PROGRAMMING AND DATA STRUCTURES

UNIT-I [Part-I]

Objectives:-

→ Computer System Concepts
→ Computing Environments and Components
→ History Of Computer languages
→ Steps in development Of Computer Program
→ System Development life Cycle

Computer Systems:-

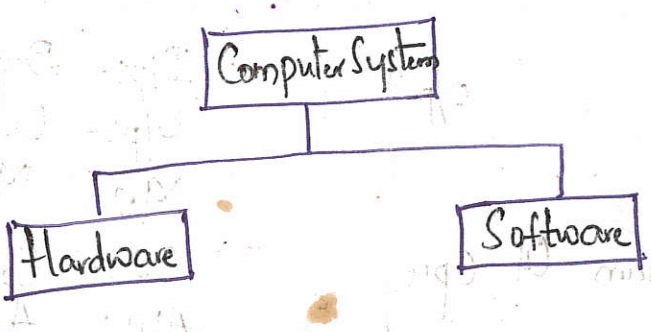
Definition Of A Computer:-

General:- A computer is an electronic device which can perform both mathematical and logical operations.

→ A computer consits of two major components
hardware
&
Software.

Hardware:- It is an physical equipment.

Software:- Collection of programs (instructions) that allow hardware to do its job.



Computer Hardware:-

Hardware component of a computer system consits of five parts:

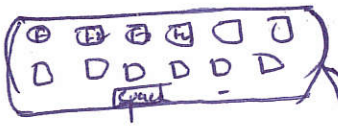
→ Input devices
→ Central processing Unit (Cpu)
→ Primary storage
→ Output devices (and)
→ auxiliary storage devices
(additional, supplementary, reserve)

Input Devices:-

Keyboard is a best example for an input devices where programs and data are entered into the computer.

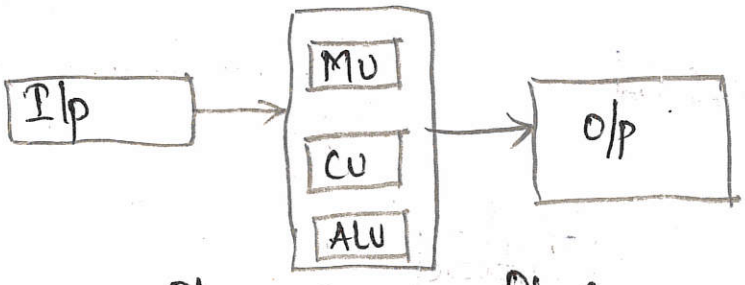
Examples:- Mouse, pen (stylus), touchscreen, joystick, audio input unit.

A pen-shaped device used in olden days one end for writing and other end for erasing.


Keyboard

Central Processing Unit:-

→ Responsible for executing instructions likely arithematic and logical expressions,
→ Comparison of data
→ Movement of data inside the system



Block-Diagram Of Cpu

I/p:- Input Device
O/p:- Ouput Device
MU:- Memory Unit
CU:- Control Unit
ALU:- Arithematic & Logical Unit.

→ It consits of a primary memory (Main memory) used to hold the data temporarily.

→ Once system is turned off it will be erased (or) if the system is in time sharing.

Page 33

Output Devices:-

Generally monitor is an output device, if the output is shown on the screen or it is displayed or called as an "Softcopy", once if soft copy is printed through printer it is called as an "Hard Copy."

Auxiliary Storage:-

→ Auxiliary storage is known as a secondary storage
→ This storage unit is used for both input/Output.
→ This memory unit is used for permanent storage for both input/output
→ Permanent storage (stored in place i.e., ready for the next time we need)

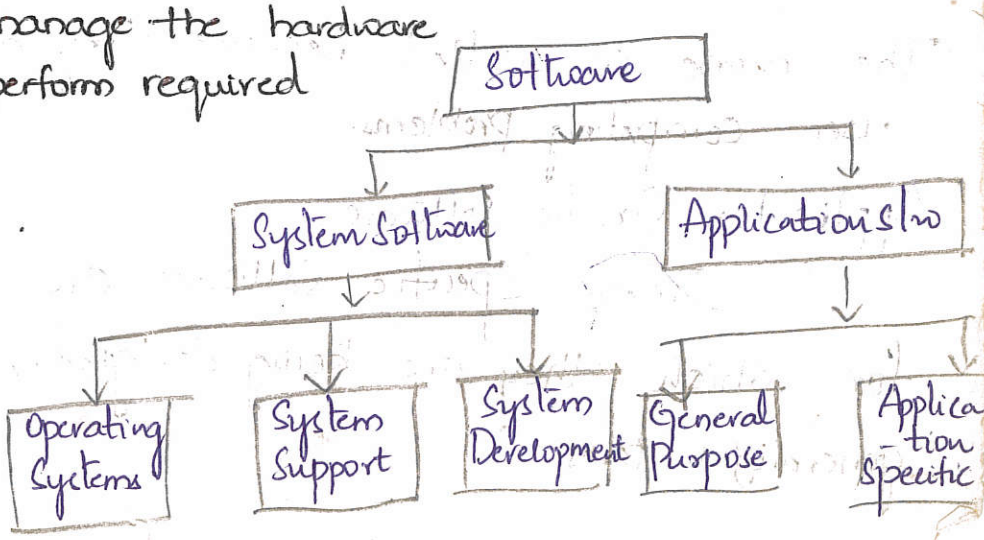
Computer Software

Computer Software is divided into two categories
⊙ System software (and) ⊙ Application software

