression. At a time only one operator under consideration is taken. If the operand are of different type, then the operand with a lower type is upgraded to the higher type and then th operation is performed. So, type conversion is implicit i.e automatic.

For example,

```
int  a,b;
float  c;
double  d;
.
.
.
b  =  a  *c  +d  /10;
```

In the above example, a*c will done first, and a will upgraded to float because 'c' is float type. Then d/10 will evaluated, 10 will be upgraded to 10.0 because 'd' is double.

Explicit type conversion is also known as type casting. Type casting is explicitly specified by the programmer in the program.

Explicit type conversion is specified in following form:

(typename) expression;

For example, float(10) converts integer 10 to float 10.0

8. Explain operator precedence and associativity.

Ans : When arithmetic expressions are evaluated, the answer of that expression depends on the value of operands, the operators used and also on the precedence and associativity of the operators used.

Precedence tells which operator has higher priority than other. For example, * / % operators have higher precedence than + - operators.

While associativity tells in an expression if the operators having same priority occurs then which operator will be evaluated first? Some operators have left to right associativity while some have right to left associativity. Following table shows partial table for associativity. The shortcut operators have right to left associativity.

Operators	Associativity
* / %	Left to right
+ -	Left to right
= += *= -= %= ^= != <<= >>=	Right to left

: SHORT QUESTIONS :

1. Which arithmetic operator gives remainder after division?

⇒ %

2. Which relational operator is used to check equality?

⇒ = =

3. Which operator is known as ternary operator? What is its other name?

⇒ ? is ternary operator. The other name of ternary operator is conditional operator.

4. What is the use of sizeof operator?

⇒ The sizeof operator is used to find the storage requirements of an operand in memory. It returns the size in bytes. Its syntax is: sizeof(operand)

5. How type conversion and type casting are different?

⇒ Type conversion is automatic while type casting is explicitly specified by programmer in the program. Type casting syntax is : (typename) expression;

6. What are header files?

⇒ The files included at the beginning of a program and having an extension .h are called header files. They are included using preprocessor directive include. For example, #include <stdio.h>

7. What are macros?

⇒ Macros are abbreviations for lengthy and frequently used statements. When a macro is called the entire code is substituted by a single line though the macro definition is of several lines.

8. What is a preprocessor?

⇒ It is normally a part of the compiler which preprocesses the source code using its directives called as preprocessor directives. In 'C' language preprocessor directives begin with symbol #.

❖

CHAPTER

12 Dynamic Memory Allocation

12.1 INTRODUCTION

12.2 DYNAMIC MEMORY ALLOCATION

12.3 ALLOCATING A BLOCK OF MEMORY – malloc() FUNCTION

12.4 ALLOCATING MULTIPLE BLOCKS OF MEMORY – calloc() FUNCTION

12.5 RELEASING THE USED SPACE: free() FUNCTION

12.6 LINKED LISTS

- SUMMARY
- MCQs
- EXERCISES AND ANSWERS TO SELECTED EXERCISES
- SHORT QUESTIONS

12.1 INTRODUCTION :

In previous chapters, we have seen different programs using a category of memory allocation known as – **static memory allocation**, where we inform to the compiler in advance what type of the variables we are going to use in our program and what will be the storage requirement. This type of static memory allocation method works fine, when we know the memory requirement i.e number of variables and their size etc. in advance before writing the program, but this method fails or sometimes not optimized when we do not know the memory requirements at the time of writing the program. For example, suppose we declare the array as **int a[5]**, the program will work correctly as long as number of elements are less than or equal to 5. If we increase the size of an array, let us say to 1000 and use only 10 elements, then the remaining storage space (1000 –10 =990) is wasted.

So, for programs where the number of data items required go on changing during the execution of a program, where list grows or shrinks can be handled more efficiently by dynamic data structures using the dynamic memory management techniques. Dynamic data structures provide more flexibility in manipulating data at run time in the operation of add, delete data items.

12.2 DYNAMIC MEMORY ALLOCATION :

Dynamic memory allocation techniques allow us to allocate memory and release unwanted memory at runtime. 'C' language provides a way to solve the wastage of storage by providing the other method of memory allocation known as **dynamic memory allocation**. In this method, the storage space requirements are decided at run–time of the program and not at compile time. Whenever a memory is required to store data, the programmer can get a block of memory directly from the operating system from the free memory area called as **heap**. So, we can say that dynamic memory allocation enables us to create data types and structures of any length and size which suits our program requirements. 'C' language provides memory management functions in **malloc.h** and **stdlib.h** header file.

The important memory allocation functions are explained in following table :

Function	What it does
malloc	Allocates the memory block of requested size and returns a pointer to the first byte of block
calloc	Allocates the multiple memory blocks each of same size and also initializes all bytes to zero
free	Frees the previously allocated block from program back to memory
realloc	Modifies the size of previously allocated memory block
sizeof	Returns the number of bytes required to store a specific data type or variable

Note: sizeof() is not a memory allocation function, but listed here because it is mostly used with memory allocation functions.

The memory allocation functions allocate a memory to the program from a memory area which is not used for any other purpose. It should be noted that some portion of memory is used by program itself, some for storing local variables, global variables. The remaining free area is called as heap, whose size goes on changing as the program executes. If the memory allocation function is unable to find enough area from heap it returns a value NULL (it means the function was unsuccessful in allocating memory block).

12.2.1 Advantages of dynamic memory allocation over static memory allocation :

- Memory allocation at run–time.
- No wastage of memory.
- The memory blocks which are no longer required can be made free.
- Faster speed.
- Data can be rearranged efficiently.

12.3 ALLOCATING A BLOCK OF MEMORY – malloc() FUNCTION :

We know what is dynamic memory allocation, let us now understand what the malloc() function does and how it can be used.

The prototype of malloc() function is:

void *malloc(int);

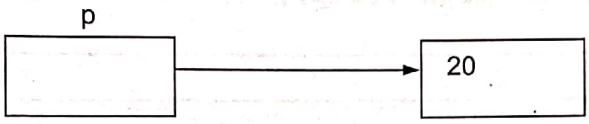
So, input to malloc is integer number which is the required number of bytes of a block, while it returns a pointer to a memory block, which is a pointer to void. As the return type is a pointer to void, we need to type cast to appropriate type. Remember, type cast is written in brackets before the name of variable or function. So, the use will be like

ptr = (cast_type *) malloc(size);

Examples :

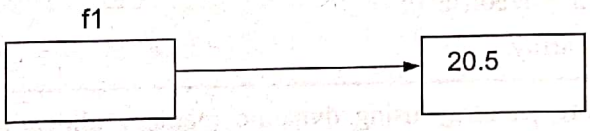
- For storing integer:

```
int *p;
p = (int *) malloc(sizeof(int));  /* p points to the block allocated by malloc */
  *p=20;  /* store value 20 in block allocated */
```



- For storing float:

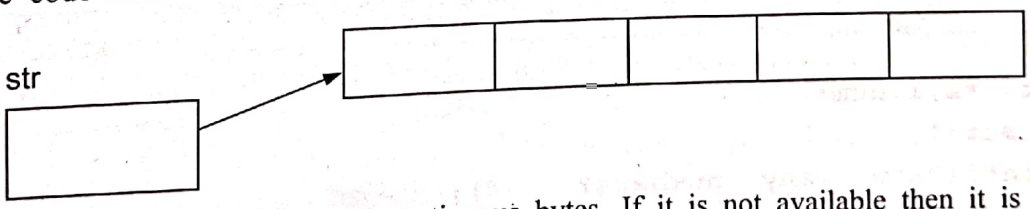
```
float *fl;
fl = (float *) malloc(sizeof(float));
*fl=20.5;
```



- For storing array:

```
char *str;
str = (char *) malloc (5 *sizeof(char));
```

Above code allocates a block for storage of 5 character size array.



Function malloc() allocates a block of contiguous bytes. If it is not available then it is returns a NULL. So, before proceeding we should check what is the return value of malloc().

Following program explains how array can be stored dynamically. It also checks for NULL value returned by malloc() function. Also note that the array is not declared, only pointer to the character is declared. Required memory block is requested from the memory block using malloc() function.

```
/* Demonstrate the use of malloc() for storing a character array.
*/
#include <stdio.h>
main()
{
```

```
    char  *str,*temp;
    int  i;
    clrscr();
    str  =  (char  *)  malloc  (5  *sizeof(char));
    if  (str  ==  NULL)
        {printf("Error  in  allocation\n");
        exit(1);
        }
    temp  =  str;          /*  store  address  in  temp,  because  initial
                                value  of  str  will  change*/

    for(i=0;i<4;i++)
        {
        *str  =  65  +i;
        str++;
        }
    *str  =  NULL;
    printf("String  =  %s\n",temp);
}
```

Output :

```
String  =  ABCD
```

Following code can be used for storing an integer array

```
int  *ptr;
ptr = (int *) malloc (n *sizeof(int));
```

where, n is the size of an array.

/* Write a program to store an array using dynamic memory allocation and sort it using function */

```
#include  <stdio.h>
#include  <malloc.h>
void  sort(  int  *a,  int  n);
main()
{
    int  *s,i,num;
    clrscr();
    printf("How  many  numbers?    ");
    scanf("%d",&num);
    s  =  (int  *)  malloc  (num  *  sizeof(int));
    /*  request  block  of  memory*/
    if  (s  ==  NULL)
        {printf("Error  in  allocation\n");
        exit  (1);
        }
    printf("Enter  numbers\n");
    for  (i=1;i<=num;i++)
        {
```

```
        scanf("%d",s);    /* store the data in memory block provided
*/
        s++;
        }
    s  =  s-num; /*  Point  s  to  the  beginning
    sort(s,num);     /*  Call  function  to  sort  data  of  memory  block  */
    printf("Sorted  data  \n");
    for(i=1;i<=num;i++)          /*  Print  sorted  data  */
        {
        printf("%d  "  ,*s);
        s++;
        }
}
void  sort(int  *a,  int  n)
    {
    int  i,j,temp;
    for(i=0;i<n-1;i++)
        {
        for(j=i+1;j<n;j++)
            {
                if  (*(a+i)  >  *(a+j))    /*  exchange
                                    numbers  if  out  of  order  */
                {
                temp  =  *(a+i);
                *(a+i)  =  *(a+j);
                *(a+j)  =  temp;
                }
            }
        }
    }
```

Output :

```
How  many  numbers?
5
Enter  numbers
45
64
34
49
11
Sorted  data
11  34  45  49  64
```

12.4 ALLOCATING MULTIPLE BLOCKS OF MEMORY – calloc() FUNCTION :

Function calloc() is normally used for allocating memory for derived data types like array and structure. As mentioned earlier calloc() allocates multiple blocks each of same size and initializes them with zero. The prototype of calloc() function is:

```
void  *calloc(int n, int size);
```

As in the case of malloc, we have to use type cast here because it also returns a pointer to void. So, the use will be like

```
ptr =  (cast_type *) calloc(n, size);
```

The above statement allocates contiguous memory space of n blocks each of size bytes initialized to zero and pointer to the first byte of the allocated memory is returned. If it is unsuccessful, it returns NULL value.

Example:

An int array 'a' of 5 elements can be allocated as:

```
int  *a = (int *) calloc (5, sizeof (int));
```

We can do the same thing using malloc() function as:

```
int  *a = (int *) malloc (sizeof (int) * 5);
```

12.4.1 Reallocating using realloc() :

It may happen after allocation of a block, we came to know that the allocated size is not proper, we may reduce or increase the size of memory block allocated. This function allocates a new memory space. We can change the size of memory block allocated previously using malloc() or calloc() function with a function known as realloc().

The prototype is:

```
void * realloc (void * ptr, int size);
```

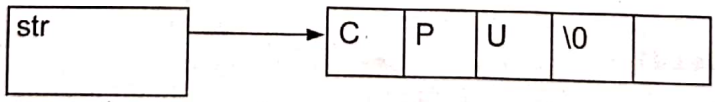
It returns a pointer to void, so it also should be type cast to appropriate type.

For example,

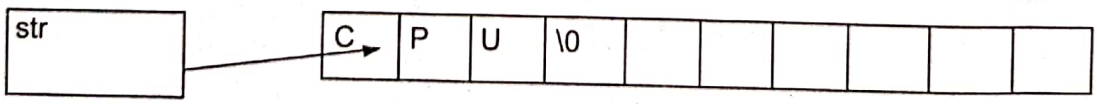
```
char *str;
str = (char *) malloc (5 *sizeof(char));
stcrpy(str,"CPU");
str = (char *) realloc (str, 10 * sizeof(char);
```

will reallocate a new block of 10 character array and pointer to the first byte is again stored in variable 'str'. The old data will be preserved and stored safely in the newly allocated memory space.

Before reallocation :



After reallocation :



12.5 RELEASING THE USED SPACE: free() FUNCTION :

In dynamic memory allocation of memory, it is the duty of the programmer to release the memory blocks which are no longer needed in the program. The release of memory is important particularly when the storage space is limited. 'C' language supports a function which releases the memory which was earlier allocated. The prototype is

```
void free ( * ptr);
```

Here, ptr is a pointer to the allocated memory block earlier, which we want to release now. We are not releasing a pointer but memory block it points to. Usage of invalid pointer may create problems and lead to system crash also. We cannot release individual elements of memory block allocated using calloc() function, instead we release the whole block by passing a pointer to the first byte of a block.

Memory leakage occurs when memory allocated is not released even though it is not required. It is basically wastage of memory resource. This leads to less memory available for other programs and sometimes results in slowdown or system crash problem. So, it is a normal practice to release the memory block which is no longer required by using free() function.

12.6 LINKED LISTS :

We know that list is a collection of items referred sequentially. The list can be stored in memory physically in two different ways – sequentially as it is in array. The other way of storing list in memory is to use structure and pointers so that the elements are not necessarily sequentially stored but their ordering is taken care by the use of pointers. This type of list is known as linked list where in the ordering is shown by the pointer from one node to the next node. Figure 12.1 shows linked list of nodes having two fields, where first one is the data and second one is a pointer which points to next node.

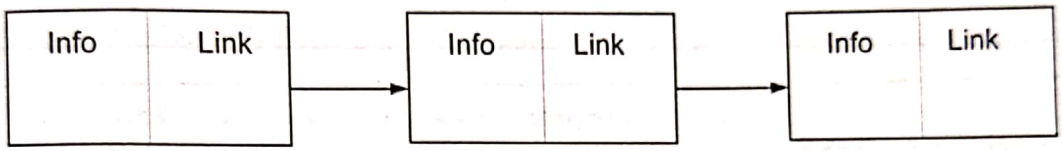


Figure 12.1 Singly linked list

Element of a linked can be created using a structure such as:

```
struct node
{
int info;
struct node *link;
}
```

The element name is node having first field as info and second field link is a pointer which points to other node. The first field is taken as integer for simplicity but it can be any valid data type. As the member field points to another data item of same type structure it is also called as self referential structure.

Other types of linked list are circular linked list, doubly linked list and circular doubly linked list. Some of the operations on linked list are:

- Create a linked list
- Insert an element
- Print the elements of a list
- Traverse a list
- Delete an element
- Count the elements of a list

12.7 SOLVED PROGRAMMING EXAMPLES :

/* Write a program that uses an array of integer values. The size of array will be specified at run time interactively. */

```
#include  <stdio.h>
#include  <stdlib.h>
main()
{
    int  *ptr,*a;
    int  size;
    clrscr();
```

```
    printf("Give size of array\n");
    scanf ("%d", &size);
    a = (int *) malloc (size *sizeof(int));
    /* Get block of memory */
    if (a == NULL)      /* Check for failure */
    {
        printf("Unable to get memory block\n");
        exit(1);
    }
    printf("Give array values one by one\n");
    for( ptr =a; ptr < a+size; ptr++)
        scanf("%d",ptr);
    printf("\n The values are\n");
    for( ptr =a; ptr < a+size; ptr++)
        printf("%d is stored at location %u\n",*ptr,ptr);
    free(a);    /* Release memory */
}
```

Output :

```
Give size of array
6
Give array values one by one
2
44
3
56
77
33
The values are
2 is stored at location 1952
44 is stored at location 1954
3 is stored at location 1956
56 is stored at location 1958
77 is stored at location 1960
33 is stored at location 1962
```

/* Write a program to store a character string in a memory block created using malloc() function and then modify it to store a larger size string
*/

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char *str;
    int size;
    clrscr();
    printf("Give size of string\n");
    scanf ("%d", &size);
```

```
    flushall();     /* Flush buffer */
    str = (char *) malloc (size *sizeof(char));
                                        /* Get memory block */
    if (str == NULL)
    {
        printf("Unable to get memory block\n");
        exit(1);
    }
    printf("Give any string upto %d characters\n",size);
    gets(str);
    printf("Given string is: %s\n",str);
    printf("Give new increased size of string\n");
    scanf ("%d", &size);
    flushall();/* Flush buffer */
    str = (char *) realloc (str, size *sizeof(char)); /* reallocation
for new size */
    if (str == NULL)
    {
        printf("Unable to get memory block\n");
        exit(1);
    }
    printf("Newly allocated block still contains old string %s\n",str);
    printf("Give any string upto %d characters\n",size);
    gets(str);
    printf("Given string is: %s",str);
}
```

Output :

```
Give size of string
7
Give any string upto 7 characters
surat
Given string is: surat
Give new increased size of string
12
Newly allocated block still contains old string surat
Give any string upto 12 characters
gandhinagar
Given string is: gandhinagar
```

/* Write a program to create a singly linked list and print the elements */

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *link;
```

```
};
main()
{
    struct node *first,*prev;
        /* first points to first node of list, prev is a temporary
pointer */ struct node *newn;
        /* newn is pointer which points to newly allocated node by
malloc() */
    int num;
    clrscr();
    first = NULL;    /*initially list is empty */
    printf("Enter numbers. Give last number as -1\n");
    scanf ("%d", &num);
    while (num != -1)
        {
        newn = (struct node *) malloc(sizeof(struct node));
        if (newn == NULL)
            {
            printf("Unable to get memory\n");
            exit(1);
            }
        newn -> info = num; /*store data in structure */
        newn -> link = NULL;    /* link field set to NULL if it
last node */
        if (first == NULL)
            first = newn; /* Very first node of list */
        else
            prev -> link = newn; /* not the first node */
        prev = newn;
        scanf("%d", &num);
        }
    printf("Linked list elements are\n");
    prev = first; /* start printing from first node */
    while (prev != NULL)
        {
        printf("%d\n", prev ->info);
        prev = prev ->link;
        }
}
```

Output :

```
Enter numbers. Give last number as -1
3
4
5
2
32
-1
```

```
Linked list elements are
3
4
5
2
32
```

: SUMMARY :

- **Static memory allocation** is a method of writing programs, where we inform to the compiler in advance what type of the variables we are going to use in our program and what will be the storage requirement. This method fails in solving real life complex problems.
- **Dynamic data structures** are data structures where the number of data items required go on changing during the execution of a program, where list grows or shrinks can be handled more efficiently by using the dynamic memory management techniques.
- **Dynamic memory allocation** techniques allow us to allocate memory and release unwanted memory at runtime. In this method, the storage space requirements are decided at run–time of the program and not at compile time. So, we can say that dynamic memory allocation enables us to create data types and structures of any length and size which suits our program requirements.
- Important **memory allocation functions** are: malloc(), calloc(), realloc() and free().
- **Advantages of dynamic memory allocation over static memory allocation are:** Memory allocation at run–time, No wastage of memory, unused memory can be made free, faster speed and efficient rearrangement of data.

Function **malloc()** returns a pointer to a memory block of required size which is allocated. The pointer points to void, so we need to type cast it to appropriate type.

Function **calloc()** allocates multiple blocks each of same size and initializes them with zero. The pointer points to void, so we need to type cast it to appropriate type.

Function **realloc()** is used to change the size of memory block allocated previously using malloc() or calloc() function. The pointer points to void, so we need to type cast it to appropriate type.

- In dynamic memory allocation of memory, it is the duty of the programmer to release the memory blocks which are no longer needed in the program. Function **free()** is used to release a memory block.
- **Memory leakage** occurs when memory allocated is not released even though it is not required. It is basically wastage of memory resource. This leads to less memory available for other programs and sometimes results in slowdown or system crash problem.
- **Linked list** is a collection of items which are referred sequentially. The elements are not stored sequentially in memory, but pointers are used in such a way that they appear in ordered form logically.
- Some of the **operations on linked list** are: Create a linked list, Traverse a list, Insert an element, Delete an element, Print the elements of a list, Count the elements of a list etc.

: MCQs :

1. Which function allocates one block of memory?
   (a) Calloc() (b) malloc() (c) Both a and b (d) None of above
2. Which function allocates multiple memory blocks of same size
   (a) Calloc() (b) malloc() (c) Both a and b (d) None of above
3. What is the return type of malloc() function?
   (a) Void pointer (b) Int pointer (c) char pointer (d) Integer
4. When storage requirements are not clear at the time of writing the program we use
   (a) Dynamic memory allocation (b) Static memory allocation
5. Function used to release memory block not required is
   (a) realloc() (b) alloc() (c) malloc (d) free()

6. Which function reallocates memory?
   (a) realloc (b) calloc (c) malloc (d) None of these
7. Difference between calloc() and malloc()
   (a) calloc() takes a single argument while malloc() needs two arguments
   (b) malloc() takes a single argument while calloc() needs two arguments
   (c) malloc() initializes the allocated memory to ZERO
   (d) calloc() initializes the allocated memory to NULL
8. The function calloc initialises memory with all bits set to zero.
   (a) True (b) False
   (c) Depends on Compiler (d) Depends on standard
9. What function should be used to release allocated memory which is not needed?
   (a) dealloc( ) (b) free( ) (c) release( ) (d) unalloc( )

: ANSWERS :

1. (b) **2.** (a) **3.** (a) **4.** (a) **5.** (d) **6.** (a) **7.** (b)
8. (a) **9.** (b)

: Exercise :

1. What is Dynamic memory allocation? How does it help in solving complex problems of programming?
2. List main functions of Dynamic memory allocation.
3. What is the main difference between malloc() and calloc() functions?
4. Explain the function used to release memory. What care should be taken while using that function?
5. What the following code will do?

```
int *ptr;
ptr = malloc(sizeof(int));
```

6. What the following code will do?

```
int *ptr;
ptr = malloc(5 * sizeof(int));
```

7. Write short note on realloc() function.
8. What is memory leakage?

: Answers to Selected Exercises :

5. A memory block for one integer (i.e 2 bytes) storage will be allocated and its address is stored in variable ptr.
6. A memory block for storing five integers (i.e 10 bytes) storage will be allocated and its address is stored in variable ptr.
8. **Memory leakage** occurs when programmer does not release the memory allocated even though it is not required. It is basically wastage of memory resource. This leads to less memory available for other programs and sometimes results in slowdown or system crash problem. It is a problem. So, it is a normal practice to release the memory block which is no longer required by using free() function to avoid the problem of memory leakage.

: SHORT QUESTIONS :

1. What is dynamic memory allocation?
⇒ Dynamic memory allocation is a technique which allows us to allocate memory and release memory at runtime.
2. List important memory allocation functions in 'C'.
⇒ Memory allocation functions are : malloc(), calloc(), free() realloc() .
3. Which function reallocates memory?
⇒ The function to reallocate memory is : realloc().
4. Can a structure contain a pointer to itself?
⇒ Yes, such structures are called self-referential structures. It is used in linked lists.

❖

CHAPTER

7

Looping Control Structure

7.1 INTRODUCTION

7.2 TYPES OF LOOPS

7.3 while LOOP

7.4 do...while LOOP

7.5 for LOOP

7.6 NESTING OF LOOPS

7.7 break STATEMENT

7.8 continue STATEMENT

❖ SUMMARY

❖ MCQs

❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES

❖ SHORT QUESTIONS

7.1 INTRODUCTION :

In programming, certain statement(s) are required to be repeated for fixed number of times or till certain condition is satisfied. If the language does not provide any looping structures then we will have to write those statements again and again in our program. The result will be very lengthy program, more typing and unmanageable code. In that case we will have no option but to use goto statement. But 'C' language provides the looping structures. The looping structures provided in 'C' language are:

- `while loop,`
- `do...while loop`
- `for loop.`

Which type of looping statement to use depends on the situation. For example, if the number of times the statements are to be executed is known in advance, then we can use for loop. Otherwise we have to use while loop or do...while loop.

The loop is controlled by a condition. The condition decides how many times the statements will be executed. If there are more than one sentence (which will normally be the case) in the loop, then those sentences are put in { } symbols. The statements within { } symbols are called body of the loop. We can say that there are two parts of loop : **condition** and **body**.

7.2 TYPES OF LOOPS :

Depending on where the condition is checked, we can have two types of loop structures:

1. **Entry Control**
2. **Exit Control**

In entry control loop, the condition is written first and then the body of statements. While in exit control loop, the body of statements is written first and then condition is written. This means that body of statements in exit control loop will be executed at least once.

Figure 7.1 explains Entry control loop while, figure 7.2 explains Exit control loop.

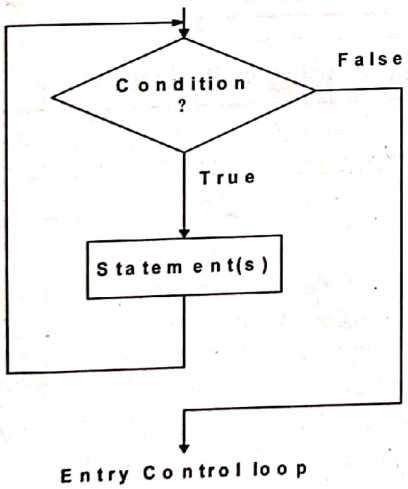


Figure 7.1

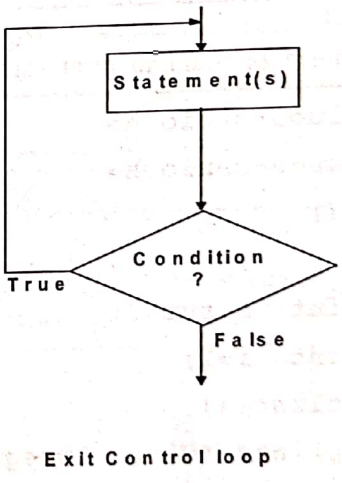


Figure 7.2

7.3 while LOOP :

While loop is helpful, when we do not know in advance, how many times the statements are to be repeated. The syntax of while loop is:

```
while (condition)
      {
          statement(s);
      }
```

The statements within the { } symbols are known as body part of the loop. The { } are not required if there is only one statement in the body part. Here, the condition is checked first and then only the statements are executed. So, while loop is **entry controlled** loop, its **flowchart is same as entry control** loop shown above.

The statements in the body part will be executed till the condition is TRUE. As soon as the condition becomes false, the control will come out of the while loop. If the condition remains TRUE always then, it will lead to infinite loop. So, writing a valid condition is important.

Look at the following code segment which prints first n numbers :

```
.
.
.
i  =1;

while  (  i  <=  n)
{
    printf("%d\n",i);
    i++;
}
.
.
.
```

Condition

body

In above code segment, variable i is initialized to 1. Then while statement with condition i <= n is written in brackets. In the body part, there are two statements. The value of i variable is printed and then incremented. The statements within the loop body executed till the condition remains TRUE. When the condition becomes FALSE, the while loop terminates and the control is transferred to next statement after while loop.

Program :

```
/* Write a program to print sum of first n integer numbers  using while loop */
#include<stdio.h>
#include<conio.h>
main()
{
    int  n,sum=0;
    int  i=1;
    clrscr();
    printf("Give  Integer  number:    ");
    scanf("%d",&n);
    while(i<=n)
        {
        sum  =  sum  +i;
        i++;
        }
    printf("Sum  of  first  %d  numbers  =  %d\n",n,sum);
}
```

Output :

```
Give Integer number:5
Sum of first 5 numbers = 15
```

Program :

/* Program to find LCF of 2 integers */

```
#include<stdio.h>
#include<conio.h>

void main()
{
    int a,b,m, i=1;
    clrscr();
    printf("Enter 2 Integer Numbers ");
    scanf("%d%d",&a,&b);
    m=a;
    while(m%b!=0)
    {
        i++;
        m=a*i;
    }
    printf("LCF of %d & %d = %d\n",a,b,m);
    getch();
}
```

Output :

```
Enter 2 Integer Numbers 4 6
LCF of 4 & 6 = 12
```

Program :

/* Write a program to print the individual digits of a given integer number using while statement. **This program uses % operator to find out remainder after division.** */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int n,i;
    clrscr();
    printf("Give Integer number: ");
    scanf("%d",&n);
    while(n!=0)
        {
        i = n %10; /* separate the digit */
```

```
        printf(" The digit is = %d \n",i);
        n = n/10;
          /* new value of n after separation of digit */
        }
}
```

Output :

```
Give Integer number: 23567
The digit is = 7
The digit is = 6
The digit is = 5
The digit is = 3
The digit is = 2
```

Program :

```
/* Write a program to reverse a given integer number */
#include<stdio.h>
#include<conio.h>
main()
{
    int num;
    int i;
    clrscr();
    printf("Give integer number\n");
    scanf("%d",&num);
    printf("reverse of %d is =",num);
    while(num !=0)
        {
        i = num%10;
        printf("%d",i);
        num = num/10;
        }
}
```

Output :

```
Give integer number
25436
reverse of 25436 is =63452
```

Program :

```
/* Write a program to check the given number is armstrong number or not.
Armstrong number is equal to summation of cubes of its individual digits.
Here, we will use the logic of previous program to separate the individual digits and then sum the cubes
of individual digits. */
#include<stdio.h>
#include<conio.h>
main()
{
    int n,sum =0;   /* sum initialized to 0 */
    int i,num; /* separated digit stored in i */
    clrscr();
    printf("Give Integer number:    ");
    scanf("%d",&n);
    num =n;/* n stored in num, because n
          will change in loop */
        while(n!=0)
        {
        i = n %10;              /* separate the digit */
        sum = sum + i*i*i;  /* find cube of digit
                            and add to sum */
        n = n/10;   /* new value of n after
            separtation of digit */
        }
    if(sum == num)
        printf("Given number %d is armstrong\n",num);
    else
        printf("Given number %d not armstrong\n",num);
}
```

Output-1 :

```
Give Integer number:    135
Given number 135 not armstrong
```

Output-2 :

```
Give Integer number: 153
Given number 153 is armstrong
```

Output-3 :

```
Give Integer number:    39
Given number 39 not armstrong
```

Program :

/* Write a program to print first N prime numbers. Prime number is a number which is divisible by 1 and itself only.

Logic for checking prime, we need only check the divisibility of that number by upto square root of that number. First prime number is 2, remaining prime numbers can be odd only.

For example, 2, 3, 5, 7, 11 etc are prime numbers */

```
#include<stdio.h>
#include<conio.h>
#include  <math.h>
main()
{
    int  i,n,count,temp;
    int  num;
    /*  count  stores  the  numbers  displayed
        num  stores  the  current  number  checked  for prime
        temp  is  a  temporary  variable  stores  sqrt(num).
        i  starts  from  2  goes  maximum  upto  temp  */
   clrscr();
   printf("How  many  prime  numbers?\n");
   scanf("%d",&n);
   printf("The  prime  numbers  are\n");
   num  =2;
   printf("%4d",num);
   count  =1;
   num++;
   while(count  <  n)
       {
       temp  =  sqrt(num);
       i  =  2;
       while  (i  <=  temp)
           {
           if  (num  %  temp  ==  0)
                   break;
           i++;
           }
       if  (i  >temp)
           {
           printf("%4d",num);
           count++;
           }
       num  =  num  +2;
       }
}
```

Output :

```
How many prime numbers?
8
The prime numbers are
    2   3   5   7   11   13   17   19
```

Output :

```
How many prime numbers?
2
The prime numbers are
    2   3
```

Program :

/* Write a program to print first N Fibonacci numbers.
Fibonacci number is a number which is a summation of previous 2 numbers in series.0 and 1 are first two Fibonacci numbers.
For example, 0, 1, 1, 2, 3, 5, 8, 13 etc is Fibonacci series */

```
#include<stdio.h>
#include<conio.h>
#include  <math.h>
main()
{
    int  num1,num2,num3,n;
    int  count;
    clrscr();
    num1  =0;
    num2  =1;
    printf("How  many  fibonacci  numbers?\n");
    scanf("%d",&n);
    if  (n  >2)
        {
        printf("The  fibonacci  numbers  are\n");
        printf("%5d%5d",num1,num2);
        count  =2;
        while(count  <n)
            {
            num3  =  num1  +num2;
            printf("%5d",num3);
            num1=num2;
            num2=num3;
            count++;
            }
```

```
        }
    else
        printf("By definition first two numbers are 0 1\n");
}
```

Output :

```
How many fibonacci numbers?
10
The fibonacci numbers are
   0   1   1   2   3   5   8   13   21   34
```

Program :

/* Write a program to compute the sum of following series
1- 1/2 + 1/3 - ....+ 1/n */

```
#include<stdio.h>
#include<conio.h>
#include<math.h>      /* use of pow() function */
main()
{
    int n;
    float sum =1.0; /* sum initialized to 1 */
    int i=2;
    clrscr();
    printf("Give Integer number:    ");
    scanf("%d",&n);
    while(i <= n)
        {   /* pow(x,y) function calculates x^y.
            In, pow(-1,i) will be positive for
            even value of i and will be negative
            for odd value of i */
        sum = sum - pow(-1,i)/i;
        i++;
        }
    printf("Summation of given series = %7.2f\n", sum);
}
```

Output-1 :

```
Give Integer number : 3
Summation of given series = 0.83
```

Output-2 :

```
Give Integer number : 5
Summation of given series = 0.78
```

Program :

/* Write a program which accepts a string and counts the number of blank, tab, newline and other characters until EOF */

```
#include<stdio.h>
#include<conio.h>
main()
{
    char  c;
    int  blank=0,     newline=0,tab=0,other=0;
                      /*counters  initialized  */
    clrscr();
    printf("Give  string  that  can  be  multiline  terminated  by  ^z\n");
    c  =getchar();          /*  comment  */
    while(c  !=  EOF)
        {
        switch(c)
            {
                case  ' ':
                      blank++;
                      break;
                case  '\t':
                      tab++;
                      break;
                case  '\n':
                      newline++;
                      break;
                default  :
                      other++;
            }
        c=getchar();              /*  comment  */
        }
    printf("  Blanks=  %d,  Tabs=  %d\n",blank,tab);
  printf("  New  Lines  =  %d,  Others  =  %d\n",newline,other);
}
```

Output :

```
Give  string  that  can  be  multiline  terminated  by ^z
Computer  Programming  is
an  art and science
^z
Blanks=  2,  Tabs=  3
New  Lines  =  2,  Others  =  36
```

In above program, we can also write the while statement as

while((c=getchar()) != EOF). Here, the c=getchar() executes first, so c variable stores the character and then comaparison with EOF takes place. **If you write**

while((c=getchar()) != EOF) then, c=getchar(); statement is not required at both places inside and outside the loop. Just remove the c=getchar(); statements mentioned as /* comment */ in the program and change while statement as shown above.

7.4 do...while LOOP :

do...while is an exit control loop, where the condition is checked later. **The flowchart is same as that for exit control loop.** At least once the body of the loop will be executed. Whenever you want to make sure that loop statements must execute at least one time, you should use do...while loop instead of while loop.

The syntax is :

```
do
{
    statement(s);
} while (condition);
```

Here, the statement(s) are executed till the condition is TRUE. As soon as the condition evaluates to FALSE the loop terminates and the next statement after do...while loop is executed.

We can always use do...while loop wherever we have used while loop. It means that all the program written in previous section using while loop can always be written using do...while loop.

For example,

Program :

/* Write a program to compute the sum of following series 1– 1/2 + 1/3 - ....+ 1/n */

```
#include<stdio.h>
#include<conio.h>
#include<math.h>    /* use of pow() function */
main()
{
    int n;
    float sum =1.0; /* sum initialized to 1 */
    int i=2;
    clrscr();
    printf("Give Integer number:    ");
    scanf("%d",&n);
    do
        {
        sum = sum - pow(-1,i)/i;
            /* pow(x,y) function calculates x^y */
        i++;
        }
    while(i <=n);
    printf("Summation of given series = %7.2f\n", sum);
}
```

Output :

```
Give Integer number:    3
Summation of given series = 0.83
```

Above program was already written using while loop. You can see how while loop can be converted into do...while loop. Similarly you can convert the other programs shown in previous section. **Please remember that reverse is not always true.**

Program :

```
/* Write a program to sum the individual digits of a given positive number
For example, sum of digits of 1234 is 1+2+3+4 = 10  */
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
    int  n,sum=0;
    int  digit;
    clrscr();
    printf("Give  Positive  Integer  number:    ");
    scanf("%d",&n);
    printf("Given  number  =  %d\n",n);
    if  (n  <0  )
        printf("Please  give  positive  number\n");
    else
        {
            do
            {
            digit  =  n  %10;
            sum  =  sum  +  digit;
            n  =  n/10;
            }  while  (n  >0);
            printf("Summation  of  individual  digits  =  %d\n",  sum);
        }
}
```

Output-1 :

```
Give  Positive  Integer  number:  1234
Given  number  =  1234
Summation  of  individual  digits  =  10
```

Output-2 :

```
Give  Positive  Integer  number  :
-23
Given  number  =  -23
Please  give  positive  number
```

Program :

```
/* Write a program to reverse a given integer number and also check for palindrome.
Palindrome is a number which is same after reversal also.
For example, 121 ,222 is palindrome, while 123 is not palindrome. */
#include<stdio.h>
#include<conio.h>
main()
{
    int  num,temp;
    int  i,rev=0;
    clrscr();
    printf("Give  integer  number\n");
    scanf("%d",&num);
    printf("reverse  of  %d  is  =",num);
    temp  =  num;
    while(num  !=0)
        {
        i  =  num%10;
        rev=  rev*10  +  i  ;
        num  =  num/10;
        }
    printf("%d\n",rev);
    if  (temp  ==  rev)
        printf("Given  number  %d  is  palindrome\n",temp);
    else
        printf("Given  number  %d  is  not  palindrome\n",temp);
}
```

Output-1 :

```
Give  integer  number
121
reverse  of  121  is  =121
Given  number  121  is  palindrome
```

Output-2 :

```
Give  integer  number
123
reverse  of  123  is  =321
Given  number  123  is  not  palindrome
```

Explanation :

This program we had already written using while loop. But, in that program, we were not storing the reverse number in any variable. In the above program, we have to compare the original number with the reversed number, so it requires to be stored in a variable.

Program :

```
/* Write a program to print all integers between starting and ending range divisible by 11. */
#include<stdio.h>
#include<conio.h>
main()
{
    int  start,end,temp;
            /*  temp  variable  used  if  exchange  of
            start  and  end  is  needed.If  start  value
            is  higher  than  end  value  */
    int  divisor=11;
    clrscr();
    printf("Give  starting  range\n");
    scanf("%d",&start);
    printf("Give  ending  range\n");
    scanf("%d",&end);
    if(end  <start)        /*  exchange  end  and  start  if
                        wrong  range  input  */
        {
        temp  =  start;
        start  =end;
        end  =  temp;
        }
    printf("Numbers  divisible  by  %d  between  %d  and  %d
are\n",divisor,start,end);
    do
    {
        if  (start  %  divisor  ==0)
            printf("%4d",start);
        start++;
    }  while(start  <=end);
}
```

Output :

```
Give  starting  range
100
Give  ending  range
200
Numbers  divisible  by  11  between  100  and  200  are
110  121  132  143  154  165  176  187  198
```

Output :

```
Give starting range
200
Give ending range
100
Numbers divisible by 11 between 200 and 100 are
  110  121  132  143  154  165  176  187  198
```

7.5 for LOOP :

We have seen in previous section that when we do not know number of iterations required, we can use while or do...while loop. But, in certain cases, we need to execute some steps certain number of times. In such situations where we know the number of iterations in advance, then we can use for loop.

The syntax of for loop is :

```
for(initialization; condition; increment/decrement)
    {
        statement(s);
    }
```

We require one variable which is used to control the loop, which is called as control variable. The control variable is initialized in initialization expression. This statement executes only once. The condition expression actually checks the value of control variable. If the condition expression is TRUE, then the statements written are executed. These statements (also called as body of for loop) if they are more than one, than put in { }, After every execution of statements, the last expression increment/decrement is executed. Then again the condition is checked and body is executed if the condition evaluates to TRUE. Loop terminates when the condition evaluates to FALSE.

Figure 7.3 explains flowchart of for loop.

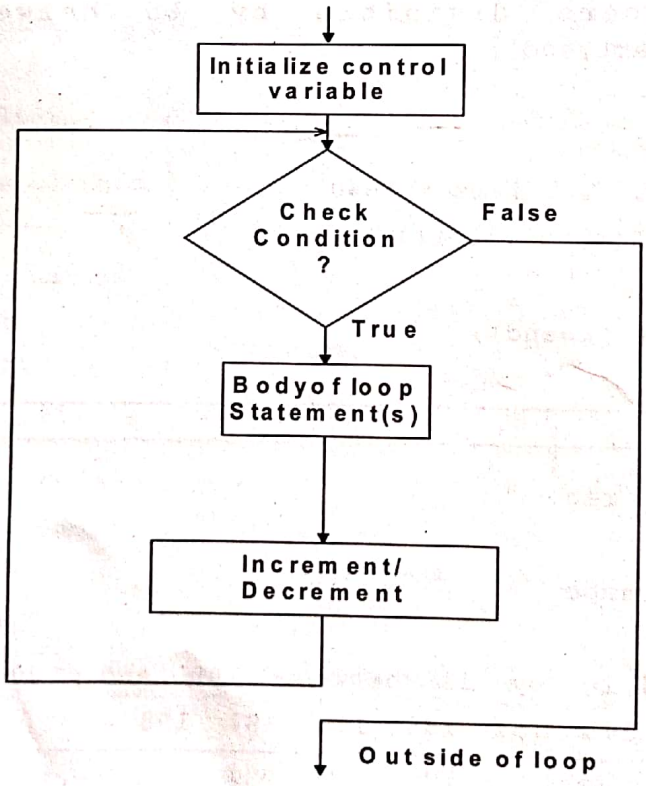


Figure 7.3

For example, following for loop will print the numbers 1 to 10 each on separate line.

```
.
.
for(i=1;i<=10,i++)
        printf("%d\n",i);
.
.
```

Here, i=1; statement is initialization which will execute only once. Then the second expression i<=10 will be evaluated. It is true, so loop body will execute i.e. printf() statement will be executed, then i++ statement will execute, which will make i=2, then again the condition i<=10 will be evaluated, which is true, so again printf() statement will be executed. This sequence will go on till the condition is TRUE, when i becomes 11, at that time condition evaluates to FALSE, so loop is over.

If we execute following code fragment:

```
main()
{
        int  n=6,  t=1;
        for  (;  n<10;  n  =  n+2)
            printf("%d  %d\n"  ,n,  ++t);
}
```

Here, int n=6, t=1; statement initializes n=6 and t=1. Then, in for loop there is no initialization part, so condition n< 10 is true and printf executes, which prints (n=6 and t becomes 2) 6 2

Then n = n+2 statement executes and n becomes 8, again the condition is true, so printf executes, which prints 8 3

Then n = n+2 statement executes and n becomes 10, and the condition n< 10 is false, so loop is over.

So, final output of above code will be

6 2

8 3

Program :

```
/* Write a program to print a number and its square, cube for numbers 1 to 10 */
#include<stdio.h>
#include<conio.h>
main()
{
    int  i;
    int  sq,  cu;
    clrscr();
    printf("Number\tSquare\tCube\n");
    printf("======\t=====\t====\n");
    for(i=1;i<=10;i++)
        {
        sq  =  i  *i;
        cu  =  sq  *i;
        printf("%d\t%d\t%d\n",i,sq,cu);
        }
}
```

Output :

```
Number  Square  Cube
======  =====   ====
1   1   1
2   4   8
3   9   27
4   16  64
5   25  125
6   36  216
7   49  343
8   64  512
9   81  729
10  100 1000
```

Program :

/* Write a program to find factorial of a given number */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  i,n;
    long  int  fact;
    clrscr();
    printf("Give  positive  number  for  which  factorial  is    to  be
found\n");
    scanf("%d",&n);
    if  (  n<0)
       printf("Factorial  of  -ve  number  not  defined\n");
    else
        {
        fact  =1;
        for(i=1;i<=n;i++)
            fact  =  fact  *i;    /*  only  one  statement,
                                  so  {  }  not  required  */
        printf("Factorial  of  %d=  %ld\n",n,fact);
        }
}
```

Output-1 :

```
Give  positive  number  for  which  factorial  is  to  be  found
5
Factorial  of  5=  120
```

Output-2 :

```
Give positive number for which factorial is to be found
8
Factorial of 8= 40320
```

Output-3 :

```
Give positive number for which factorial is to be found
-3
Factorial of -ve number not defined
```

Program :

/* Program to illustrate how user authentication is made before allowing the user to access resources. It asks for the username and password. The password that you enter will not be displayed, instead each character of password is replaced by '*' symbol */

```
#include <stdio.h>
#include <conio.h>

void main()
{
    char password[10],username[10], ch;
    int i;

    clrscr();

    printf("Enter User name: ");
    gets(username);
    printf("Enter the password any 8 characters: ");

    for(i=0;i<8;i++)
    {
        ch = getch();
        password[i] = ch;
        ch = '*' ;
        printf("%c",ch);
    }

    password[i] = '\0';
    printf("\nYour password is :");

    for(i=0;i<8;i++)
    {
        printf("%c",password[i]);
    }
}
```

Output :

```
Enter User name: Samir
Enter the password any 8 characters: ********
Your password is :Samir123
```

Program :

/* Write a program to calculate sum of following series

$$\frac{1}{x} - \frac{2}{x^2} + \frac{3}{x^3} - \frac{4}{x^4} \quad \ldots\ldots$$

Here, alternate terms have different sign. The next denominator is found by multiplying previous value with x, while numerator is the term number. For example, numerator in 3rd term is 3, for 5th term it is 5.*/

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  i,n;
    float  sum,  term,  x;
    clrscr();
    sum=0;       /*  initialize  */
    term  =1;
    printf("Give  value  of  x\n");
    scanf("%f",&x);
    printf("Give  value  of  n\n");
    scanf("%d",&n);
    for  (i=1;i<=n;i++)
    {
        term  =  term  *1/x;          /*  get  denominator  */
        sum  =  sum  +  i  *term;
                        /*  i  value  denotes  numerator  */
        term  =  -term;                  /*  alternate  +/-  */
        }
    printf("Sum  =  %f\n",sum);
}
```

Output :

```
Give  value  of  x
3
Give  value  of  n
4
Sum  =  0.172840
```

If we execute following code,

```
int  i;
```

```
for(i=5;i<15;i++)  {
    printf("%d\n",i);
    i  =  i  -  1;
}
```

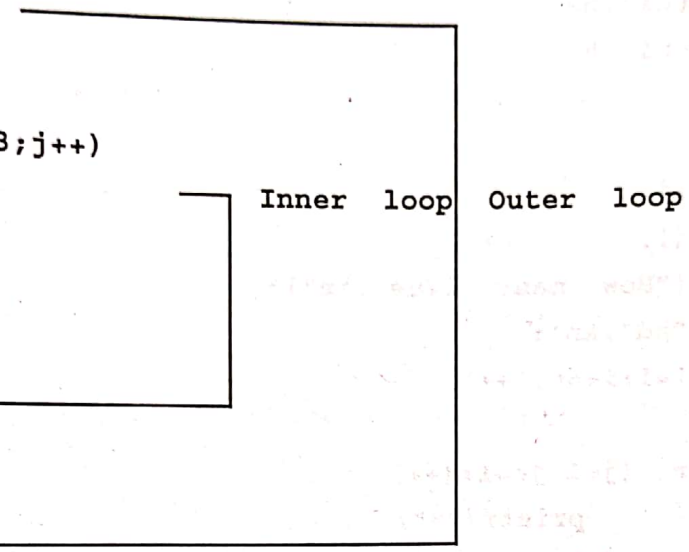
Prints value 5 for infinite time. Because i value will remain 5 only. Actually it is not advisable to alter the value of loop control variable of for loop inside the body of a for loop.

7.6 NESTING OF LOOPS :

When a loop inside another loop is used, it is called nesting of loops. The nesting can be for any number of levels. For certain problems, nesting of loops are needed. When nesting is done, some care is required. The outer loop should not end between the starting of inner loop and ending of inner loop.

Following example shows nesting up to 2 levels.

```
for(i=1;i<=3;i++)
        {
        .
        .
        for(j=1;j<=3;j++)
            {
            .
            .
            .
            }
        .
        .
        }
    .
    .
    .
```



In above example, for(i=1;i<=3;i++) is outer for loop. We can say that control variable for outer loop is 'i'. for(j=1;j<=3;j++) is inner for loop. Variable 'j' is control variable for inner for loop. For each value of outer loop control variable i.e 'i', all the values of inner loop control variable are tried i.e 'j'.

Following table explains the set of values for variable 'i' and 'j'for above example.

i	j
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

From above table, it is very clear that for i=1, j goes from 1 to 3. When all values of j are done, i gets next value i.e i=2, again j goes from 1 to 3. Similarly for i=3, j goes from 1 to 3.

Program :

```
/* Write a program to print following pattern of stars
*
**
***
****
*****
.
.
Here, from the pattern, it is clear that number of stars in line i is = i.
So, this example can be solved using nesting of loops. The outside loop will count number of lines ar
provide newline character while, inner loop will count number of stars in current line. */
#include<stdio.h>
#include<conio.h>
main()
{
    int  i,j,n;
    clrscr();
    printf("How  many  lines?\n");
    scanf("%d",&n);
    for  (i=1;i<=n;i++)
        {
        for  (j=1;j<=i;j++)
                printf("*");
        printf("\n");
        }
}
```

Output :

```
How  many  lines?
6
*
**
***
****
*****
******
```

If you want to generate a pattern like

1
12
123
1234
12345

.....

.....

Then, in above program, you require **to change the printf("*") statement with printf("%d",j)** in inner loop as shown below.

Program :

```
/* Write a program to print following pattern of numbers
1
12
123
1234
12345
....
....
*/
#include<stdio.h>
#include<conio.h>
main()
{
    int  i,j,n;
    clrscr();
    prin    tf("How  many  lines?\n");
    scanf("%d",&n);
    for  (i=1;i<=n;i++)
        {
        for  (j=1;j<=i;j++)
            printf("%d",j);
        printf("\n");
        }
}
```

Output :

```
How  many  lines?
6
1
12
123
1234
12345
123456
```

```
/*
Write a program to display following pattern using nested loops
*****
****
***
**
*
*/
#include<stdio.h>
#include<conio.h>

void  main()
{
     int  n,i,j;
     n=5;
     clrscr();
     for(i=n;i>=0;i-)
          {
          for(j=1;j<=i;j++)
               printf("*");
          printf("\n");
          }
     getch();
}
```

Output :

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

Program :

/* Write a program to print following pattern using loop

1

1 3

1 3 5

1 3 5 7

.....

Here, from the pattern, it is clear that number of numbers in line i is = i

So, this example can be solved using nesting of loops. The outside loop will count number of lines and provide newline character, while inner loop will count number of numbers in current line. */

```
#include <stdio.h>
main()
{
    int i,j,n;
    clrscr();
    printf("How many lines?\n");
    scanf("%d",&n);
    for (i=1;i<=n;i++)
        {
        for (j=1;j<=i;j++)
            printf(„%d „, 2*j-1);
        printf(„\n");
        }
}
```

Output :

```
How many lines?
5
1
1 3
1 3 5
1 3 5 7
1 3 5 7 9
```

Program :

/* Write a program to print following pattern of numbers for n = 6

1
12
123
1234
12345
123456

Here, in first line 5 blanks are followed by number 1. In 3rd line, 3 blanks followed by number 123. So, in general, in line i, n-i blanks are followed by number 1 to i.

We require outer loop to control number of lines. One inner loop to control number of blanks in current line and one other inner loop to print numbers 1 to i in current line. */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,j,k,n;
    clrscr();
    printf("How many lines?\n");
```

```
    scanf("%d",&n);
    for  (i=1;i<=n;i++)  /*  controls  number  of  lines  */
        {
        for  (j=1;j<=n-i;j++)
                    /*  controls  number  of  blanks  */
            printf("  ",j);
        for  (k=1;k<=i;k++)
                    /*  controls  number  printing  */
            printf("%d",k);
        printf("\n");
        }
}
```

Output :

```
How  many  lines?
6
          1
         12
        123
       1234
      12345
     123456
```

Similarly, you can try to write the program for printing the pattern

```
          1
         121
        12321
       1234321
      123454321
     12345654321
```

Where, you require to use the same logic. You have to write an additional inner for loop for printing the numbers in line i from i to 1.

Program :

```
/* Write a program to generate pythagorian triplet between 1 and 25
for example, 3 4 5 is pythagorian triplet, where 3*3 + 4*4 = 5*5 */
#include<stdio.h>
#include<conio.h>
main()
{
    int  i,j,k;
    clrscr();
    for  (i=1;i<=23;i++)
        {
```

```
        for(j=i+1;j<=24;j++)
            {
            for(k=j+1;k<=25;k++)
                {
                if  (i*i  +  j*j  ==  k*k)
                    printf("%2d  %2d  %2d\n",i,j,k);
                }
            }
        }
}
```

Output :

```
 3   4   5
 5  12  13
 6   8  10
 7  24  25
 8  15  17
 9  12  15
12  16  20
15  20  25
```

Program :

/* Write a program to generate following pyramid

```
1          1
12        21
123      321
1234321
123      321
12        21
1          1
```

Here, number of characters in line are **dig = 2*n-1**, In line i, there are **dig – 2*i** blanks, except line n. Upper part is up to line n. In upper part, the numbers in line i are from 1 to i, then blanks and then numbers from i to 1.

Then the lower part starts. In lower part again, line number starts from n-1 to 1. In each line i, number from1 to i are printed, then blanks and then the number from i to 1. */

```
#include  <stdio.h>
#include<conio.h>
main()
{
    int  i,j,k,n,dig;
    clrscr();
    printf("Give  value  of  n\n");
    scanf("%d",&n);
```

```
    dig= 2 *n -1;    /* number of digits including
                        space in line n */

    printf("\n\n");
    for (i=1;i<=n;i++)        /* print upper part */
        {
        for(j=1;j<=i;j++)    /* print numbers
                                before blanks*/

            printf("%d",j);
        for(k=1;k<=dig-2*i;k++)  /* print blanks */
            {
            printf("  ");
            }
        for(j=i;j>=1;j--)          /* print numbers
                                      after blanks */

            if (j != n)
            printf("%d",j);
        printf("\n");
        }                      /* upper part over */
    for (i=n-1;i>=1;i--)          /* print lower part */
        {
        for(j=1;j<=i;j++)
                    /* print numbers before blanks*/
            printf("%d",j);
        for(k=1;k<=dig-2*i;k++)       /* print blanks */
            {
            printf("  ");
            }
        for(j=i;j>=1;j--)
                    /* print numbers after blanks */
            printf("%d",j);
        printf("\n");
        }
}
```

Output :

```
Give value of n
4
1      1
12    21
123  321
1234321
123  321
12    21
1      1
```

Program :

```
/* Program to Display Floyd's triangle */
#include <stdio.h>
main( )
{
    int i, j, k = 1;
    printf("Floyds triangle is\n");
    for( i = 1; k <= 20; ++i )
    {
        for( j = 1; j <= i; ++j )
            printf( "%d ", k++ );
            printf( "\n\n" );
    }
}
```

Output :

```
Floyds triangle is
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
16  17  18  19  20  21
```

7.7 break STATEMENT :

We have already used break statement in switch statement for separating the cases. We can also use break statement in loops also. Whenever, break statement is encountered in a loop, the loop is terminated and control transfers to the statement immediately after the body of loop. It is clear that break statement will be one of the statements in the body of the loop. The break statement will be executed if some condition is satisfied. When break statement executes, all the statements after that up to the end of body are skipped and loop terminates.

Figure 7.4 explains the working of break statement.

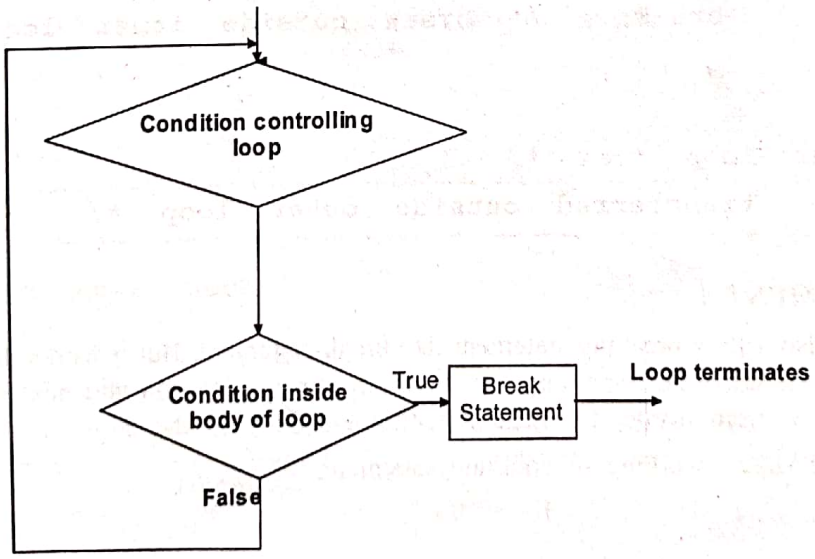


Figure 7.4

In case of nested loops, if break statement is used inside the inner loop, then the inner loop terminates and the next statement following inner loop executes. If the break statement is used inside outer loop, then the outer loop terminates and control is transferred to the statement following the outer loop.

Following example shows the break inside inner loop, so control goes to the outer loop.

```
for(      ;    ;          )
{
    for(      ;        ;          )
        {
        .
        .
        if  (condition)
                              break;
        .
        .
        }      /*  inner  loop  over  */
        .  /*  control  comes  here  when  break  executed  */
        .
} /*  outer  loop  over  */
```

Following example shows the break inside outer loop, so control goes to the statement outside outer loop.

```
for(      ;    ;          )
{
                  for(      ;        ;          )
                  {
                    .
                    .
                    .
                    }       /*  inner  loop  over  */
                    .
                    .
                    if  (condition)
                break;    /*  break  outside  inner  loop  */
                    .
                    .
}    /*  outer  loop  over  */
.    /*control  transferred  outside  outer  loop  */
.
```

7.8 continue STATEMENT :

This statement is also a flow breaking statement like break statement. But it works differently. When continue statement executes, it skips the remaining statements of current iteration and next iteration starts. Normally, continue statement is used inside if statement within the body of the loop.

Following example shows working of continue statement.

```
for(      ;      ;          )
{
```

```
        .
        .
        if (condition)
                        continue;
        .
    /* when continue executes the control
    comes here i.e next iteration starts */
}
```

Program :

```
/* Write a program to display even numbers up to n. Use break and continue */
#include <stdio.h>
#include <conio.h>
void main()
{
    int n,i=0;
        clrscr();
        printf("Give last even number\n");
        scanf("%d",&n);
        printf("Even numbers are :\n");
        while(1)
    {
    if(i%2 ==1)          /* number odd */
        {
                                        /* increment */
        i++;
        continue;          /* skip remaining steps in
                          current iteration */
        }
    if (i > n)
        break;                      /* work over */
    printf("%d\t",i);        /* print even number */
    i++;
    }
}
```

Output :

```
Give last even number
24
Even numbers are :
0    2    4    6    8    10    12    14    16    18    20    22    24
```

7.9 SOLVED PROGRAMMING EXAMPLES :

Program :

```
/* Write a program to count odd and even numbers from given numbers.
   Give last number as -99 to terminate the list */
#include  <stdio.h>
#include  <conio.h>
main()
{
    int  odd,even,n;
    clrscr();
    odd=0;
    even=0;
    printf("Give  last  number  as  -99\n");
    for  (  ;  ;)                          /*  infinite  loop  */
        {
        printf("\nGive  number    :");
        scanf("%d",&n);
        if(  n  ==  -99)
            break;  /*  break  will  terminate  the  loop  */
        if  (  n%2  ==  0)
            even++;
        else
            odd++;
        }
    printf("Number  of  evens  =  %d\n",even);
    printf("Number  of  odds  =  %d\n",odd);
}
```

Output :

```
Give  last  number  as  -99
Give  number  :  1
Give  number  :  2
Give  number  :  3
Give  number  :  4
Give  number  :  5
Give  number  :  -99
Number  of  evens  =  2
Number  of  odds  =  3
```

/* Assume that you want to make the sum of 1 to 100. Give the necessary code to perform the same using (1) for loop (2) while loop (3) do-while loop */

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i,sum=0;
/* for loop */
for(i=1;i<=100;i++)
    sum = sum +i;
printf("%d\n",sum);
/* while loop */
i=1;
sum =0;
while( i<=100)
    {
    sum = sum +i;
    i++;
    }
printf("%d\n",sum);
/* do while loop */
i=1;
sum =0;
do
{
    sum = sum +i;
    i++;
} while (i<=100);
printf("%d\n",sum);
}
```

Output :

```
5050
5050
5050
```

Program :

/* Write a program to check the input number for prime number */

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
```

```
    int  i,j,n;
    clrscr();
    printf("Give  value  of  n\n");
    scanf("%d",&n);
    j  =  sqrt(n);
    for(i=2;i<=j;i++)
        {
        if  (n  %i  ==  0)
            break;
        }
    if  (  i>j)
        printf("Number  %d  is  prime\n",n);
    else
        printf("Number  %d  is  not  prime\n",n);
}
```

Output-1 :

```
Give  value  of  n
13
Number  13  is  prime
```

Output-2 :

```
Give  value  of  n
39
Number  39  is  not  prime
```

Program :

/* Write a program to print all prime numbers in a given range. */

```
#include  <stdio.h>
#include  <conio.h>
#include  <math.h>
void  main()
{
        int  start,  end,i,d,flag;
        clrscr();
        printf("Give  start  and  end  range  numbers\n");
        scanf("%d%d",&start,&end);
        printf("Prime  numbers  are  :\n");
        for  (i=start;  i<=end;i++)
        {
        flag  =1;
        for  (d=2;d<sqrt(i);d++)
            if  (  i%d  ==  0)
```

```
                {
                flag  =0;
                break;
                }
        if(flag  ==1)
            printf("%d  ",i);
        }
}
```

Output :

```
Give  start  and  end  range  numbers
5  50
Prime  numbers  are  :
5    7  11  13  17  19  23  29  31  37    41  43    47
```

Program :

/* Write a program to check the number is perfect or not. */

```
#include  <stdio.h>
#include  <conio.h>
#include  <math.h>
void  main()
{
        int  n,i,sum  =0;
        clrscr();
        printf("Give  number  to  be  checked  for  perfect\n");
        scanf("%d",&n);
        for  (i=1;  i<n;i++)
        {
        if  (  n%i  ==  0)
            sum  =  sum  +i;
        }
        if  (sum  ==  n)
            printf("Given  number  %d  is  perfect\n",n);
        else
            printf("Given  number  %d  is  not  perfect\n",n);
}
```

Output-1 :

```
Give  number  to  be  checked  for  perfect
6
Given  number  6  is  perfect
```

Output-2 :

```
Give number to be checked for perfect
14
Given number 14 is not perfect
```

Program :

/* Write a program to print all Armstrong numbers in a given range.
Armstrong number is equal to summation of cubes of its individual digits. */

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
        int start, end,i,d;
        int temp, sum =0;
        clrscr();
        printf("Give start and end range numbers\n");
        scanf("%d%d",&start,&end);
        printf("Armstrong numbers are :\n");
        for (i=start; i<=end;i++)
        {
        temp=i;
        sum =0;
        do
        {
        d = temp%10;
        sum = sum + d*d*d;
        temp = temp/10;
        }
        while (temp !=0);
        if (i == sum)
    printf("%d ",i);
    }
}
```

Output :

```
Give start and end range numbers
100    400
Armstrong numbers are :
153  370  371
```

Program :

/ *Write a C program to print multiple of N from given range of unsigned integers. For example, if N=5 and range is [17, 45] it prints 20,25,30,35,40,45. */

```
#include<stdio.h>
main()
{
    int  i,n;
    int  start,end;
    int  temp,num;
    printf("Give  value  of  n\n");
    scanf("%d",&n);
    printf("give  start  and  end\n");
    scanf("%d  %d",&start,  &end);
    temp  =  start/n;
    num  =  temp  *  n;
    if  (num<  start)
        num  =  num  +n;
    while  (  num  <=  end)
        {
        printf("%d  ",  num);
        num  =  num  +n;
        }
}
```

Output :

```
Give  value  of  n
5
give  start  and  end
17  45
20  25  30  35  40  45
```

Program :

/* Write a program to solve following series
1 - x^2/2! + x^4/4!-x^6/6! + ...+ x^n/n!
*/

```
#include  <stdio.h>
#include  <conio.h>
#include  <math.h>
void  main()
{
    int  x,n,fact,i,j;
    int  sign  =-1;
```

```
            /* alternate terms have opposite sign */
    float term, sum =1;          /* initialize */
    clrscr();
    printf("Give values of x and n\n");
    scanf("%d%d",&x,&n);
    for(i=2;i<=n;i=i+2)
        {
        fact =1;
        for(j=1;j<=i;j++)
            fact = fact *j;        /* factorial */
        term =1;
        for(j=1;j<=i;j++)
            term = term *x;    /* x term with power */
        term = term /fact *sign;    /* final term */
        sum = sum +term;                /* addition */
        sign = -sign;                  /* change sign */
        }
    printf("Sum = %6.2f\n",sum);
}
```

Output :

```
Give values of x and n
2  3
Sum =    -1.00
```

Program :

/* Write a program to solve follwoing series
x- x^3/3! + x^5/5!-x^7/7! + ...+ x^n/n!
*/

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    int x,n,fact,i,j;
    int sign =1;
            /* alternate terms have opposite sign */
    float term, sum =0; /* initialize */
    clrscr();
    printf("Give values of x and n\n");
    scanf("%d%d",&x,&n);
    for(i=1;i<=n;i=i+2)
        {
```

```
        fact  =1;
        for(j=1;j<=i;j++)
            fact  =  fact  *j;       /*  factorial  */
        term  =1;
        for(j=1;j<=i;j++)
            term  =  term  *x;     /*  x  term  with  power  */
        term  =  term  /fact  *sign;  /*  final  term  */
        sum  =  sum  +term;                /*  addition  */
        sign  =  -sign;                      /*  change  sign  */
        }
    printf("Sum  =  %6.2f\n",sum);
}
```

Output-1 :

```
Give  values  of  x  and  n
1  5
Sum  =  0.84
```

Output-2 :

```
Give  values  of  x  and  n
2  5
Sum  =  0.93
```

Program :

/ * Write a program to find sum of first N odd numbers */

```
#include  <conio.h>
main()
{
int  i,  count,  n,  sum  =0;
printf("How  many  first  odd  numbers  to  sum?\n");
scanf("%d",  &n);
i  =1;  count  =0;
while  (count  <=n  )
    {
    if  (i  %  2  ==  1)
        {
        sum  =  sum  +i;
        i  =  i  +2;
        count++;
        }
    }
printf("Sum  of  first  odd  %d  numbers  =  %  d\n",  n,  sum);
}
```

Output :

```
How many first odd numbers to sum?
5
Sum of first 5 odd numbers = 36
```

Program :

```
/ * Write a program to find sum of first N even numbers */
#include <stdio.h>
#include <conio.h>
main()
{
int i, count, n, sum =0;
clrscr();
printf("How many first even numbers to sum?\n");
scanf("%d", &n);
i =2; count =0;
while (count < n )
    {
    if (i % 2 == 0)
        {
        sum = sum + i;
        i = i + 2;
        count++;
        }
    }
printf("Sum of first even %d numbers = % d\n", n, sum);
}
```

Output :

```
How many first even numbers to sum?
4
Sum of first even 4 numbers = 20
```

Program :

```
/* Write program to calculate average, mean and standard deviation of real numbers.
Formulas are:

average = Σ ai/n
variance = Σ (a[i]-avg)2
standard deviation = sqrt(variance)                    */

#include <stdio.h>
#include <math.h>
#define MAXSIZE 10
```

```
void main()
{
    float a[MAXSIZE];
    int i, n;
    float avg, var, std_dev;
    float sum =0;
    printf("Enter value of n ");
    scanf("%d", &n);
    printf("Enter %d real/integer numbers\n",n);
    for(i =0; i<n; i++)
        {
            scanf("%f", &a[i]);
            sum = sum + a[i];
        }
    avg = sum/(float) n;
    sum =0;
    for(i=0; i<n; i++)
        {
            sum = sum + pow((a[i]-avg),2);
        }
    var = sum /(float)n;
    std_dev = sqrt(var);
    printf("Average = %f\n", avg);
    printf("Variance = %f\n", var);
    printf("Standard deviation = %f\n", std_dev);
    }
```

Output :

```
Enter value of n 5
Enter 5 real/integer numbers
88
34
10
12
32
Average = 35.200001
Variance = 794.559937
Standard deviation = 28.187939
```

: SUMMARY :

- Many real life problems require certain statements to be executed repeatedly. For that 'C' language provides looping control structures. They are: **while** loop, **do..while** loop and **for** loop.
- There are two types of loops from the viewpoint of where the condition is checked – **Entry control** (Condition checked at the beginning), **Exit control** (Condition checked at the end).

- **while** loop is an entry controlled loop. It is used when we do not know in advance how many times th statements are to be executed. The statements of while loop executes till the condition is TRUE. As soo as the condition becomes FALSE, the while loop terminates.
- **Do..while** loop is an exit control loop, where the condition is checked at the end. It is used when we wa to make sure that at least once the statements inside the body of while loop are executed. The statemen of while loop executes till the condition is TRUE. As soon as the condition becomes FALSE, the whi loop terminates.
- **for** loop is used when we know in advance how many times the statements are to be executed. We requi to use a loop control variable. Syntax of for loop has three parts. The second part is a condition. A soon as the condition becomes false the loop terminates.
- Using a loop inside a loop is known as **nesting of loops**. The nesting can be for any number of leve Each loop will have its own control variable. The outer loop should not end before the inner loop is ov
- **break** statement is a flow breaking statement. If used inside a loop, it terminates the loop and contr is transferred to a statement immediately after the body of the loop.
- **continue** is also a flow breaking statement, if used inside a loop, it skips the remaining statements of curre iteration and next iteration starts.

: MCQs :

1. Which are looping structures?
   (a) for loop (b) while loop (c) do..while loop (d) All of above
2. In which type of loop, body of statements will be executed at least once?
   (a) Entry controlled (b) Exit controlled (c) Both a and b (d) None of above
3. What is the output of following code fragment?

```
num = 123;
while(num !=0)
    {
    i = num%10;
    printf("%d",i);
    num = num/10;
    }
```

   (a) 123 (b) 321 (c) 6 (d) 0
4. How many times "CPU" will be printed for following code?

```
main()
{
    int x;
    for(x=-1; x<=10; x++)
    {
        if(x < 5)
            continue;
        else
            break;
        printf("CPU");
    }
}
```

   (a) 0 (b) infinite (c) 10 (d) 11

5. Which one is correct syntax of for loop?
   (a) for(condition; initialization; increment/decrement) (b) for(initialization; condition; increment/decrement)
   (c) for(increment/decrement; initialization; condition) (d) All of above

6. While nesting loops, what is true?
   (a) Any loop can end anywhere (b) Inner loop can end anywhere inside outer loop
   (c) outer loop cannot end inside inner loop (d) All of above

7. While nesting, if break statement is used inside inner loop?
   (a) Control goes to outer loop (b) Control goes to outside of outer loop
   (c) None of above

8. Which loop is entry controlled?
   (a) Do.. while (b) while (c) for (d) All of above

9. Which loop is exit controlled?
   (a) Do.. while (b) while (c) for (d) All of above

10. Which statements are flow breaking statements?
    (a) break (b) continue (c) Both a and b (d) None of above

11. Which of following loop is executed at least once?
    (a) do–while (b) for (c) if (d) while

12. The first expression in a for ... loop is
    (a) Step value of loop (b) Value of the counter variable
    (c) Condition statement (d) None of the above

13. Continue statement used for
    (a) To continue to the next line of code
    (b) To stop the current iteration and begin the next iteration from the beginning
    (c) To handle run time error
    (d) None of above

14. How many times the following code prints the string "Hello".

```
for (i=1; I <= 50; i++) ;      /* Semicolon here */
    printf("Hello");
```

    (a) 1 (b) 0 (c) 50 (d) None of above

15. How many times the following code prints the string "Hello".

```
for (i=1; I <= 50; i++)
    printf("Hello");
```

    (a) 1 (b) 0 (c) 50 (d) None of above

: ANSWERS :

1.	(d)	2.	(b)	3.	(b)	4.	(a)	5.	(b)	6.	(c)	7.	(a)
8.	(b)	9.	(a)	10.	(c)	11.	(a)	12.	(b)	13.	(b)	14.	(a)
15.	(c)												

: EXERCISE :

1. What is loop? Why the looping structures are needed?
2. What are the different types of looping structures?
3. Which type of looping structure will execute at least once?

4. Write the syntax of for loop. How it differs form while loop?
5. Compare break statement with continue statement.
6. Give differences between while and do...while loop.
7. What will be the output of following code?

(i)

```
for(i=0;i<10;i++)
{
if  (  i  %2    ==  0)
        j  =  i  *i;
printf("i=  %d  j  =%d\n",i,j);
}
```

(ii)

```
num  =  3;
while  (num  <  15)
{
        if  (  num  <=10)
        {
        printf("%d\n",num);
        continue;
        }
        if  (num  ==  13)
        {
        printf("%d\n",num);
        break;
        }
        num++;
    }
```

8. Write a program to print following pattern

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

9. Write a program to print multiplication table form 1 to 5. i.e 1 *1 = 1*2 =2 upto 5 *10 =50. Display it in 10 rows and 5 columns.
10. State the difference between entry control loop and exit control loop.

: Answers to selected exercises :

2. What are the different types of looping structures?

Ans:

The looping structures provided in 'C' language are:

- **while** loop,

- **do...while** loop and
- **for** loop.

The syntax of while loop is:

```
while (condition)
    {
statement(s);
    }
```

The statements in the body part will be executed till the condition is TRUE. As soon as the condition becomes false, the control will come out of the while loop.

The syntax of do...while loop is:

```
do
{
    statement(s);
} while (condition);
```

At least once the body of the loop will be executed. Whenever you want to make sure that loop statements must execute at least one time, you should use do...while loop instead of while loop.

The syntax of for loop is:

```
for(initialization; condition; increment/decrement)
    {
        statement(s);
    }
```

The control variable is initialized in initialization expression. This statement executes only once. The condition expression actually checks the value of control variable. If the condition expression is TRUE, then the statements written are executed. After every execution of statements, the last expression increment/decrement is executed.

5. Explain break statement and continue statement with example.

Ans:

Break and continue statements are flow breaking statements normally used inside a loop.

Whenever, break statement is encountered in a loop, the loop is terminated and control transfers to the statement immediately after the body of loop. When break statement executes, all the statements after that up to the end of body are skipped and loop terminates.

Following example shows the break inside inner loop, so control goes to the outer loop.

```
for(;;)
{
        for(;;)
        {
        .
        .
        if (condition)
        break;
        .
        .
        }   /* inner loop over */
        .   /* control comes here when break executed */
        .
}           /* outer loop over */
```

Continue statement is also a flow breaking statement like break statement. But it works differently. When continue statement executes, it skips the remaining statements of current iteration and next iteration starts.

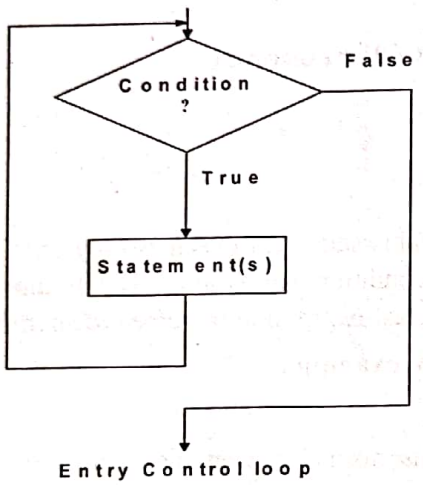
Following example shows working of continue statement.

```
for(;;)
{
        .
        .
        if  (condition)
        continue;
        .
        /* when continue executes the control comes here
        i.e next iteration starts */
}
```

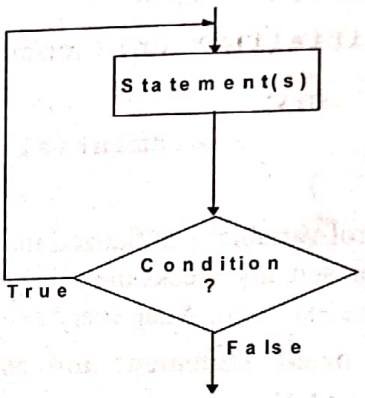
10. State the difference between entry control loop and exit control loop.

Ans:

Flow chart



Entry Control loop



Exit Control loop

Entry Control loop	Exit Control loop
Condition checked first and then the body.	Body executes first and then the condition is checked.
This loop may execute 0 or more times.	This loop executes 1 or more times i.e minimum once.

: SHORT QUESTIONS :

1. Depending on where we check the condition, what are the types of loop structure?

⇒ There are two types of loops: Entry control and exit control.

2. Differentiate between for loop and a while loop?

⇒ For executing a set of statements fixed number of times we use for loop, while when the number of iterations to be performed is not known in advance we use while loop.

3. Which are flow breaking statements?

⇒ Continue are break statements are flow breaking statements.

4. What is nesting of loops?

⇒ Using a loop inside a loop is called as nesting of loops. It can be for any number of levels. Every loop has its own control variable.

❖

CHAPTER

8

Arrays and Strings

8.1 ARRAY

8.2 SINGLE DIMENSIONAL ARRAY

8.3 INITIALIZATION OF SINGLE DIMENSIONAL ARRAY

8.4 TWO DIMENSIONAL ARRAY

8.5 INITIALIZATION OF TWO-DIMENSIONAL ARRAY

8.6 MULTI-DIMENSIONAL ARRAY

8.7 STRING

8.8 READING STRINGS

8.9 PRINTING STRINGS

8.10 STRING HANDLING BUILT-IN FUNCTIONS

❖ SUMMARY

❖ MCQs

❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES

❖ SHORT QUESTIONS

8.1 ARRAY :

Array is a collection of variables of same type known by a single name. The individual elements of an array are referred by their index or sub-script value. The array is an important concept and helps the programmer in managing many variables of the same type, because all the elements of an array share a single name. Arrays can be divided into two categories

1. Single dimensional array
2. Multi-dimensional array

8.2 SINGLE DIMENSIONAL ARRAY :

The syntax for declaring a single dimensional array is :

```
datatype   arrayname[size];
```

Here, datatype defines the type of array elements, it can be int, char, float , long int etc. The arrayname is the name of a variable which represents the array, while size which is represented in [ ] symbols represents the size of array. If size is 10, it means the number of elements an array can store is 10.

For example,

```
int a[10];
```

here, int is datatype, **a** is the name of an array variable and 10 is the size of an array. It defines 10 integer variables, all known by variable name **a**. In 'C' language array index or subscript starts from 0. So, for above example, we have 10 different integer variables as a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8] and a[9], where a[0] is the first element of an array, while a[9] is the last element.

Graphically, array **a** can be represented like this.

Array elements

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
	12	45	34	2	33	65	76	8	6	67
index	0	1	2	3	4	5	6	7	8	9

Figure 8.1

From figure 8.1, we can see that we have 10 different variables, a[0] to a[9]. Index value goes from 0 to 9.

Element a[5] has value 65, element a[8] has value 6, while element a[0] has value 12. In a[5], 5 is the index value, similarly in a[8], 8 is the index value.

The array element can be used in any expression, just like a normal variable is used.

Following code of for loop reads 10 different integer numbers from keyboard and stores in array **a**.

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

The above code stores the first number in a[0], second number in a[1] and 10th number in a[9]. Similarly following code of for loop will print the values of array **a** starting from a[0] to a[9].

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

Let us write a small program which explains how we can store data in an array and use the array elements in a program.

Program :

/* Write a program to get n numbers and find out sum and average of numbers */

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a[10];      /* array of size 10 defined */
    int i,n;
    float avg,sum =0;    /* sum initialized to 0 */
    clrscr();
    printf("Give value of n (not more than 10)\n");
    scanf("%d",&n);  /* actual array size in n */
    for(i=0;i<n;i++)
        {
        printf("Give number\n");
        scanf("%d",&a[i]);
                /* store array elements in array a */
        sum = sum + a[i];
              /* go on adding array element to sum */
        }
    avg = sum/n;     /* sum over. Calculate average */
    printf("Array elements are :\n");
    for (i=0; i<n;i++)      /* print array elements */
          printf("%4d",a[i]);
    printf("\nSum = %f Average = %6.2f\n",sum,avg);
                          /* print answer */
}
```

Output :

```
Give value of n (not more than 10)
4
Give number 12
Give number 15
Give number 20
Give number 18
Array elements are :
  12   15   20   18
Sum = 65.000000 Average =  16.25
```

Explanation :

In this program, we have declared an array of size 10. It means we can store only 10 integer numbers. For finding average, we need to sum all numbers. So, **sum variable is initialized to zero.** In for loop i variable

goes from 0 to n-1, and we read n values one by one and go on adding into sum variable. When the for loop is over, we calculate average by using **avg=sum/n** statement. The avg variable is declared as floating because division of integer by integer can be float number.

The following code prints the array data and then prints the sum and average as result.

Note that in the for loop only one printf statement is there, while second printf statement is outside for loop.

```
for (i=0; i<n;i++)
      printf("%4d",a[i]);
printf("\nSum = %f Average = %6.2f\n",sum,avg);
```

Program :

```
/* Write a program using an array to find largest and smallest number from given n numbers. */
#include <stdio.h>
#include <conio.h>
main()
{
    int a[10];     /* array of size 10 defined */
    int i,n;
    int max,min;
    clrscr();
    printf("Give value of n (not more than 10)\n");
    scanf("%d",&n);  /* actual array size in n */
    for(i=0;i<n;i++)
        {
        printf("Give number\n");
        scanf("%d",&a[i]);
        }
    max = a[0];       /* initialize min and max */
    min = a[0];
    for (i=1; i<n;i++)
        {
        if (max < a[i])
            max = a[i];
        if (min > a[i])
            min = a[i];
        }
     printf("Array elements are :\n");
     for (i=0; i<n;i++)
          printf("%4d",a[i]);
   printf("\nLargest = %d Smallest = %d\n",max,min);
}
```

Output :

```
Give value of n (not more than 10)
5
Give number
2
Give number
4
Give number
7
Give number
8
Give number
-4
Array elements are :
   2    4    7    8   -4
Largest = 8 Smallest = -4
```

Explanation :

Here, max and min variable are used to store maximum and minimum values respectively. The input data is stored in an array using scanf() statement in for loop. Then max and min are initialized as the first value of array by following statements:

```
max = a[0];
min = a[0];
```

Then, in the for loop, which is written as below,

```
for (i=1; i<n;i++)
    {
    if (max < a[i])
        max = a[i];
    if (min > a[i])
        min = a[i];
    }
```

starting from index value 1 to n-1, each number is compared with max and min, if the current maximum or minimum is found, it is stored in corresponding max or min variable. So, max and min variable will always hold the current maximum and current minimum number found so far. When the for loop is over, the required answer is available in max and min variables, which are printed.

Program :

```
/* Write a program to find number of odd and even numbers from given n numbers.*/
#include <stdio.h>
#include <conio.h>
main()
{
```

```
    int a[10];     /* array of size 10 defined */
    int i,n;
    int odd=0,even=0;   /* initialize counts */
    clrscr();
    printf("Give value of n (not more than 10)\n");
    scanf("%d",&n);   /* actual array size in n */
    for(i=0;i<n;i++)
        {
        printf("Give number\n");
        scanf("%d",&a[i]);
        if (a[i] %2 == 0)
            even++;      /* increment even count */
        else
            odd++;      /* increment odd count */
        }
    printf("Array elements are :\n");
    for (i=0; i<n;i++)
          printf("%4d",a[i]);
    printf("\nNumber of ODDS = %d EVENS = %d\n",odd,even);
}
```

Output :

```
Give value of n (not more than 10)
7
Give number
2
Give number
4
Give number
3
Give number
1
Give number
5
Give number
6
Give number
7
Array elements are :
   2   4   3   1   5   6   7
Number of ODDS = 4 EVENS = 3
```

Explanation :

Here, we require to maintain two counters, one for odd numbers and other for even numbers. Following statement declares and initializes the counter to zero.

```
int odd=0,even=0;
```

In for loop, the numbers are read, and checked for odd or even by using % (modulo division) operator, and corresponding counter is incremented. The statements

```
for (i=0; i<n;i++)
    printf("%4d",a[i]);
printf("\nNumber of ODDS = %d EVENS = %d\n",odd,even);
```

prints the original list and the odd, even counts.

Program :

/* Write a program which declares array of 10 integers, enter the data and sum all the elements which are even. Also find the maximum number from the even numbers.*/

```
#include <stdio.h>
main()
{
    int a[10];      /* array of size 10 defined */
    int i,n;
    int sum=0,max=0;   /* initialize counts */

    printf("How many numbers?\n");
    scanf("%d",&n);   /* actual array size in n */

    for(i=0;i<n;i++)
        {
        printf("Give number\n");
        scanf("%d",&a[i]);
        if (a[i] %2 == 0)
            {
            sum = sum + a[i];
            if (max < a[i])
            max = a[i];
            }
        }
    printf("Sum of even numbers = %d and maximum = %d",
    sum,max);
}
```

Output :

```
How many numbers?       8
Give number             2
Give number             4
Give number             5
Give number             6
Give number             8
Give number             7
Give number            12
Give number            56
Sum of even numbers = 88 and maximum = 56
```

Program :

```
/* Delete a desired position element from array */
#include<stdio.h>
#include<conio.h>
void  main()
{
    int  a[20],i,n,pos,x;
    clrscr();
    printf("Enter  how  many  values  in  array\n");
    scanf("%d",&n);
    printf("Enter  %d  values  \n",n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Which  position  value  to  be  delete  :");
    scanf("%d",&pos);
    printf("Your  Existing  List  is  :\n  ");
        for(i=0;i<n;i++)
            printf("%5d",a[i]);
        for(i=pos-1;i<n;i++)
            a[i]=a[i+1];
        printf("\n\After  deletion  the  list  is  :\n  ");
        for(i=0;i<n-1;i++)
            printf("%5d",a[i]);
    getch();
}
```

Output :

```
Enter  how  many  values  in  array
6
Enter  6  values
16
26
36
46
56
66
Which  position  value  to  delete  :  2
Your  Existing  List  is  :
        16      26      36      46      56      66
After  deletion  the  list  is  :
        16   36  46      56      66
```

Program :

/* Write a program which finds maximum of N numbers and also finds how many times maximum is repeated. */

```
#include <stdio.h>
main()
{
    int a[100],n,max,i, count;
    printf("How many numbers?");
    scanf("%d", &n);
    printf("Enter numbers\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    max = a[0];
    for(i=1;i<n;i++)
        {
        if (max < a[i])
            max = a[i];
        }
    count =0;
    for(i=0;i<n;i++)
        {
        if (max == a[i])
            count++;
        }
    printf("Maximum = %d and is repeated %d times\n",max,count);
    }
```

Output :

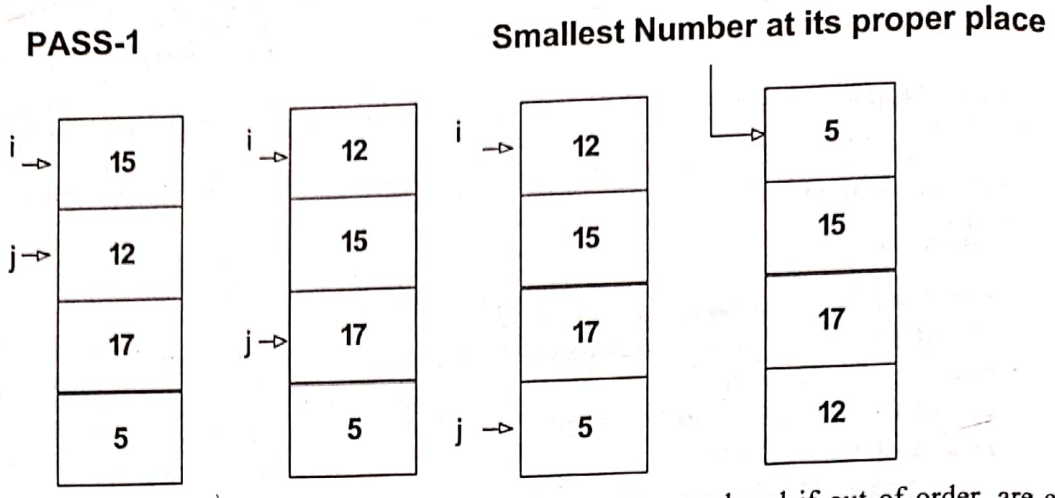
```
How many numbers?8
Enter numbers
12
34
23
34
32
4
34
24
Maximum = 34 and is repeated 3 times
```

SORTING THE LIST :

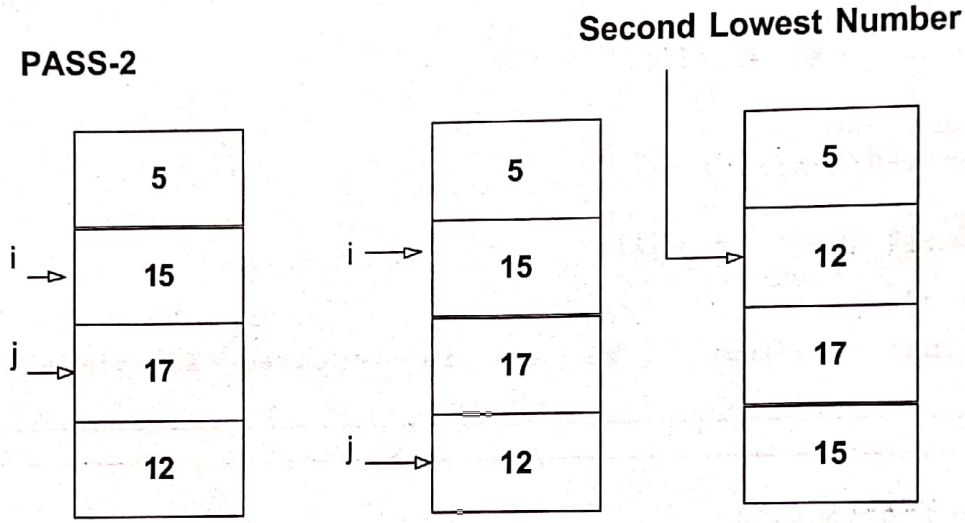
Suppose we want to arrange the numbers in either ascending or descending order. There are many algorithms available to do the sorting. In this program we will use the simplest algorithm.

In general, if there are n numbers in the list, then you require to make n-1 passes (scan the list n-1 times) and in each pass the smallest number is put at its correct position. It means that after every pass, one number is put at its correct position. In each pass this is done by exchanging the two numbers which are out of order (i.e larger number is above and smaller number is below if we are considering ascending order).

This can be explained by following example assuming n=4



As shown above in pass 1, the numbers denoted by i and j are compared and if out of order, are exchange The i value indicates pass number.



As shown in pass 2, the numbers denoted by i and j are compared and if out of order, are exchanged. The value indicates pass number.

Similarly, upto n-1 passes will be carried out to sort the list.

Program :

/* Write a program to sort given n numbers and display them in ascending and descending order. */

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a[10];     /* array of size 10 defined */
    int i,j,n,temp;
    clrscr();
    printf("Give value of n (not more than 10)\n");
    scanf("%d",&n);   /* actual array size in n */
    for(i=0;i<n;i++)    /* input data */
```

```
        {
        printf("Give number\n");
        scanf("%d",&a[i]);
        }
    for(i=0;i<n-1;i++)
        {
        for(j=i+1;j<n;j++)
            {
            if (a[i]> a[j]) /* exchange two numbers */
                {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
                }
            }
        }
    printf("Ascending order data is :\n");
    for (i=0; i<n;i++)  /* print from first to last */
        printf("%4d",a[i]);
    printf("\nDescending order data is :\n");
    for (i=n-1; i>=0;i—) /* print from last to first */
        printf("%4d",a[i]);
}
```

Output :

```
Give value of n (not more than 10)
7
Give number
-4
Give number
4
Give number
-3
Give number
6
Give number
20
Give number
16
Give number
10
```

```
Ascending order data is :
  -4  -3   4   6  10  16  20
Descending order data is :
  20  16  10   6   4  -3  -4
```

Explanation :

In a particular pass i, the numbers which are out of order are exchanged by following piece of code.

```
if (a[i]> a[j])
{
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
```

While, the loop **for(i=0;i<n-1;i++)** indicates pass number, and in each pass the numbers are compared by the loop **for(j=i+1;j<n;j++).**

Once we get sorted data (ascending order data) in an array & print from first number to last, we get ascending order list as done by following code

```
for (i=0; i<n;i++)
    printf("%4d",a[i]);
```

If we print from last number to first, we get descending order list as done by following code

```
for (i=n-1; i>=0;i--)
    printf("%4d",a[i]);
```

8.3 INITIALIZATION OF SINGLE DIMENSIONAL ARRAY :

When the elements of the array are fixed & there is no need to take the values from keyboard, we can initialize the array in the program itself. For example, if you want to store number of days in each month from January to December, it can be initialized in the program itself while declaring array in the program.