System Software:-

System software manages the computer resources. It provides the interface between system hardware and user (instructions)

- Consits of programs that manage the hardware sources of a computer and perform required formation processing tasks.
- Consits of three classes.
  - ⊙ operating Systems
  - ⊙ System Support and
  - ⊙ System development



Operating System:-

Operating system provides such as user interface (communication or interaction) files and database access.

→ Major role of this operating system is to keep the system in operating in an efficient manner to allow users to access the system.

ystem Support Systems :-

Page 4

→ This provides system utilities (useful) and other operating services

Examples:- Sort programs and disk format programs.

ystem Development Software:-

→ Includes translators that convert programs into machine anguage for execution, debugging tools (used for making error-free)

Application Software

Application software is classified into two categories.

⊙ General purpose software ⊙ Application specific software

eneral purpose Software:-

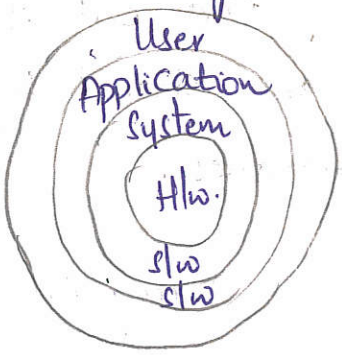
This software is got from a software developer and can be used for more than one application.

Examples:- Includes word processors, data base management systems etc.., The name so called because they can solve a variety of user computing problems.

Application Specific Software :-

These specific softwares are used only for the task for which they are being designed., cannot be used for generalized tasks.

Relation Between System & Application Software.



Page 5

Explination:-

→ The inner core is hardware. The user is represented by the outer layer.

→ User uses some for application software. In turn application software interacts with the operating systems, part of the system software layer.

→ The system software provides direct interaction with the hardware.

COMPUTING ENVIORNMENTS:

In olden days there was only one environment that is he mainframes hidden in a central computing department. e have many different environments.

Personal Computing Environments:-

In 1971, Marcian E. Hoff working for intel, combined basic elements of the central processing unit into the microprocessor. The first computer on a chip was INTEL 4004

→ If we are using a personal computer all are tied into a single unit.

Time Sharing Environment:-

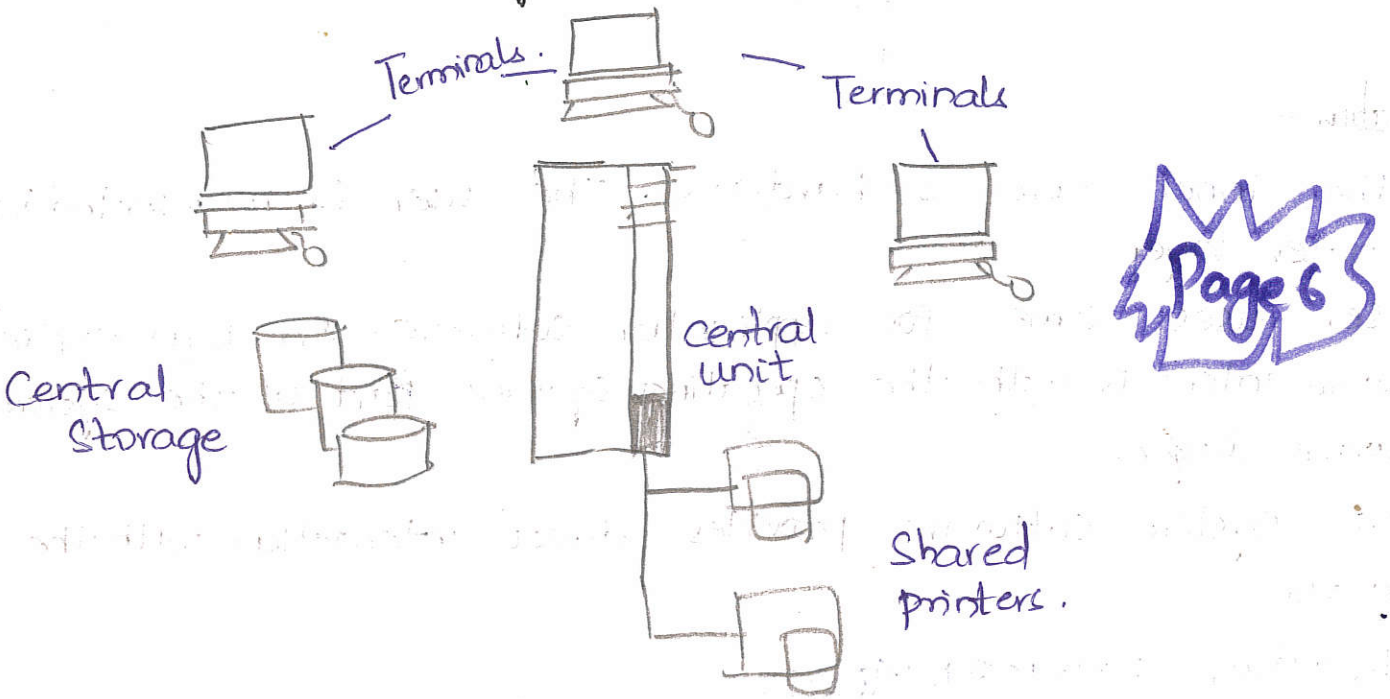
→ In time sharing systems many users are connected to one or more ~~components~~ computers.