In following program the line of code

```
int months[] = { 31,28,31,30,31,30,31,31,30,31,30,31};
```

does that. Here, months[] is a single dimensional array. Note that the brackets are not having the size value. There are total 12 values in the { }, so the size of the array will be automatically calculated.

While { 31,28,31,30,31,30,31,31,30,31,30,31} is the value of elements of an array. It means

```
months[0] = 31
months[1] = 28
.
.
.
months[10] = 30
months[11] = 31.
```

Same way, **int grp[] ={0,0,0};**

is also a single dimensional array whose value of all elements are initialized to zero. The size of this array

3, because there are 3 categories, months of 28 days, 30 days &31 days.

The line

```
enum day {t8,t0,t1};
```

is user defined data type. **day** values can be t8(for two eight), t0(for three zero), t1(for three one). Here, t8 has value 0, t0 has value 1 while t1 has value 2.

In following program, grp[t8] (which is same as grp[0]) is a counter of 28 day month, grp[t0] (which is same as grp[1]) is a counter of 30 day months and grp[t1] (which is same as grp[2]) is a counter of 31 day months.

Program :

/* Write a program to give month numbers as input. Your program should group the month numbers according to number of days.

For example, if the month number input are :

1 2 4 6 9

then no.of months of 31 day = 1

no. of months of 30 day = 3

no. of months of 28 day = 1 */

```
#include <stdio.h>
#include <conio.h>
enum day {t8,t0,t1};
main()
{
    int months[] =
              { 31,28,31,30,31,30,31,31,30,31,30,31};
    int grp[] ={0,0,0};
    int a[12];
    int i,n;
    clrscr();
    printf("Give total number of months n
                    (not more than 12)\n");
    scanf("%d",&n);   /* actual array size in n */
    for(i=0;i<n;i++)    /* input data */
        {
        printf("Give month number\n");
        scanf("%d",&a[i]);
        }
    for(i=0;i<n;i++)
        {
        if (months[a[i]-1] == 28)
            grp[t8]++;
        else if (months[a[i]-1] == 30)
            grp[t0]++;
```

```
        else if(months[a[i]-1] == 31)
            grp[t1]++;
        }
    printf("28 day month = %d 30 day month =
                          %d ",grp[t8],grp[t0]);
    printf("31 day month =%d\n",grp[t1]);
}
```

Output :

```
Give total number of months n (not more than 12)
5
Give month number
1
Give month number
2
Give month number
3
Give month number
4
Give month number
5
28 day month = 1 30 day month =1 31 day month =3
```

8.4 TWO DIMENSIONAL ARRAY :

Sometimes we need to store the data where more than one dimensions are involved, like sales informatio a company, or for mathematical calculations we need to use matrix.

The syntax for two-dimensional array is :

```
datatype variablename[rowsize][colsize];
```

where, **variablename** represents the name of an array, **rowsize** indicate the number of rows in table, co indicates number of columns in an array.

In the case of a sales data of a company, each row lists items and columns represent the month name. intersection of row and column represents the sales data for that particular item and month.

Sales data of company XYZ Ltd

Month / Item	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
A	50	60	70	55	44	55	55	44	33	55	33	101
B	10	12	12	12	03	13	16	19	20	21	25	20
C	01	02	40	40	50	32	34	40	33	38	45	49

To store the above type of data, we need to declare two-dimensional array as:

```
int   sales[3][12];
```

Here, the row-size is 3, so row number spans from 0 to 2, while column number spans from 0 to 11.

Row	0	1	2	3	4	5	6	7	8	9	10	11
0	50	60	70	55	44	55	55	44	33	55	33	101
1	10	12	12	12	03	13	16	19	20	21	25	20
2	01	02	40	40	50	32	34	40	33	38	45	49

In above figure, the cell a[i][j], refers to the cell in ith row jth column. So, a[0][0]=50, a[1][0] = 10, a[2][0] =01, a[1][5] = 13,a[1][11] =20, a[2][0]=01, a[2][2]=40,a[2][11]=49 & like that.

Similarly, in solving mathematical problems involving matrices, we can use two-dimensional array.

Program :

/* Write a program to add two given matrices. Matrices can be added only if number of columns are same as well as number of rows are same in both matrices.

For example, if Matrix A[3][3] and matrix B[3][2] then, addition not possible because column values are different.*/

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a[4][4], b[4][4],c[4][4];
        /* matrix c stores result i.e c = a +b  */
    int m,n,p,q;
        /* m no.of rows in a, n no. of cols in a*/
    int i,j;  /* p no.of rows in b, q no. of cols in b*/
    clrscr();
    printf("Give number of rows in first matrix\n");
    scanf("%d",&m);
    printf("Give number of columns in first matrix\n");
    scanf("%d",&n);
    printf("Give number of rows in second matrix\n");
    scanf("%d",&p);
    printf("Give number of columns in second matrix\n");
    scanf("%d",&q);
    if( m!= p || n!= q) /* check size match or not */
        {
    printf("Size do not match. Addition not possible\n");
        return 0;
        }
    printf("Enter matrix A row-wise\n");
    for(i=0;i<m;i++)    /* Get first matrix data */
        {
```

```
        for(j=0;j<n;j++)
            {
            printf("a[%d][%d]=   ",i,j);
            scanf("%d",&a[i][j]);
            }
        printf("\n");
        }
    printf("Enter matrix B row-wise\n");
    for(i=0;i<p;i++)    /* Get second matrix data */
        {
        for(j=0;j<q;j++)
            {
            printf("b[%d][%d]=   ",i,j);
            scanf("%d",&b[i][j]);
            }
        printf("\n");
        }
    printf("Matrix A \n");
    for(i=0;i<m;i++)
                /* display first matrix row-wise */
        {
        for(j=0;j<n;j++)
            printf("%4d",a[i][j]);
        printf("\n");
        }
    printf("Matrix B \n");
    for(i=0;i<p;i++)
                /* display second matrix row-wise */
        {
        for(j=0;j<q;j++)
            printf("%4d",b[i][j]);
        printf("\n");
        }
    printf("\nSum Matrix C \n");
    for(i=0;i<m;i++)
        /* calculate sum and display result matrix */
        {
        for(j=0;j<n;j++)
            {
            c[i][j] = a[i][j] + b[i][j];
            printf("%4d",c[i][j]);
            }
        printf("\n");
        }
}
```

Output-1 :

```
Give number of rows in first matrix
3
Give number of columns in first matrix
3
Give number of rows in second matrix
3
Give number of columns in second matrix
2
Size do not match. Addition not possible
```

Output-2 :

```
Give number of rows in first matrix
3
Give number of columns in first matrix
3
Give number of rows in second matrix
3
Give number of columns in second matrix
3
Enter matrix A row-wise
a[0][0]= 1
a[0][1]= 2
a[0][2]= 3
a[1][0]= 4
a[1][1]= 5
a[1][2]= 6
a[2][0]= 7
a[2][1]= 8
a[2][2]= 9
Enter matrix B row-wise
b[0][0]= 1
b[0][1]= 2
b[0][2]= 3
b[1][0]= 4
b[1][1]= 5
b[1][2]= 6
b[2][0]= 7
b[2][1]= 8
b[2][2]= 9
```

```
Matrix A
1    2    3
4    5    6
7    8    9
Matrix B
1    2    3
4    5    6
7    8    9
Sum Matrix C
2     4    6
8    10   12
14   16   18
```

Program :

/* Write a program to read 5 * 5 matrix and find maximum number with its row and column index */

```
#include <stdio.h>
main()
{
        int a[5][5];
        int m,n;
        int i,j;
        int row,col,max;

        clrscr();
        printf("How many rows?\n");
        scanf("%d",&m);
        printf("How many columns?\n");
        scanf("%d",&n);

       printf("Enter matrix   row-wise\n");
       for(i=0;i<m;i++)    /* Get matrix data */
           {
           for(j=0;j<n;j++)
               {
               printf("a[%d][%d]=   ",i,j);
               scanf("%d",&a[i][j]);
               }
          printf("\n");
          }
     max = a[0][0];
     for(i=0;i<m;i++)
         {
         for(j=0;j<n;j++)
             {
             if (max < a[i][j])
                 {
                 max = a[i][j];
                 row=i;
```

```
                    col=j;
                    }
                }
            }
    printf("Maximum = %d at row = %d and column =%d \n",max,row+1,col+1);
    }
```

Output :

```
How many rows?    3
How many columns?    4
Enter matrix   row-wise
a[0][0]=   1
a[0][1]=   2
a[0][2]=   3
a[0][3]=   4

a[1][0]=   5
a[1][1]=   4
a[1][2]=   3
a[1][3]=   2

a[2][0]=   3
a[2][1]=   8
a[2][2]=   5
a[2][3]=   4
Maximum = 8 at row = 3 and column =2
```

8.5 INITIALIZATION OF TWO-DIMENSIONAL ARRAY :

The two dimensional array can be initialized at the time of declaration as in the case of single dimensional array. If the array has **m** rows and **n** columns, total elements of an array are **m*n**. If less than m*n values are used, then remaining values are treated as zero. For example,

```
int a[3][2] = { 0,0,1,1,2,2};
```

initializes first row data as {0,0}, second row data as {1,1}, third row data as {2,2}.

The above initialization is same as

```
int a[3][2] = {   {0,0},
                  {1,1},
                  {2,2}
              };
```

Suppose, we initialize as below, where less than 3*2 =6 values are given, then remaining elements will be treated as zero.

```
int a[3][2] = { 0,0,1,1};
```

Here, only 4 elements are given, so last row data i.e third row data will be taken as {0,0}.

8.6 MULTI-DIMENSIONAL ARRAY :

In general, we can have n-dimensional array. If n ≥ 2, then we call that array as multidimensional array. For example, suppose we want to store 5 students marks information for 2 different exams carried out in the term for 4 different subjects, this data can be stored by using 3-dimensional array like,

```
int marks[5][2][4];
```

where, 5 students, 2 exams and 4 subjects. So, total entries in

The array will span from marks[0][0][0] to marks [4][1][3].

8.7 STRING :

String is a sequence of characters enclosed in double quotes. Normally string is useful for storing data like nam address, city etc. ASCII code is internally used to represent string in memory. In 'C' each string is terminat by a special character called null character. In 'C', null character is represented as '\0' or NULL. Because this reason, the character array must be declared one size longer than the string required to be stored.

C O M P U T E R \0

Here, the string stored is "COMPUTER", which is having only 8 characters, but actually 9 characters are stor because of NULL character at the end.

```
char name[20];
```

This line declares an array name of characters of size 20. We can store any string up to maximum length

String is basically an array of characters, so we can initialize the string by using the method of initializing single dimensional array as shown below.

```
char name[] = { 'C', 'O' , 'M', 'P', 'U', 'T','E','R','\0'};
```

or

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

When the string is written in double quotes, the NULL character is not required, it is automatically taken.

8.8 READING STRINGS :

When we understood the scanf() statement, it was mentioned that it can also be used to read string fr keyboard. The format specifier %s is used to read a string in the character array.

For example,

```
char name[20];
.
.
scanf("%s",name);
```

will read the string in the array name. There is no need to put & symbol before the name, because name its is the address of the string name.

We can also use the function **gets()** for reading a string. The array name should be written within bracke For example,

```
char name[20];
.
.
gets(name);
```

The advantage of gets() is that we can read strings involving blanks, tabs. While the scanf() reads only up blank or tab character. So, scanf() is used to read word while gets() can be used read sentence involving mar words.

8.9 PRINTING STRINGS :

The string can be printed using printf() function with %s format specifier, or by using puts() function as shown below.

```
char name[] = "Sanjay";
.
.
printf("%s", name);
.
.
puts(name);
```

puts() can print only one string at a time, while one printf() can print multiple strings by using %s many times in the printf().

For example,

```
char name[] = "Sanjay";
char surname[] ="Shah";
.
.
printf("%s %s", name,surname);
.
.
puts(name);
puts(surname);
```

Here, one statement,

```
printf("%s %s", name,surname);
```

prints two strings, while two statements as shown below,

```
puts(name);
puts(surname);
```

are required to print name and surname using puts().

Program :

```
/* Write a program to find length of a given string */
#include <stdio.h>
#include <conio.h>
main()
{
    char str[30];      /* array of size 30 defined */
    int i,count =0;    /* initialize count to zero */
    clrscr();
```

```
    printf("Give string \n");
    gets(str);
    for(i=0; str[i] != NULL;i++)
        count++;                    /* Increment count */
    printf("Length of string %s = %d\n",str,count);
}
```

Output :

```
Give string
vansh
Length of string vansh = 5
```

Explanation :

The count variable is initialized to zero. Then in for loop, we go on incrementing the count variable until the NULL value is found.

Program :

```
/* Write a program that takes a string and displays each character in one line with its ASCII value. */
#include <stdio.h>
#include <conio.h>
main()
{
      char str[30];     /* array of size 30 defined */
      int i;
      clrscr();
      printf("Give string \n");
      gets(str);
      for(i=0;i< str[i] != NULL;i++)
           printf("%c\t%d\n",str[i],str[i]);
}
```

Output :

```
Give string
Computer
C     67
o     111
m     109
p     112
u     117
t     116
e     101
r     114
```

Program :

```
/* Write a program that takes a string and reverses it. Display original string as well as reverse string. */
#include <stdio.h>
#include <conio.h>
main()
{
   char str[30], rev[30]; /* array of size 30 defined */
   int i,j,count=0;    /* initialize count to zero */
   clrscr();
   printf("Give string \n");
   gets(str);
   for(i=0;i< str[i] != NULL;i++)
              /* Get length of string in count */
      count++;
   for(i=count-1,j=0;i>=0;i--,j++)
   /* Last to first character stored in array rev */
      rev[j] = str[i];
   rev[j] = NULL;
      /* Terminate the reverse string with NULL */
   printf("Original string %s\t Reverse string %s\n",str,rev);
}
```

Output :

```
Give string
patan
Original string patan  Reverse string natap
```

Explanation :

Length of string is found, then starting from the last character up to the first character are stored in array **rev** using following the code:

```
for(i=count-1,j=0;i>=0;i--,j++)
    rev[j]  =  str[i];
```

Here, **i variable works with str array, while j variable works with rev array**. Variable i goes from count-1 to 0, while j starts from 0 and goes on incrementing till the loop is over.

Following statement terminates the reversed string by putting NULL at the end.

```
rev[j] = NULL;
```

8.10 STRING HANDLING BUILT-IN FUNCTIONS :

In this chapter, we have done many operations on strings by writing a code for getting length of string and reverse a string.

We can do other operations on strings like copy a string, concate two strings and get one string, or compare two strings for equality. 'C' language provides this functionality by providing built-in functions for doing string processing. We require to include **string.h** header file for using built-in string functions.

Following table shows the important built-in library function with its meaning

Name of function	Syntax	Meaning	Example
strlen	strlen(s)	Finds the length of string s.	l=strlen("CP&U"); returns string length 4 in variable 1.
strcpy	strcpy(dest,src)	Copies the string src to dest.	If dest="Jay" and src="kay" the strcpy(dest,src): makes dest="ka
strcat	strcat(s1,s2)	Concate string s2 at the end of string s1.	If s1="Jay" and s2="kay" then strcat(s1, s2); makes s1 = "Jayk
strcmp	strcmp(s1,s2)	Compares string s1 with s2. If both are equal, it returns 0. If s1 alphabetically > s2, it returns positive number, otherwise returns negative number.	If s1="Jay" and s2="kay" then strcmp(s1, s2); returns-1 becau "kay" is alphabetically greater t "jay"
strrev	strrev(s)	Reverses the string s, the original string is overwritten.	If s="Jay" then strrev(s): make s = "yaj"
strcmpi	strcmpi(s1,s2)	Compares string s1 with s2 **ignoring the case.** If both are equal, it returns 0. If s1 alphabetically > s2, it returns positive number, otherwise returns negative number.	If s1="Jay" and s2= "kay" then strcmp(s1, s2); returns-1 becaus "kay" is alphabetically greater th "jay"
strncmp	strncmp(s1,s2,n)	Compare first n characters of s1 and s2 and return result similar to strcmp	If s1="Jay" and s2="kay" then strcmp(s1, s2,2); returns - 1 bec "ka" is alphabetically greater th "Ja"
strupr	strupr(s)	Convert string s to uppercase	If s="Jay" then strupr(s); make s = "Jay"
strlwr	strlwr(s)	Convert string s to lowercase	If s="Jay" then strupr(s); make s="jay"
strstr	strstr(s1,s2)	Returns a pointer to the first occurrence of string s2 in s1	If s1="Jay" and s2="kay" then strstr(s1, s2); returns NULL be-ca string "kay" does not occur in "J

Program :

/* Program demonstrating built-in string functions. */

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
```

```
{
    int i;
    char str1[20]= "COMPUTER";    /* size 20 */
    char str2[20]= "computer";
    clrscr();
    printf("Length of string %s = %d\n",str1, strlen(str1));
       /* strlen(str1) finds length of string str1 */
    i = strcmp(str1,str2);
            /* compare two string str1 and str2 */
    if (i==0)          /* return value 0 if strings
                          equal otherwise not equal */
         printf("Two string are equal\n");
    else
         printf("Two string are not equal\n");
    printf("Reverse of %s is ",str1);
    strrev(str1);
         /* reverse string str1 and replace str1 */
    printf(" = %s\n", str1);
    strrev(str1);
     /* again reverse str1, so str1 has original value */
    printf("concate of %s and %s ",str1,str2);
    strcat(str1,str2);
      /* concate str2 after str1. size of str1 increases */
    printf(" = %s", str1);
}
```

Output :

```
Length of string COMPUTER = 8
Two string are not equal
Reverse of COMPUTER is = RETUPMOC
concate of COMPUTER and computer = COMPUTERcomputer
```

Program :

/* Program to extract M character from a string */

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

main()
{
    char  str[100];
```

```
    int  x,i,y,l;
    clrscr();
    printf("Enter  the  string\n");
    gets(str);
    l=strlen(str);          /*  length  of  string  */
    printf("How  many  characters  you  want  to  extract?    ");
    scanf("%d",&x);        /*  Number  of  characters  to  be  extracted
    printf("From  where  to  ?    " );
    scanf("%d",&y);          /*  from  position  */
    printf("The  extracted  part  of  string  is=\n");
    for(i=y-1;  i<y+x-1;  i++)
    printf("%c\n",str[i]);
    getch();
}
```

Output :

```
Enter  the  string
computer
How  many  characters  you  want  to  extract?  3
From  where  to  ?  3
The  extracted  part  of  string  is=
p
u
t
```

8.11 SOLVED PROGRAMMING EXAMPLES :

Program :

/* Write a program to search number x from given n numbers. If x is found, display the position within an array also. */

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a[10];     /* array of size 10 defined */
    int i,n,x;    /* x is the number to be searched */
          clrscr();
    printf("Give value of n (not more than 10)\n");
    scanf("%d",&n);  /* actual array size in n */
    for(i=0;i<n;i++)    /* input data */
       {
       printf("Give  number\n");
```

```
        scanf("%d",&a[i]);
        }
    printf("Which number to be searched?\n");
    scanf("%d",&x);
    for(i=0;i<n;i++)
        {
        if (a[i] == x)
            {
            printf("%d found at position %d\n",x,i+1);
            break;  /* number found, no need to
                       check other numbers */
            }
        }
    if ( i == n)
                        /* whole array searched */
        printf("%d not in list\n",x);
}
```

Output-1 :

```
Give value of n (not more than 10)
6
Give number
2
Give number
3
Give number
7
Give number
5
Give number
4
Give number
10
Which number to be searched?
9
9 not in list
```

Output-2 :

```
Give value of n (not more than 10)
6
Give number
2
Give number
3
Give number
7
Give number
5
Give number
4
Give number
10
Which number to be searched?
5
5 found at position 4
```

Explanation :

After storing the list and the number to be searched in x, we compare each number of list with x in for l
If number found, then we break the loop and print the message using the value of i for finding the locatio
x number in list.

Program :

/*Write a C program to accept a string and print every third character from string only if it is lo
case. */

```
#include<stdio.h>
#include<string.h>
main()
{
    char  s[20];
    int  i,len;
    printf("Give  one  string\n");
    gets(s);

    len  =  strlen(s);
        for(i=2;i<len;i=i+3)
        {
```

```
        if (islower(s[i])) /* is the character lowercase ? */
            printf("%c",  s[i]);
        }

}
```

Output 1 :

```
Give  one  string
CompuTER
m
```

Output 2 :

```
Give  one  string
Computer
mt
```

Program :

/* Write a program to insert a number at a specified array index position*/

```
#include <stdio.h>
#include <conio.h>
main()
{
   int a[10];     /* array of size 10 defined */
   int i,n,x,pos; /* x is the number to be inserted */
          /* pos is the index value at which number
          is to inserted */
   clrscr();
   printf("Give value of n (not more than 10)\n");
   scanf("%d",&n);   /* actual array size in n */
   for(i=0;i<n;i++)    /* input data */
       {
       printf("Give number\n");
       scanf("%d",&a[i]);
       }
   printf("Which number to be inserted?\n");
   scanf("%d",&x);
   printf("At what index value(< 10 and must be between 0 and %d)
   ?\n",n-1);
   scanf("%d",&pos);
   if(pos <0 || pos >n-1)
```

```
        {
        printf("Wrong input \n");
        return 1;
        }
    for(i=n;i>pos;i--)
        a[i] = a[i-1];
    a[pos] = x;
    n = n +1;
    printf("Array after insertion of %d at index %d\n",x,pos);
    for(i=0;i<n;i++)
        printf("%4d",a[i]);
}
```

Output :

```
Give value of n (not more than 10)
5
Give number
4
Give number
27
Give number
16
Give number
25
Give number
20
Which number to be inserted?
15
At what index value(< 10 and must be between 0 and 4)?
2
Array after insertion of 15 at index 2
   4   27   15   16   25   20
```

Explanation :

If we have to insert the number in the list at specific position, then we have to make that position vacan the numbers below that position need to be shifted down. For that to avoid loss of data we start shifting f the last number upto specified position. That is achieved by the code:

```
for(i=n;i>pos;i--)
      a[i] = a[i-1];
```

After the insertion of data in an array, the number of elements in array increases by 1, so the statement

```
n = n +1;
```

Program :

```
/* Write a program to read n * n matrix. Display original matrix as well as its transpose matrix.
Transpose matrix can be derived by interchanging ith row data with ith column data i.e ith row values become
ith column values, while ith column values become ith row values.
Example, if a[3][3] = 1 2 3  then its transpose matrix  =  1 4 7
                     4 5 6                                2 5 8
                     7 8 9                                3 6 9
*/
#include <stdio.h>
#include <conio.h>
main()
{
    int a[4][4];
    int n;
    /* assuming rows and columns same i.e n *n matrix*/
    int i,j;
    clrscr();
    printf("Give value of n\n");
    scanf("%d",&n);
    printf("Enter matrix A row-wise\n");
    for(i=0;i<n;i++)    /* Get first matrix data */
        {
        for(j=0;j<n;j++)
            {
            printf("a[%d][%d]=   ",i,j);
            scanf("%d",&a[i][j]);
            }
        printf("\n");
        }
    printf("Matrix A \n");
    for(i=0;i<n;i++)
                /* display first matrix row-wise */
        {
        for(j=0;j<n;j++)
            printf("%4d",a[i][j]);
        printf("\n");
        }
    printf("Transpose of A \n");
    for(j=0;j<n;j++)
                /* display second matrix row-wise */
```

```
        {
        for(i=0;i<n;i++)
        printf("%4d",a[i][j]);
        printf("\n");
        }
}
```

Output :

```
Give value of n
3
Enter matrix A row-wise
a[0][0]=  1
a[0][1]=  2
a[0][2]=  3
a[1][0]=  4
a[1][1]=  5
a[1][2]=  6
a[2][0]=  7
a[2][1]=  8
a[2][2]=  9
Matrix A
1   2   3
4   5   6
7   8   9
Transpose of A
1   4   7
2   5   8
3   6   9
```

Program :

/* Write a program to read n * n matrix and check for symmetric square matrix.
In symmetric square matrix, ith row data is same as ith column data.
In other words a[i][j] = a[j][i] for all value of i and j.
Example, if a[3][3] = 1 2 3 is symmetric square, because a[i][j]=a[j][i]
2 5 8
3 8 9
*/

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a[4][4];
```

```
int n;
/* assuming rows and columns same i.e n *n matrix*/
int i,j,flag=0;
clrscr();
printf(„Give value of n\n");
scanf("%d",&n);
printf("Enter matrix A row-wise\n");
for(i=0;i<n;i++)    /* Get matrix data */
    {
    for(j=0;j<n;j++)
        {
        printf("a[%d][%d]=   ",i,j);
        scanf("%d",&a[i][j]);
        }
    printf("\n");
    }
printf("Matrix A \n");
for(i=0;i<n;i++)    /* display matrix row-wise */
    {
    for(j=0;j<n;j++)
        printf("%4d",a[i][j]);
    printf("\n");
    }
for(i=0;i<n;i++)
    {
    for(j=0;j<n;j++)
        if (a[i][j] != a[j][i])
                        /* check condition */

            {
            flag = 1;
         /* set flag and come out of inner loop*/
            break;
                /* control goes to outer loop */
            }
     if(flag==1)    /* used to avoid unnecessary
                       iteration of outer loop */

        break;
     }
 if(flag ==1)
```

```
        printf("Given matrix is not symmetric\n");
    else
        printf("Given matrix is symmetric\n");
}
```

Output :

```
Give value of n
3
Enter matrix A row-wise
a[0][0]=  1
a[0][1]=  2
a[0][2]=  3
a[1][0]=  2
a[1][1]=  5
a[1][2]=  8
a[2][0]=  3
a[2][1]=  8
a[2][2]=  9
Matrix A
1   2   3
2   5   8
3   8   9
Given matrix is symmetric
```

Program :

/* Write a program that counts number of words in a string and also print each word on separate line. */

```
#include <stdio.h>
#include <conio.h>
main()
{
    char str[30];      /* array of size 30 defined */
    int i,count=1;   /* count initialized to 1,
                        minimum one word */
    clrscr();
    printf("Give string \n");
    gets(str);              /* string in str array */
```

```
    for(i=0;i< str[i] != NULL;i++)
        {
        if (str[i] == ' ' || str[i] == '\t' )
                    /* check for word separator */
            {
            printf("\n");              /* word over */
            count++;
            }
        else
            printf("%c",str[i]);
                /* simple character, print it */
        }
    printf("\nNumber of words = %d\n",count);
}
```

Output :

```
Give string
Gandhinagar is the capital of Gujarat
Gandhinagar
is
the
capital
of
Gujarat
Number of words = 6
```

Explanation :

Words are separated by spacebar or tab character. In for loop, we check for tab or space bar. If it is found, it means the previous word is over and new word starts, so count variable is incremented. If the character is other than tab or space bar, then it is printed as a part of the word on the same line.

Program :

```
/* Write a program that converts given string into upper case and lower case.
ASCII value of A = 65, B=66,...Z=90
ASCII value of a = 97, b=98,...z=122
So, difference between ASCII value of second  alphabet and first alphabet is 97-65 =32 */
#include <stdio.h>
#include <conio.h>
main()
```

```
{
    char str[20],ustr[20],lstr[20];
                    /* array of size 20 defined */
    int i,count=0;     /* count initialized to 0 */
    clrscr();
    printf("Give string \n");
    gets(str);          /* string in str array */
    for(i=0;i< str[i] != NULL;i++)
        {
        if ( str[i] >= 'a' && str[i] <='z')
                        /* check for lower case */
            ustr[i] = str[i] - 32;
                            /* convert to upper */
        else
            ustr[i] = str[i];
        if ( str[i] >= 'A' && str[i] <='Z')
                        /* check for upper case */
            lstr[i] = str[i] + 32;
                        /* convert to lower */
        else
            lstr[i] = str[i];
        }
    lstr[i] = NULL;
        /* put NULL character at end in both array*/
    ustr[i] = NULL;
    printf("\nOriginal String is %s\n",str);
    printf("\Upper Case String is %s\n",ustr);
    printf("\Lower Case String is %s\n",lstr);
}
```

Output :

```
Give string
Computer
Original String is Computer
Upper Case String is COMPUTER
Lower Case String is computer
```

Explanation :

Here, 'a' stands for ASCII value of character a (i.e 97), same way 'z' stands for ASCII value of character z (i.e 122). Similarly for 'A' =65 and 'Z' =90. The array **ustr** stores the uppercase string, while array **lstr** store the lowercase string.

Program :

/* Write a menu driven program to perform string operations like length, reverse, copy, concate and compare using built-in string functions */

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
    char str1[20],str2[20];
                    /* array of size 20 defined */
    int choice,len;
    clrscr();
    do
        {
        printf("1. String length      2. String reverse\n");
        printf("3. String copy   4. String Concate\n");
        printf("5. String compare   6. Exit\n");
        printf("Give your choice(1 to 6)\n\n");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
        case 2:
        case 3:
            printf("Give string\n");
            scanf("%s",str1);
            if (choice ==1)
                {
                len = strlen(str1);
                printf("Length = %d\n",len);
                break;
                }
            else if (choice ==2)
                {
                strcpy(str2,str1);
                strrev(str2);
                printf("Reverse string = %s\n",str2);
                break;
                }
```

```
            else
                {
                strcpy(str2,str1);
                printf("Copied string = %s\n",str2);
                break;
                }
        case 4:
        case 5:
            printf("Give first string\n");
            scanf("%s",str1);
            printf("Give second string\n");
            scanf("%s",str2);
            if (choice ==4)
                {
                strcat(str1,str2);
                printf("Concated string = %s\n",str1);
                break;
                }
            else
                {
                if (strcmp(str1,str2) == 0)
                    {
                    printf("Two strings are equal\n");
                    break;
                    }
                else
                    {
                    printf("Two strings are not equal\n");
                    break;
                    }
                }
        }
    } while (choice != 6);
}
```

Output-1 :

```
1. String length      2. String reverse
3. String copy        4. String Concate
5. String compare     6. Exit
```

```
Give your choice(1 to 6)
1
Give string
computer
Length = 8
1. String length      2. String reverse
3. String copy        4. String Concate
5. String compare     6. Exit
Give your choice(1 to 6)
6
```

Output-2 :

```
1. String length      2. String reverse
3. String copy        4. String Concate
5. String compare     6. Exit
Give your choice(1 to 6)
4
Give first string
computer
Give second string
keyboard
Concated string = computerkeyboard
1. String length      2. String reverse
3. String copy        4. String Concate
5. String compare     6. Exit
Give your choice(1 to 6)
5
Give first string
computer
Give second string
mouse
Two strings are not equal
1. String length      2. String reverse
3. String copy        4. String Concate
5. String compare     6. Exit
Give your choice(1 to 6)
6
```

Explanation :

do...while loop is used to repeatedly display the menu, the loop is over when user gives the choice 6. In do
while loop, the switch statement checks for choice and accordingly it asks for either one input string or two str
input.

For choice 1, 2 & 3 only one string is required, while choice 4 and 5, two strings are required. Then accordi
to the choice given by user, appropriate string function is called and the work is done and message printed usi
printf() statement.

Program :

```
/* Write a program to store n number of strings and then search the given string. */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
    char names[10][20];
    char str[20];
    int i,n;
    int flag =0;
    clrscr();
    printf("How many strings\n");
    scanf("%d",&n);
    printf("Enter strings\n");
    for(i=0;i<n;i++)
        scanf("%s",names[i]);
    printf("Given strings are:\n");
    for(i=0;i<n;i++)
        printf("%s\t",names[i]);
    printf("\nEnter the string you want to search\n");
    scanf("%s",str);
    for(i=0;i<n;i++)
        {
        if (strcmp(names[i],str) == 0)
            {
            flag =1;
            break;
            }
        }
    if (flag ==1)
        printf("Found !! Given string %s found in list\n",str);
    else
        printf("Sorry !! Given string %s not found in list\n",str);
}
```

Output-1 :

```
How many strings
5
Enter strings
mumbai
ahmedabad
patan
surat
mahesana
Given strings are:
mumbai    ahmedabad    patan    surat    mahesana
Enter the string you want to search
patan
Found !! Given string patan found in list
```

Output-2 :

```
How many strings
5
Enter strings
mumbai
ahmedabad
patan
visnagar
surat
Given strings are:
mumbai    ahmedabad    patan    visnagar    surat
Enter the string you want to search
gandhinagar
Sorry !! Given string gandhinagar not found in list
```

Explanation :

To store more than one string, we require to use two-dimensional array of characters. The line

```
char names[10][20];
```

declares 2-dimensional array which can store 10 names, each name can be of maximum 20 characters. It means that above declaration is of 10 arrays each of size 20 characters.

The code

```
for(i=0;i<n;i++)
    scanf("%s",names[i]);
```

gets n different strings from user & stores them in an array. **Here, names[i] is equivalent to &names[i][0].** So, there is no need to put & symbol before names[i]. Why? It will become clear when we study the pointers in detail.

The string to be searched is stored in array str. Then, the string str is searched one by one from names[][]. string is found, flag variable is made 1 and loop terminated by break i.e there is no need to search remainin numbers. Appropriate message printed, which depends on the value of flag variable.

/* Write a program which accepts a long string and print only characters at positions which are multiple of 3. */

```
#include<stdio.h>
#include<conio.h>
#include  <string.h>

void  main()
{
    char  str[25];
    int  i=2,len;  /*  i  indicates  third  character  */
    clrscr();
    printf("Enter  a  string:    ");
    gets(str);
    len=  strlen(str);

    while  (i<len)
        {
        putchar(str[i]);
        i  =  i+3;  /*  next  multiple  of  three  */
        }
    getch();
}
```

Output :

```
Enter  a  string:    computer
mt
```

: SUMMARY :

- **Array** is a collection of variables of same data type, which is known by a single name. We refer individ elements of an array by their subscript value or index value. Array can be – **single dimensional** or **mu dimensional.**
- **Array index** starts from 0. So, if an array size is 10, and then the index starts from 0 and goes up 9.
- In the program itself if we assign values to individual elements, it is called **initialization of an array.** can be done at the time of declaration of an array or by specific assignment to individual elements separate in the program.
- **Multi dimensional array** is used when we want to store tabular information in an array. Two dimension array is a matrix of m * n size, where m is the rows and n is the columns.
- **String** is a sequence of characters enclosed in double quotes. We can read a string by using %s form **specifier** in scanf() function or by using **gets()** function. We can print a string by using %s format specifi in printf() function or by using **puts()** function.

String handling built in functions are defined in **string.h** header file. We need to include string.h file for using built in string functions.

Some of the **string operations** are: Concate, copy, find length, compare two strings, reverse, convert to upper case, convert to lower case, find substring etc.

: MCQs :

Which declaration of array is correct?

(a) int a(10); (b) int a{10}; (c) int a[10]; (d) int [10] a

What will happen if you assign a value to an array element whose subscript value exceeds the size of array?

(a) Compiler will report an error (b) It may crash the program

(c) Size will be automatically adjusted (d) 0 value assigned

What is the size of an array which is initialized like?

int a[] = { 31,28,31,30,31,30,31,31,30,31,30,31};

(a) Syntax error (b) 0 (c) 12 (d) 11

Which declaration of two dimensional array is correct?

(a) sales[5][5]; (b) sales [5,5]; (c) sales (5,5) (d) None of above

In declaration int a[12][10]; how many elements are declared?

(a) 22 (b) 99 (c) None of above

If we initialize the two dimensional array as

int a[3][2] = { 0,0,1,1};

what will be the value of a[2][1]?

(a) Syntax error in initialization (b) 0

(c) Random value (d) 1

Every string is terminated by NULL character. How it is represented?

(a) '\0' (b) NULL (c) Both a and b (d) None of above

If we want to read a sentence having blanks, tabs we need to use

(a) scanf() (b) gets() (c) Both a and b

How many arguments are required by built–in strlen() function?

(a) 1 (b) 2 (c) 3 (d) 0

Built–in function strcpy(str1,str2) copies string

(a) str1 to str2 (b) str2 to str1 (c) None of above

Array index start at

(a) 1 (b) User defined (c) 0 (d) None of above

String handling built–in functions are defined in

(a) conio.h (b) stdio.h (c) stdlib.h (d) None of above

If an array is declared as double a[5], how many bytes are allocated?

(a) 5 (b) 10 (c) 20 (d) 40

ASCII code for a–z ranges from

(a) 0–26 (b) 97–123 (c) 65–91 (d) None of above

15. Function to add a string at the end of another string is
    (a) stradd() (b) strcpy() (c) strcat (d) strstr()
16. Which of the following function is more appropriate for reading in a multiword string?
    (a) printf() (b) scanf() (c) gets() (d) puts
17. What will be the output of following program

```
#include
main()
{
int x, y = 10;
x = y * NULL;
printf(\"%d\",x);
}
```

    (a) error (b) 0 (c) 10 (d) garbage value
18. ASCII code for A–Z ranges from
    (a) 0–26 (b) 97–123 (c) 65–91 (d) None of above
19. ASCII code for 0–9 ranges from
    (a) 0–26 (b) 97–123 (c) 65–91 (d) 48–57

: ANSWERS :

1	(c)	2	(b)	3.	(c)	4.	(a)	5.	(c)	6.	(b)	7	(c)
8.	(b)	9.	(a)	10.	(b)	11	(c)	12	(d)	13.	(c)	14.	(b)
15.	(c)	16	(c)	17	(a)	18.	(c)	19.	(d)				

: EXERCISE :

1. What is an array? What are its types?
2. For the declaration as

```
int a[10];
```

   the last element is stored at a[9]. Is it TRUE? Explain your answer.
3. Declare an array to store a matrix of integer numbers having 4 rows and 3 columns.
4. For following initialization, what is the size of array ?

```
int a[] = {30,20,30,25,40,45,35};
```

5. Find errors if any in following code
   (i) `int num[] = { 1,2,3,4,5};`
   (ii) `char city = {'V','i','s','n','a','g','a','r'};`
   (iii) `float f[3] = { 1.3,3.4,2.5};`
   (iv) `double d=0;`
   (v) `double db[] = 0;`
6. What is a string? What are the operations that can be performed on string?
7. Write any 5 built-in string functions with their syntax and meaning.
8. Write a program to multiply two matrices A and B of size 3 * 3 and store the answer in matrix C. i.e C = A * B.
9. Write a program to count how many times a particular character occurs in a given string. e.g. character 'a' occu

2 times in string "Gujarat".

Write a program to check whether the given string is palindrome or not without using library functions. Palindrome strings are string which spell the same after reversal also. Following strings are palindrome: madam, malayalam, abba.

Describe various string handling functions with sample 'C' code.

Write a program that read two strings and compare them using the function strcmp() and print the message that first string is equal, less, greater than second one.

Write a 'C' program to read an array of integers and print its elements in reverse order.

Find out the error if any in following code and correct it and give output.

```
void main()
{
   int iNo[3] = {10,20,30,40};
   char cName[]="exam";
   char cMidd[]="university";
   strcpy(cName,cMidd);
   printf("%d",iNo[0]);
   printf("%s",cName);
}
```

Explain following string manipulation functions.

`strncmp(), strlen(), strcat(), strstr(), strchar()`

: Answers to selected exercises :

What is a string? What are the operations that can be performed on string?

String is a sequence of characters enclose in double quotes. Normally string is useful for storing data like name, address, city etc. ASCII code is internally used to represent string in memory. In 'C' each string is terminated by a special character called null character. In 'C', null character is represented as '\0' or NULL. Because of this reason, the character array must be declared one size longer than the string required to be stored.

C	O	M	P	U	T	E	R	\0

Here, the string stored is "COMPUTER", which is having only 8 characters, but actually 9 characters are stored because of NULL character at the end.

Operations on strings:

1. String Concatenation
2. String Copy
3. String Length
4. Compare two string
5. Reverse a string
6. Convert string to upper case
7. Convert string to lower case

8. Write a program to multiply two matrices A and B of size 3 * 3 and store the answer in matrix C. i.e C = A * B.

Ans:

```
#include<stdio.h>
void main()
{
      int a[3][3], b[3][3], c[3][3];
      int m,n,o,p;
      int i,j,k;
      clrscr();
      printf("how many rows in first matrix?\n");
      scanf("%d",&m);
      printf("how many columns in first matrix?\n");
      scanf("%d",  &n);
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
      {
      printf("Enter value of a[%1d][%1d]\n",i+1,j+1);
      scanf("%d",&a[i][j]);
      }
      printf("how many rows in second matrix?\n");
      scanf("%d",&o);
      printf("how many columns in second matrix?\n");
      scanf("%d",  &p);
      if (n !=o ) /* number of columns of A and number
                  of rows in B should match */
            {
            printf("Multiplication not possible.
                  The columns of first\n");
            printf("matrix should be same as rows of
                  second\n");
            return;
            }
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
      {
            printf("Enter value of a[%1d][%1d]\n",i+1,j+1);
            scanf("%d",&a[i][j]);
            }
      for(i=0;i<o;i++)
```

```
        {
        for(j=0;j<p;j++)
        {
        printf("Enter value of
        b[%ld][%ld]\n",i+1,j+1);
        scanf("%d",&b[i][j]);
    }
  for(i=0;i<m;i++) /* C matrix of size  m*p*/
        {
        for(j=0;j<p;j++)
        {
              c[i][j]=0;
              for(k=0;k<n;k++)
                    c[i][j]= c[i][j]+ a[i][k]*b[k][j];
        }
        }
  printf("Matrix A\n");
  for(i=0;i<m;i++)
  {
  for(j=0;j<n;j++)
        printf("%4d ",a[i][j]);
     printf("\n");
  }
        printf(„Matrix B\n");
  for(i=0;i<o;i++)
  {
  for(j=0;j<p;j++)
        printf(„%4d „,b[i][j]);
     printf(„\n");
  }
        }
  printf("Matrix C= A*B\n");
  for(i=0;i<m;i++)
  {
        for(j=0;j<p;j++)
              printf(„%4d „,c[i][j]);
        printf("\n");
  }
}
```

12. Write a program that read two strings and compare them using the function strcmp() and print the message that first string is equal, less, greater than second one.

Ans:

Following program explains the use of strcmp() built-in library function.

```
#include<stdio.h>
void main()
{
      char str1[50], str2[50];
      int ans;
      clrscr();
      printf("Give first string\n");
      gets(str1);

      printf("Give second string\n");
      gets(str2);

      ans = strcmp(str1,str2);
      if (ans == 0)  /* both equal */
            printf("Both string %s and %s are
            equal\n",str1,str2);
      else if (ans > 0) /* first greater */
            printf("First string %s is greater than second
            %s\n",str1,str2);
      else /* second greater */
            printf("First string %s is less than second
            %s\n",str1,str2);
}
```

13. Write a 'C' program to read an array of integers and print its elements in reverse order.

Ans:

```
#include<stdio.h>
void main()
{
      int i;
      int a[100];
      int n;
      printf("How many numbers? \n");
      scanf("%d", &n);
      for(i=0;i<n;i++)
            {
            printf("\nGive number %d: ", i+1);
```

```
        scanf("%d", &a[i]);
        }
    printf("\nThe list in reverse order is\n");
    for(i=n-1;i>=0;i--)
        printf("\n%d", a[i]);
}
```

Find out the error if any in following code and correct it and give output.

```
void main()
{
int iNo[3] = {10,20,30,40};
char cName[]="exam";
char cMidd[]="university";
strcpy(cName,cMidd);
printf("%d",iNo[0]);
printf("%s",cName);
}
```

s:

```
int iNo[3] = {10,20,30,40}; replace this line by
int iNo[4] = {10,20,30,40};
```

The output will be

```
10
university
```

. Explain following string manipulation functions.

strncmp(), strlen(), strcat(), strstr(), strchar()

ns:

Name of function	Syntax	Meaning
strncmp	strncmp(s1,s2,n)	Compare first n characters of s1 and s2 and return result similar to strcmp
strlen	strlen(s)	Finds the length of string s.
strcat	strcat(s1,s2)	Concate string s2 at the end of string s1.
strstr	strstr(s1,s2)	Returns a pointer to the first occurrence of string s2 in s1
strchr	strchr(s,c)	Finds and returns the pointer to the first occurrence of character c in string s, otherwise returns NULL

: SHORT QUESTIONS :

1. **What is an array?**

Array is a collection of variables of same type known by a single name. The individual elements of an array are referred by their index value.

2. **Can we initialize an array in the program itself?**

Yes.

3. Which function should be used to read a sentence having words, blank, tab etc?

⇒ Function gets()should be used to read a sentence having words, blank, tab etc.

4. Which built in functions reverses a string?

⇒ Built in function strrev(s) reverse a string, s is the string to be reversed.

5. Which built in function appends a string at the end of a string?

⇒ Built in function strcat(s1,s2) appends a string at the end of a string, s2 string is concatenated at the end of string s1.

6. Is it possible to have negative index in an array?

⇒ Yes it is possible to index with negative value. It is illegal to refer to the elements that are out of array bounds, the compiler will not produce error because 'C' has no check on the bounds of an array.

7. Why is it necessary to give the size of an array in an array declaration?

⇒ When an array is declared, the compiler reserves enough space in memory for all the elements of the array. The size is required to allocate the required space and hence size must be mentioned.

❖

CHAPTER

9

Pointers

9.1 WHY POINTERS ?

9.2 CONCEPT OF POINTERS

9.3 POINTER DECLARATION

9.4 POINTER ARITHMETIC

9.5 POINTER TO POINTER

9.6 POINTERS AND ARRAYS

9.7 ARRAY OF POINTERS

9.8 POINTERS AND STRINGS

- SUMMARY
- MCQs
- EXERCISES AND ANSWERS TO SELECTED EXERCISES
- SHORT QUESTIONS

We have used variables which store different types of values depending on its type. Pointer is a variable that stores the address of another variable within the memory. So, we can say that pointer is a special type of variable which store the address of another variable as its value.

9.1 WHY POINTERS ?

- It enhances the capability of the language in manipulating the data.
- It reduces the size of the program.
- It is used for creating data structures such as linked lists, trees, graphs etc.
- It is used to pass information between different functions in both directions.
- It increases the execution speed of the program.
- If used with array of strings, then it reduces the storage space requirement.
- It allows working with dynamic memory allocation.

9.2 CONCEPT OF POINTERS :

All the variables stored in memory occupy some storage space in main memory of computer. Each location in main memory of computer is having its address. Each location in memory stores one byte of data. Multi-byte data types like integer, float, double requires more than one byte of storage space. Character requires 1 byte; integer requires 2 bytes; float requires 4 bytes and double requires 8 bytes. The addresses are sequential starting from 0.

For example, int a=2;

The compiler will automatically allocate some memory for this variable **a**. So, location where the value of **a** is stored, is called as the address of variable **a**. If the variable **a** is stored at location 100, then we cay say that the address of variable **a** is 100, while the value of variable **a** is 2.

a	←	name of variable
2	←	value
100	←	Address of variable

Now, if the 100 value is stored in other variable, then that variable can be called as the pointer to variable a, because that variable will hold the address of variable **a** as its value.

ptr	←	name of variable
100	←	value
450	←	Address of variable

As shown in above figure, variable **ptr** holds value 100 (which is the address of variable **a)** and the **ptr** is stored at location 450. Here, we can say that **ptr** variable is a pointer to the variable **a.**

How to assign the address of variable to other variable ?

To access the memory address of variable, the operator & is used, which is called as the address operator. We can assign the address of variable **a** to another variable **ptr** using & operator as :

```
ptr = &a;
```

This statement will assign the address of a (which is 100 in above example) to variable **ptr**. Thus, **ptr** variable is a pointer to variable **a.**

How to access value of variable using pointer variable ?

We can access the value of variable **a,** through the pointer variable **ptr.** We need to use the operator * (called as derefencing operator or indirection operator). So, *ptr (i.e * operator before pointer variable), represents

variable **a** in above example, it effectively means that later in the program, variables **a** and ***ptr** represent same value (2 in above example). **We can use *ptr or a to refer to the value of variable a,** because **ptr** is the pointer to **a**.

POINTER DECLARATION :

We have seen that pointer is a variable which store the address of another variable. But, how to declare a pointer variable? In above example, we have seen that **ptr** is a pointer, because we have stored the address of variable a. But, we have not mentioned how ptr variable is declared.

Syntax for pointer declaration is :

```
data_type  *varname;
```

where data_type means pointer to which type of data, varname is the name of a variable, while * symbol before the name indicates that the variable is a pointer type variable.

For above example,

```
int  *ptr;
```

declaration is needed because **ptr** variable stores the address of variable **a** (which is integer). This declaration says that **ptr** variable can store the address of any integer variable.

The declaration,

float *fptr; means that fptr is a pointer which points to float type of variable,

char *cptr; means that cptr is a pointer which points to char type of variable.

Program :

/* Write a program to print the address of variable and its value */

```
#include<stdio.h>
#include<conio.h>
main()
{
char ch = 'a';
float fl = 30.5;
int i = 50;
clrscr();
printf("Name\t\tValue\t\tAddress\n");
printf("ch\t\t%c\t\t%X\n",ch,&ch);
printf("fl\t\t%f\t\t%X\n",fl,&fl);
printf("i\t\t%d\t\t%X\n",i,&i);
}
```

Output :

```
Name          Value       Address
ch        a               FFF5
fl        30.500000  FFF0
i         50              FFEE
```

As in the case of variable declaration and initialization, we can also initialize pointer at the time of declaration.

For example,

```
int  n  =  10;
int  *iptr  =  &n;
```

The above code declares variable **n** with value 10 and pointer variable **iptr** which is also initialized with the address of variable **n**.

Now, the statement, *iptr = 20; will make the value of n = 20.

Consider following code,

```
int a=10;
int *ptr1 = &a;     /* ptr1 points to a */
int *ptr2;
ptr2=ptr1;          /* ptr2 also points to a*/
a = a+1;            /* a=11 */
*ptr1  = *ptr1 +1;     /* a=12 */
*ptr2  = *ptr2 +1;     /* a=13 */
```

After above statements are executed value of a will be equal to 13. From above code it is very clear that a = *ptr1 = *ptr2.

Another important point is that pointer must be initialized before using it.

These concepts are used in following program.

Program :

/* Write a program to access the data using pointer variables*/

```
#include<stdio.h>
#include<conio.h>
main()
{
char ch = 'a';
float fl = 30.5;
int i = 50;
char *cptr;        /* declare pointers */
float *fptr;
int *iptr;
cptr = &ch; /* assign addresses, pointers initialized*/
```

```
fptr= &fl;
iptr = &i;
clrscr();
printf("Using variables\n\n");
printf("Name\t\tValue\t\tAddress\n");
printf("ch\t\t%c\t\t%X\n",ch,&ch);
printf("fl\t\t%f\t\t%X\n",fl,&fl);
printf("i\t\t%d\t\t%X\n",i,&i);
printf("\nUsing pointer variables\n\n");
printf("ch\t\t%c\t\t%X\n",*cptr,cptr);
                /* access variable through pointers */
printf("fl\t\t%f\t\t%X\n",*fptr,fptr);
printf("i\t\t%d\t\t%X\n",*iptr,iptr);
*cptr = *cptr +1;    /* increment ch through pointer*/
*fptr = *fptr +1;   /* increment fl through pointer */
*iptr = *iptr +1;   /* increment i through pointer */
printf("\nAfter incrementing by 1 using pointer\n\n");
printf("ch\t\t%c\t\t%X\n",ch,&ch);
printf("fl\t\t%f\t\t%X\n",fl,&fl);
printf("i\t\t%d\t\t%X\n",i,&i);
}
```

Output :

```
Using variables
Name        Value        Address
ch      a              FFF5
fl      30.500000      FFF0
i       50             FFEE
Using pointer variables
ch      a              FFF5
fl      30.500000      FFF0
i       50             FFEE
After incrementing by 1 using pointer
ch      b              FFF5
fl      31.500000      FFF0
i       51             FFEE
```

9.4 POINTER ARITHMETIC :

As pointer variables store the address value, which are numbers, we can do arithmetic operations on poin variables.

Some valid arithmetic operations on pointers are :

- Increment
- Decrement
- Adding integer number to pointer variable
- Subtracting integer number from pointer variable
- Subtracting one pointer from other pointer if pointing to same array

But, following arithmetic operations are not valid

- Addition of two pointers
- Multiplication operation with any number
- Division operation with any number

Following table explains valid operations. **ptr** is a pointer type variable.

Operation	Expression	Initial	New value of ptr for			Meaning
			Character	Integer	Float	
Increment	ptr++;	100	101	102	104	Increment to next location of same type
Decrement	ptr--	100	99	98	96	Decrement to next location of same type
Adding integer number	ptr=ptr+5;	100	105	110	120	Points to 5 locations after current location of same data type
Subtracting integer number	ptr=ptr-3;	100	97	94	88	Points to 3 locations before current location of same data type
Subtraction of one pointer from another	ptr=ptr1-ptr2;	ptr1=105 ptr2=100	5	5	5	All pointers must point to the same array

Consider following code,

```
int a=10;
int *ptr1 = &a;                    /* ptr1 points to a */
a = a+1;                           /* a=11 */
++*ptr1;   /* a=12, Here, bracket is not needed */
(*ptr1)++   /* a=13 Here, bracket is needed */
```

Here, ++ *ptr1; statement evaluation takes place as ++(*ptr1) i.e the content where ptr1 points is incremented by 1. But, (*ptr1)++; statement requires brackets, because ++ has higher precedence than * operator. If we forget to put brackets, pointer address will be incremented first, and then value at that new address will be referred.

If we execute following code fragment:

```
main()
{
        int a=3, b=5,c, *p,*q;
        p = &b; q=&a;
        c = *p % *q;
        ++ (*p);
        printf("%d %d\n" ,*p, *q);
        printf("\n %d %d ",c,b);
}
```

In above program,

Statement	Effect
`int a=3, b=5,c, *p,*q;`	a =3, b=5
`P = &b; q=&a;`	p points to b and q points to a, so *p =b=5 and *q =a=3
`C = *p % *q;`	c = 5% 3 = 2
`++ (*p);`	*p = *p +1 =5 + 1 =6, so b becomes 6 (*p is other way of referring b)
`printf("%d %d\n" ,*p, *q);`	prints 6 3
`printf("\n %d %d ",c,b);`	prints 2 6

So, final output of above code will be

6 3

2 6

Program :

/* Write a program to demonstrate pointer arithmetic */

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch ='a', *cptr;
    float fl = 30.5 , *fptr;
    int i =50, *iptr;
    clrscr();
    cptr = &ch;
    fptr = &fl;
    iptr = &i;
    printf(„\nchar    float int\n");
    printf(„%x  %x    %x\n",cptr,fptr,iptr);
    cptr++;        /* increment pinters*/
    fptr++;
    iptr++;
    printf("\nAfter increment operation");
    printf("\nchar    float int\n");
    printf("%x  %x    %x\n",cptr,fptr,iptr);
    cptr–;        /*decrement pointers */
    fptr–;
    iptr–;
    printf("\nAfter decrement operation");
    printf("\nchar    float int\n");
    printf("%x  %x    %x\n",cptr,fptr,iptr);
    cptr = cptr +2;    /* addition by number 2*/
    fptr = fptr +2;
    iptr = iptr +2;
    printf("\nAfter adding by number 2");
    printf("\nchar    float int\n");
    printf("%x  %x    %x\n",cptr,fptr,iptr);
    cptr = cptr -4;    /*subtract by number 4 */
    fptr = fptr -4;
    iptr = iptr -4;
    printf("\nAfter decrementing by number 4");
    printf("\nchar    float int\n");
    printf("%x  %x    %x\n",cptr,fptr,iptr);
}
```

Output :

```
char        float          int
fff5        fff0           ffee
After increment operation
char        float          int
fff6        fff4           fff0
After decrement operation
char        float          int
fff5        fff0           ffee
After adding by number 2
char        float          int
fff7        fff8           fff2
After decrementing by number 4
char        float          int
fff3        ffe8           ffea
```

Comparison of pointers :

In an expression involving pointers, relational operators can be used if the pointer variables belong to same data type.

Example,

```
int *ptr1, *ptr2;
```

then expression if (ptr1 > ptr2) is valid because both pointers point to same data type.

Normally, comparison of pointers is useful when we are dealing with pointers which point to the different elements of the same array.

POINTER TO POINTER :

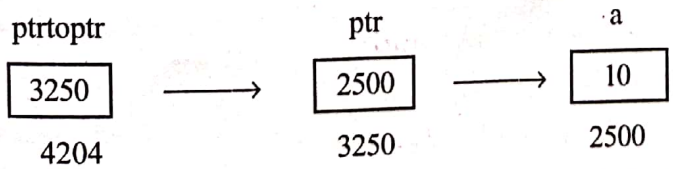
As the name suggests it is a pointer variable, which stores the address of another pointer variable. This type of variable can be declared using the syntax:

data_type **ptrtoptr; where ** indicates that variable named ptrtoptr is a pointer to pointer of data_type.

Example,

```
int a =10;
int * ptr= &a;
int **ptrtoptr;
ptrtoptr = &ptr;
```

above code explains the following relationship between variables in memory.



Here, **ptrtoptr** variable stores the address of **ptr** variable which is 3250, while **ptr** variable stores the address of variable **a**, which is 2500.

```
*ptrtoptr = ptr  and *ptr =a. So, **ptrtoptr =a.
```

Program :

```
/* Write a program to demonstrate pointer to pointer arithmetic. */
#include<stdio.h>
#include<conio.h>
main()
{
    int i = 50, *iptr, **iptrtoptr;
    clrscr();
    iptr = &i;          /* address of i in iptr */
    iptrtoptr = &iptr;
                /* address of iptr in iptrroptr */
    printf("iptrtoptr\t\tiptr\ti\n");
    printf("%d\t\t%d\t%d\n",**iptrtoptr,*iptr,i);
                            /* print values */
    printf("%x\t\t%x\t%x\n",&iptrtoptr,&iptr,&i);
                            /* print addresses */
}
```

Output :

```
iptrtoptr    iptr      i
   50         50       50
   fff0       fff2     fff4
```

9.6 POINTERS AND ARRAYS :

Pointers and arrays are very much related. When we studied array, at that time we did not mention ab pointers. But, every program written using arrays can always be written using pointers. As we have seen t pointers make our program small and faster but too much use of pointer complicates the program to understa and debug. As was mentioned in the chapter on strings and arrays, the name of the array variable itself wo as the address of array. By address of an array, we mean the address of the first element of the array.

Consider following declaration of an array :

```
int  a[5];
```

Here, the array name **a** works as an address of first element a[0]. The subscripted variable a[i] is interna represented using a pointer as *(a+i) by 'C' language. This is an example of pointer arithmetic, where a is t pointer variable and i is some integer number.

This can be shown pictorially as below.

a[0] or *(a+0)	100
a[1] or *(a+1)	102
a[2] or *(a+2)	104
a[3] or *(a+3)	106
a[4] or *(a+4)	108

In above figure, the address of array i.e address of first element a[0] is assumed as 100. Each integer occupi 2 bytes, so a[0] is at 100, a[1] is at location 102 and like that.

One point to remember about the name of an array is that it works as a pointer to first element of an array, but it is a constant pointer i.e its value can not be changed. So, we can not assign any other variables address to array name, a in our example. It also means that pointer arithmetic can not be done with array name.

So, if we want to access the individual elements of an array using pointer, we need to declare a pointer variable in the program, and we can do all the valid arithmetic operations on it.

Following program explains how we can access individual elements of an array using pointer variable.

gram :

```
/* Write a program to access array elements using pointer. */
#include<stdio.h>
#include<conio.h>
main()
{
   int a[10],i,n;
   int *ptr;
   clrscr();
   ptr =a;
   /* initialize ptr to start address of array  a*/
   printf("How many numbers?\n");
   scanf("%d",&n);
   printf("Give numbers\n");
   for(i=0;i<n;i++)
   {
   scanf("%d",ptr);
   ptr++;    /* go to next location in array */
   }
   ptr =a; /* initialize ptr to start address of array a*/
   printf("Array elements are:\n");
   for(i=0;i<n;i++)
       {
       printf("%d\n",*ptr); /* print array element */
       ptr++; /* go to next location in array */
       }
}
```

utput :

```
How many numbers?
4
Give numbers
1
2
```

```
4
3
Array elements are:
1
2
4
3
```

Above program can also be written as below, where a[i] = *(a+i) concept is used.

Program :

```
/* Write a program to print array elements in reverse using pointer.*
#include<stdio.h>
main()
{
        int a[10],i,n;
        int *ptr;
        clrscr();
        ptr =a;
        printf("How many numbers?\n");
        scanf("%d",&n);
        printf("Give numbers\n");
        for(i=0;i<n;i++)
        {
            scanf(„%d",ptr);
            ptr++;
        }
        ptr =a+n-1;
            /*points to the last element of array */
        printf("Array elements in reverse are:\n");
        while (ptr >= a)
        {
            printf("%d\n",*ptr);
            ptr–;
        }
}
```

Output :

```
How many numbers?
5
Give numbers
23
12
```

```
44
55
343
Array elements in reverse are:
343
55
44
12
23
```

Program :

```
/* Write a program to access array elements using pointer. */
#include<stdio.h>
#include<conio.h>
main()
{
    int a[10],i,n;
    clrscr();
    printf("How many numbers?\n");
    scanf("%d",&n);
    printf("Give numbers\n");
    for(i=0;i<n;i++)
        {
        scanf("%d",(a+i));
        }
    printf("Array elements are:\n");
    for(i=0;i<n;i++)
        {
        printf("%d\n",*(a+i));
        }
}
```

Output :

```
How many numbers?
4
Give numbers
1
2
4
3
Array elements are:
1
2
4
3
```

9.7 ARRAY OF POINTERS :

As we have an array of char, int, float etc, same way we can have an array of pointers, individual elements of an array will store the address values. So, array of pointers is a collection of pointers of same type known by single name.

Syntax is :

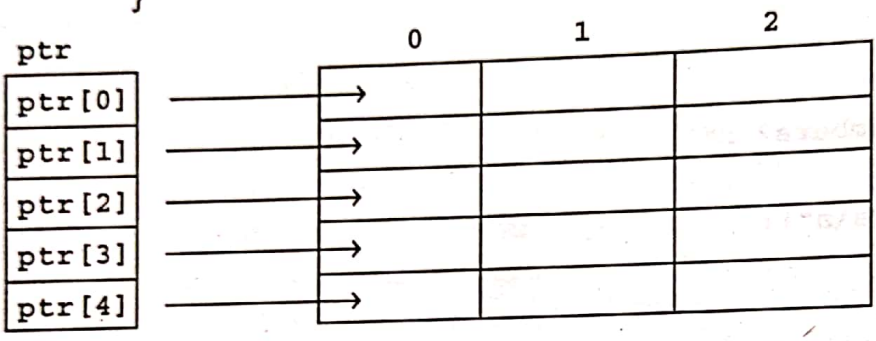
```
data_type  *name[size];
```

Example :

```
int *ptr[5]; declares an array of integer pointers of size 5
int mat[5][3]; declares a matrix of 5 rows and 3 columns
```

Now, the array of pointers **ptr** can be used to point to different rows of matrix as follow

```
for(i=0;i<5;i++)
    {
     ptr[i] = &mat[i][0];
     }
```



As stated above, by using dynamic memory allocation, we do not require to declare two-dimensional array, i can be created dynamically using array of pointers.

9.8 POINTERS AND STRINGS :

As we have studied in earlier chapter, string is a sequence of characters. It is a single dimensional array o characters. We can use pointers in string processing operations. Following program explain the use of pointer to manipulate strings.

Program :

/* Write a program to manipulate string using pointers. */

```
#include<stdio.h>
main()
{
    char str[20];
    int i,n,count=0;
    char *cptr;
    clrscr();
    printf("Enter some string\n");
    gets(str);              /* get string */
    cptr = str;
```

```
    /* cptr now points to first character of string */
    while(*cptr != '\0')   /* count length of string */
    {
        count++;
        cptr++;
    }
    printf("Length of string %s is = %d\n",str,count);
    cptr--;     /* set cptr to last character of
                   string from NULL character */
    printf("reverse of string %s is ",str);
    while(count >0)
    {
        putchar(*cptr);
        /* print from last to first character */
        count--;
        cptr--;
    }
}
```

tput :

```
Enter some string
Programming
Length of string Programming is = 11
reverse of string Programming is gnimmargorP
```

We can process an array of strings using dynamic memory allocation and array of pointers; we will study it in detail in the chapter on functions.

: SUMMARY :

Pointer is a variable that stores the address of another variable. It enhances the capability of the language and reduces the size of the program. It allows working with dynamic memory allocation. We use with pointer an operator * called as dereferencing operator or indirection operator.

Some of the **valid operations on pointer** are: increment, decrement, add integer number, subtract integer number and subtract one pointer from other if pointing to same array.

Invalid operations on pointer are: add two pointers, multiplication of pointer with any number and division of pointer with any number.

Pointer to pointer is a variable which store the address of another pointer variable. For example, int **ptrtoptr;

Pointers and array have close relation. Every program written using array can be written using a pointer. The name of array variable works as the address of the first element of an array. The name of an array is a constant pointer; we cannot assign any other variable address to an array name.

: MCQs :

1. Which operation is invalid for pointers?
   (a) Addition of two pointers (b) Increment
   (c) Decrement (d) Adding integer value to pointer variable
2. If ptr is a pointer to int, having value ptr =100. After ptr++, what is the value of ptr ?
   (a) 100 (b) 101 (c) 102 (d) 104
3. Which operator retrieves lvalue of a variable?
   (a) * (b) & (c) -> (d) None of above
4. For declaration of pointer variable, we use operator
   (a) * (b) -> (c) . (d) &
5. If array is declared as int a[5], then *(a+3) refers to
   (a) a[1] (b) a[2] (c) a[3] (d) None of above
6. The declaration int *a[5] means
   (a) Array of 5 integers (b) pointer to integers
   (c) Both a and b (d) Array of 5 pointers to integer
7. Pointers can be used to create complex data structures like
   (a) Stack (b) Queue (c) Linked list and trees (d) All of above
8. A pointer value refers to
   (a) A float value (b) An integer constant
   (c) Any valid address in memory (d) None of above
9. A pointer is
   (a) a keyword used to create variables
   (b) a variable that stores the address of an instruction.
   (c) a variable that stores address of other variable.
   (d) All of above
10. Prior to using pointer variable:
    (a) It should be declared (b) It should be initialized
    (c) It should be both declared & initialized (d) None of these
11. Comment on this pointer declaration:
    int *X, Y;
    (a) X is pointer to integer, Y is not. (b) X & Y both are pointer to integer.
    (c) X is pointer to integer, Y may or may not be (d) X & Y both are not pointer to integer.
12. Which one of following is valid to access the address of variable A using pointer P?
    (a) P = *A (b) P = &A (c) P = &&A (d) P = A
13. If M and N are pointers then which one of following is valid?
    (a) M + 3 (b) M / N (c) N / 3 (d) M * N
14. Pointers reduce length and complexity of programs.
    (a) True (b) False (c) Can't say
15. P1 and P2 are pointer variables. Which is an illegal pointer expression?
    (a) P1++ (b) P2 + P1 (c) P1 – P2 (d) P2 > P1
16. Local variables in C are stored in memory which is known as:
    (a) Heap (b) Permanent storage area
    (c) ROM (d) Stack
17. main( )

```
        {   int C=99, *ptr;
            ptr = &C;
printf("%d %u %u", *ptr, ptr, &C);  }
```

    (a) 99 3500 3502 (b) 66 3500 3502 (c) 99 3502 3502 (d) 66 3502 3502

18.
```
main( )
{
    int x[ ]={22, 33};
    int *p=x;
    printf("%d    %d ", ++*p,*p);
}
```

(a) 33 22 (b) 22 33 (c) 22 23 (d) 23 22

: ANSWERS :

1.	(a)	2.	(c)	3.	(b)	4.	(a)	5.	(c)	6.	(d)	7.	(d)
8.	(c)	9.	(c)	10.	(c)	11.	(a)	12.	(b)	13.	(a)	14.	(c)
15.	(d)	16.	(a)	17.	(c)	18.	(d)						

: EXERCISE :

1. What is pointer? Explain how pointers are declared an initialized.
2. What are the advantages and disadvantages of pointers?
3. Find errors if any in following code:

```
int *a, *b;
int p;
char *c;
p = a +b;
p= a-b;
p = b-a;
a = b-2;
p = a-c;
p = a *b;
```

4. Write a program in 'C' to add two numbers using pointers.
5. In pointer arithmetic, which operations are invalid?
6. Explain array of pointer with example.
7. How an array of pointers can be used to store a collection of strings?
8. What is the relationship between an array name and pointer? Explain with suitable example.

: Answers to selected exercises :

1. What is pointer? Explain how pointers are declared an initialized.

Ans:

Pointer is a variable that stores the address of another variable within the memory. So, we can say that pointer is a special type of variable which store the address of another variable as its value.

Syntax for pointer declaration is:

```
data_type *varname;
```

where data_type means pointer to which type of data, varname is the name of a variable, while * symbol before the name indicates that the variable is a pointer type variable.

For example,

```
int *ptr;
```

This declaration says that ptr variable can store the address of any integer variable.

The pointer must be initialized with proper address; otherwise it will point to garbage value.

For example,

```
int n = 10;
int *iptr = &n;
```

The above code declares variable n with value 10 and pointer variable iptr which is also initialized with the address of variable n.

Now, the statement, *iptr = 20;
will make the value of n = 20.

6. Explain array of pointer with example.

Ans:

Pointer is a variable that stores the address of another variable within the memory. So, we can say that pointer is special type of variable which store the address of another variable as its value.

For example,

```
int a;
int *ptr;
ptr = &a;
```

The above code assign the address of variable 'a' to variable ptr. We can refer to variable 'a' through pointer by using *ptr in the program.

Array of pointers is a collection of pointers of same type known by single name.

Syntax is:

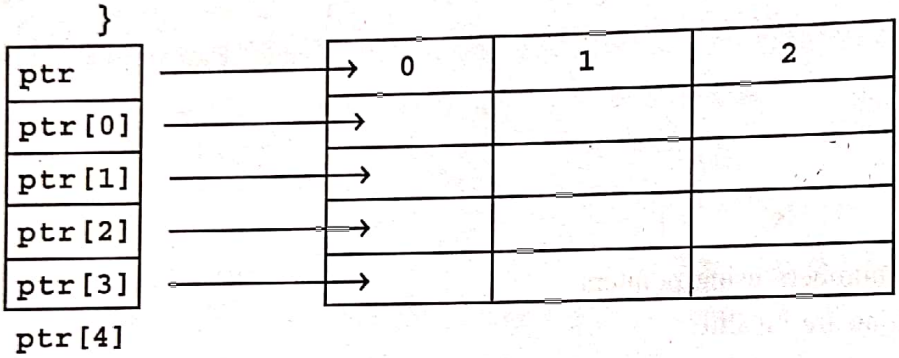
```
data_type *name[size];
```

Example :

```
int *ptr[5];      declares an array of integer pointers of size 5
int mat[5][3];    declares a matrix of 5 rows and 3 columns
```

Now, the array of pointers ptr can be used to point to different rows of matrix as follow

```
for(i=0;i<5;i++)
{
    ptr[i] = &mat[i][0];
}
```



As stated above, by using dynamic memory allocation, we do not require to declare two-dimensional array, it can created dynamically using array of pointers.

: SHORT QUESTIONS :

1. What is pointer?

⇒ Pointer is a variable that stores address of another variable. It is mechanism of indirectly referring another variab

2. Which arithmetic operations on pointer is not valid?

⇒ Addition of pointers, multiplication of pointer with any number, division of pointer with any number is inva arithmetic operations.

3. Can we perform pointer arithmetic on array name?

⇒ Array name is a constant pointer. We cannot change constant pointer value. So, pointer arithmetic can be performed on array name.

4. In 'C', why is the void pointer useful? When would you use it?

⇒ The void pointer is useful because it is a generic pointer that any pointer can be cast into and back ag without loss of information.

5. Are the expressions arr and &arr same for an array of integers?

⇒ Yes for array of integers they are same.

❖

CHAPTER

11

Structures and Unions

11.1 INTRODUCTION

11.2 WHAT IS STRUCTURE ?

11.3 ACCESSING STRUCTURE MEMBERS

11.4 STRUCTURE INITIALIZATION

11.5 ARRAY OF STRUCTURES

11.6 NESTED STRUCTURES

11.7 STRUCTURE AND FUNCTIONS

11.8 POINTERS AND STRUCTURES

11.9 ARRAY OF POINTERS TO STRUCTURES

11.10 UNION

11.11 SOLVED PROGRAMMING EXAMPLES

- SUMMARY
- MCQs
- EXERCISES AND ANSWERS TO SELECTED EXERCISES
- SHORT QUESTIONS

11.1 INTRODUCTION :

In previous chapters, we have used the basic data type as well as arrays. The basic data types and variables can store only one type of data. In real life, we find situations where we need to store data items that are logically related but contain different type of information which require different data types. For example, Student record is a collection of data items such as roll number, name, address, city, sex, age etc. which are dissimilar data items. Here, roll number is an integer type, name, address, city is array of characters, sex is a character type variable, and age is an integer. Similarly, Book is a collection of data items such as title, author name, publisher name, number of pages, price, year of publication etc. Here again, title, author name, publisher name are array of characters, while number of pages and year of publication is integer, while price can be float number. To handle the data of dissimilar type, which are logically related to each other, 'C' language provides a new data type called as structure.

11.2 WHAT IS STRUCTURE ?

Structure is a collection of logically related data items of different data types grouped together and known by a single name. So, we can say that structure is a group of variables of different types known by a single name. Structure is similar to records in DBMS. As record consists of fields, the data items of a structure are called as its members or fields.

Syntax for defining the structure is :

```
struct  struct_name
{
datatype  var1;
datatype  var2;
.
.
};
```

Here, the keyword struct is used to define a structure template. struct_name is an optional tag which provides a name of a structure. It is a user defined name. The members of the structure are written in Curly {} braces. The structure definition is terminated by semicolon (;) symbol. The struct_name can be used to declare variables of its type. So, effectively we are creating a user defined data type named as struct_name.

Example : For the Student, we can write the structure as shown below:

```
struct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    int  age;
};
```

Above definition declares a structure named as student, having 6 member variables. rollno and age are integers, while name, address, city are arrays, and sex is a character type variable. Writing only the declaration does not declare any structure variables. If we want to store data in the structure using above prototype, we need to declare variables of the structure type. The structure variables can be declared in different ways.

- Variable declaration with the template itself.
- Variable declaration any where in the program.
- Array of structures

Following example shows s1 and s2 as struct variables of student type with the template itself.

```
struct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    int  age;
} s1,s2;
```

Note that s1 and s2 are written immediately after } symbol and terminated by semicolon symbol. Following example shows s1 and s2 as struct variables of student type. The declaration can be anywhere in the program.

```
struct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    int  age;
};
.
.
.
struct  student  s1,s2;
```

Note that first the template is defined, and the variables are declared by the statement

```
struct  student  s1,s2;
```

This statement can appear at any place where we can declare variables of other types also.

3 ACCESSING STRUCTURE MEMBERS :

We can access the individual members of a structure using the dot (.) operator. The dot (.) operator is called as structure member operator because it connects the structure variable and its member variable.

The syntax is :

```
struct_var.struct_member;
```

where, struct_var is a variable of structure type, while struct_member is a name of a member variable of structure.

For example, for the struct student with s1 and s2 as variables,

```
s1.rollno  =5;
```

```
s1.age  =  20;
s2.rollno  =7;
s2.age  =  21;
s1.sex  =  'm';
s2.sex  =  'f';
```

Above code assigns student s1 rollno = 5, age = 20 and sex = 'm' while for student s2 rollno = 7, age =21 and sex ='f'.

Program :

/* Write a program to input and print following details of student using structure - roll number, name, address, city, sex and age */

```
#include  <stdio.h>
#include  <conio.h>
void  main()
{
    struct  student        /*  structure  definition*/
    {
    int  rollno;
    char  name[20];
    char  address[30];
    char  city[20];
    char  sex;
    int  age;
    };
    struct  student  s1;   /*  structure  variable  */
    clrscr();
    printf("Give  roll  number\n");
    scanf("%d",&s1.rollno);      /* get roll number */
    fflush(stdin);               /*  clear  buffer  */
    printf("Give  name\n");
    gets(s1.name);               /*  get  name  */
    printf("Give  address\n");
    gets(s1.address);            /*  get  address  */
    printf("Give  city\n");
    gets(s1.city);               /*  get  city  */
    printf("Give  sex\n");
    scanf("%c",&s1.sex);         /*  get  sex  */
    printf("Give  age\n");
    scanf("%d",&s1.age);         /*  get  age  */
    printf("Roll  Number  =  %d,  Name  =  %s,
```

```
    Address  =  %s\n",s1.rollno,  s1.name,  s1.address);
    printf("City  =  %s,  Sex  =  %c,  Age  =  %d\n",  s1.city,s1.sex,s1.age);
}
```

Output :

```
Give  roll  number
1
Give  name
vansh
Give  address
university  road
Give  city
patan
Give  sex
m
Give  age
3
Roll Number  =  1,  Name  =  vansh,  Address  =  university  road
City  =  patan,  Sex  =  m,  Age  =  3
```

STRUCTURE INITIALIZATION :

As applicable to basic data types, we can also initialize the structure variable at the time of creation of variables. For our student structure example, it can be written as:

```
struct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    int  age;
};
.
.
struct  student  s1  =  {  1,  "Vansh",  "University
                           Road","Patan",'m',3};
struct  student  s2  =  {  2,  "Saniya",  "Krishna  Nagar",
                           "Ahmedabad",'f',4);
```

In above code, the two lines

```
struct  student  s1  =  {  1,  "Vansh",  "University
                           Road","Patan",'m',3};
struct  student  s2  =  {  2,  "Saniya",  "Krishna  Nagar",
                           "Ahmedabad",'f',4);
```

initializes the structure variables s1 and s2 with the values written in { } brackets.

If all the values in initialization are not supplied, then rest of the member variables will be initialized to zero for numbers and NULL to strings. For example,

```
student s1 = { 1, "Vansh"};
```

will initialize only the rollno and name, while other variables are initialized to zero or NULL whichever is applicable.

Program :

```
/* Write a program to demonstrate initialization of structure members. */
#include <stdio.h>
#include <conio.h>
struct book
{
char title[25];
char author[20];
char publisher[20];
int pages;
float price;
};
void main()
{
struct book bk = { "C Programming", "XYZ","ABC Ltd",441,275};
clrscr();
printf("Title = %s, Author Name = %s\n", bk.title, bk.author);
printf("Publisher = %s\n",bk.publisher);
printf("Pages = %d, Price = %6.1f\n",bk.pages,bk.price);
}
```

Output :

```
Title = C Programming, Author Name = XYZ
Publisher = ABC Ltd
Pages = 441, Price = 275.0
```

11.5 ARRAY OF STRUCTURES :

As we have an array of basic data types, same way we can have an array variable of structure. For our student example, suppose we want to store details of 50 students, in stead of declaring 50 different variables of student type, we can declare an array of size 50 to store details of 50 students.

Following example shows how an array of structures can be declared.

```
struct student
{
    int rollno;
    char name[25];
    char address[30];
    char city[15];
```

```
    char  sex;
    int  age;
};
.
.
.
struct  student  s[50];
```

Above example, defines the student structure, and the line

```
    struct  student  s[50];
```

declares an array s[50] of student type structure. We can access individual members of an array by using the subscript variable like s[0], s[1], s[2] etc.

Program :

/* Write a program to calculate net salary of N employees using structure.
Total = Basic + DA + HRA +TA, Net = Total - Itax
Itax = 5 % of Total ,DA = 50 % of Basic, HRA = 15 % of Basic TA = 400 fix */

```
#include  <stdio.h>
#include  <conio.h>
struct  employee
{
    char  name[20];
    double  basic;
    double  da;
    double  hra;
    double  ta;
    double  total;
    double  net;
};
void  main()
{
    struct  employee  emp[50];
    int  i,n;
    double  temp;
    clrscr();
    Printf("How  many  employees  ?\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
        {
        printf("Give  name  :");
        scanf("%s",emp[i].name);
```

```
        printf("\nGive  basic  :");
        scanf("%lf",&temp);
        emp[i].basic  =  temp;
        emp[i].da  =  emp[i].basic  *0.5;
        emp[i].hra  =  emp[i].basic  *0.15;
        emp[i].ta  =  400;
        emp[i].total = emp[i].basic + emp[i].da +  emp[i].hra +emp[i].ta;
        emp[i].net  =  emp[i].total  -  emp[i].total  *  0.05;
        }
    printf("Name  Basic  Da  Hra  Ta  Total  Net\n");
    for(i=0;i<n;i++)
    printf("%-10s  %6.1f  %6.1f  %6.1f  %6.1f  %7.1f%7.1f\n",
    emp[i].name,emp[i].basic,                                  emp[i].da,
    emp[i].hra,emp[i].ta,emp[i].total,emp[i].net);
}
```

Output :

```
How  many  employees  ?
2
Give  name  :sms
Give  basic  :12000
Give  name  :bvb
Give  basic  :18000
Name  Basic   Da      Hra     Ta     Total    Net
sms   12000.0 6000.0  1800.0  400.0  20200.0  19190.0
bvb   18000.0 9000.0  2700.0  400.0  30100.0  28595.0
```

Program :

/* Write a program to calculate average marks of N students for 3 different subjects using structures. */

```
#include  <stdio.h>
#include  <conio.h>
struct  stud_exam
{
    char  name[20];
    int  m1;
    int  m2;
    int  m3;
    float  avg;
};
void  main()
{
```

```
    struct  stud_exam  st[50];
    int  i,n;
    clrscr();
    printf("How  many  students  ?\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
         {
         printf("Give  name  :");
         scanf("%s",st[i].name);
       printf("\nGive marks of subject 1, 2 and 3 :");
                          scanf("%d%d%d",&st[i].m1,&st[i].m2,&st[i].m3);
       st[i].avg = (st[i].m1 + st[i].m2 + st[i].m3)/3;
       }
    printf("Name  Subject1  Subject2  Subject3  Average\n");
    for(i=0;i<n;i++)
    printf("%-10s    %7d    %7d    %7d    %7.2f\n",    st[i].name,
    st[i].m1,st[i].m2,st[i].m3,st[i].avg);
}
```

Output :

```
How  many  students  ?
2
Give  name  :shrey
Give  marks  of  subject  1,  2  and  3  :69  56  60
Give  name  :saniya
Give  marks  of  subject  1,  2  and  3  :44  55  66
Name      Subject1     Subject2     Subject3     Average
shrey     69           56           60           61.00
saniya    44           55           66           55.00
```

NESTED STRUCTURES :

When the member variable of a structure itself is a structure, it is called as nesting of structures. The structure which is nested inside other structure must be declared as a structure before the structure in which it is nested.

Suppose we want to include the birth date in stead of age in student structure, we can first define a structure for date as shown.

```
struct  date
{
int  day;
int  month;
int  year;
```

```
};
stuct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    struct  date  bdate;
} ;
```

Here, in the definition of student structure, last member bdate itself is a structure of type date. The date structure is declared as having three parts namely: day, month and year.

If the date of birth is 1/1/1995, it can be written as

```
struct  student  s1;
s1.bdate.year  =  1995;
s1.bdate.month=1;
s1.bdate.day  =1;
```

In above declaration of date and student, we can declare variables of date as well as student type.

But if the above definition is written as

```
    stuct  student
    {
        int  rollno;
        char  name[25];
        char  address[30];
        char  city[15];
        char  sex;
        struct  date
            {
            int  day;
        int  month;
        int  year;
        }  bdate;
} ;
```

Then we can create variables of student type only. We can not create variables of date type. So, effectively date structure is not global data type, but it becomes local to student data type.

If we have declared the variable s1 as

struct student s1;

then we can access day by statement :

```
    s1.bdate.day;
```

Program :

```
/* Write a program using nested structure to accept and print details of an employee. */
#include  <stdio.h>
#include  <conio.h>
struct  address
{
char  add1[30];
char  add2[30];
char  area[30];
char  city[20];
};
struct  employee
{
char  name[20];
struct  address  a;
int  sal;
};
main()
{
struct  employee  e;
clrscr();
printf("Give  name  :");
scanf("%s",e.name);
printf("Give  address  line1,  line2,  area  and  city  name\n");
gets(e.a.add1);
gets(e.a.add2);
gets(e.a.area);
gets(e.a.city);
printf("Give  salary\n");
scanf("%d",&e.sal);
printf("Details  of  employee:\n");
printf("Name  :  %s\n",e.name);
Printf("Address  :");
printf("%s",e.a.add1);
Printf("%s",e.a.add2);
Printf("%s",e.a.area);
Printf("%s",e.a.city);
Printf("\nSalary  =  %d\n",e.sal);
}
```

Output :

```
Give name :sanjay
Give address line1, line2, area and city name
a-2 ashish society
near railway station
unjha
Give salary
20000
Details of employee:
Name : sanjay
Address : a-2 ashish society,near railway station,unjha
Salary = 20000
```

11.7 STRUCTURE AND FUNCTIONS :

As we can pass a variable of basic data type and an array to function as an argument, same way we can pass the structure member as well as whole structure to the function.

When we want to process only the structure member, we pass only its value using structure variable and dot (.) operator and the name of member variable, but when we want to process the whole structure, we pass a structure variable as an argument.

For example,

If fun1() is to be called then calling fun1() as ,

fun1(s1.rollno); is an example of passing the structure member rollno only.

While the call,

fun1(s1); is an example of passing whole structure as an argument.

Program :

/* Write a program using function and structure to accept Item information such as name, quantity, unit price, and cost. Then print the details of each item. */

```
#include <stdio.h>
#include <conio.h>
struct item
{
char name[25];
int qty;
float price;
float cost;
};
float calculate(int q, float p)
    {
    return q*p;
    }
```

```
void print_items(struct item i)
   {
   printf("\n%-12s %5d %6.1f
         %6.1f\n",i.name,i.qty,i.price,i.cost);

}
main()
{
struct item it[20];
float temp;
int i,n;
clrscr();
printf("How many items ?" );
scanf(„%d",&n);
for(i=0;i<n;i++)
{
flushall();
printf("Give name of item:");
gets(it[i].name);
//flushall();
printf("Give quantity :");
scanf("%d",&it[i].qty);
printf("Give unit price:");
scanf("%f",&temp);
it[i].price = temp;
it[i].cost = calculate(it[i].qty,it[i].price);
}
printf("\nName        Quantity    Price    Cost\n");
for(i=0;i<n;i++)
    print_items(it[i]);
}
```

Output :

```
How many items ?3
Give name of item:shampoo
Give quantity :30
Give unit price:45
```

```
Give  name  of  item:soap
Give  quantity  :35
Give  unit  price:13
Give  name  of  item:hair  oil
Give  quantity  :20
Give  unit  price:30
Name          Quantity     Price       Cost
shampoo          30         45.0      1350.0
soap             35         13.0       455.0
hair  oil        20         30.0       600.0
```

11.8 POINTERS AND STRUCTURES :

For representing complex data structures, we can use structures and pointers together. Pointer can be a member of structure, or a pointer to a structure, or an array of pointers to structure.

We can declare a pointer to structure as we do for pointers to basic data types.

```
struct  student
{
int  rollno;
char  name[20];
char  address[30];
char  city[20];
char  sex;
int  age;
} s1,*sptr;       /*Here,  s1  is  structure  variable,
                  while  sptr  is  a  pointer  to  structure  */
.
.
sptr  =  &s1;
/*  assign  address  of  struct  variable  s1  to  sptr  */
```

Now, sptr points to s1 and can be used to access member variables of struct student. To access member variables using pointer to structure, we have to use an arrow -> operator. The syntax is :

```
struct_pointer  ->  member_var;
```

the struct_pointer is a pointer to structure, and member_var is the name of member variable. For our example, we can use like this:

```
s1  ->  rollno  =  20;
```

We can also use . operator to access member variable with pointer to structure like:

```
(*s1).rollno  =  20;
```

Both above statements do the same thing. But if we use dot (.) operator, then parenthesis is necessary because the dot (.) operator has higher precedence than * operator.

We can also use pointer to structure to create the structure dynamically using dynamic memory allocation. For example the statement,

```
sptr  =  (struct  student  *)  malloc  (sizeof  (struct  student));
```

allocates a one block of storage for student structure and stores the pointer to that in sptr variable. Similarly, we can write the statement,

```
sptr  =  (struct  student  *)  malloc  (10  *  sizeof  (struct  student));
```

to allocate memory for array of 10 structures of student type and stores the address of first block to variable sptr.

We can use sptr in a loop using -> operator to access individual members of structure. When we write statement like

```
sptr++;
```

then it points to the next structure element.

For example, following for loop will get 10 values from keyboard and store it in structure array created dynamically.