→ These computers may be mini computers or central mainframes.

→ The terminals are non-programmable but they are in use.

→ Time sharing environment, the output devices and auxiliary storage devices (such as disks) are shared by all of the users.

Example:- College laboratories.



Page 6

→ Time sharing environment, all computing must be done by central computer, carries out many tasks

- control shared resources
- manage shared data and printing.
- computing

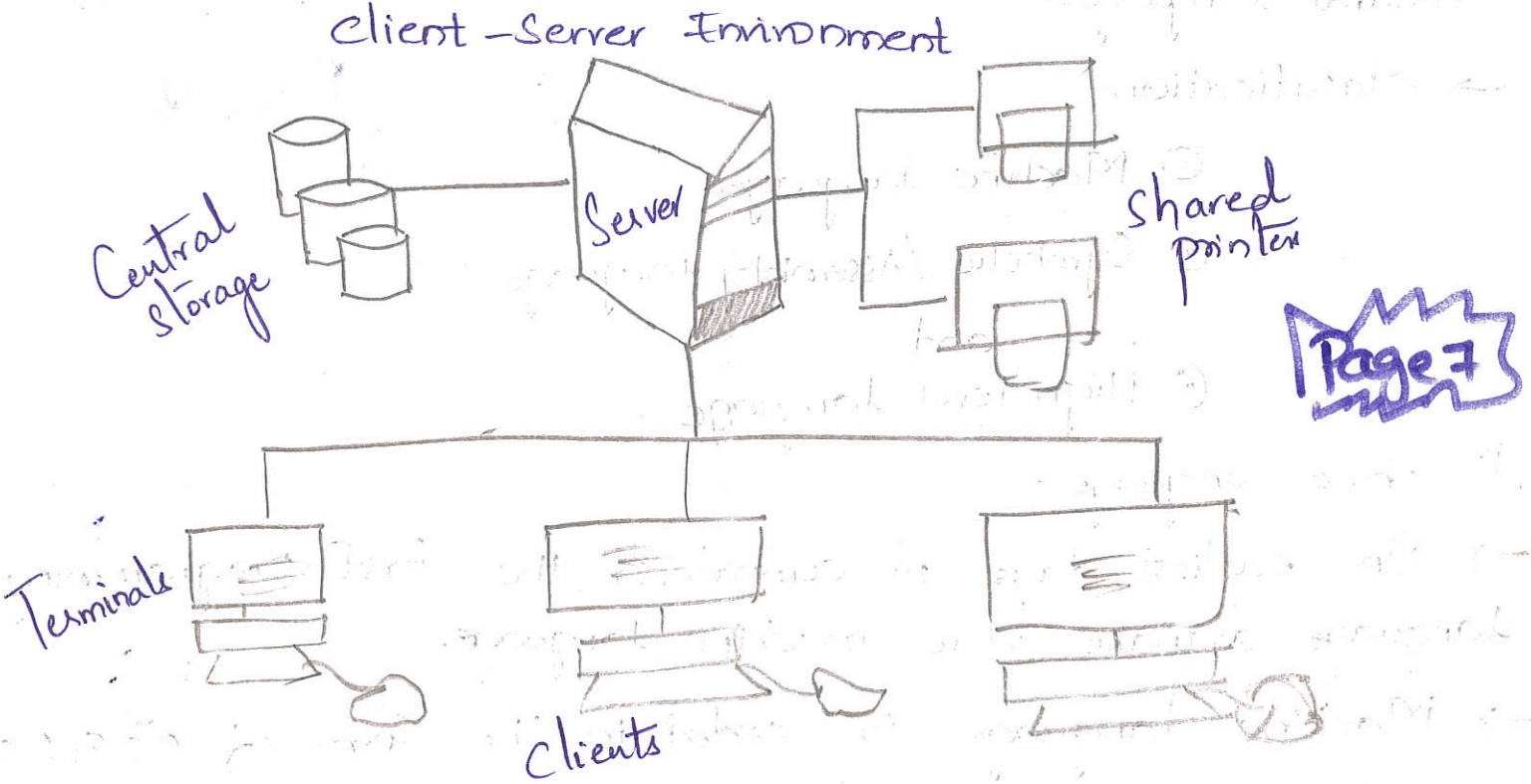
Client / Server Environment:-

→ A client/server computing environment splits the computing function between a central computer and the user computers.

→ Users are given personal computers or workstations so that some of the computation responsibility can be moved from central computer and assign to work stations. (clients)

→ The central computer which may be a powerful microcomputer, minicomputer, or central mainframe system is known as server.

→ Because the work is now shared between the users' computers and the central computer, response time and monitor display are faster and the users are more productive.
(have power of producing [creative])



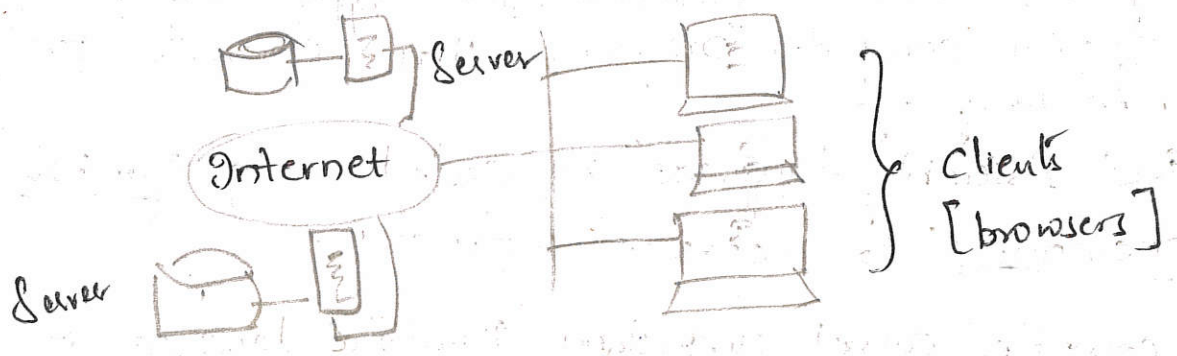
Page 7

Distributed Computing:-

→ Distributed computing environment provides a seamless [smoothly continous (or) uniform] integration [instance] of computing functions between different clients and servers.

→ For example the internet provides connection through out the world.

→ This environment provides a reliable, scalable and highly available networks.
[accuracy]



COMPUTER LANGUAGES:-

→ Languages are generally used for communications.

→ Languages are evolved from machine languages to natural languages.

→ Classification.

- ⊙ Machine language
- ⊙ Symbolic / Assembly language

and

- ⊙ High level language

Machine language:-

→ In earliest days of computers, the first programming language available were machine language.

→ Machine language is coded in the form of 0's & 1's

→ It is so because the internal circuit is made up of switches, transistors and other electronic devices that are of two states

- ⊙ Off state (Represent '0')
- ⊙ ON state (Represent '1')

Symbolic language / Assembly language:-

→ As machine - language became very difficult in coding., In 1950's Admiral Grace Hopper developed the concept of special computer program that converts programs into machine language.

→ Assembly language is coded using symbols (or) mnemonics to represent instruction.

→ A computer cannot understand symbolic language it is again

again translated into machine language with the help of an assembler. [translate symbolic language into machine language].

Page 9

High-level languages.

⇒ Even Symbolic languages greatly improved program efficiency, but it is still difficult to code each machine instruction to code individually.

⇒ To solve this problem they have developed high-level languages.

⇒ High-level languages are portable to many different computers,

⇒ These languages are designed to relieve the programmer from details of symbolic languages.

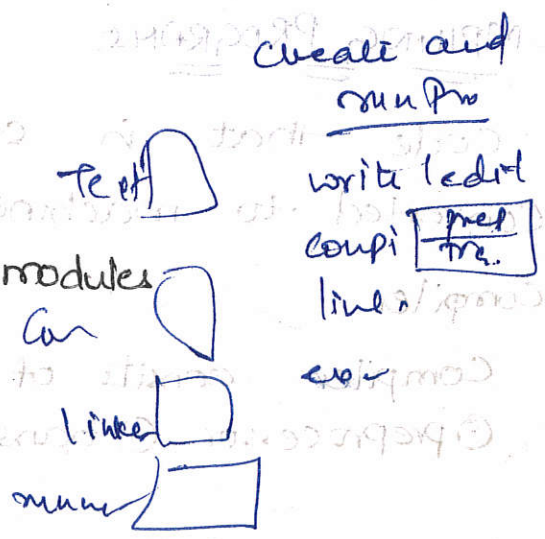
⇒ They must be converted to machine language called "compilation". [FORTRAN, COBOL, C].