```
for (i =0;i<10;i++)
    {
    printf("Give rollno and age of student\n");
          /* only rollno and age values shown */
    scanf("%d%d",&sptr->rollno, &sptr->age);
    sptr++;
    }
sptr = sptr -10;
/* point sptr back to first element in an array*/
```

Program :

/* Write a program to print the number with its cube using pointer to structure */

```
#include <stdio.h>
#include <conio.h>
struct cube
{
    int num;
    int c;
};
void main()
{
    struct cube *ptr;
    int i,n;
    clrscr();
    printf("How many numbers ?\n");
    scanf("%d",&n);
    ptr = (struct cube *)malloc (n * sizeof(struct cube));
    for(i=0;i<n;i++)
        {
        printf("Give number :");
        scanf("%d", &(ptr->num));
        ptr ->c = ptr -> num * ptr -> num *ptr -> num;
        ptr++;
        }
```

```
    ptr = ptr -n;
    printf("Number    Cube\n");
    for(i=0;i<n;i++)
        {
        printf("%d\t  %d\n",ptr->num,ptr->c);
        ptr++;
        }
}
```

Output :

```
How many numbers ?
2
Give number :1
Give number :2
 Number     Cube
   1          1
   2          8
```

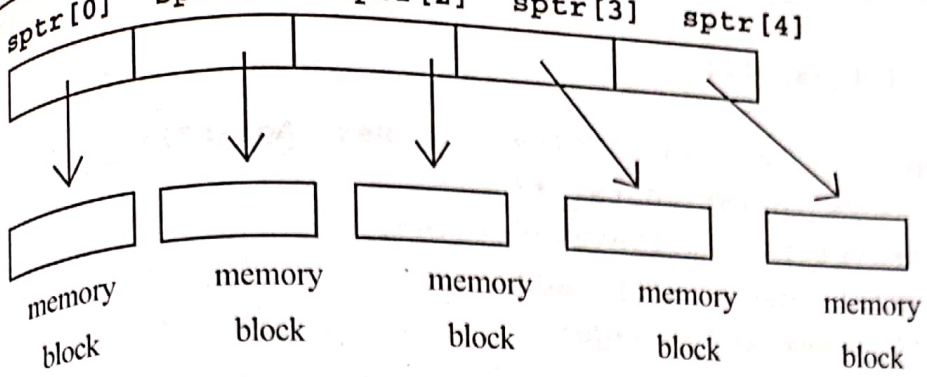
11.9 ARRAY OF POINTERS TO STRUCTURES :

If we declare an array of structures, then it occupies more memory. For example, if we take student structure defined earlier, we can find out the exact number of bytes required to store one structure data using sizeof() statement. Let us assume that it is 75 bytes. So, if we have an array of size 20 then total number of bytes required is 75 * 20 = 1500 bytes. At run-time if we have only 5 array elements, then the remaining bytes in memory are wastage. This wastage of memory can be avoided by using dynamic memory allocation, where we declare an array of pointers to structure in stead of array of structures.

For the student example,

```
struct student
{
int rollno;
char name[20];
char address[30];
char city[20];
char sex;
int age;
} *sptr[20];
```

declares an array of pointers to structure of size 20. In this case only memory required to store 20 pointer variables is used in stead of 20 structure elements. We can use dynamic memory allocation method to get a block of memory and assign its address to the individual members of array of pointers. Suppose we use only five student data then it will pictorially look like this.



Program :

/* Write a program to input and print details of students using array of pointers to structure. */

```
#include<stdio.h>
#include<conio.h>
#include<conio.h>
struct  student
{
int  rollno;
char  name[20];
char  address[30];
char  city[20];
char  sex;
int  age;
}*st[20];           /*  array  of  pointers  to  structures  */
main()
{
int  i,n;
clrscr();
printf("How  many  students?  ");
scanf("%d",&n);
for(i=0;i<n;i++)
    {
    st[i]  =  (struct  student  *)  malloc
     (sizeof(struct  student));  /*  get  one  block  */
    printf("\nGive  roll  number  :  ");
            /*  get  data  and  store  in  structure  */
    scanf("%d",&(st[i]->rollno));
    printf("\nGive  student  name  :  ");
    scanf("%s",st[i]->name);
    printf("\nGive  address:  ");
    scanf("%s",st[i]->address);
    printf("\nGive  city:  ");
    scanf("%s",st[i]->city);
    printf("\nSex?  :  ");
    scanf("%c",&(st[i]->sex));
```

```
    printf("Age?  :");
    scanf("%d",&(st[i]->age));
    }
printf("Roll  No\tName    Address    City      Sex  Age\n");
for(i=0;i<n;i++)      /*  print  data  */
    printf("%d\t%-15s\t%-15s\t%-15s\t%c\t%d\n",
    st[i]->rollno,st[i]->name,st[i]->address,
    st[i]->city,st[i]->sex,st[i]->age);
getch();
}
```

Output :

```
How  many  students?  2
Give  roll  number  :  1
Give  student  name  :  paresh
Give  address:  rajmahel
Give  city:  mahesana
Sex?  :  m
Age?  :20
Give  roll  number  :  2
Give  student  name  :  sanket
Give  address:  Ullasnagar
Give  city:  mumbai
Sex?  :  m
Age?  :21
Roll  No    Name      Address      City      Sex    Age
   1        paresh    rajmahel     mahesana  m      20
   2        sanket    Ullasnagar   mumbai    m      21
```

11.10 UNION :

It is a user defined data type like structure. It also allows grouping of dissimilar data items by a single name. But, the difference is in their working. Structures occupy more memory because all the member variables occupy separate storage, while unions occupy less memory because all the member variables are not allocated separate storage but share a memory area between all the memory variables. So, unions allow storage of one member at a time.

The syntax for defining union is same as that for structure except that a keyword union is used in place of struct.

Syntax :

```
union  union_name
{
data_type  member1;
data_type  member2;
.
```

};

Example of union :

```
union  student
       {
       int  rollno;
       char  name[20];
       char  address[30];
       char  city[20];
       char  sex;
       int  age;
       }s1,s2;
```

We can get the address of a union variable by using & operator. We can also pass the union to function as an argument.

ogram :

/* Program demonstrating union */

```
#include<stdio.h>
#include<conio.h>
union  uni
{
int  i;
char  c;
float  f;
} u;
main()
{
clrscr();
u.i  =  5;
printf("Member  i  =  %d\n",u.i);
u.c  =  'a';
printf("Member  c  =  %d\n",u.c);
u.f  =  3.5;
printf("Member  f  =  %f\n",u.f);
printf("Following  printf  will  produce  wrong  value  of  i\n");
printf("because  i  member  is  not  active.  Currently  active  member  is
f\n");
printf("Member  i  =  %d\n",u.i);
getch();
}
```

Output :

```
Member i = 5
Member c = 97
Member f = 3.500000
Following printf will produce wrong value of i
because i member is not active. Currently active member is f
Member i = 0
```

Following table shows important differences between structure and union.

Structure	Union

Each member has its own separate storage space
Memory occupied is the total required to store all the members
All members can be accessed any time
Example:
struct employee
{
int empid;
char name[20];
int age;
float salary;
};
Total size required =
2 +20+2+4 = 28 bytes
	All members share the storage space
	Memory occupied is only that much which is required to store the largest member of union
	Only one member can be active at a time
	Example:
	Union employee
	{
	int empid;
	char name[20];
	int age;
	float salary;
	};
	Total memory required =
	max(2,20,2,4) = 20 bytes

The syntax and meaning for use of dot (.) operator and -> operator is same as in the case of structure.

Program :

```
/* Write a program to demonstrate the difference between structure and union*/
#include<stdio.h>
#include<conio.h>
union uni
{
int i;
char c;
float f;
} u;
struct stu
{
int i;
char c;
float f;
```

```
} s;
main()
{
int k;
clrscr();
k = sizeof(u);
printf("Total size required by union u is %d bytes\n",k);
k = sizeof(s);
printf("Total size required by structure s is %d bytes\n",k);
getch();
}
```

utput :

```
Total size required by union u is 4 bytes
Total size required by structure s is 7 bytes
```

1 SOLVED PROGRAMMING EXAMPLES :

rogram :

/* Write a program to calculate and print total salary of employees using structure. Hra = 15% of basic, Da = 50 % of basic, Ta = 5% of basic */

```
#include<stdio.h>
#include<conio.h>
struct employee
{
char name[20];
long int basic;
float da;
float hra;
float ta;
float total;
};
main()
{
struct employee *emp;
int i,n;
clrscr();
printf("How many employees\n");
scanf("%d",&n);
emp = (struct employee *) malloc(n*sizeof(struct employee));
```

```
for(i=0;i<n;i++)
    {
    printf("\nGive employee name: ");
    scanf("%s",emp->name);
    printf("\nGive basic salary: ");
    scanf("%ld",&(emp->basic));
    emp->da = emp->basic * 0.5;
    emp ->hra = emp->basic * 0.15;
    emp-> ta = emp->basic * 0.05;
    emp->total = emp->basic + emp->da +emp->hra +emp->ta;
    emp++;
    }
emp = emp-n;
printf("Name\t\Basic\tDa\tHra\tDa\tTotal\n");
for(i=0;i<n;i++)
    {
    printf("%s\t%ld\t%6.0f\t%6.0f\t%6.0f\t%7.0f\n",
    emp->name,emp->basic,emp->da,emp->hra,emp->ta,
    emp->total);
    emp++;
    }
getch();
}
```

Output :

```
How many employees
3
Give employee name: Reena
Give basic salary: 12200
Give employee name: Ramesh
Give basic salary: 12400
Give employee name: Hari
Give basic salary: 10325
Name    Basic    Da    Hra    Da    Total
Reena   12200    6100  1830   610   20740
Ramesh  12400    6200  1860   620   21080
Hari    10325    5162  1549   516   17552
```

gram :

/* Write a program to store information about books and display those books which cost less than Rs. 300*/

```
#include  <stdio.h>
#include<conio.h>
struct  book
{
   char  title[25];
   char  author[20];
   char  publisher[20];
   int  pages;
   float  price;
};
void  main()
{
   struct  book  bk[20];
   int  i,n;
   float  temp;
   clrscr();
   printf("How  many  books?\n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
       {
       flushall();
       printf("Give  title  of  book:  ");
       gets(bk[i].title);
       printf("Give  author  name:  ");
       gets(bk[i].author);
       printf("Give  publisher  name:  ");
       gets(bk[i].publisher);
       printf("Give  pages:  ");
       scanf("%d",&(bk[i].pages));
       printf("Give  price:  ");
       scanf("%f",&temp);
       bk[i].price  =  temp;
       }
   Printf("\n\nBooks  \n");
   for(i=0;i<n;i++)
       {
```

```
            puts(bk[i].title);
            puts(bk[i].author);
            puts(bk[i].publisher);
            printf("pages = %d price = %5.1f\n", bk[i].pages,bk[i].price);
        }
    printf("\n\nBooks costing < Rs. 300 \n");
    for(i=0;i<n;i++)
        {
            if(bk[i].price < 300)
            {
            puts(bk[i].title);
            puts(bk[i].author);
            puts(bk[i].publisher);
            printf("pages = %d price = %5.1f\n\n",
            bk[i].pages,bk[i].price);
            }
        }
}
```

Output :

```
How many books?
3
Give title of book: C programming
Give author name: Herbert
Give publisher name: Mcgraw Hill
Give pages: 350
Give price: 200
Give title of book: Distributed O S
Give author name: Pradeep Sinha
Give publisher name: Prentice Hall
Give pages: 600
Give price: 350
Give title of book: Computer networks
Give author name: Forouzan
Give publisher name: Tata Mcgrawhill
Give pages: 450
Give price: 250

Books
C programming
Herbert
Mcgraw Hill
pages = 350 price = 200.0
```

```
Distributed  O  S
Pradeep  Sinha
Prentice  Hall
pages  =  600  price  =  350.0
Computer  networks
Forouzan
Tata  Mcgrawhill
pages  =  450  price  =  250.0

Books  costing  <  Rs.  300
C  programming
Herbert
Mcgraw  Hill
pages  =  350  price  =  200.0
Computer  networks
Forouzan
Tata  Mcgrawhill
pages  =  450  price  =  250.0
```

: SUMMARY :

Structure is a collection of logically related data items of different data types grouped together and known by a single name. It is a user defined data type. By declaring a structure no variables are created. We need to create the variables of the structure type.

Structure **members** can be accessed using the dot (.) operator. It connects the structure variable with member variable.

We can have **nesting of structures** if the member variable itself is a structure. The nested structure must be declared before the structure in which it is nested.

We can have **pointer to structure** as we have pointer to standard data types. To access member variables using pointer to structure, we use –> **(arrow) operator.**

Union is a user defined data type similar to structure, except that it uses less memory than structure, because it shares the memory between member variables. The memory required is equal to that required for the largest data type of member variables. Only one member is active.

: MCQs :

What is not TRUE about structures?
(a) Collection of variables
(b) variables must be of same type
(c) variables can be different type
(d) All of above

Structure and union are user defined data type
(a) True
(b) False
(c) Structure only is user defined
(d) Union only is user defined

It is legal to have structure variable as a member variable of other structure i.e structure within structure.
(a) False
(b) True

4. If we have
   struct student s1;
   to access member variables with s1 we use
   (a) . (b) –> (c) & (d) *

5. If we have
   struct student *sptr;
   to access member variables with s1 we use
   (a) . (b) –> (c) & (d) *

6. Which data type allows storage of only one data item?
   (a) Union (b) struct (c) Both and b

7. In which data type all the members are active?
   (a) Union (b) struct (c) Both a and b

8. Structure can contain elements of the same data type
   (a) true (b) false

9. Which of the following operator is used to select a member of a structure variable
   (a) .(dot) (b) ,(comma) (c) : (colon) (d) ;(semicolon)

10. Which of the following is collection of a different data types?
    (a) String (b) Array (c) Structure (d) Files

11. With reference to structures, S1 –> S2 is syntactically correct if :
    (a) S1 and S2 are structure
    (b) S1 is pointer to structure and S2 is a member of structure
    (c) S1 is structure and S2 is pointer to structure
    (d) S2 is structure and S1 is pointer to structure

12. The link between member and variable in structure is established using:
    (a) Arrow operator (–>) (b) Pointer operator (*)
    (c) Pointer operator (&) (d) Dot operator (.)

13. Compile time initialization of a structure variable must have:
    (a) Tag name with keyword struct
    (b) Name of variable to be declared
    (c) Terminating semicolon
    (d) All of above

14. Which of following is possible combination with structure?
    (a) Structure within structure (b) Arrays within structure
    (c) Structure and function (d) All of above

15.
```
main()
    {   struct class
        {
            int W, H;
            }stu = {60, 170};
            printf("%d   %d", stu.H, stu.W);
        }
```
    (a) 60 60 (b) 60 170 (c) 170 60 (d) 170 170

: ANSWERS :

1. (b) 2. (a) 3. (b) 4. (a) 5. (b) 6. (a) 7. (b)
8. (a) 9. (a) 10. (c) 11. (b) 12. (d) 13. (d) 14. (d) 15. (c)

: EXERCISE :

1. What is structure? Explain its syntax with suitable example.
2. Explain how the members of structure can be accessed.
3. How the size of a structure can be determined in 'C' program?
4. Explain clearly the difference between dot (.) operator and arrow (->) operator.
5. What is nested structure? Explain with suitable example.
6. Explain with suitable example, pointer to structure and use of arrow -> operator.
7. What is an array of pointers to structure? How it helps in efficient memory usage?
8. What is union? How it differs from structure?

Answers to selected exercises:

1. What is structure? Explain its syntax with suitable example.

Ans:

Structure is a collection of logically related data items of different data types grouped together and known by a single name.

Syntax for defining the structure is:

```
struct  struct_name
{
datatype  var1;
datatype  var2;
.
.
};
```

For the Student, we can write the structure as shown below:

```
struct  student
{
int  rollno;
char  name[25];
char  address[30];
char  city[15];
char  sex;
int  age;
};
struct  student  s1,s2;
```

2. Explain how the members of structure can be accessed.

Ans:

We can ac cess the individual members of a structure using the dot (.) operator. The dot (.) operator is called as structure member operator because it connects the structure variable and its member variable.

The syntax is:

```
struct_var.struct_member;
```

where, struct_var is a variable of structure type, while struct_member is a name of a member variable of structure.

```
struct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
    char  sex;
    int  age;
};
struct  student  s1,s2;
```

For above structure example, for the struct student with s1 and s2 as variables,

```
s1.rollno  =5;
s1.age  =  20;
s2.rollno  =7;
s2.age  =  21;
s1.sex  =  'm';
s2.sex  =  'f';
```

Above code assigns student s1, rollno = 5, age = 20 and sex = 'm' while for student s2, rollno = 7, age =21 and sex ='f'.

5. What is nested structure? Explain with suitable example.

Ans:

When the member of a structure itself is a structure, it is called as nesting of structure.

The structure which is nested inside other structure must be declared as a structure before the structure in which it is nested.

Following example shows **date structure** used inside **student structure.**

```
struct  date
{
int  day;
int  month;
int  year;
};
stuct  student
{
    int  rollno;
    char  name[25];
    char  address[30];
    char  city[15];
```

```
        char  sex;
        struct  date  bdate;
    }  ;
```

Here, in the definition of student structure, last member bdate itself is a structure of type date. The date structure is declared as having three parts namely: day, month and year.

Following 'C' program explains the use of nested structure

```
#include  <stdio.h>
struct  address
{
char  add1[30];
char  add2[30];
char  area[30];
char  city[20];
};
struct  employee
{
char  name[20];
struct  address  a;
int  sal;
};
main()
{
struct  employee  e;
clrscr();
printf("Give  name  :");
scanf("%s",e.name);
printf("Give  address  line1,  line2,  area  and  city  name\n");
gets(e.a.add1);
gets(e.a.add2);
gets(e.a.area);
gets(e.a.city);
printf("Give  salary\n");
scanf("%d",&e.sal);
printf("Details  of  employee:\n");
printf("Name  :  %s\n",e.name);
printf("Address  :");
printf(",%s",e.a.add1);
printf(",%s",e.a.add2);
printf(",%s",e.a.area);
printf(",%s",e.a.city);
printf("\nSalary  =  %d\n",e.sal);
}
```

8. What is union? How it differs from structure?

Ans:

Following table shows important differences between structure and union.

: SHORT QUESTIONS :

1. Which data type is used to group different data type but logically related with each other?

⇒ structure and union data types.

2. Which operator is used to access member variables of structure?

⇒ Dot (.) operator is used to access member variables of structure.

3. Which operator is used to access member variables of structure using pointer to structure?

⇒ Using pointer to structure, we use arrow -> operator to access member variables of structure.

❖

CHAPTER

2

Flowcharts & Algorithms

2.1 INTRODUCTION

2.2 PROBLEM SOLVING TECHNIQUES

2.3 FLOWCHART

2.4 FLOWCHART EXAMPLES

2.5 ALGORITHM

2.6 PSEUDO CODE

2.7 ALGORITHMS EXAMPLES

2.1 INTRODUCTION :

In chapter 1, we learned about computer fundamentals. In this chapter we have discussed tools which can be used to develop logic of program before writing any program. Flowcharts & Algorithms are example of such tools, which are used for development of software in C & C++. These provides convenience in development of programs.

2.2 PROBLEM SOLVING TECHNIQUES :

The problem solving technique is used for solving the problem. When any problem is to be solved with help of computer programming language, computer cannot solve this problem by its own way, it require to follow following steps.

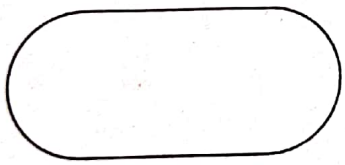
1. Problem definition
2. Analysis of problem
3. Find out input and output if required
4. Choose appropriate tools to solve problem (algorithm or flowchart)
5. Choose appropriate control structure for developing program(code design)
6. Run and debug the program (error finding)
7. Verify the output as per definition

Let us start first with flowchart, how to develop logic of program.

2.3 FLOWCHART :

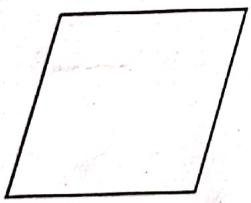
A **Flowchart** is graphical or diagrammatical representation of sequence of any problem to be solved by computer programming language. It is used before writing any program. It is also used to correct and debug a program flow after coding part is completed. Through flowchart it is easy to understand the logic and sequence of problem steps. After drawing flowchart it becomes easy to write any program. There are various standard symbols available to represent logic of problem which are...

1. Start and Stop :



Used to start and stop or end flow chart.

2. Input / Output :

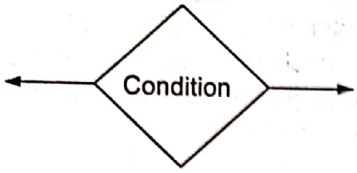


Used for input and output operation

3. Process or expression representation :

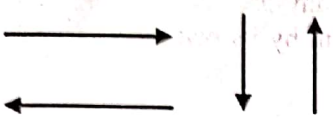
used to represent any assignment or expression

4. **Decision**



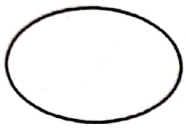
used for any decision making statements.

5. **Direction of data flow or flow lines**



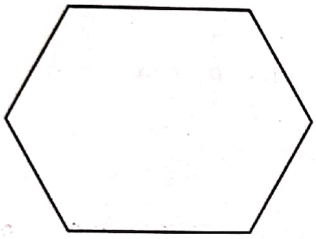
indicates flow of sequence or control.

6. **connector**



connecting flow lines from different places.

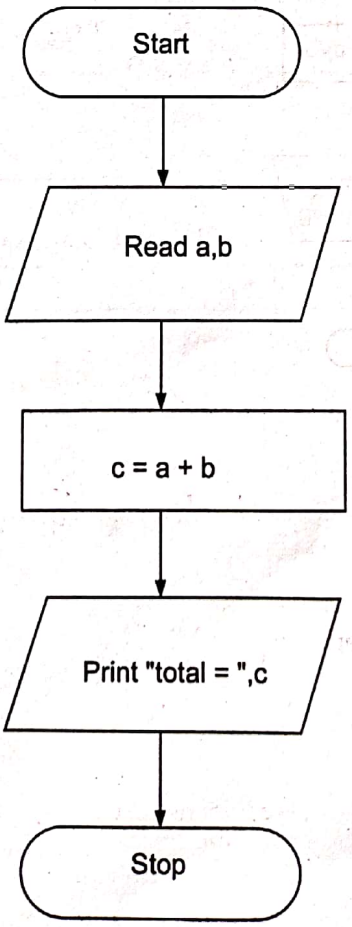
7. **loop**



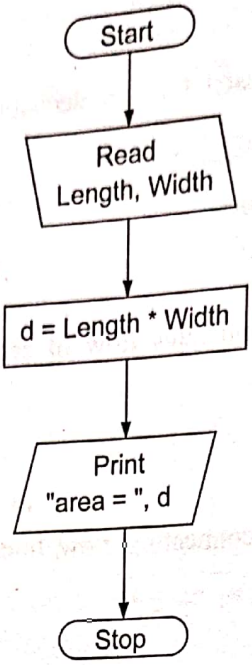
used for iteration or looping statement.

2.4 FLOWCHART EXAMPLES :

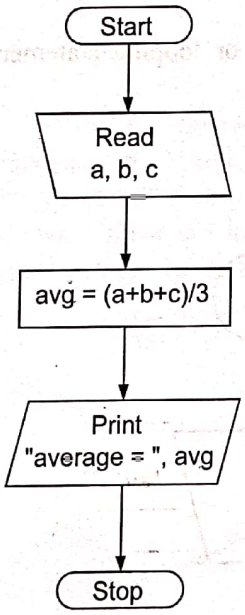
1. Draw a flowchart to find sum two numbers.



2. Draw a flow chart to find area of rectangle.

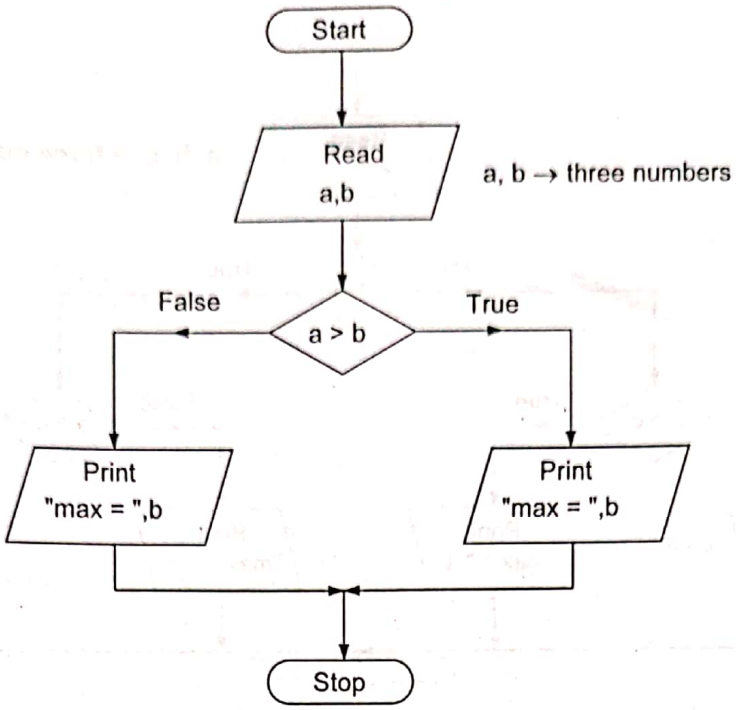


3. Draw a flowchart to find averge of three subjects marks.

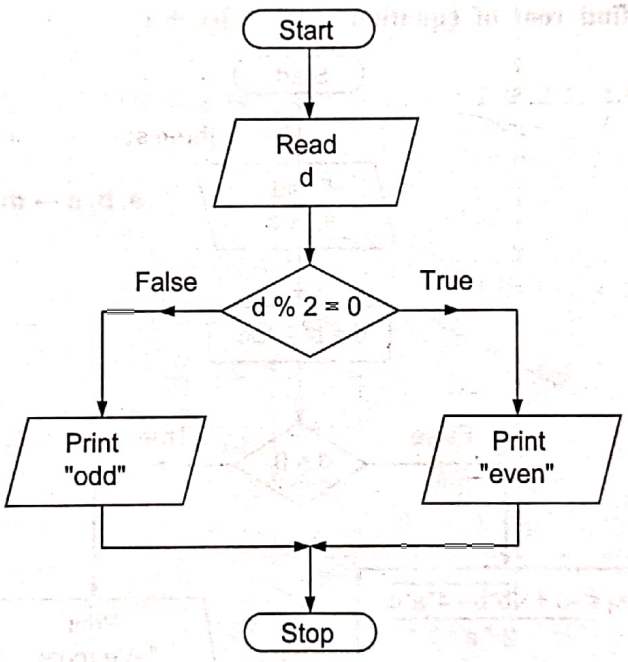


a, b, c → three subject marks

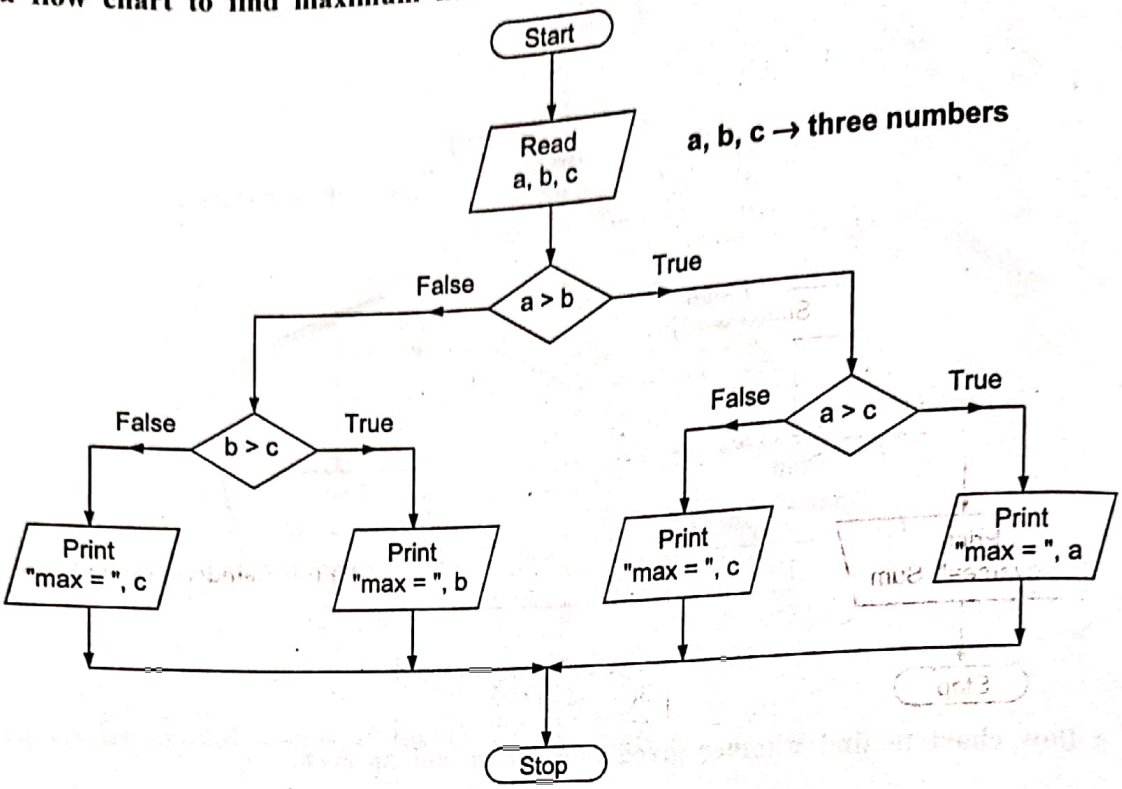
4. Draw a flowchart to find maximum number from two numbers.



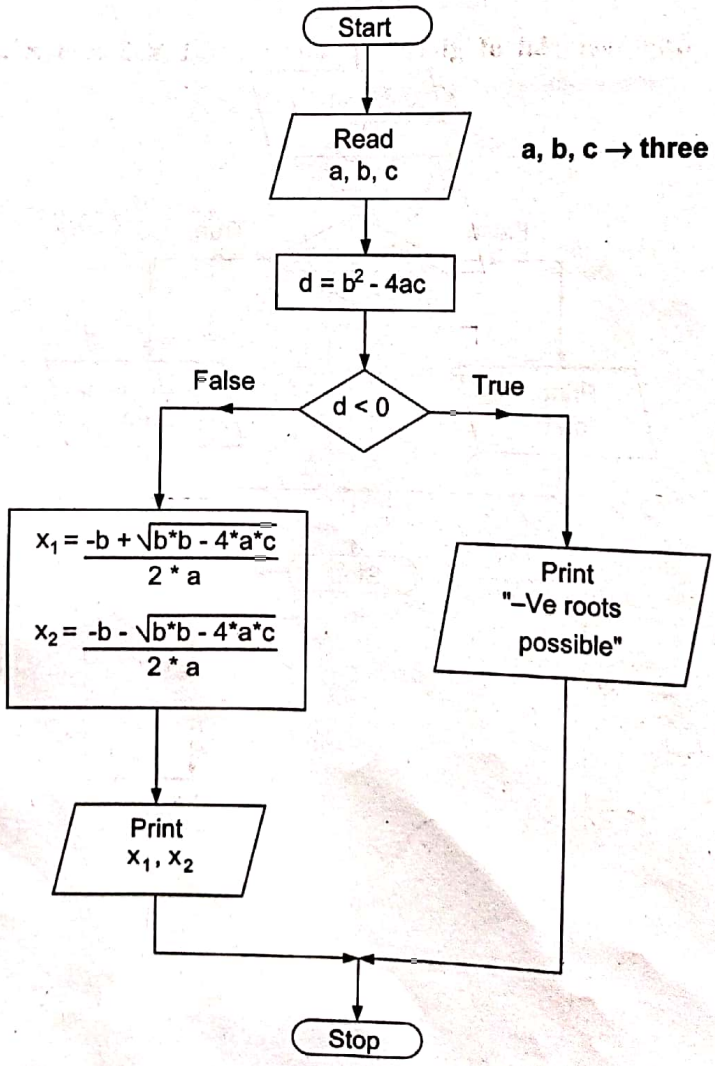
5. Draw a flow chart to find whether given number is ood or even.



6. Draw a flow chart to find maximum number from three different numbers

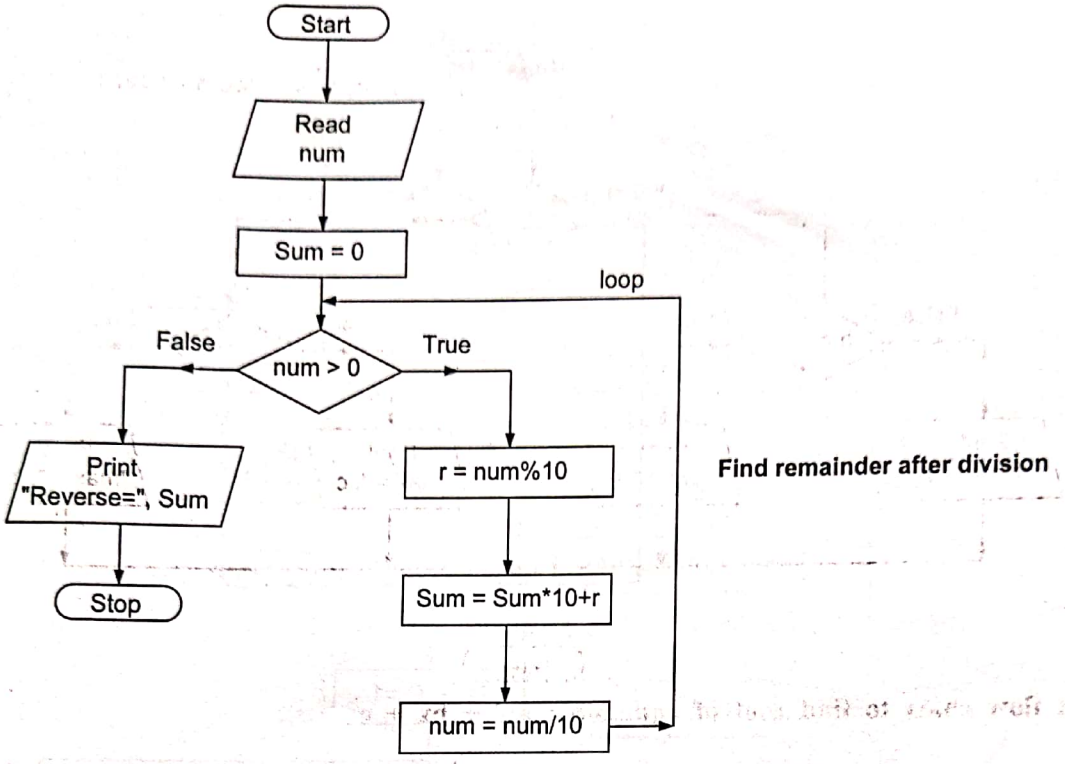


7. Draw a flow chart to find root of equation : $ax^2 + bx + c$

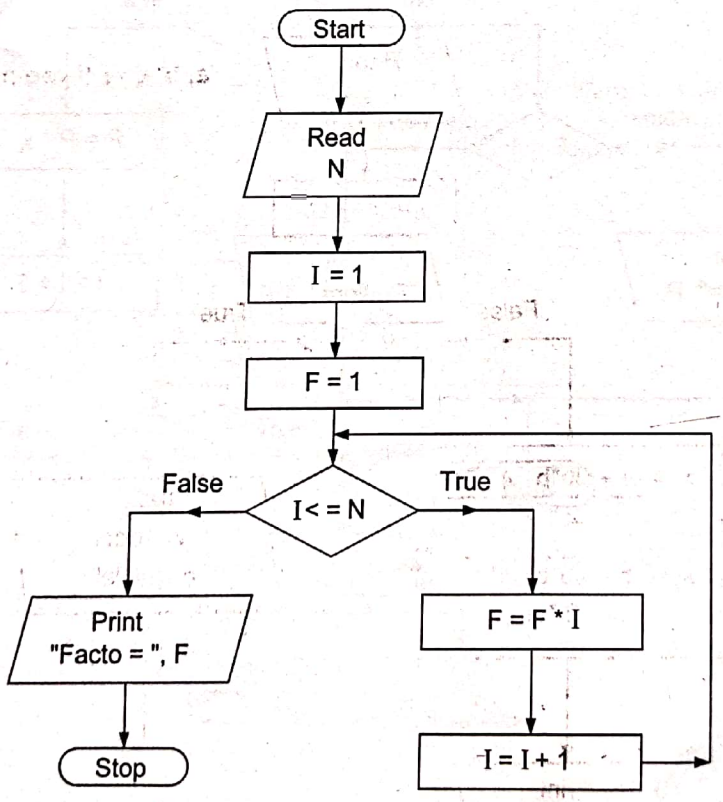


8. Draw a flow chart to reverse given number.

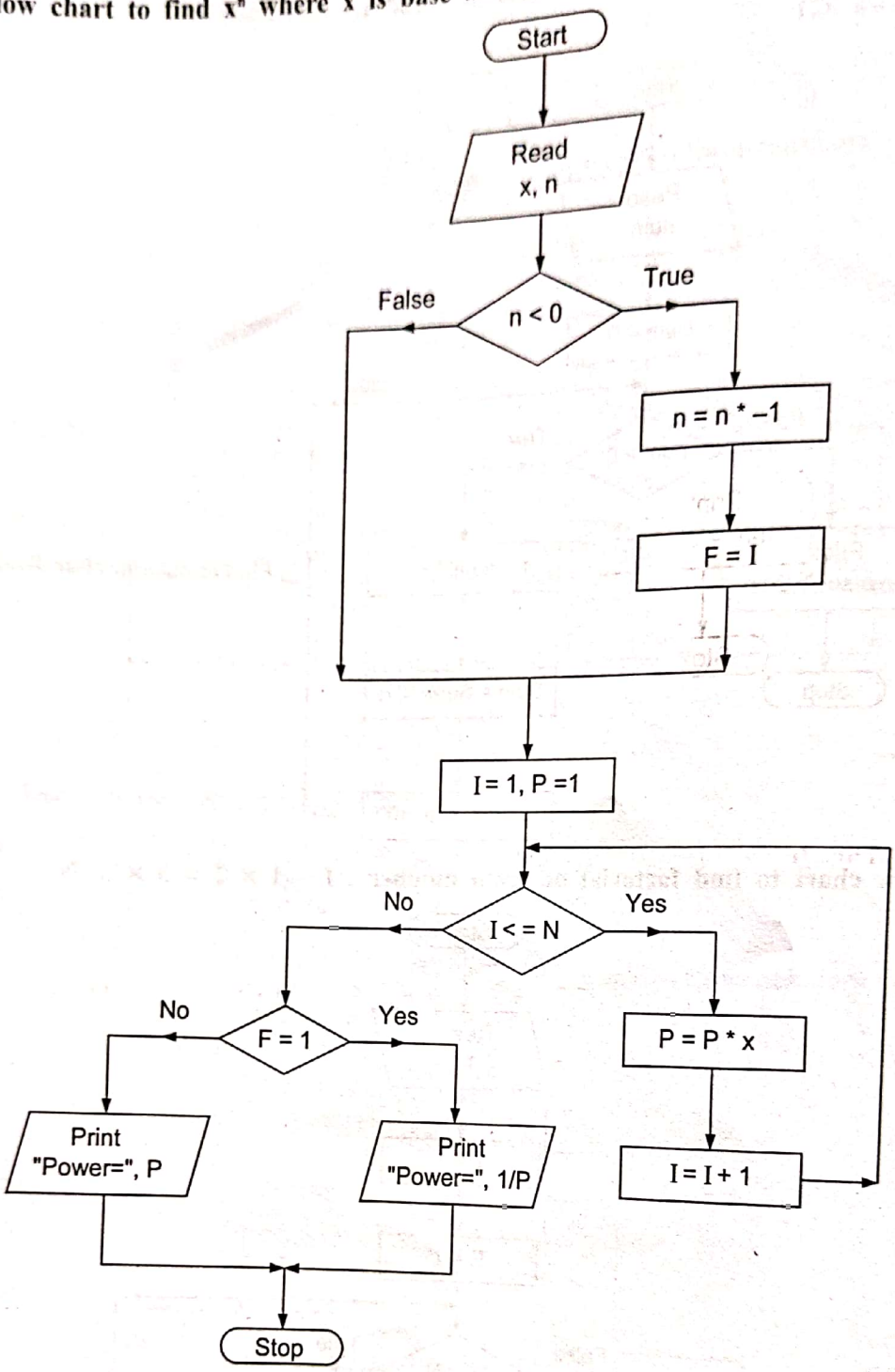
e.g. 121 → 321



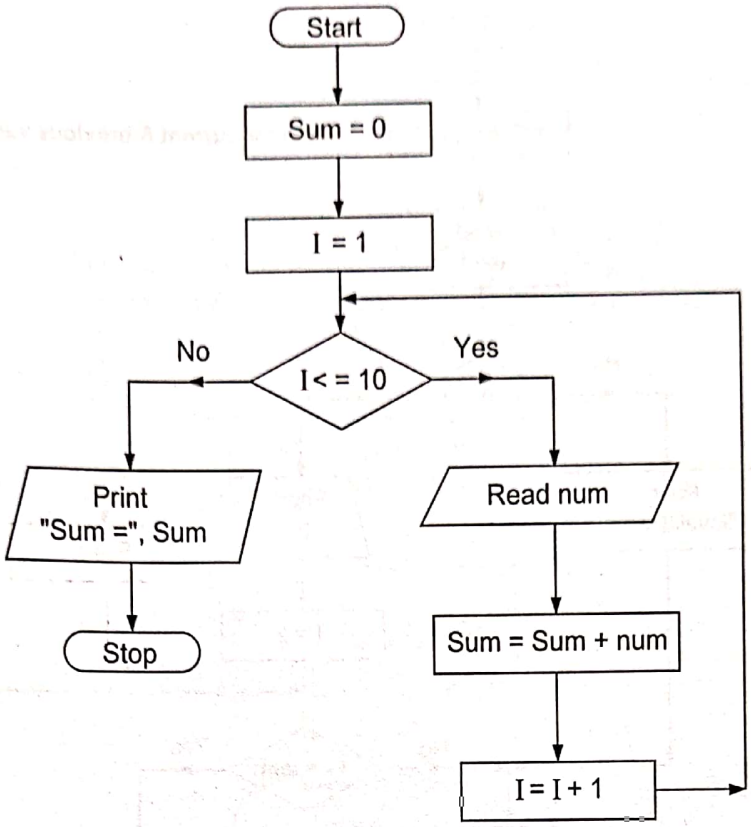
9. Draw a flow chart to find factorial of given number : f = 1 × 2 × 3 × ... N



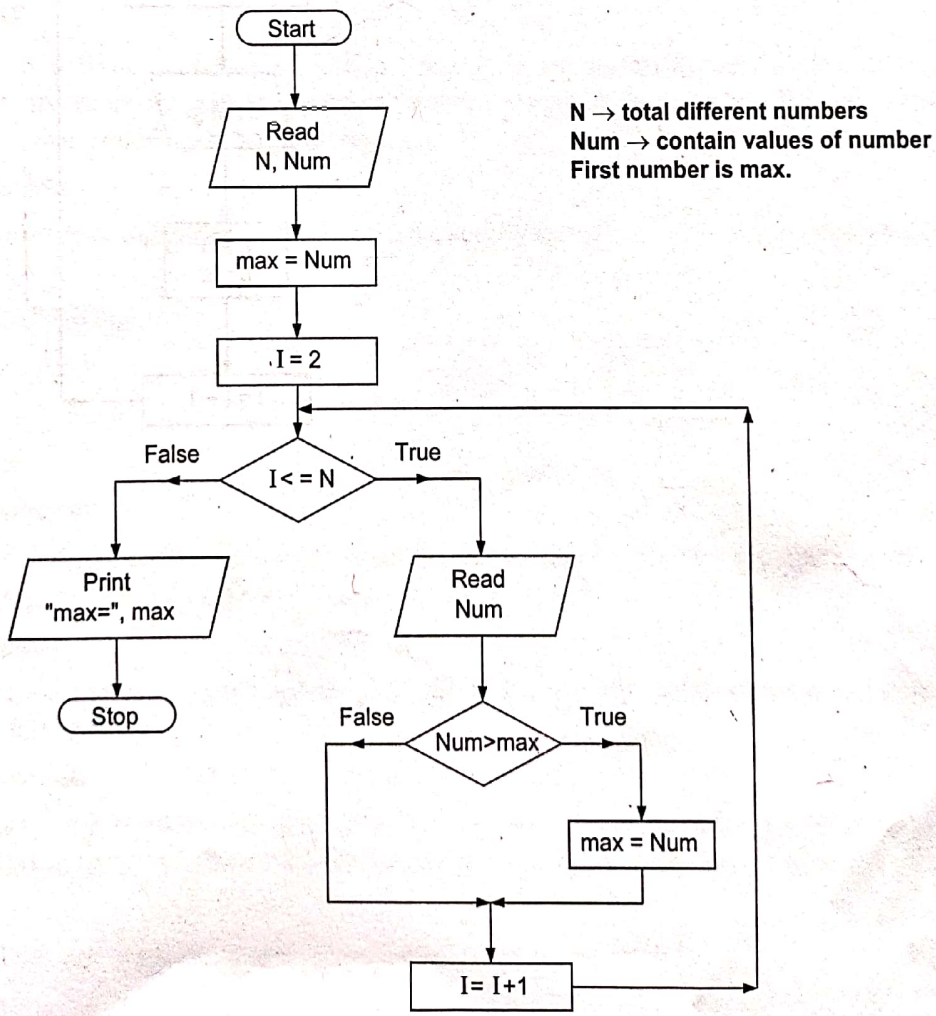
10. Draw a flow chart to find x^n where x is base and n is power.



11. Draw a flow chart to do the sum of 10 elements read from user.

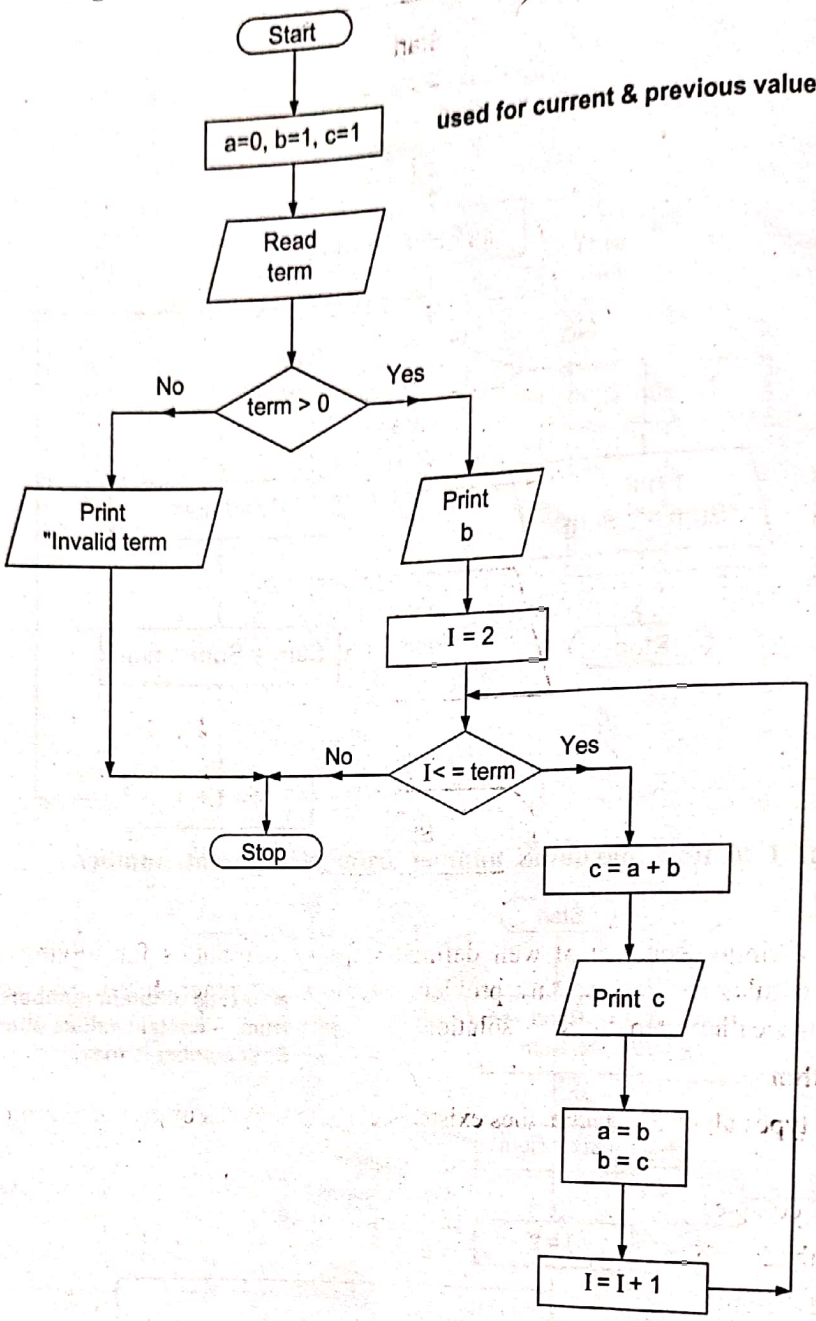


12. Draw a flow chart to find maximum number from N different number.

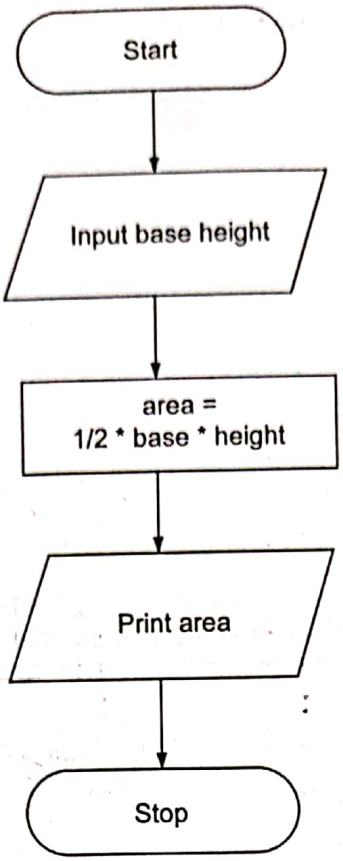


N → total different numbers
Num → contain values of number
First number is max.

13. Draw a flow chart to generate fibbonacci series upto given term.



14. Draw flow chart to find area of triangle.



2.5 ALGORITHM :

An **algorithm** is a finite sequence of well defined steps or operations for solving a problem in systematic manner. These are rules for solving any problems in proper manner. Instruction are written in the natural language. It is also called step by step solution.

Types of algorithm

There are various types algorithm techniques exists, according to types of problem appropriate types of algorithm used.

1. divide and conquer
2. greedy method
3. branch and bound
4. recursion

1. Divide and conquer :

The divide and conquer technique is used to solve complex problem easily. Complex problems are decomposed into several step, which make problem easy to solve.

2. Greedy method :

This method is used to solve optimization problem. With several possible solution one best solution is selected with help of this method.

3. Branch and bound :

When there are several statement or certain part of logic repeated this type of concept is used. A branch and bound algorithm computes a number(bound) at a node to determine whether the node is promising.

4. Recursion :

When procedure call itself is called recursion.

2.6 PSEUDO CODE :

It is language or part of language used to convert program into easy form. It allows the programmer to write logic of flowchart or algorithm in symbolic form. Hence, pseudo code is a computer instruction without referring to any specific compiler or programming language.

The pseudo code normally consists of phrases of English language to describe the logic steps of a programming module.

Pseudo code can be used for following situation.

1. storage declaration
2. assignment of any value.
3. procedure call
4. input/output
5. constant value

2.7 ALGORITHMS EXAMPLES :

1. Write an algorithm to find out sum of two numbers.

Step 1 : input two numbers : a,b

Step 2 : calculate sum = a+b

Step 3 : Print "total = ",sum

Step 4 : stop

2. Write a algorithm to find out are of square.

Step 1 : input length : len

Step 2 : calculate area=len*len

Step 3 : print " area = ", area

Step 4 : stop

3. Write an algorithm to find average of three numbers.

Step 1 : input three numbers : a,b,c

Step 2 : calculate sum=a+b+c

Step 3 : avg= sum /3

Step 4 : print "average = ",avg

Step 5 : stop

4. Write an algorithm to find whether given number is positive or negative.

Step 1 : input number : num
Step 2 : check if num < 0 then go to step 5
Step 3 : print "positive "
Step 4 : stop
Step 5 : print "negative "
Step 6 : stop

5. Write an algorithm to find out minimum number from three input numbers.

Step 1 : input three numbers : a,b,c
Step 2 : if a < b then go to next step otherwise (else) go to step 8
Step 3 : if a < c then go to next step else go to step 6
Step 4 : print " minimum = " , a
Step 5 : stop
Step 6 : print " minimum = " , c
Step 7 : stop
Step 8 : if b < c then go to next step else go to step 11
Step 9 : print "minimum = " , b
Step 10 : stop
Step 11 : print " minimum = " , c
Step 12 : stop

6. Write an algorithm to find factorial of given number

Step 1 : input number : num
Step 2 : i=1
Step 3 : f = 1
Step 4 : repeat from step 4 to step 6 until I <= num
Step 5 : f=f *i
Step 6 : i=i+1
Step 7 : print " factorial = ",f
Step 8 : stop

7. Write an algorithm to reverse given number.

Step 1 : input number : num
Step 2 : sum=0
Step 3 : repeat from step 3 to step 6 until num > 0
Step 4 : calculate r = num % 10
Step 5 : calculate sum = sum * 10 + r
Step 6 : calculate num = num / 10
Step 7 : print " reverse number = ",sum
Step 8 : stop

8. Write an algorithm to solve following series 1! + 2! + 3! + ... + n !

Step 1 : input number : num
Step 2 : f=1
Step 3 : sum=0
Step 4 : I =1
Step 5 : repeat from step 5 to step 8 until I <= num
Step 6 : f=f*I
Step 7 : sum=sum + f
Step 8 : I = I +1
Step 9 : print " sum = " << sum
Step 10 : stop

9. Write an algorithm to find whether given number is armstrong or not.

Step 1 : input number : num
Step 2 : sum=0, temp = num // store num to temp
Step 3 : repeat from step 3 to step 6 until num > 0
Step 4 : calculate r = num / 10
Step 5 : calculate sum = sum + r * r * r
Step 6 : calculate num = num % 10
Step 7 : if temp = sum then next step else go t step 9
Step 8 : print temp, "is Armstrong number"
Step 9 : stop
Step 10 : print temp, "is not Armstrong number"
Step 11 : stop

10. Write an algorithm to find maximum number from n numbers.

Step 1 : input first number : num
Step 2 : max = num
Step 3 : print "how many different numbers"
Step 4 : input n
Step 5 : I = 2
Step 6 : repeat step 6 through step 11 until I <= n
Step 7 : print " input next number "
Step 8 : input num
Step 9 : if num > max then next step else go to step 11
Step 10 : max = num
Step 11 : I = I + 1
Step 12 : print " maximum = " , max
Step 13 : stop

11. Write an algorithm to generate Fibonacci series up to given term.
1 1 2 3 5 8 13 . . . . .

Step 1 : input term
Step 2 : a=0, b=1, c=1 // a,b,c are variable used generate series with previous value
Step 3 : if term > 0 then go to next step else go to step 13
Step 4 : print b
Step 5 : I = 2 // repeat loop up to n times by using I
Step 6 : repeat from step 6 through step 12 until I <= term
Step 7 : print c
Step 8 : c = a+b
Step 9 : a=b
Step 10 : b=c
Step 11 : I = I + 1
Step 12 : stop
Step 13 : print " invalid term " // else part of if
Step 14 : stop

12. Write an algorithm to store sum of 10 different numbers

Step 1 : sum =0
Step 2 : I = 1
Step 3 : repeat from step 3 through step 6 until I <= 10 // 10 times loop
Step 4 : read num
Step 5 : sum = sum + num
Step 6 : I = I +1
Step 7 : print " total = " , sum
Step 8 : stop

13. Write an algorithm to find whether given number is prime or not

Step 1 : read num
Step 2 : I = 2
Step 3 : repeat through step 3 to step 7 until I < num // up to num – 1
Step 4 : if num % I = 0 then go to next step else go to step 7
Step 5 : print num " is not prime "
Step 6 : stop
Step 7 : I = I + 1
Step 8 : print num, " is prime "
Step 9 : stop

14. Write an algorithm to find occurrence of given number from 10 different numbers by using arra

Step 1 : dim a(10) // array of ten elements
Step 2 : I = 1
Step 3 : repeat from step 3 to step 6 until I <=10
Step 4 : read a(I)
Step 5 : I = I + 1
Step 6 : read num // num value is to be search from array
Step 7 : count = 0 // count gives occurrence of number
Step 8 : I = 1
Step 9 : repeat from step 9 through step 12 until I < = 10 // search number
Step 10 : if num == a (I) then go to next step else go to step 12
Step 11 : count = count + 1
Step 12 : I = I +1
Step 13 : print num " occurs " count " times "
Step 14 : stop

15. Write an algorithm to find area of triangle.

Step 1 : input base, height
Step 2 : area = $\frac{1}{2}$ * base * height
Step 3 : print "Area of triangle = ", area
Step 4 : stop

: SUMMARY :

- **Flowchart** is a graphical representation of sequence of steps to be used to solve a problem using programm language. Main symbols used are for start/stop, input/output, process, decision, connector etc.
- **Algorithm** is a textual representation of sequence of steps to be executed in English like language. It pseudo code language.

: MCQs :

1. _____ is a diagrammatic way to represent steps to solve any problem, while _____ is a sequence of instructional steps to be followed to solve any problem.
   (a) Flowchart, Instructions (b) Flowchart, Algorithm
   (c) Workflow, Algorithm (d) None of above
2. Diamond shape symbol is used to represent _____ in flowchart
   (a) Start (b) Stop (c) Input/Output (d) Decision
3. In flowchart, Rectangle shape is used to represent
   (a) Start (b) Process (c) Stop (d) Decision
4. What is algorithm?
   (a) Application code (b) Type of programming language
   (c) Step by step procedure for calculations (d) None of above

: ANSWERS :

1. (b) **2.** (d) **3.** (b) **4.** (c)

: EXERCISES :

1. Explain what is an algorithm.
2. List types of algorithm. explain it.
3. Explain pseudo code.
4. What is flow chart ? explain it.
5. Explain with diagram various symbol of flow chart.
6. Give difference between flow chart and algorithm.
7. List various properties of good algorithm.

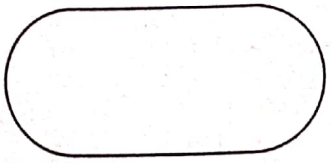
: ANSWERS TO SELECTED EXERCISES :

1. What is Algorithm and Flowchart? Explain with some suitable example.

Ans. :

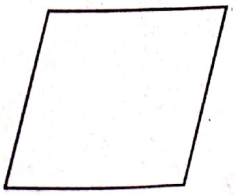
A **Flowchart** is graphical or diagrammatical representation of sequence of any problem to be solved by computer programming language. It is used to prepare first before to write any program. It is also used to correct and debug a program flow after coding part is completed. Through flowchart it is easy to understand the logic and sequence of problem. After drawing flowchart it become to easy write any program. There are various standard symbols available to represent logic of problem which are ...

1. Start and Stop



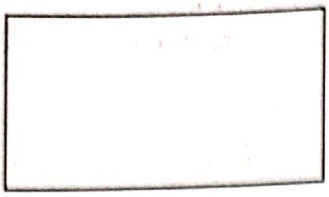
Used to start and stop or end flow chart.

2. Input / Output



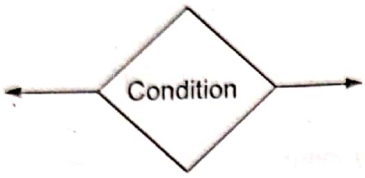
Used for input and output operation

3. **Process or expression representation**



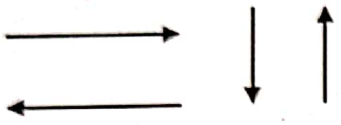
Used to represent any assignment or expression

4. **Decision**



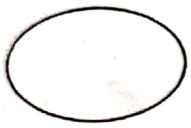
Used for any decision making statements.

5. **Direction of data flow or flow lines**



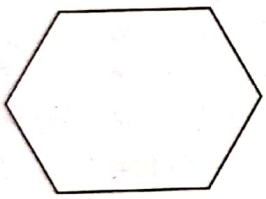
Indicates flow of sequence or data.

6. **Connector**



Connecting flow lines from different places.

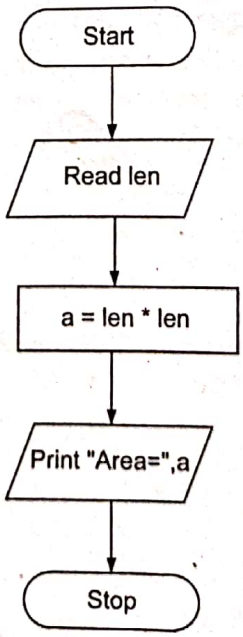
7. **Loop**



Used for iteration or looping statement.

Example :

1. Draw a flowchart to find area of square.



Algorithm

An **algorithm** is a finite sequence of well defined steps or operations for solving a problem in systematic manner. These are rules for solving any problems in proper manner. Instruction are written in the natural language. It is also called step by step solution.

Types of algorithm

There are various types algorithm techniques exists, according to types of problem appropriate types of algorithm used.

1. Divide and conquer
2. Greedy method
3. Branch and bound
4. Recursion
5. Effectiveness

Example

Write an algorithm to find out area of square.

Step 1 : input length : len

Step 2 : calculate area=len*len

Step 3 : print area= " area = ", area

Step 4 : stop

: SHORT QUESTIONS :

1. What is pseudo code?

⇒ It is a language or part of language used to help programmer to write logic of flowchart or algorithm in symbolic form.

2. What is an algorithm?

⇒ Algorithm is a step by step procedure for solving a given problem.

❖

CHAPTER

5

Input–Output

5.1 INTRODUCTION

5.2 INPUT WITH scanf() FUNCTION

5.3 OUTPUT WITH printf() FUNCTION

5.4 CHARACTER INPUT

5.5 CHARACTER OUTPUT

5.6 STRING INPUT/OUTPUT

5.7 FORMATTED INPUT USING FORMAT SPECIFIERS

5.8 FORMATTED OUTPUT

5.9 SOLVED PROGRAMMING EXAMPLES

❖ SUMMARY

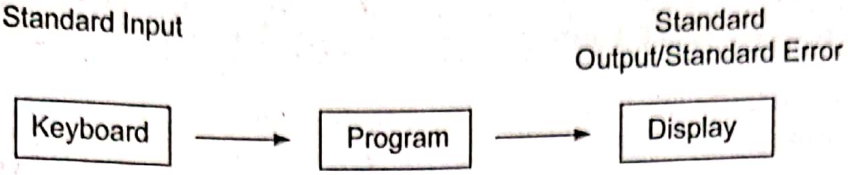
❖ MCQs

❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES

❖ SHORT QUESTIONS

5.1 INTRODUCTION :

Every language has some feature that provides an interaction between program and the user of the program. When the program requires any type of data from the user, the program displays the appropriate message to the user and the user provides the data to the program. The 'C language uses following environment for input, output and error messages.



From above figure, it is clear that the keyboard is used as the standard input device i.e program gets the data from user through keyboard, while display/monitor is used as standard output device as well as standard error device i.e output form the program as well as any error messages generated by the program are displayed on the monitor.

'C' language uses the references **stdin** for standard input device, **stdout** for standard output device, and **stderr** for standard error device.

The 'C' language does not provide any input/output operations as part of the language. The operations are available as library functions. The input/output functions are provided as library through the header file stdio.h. As we have already understood some programs where the input from the user is taken by using scanf() function, and the output is displayed using printf() function, we have to include the header file stdio.h is because in that file the scanf() and printf() functions are declared. The header file stdio.h contains the definitions of stdin, stdout and stderr.

Every input/output device has buffer (memory) associated with it. Input/output operations take place through the buffer associated with it. So, we can say that standard input/output operations are buffered.

5.2 INPUT WITH scanf() FUNCTION :

We have already used scanf() function in previous chapter programs without explaining the details. The scanf() function is used to get formatted input from standard input device which is keyboard. The syntax of scanf() function is