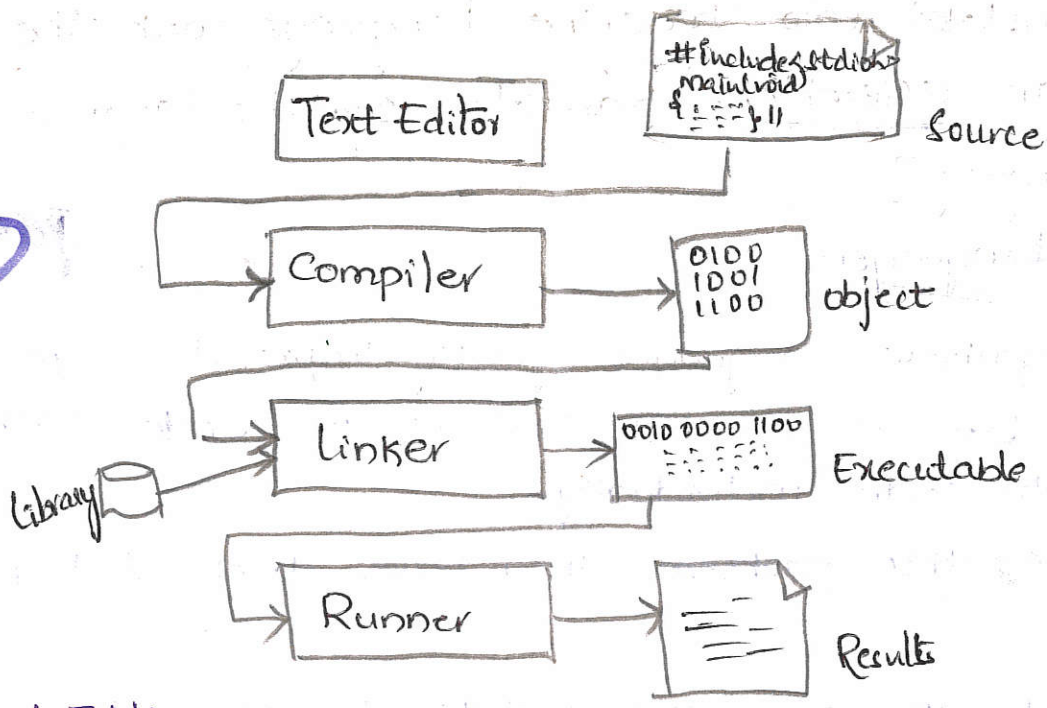
CREATING AND RUNNING PROGRAMS

⇒ We have known the computer hardware understands only machine language., now we shall know the procedure for turning a program written in c into machine-level-language.

⇒ There are basic four steps in developing a process to work.

i) writing and editing the program
ii) Compiling the program
iii) linking program with required library modules
iv) Executing the program.





01

Writing and Editing Programs:-

→ The software used to write programs is known as "text editor."

→ A text editor help us to enter, change and store character data.

→ The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with characters & lines.

→ Our text editor is more often with features like. search, locate, replace, copy & paste commands etc...

→ The file which we save on disk is an input to the compiler, it is called as a SOURCE FILE.

COMPILING PROGRAMS:-

→ Code that is stored in a source file has to be converted to machine language done with help of a compiler.

→ Compiler consits of two seperate programs
① preprocessor ② translator.

PRE-PROCESSOR:-

The pre-processor reads the source code and then prepares it for the translator.

→ while preparing the code, it scans for special instructions known as preprocessor commands.

Page 11

→ These commands tell the pre-processor for preparing the code for translation into machine language.

→ The result of processing is called translation unit

Translator:-

→ After code has been ready for compilation, then the translator starts converting program into machine language.

→ Hence the code is converted into object module.

→ object code is also coded in machine-language and is not yet ready to run, as does not have require 'c' modules

LINKING THE PROGRAM.

→ Hence the linker assembles all these functions and converts object code into executable code.

EXECUTING THE PROGRAM:-

→ Once our program is linked its ready for execution.

→ Getting the program into memory is function of an operating system known as loader.

→ It locates the executable program and reads it into memory.

→ During program execution it reads data for processing, either from the user or from a file.

→ After the program processes the data it prepares the output. [executable code].

SYSTEM DEVELOPMENT:-

- Designing of a program plays a major role. In good structured development techniques, our programs will be efficient, error-free and easy to maintain.

SYSTEM/SOFTWARE DEVELOPMENT LIFE-CYCLE:-

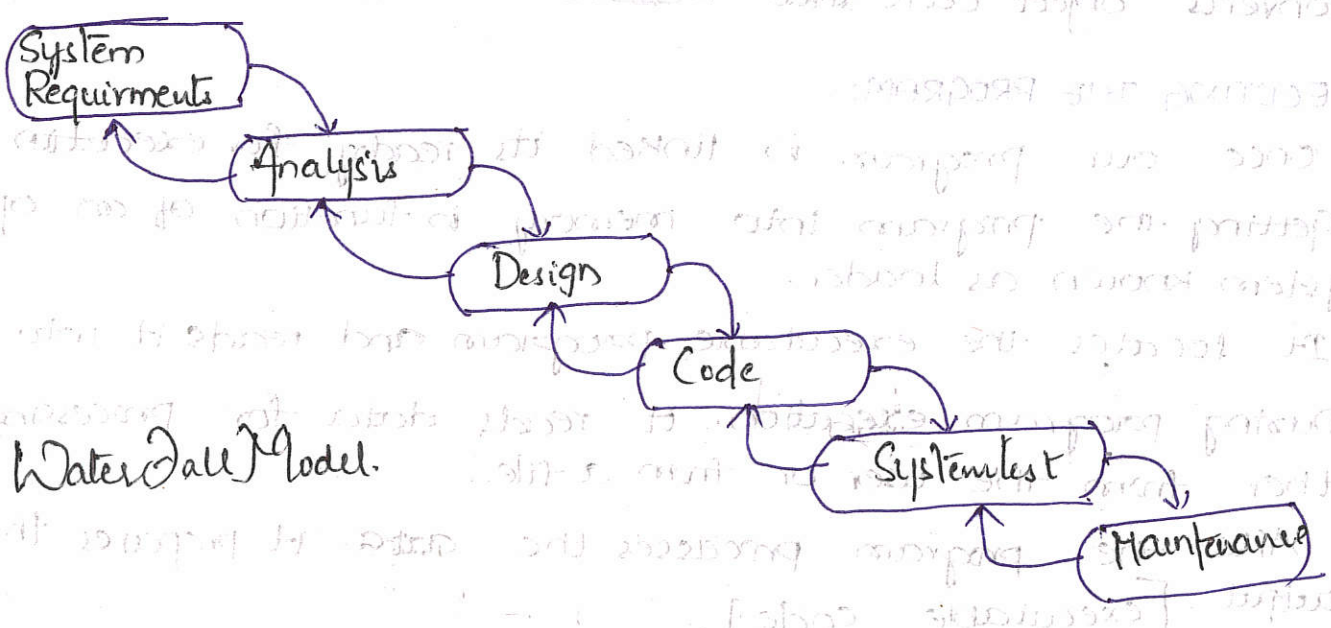
→ Modern programming projects are built using a series of interrelated phases called as system development lifecycle.

→ Popular life cycle used is waterfall model.

→ The basics steps involved in this model are

- → System requirments
- → Analysis
- → Design
- → Code
- → System test and maintenance.

Page-12



Waterfall Model.

Page 13

System Requirements:-

In this phase, the system analyst defines requirements that specify what proposed system accomplish. Requirements are usually stated in terms that the user understands.

Analysis Phase:-

In this phase, looks at different alternatives from a systems point of view

Design phase:-

In this phase we determine how the system is built, the functions of the individual programs that will make up the system are designed determined in design of the files are the database is completed.

Coding phase:-

In this phase we use application programs for development of the system.

Testing phase:-

Code is tested for users satisfaction

Maintenance:-

Keeps the system working once it has been put into production.

→ Although the implication of the waterfall approach is that the phases flow in a continous stream from first to last, this is not really the case. As each phase is developed, errors and omissions will often be found in previous work. when it is happen its necessary to go back to previous phase for rework.

PROGRAM DEVELOPMENT:-

Program development is a multi-step process that requires that we understand the problem, develop a solution, write a program and test it.

- Once after understanding the problem we need to develop our solution.
- The tools help in this task are:
  (i) Structure charts
  (ii) Pseudocode and algorithm (and)
  (iii) Flowcharts

Structure Chart:-

- Structure chart is used to design the whole program, its also known as hierarchy chart.
- This task is similar task of design engineer.
- The major difference between design built by a programmer and the design built by an engineer is that the programmers product is software that exists inside the computer, whereas the engineers product is something that can be see and touch.

A structure chart shows how we are going to break our program into logical steps, each step will be a seperate module.

- A structure chart shows interaction between all the parts (modules) of our program.

→ In the first case, what they find is that they did not fully understand the problem, by taking the time to design the program, they will raise more questions.

→ The second reason programmers code before completing the design is just, programming is tremendously exciting task.

→ In business world when we complete a structure chart design, we review a panel consisting of one or two peer programmers, system analyst and a representing member of testing organization.

Primary intention of review is to increase quality and to save time.

If this is tough to handle we go for algorithm and flowcharts.

ALGORITHM:- / [PSEUDOCODE]

The step-by-step procedure to perform a particular task is known as algorithm.

→ pseudocode loosely defines a syntax and are used to convey design of an algorithm.

→ These are very easy to understand

FLOWCHART:-

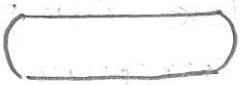
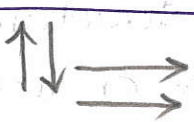
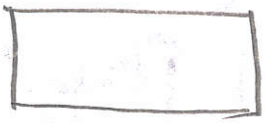
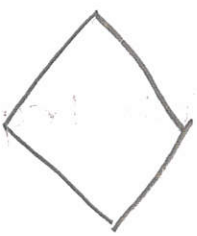
A graphical / pictorial representation of an algorithm is called as a flowchart

→ Representation of flowchart is done with help of different symbols.