```
scanf("control string", list of addresses of variables);
```

where, control string is a string which specifies the format specifiers preceded by symbol %. After the control string, the variables with their addresses are written separated by comma.

The format specifiers are shown in following table

Character	Meaning
c	Character
d	Integer
f	Float
s	String
e	Double in scientific notation
o	Octal integer
x	Hexadecimal integer
u	Unsigned integer
%	Print % character

For example,

```
char  x;
int  y;
float  z;
scanf("%c  %d  %f",  &x,  &y,  &z);
```

in above scanf() function, **"%c %d %f" is control string**, where format specifiers are used, each prec... by % symbol. The first input will be treated as character, second as integer, and third as float, and ... be stored in x, y and z respectively. The variables x, y and z are all preceded by & symbol, because sca... function requires the address of the variable in which the value is to be stored.

With the format specifier s which is used to get string from standard input, we do not require to prec... the variable name with & symbol. Format specifier s will be explained later when we will study the strin...

5.3 OUTPUT WITH printf() FUNCTION :

We have already used printf() function in previous chapter programs. It is used to display formatted outp... on the standard output device, which is screen. The syntax of printf() function is

```
printf("Control string", list of variables);
```

control string is a string which specifies the format specifiers preceded by symbol %.

In addition to format specifires we can write some string messages we want to display and also the esca... sequences like \n \t etc. After the control string, the variables are written separated by comma.

For example,

```
float avg=13.7;
printf(" Average = %f\n", avg);
printf("Over");
```

Here, "Average =" is user message, %f is the format specifier and \n is the escape sequence. See th... all three things are combined into one string: " Average = %f\n". After the control string comma (,) symb... is used and then the name of variable whose value is to be printed. The next printf() statement output w... be shown in the next line because \n escape sequence at the end of first printf() control string.

The output will be :

```
Average = 13.700000
Over
```

Program :

/* Write a program to calculate simple interest using the formula I = PRN/100 where P is principal amount, R is rate of interest annually and N is duration in years */

```
#include <stdio.h>
#include <conio.h>
main()
{
    int p,n;
    float r,i;
    clrscr();
    printf("Please give principal amount\n");
    scanf("%d", &p);
    printf("Please give rate of interest and number of years\n");
    scanf("%f %d", &r,&n);
    i=(p*r*n)/100;
    printf("Interest of amount Rs. %d at rate %f%% for %d years
    = %f\n",p,r,n,i);
}
```

Output :

```
Please give principal amount
10000
Please give rate of interest and number of years
9.5
2
Interest of amount Rs. 10000 at rate 9.500000% for
2 years = 1900.000000
```

5.4 CHARACTER INPUT :

'C language provides getchar() function for getting character input from keyboard. As we know input/output is buffered, so getchar() function reads the next character from the input buffer and returns the ASCII value of the character. The syntax of getchar() is

```
character_variable = getchar();
```

where, character_variable is a **char** type of variable.

For example, the statements

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

will store the ASCII value of the character pressed from the keyboard.

We can also use scanf() function using format specifier %c to read a character from keyboard.

Program :

```
/* Program explaining the character input using scanf() functions */
#include <stdio.h>
#include <conio.h>
main()
{
    char c;
    printf("Enter one character\n");
    scanf("%c",&c);
    printf("Input character is = %c\n",c);
}
```

Output :

```
Enter one character
a
Input character is = a
```

Now, in above program, instead of scanf() function, we can use getchar() function.

Replacing **scanf("%c",&c);** statement with **c=getchar();** statement, and running the program gives same output.

Output with getchar()

```
Enter one character
a
Input character is = a
```

Program :

```
/* Write a program which accepts a character from keyboard and display it's ASCII code */
#include <stdio.h>
#include <conio.h>
main()
{
    char c;
    printf("Enter one character\n");
    c=getchar();
    printf("ASCII code of input character %c is = %d\n",c,c);
}
```

Output :

```
Enter one character
a
ASCII code of input character a is = 97
```

As we can see that the getchar() function requires ENTER key to be pressed to terminate the input. Some times we want to simply read the characters (particularly) for reading the strings without terminating each character with ENTER key. In that case we can use **getch() or getche()** functions. **getch()** function does not echo the input character, while **getche()** function echoes the input character on the screen. The conio.h header file supports the above two functions. The prototype is

```
int getch();
int getche();
```

Program :

```
/* Write a program which accepts two characters one after other using getch() function */
#include<stdio.h>
#include<conio.h>
main()
{
    char c1,c2;
    clrscr();
    printf("Enter first character\n");
    c1 = gecth();
    printf("Enter second character\n");
    c2 = getch();
    printf("The first character =%c and second = %c\n",c1,c2);
}
```

Output :

```
Enter first character
a  Input will not be echoed on screen
Enter second character
b  Input will not be echoed on screen
The first character =a and second = b
```

Program :

```
/* Write a program which accepts two characters one after other using getche() function*/
#include<stdio.h>
#include<conio.h>
main()
{
    char c1,c2;
    clrscr();
    printf("Enter first character\n");
    c1 = gecthe(); /* echo character */
    printf("Enter second character\n");
    c2 = getche(); /* echo character */
    printf("The first character =%c and second = %c\n",c1,c2);
}
```

Output :

```
Enter first character
aEnter second character
bThe first character =a and second = b
```

5.5 CHARACTER OUTPUT :

'C language provides putchar() function for displaying one character on the screen. It prints the character whose ASCII- value is provided to it. The syntax of putchar() is

```
putchar(character_variable);
```

where, character_variable will be char type variable holding the ASCII value.

For example, the statement sequence,

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

will display the character 'a' on output.

Program :

```
/* Write a program to convert given lowercase letter to uppercase character.
uppercase ASCII value = lowercase ASCII value -32 */
#include <stdio.h>
#include <conio.h>
main()
{
    char c;
    clrscr();
    printf("Enter any one lowercase character\n");
    c= getchar();
    printf("lowercase character = ");
    putchar(c);
    printf(" converted to uppercase character = ");
    putchar(c-32);
}
```

Output :

```
Enter any one lowercase character
a
lowercase character = a converted to uppercase
character = A
```

Program :

```
/* Write a program which accepts two characters one after other using getchar() or scanf() functions */
#include<stdio.h>
#include<conio.h>
main()
{
    char c1,c2;
    clrscr();
    printf("Enter first character\n");
    scanf("%c",&c1);
    printf("Enter second character\n");
    scanf("%c",&c2);
    printf("The first character =%c and second = %c\n",c1,c2);
}
```

Output :

```
Enter first character
a
Enter second character
The first character =a and second =
```

Explanation :

Here, the program does not work correctly, it does not wait for the next character input. This happens because %c format specifier of scanf() function requires that the character input to be terminated by ENTER key, for our example character 'a' terminated by ENTER key. Now ENTER itself is character, which remains in the memory buffer, so when the next scanf() statement is executed, it gets that character as input.

If you use getchar() in place of scanf(), same thing will happen. So, what is the solution? The solution is somehow we should clear the memory buffer before we read the next character. So, ENTER key of first input will be removed from input buffer.

We can use **fflush()** function which is again defined in stdio.h file.

Program :

```
/* Write a program which accepts two characters one after other using getchar() or scanf() functions.
Use fflush() to avoid buffer problem */
#include<stdio.h>
#include<conio.h>
main()
{
    char c1,c2;
    clrscr();
    printf("Enter first character\n");
    scanf("%c",&c1);
    fflush(stdin); /* clear input buffer */
    printf("Enter second character\n");
    scanf("%c",&c2);
    printf("The first character =%c and second = %c\n",c1,c2);
}
```

Output :

```
Enter first character
a
Enter second character
b
The first character =a and second =b
```

5.6 STRING INPUT/OUTPUT :

String is a sequence of characters. 'C' language provides the function gets() to get string type of input from keyboard, while function puts() is used to display string on the display screen. We can also use scanf() and printf() with format specifier %s for strings. As string is an array of characters, we will study gets() and puts() functions in detail when we will study the arrays.

5.7 FORMATTED INPUT USING FORMAT SPECIFIERS :

We have already studied scanf() function, with its syntax. Here, we will try to use scanf() function to read various types of data in different formats.

The format specification for integer data is %wd

Where, w stands for width (number of digits), while d stands for integer number.

For example,

scanf(" %3d %4d", &num1,&num2); statement is used and if the input is given like

234 4563

then, 234 will be assigned to num1 and 4563 will be assigned to num2.

But, if the input is 2344 563

Then, num1=234 and num2=4. Here, the num1 is assigned first three digits of input, while the variable num2 is assigned only 4, because scanf() function tries to read 4 digits for num2, but immediately after 4 it finds space, where it stops further reading. The scanf() function reads either number of characters specified or upto the first nondigit character or white space character like blank, spacebar or newline.

For floating point numbers or double numbers, width is not necessary. To read **float** number, use **%f**, while for **double** number use **%lf.**

5.8 FORMATTED OUTPUT :

It is displaying the output in particular format. By using formatted output, various types of reports in particular formats can be generated. We have already seen that printf() function is used to display the output on the screen. Here we will try to understand how width, justification etc can be specified.

The format specification for integer data is %wd

Where, w stands for width (number of digits) to be used for printing, while d stands for integer number. By default integer is printed right justified, we can change the justification by using hyphen (-) flag.

Following table lists output flags :

Flag	Meaning
-	Left justified print, remaining spaces left blank
+	Print + or – before the number (signed number)
0	Leading zeros are printed
#o	0 before octal number
#x	0x before hex number
e	Scientific notation

Some Examples :

Format	Output	
`printf("%d",456);`	`456`	
`printf("%5d",456);`	`bb456`	b stands for blank
`printf("%2d",456);`	`456`	
`printf("%05d",456);`	`00456`	
`printf("%-5d",456);`	`456bb`	b stands for blank
`printf("%+5d",456);`	`b+456`	b stands for blank
`printf("%#5o",456);`	`b0710`	b stands for blank
`printf("%#5x",456);`	`0x1c8`	

The format specification for floating data is %w.pf

Where, w denotes width, p denotes number of digits after decimal point, and f denotes float number. If **e** is used in place of **f**, then the number is displayed in scientific notation.

If width is not specified, then the floating point number is displayed with six decimal point digits.

Format	Output	
printf("%f",45.678);	45.678000	
printf("%7.2f",45.678);	bb45.68	b stands for blank
printf("%-7.2f",45.678);	45.68	
printf("%7.2e",45.678);	4.57e+01	
printf("%e",45.678);	4.567800e+01	

Program :

```
/* Program demonstrating formatted input and output */
#include<stdio.h>
#include  <conio.h>
main()
{
    int  a,b;
    floa    t  p,q;
    double  x,y;
    clrscr();
    printf("Enter  two  integer  numbers\n");
    scanf("%3d  %4d",&a,&b);
    printf("%d  %d\n",a,b);
    printf("Enter  two  float  numbers\n");
    scanf("%f  %f",&p,&q);
    printf("%7.2f  %7.2e",p,q);
    printf("Enter  two  double  numbers\n");
    scanf("%lf  %lf",&x,&y);
    printf("%e  %7.2e\n",x,y);
}
```

Output :

```
Enter  two  integer  numbers
46
46
Enter  two  float  numbers
4.5  4.5
    4.50  4.50e+00
Enter  two  double  numbers
7.90  7.90
7.900000e+00  7.90e+00
```

5.9 SOLVED PROGRAMMING EXAMPLES :

Program :

```
/* Write a program to exchange two variables. Get the values from user. */
#include <stdio.h>
#include <conio.h>
void main()
{
    int x,y,temp;
    clrscr();
    printf("Give two numbers\n");
    scanf("%d%d",&x,&y);
    printf("Before exchange x =%d and y =%d\n",x,y);
    temp =x;
    x = y;
    y = temp;
    printf("After exchange x =%d and y =%d\n",x,y);
}
```

Output :

```
Give two numbers
10
35
Before exchange x =10 and y =35
After exchange x =35 and y =10
```

Program :

```
/* Write a program to print given decimal number into hexadecimal and octal number */
#include <stdio.h>
#include <conio.h>
void main()
{
    int x;
    clrscr();
    printf("Give the decimal number\n");
    scanf("%d",&x);
    printf("Decimal Number = %d\n",x);
    printf("Hexadecimal Number = %x\n",x);
    printf("Octal Number = %o\n",x);
}
```

Output :

```
Give the decimal number
16
Decimal Number = 16
Hexadecimal Number = 10
Octal Number = 20
```

Program :

/* Write a program to convert the distance in feet and inches into corresponding centimeter. */

```
#include <stdio.h>
#include <conio.h>
void main()
{
    float feet, inch,cm;
    clrscr();
    printf("Give distance feet and inch\n");
    scanf("%f%f",&feet,&inch);
    cm = feet * 30 + inch * 2.5;
    printf("Distance is %5.2f feet and %5.2f inch \n",feet,inch);
    printf("Correspsonding distance in cm = %6.2f\n",cm);
}
```

Output :

```
Give distance feet and inch
5 5
Distance is 5.00 feet and 5.00 inch
Correspsonding distance in cm = 162.50
```

Program :

/* Write a program to calculate volume of a cylinder. Formula $V = \pi r^2 h$ */

```
#include <stdio.h>
#include <conio.h>
#define pi 3.1415
void main()
{
    int r,h;
    float v;
    clrscr();
    printf("Give radius and height of cylinder\n");
    scanf("%d%d",&r,&h);
    v = pi * r * r *h;
    printf("Volume of cylinder = %6.2f\n",v);
}
```

Output :

```
Give radius and height of cylinder
2 3
Volume of cylinder = 37.70
```

Program :

```
/* Program to find the area of a triangle, given three sides*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    int    a, b, c;
    double s, area;
    clrscr();

    printf("Enter the values of three sides of triangle\n");
    scanf ("%d %d %d", &a, &b, &c);

      /* compute s*/

    s = (a + b + c) / 2.0;
    area = sqrt ( s * (s-a) * (s-b) * (s-c));
    printf ("Area of a triangale = %lf\n", area);
}
```

Output :

```
Enter the values three sides of triangle
3
4
5
Area of a triangale = 6.000000
```

: SUMMARY :

- 'C' language uses the references **stdin** for standard input device, **stdout** for standard output device, and **stderr** for standard error device.
- Header file **stdio.h** contains the definitions of stdin, stdout and stderr.
- Function **scanf()** is used to get formatted input from keyboard.
- Function **printf()** is used to display formatted output on the standard output device, which is screen display.
- Function **getchar()** is used to get character from keyboard. As I/O is buffered, getchar() reads next character from input buffer returns the ASCII value of the character.
- Function **putchar()** is used to print one character on output device screen.
- We can format the output by using **output flags** in printf() function.

: MCQs :

1. In scanf(), which format specifier is used to read integer?
   (a) f (b) c (c) d (d) x
2. Which header file supports getch() function?
   (a) stdio.h (b) conio.h (c) math.h (d) string.h
3. What is the output of statement
   printf("%05d",456);
   (a) 00456 (b) bb456 (c) 45600 (d) None of above
4. What is the output of
   printf("%–7.2f",45.678);
   (a) 45.68 (b) 00045.68 (c) 45.67 (d) 4.55 e+01
5. In scanf(), which format specifier is used to read float?
   (a) f (b) c (c) d (d) x

: ANSWERS :

1. (d) **2.** (b) **3.** (a) **4.** (a) **5.** (a)

: EXERCISES :

1. Explain stdin, stdout and stderr in 'C' language.
2. Write the syntax of scanf() function.
3. Write the syntax of printf() function.
4. What is formatted output. How it can be done using printf() statement?
5. Explain use of getchar() and putchar() functions.
6. What is the difference between reading a string with scanf() and gets()?
7. Write a program to separate integer part and fractional part from given floating point number. For example, if floating point number 26.645, integer part is 26 and fractional part is 0.645.
8. What will be the output of following code for input values 20 and 30 int a,b,c;

```
c = scanf("%d%d", &a,&b);
```

9. Find out the output of the following 'C' code.

```
#include<stdio.h>
void main()
{
    int iX=7, iY=8,iZ=9;
    printf("\n%d %d %d",++iX,iX,iX++);
    printf("\n%d %d %d",iZ,++iZ,iZ++);
}
```

10. Find out the output of the following 'C' code.

```
#include<stdio.h>
void main()
{
    char caddress[30]="Gtu Examination";
    int iX= 67584;
```

```
    float  fZ=95.7658;
    printf("%.6s\n",caddress);
    printf("%7s\n",caddress);
    printf("%010d\n",iX);
    printf("%10d\n",iX);
    printf("%*.*f\n",7,2,fZ);
    printf("%e\n",fZ);
    printf("%12.4e\n",-fZ);
}
```

: ANSWERS TO SELECTED EXERCISES :

9. Find out the output of the following 'C' code.

```
#include<stdio.h>
void  main()
{
    int  iX=7,  iY=8,iZ=9;
    printf("\n%d  %d  %d",++iX,iX,iX++);
    printf("\n%d  %d  %d",iZ,++iZ,iZ++);
}
```

Ans. :

```
9   8   7
11  11  9
```

Explanation :

For postfix increment, value will be printed first and then increment takes place. For prefix increment, increment takes place first and then the new value will be printed. And evaluation starts from last expression in printf. For example in printf("\n%d %d %d",++iX,iX,iX++); statement, evaluation sequence is

```
iX++;   /*  7  printed  and  then  iX  becomes  8  */
iX; /*  8  printed  */
++iX;   /*  ix  becomes  9  and  then  printed  */
```

So, output of first line is 9 8 7.

Similarly, second printf statement output is generated.

10. Find out the output of the following 'C' code.

```
#include<stdio.h>
void  main()
{
    char  caddress[30]="Gtu  Examination";
    int  iX=  67584;
    float  fZ=95.7658;
    printf("%.6s\n",caddress);
    printf("%7s\n",caddress);
    printf("%010d\n",iX);
    printf("%10d\n",-iX);
    printf("%*.*f\n",7,2,fZ);
```

```
    printf("%e\n",fZ);
    printf("%12.4e\n",-fZ);
}
```

Ans :

```
Gtu  Ex
Gtu Examination
    0000002048
    -2048
    95.77
    9.576580e+01
    -9.5766e+01
```

Explanation :

printf("%.6s\n",caddress);	here, precision specifier, so first 6 characters printed
printf("%7s\n",caddress);	here width specified is less than actual requirement, so whole string printed without truncation.
printf("%010d\n",iX);	iX is integer which can store maximum 32768. but value assigned is 67584. So, actual value stored in
	iX = 67584-32768-32768= 2048 which is printed in 10 digits with preceding 0s.
printf("%10d\n",-iX);	-iX printed in 10 digits.
printf("%*.*f\n",7,2,fZ);	fZ printed with 2 decimal digit accuracy.
printf("%e\n",fZ);	fZ printed in scientific notation
printf("%12.4e\n",-fZ);	fZ printed in scientic notation with 4 digits decimal accuracy in 12 width

: SHORT QUESTIONS :

1. **What is the difference between getch() and getche() function?**

⇒ getch() function does not echo the input character on the screen, while getche() function echoes the input character on the screen. The header file conio.h supports the functions getch() and getche().

2. **Which output flag is used to display the number in scientific notation?**

⇒ The flag is used to display the number in scientific notation is: e

3. **Output flags are used in which function?**

⇒ Output flags are used in printf() function.

❖

CHAPTER

6

Decision Making Structure

6.1 INTRODUCTION

6.2 if STATEMENT

6.3 if...else STATEMENT

6.4 NESTED if STATEMENT

6.5 If...else...if LADDER (MULTIPLE if...else) STATEMENT

6.6 switch STATEMENT

6.7 break STATEMENT

6.8 default KEYWORD

6.9 goto STATEMENT

6.10 SOLVED PROGRAMMING EXAMPLES

❖ SUMMARY

❖ MCQs

❖ EXERCISES AND ANSWERS TO SELECTED EXERCISES

❖ SHORT QUESTIONS

6.1 INTRODUCTION :

In real life, problems are not simple. Many times certain actions are to be performed depending on whether a particular condition is satisfied. So, actions are based on fulfillment of certain conditions. 'C' language provides control structures for decision making as well as for looping. In this chapter we will understand decision structures, while the looping control structures will be discussed in next chapter.

For example, as per Indian law, the minimum age of a boy for marriage is 21 years, while that of a girl is 18 years. So, the decision whether marriage is valid or not that depends on two things: sex and age of a person.

'C' language provides different control structures for decision making- like if statement, if...else statement, else-if ladder, switch statement, goto statement and break statement.

6.2 if STATEMENT :

This is the simplest statement which checks a condition. If the condition is true then the statements are executed, otherwise they are not executed.

The syntax of if statement is:

```
if    (condition)
      statement(s);
```

Figure 6.1 explains it

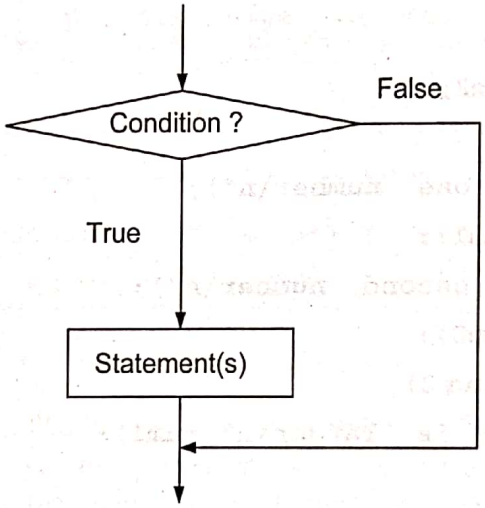


Fig. 6.1

If there are more than one statements to be executed then they are put within { } symbols.

6.3 if...else STATEMENT :

In certain cases, we want to execute one set of statements, if condition is satisfied, and other set of statements if condition is false. In that case we can use if... else statement.

The syntax is :

```
if(condition)
      statement(s)1;
else
      statement(s)2;
```

If condition is TRUE, then statement(s)1 are executed, otherwise statement(s)2 are executed.

Figure 6.2 explains it.

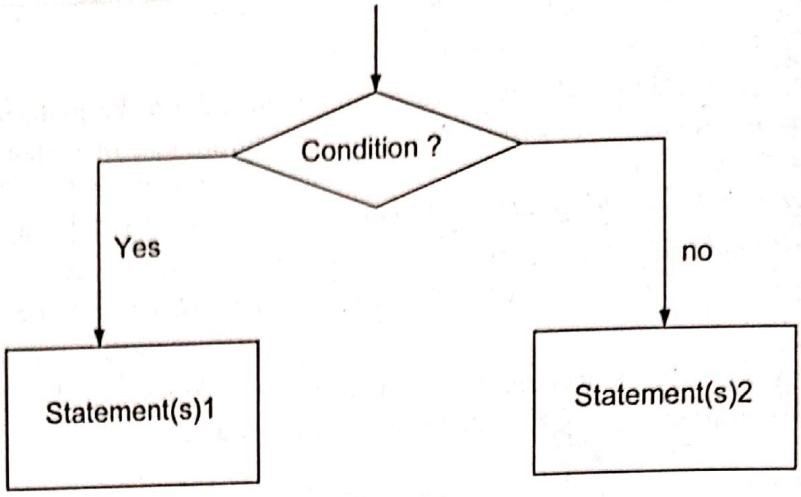


Fig. 6.2

Following program explains the use of if statement.

Program :

```
/* Write a program to find larger of two numbers using if statement */
#include<stdio.h>
#include<conio.h>
main()
{
    float  num1,num2;
    clrscr();
    printf("Enter  one  number\n");
    scanf("%f",&num1);
    printf("Enter  second  number\n");
    scanf("%f",&num2);
    if  (num1  >  num2)
        printf("%f  is  larger\n",num1);
    else
        printf("%f  is  larger\n",num2);
}
```

Output :

```
Enter  one  number
3.4
Enter  second  number
7.5
7.500000  is  larger
```

Program :

```
/* Program to read x and y co-ordinates and finds quadrant of point */
#include<stdio.h>
#include<conio.h>

void main()
{
    float x,y;
    clrscr();
    printf("Enter x and y co-ordinates of a point\t");
    scanf("%f%f",&x,&y);
    if(x==0 && y==0) printf("Point lies on Origin\n");
    if(y==0 && x!=0) printf("Point lies on x-Axis\n");
    if(x==0 && y!=0) printf("Point lies on y-Axis\n");
    if(x>0 && y>0)     printf("Point in the First Quadrant\n");
    if(x>0 && y<0)     printf("Point in the Second Quadrant\n");
    if(x<0 && y<0)     printf("Point in the Third Quadrant\n");
    if(x<0 && y>0)     printf("Point in the Fourth Quadrant\n");
    getch();
}
```

Output :