→ It is a tool to show the logical flow of a program

Page 16

Different symbols in representing flowchart -

S.NO	SYMBOL	NAME OF SYMBOL	PURPOSE / USE OF IT.
1.		Terminal	Shows beginning/ending of an algorithm.
2.		Flowline	Shows action order in an algorithm, simply flow of a program.
3.		connector	Shows connecting points in an algorithm.
4.		Rectangle.	Assignment statements computation statements.
5.		Parallelogram	For representation of input/output statements
6.		A rectangle with compound vertical lines (module)	Simply made used for void statements.
7.		Rhombous (selectional statements)	Used for conditional statements that cause flow of program to change.
8.		Angular rectangular box	Used for loop controllers.

TESTING METHODS :-

There are Two ways of testing.

Black-box testing

and

White-box testing

Page 17

Black-box testing :-

→ This testing is used to test the program without knowing what is inside(code) and how it works.

→ Developed by looking at requirments statements.

→ Test engineer's plan will help us fully understand the requirments and help us to create our own test plan.

White-box testing :-

→ In this testing user knows everything regarding the program.

→ Its simply like a glass house where everything is visible

→ It is not a simple task, we must make sure that every instruction and every possible situation has been tested.

SOFTWARE ENGINEERING:-

→ Software engineering is the establishment and is used for principles to obtain software that is reliable and that works on real machines.

→ → →

page 18

end of Chapter-1

INTRODUCTION TO C-LANGUAGE

Chapter-2

Introduction:-

→ C is a powerful structured-Programming language

→ 'C' is combined features of a high-level-language and assembly-language, so it is suitable for writing both system software and application Packages.

→ Applications of a c-language:-

Page-1

→ C language is highly-portable

→ It is highly structured-oriented language

→ 'C' language is commonly termed as a middle-level -language

→ C - is termed as Robust language.

→ C language is fastest and highly Extensible.

Description for Applications:-

→ 'C' is a robust language whose rich set of built-in functions and operators can be used to write any complex programs.

→ Programs written in 'C' are efficient and fast. This is due to variety of datatypes and powerful operations.

→ 'C' is highly portable that is the programs written for one computer can be run on another with little (or) no modification.

→ 'C' is well suited for structured programming, as in terms consits of function modules (or) blocks. Modular structure makes program debugging, testing and maintenance.

→ 'C' language has ability to extend itself, basically a collection of functions supported by c-library, i.e., we can continuously add our functions to 'C' library with availability of a large number of functions, which makes programming task simple.

EVOLUTION/HISTORY OF C-LANGUAGE:-

→ 'C' is one of most popular computer languages today because it is a structured, high-level, and machine-independent language.

→ Algol was the first computer language to use a block structured; later it introduced the concept of structured programming.

→ In 1967, Martin Richards developed a language called BCPL (Basic combined programming language).

→ 1970, Ken Thompson created a language using many features of BCPL and called it simply B.

→ In "1972", the evolution of 'C'-language has been developed by "Dennis Ritchie" at Bell laboratories.

Characteristics Of C-Language:-

→ It is a highly structured language
→ It uses features of high-level language
→ It can handle bit-level operations.
→ 'C' is machine-independent language, and highly portable.
→ It supports a variety of datatypes and powerful set of ~~operations~~ operators.
→ It supports dynamic memory-management
→ It enables the implementation of hierarchial and modular programming with help of functions.
→ 'C' can extend itself by addition of functions to its library continuously.

/* Example program */

```
main()
{
  /* printf is an output function */
  printf(" WELCOME TO C-LANGUAGE");
}
```

Main- Function:-

→ Main is a part of every C-program.

→ 'C' permits different forms of main statement.

- main()
- int main()
- void main()
- main(void)
- void main(void)
- int main(void)

→ The empty pair of parentheses `()` indicates that the function has no arguments, (or) we can use void inside the paranthesis.

→ We also specify the keyword void (or) int before the word main, the keyword `void()` means the function does not return any information to operating system.

→ the keyword `int()` means the function returns an integer value to the operating system, if int is used before word main() last statement must be return 0.

TYPES OF DIRECTIVES:-

(1) `#define directive:-`

→ The instruction defines value to a symbolic constant for use in the program.

→ `#define` is a preprocessor compiler directive and not a statement.

→ Hence `#define` lines do not end with a semi-colon.

→ Symbolic constants are generally written in uppercase so that they are easily distinguished.

→ If a symbolic constant is encountered, the compiler substitutes the value associated with the name automatically. If to change value, we have to simply change definition.

Explination:-

→ main() is a special function used by the 'C' language to tell the computer where the program starts. (2)

→ Every program must have exactly one main function.

→ If we use more than one main function the compiler cannot understand which one marks the beginning of the program.

→ The opening brace { in the program after function main() indicates beginning and } indicates end of the function.

→ The function-body contains a set of instructions to perform a given task.

→ The line beginning with /* and ending with */ are known as comment lines.

→ These are used in a program to enhance readability and understanding.

→ comment lines are not executable statements and any thing between comment lines are ignored by the compiler.

→ printf() is a predefined standard c-function for printing output predefined means that it is a function that has already been written and compiled, and linked together with our program at the time of linking.