```
Enter x and y co-ordinates of a point -3 3
Point in the Fourth Quadrant
```

6.4 NESTED if STATEMENT :

For complex problems we need to check more than one condition. The if statement can be used inside another if statement. This type of use of if statement is called as nested if statement. The nesting can be up to any level. The flowchart for finding largest of three numbers is given in figure 6.3.

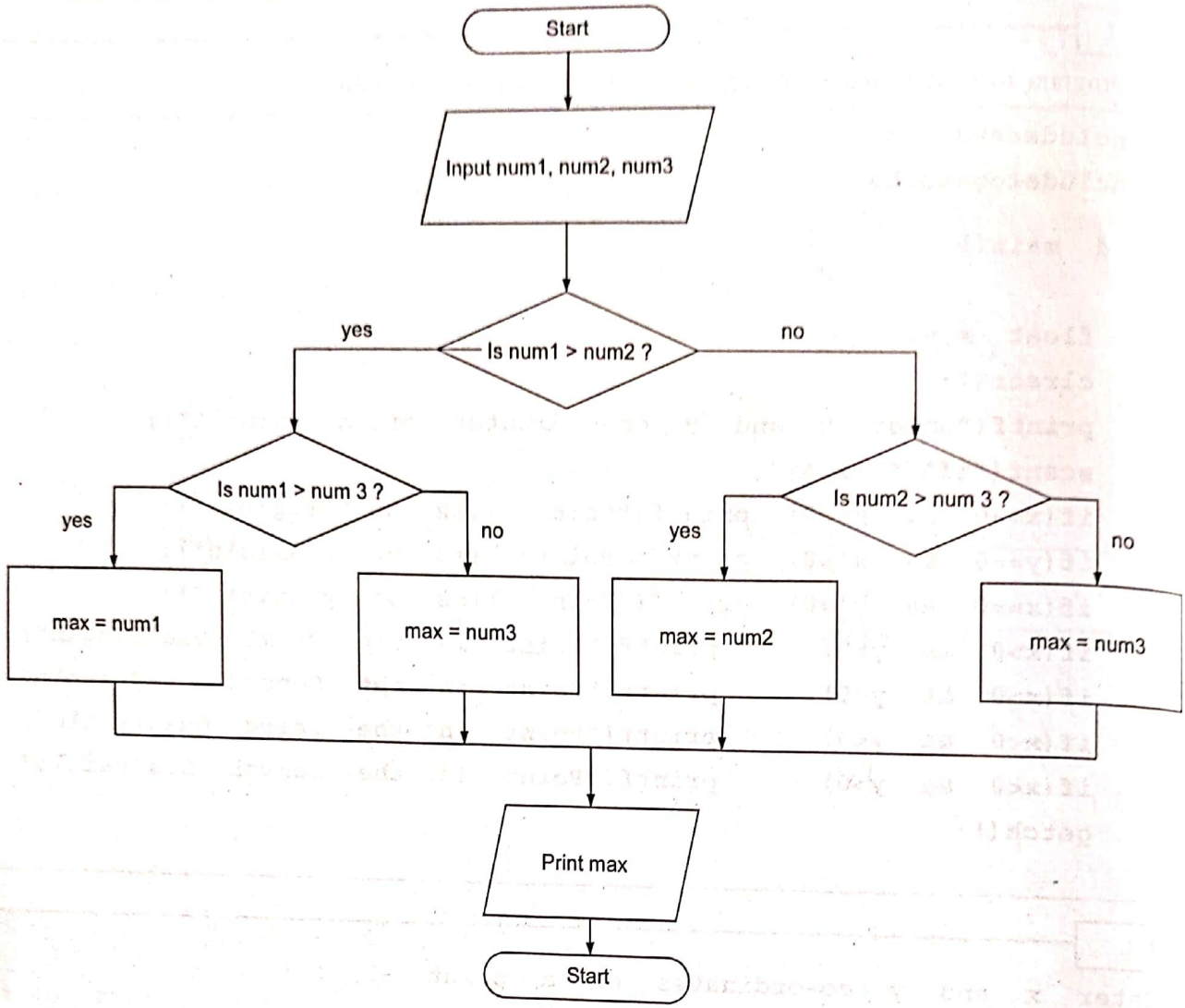


Fig. 6.3

Following program explains the nested if statement.

Program :

```
/* Write a program to find maximum of three numbers */
#include<stdio.h>
#include<conio.h>
main()
{
    int num1,num2,num3;
    int max;
    clrscr();
    printf("Enter three numbers\n");
    scanf("%d%d%d",&num1,&num2,&num3);
    if (num1 > num2)
    {
        if (num1 > num3)
            max = num1;          /* num1 largest */
        else
```

```
            max = num3;     /* num3 largest */
    }
    else
    {
    if (num2 > num3)
        max = num2;     /* num2 largest */
    else
        max = num3;     /* num3 largest */
    }
    printf("maximum of %d, %d and %d is = %d\n",num1,num2,num3,max);
}
```

Output-1 :

```
Enter three numbers
23 34 15
maximum of 23, 34 and 15 is = 34
```

Output-2 :

```
Enter three numbers
-4 -23 -45
maximum of -4, -23 and -45 is = -4
```

6.5 If...else...if LADDER (MULTIPLE if...else) STATEMENT :

The two-way decision provided by if...else statement is not sufficient when we are having many choice, 'C' language provides a multiple if...else statement for that purpose.

The syntax is:

```
if (condition1)
    statement(s)1;
else if (condition2)
    statement(s)2;
else if (condition3)
    statement(s)3;
.
.
.
else if(conditionN)
    statement(s)N;
else
    default_statement(s);
```

From the syntax, it is very clear that only one set of statement(s) will be executed, depending on the condition being TRUE.

Flow chart of Figure 6.4 explains multiple if...else statement.

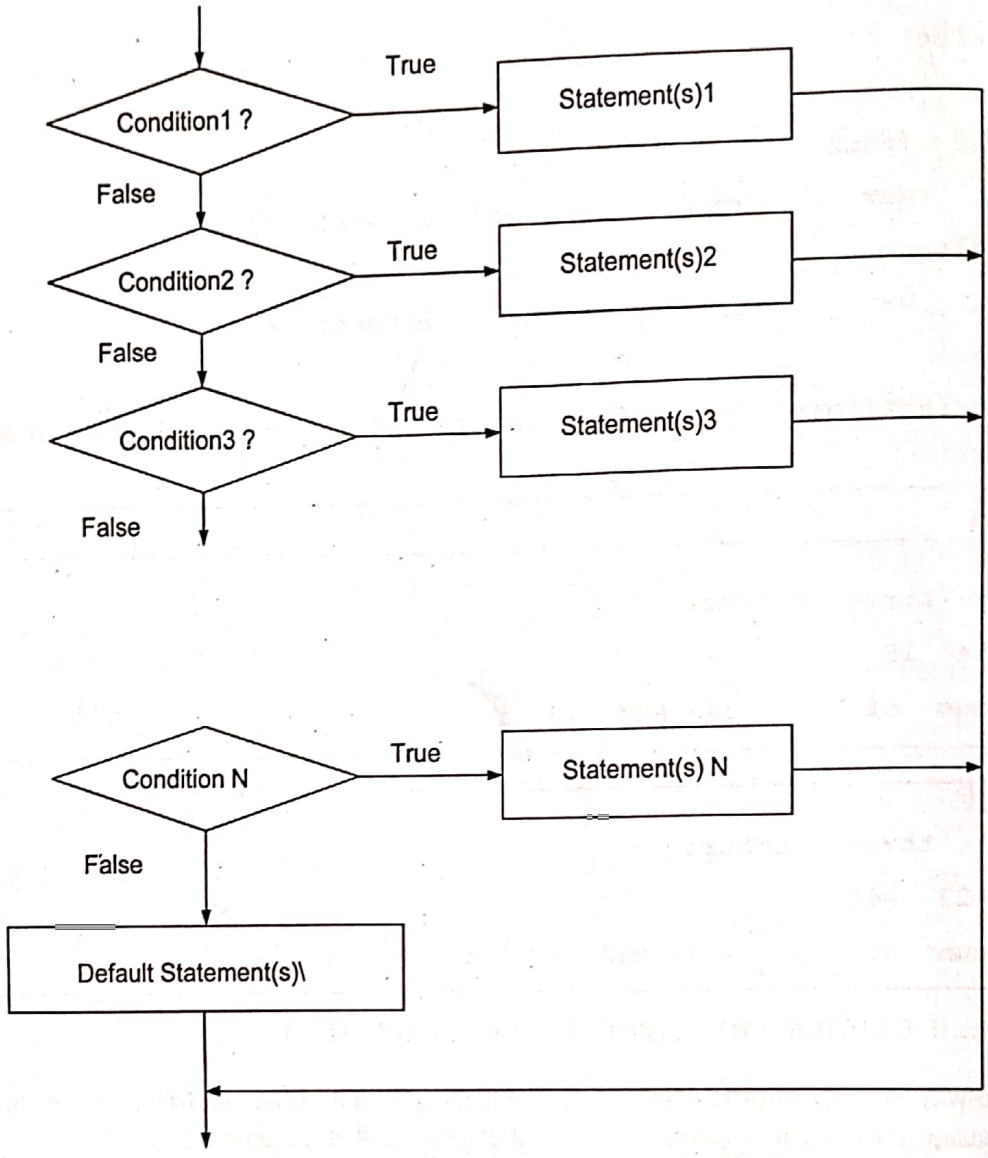


Fig. 6.4

Program :

/* Write a program which asks day number and prints corresponding day name i.e if input is 5, it wi print the fifth day of week which is Thursday. Sunday being the first day. */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  day;
    clrscr();
    printf("Enter  day  number  between  (1-7)\n");
    scanf("%d",&day);
    if(day==1)
        printf("Sunday\n");
        /*  equality  checked  with  ==  (=  symbol  twice)
            and  not  single  =  */
```

```
    else  if(day==2)
        printf("Monday\n");
    else  if(day==3)
        printf("Tuesday\n");
    else  if(day==4)
        printf("Wednesday\n");
    else  if(day==5)
        printf("Thursday\n");
    else  if(day==6)
        printf("Friday\n");
    else  if(day==7)
        printf("Saturday\n");
    else
        printf("Wrong  input\n");
}
```

Output-1 :

```
Enter  day  number  between  (1-7)
4
Wednesday
```

Output-2 :

```
Enter  day  number  between  (1-7)
9
Wrong  input
```

Program :

/* Write a program to check the category of given character. Digit, Upper Case, Lower Case or other symbol */

```
#include<stdio.h>
#include<conio.h>
main()
{
    char  ch;
    clrscr();
    printf("Enter  one  character\n");
    scanf("%c",&ch);
    if  ((  ch  >=  '0')  &&  (ch  <=  '9'))
        printf("The  character  %c  is  digit\n",ch);
    else  if  ((  ch  >=  'a')  &&  (ch  <=  'z'))
        printf("The  character  %c  is  Lower  Case\n",ch);
    else  if  ((  ch  >=  'A')  &&  (ch  <=  'Z'))
        printf("The  character  %c  is  Upper  Case\n",ch);
```

```
    else
        printf("The character %c is other symbol\n",ch);
}
```

Output-1 :

```
Enter one character
6
The character 6 is digit
```

Output-2 :

```
Enter one character
a
The character a is Lower Case
```

Output-3 :

```
Enter one character
S
The character S is Upper Case
```

Output-4 :

```
Enter one character
*
The character * is other symbol
```

6.6 switch STATEMENT :

'C' language provides a switch statement. In a complex problem, if many alternative values are available, then writing the code using multiple if...else becomes lengthy and also difficult to manage. At that time switch statement which is a multi-way decision statement is handy, because management of code becomes easy.

The syntax of switch statement is :

```
switch (variablename or expression)
    {
    case value1  :   statement(s)1;
                     break;
    case value2  :   statement(s)2;
                     break;
        .
        .
        .
    case valueN  :   statement(s)N;
                     break;
    default      :   default_statement(s);
    }
```

Here, statement(s)1 will be executed if the value of variablename or expression is value1, similarly statement(s)2 will be executed if the value of variablename or expression is value2. If the value of variablename or expression

does not match any values from value1 to valueN, then the default_statement(s) are executed.

It is to be noted that writing the default value and its corresponding default_statement(s) is optional. **The value of expression or variable written after the switch statement must be either character or integer value.**

In multiple if...else statement, we have many conditions, while in switch statement, we have only one expression (which has only one value), depending on the value of expression different set of statements are executed. It also means that whatever we can do with switch statement can be done by multiple if...else statement, but reverse is not always true.

Figure 6.5 is a flow chart which explains the switch statement.

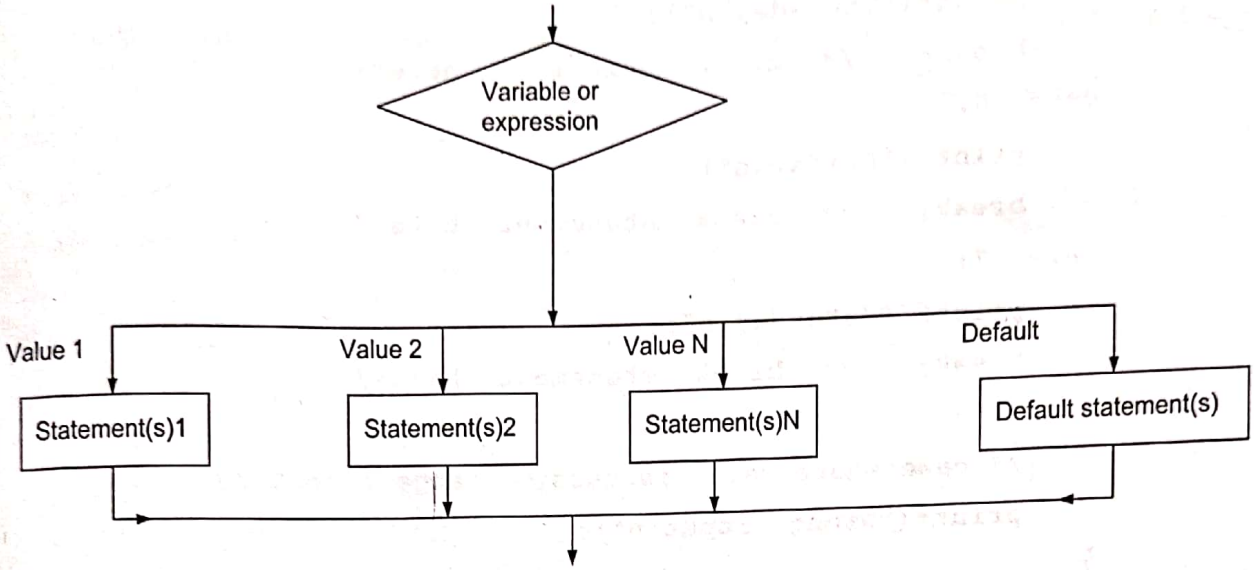


Fig. 6.5

Program :

/* Write a program which asks day number and prints corresponding day name.

This program we have already written using multiple if...else statement */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  day;
    clrscr();
    printf("Enter  day  number  between  (1-7)\n");
    scanf("%d",&day);
    switch(day)    /*  day  variable  has  only  integer  value*/
        {
        case  1:
            printf("Sunday\n");
            break;    /*  break  statement  here*/
        case  2:
            printf("Monday\n");
            break;    /*  break  statement  here*/
```

```
        case  3:
            printf("Tuesday\n");
            break;    /*  break  statement  here*/
        case  4:
            printf("Wednesday\n");
            break;    /*  break  statement  here*/
        case  5:
            printf("Thursday\n");
            break;    /*  break  statement  here*/
        case  6:
            printf("Friday\n");
            break;    /*  break  statement  here*/
        case  7:
            printf("Saturday\n");
            break;    /*  break  statement  here*/
        default:
            /* case where value is outside range 1 to 7 */
            printf("Wrong  input\n");
        }
}
```

Output-1 :

```
Enter  day  number  between  (1-7)
1
Sunday
```

Output-2 :

```
Enter  day  number  between  (1-7)
10
Wrong  input
```

If the code to be executed for different cases is same, then that code for different cases can be grouped together by writing those cases with their values as follows.

For example, if x is an integer variable, whose valid value are from 1 and 3, and if code for value 1 and value 3 are same, then it can be written like this.

```
switch(x)
    {
    case  1:
    case  3:
        statement(s);
        break;
```

```
    case  2:
        statement(s);
        break;
    default;
        statement(s);
    }
```

Program :

/* Write a program to print number of days in a given month. The program requires month number(between 1 to 12) as an input and then display the days in that month

Here, the code for many cases is same*/

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  month;
    clrscr();
    printf("Give  month  number  between  (1-12)\n");
    scanf("%d",&month);
    switch(month)
        {
            /*  number  of  days  in  month  1,3,5,7,8,10 and
                12=31  so  same  case  for  all    */
        case  1:
        case  3:
        case  5:
        case  7:
        case  8:
        case  10:
        case  12:
            printf("Number  of  days  in  month  number  %d  is  =  31\n");
            break;
        case  4:
              /*  number  of  days  in  month  4,6,9  and 11=30
                    so  same  case  for  all  */
        case  6:
        case  9:
        case  11:
            printf("Number  of  days  in  month  number  %d  is  =  30\n");
            break;
```

```
        case 2:
            printf("Number of days in month number %d is = 28
            29\n");
            break;
        default:
            printf("Wrong input\n");
            break;
        }
    }
```

Output-1 :

```
Give month number between (1-12)
12
Number of days in month number 12 is = 31
```

Output-2 :

```
Give month number between (1-12)
15
Wrong input
```

6.7 break STATEMENT :

We have used the break statement in switch statement. It causes an exit from the switch body. If it not written after each case statement, then control passes to the next statement, so remaining statemen of next case will also execute even if the case value do not match and the program will not function properl Try this example, where the break statement is omitted. It will not work as per our requirement.

Program :

```
/* Program demonstrating switch statement without break statement */
#include<stdio.h>
#include<conio.h>
main()
{
    int choice;
    clrscr();
    printf("Menu of operations\n");
    printf("==================\n");
    printf("1   Addition\n");
    printf("2   Subtraction\n");
    printf("3   Multiplication\n");
    printf("4   Division\n");
    printf("5   Modulo Division (Remainder)\n");
    printf("==================\n");
```

```
    printf("Give your choice:    ");
    scanf("%d",&choice);
    switch(choice)
        {
        case 1:
            printf("You selected Addition\n");
        case 2:
            printf("You selected Subtraction\n");
        case 3:
            printf("You selected Multiplication\n");
        case 4:
            printf("You selected Division\n");
        case 5:
            printf("You selected Modulo Division\n");
        default:
            printf("Wrong choice\n");
        }
}
```

Output-1 :

```
Menu of operations
====================
1   Addition
2   Subtraction
3   Multiplication
4   Division
5   Modulo Division (Remainder)
====================
Give your choice:
1
You selected Addition
You selected Subtraction
You selected Multiplication
You selected Division
You selected Modulo Division
Wrong choice
```

Output-2 :

```
Menu of operations
====================
1   Addition
2   Subtraction
3   Multiplication
```

```
4   Division
5   Modulo  Division  (Remainder)
====================
Give  your  choice:
4
You  selected  Division
You  selected  Modulo  Division
Wrong  choice
```

Now, modify the above program by writing break statement at the end of each case statement(s). Then run the program, it should work correctly.

6.8 default KEYWORD :

This statement is also used in switch statement. It is optional, but if it is used, then the statements written in that part get executed if any of the previous case values do not match. Normally, the default keyword is used in switch statement and written after all the cases.

6.9 goto STATEMENT :

The control structures we have studied in this chapter like if, if...else, multiple if ...else, switch statement provide branching in program which is conditional. 'C' language also provides **unconditional branching** mechanism called as goto statement. Structured programming practice does not encourage the use of goto statement, because use of goto statement in the program makes it difficult to understand and debug. 'C' provides the control structures using which we can always write a program without using goto statement.

The syntax is:

```
goto  label;
```

where, label is a valid 'C' identifier followed by colon (:) symbol. After the colon there can be any valid 'C' statement like,

```
label  :  statement;
```

The label referred in the goto statement can be declared before or after the goto statement.

For example,

```
goto  label;
.
.
.
label  :  statement;
```

is called as **forward reference.**

For example,

```
label  :  statement;
.
.
.
goto  label;
```

is called as **backward reference.**

Program :

/* Write a program to print first n numbers using goto statement */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  n;
    int  i=1;
    clrscr();
    printf("Give  Integer  number:    ");
    scanf("%d",&n);
    loop:          /*  identifier  for  goto  */
    if(i<=n)
        {
        printf(  "number  =  %d\n",i);
        i++;
        goto  loop;  /*  goto  here.  backward  reference */
        }
}
```

Output :

```
Give  Integer  number:    4
number  =  1
number  =  2
number  =  3
number  =  4
```

Program :

/* Write a program to print sum of first n integer numbers using goto statement */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int  n,sum=0;
    int  i=1;
    clrscr();
    printf("Give  Integer  number:    ");
    scanf("%d",&n);
    loop:          /*  identifier  for  goto  */
    if(i<=n)
        {
```

```
        sum = sum +i;
        i++;
        goto loop; /* goto here. backward reference */
        }
    printf("Sum of first %d integers = %d\n",n,sum);
}
```

Output :

```
Give Integer number:    9
Sum of first 9 integers = 45
```

6.10 SOLVED PROGRAMMING EXAMPLES :

Program :

/* Write a program that accepts an age value and checks whether the person is eligible for voting not. */

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int age;
    clrscr();
    printf("Give age value\n");
    scanf("%d",&age);
    if(age >= 18)
        printf("Person is eligible to vote\n");
    else
        printf("Person is not eligible to vote\n");
}
```

Output-1 :

```
Give age value
32
Person is eligible to vote
```

Output-2 :

```
Give age value
15
Person is not eligible to vote
```

Program :

/* Write a program to find the roots of quadratic equation $ax^2 + bx + c = 0$.
if $b^2 - 4ac = 0$, only one root,
if $b^2 - 4ac > 0$, two roots,
if $b^2 - 4ac < 0$, no roots. formula is $x = -b \pm \sqrt{b^2 - 4ac}/(2a)$*/

```
#include  <stdio.h>
#include  <conio.h>
#include<math.h>  /*  sqrt()  function  used.  sqrt()
                finds  square  root  of    number*/
main()
{
float  a,b,c,d;
float  r1,r2,img;  /*  img  stores  imaginary  part
                if  b*b-4*a*c  <0  */
clrscr();
printf("Give  values  of  co-efficients  a,b  and  c\n");
scanf("%f%f%f",  &a,&b,&c);
if  ((a  ==  0)  &&  (b  !=0))    /*  If  a=0  then  equation
                    is  bx  +c  =0  so,  x  =  -c/b  */
    {
    printf("Single  root  =  %f\n",  -c/b);
    }
else
    {
    d  =  b*b  -  4*a*c;  /*  calculate  d  */
    if  (d>0)              /*  real  and  distinct  roots */
        {
        r1  =  (-b  +  sqrt  (d))/(2*a);
        r2  =  (-b  -  sqrt  (d))/(2*a);
        printf("Real  and  distinct  roots  are:  %7.2f  %7.2f\n",r1,r2);
        }
    else if( d==0) /* real and equal i.e only one  root */
        {
        r1  =  -b/(2*a);
        printf("Real  and  equal  roots  are:  %7.2f  %7.2f\n",r1,r1);
        }
    else                    /*  imaginary  roots  */
        {
        r1  =  -b/(2*a);
```

```
            img = sqrt(-d)/(2*a);
                /* sqrt of negative not defined */
            printf("Imaginary roots\n");
            printf("Roots are: %7.2f + %7.2f\n",r1,img);
            printf("Roots are: %7.2f - %7.2f\n",r1,img);
            }
        }
}
```

Output-1 :

```
Give values of co-efficients a,b and c
1 5 6
Real and distinct roots are: -2.00 -3.00
```

Output-2 :

```
Give values of co-efficients a,b and c
4 2 6
Imaginary roots
Roots are: -0.25 + 1.20
Roots are: -0.25 - 1.20
```

Output-3 :

```
Give values of co-efficients a,b and c
0 3 5
Single root = -1.666667
```

Output-4 :

```
Give values of co-efficients a,b and c
3 6 3
Real and equal roots are: -1.00 -1.00
```

Program :

/ Write a program to calculate total salary and net salary of an employee. net salary = total salary - income tax, total salary = basic + da + hra +ta da = 50% of basic

Below mentioned table shows how da, hra and ta can be calculated from basic

Basic	hra	ta
upto 6000	500	200
from 6001 to 10000	1500	400
from 10001 to 20000	2500	800
20001 & above	3500	1600

Income tax rate

Total salary	Income tax rate (%)
upto 10000	nil
from 10001 to 20000	10%
20001 & above	20%

Calculate total salary, income tax and net salary. */

```
#include<stdio.h>
#include  <conio.h>
main()
{
    float  basic,  net,  tot,hra,ta  ,da,itax;
        /*  variable          represents
            basic             Basic  Salary
            net               Net  Salary
            tot               Total  Salary
            itax              Income  Tax
            da                Dearness  Allowance
            hra               House  Rent  Allowance
            ta                Transport  Allowance
        */
    clrscr();
    printf("Enter  Basic  salary  of  an  employee\n");
    scanf("%f",&basic);
    da  =  basic  *0.5;      /*  da  calculation  */
    if  (basic  <=  6000)
                /*  basic  less  than  or  equal  6000  */
        {
        hra  =  500;
        ta  =  200;
        }
    else  if  (  basic  >=6001  &&  basic  <=10000)
            /*  basic  more  than  6000  but  less  than
                or  equal  to  10000  */
        {
        hra  =  1500;
        ta  =  400;
        }
      else  if  (  basic  >=10001  &&  basic  <=18000)
            /*  basic  more  than  10000  but  less
                than  or  equal  to  18000  */
```

```
        {
        hra  =  2500;
        ta  =    800;
        }
      else  if  ( basic  >=18001)
            /*  basic  more  than  18000  */
        {
        hra  =  3500;
        ta  =  1600;
        }
      tot  =  basic  +  da  +hra  +ta;
           /*  total  salary  calculated  */
     if  (tot  <=  10000)
       itax  =0;
   else  if(tot  >=  10001  &&  tot  <=20000)
                  /*  10%  income  tax  rate  */
       itax  =  tot  *0.1;
   else
       itax  =  tot  *0.2;    /*  20%  income  tax  rate */
   net  =  tot-  itax;      /*  net  salary  */
   printf("==============================\n");
   printf("BASIC          =     %8.2f\n",basic);
   printf("HRA            =     %8.2f\n",hra);
   printf("TA             =   %8.2f\n",ta);
   printf("DA             =   %8.2f\n",da);
   printf("==============================\n");
   printf("Total  Salary  =  %8.2f\n",tot);
   printf("Income  Tax    =  %8.2f\n",itax);
   printf("==============================\n");
   printf("Net  Salary    =  %8.2f\n",net);
}
```

Output-1 :

```
Enter  Basic  salary  of  an  employee
10000
==============================
BASIC              =  10000.00
HRA                =   1500.00
```

```
TA                    =    400.00
DA                    =   5000.00
==============================
Total  Salary         =  16900.00
Income  Tax           =   1690.00
==============================
Net  Salary           =  15210.00
```

Output-2 :

```
Enter  Basic  salary  of  an  employee
5000
==============================
BASIC                 =   5000.00
HRA                   =    500.00
TA                    =    200.00
DA                    =   2500.00
==============================
Total  Salary         =   8200.00
Income  Tax           =      0.00
==============================
Net  Salary           =   8200.00
```

Program :

```
/* Write a program to check the entered character is vowel or not */
#include<stdio.h>
#include  <conio.h>
main()
{
    char  ch;
    clrscr();
    printf("Enter  one  character\n");
    scanf("%c",&ch);
    if( ((ch  >=  'a')  &&  (ch  <='z'))  ||  ((ch>='A')  &&  (ch<='Z')))
    {
    switch(ch)
        {
        case  'a':   case  'A':
        case  'e':   case  'E':
        case  'i':   case  'I':
        case  'o':   case  'O':
```

```
        case 'u':   case 'U':
            printf("Entered character %c is vowel\n",ch);
            break;
        default:
            printf("Entered character %c is not vowel\n");
        }
    }
    else
        printf("Entered character %c is not from alphabet\n",ch);
}
```

Output-1 :

```
Enter one character
e
Entered character e is vowel
```

Output-2 :

```
Enter one character
I
Entered character I is vowel
```

Output-3 :

```
Enter one character
?
Entered character ? is not from alphabet
```

Program :

/* Write a menu driven program to perform the arithmetic operations +, -, *, / and % */

```
#include<stdio.h>
#include<conio.h>
main()
{
    int choice,x,y;
    float ans;
    clrscr();
    printf("Menu of operations\n");
    printf("==================\n");
    printf("1   Addition\n");
    printf("2   Subtraction\n");
    printf("3   Multiplication\n");
    printf("4   Division\n");
    printf("5   Modulo Division (Remainder)\n");
```

```
    printf("====================\n");
    printf("Give your choice:    ");
    scanf("%d",&choice);
    printf("Give two numbers\n");
    scanf("%d%d",&x,&y);
    switch(choice)
        {
        case 1:
            ans = x +y;
            break;
        case 2:
            ans = x -y;
            break;
        case 3:
            ans= x *y;
            break;
        case 4:
            if (y==0)
                printf("division not possible\n");
            else
                ans = x /y;
            break;
        case 5:
            ans = x%y;
            break;
        default:
            printf("Wrong choice\n");
            break;
        }
    if ((choice != 4) && (y != 0)) printf("Answer = %7.2f\n",ans);
}
```

Output-1 :

```
Menu of operations
====================
1   Addition
2   Subtraction
3   Multiplication
4   Division
5   Modulo Division  (Remainder)
====================
Give your choice:3
Give two numbers
1
```

```
2
Answer  =  2.00
```

Output-2 :

```
Menu  of  operations
====================
1   Addition
2   Subtraction
3   Multiplication
4   Division
5   Modulo  Division  (Remainder)
====================
Give  your  choice:
4
Give  two  numbers
25  0
division  not  possible
```

Output-3 :

```
Menu  of  operations
====================
1   Addition
2   Subtraction
3   Multiplication
4   Division
5   Modulo  Division  (Remainder)
====================
Give  your  choice:5
Give  two  numbers
26
5
Answer  =  1.00
```

: SUMMARY :

- **Decision making structures** are useful when flow is not sequential. The action to be taken depends on certain conditions. The decision making structures are **if statement, if..else statement, else–if ladder, switch statement, goto statement** and **break statement.**
- In **If statement**, If the condition is true then the statements are executed, otherwise they are not executed.
- **If..else statement** is executed if we want to execute one set of statements from available two sets of statements.
- ***Nested if statement*** checks more than one conditions, where in if statement is used inside another if statement. *The nesting can be up to any level.*

- **If..else..if ladder** is also known as multiple if..else statement.
- For complex problems, **Multiple if..else statement** is difficult to manage, a better alternative of multiple if..else statement is to use **switch** statement. In switch managing the code is easy.
- **break** statement breaks the control flow from the current block. It causes exit from the switch body if used inside the switch statement.
- **default** is normally used inside switch statement. Statements written in that part get executed if any of the previous cases do not match. Normally, it is written after all the cases in switch statement.
- **goto** statement is an unconditional branching statement. It becomes difficult to understand and debug program if we use goto statement. Structured programming practice do not encourage use of goto in programming.

: MCQs :

1. In If statement it is compulsory to write the condition within brackets
   (a) < > (b) ( ) (c) { } (d) All of above
2. Switch statement is an alternative to
   (a) If statement (b) if..else statement
   (c) multiple..if else statement (d) None of above
3. In switch statement, the value of variable or expression must be
   (a) Integer (b) character
   (c) Can be either int or char (d) Can be any value
4. In switch statement, it is written after every case
   (a) break (b) continue (c) goto (d) default
5. Branching statement goto is a
   (a) conditional (b) unconditional (c) Both a and b (d) None of above
6. In the code fragment shown below, which type of reference is used?

```
label : statement;
.
.
goto label;
```

   (a) Full reference (b) partial reference
   (c) Forward reference (d) Backward reference
7. To use sqrt() function we need to include header file
   (a) stdio.h (b) conio.h (c) string.h (d) None of above
8. Which statement is used at the end of all cases in switch statement?
   (a) default (b) break (c) continue (d) goto
9. Which statement does not belong to decision making structures?
   (a) if (b) default (c) switch (d) break
10. Which among the following is a unconditional control structure.
    (a) goto (b) for (c) do–while (d) if–else

: ANSWERS :

1. (b) **2.** (c) **3.** (c) **4.** (a) **5.** (b) **6.** (d) **7.** (d)
8. (a) **9.** (b) **10.** (a)

: EXERCISES :

1. Explain the syntax of multiple if...else statement.
2. Explain switch statement with example.
3. How switch statement and multiple if...else statement differ?

4. If we forget to write break statement between two cases in switch statement, what will be its effe

5. What will be the output of following?

(i)

```
if  (x>0  &&  x<10)
        printf("Single  digit  number\n");
    else  if(  x>10  &&  x<100)
        printf("Double  digit  number\n");
    else
        printf("More  than  two  digit  number\n");
```

(ii)

```
if(choice  ==  1)
        printf("Addition\n");
    else  if(choice  ==  2)
        printf("Subtraction\n");
    else  if(choice  ==  3)
        printf("Multiplication\n");
    else  if(choice  ==  4)
        printf("Division\n");
    else
        printf("Wrong  Choice\n");
```

What will the output if choice = 2 and 4?

6. Write following code using ternary? Operator.

```
if(  a>10)
    if(a<20)
        cost  =  100;
    else
        cost  =150;
else
    cost  =  50;
```

7. What will be the output of following code for x = 5 and x=10?

```
a=  (x  %2)?  1:  0;
switch(  a)
{
case  0:
    printf("Even  Number\n");
case  1:
    printf("Odd  Number\n");
}
```

8. Write a program to find minimum of three numbers.

: Answers to Selected Exercises :

6. Write following code using ternary? Operator.

```
if(  a>10)
    if(a<20)
        cost  =  100;
```

```
        else
            cost  =150;
    else
        cost  =  50;
```

Ans :

```
cost=(a>10)?((a<20)?100:150):50
```

7. What will be the output of following code for x = 5 and x=10?

```
a=  (x  %2)?  1:  0;
switch(  a)
{
case  0:
    printf("Even  Number\n");
case  1:
    printf("Odd  Number\n");
}
```

Ans:

For x=5, output is Even number.

For x=10, output is Odd number.

: SHORT QUESTIONS :

1. **List various control structures in 'C'.**

⇒ Control structures in 'C' are: if statement, if...else statements, else-if ladder, switch statement, goto statement and break statement.

2. **What type of value an expression in switch statement must have?**

⇒ The value of expression or variable written after the switch statement must be either character or integer value.

3. **Is default statement compulsory in switch statement?**

⇒ No, it is not compulsory. It is optional and written after all cases so that it executes if no cases match.

4. **The goto statement is which type of branching statement?**

⇒ The goto statement is an unconditional branching statement. Structured programming does not encourage use of goto statement.

5. **Which header file should be included to use sqrt() function?**

⇒ Header file math.h.

6. **In header files whether functions are declared or defined?**

⇒ Functions are declared within header file. Function prototypes exist in a header file, not function bodies. They are defined in library (lib).

❖

CHAPTER

10

Functions

10.1 INTRODUCTION

10.2 USER DEFINED FUNCTIONS

10.3 LIBRARY FUNCTIONS

10.4 HOW THE CONTROL FLOW TAKES PLACE IN MULTI-FUNCTION PROGRAM ?

10.5 FUNCTION DECLARATION

10.6 FUNCTION DEFINITION

10.7 CATEGORY OF FUNCTIONS

10.8 SCOPE OF VARIABLES

10.9 PARAMETER PASSING TO FUNCTION

10.11 PARAMETERS AS ARRAY

10.12 PARAMETERS AS STRING

10.13 FUNCTION RETURNING A POINTER

10.14 STORAGE CLASSES

10.15 SOLVED PROGRAMMING EXAMPLES

- SUMMARY
- MCQs
- EXERCISES AND ANSWERS TO SELECTED EXERCISES
- SHORT QUESTIONS

10.1 INTRODUCTION :

Every 'C' program is a collection of functions. In every 'C' program, main() function must be there, because the execution of program starts from main() function. **Function is a group of statements as a single unit known by some name which is performing some well defined task.** The functions which are created by programmer in a program are called as **user-defined functions,** while functions which are readily available in library are called as **library functions or built-in functions.**

Advantages of functions :

- It facilitates top-down modular programming, whereby the large problem is divided into small parts and each small part is handled later in depth.
- It reduces the size of code, if that code is repetitively used in program.
- Function can be used in other program also, so labor is reduced.
- Function can call itself, which is called as recursion, it reduces the coding, some problems are recursive in nature, at that place recursion can be used.
- It allows team-work for large project, individual members of team has to concentrate on the function given to them and not on the whole complex problem.

10.2 USER DEFINED FUNCTIONS :

User defined functions are created for doing some specific task in a program. The code is written as one unit having a name. As the size of program increases, it becomes difficult to understand, debug and maintain it. If we write the code in main() function only. When some portion of the code is repeated in the program and when there is a definite purpose of the code, we can write that part of the code as a function. So, use of function reduces the size and complexity of the program. The code of the function is written only once in the program and called for any number of times from the program whenever required.

10.3 LIBRARY FUNCTIONS :

These are the functions which are readily available and placed inside the library. Their prototypes are declared in various header files depending on the category to which they belong to. For example, string processing functions are declared in **string.h** header file, mathematical functions are declared in **math.h** header file. We need to understand the prototype of built-in function and include that header file in our program.

10.4 HOW THE CONTROL FLOW TAKES PLACE IN MULTI-FUNCTION PROGRAM ?

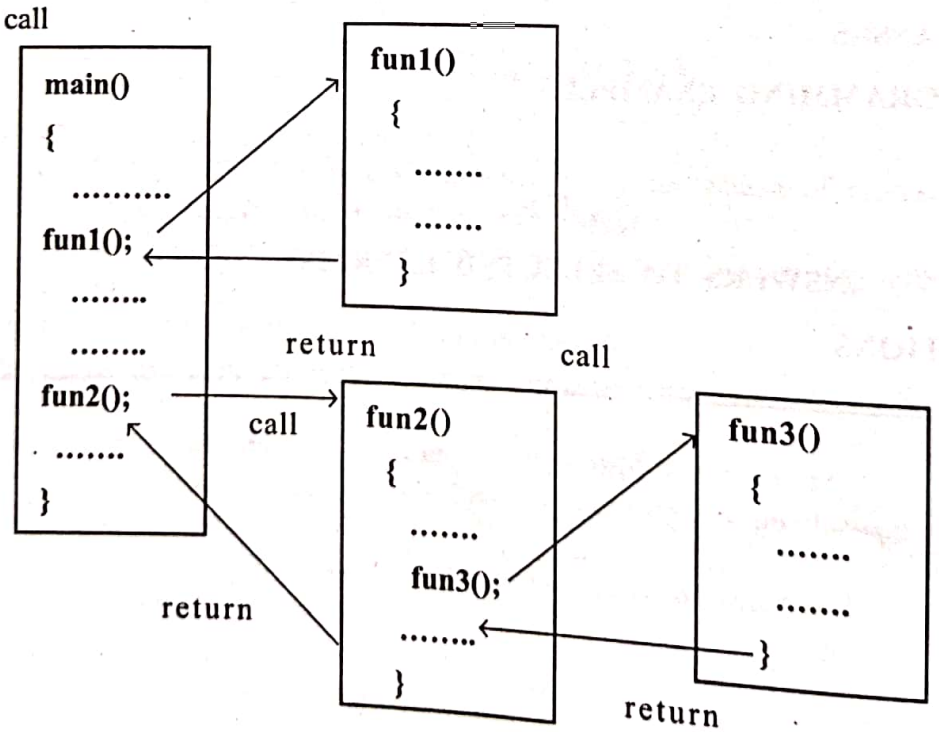


Figure 10.1

In figure 10.1, the call and return from function is explained by arrows in different directions. Above figure shows the program with main() function and three user-defined functions fun1(), fun2() and fun3(). From main() function first fun1() function is called, so code of fun1() function is executed, and control goes to the next statement after call to fun1() in main() function. Again fun2() function is called form main() function, so code of fun2() function is executed, from fun2() function internally fun3() function is called, so code of fun3() function is executed, then control returns back to next statement in fun2() statement, when last statement of function fun2() is executed, control goes back to next statement after call to fun2() in main() function.

In figure, main() function calls fun2() function which internally calls another function fun3(). When function fun3() is over, control returns to the function which called it i,e. function fun2(), when function fun2() is over, control returns back to main() function. **This is called as nesting of functions**. The nesting can be done up to any level.

With function, three things are related: declaration, definition and call. As applicable to variables, all the user defined functions must be declared before their use in the program.

FUNCTION DECLARATION :

The syntax for declaring function is:

```
type function_name(argument(s));
```

Here, type specifies the type of value returned, function_name specifies name of user-defined function, and in bracket argument(s) specifies the arguments supplied to the function as comma separated list of variables. If there are no arguments, then the bracket is left empty. The declaration of a function is terminated by semicolon (;) symbol.

The declaration is also called as prototype of a function.

Example, int sum(int x, int y); takes two integers and returns an integer

void printmessage(); takes no arguments and no return value

FUNCTION DEFINITION :

The syntax for definition of function is :

```
type function_name(argument(s))
    {
       statement(s);
    }
```

Here, type, function_name and argument(s) have same meaning as before. The statement(s) within { } symbols indicate the body of a function. Here, the actual logic of the work of function is implemented.

If the return type is not mentioned, then by default it is taken as **int.** We can not define function inside a function. We can call one function from other. If the function return type is other than void, there must be **return** statement in the function. The returned value can be put in brackets but it is not compulsory i.e following are valid return statements.

```
return; if the return type is void.
return x;     value of x returned.
return  (x); value of x returned.
```

Program :

```
/* Demo of function in program */
#include <stdio.h>
#include <coino.h>
void printmessage();    /* declaration */
main()
{
  clrscr();
  printmessage();              /* call */
  getch();
}
void printmessage()          /* definition */
    {
    printf("Hello!!\n");
    }
```

Output :

```
Hello!!
```

Explanation :

In above program, the line

```
void printmessage();
```

is declaration of function printmessage, which returns no value(void), and do not require any arguments, so emp bracket (). The line is terminated by ; symbol. The above line tells the compiler that somewhere in the progra printmessage will be encountered, which is a name of function, returning void.

The code,

```
void printmessage()
{
printf("Hello!!\n");
}
```

is the definition of function. In the body of function only one statement printf() is written. When the abo function is called, **Hello!!** Message will be printed on screen. **Note that the definition of function is outsi the main() function. There is no semicolon (;) after } symbol.**

The line in main() function,

```
printmessage();
```

is a call to the printmessage() function, so the statements in the body of function will be executed.

In above program, declaration of printmessage() function is needed. It is not necessary to write the declaratio in all programs. Declaration is needed if the function definition is written after the call to the function i.e if th function definition is written before the call to the function, then declaration of the function is not required. Th above program can be rewritten without using declaration of function as shown below.

Program :

```
/* Demo of function without declaration */
#include <stdio.h>
#include <conio.h>
   /* declaration of function not needed */
void printmessage()
                                   /* definition */
   {
   printf("Hello!!\n");
   }
main()
{
clrscr();
printmessage();                    /* call */
getch();
}
```

Output :

```
   Hello!!
```

Explanation :

In above program, the only change is that the definition of function is written before main() function. Because of this, declaration is not required. New sequence is definition first and then call of function in main(). **Note that in the program, if the definition comes first, and then the call to the function is made, then declaration is not needed.**

10.7 CATEGORY OF FUNCTIONS :

The functions in 'C' language can be divided in following categories :

1. Functions with no arguments and no return value
2. Functions with arguments and no return value
3. Functions with arguments and with return value.

The function example we have seen in previous programs belong to first category i.e no arguments and no return value. When we have function which takes arguments, in that case we need to understand the formal parameters and actual parameters.

The variables or parameters used in definition of function are called as formal parameters. While, parameters used in the call to the functions are called as actual parameters.

Following program shows the function with arguments but no return value.

void sum(int a, int b); declaration says that sum function takes two values but does not return any value.

Program :

/* Write a program to sum given two integer numbers using function */

```
#include <stdio.h>
#include <conio.h>
void sum(int a, int b);        /*declaration */
main()
{
int i, j;
clrscr();
printf("Give two integer numbers\n");
scanf("%d %d",&i,&j);
sum(i,j);     /* call sum */
}
void sum(int a,int b)        /* definition */
{
    int c;
    c = a+b;
    printf("Sum of %d and %d = %d\n",a,b,c);
}
```

Output :

```
Give two integer numbers
2
3
Sum of 2 and 3 = 5
```

For example, in following program, the line

```
int sum(int a, int b);
```

is declaration of **sum** function. Here, **a and b are formal parameters.** Formal parameters are variables whi are used in the body of the function and in prototype.

When a call to the function is made in main() function, the actual parameters are passed. For our example, t line

```
ans = sum(i,j);
```

calls the function **ans**, with variable i and j. **The variables i and j work as actual parameters.** Internally, t value of i is assigned to a, and that of j is assigned to b. The return type of sum function is int. So, return val of function which is int is stored in variable sum using = operator.

We can not call the function simply as

```
sum(i,j);
```

because the function returns integer.

Whenever, a function having arguments is used, the formal parameters and actual parameters should match

type and number. If there is a type mismatch or size mismatch, then the compiler will generate an error message.

Above concepts are used in following program, **which shows the function which take arguments and also return value.**

Program :

```
/* Write a program to sum given two integer numbers using function */
#include <stdio.h>
#include <conio.h>
int sum(int a, int b);          /*declaration */
main()
{
int i, j;
int ans;
clrscr();
printf("Give two integer numbers\n");
scanf("%d %d",&i,&j);
ans = sum(i,j); /* call sum. returns integer so its
            return value stored in ans which is int*/
printf("Sum of %d and %d using function = %d\n",i,j,ans);
}
int sum(int a,int b)          /* definition */
{
   return a+b;                /* return integer value */
}
```

Output :

```
Give two integer numbers
34
56
Sum of 34 and 56 using function = 90
```

Following program explains how the user defined function can be useful if it is used inside a loop.

Program :

/* Write a program to find largest of three numbers using function. Ask the user to continue or not. */

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b, int c);  /* prototype */
main()
{
int i, j,k;
int ans;
char ch='y';          /* choice initialized to 'y'*/
clrscr();
while (ch == 'y')                /* loop */
    {
    printf("Give three integer numbers\n");
    scanf("%d %d %d",&i,&j,&k);
    ans = max(i,j,k);          /* call function */
    printf("Largest of %d, %d and %d = %d\n",i,j,k,ans);
    fflush(stdin);          /* clear input buffer */
    printf("Do you want to continue?(y/n)\n");
    scanf("%c",&ch);          /* get choice */
    }
}
int max(int a,int b,int c)  /* max defined here */
    {
        int large;
        large = a;
        if( b> large)
            large =b;
        if (c> large)
            large =c;
        return large;   /* return largest number */
    }
```

Output :

```
Give three integer numbers
23
11
56
Largest of 23, 11 and 56 = 56
Do you want to continue?(y/n)
y
Give three integer numbers
```

```
43
55
24
Largest of 43, 55 and 24 = 55
Do you want to continue?(y/n)
n
```

Explanation :

In above program, max function is user defined function. It is used inside the while loop. While loop executes till ch='y'. Here, i, j & k are actual parameters passed to function when called, while a, b & c are formal parameters. The function finds out the largest number and returns that value. The statement

```
ans = max(i,j,k);
```

stores the returned value in ans variable. The function fflush(stdin) clears the input buffer so that character reading can be done properly.

ogram :

/* Write a program using function which receives number as argument and return sum of digits of that number */

```
#include <stdio.h>
int sum_digits(int num)   /* function which takes number as input and
return number of digits in that number */
{
        int sum =0;
        while (num >0)
             {
             sum = sum + num %10;
             num = num /10;
             }
        return sum;
}
main()
{
printf("\n%d\n",sum_digits(123));   /* num =123 */
printf("%d\n",sum_digits(4523));    /* num = 4523 */
}
```

utput :

```
6
14
```

Program :

/* Write a program to check leap year using function. */

```
#include <stdio.h>
#include <conio.h>
char check_leap(int year);
                /* prototype return. type character*/
main()
{
int year;
clrscr();
printf("Enter year\n");
scanf("%d",&year);
if(check_leap(year) == 't')  /*call */
        printf("Year %d is leap year\n",year);
else
        printf("Year %d is not leap year\n",year);
}
char check_leap(int year)    /* definition */
    {
    if((year % 4 == 0) && (year %100 !=0) || (year %400 ==0))
            return 't';
        else
            return 'f';
    }
```

Output-1 :

```
Enter year
2008
Year 2008 is leap year
```

Output-2 :

```
Enter year
2009
Year 2009 is not leap year
```

Program :

```
/* Program to find HCF of given Numbers without Recursion */
#include <stdio.h>
int hcf(int, int);
int main()
{
    int a, b, result;
    printf("Enter two numbers to find their HCF: ");
    scanf("%d%d", &a, &b);
    result = hcf(a, b);
    printf("The HCF of %d and %d is %d.\n", a, b, result);
}
int hcf(int a, int b)
{
    while (a != b)
    {
        if (a > b)
        {
            a = a - b;
        }
        else
        {
            b = b - a;
        }
    }
    return a;
}
```

Output :

```
Enter two numbers to find their HCF: 4 6
The HCF of 4 and 6 is 2.
```

10.8 SCOPE OF VARIABLES :

By scope of a variable, we mean in what part of the program the variable is accessible. In what part of the program the variable is accessible is dependent on where the variable is declared. There are two types of scope - **Local** and **Global**.

Local variables :

The variables which are declared inside the body of any function are called as local variables for that function. These variables can not be accessed outside the function in which they are declared. Thus, local variables are internal to the function in which they are defined. So far we have used all the variables as local because all the variables were declared inside of either main() function or inside user defined function.

Global variables :

The variables which are declared outside any function definition is called as global variable. These variables are accessible by all the functions in that program, i.e all the functions in program can share global variable.

Difference between local variables and global variables :

Following table explains the differences.

Local variables	Global variables
Declared inside function body	Declared outside function body
Not initialized automatically	Initialized automatically by value 0
Use of local variables advisable	Too much use of global variables make program difficult to debug, so use with care

Following program explains the concept of local and global variables.

Program :

```
/* Write a program to sum given series using function
1 + x + x^2 + x^3 + x^4 +....+x^n */
#include <stdio.h>
#include <conio.h>
long int sum =1, term; /* global variables. so can be
            used inside all functions */
void cal_term(int x, int p)
    {
    int i;  /* local variable. its value is
        available inside this function only*/
term =1;    /* global variable accessed */
for  (i=1;i<=p;i++)
    term = term *x;
}
void series(int x, int n)
{
int i;  /* local variable. its value is
        available inside this function only.
        It is  different from variable i in
        function cal_term */
for(i=2;  i<=n;i++)
    {
    cal_term(x,i);    /* call function with
                      values of x and i */
    sum =sum +term; /* global variables accessed */
```

```
    }
printf("Answer=%ld\n",sum);
}
main()
{
int x,n;     /* local variables */
clrscr();
printf("Give value of x and n \n");
scanf("%d  %d",&x,&n);
series(x,n);
       /*call function with values of x and n */
}
```

Output :

```
Give value of x and n
2 4
Answer=29
```

10.9 PARAMETER PASSING TO FUNCTION :

When a function is called from other function, the parameters can be passed in two ways.

- Call by value
- Call by reference

In call by value, argument values are passed to the function, the contents of actual parameters are copied into the formal parameters. The called function can not change the values of the variables which are passed. Even if a function tries to change the values of passed parameters, those changes will occur in formal parameters, not in actual parameters.

Example :

In all the programs which have seen so far, use call by value, where we pass the value of actual parameter. In previous program the call

```
series(x,n);
```

and

```
cal_term(x,i);
```

are examples of call by value.

In call by reference, as the name suggests, the reference of the parameter **i.e address (pointer) is passed** to the function and not the value. Because pointer is passed, the called function can change the content of the actual parameter passed. **Call by reference is used whenever we want to change the value of local variables declared in one function to be changed by other function.**

Program :

/* Write a program to exchange the value of two variables using function*/

```
#include <stdio.h>
#include <conio.h>
void exch(int a, int b)
{       /* a and b formal parameters */
    int temp;
    temp=a;
    a=b;
    b=temp;
    printf("Printing from inside exch function\n");
    printf("a = %d b = %d\n",a,b);
                    /* local copies exchanged */
}
main()
{
    int x,y;     /* local variables */
    clrscr();
    printf("Give value of x and y \n");
    scanf("%d %d",&x,&y);
    exch(x,y);
        /*call by value. x and y actual parameters*/
    printf("Printing from main function\n");
    printf("x = %d y = %d\n",x,y);
                    /* x and y remain unchanged */
}
```

Output :

```
Give value of x and y
3
5
Printing from inside exch function
a = 5 b = 3
Printing from main function
x = 3 y =5
```

Above program is not working as per our expectation, the values of x and y which are declared inside the main function are not changed. The exch() function logic is correct, and it is able to exchange the values of a and

b inside function. The problem is that we are trying to change the values of variables x and y (which are local to main() function) by outside function exch() function using call by value, which is not possible.

We should use call by reference to get the correct answer i.e exch() function definition and call to exch() function should be modified to use call by reference. In call by reference, we pass the address of the variable, and the formal parameters should be declared as pointers of appropriate type.

So, call to exch() function should be written as

```
exch( &x, &y);
```

& symbol specifies address of

and the prototype of exch() function should be written as

```
exch(int *a, int *b)
```

so, **call to exch() function assigns address of x to variable a and address of y to variable b.** Now, the exch() function has the address of actual parameters, so it is able to change the values of actual parameters. It is shown in following program.

Program :

/* Write a program to exchange the value of two variables using function */

```
#include <stdio.h>
#include <conio.h>
void exch(int *a, int *b)
{   /* a and b formal parameters which are pointers*/
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
    printf("Printing from inside exch function\n");
    printf("a = %d b = %d\n",*a,*b);
                        /* local copies exchanged */
}
main()
{
    int x,y;     /* local variables */
    clrscr();
    printf("Give value of x and y \n");
    scanf("%d %d",&x,&y);
exch(&x,&y);
 /*call by reference. x and y actual parameters*/
Printf("Printing from main function\n");
Printf("x = %d y = %d\n",x,y);
                            /* x and y are changed */
}
```

Output :

```
Give value of x and y
3
5
Printing from inside exch function
a = 5 b = 3
Printing from main function
x = 5 y = 3
```

If we execute following code,

```
void  main()
{
    int  x[]  =  {1,2,3,4,5};
    printf("%d",fun(&x[2]));
}
int  fun(int  *a)
{
    int  i=0,sum=0;
    for(i=0;i<2;i++,a++)
        sum  +=  *a;
    return(sum);
}
```

In above program, main() function calls function fun() with call by pointer value.

Statement	Effect
int x[] = {1,2,3,4,5};	x[0]= 1, x[1]=2 … x[4]=4
printf("%d",fun(&x[2]));	Call to function fun with address of x[2], so *a = x[2]=3
int i=0,sum=0;	i=0, sum=0
for(i=0;i<2;i++,a++)	sum += *a; sum = sum + x[2] so, sum = 0+3 =3 sum = sum + x[3] so, sum = 3 +4 =7
return(sum);	Return value 7 to main() which is printed by main() function

So, output

```
7
```

10.10 RECURSION :

Sometimes, the function is defined in terms of itself. For example factorial of a number, we can say that N! = N * (N-1)!. Here, factorial of N is defined in terms factorial of N-1.

Again (N-1)! = N-1 * (N-2)! and so on. We can write the function which calls itself internally. **Thus, recursion is the process by which a function calls itself. Because the function calls itself, there must be some condition to stop recursion; otherwise it will lead to infinite loop.** The condition to stop recursion is called as terminating condition. For above example of factorial, the terminating condition is 0! = 1. Thus,

N! = N * (N-1) !

0! = 1 terminating condition.

Types of recursion :

There are two types of recursion based on how recursion takes place: Direct recursion and indirect recursion. In the case of direct recursion, a function explicitly calls itself. While in the case of indirect recursion, the function calls another function, which ultimately calls the caller function.

Example of indirect recursion,

fun1()
{
 	
....
fun2()
...
 	
}
	fun2()
	{
	
	...
	fun1()
	...
	
	:

ADVANTAGES OF RECURSION :

- Easy solution for recursively defined problems.
- Complex programs can be easily written in less code.

DISADVANTAGES OF RECURSION :

- Recursive code is difficult to understand and debug.
- Terminating condition is must, otherwise it will go in infinite loop.
- Execution speed decreases because of function call and return activity many times.

Program :

/* Write a program to find factorial of a given number using recursion */

```
#include <stdio.h>
#include <conio.h>
long int fact(int n);      /* prototype of function */
main()
{
int m;
long int ans;       /* long int store large number */
clrscr();
printf("Give the number\n");
scanf("%d",&m);
ans = fact(m);
      /* call function fact() with call by value*/
```

```
printf("Factorial of %d using function = %ld\n",m,ans);
}
long int fact(int n)   /* function body here */
{
    if (n==1)   /* condition to terminate recursion */
        return 1;
    else
        return  n* fact(n-1); /* recursion here. fact()
                              calls fact() internally*/
}
```

Output :

```
Give the number
6
Factorial of 6 using function = 720
```

Program :

/* Write a program to calculate nCr using function
nCr . = n! / (r ! * (n-r) ! */

```
#include <stdio.h>
#include <conio.h>
long int fact(int n);
main()
{
int n,r;
long int ans;
clrscr();
printf("Give the values of n and r \n");
scanf("%d%d",&n,&r);
if ( n<0 || r <0 || n<r)
    printf("Input data invalid\n");
else
{
    ans = fact(n) / (fact(r) * fact(n-r));
    printf("Answer = %ld\n",ans);
}
}
```

```
long int fact(int n)
{
if (n==0)   /* condition to terminate recursion */
   return 1;
else
   return  n* fact(n-1); /* recursion here. fact()
                         calls fact() internally*/
}
```

Output-1 :

```
Give the values of n and r
6 6
Answer = 1
```

Output-2 :

```
Give the values of n and r
10 7
Answer = 120
```

Similarly, we can calculate x^y using recursion. We can write

$$x^y = x * x^{(y-1)}$$

$x^0 = 1$, terminating condition i.e y =1.

Program :

```
/* Sum of digits using Recursive function*/
#include<conio.h>
#include<stdio.h>
int  sumofdigit(int  x);
void  main()
{
    int  n,sum;
    printf  ("Enter  Integer  number  \n");
    scanf("%d",&n);
    sum=sumofdigit(n);
    printf  ("Sum  of  Digits  %d\n",  sum);
    getch();
}
    /*  Recursive  Function  sum  of  digits  */
    int  sumofdigit(int  x)
    {
    int  s=0,d;
```

```
    if(x  ==  0)
        return  (0);
    d=  x%10;
    s=  s  +  d  +  sumofdigit(x/10);
    return(s);
}
```

Output :

```
Enter  Integer  number
345
Sum  of  Digits  12
```

Program :

/*Write a program to calculate x^y using recursive function */

```
#include <stdio.h>
#include <conio.h>
int power(int a, int b); /* prototype of function */
main()
{
int x,y;
int ans;
clrscr();
printf("Give the numbers x and y\n");
scanf("%d%d",&x,  &y);
ans = power(x,y);        /* call function */
printf("%d^%d = %d\n",x,y,ans);
}
int power(int a, int b)  /* function body here */
{
        if (b==0)/* condition to terminate recursion */
        return 1;
    else
        a * power(a,b-1);/* recursion here. */
}
```

Output :

```
Give the numbers x and y
2 5
2^5 = 32
```

Similarly, we can write a function for generating Fibonacci numbers recursively. Because Fibonacci series is recursive in nature, current number is generated by adding previous two Fibonacci numbers. We can pass the two previous numbers as an argument as well as the number of Fibonacci numbers remaining to be generated. Here, the terminating condition will be the number of Fibonacci numbers yet to be generated becoming zero. i.e function prototype will be

```
fibo(int n1,int n2,int n);
```

where, n1,n2 are previous fibonacci numbers and n is the numbers yet to be generated. As long as n > 0, fibo() function will call itself, when n=0, recursion terminates.

Program :

```
/*Program to calculate Highest Common Factor (HCF) of three integers */
#include<stdio.h>
#include<conio.h>
#include<math.h>

int  hcf(int  x,  int  y);
void  main()
{
    int  a,b,c,h;
    clrscr();
    printf("  ENTER  3  integers  \n");
    scanf("%d%d%d",&a,&b,&c);
    h=hcf(hcf(a,b),c);
    printf("HCF  =  %d\n",h);
    getch();
}

int  hcf(int  x,int  y)
{
    while(x!=y)
    {
        if(x>y)
        x=x-y;
        else
        y=y-x;
    }
    return  x;
}
```

Output1 :

```
ENTER  3  integers
6
12
15

HCF  =  3
```

Output2 :

```
ENTER  3  integers
34
22
12

HCF  =  2
```

10.11 PARAMETERS AS ARRAY :

We can pass the whole array as a parameter to the function. We have already studied in the chapter on pointers that the name of an array is a pointer to the first element of an array. So, if we pass the name of an array as an argument, we are effectively passing whole array as an argument. Naturally, passing an address of an array is call by reference.

Following program explains how whole array can be passed as an argument to the function. In this program, two user defined functions calculate() and input() are used. From main() function these two functions are called with array as parameter as

```
input(x,n);
```

and

```
calculate(x,n);
```

where, x is the array name.

Program :

```
/* Write a program to sum and average the given numbers using function */
#include <stdio.h>
#include <conio.h>
int calculate(int a[], int n)
        /* calculates sum and average of array data*/
    {
    int i,sum=0;
    float avg;
    for(i=0;i<n;i++)
        sum = sum +a[i];
    avg = (float)sum/n;
    printf("Sum of values = %d and Average  = %5.2f\n",sum,avg);
    }
void input(int a[], int n)
                /* gets data in array from user */
    {
    int i;
    for(i=0; i<n;i++)
        {
        printf("Enter value %d\n",i+1);
```

```
        scanf("%d",&a[i]);
        }
    }
main(   )
{
    int x[20],n;
    clrscr();
    printf("How many numbers?\n");
    scanf("%d",&n);
    input(x,n);     /* call function to get data and
                store in array. Whole array passed */
    calculate(x,n); /* call function to process
                    the data. Whole array passed */
}
```

Output :

```
How many numbers?
Enter value 1
1
Enter value 2
2
Enter value 3
3
Enter value 4
4
Enter value 5
5
Enter value 6
6
Sum of values = 21 and Average  =  3.50
```

Following program explains how two-dimensional array can be passed as a parameter to the function. Here, also if we pass the name of an array, whole two-dimensional array is passed to the function.

Program :

```
/* Write a program to sum major diagonal elements of square matrix
 (number of row and columns same)  using function */
#include <stdio.h>
#include <conio.h>
int calculate_major(int a[][5], int m, int n)
                        /* calculates sum */
```

```
        {       /* int a[][5] number of rows not
                   needed in formal parameter */
    int i,j,sum=0;
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            if (i == j) /* check diagonal element */
                sum = sum + a[i][j];
    printf("Summation of diagonal elements is =
                                    %d\n",sum);

}
void input(int a[][5], int m, int n)
                /* gets data in array from user */
                /* in two-dimensional array number
                    of rows not required */
    {
    int i,j;
    for(i=0;  i<m;i++)
        {
        printf("Give row %d data\n",i+1);
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
        }
    }
main()
{
    int x[5][5],m,n; /* x is two-dimensional matrix */
    clrscr();
    printf("What is the size of matrix?    ");
    scanf("%d",&m);
    n= m;      /* row and columns same */
    input(x,m,n);
    /* call function to get data and store in array */
    calculate_major(x,m,n);
            /* call function to process the data */
}
```

Output :

```
What is the size of matrix?
4
Give row 1 data
1 2 3 4
Give row 2 data
5 6 7 8
Give row 3 data
2 3 4 5
Give row 4 data
6 7 8 9
Summation of diagonal elements is = 20
```

12 PARAMETERS AS STRING :

Actually, it is character array as parameter, because string is stored in a single dimensional array of character type. In the chapter on arrays and strings we have used the built-in string functions for manipulating strings. We can write user defined functions to manipulate strings. In this section, we will understand how a string or collection of strings can be passed as a parameter to a function.

The next program shows how the user defined function for finding length of a string can be written. Here, length() is a user-defined function, which takes pointer to character as an argument, and returns integer value.

Program :

```
/* Write a program to find length of a given string with user defined function */
#include <stdio.h>
#include <conio.h>
int length(char *s)
    {
    int cnt=0;
    while(*s != NULL)
        {
        s++;
        cnt++;
        }
    return cnt;
    }
main()
{
    char str[20];
    int len;
    clrscr();
```

```
    printf("Enter a string\n");
    gets(str);
    len =length(str);
    printf("Length of string %s = %d\n",str,len);
}
```

Output :

```
Enter a string
vansh
Length of string vansh = 5
```

Next program shows how we can write the user-defined function to reverse given string. The function internally counts the length of string passed. Then the function maintains two variables one starting from first character and the other starting from the last character of the string and then exchanges first character with last character, second character with second last character and so on till both variable values cross each other.

Program :

```
/* Write a program to reverse a given string with user defined function*/
#include <stdio.h>
#include <conio.h>
void reverse(char s[])
    {
    int i,j,len,temp;
    len=0;    /* length of string initialized to 0 */
    while(s[len] != NULL)
        len++;
    j=len-1;      /* last character in string
                     at length -1 position */
    for(i=0; i<j; i++)
        {              /* exchange s[i] and s[j] */
        temp = s[i];
        s[i]= s[j];
        s[j]= temp;
        j--;
        }
    }
main()
{
char str[20];
int len;
```

```
clrscr();
printf("Enter a string\n");
gets(str);
    printf("Reverse of %s is ",str);
    reverse(str);
    printf(" %s \n",str);
}
```

Output :

```
Enter a string
Mahajan
Reverse of Mahajan is  najahaM
```

Following program explains use of array of strings with functions. As we have already studied, we require two-dimensional character array to store array of characters.

Program :

/* Write a program that takes input student name and their 3 subject marks and calculate total marks and display the data on screen using function */

```
#include <stdio.h>
#include <conio.h>
void disp(char stud[10][20], int mark[10][3],int n);
                                        /* prototype of function */
void starline();
main()
{
char names[10][20];   /* assuming max 10 students */
int marks[10][3];
int i,j,n;
clrscr();
printf("How many students?\n");
scanf("%d",&n);
for(i=0;i<n;i++)
    {
    printf("Enter name of student\n");
    scanf("%s",names[i]);
    printf("Enter 3 subject marks (out of 100)
                                in sequence\n");

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

```
    }
starline();
disp(names,marks,n);
}
void starline()   /* prints line of 50 stars */
{
int i;
for(i=0;i<50;i++)
    putchar('*');
printf("\n");
}
void disp(char stud[][20], int mark[][3],int n)
{  /* displays student name, subject marks and total */
    int i,j,tot;
    for(i=0;i<n;i++)
        {
        tot =0;
        puts(stud[i]);
        for(j=0;j<3;j++)
            {
            printf("%d",mark[i][j]);
            tot = tot + mark[i][j];
            }
        printf("Total = %d\n",tot);
        starline();
        }
}
```

Output :

```
How many students?
3
Enter name of student
sanjay
Enter 3 subjects marks (out of 100) in sequence
69 75 70
Enter name of student
paresh
Enter 3 subjects marks (out of 100) in sequence
68 76 68
```

```
Enter name of student
milan
Enter 3 subjects marks (out of 100) in sequence
40 50 45
**************************************************
sanjay
69 75 70 Total = 214
**************************************************
paresh
68 76 68 Total = 212
**************************************************
milan
40 50 45 Total = 135
**************************************************
```

10.13 FUNCTION RETURNING A POINTER :

Till now we have seen functions that take arguments as pointers and returning only the standard data type. We can have a function which does not return a value but a pointer to a value. The prototype of a function returning a pointer is

```
data_type   *func_name(arguments);
```

Here, * before the name of a function means that the function returns a pointer of the data_type mentioned. So, in the calling function the value must be assigned to a pointer of the appropriate type.

Following program explains these concepts.

Program :

/* Write a program to find out largest of two numbers using a function which returns a pointer to the largest number. */

```
#include <stdio.h>
#include <conio.h>
int *large(int a, int b);
main()
{
    int x,y, *ptr;
    clrscr();
    printf("Give two numbers  ");
    scanf("%d%d",&x, &y);
    ptr = large(x,y);
    printf("Larger of %d and %d is %d\n",x,y,*ptr);
}
int *large(int a, int b)
    {
    int *p, *q;
    p = &a;
    q = &b;
    if (*p > *q)
        return p;
```

```
    else
        return q;
    }
```

Output :

```
Give two numbers
3 5
Larger of 3 and 5 is 5
```

Note following difference

```
int  *f(int,int);
```

Here, f is a function returning a pointer to an integer.

```
int  (*f)(int,int)
```

Here, f is a pointer to a function returning an integer.

10.14 STORAGE CLASSES :

The storage class of a variable determines the scope and lifetime of a variable. The variable values can be stored in computer memory or in registers of CPU. There are four storage classes

- Auto
- Register
- Static
- External

The storage class of a variable decides where the variable will be stored, what will be the default value, what is the scope and what is the life time of a variable.

The syntax for giving storage class to a variable is :

```
storage_specifier data_type variable_name;
```

Auto :

If we do not specify the storage specifier with variable, by default it is auto. All the local variables have auto storage class. Auto variables are **stored in memory, garbage initial value, scope local to function** and **lifetime till the control remains in that function.** As the name suggests, auto variables are created when the function is called and destroyed automatically when the function terminates.

Following program explains the use of auto variables. Remember that all the local variables are by default have auto storage space.

Program :

```
/* Write a program to uses auto variables */
#include <stdio.h>
#include <conio.h>
void fun1();
main()
    {
    auto int a = 100;
    clrscr();
```

```
    fun1(); /* call function */
    printf("%d\n",a);

}
void fun1()
{
auto float a = 25.5;
printf("%f\n",a);
}
```

Output :

```
    25.500000
    100
```

Register :

Variables which are to be stored in the registers of CPU are called as register variables. **register** keyword is used for that purpose, for example,

```
register int a;
```

declares the variable a as register variable. Register storage class should be used for variables which require fast access. As the number of registers of a CPU is limited, we can have limited number of register variables. If the value can not be stored in the register, the compiler may convert register variable to non-register variable. **Storage of register variable take place in registers, have garbage value, scope local to function** and are **created when the function is called and destroyed automatically when the function terminates.**

Static :

All the global variables are by default of static storage class. Default value of static variable is 0. Static global variable is available throughout the entire file in which it is declared, while static local variable is available in the function in which it is defined, but is initialized only once and retains its value even when the control goes out of the function.

So, static variables are **stored in memory, default initial value 0, scope is local to function or file in which declared** and **lifetime is entire program execution with retaining values between function calls.**

Following program explains the use of static variable. Static variable preserves its value between function calls i.e it retains its value even if the function in which it is declared terminates till the end of the program. The variable n in fun1() is declared as static with initial value of 2. This function is called 5 times from main() function. Every time the function fun1() doubles latest value of n. This is possible because variable n is declared as static.

Program :

```
/* Write a program demonstrating static variables */
#include <stdio.h>
#include <conio.h>
void fun1();
main()
    {
    int i;
    clrscr();
```

```
    for (i=1; i<=5; i++)
        fun1(); /* call function */
    }
void fun1()
    {
    static int n=2;
        /* static variable initialized only once */
    n = n *2;
    printf("n = %d\n",n);
    }
```

Output :

```
n = 4
n = 8
n = 16
n = 32
n = 64
```

Extern :

Normally, the scope of global variable starts from its definition up to the end of the program. So, the functions which are defined before the definition of global variable can not access it.

Consider following scenario :

```
main()
{
.....
.....
}
int a;  /* global variable after definition
           of main() function */
void fun1()
{
.....
.....
}
```

In this scenario, even though variable a is global, it is not available in main() function because a is defined below the definition of main() function. To allow access of variable a in main() function, we have to use extern storage class in the declaration as shown below.

```
main()
{
extern int a;   /* extern declaration i.e this variable
                   is already declared some where else */
.....
```

```
.....
}
int a;                  /* global variable after definition
                           of main() function */

void fun1()
{
.....
.....
}
```

Above example shows the use of extern in one file only.

When a program is divided into two or more files, and we want one variable to be available in all the files of a program, we use a variable with extern storage class. For example, the declaration

```
extern int  a;
```

in one file, it means that the variable named as a is already declared in some other file and there is no need to allocate memory to it. **Storage in memory, default initial value 0, scope is global all the files of program and life is entire program execution.**

Following table explains how we can share a global variable between two different .c files and we can compile both the .c files separately without compilation error. Later on both .c files can be made as a part of a project and project can be run as one .exe file. The linker will take care of other things.

File1.c	File2.c

`#include <stdio.h>`
`int a;`
`void fun1();`
`main()`
`{`
`a=10;`
`clrscr();`
`printf("Value of a =`
`%d\n",a);`
`fun1();`
`}`
	`#include <stdio.h>`
	`extern int a;`
	`void fun1()`
	`{`
	`a=a + 10;`
	`printf("Value of a =`
	`%d\n",a);`
	`}`

Steps for creating project are :

- Write file1.c
- Write file2.c
- Open new project as test.prj

- Add item as file1.c
- Add item as file2.c
- Run the project file test.prj

Output :

```
Value of a = 10
Value of a = 20
```

Following are some of the important differences between auto and static storage class

Auto	Static
Declaration: auto int a;	Declaration: static int a;
It is the default storage class for local variables	It is the default storage scope for global variables
Default initial value is 0	Default initial value is garbage
The life time is from function call to function termination	Life time is till program execution
Initialized every time function called	Initialized only once
Does not retain value between function calls	Retain value between function calls

10.15 SOLVED PROGRAMMING EXAMPLES :

Program :

```
/* Write a program to print even numbers up to N using recursive function */
#include <stdio.h>
#include <conio.h>
void even(int a, int n);
main()
{
    int n;
    clrscr();
    printf("Give value of n  ");
    scanf("%d",&n);
    even(0,n);
}
void even(int a, int n)
    {
    if (a <=n)
        {
        printf("%d ",a);
        even(a+2,n);
        }
```

```
    else
        return;

    }
```

Output :

```
Give value of n  13
0 2 4 6 8 10 12
```

Program :

/* Write a program using function which rounds given real number into its nearest integer without using library function. */

```
#include <stdio.h>
int nearest(float x)
{
        int num;
        float temp;
        num = x;
        temp = x - num;
        if (temp > 0.5)
            return num +1;
        else
            return num;
}
main()
{
        float a;
        int ans;
        printf(" Give one real number\n");
        scanf("%f", &a);
        ans = nearest(a);
        printf("Nearest number = %d\n",ans);
}
```

Output :

```
Give one real number
3.123
Nearest number = 3
```

Program :

/* Write a program to calculate perimeter of a square and rectangle using function. Perimeter of a square = 4*l, while perimeter of rectangle

= 2 *l + 2 *b */

```
#include <stdio.h>
#include <conio.h>
void peri_square(int l);
void peri_rectangle(int l, int b);
main()
```

```
{
    int choice;
    int l,b;
    clrscr();
    for(;;)
        {
        printf("1 Square\n");
        printf("2 Rectangle\n");
        printf("3 Exit\n");
        scanf("%d",&choice);
        if (choice ==1)
            {
            printf("Give size\n");
            scanf("%d",&l);
            peri_square(l);
            }
        else if (choice ==2)
            {
            printf("Give length and width\n");
            scanf("%d%d",&l,&b);
            peri_rectangle(l,b);
            }
        else if (choice ==3)
            break;
        }
}
void peri_square(int l)
    {
printf("length = %d Perimeter of square = %d\n",l,4*l);
    }
void peri_rectangle(int l, int b)
    {
printf("length = %d width = %d Perimeter of rectangle  = %d\n",l,b,2*l
+2*b);
    }
```

Output :

```
1 Square
2 Rectangle
3 Exit
2
Give length and width
```

```
3 4
length = 3 width = 4 Perimeter of rectangle = 14
1 Square
2 Rectangle
3 Exit
3
```

gram :

/* Write a C function to exchange two numbers and use it to reverse an array of 10 integers accepted from user. */

```
#include<stdio.h>
#include<string.h>
void  exch(int  *a  ,  int  *b)
    {
    int  temp;
    temp  =  *a;
    *a  =  *b;
    *b  =  temp;
    }

main()
{
int  x[10]=  {1,2,3,4,5,6,7,8,9,10};

int  i;
for(i=0;i<5;i++)
    {
    exch(&x[i],&x[9-i]);
    }
for(i=0;  i<10;i++)
    printf("%d  ",x[i]);
}
```

Output :

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

Program :

```
/* Write a program to check whether the number is prime or not using function */
#include <stdio.h>
#include <conio.h>
int isprime(int num);
main()
{
    int n;
    clrscr();
    printf("Give number to be checked for prime ");
    scanf("%d",&n);
    if (isprime(n))
        printf("Number %d is prime\n",n);
    else
        printf("Number %d is not prime\n",n);
}
int isprime(int num)
{
    int i;
    for(i=2;i<num;i++)
        if (num %i ==0)
            return 0;
    return 1;
}
```

Output-1 :

```
Give number to be checked for prime 35
Number 35 is not prime
```

Output-2 :

```
Give number to be checked for prime 13
Number 13 is prime
```

Program :

```
/* Write a program to convert given number to binary using function */
#include <stdio.h>
#include <conio.h>
void convert_to_bin(int num);
main()
{
    int n;
    clrscr();
    printf("Give number to be converted to binary ");
    scanf("%d",&n);
    convert_to_bin(n);
}
void convert_to_bin(int num)
{
    unsigned int bin[16];
    int i, m =15; /* assuming max 16 bits */
    while ( num !=0)
        {
        bin[m] = num %2;
        m--;
        num = num /2;
        }
    printf("The binary equivalent = ");
    for(i=++m; i<= 15; i++)
        printf("%u",bin[i]);
}
```

Output-1 :

```
Give number to be converted to binary 128
The binary equivalent = 10000000
```

Output-2 :

```
Give number to be converted to binary 108
The binary equivalent = 1101100
```

: LIBRARY FUNCTION :

Important functions of <math.h> file

Function	Syntax	Meaning
acos(d)	double acos(double d)	Returns arc cosine of d
asin(d)	double asin(double d)	Returns arc sine of d
atan(d)	double atan(double d)	Returns arc tangent of d
ceil(d)	double ceil(double d)	Returns a value rounded up to next higher integer of d
floor(d)	double floor(double d)	Returns a value rounded up to largest integer which is lower than d
cos(d)	double cos(double d)	Returns cosine of d
sin(d)	double sin(double d)	Returns sine of d
tan(d)	double tan(double d)	Returns tangent of d
exp(d)	double exp(double d)	Finds ed
log(d)	double log(double d)	Returns natural logarithm of d, positive number
log10(d)	double log10(double d)	Returns logarithm to the base 10 of d, positive number
pow(d1,d2)	double pow(double d1, double d2)	Finds d1 raised to the power d2 i.e. d1d2
sqrt(d)	double sqrt(d)	Returns square root of number d, positive number

Important functions of <ctype.h> file

Function	Syntax	Meaning
isalpha(c)	int isalpha(int c)	Returns TRUE if c is alphabet, otherwise returns FALSE
islower(c)	int islower(int c)	Returns TRUE if c is lower case letter, otherwise returns FALSE
isupper(c)	int isupper(int c)	Returns TRUE if c is upper case letter, otherwise returns FALSE
isdigit(c)	int isdigit(int c)	Returns TRUE if c is digit, otherwise returns FALSE
isalnum(c)	int isalnum(int c)	Returns TRUE if c is letter or digit, otherwise returns FALSE
isprint(c)	int isprint(c)	Returns TRUE if c is printable character, otherwise returns FALSE
ispunct(c)	int ispunct(c)	Returns TRUE if c is punctuation mark (like , ; : etc), otherwise returns FALSE
isspace(c)	int isspace(c)	Returns TRUE if c is white space like blank, tab or newline, otherwise returns FALSE

: SUMMARY :

- 'C' program is a collection of functions. Execution of 'C' program starts from main() function. **Function** is a group of statements working as a single unit performing some well defined task. Every function is given a name. Functions are of two types: **user defined function** and **library functions.**
- **User defined functions** are written by programmer for some specific task which is to be performed many times. The code is written as a single unit.
- **Function declaration** tells us about function name, its return value and about its arguments and their type. It is for information to compiler.
- **Function definition** contains actual logic statements for carrying out the required task. If return type is not mentioned, by default it is taken as **int.** Definition of function inside another function definition is not allowed. The last statement in the function definition is return statement.
- **Functions can be categorized** based on arguments and return value as – with no arguments and no return value, with arguments and no return value and with arguments and with return value.

Scope of a variable decides in which part of the program the variable is accessible. There are two types of scope – **Local** and **Global**.

Local variables are declared inside the body of the function and are accessible inside the function only. They are not initialized automatically.

Global variables are declared outside the function body and can be accessed by all the functions in the program. They are used for sharing of data between different functions of the program. Global variables are automatically initialized to value 0.

When a function is called, the **parameters can be passed** in two ways– **call by value** and **call by reference**.

Recursion is the process of defining something in terms of itself. When a function internally calls itself it is recursion. There must be some condition to stop recursion; otherwise it may lead to infinite loop. By using recursion, complex programs can be solved using less amount of coding. But, **recursive code is difficult to debug and understand** and may lead to infinite loop if terminating condition not properly written.

We can pass array as well as string as arguments to a function.

Storage class of a variable decides the scope and lifetime of a variable. The storage classes are: **auto, register, static** and **external**.

: MCQs :

Function which is readily available in library is called

(a) User defined function (b) Built in function

(c) Library function (d) Both b and c

Function which does not return any value has return type as

(a) void (b) null (c) Nil (d) None of above

What is the return type of a function of which return type is not specified?

(a) double (b) char (c) float (d) None of above

In nesting of a function, one function is defined inside another function

(a) True (b) False

(c) Both True and false as the case may be

In function, _____ parameters are used in definition of function, while _____ parameters are used in function call.

(a) Formal, actual (b) local, global (c) Actual, Formal (d) None of above

While using functions, actual and formal parameters must match in type and numbers

(a) False (b) True (c) Not always True (d) Not always False

_____ variables are declared inside function body, while _____ variables are declared outside function body.

(a) Global, local (b) Static, Dynamic (c) Local, dynamic (d) Dynamic, static

_____ variables are initialized automatically to value 0.

(a) Local (b) int (c) Global (d) Static

If we want to change the value of actual parameters, by the function body i.e function we need to use

(a) Call by value (b) call by reference (c) Both a and b

10. The statement exch(&x, &y); is

(a) Definition (b) call by value (c) call by reference (d) declaration

11. For the function call exch(&x, &y);

What should be the prototype assuming x and y are integers?

(a) exch(int a , int b); (b) exch (int , int);

(c) exch (int *a, int *b); (d) None of above

12. Which are the features of recursion?
(a) decreases code (b) difficult to understand
(c) reduces speed of program (d) All of above

13. If we pass the name of an array as an argument in function call, we are passing _____ elements of an array to function.
(a) First (b) last (c) all (d) some

14. _____ variables retain values between function calls.
(a) Static (b) Register (c) Auto (d) External

15. _____variables are initialled only once, while _____ variables are initialized every time function is called.
(a) Local, Global (b) Static, Auto (c) Auto, Static (d) Global, Local

16. What is the value returned by floor(10.5);?
(a) 11 (b) 10 (c) 0 (d) None of above

17. Recursion is a process in which a function calls
(a) itself (b) another function (c) main() function (d) none of the above

18. Following is an example of user defined function.
(a) printf (b) scanf (c) test (d) main

19. Which of the following is element of function definition?
(a) Function name (b) Parameter list (c) Function type (d) Above all

20. A function without return statement is legal.
(a) True (b) Can't say (c) False (d) None

21. Function returns _____ value by default.
(a) Char (b) Integer (c) Float (d) Void

22. Parameter passing technique in C language is:
(a) Call by Reference (b) Call by Value (c) Both a & b (d) None

23. What does the statement declare?
int (*T)[10];
(a) T is only array of 10 elements (b) T is an array of 10 pointers
(c) T is pointer to an array of 10 integer (d) None

24.
```
main( )
    {   a( );  }
void a( )
    {  int x=5;     printf("%d",++x);      }
```
(a) 5 (b) 6 (c) 0 (d) 4

25.
```
main( )
    {   int x=25, y=5;
                printf("%d", m(x,y));
    }

int m(int a, int b )
    {  return(a%b);  }
```
(a) 0 (b) 25 (c) 5 (d) None of above

26. Default value of local variable is
(a) Garbage Value (b) 0 (c) 1 (d) depend on data type

: ANSWERS :

	2.	(a)	3.	(d)	4.	(b)	5.	(a)	6.	(b)	7.	(c)
(d)	9.	(b)	10.	(c)	11.	(c)	12.	(d)	13.	(c)	14.	(a)
(c)	16.	(b)	17.	(a)	18.	(c)	19.	(d)	20.	(a)	21.	(b)
(b)	23.	(d)	24.	(b)	25.	(a)	26.	(a)				
(c)												

: EXERCISE :

What is function? What is user-defined function? Explain actual argument and formal argument.

What is function prototype?

What are the different categories of functions? Explain any one category with example.

What are the differences between local and global variables?

Explain the difference between call by value and call by reference with suitable example.

What do you mean by recursive function? Explain with example.

What care must be taken when writing a program with recursive function?

List different types of storage classes.

Explain with example static storage class.

Write a function to search a given number in an array. Pass the number to be searched and the array also in which to be searched.

Write a program using function to print Fibonacci numbers up to N.

Write the syntax of function in 'C'. Write sample program to demonstrate "function with arguments and return value"

What is scope, visibility and life time of variables? Explain static varaibales with example.

Write a function using a pointer parameter that calculate maximum element from given array of integer numbers.

Answers to selected exercises

What is user-defined function? Explain actual argument and formal argument.

ns:

Function is a group of statements as a single unit known by some name which is performing some well defined task. The functions which are created by programmer in a program are called as user-defined functions.

When some portion of the code is repeated in the program and when there is a definite purpose of the code, we can write that part of the code as a function. So, use of function reduces the size and complexity of the program.

Formal arguments are the arguments which are used in the definition of a function, while actual arguments are the arguments which are used while calling a function. Following example explains formal and actual arguments using user defined max function..

```
#include <stdio.h>
int max( int x, int y)
{
    if (x >y)
        return x;
    else
        return y;
}
main()
{
```

```
int a=5, b=3;
int ans;
ans = max(a,b);
printf("Maximum = %d\n", ans);
}
```

In above program, x and y are formal parameters, while a and b are actual parameters.

3. What are the different categories of functions? Explain any one category with example.

Ans:

The functions in 'C' language can be divided in following categories:

1. Functions with no arguments and no return value
2. Functions with arguments and no return value
3. Functions with arguments and with return value.

Following program explain the use of second category of function i.e function with arguments and no return value.

```
#include <stdio.h>
void sum(int a, int b);                    /*declaration */
main()
{
int i, j;
clrscr();
printf("Give two integer numbers\n");
scanf("%d %d",&i,&j);
sum(i,j);     /* call sum */
}
void sum(int a,int b)        /* definition. No return
                          value and two arguments*/
{
    int c;
    c = a+b;
    printf("Sum of %d and %d = %d\n",a,b,c);
}
```

6. What do you mean by recursive function? Explain with example.

Ans:

Recusrive function is a function which calls itself. For example factorial of a number, we can say that N! = N * (N-1)!. Here, factorial of N is defined in terms factorial of N-1.

Again (N-1)! = N-1 * (N-2)! and so on.

Thus, recursion is the process by which a function calls itself. Because the function calls itself, there must be some condition to stop recursion; otherwise it will lead to infinite loop. The condition to stop recursion is called as terminating condition. For above example of factorial, the terminating condition is 0! = 1. Thus,

N! = N * (N-1) !

0! = 1 terminating condition.

Following program explains the use of recursion.

```
#include <stdio.h>
long int fact(int n);     /* prototype of function */
```

```
main()
{
int m;
long int ans;              /* long int store large number */
clrscr();
printf("Give the number\n");
scanf("%d",&m);
ans = fact(m);
                                  /* call function fact() with call by value*/
printf("Factorial of %d using function = %ld\n",m,ans);
}
long int fact(int n)   /* function body here */
{
   if (n==1)    /* condition to terminate recursion */
      return 1;
   else
       return  n* fact(n-1); /* recursion here. fact()
                                                  calls fact() internally*/
}
```

12. **Write the syntax of function in 'C'. Write sample program to demonstrate "function with arguments and return value"**

Ans:

The syntax for declaring function is:

type function_name(argument(s));

Here, type specifies the type of value returned, function_name specifies name of user-defined function, and in bracket argument(s) specifies the arguments supplied to the function as comma separated list of variables. The declaration of a function is terminated by semicolon (;) symbol.

Function with arguments and return values:

The function which takes some values as input and returns one value as output is as explained and its definition is written as

```
int max( int x, int y);
```

Here, the return value is Integer type and the function takes two Integer values as input.

The program for finding maximum of two numbers using function can be written as below.

```
#include <stdio.h>
int max( int x, int y)
{
   if (x >y)
       return x;
   else
       return y;
}
main()
{
```

```
    int a=5, b=3;
    int ans;
    ans = max(a,b);
    printf("Maximum = %d\n", ans);
}
```

13. What is scope, visibility and life time of variables? Explain static variables with example.

Ans:

By scope of a variable, we mean in what part of the program the variable is accessible. In what part of the program the variable is accessible is dependent on where the variable is declared. There are two types of scope - **Local** and **Global**.

Local variables	Global variables
Declared inside function body	Declared outside function body
Not initialized automatically	Initialized automatically by value 0
Use of local variables advisable	Too much use of global variables make program difficult to debug, so use with care

Life time of a variable means what will be the life of a variable i.e when the variable will come in to existence and how long it will hold the value. For example, local variables have the lifetime till the control remains in that function. As soon as the control moves out of the function, local variables life is over i.e destroyed.

Storage class decides the scope and lifetime of a variable. There are four storage classes.

- Auto
  - Life time till the control remains in that function.
- Register
  - Life time till the control remains in that function.
- Static
  - Life time is entire program execution with retaining values between function calls.
- External
  - Life time is entire program execution.

Following program demonstrates use of static variable.

```
#include <stdio.h>
void fun1();
main()
    {
    int i;
    clrscr();
    for (i=1; i<=5; i++)
        fun1(); /* call function */
    }
    void fun1()
    {
    static int n=2; /* static variable initialized only once */
    n = n *2;
    printf("n = %d\n",n);
```

```
}
```

It will produce output as shown below.

```
n = 4
n = 8
n = 16
n = 32
n = 64
```

4. **Write a function using a pointer parameter that calculate maximum element from given array of integer numbers.**

Ans:

```
#include<stdio.h>
int getmax(int *b,int n)    /* function  body */
{
    int max;
    int i=1;
    max= *b;
    while (i<=n)
        {
        b++;
        if ( max  < *b)
        max= *b;
        i++;
        b++;
        }
    return max;
}
void main()
{
    int i;
    int a[100];
    int n, max;
    printf("How many numbers? \n");
    scanf("%d",  &n);
    for(i=0;i<n;i++)
        {
        printf("\nGive number %d: ", i+1);
        scanf("%d",  &a[i]);
        }
    max = getmax(a,n);   /*function call */
    printf("\nMaximum number is = %d\n",max);
}
```

: SHORT QUESTIONS :

1. In every 'C' program which function must be there?

⇒ In every 'C' program, main() function must be there.

2. List the types of functions.

⇒ Depending on who created the function, there are 2 type of functions: User defined functions, and Library functions.

3. Write the syntax for declaration of function.

⇒ The syntax is :

type function_name(arguments);

4. Is declaration of function compulsory?

⇒ No, it is not compulsory. Declaration of function is compulsory only when call to the function in the program comes before the definition of the function.

5. List the types of scope of a variable.

⇒ There are two types of scope of variable: local and global

6. Which is the default storage specifier?

⇒ Default storage specifier is: auto.

7. What does static variable mean?

⇒ Static variables are the variables which retain their values between the function calls. They are initialized only once their scope is within the function in which they are defined.

8. What is the default initial value for a static variable?

⇒ Default initial value for a static variable is garbage value.

9. Can main() be called recursively?

⇒ Yes any function including main() can be called recursively.

❖