FORMAT OF SIMPLE-C PROGRAMS:-

```
main() → Function name
{ ——→ Start of program
- - -
- - - → Program statements
- - -
} ——→ End of program
```

→ Every statement in a c-program end with a semi-colon (;)

→ #define instructions are usually placed before the main()-function, and are not allowed in declaration section. ③

Example

/*write a program to find area of a circle*/

```
#include <stdio.h>
#define   PI 3.14
void main()
{
  int radius;
  float area;
Printf("Enter the value for radius \n");
Scanf("%d", &radius);
area = PI*radius*radius;
Printf("The area of a circle is %f", area);
}
```


Output:-

Enter the value for radius

2

The area of a circle is 19.7192

Explination:-

In above program to find area of a circle we need pi-value and it is constant, so with the help of # define directive we can make use of Pi value any where in between the program.

MATH FUNCTIONS:-

→ To make use of standard mathematical functions such as cos, sin, exp etc--, we shall make use of mathematical function in a program.

→ A math function is also a compiler directive that instructs the compiler to link the specified mathematical functions from library

→ The header file is <math.h>

i.e., #include<math.h>

Example

```
/*write a program using cosine function*/
#include<conio.h>
#include<math.h>
# define PI 3.1416
# define MAX 180
main()
{ int angle;
  float x,y;
  angle=0;
  printf("Angle cos(angle)\n");
  while(angle<=MAX)
  {
      x=(PI/MAX)*angle;
      y=cos(x);
      printf("%d %f", angle, y);
      angle=angle+10;
  }
}
```

Output:-

The #include directive:-

(4)

→ 'C' program are divided into, modules (or) functions. These functions are written by users, and some are stored in 'C' library.

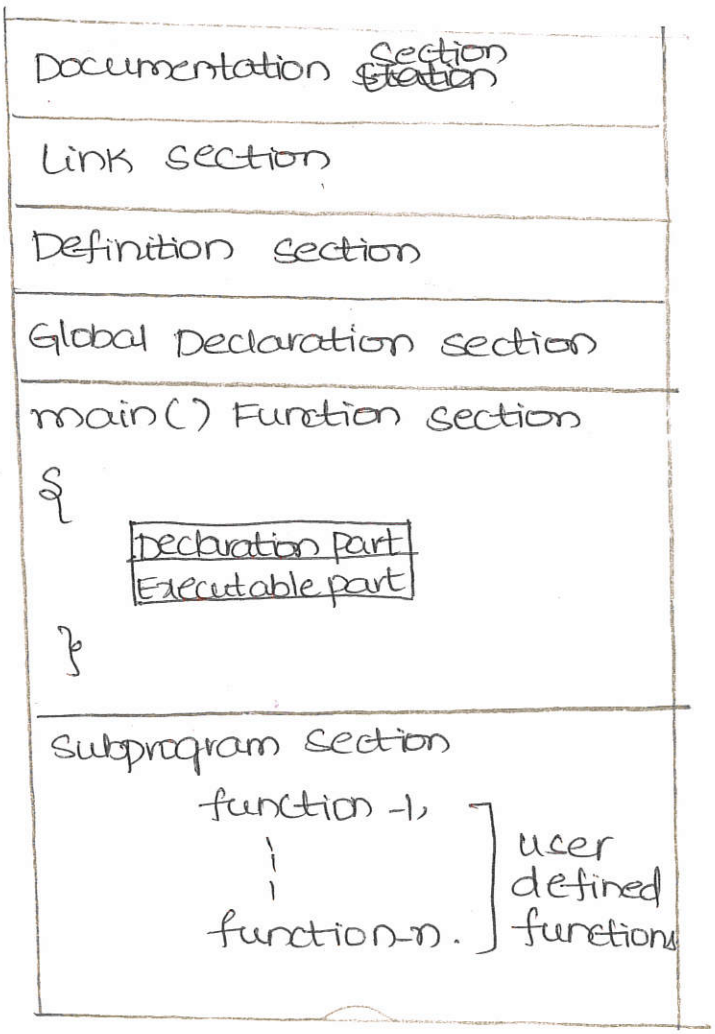
→ Library functions are categorised and stored in the different files known as header files., If we want to access these files it is acheived by making use of Pre-processor directives.

Syntax:- #include <filename>

→ The filename is the name of library file that contains the required function definition.

→ Pre-processor directives are placed at beginning of the program.

BASIC STRUCTURE OF A PROGRAM:-



Explination:-

→ 'C' program can be viewed as a group of building blocks called functions.

→ A function is a subroutine that may include one (or) more statements designed to perform a specific task.

→ 'C' program may contain one (or) more sections.

→ "Documentation Section" consits of a set of comment lines giving the name of program, the author and other details.

→ "Link-section" provides instructions to the compiler to link functions from the system-library.

→ There are some variables that are used in more than one function. Such variables are called global variables, and are declared in the global "declaration section", that is outside function.

→ Every 'C' program must have "one main() function" section. This section contains two parts declaration part and executable part.

→ The declaration part declares all the variables used in the executable part., these must appear in between the closing and opening braces.

→ "closing brace" is logical end of program.

→ All the statements in declaration and executable part ends with a semicolon (;).

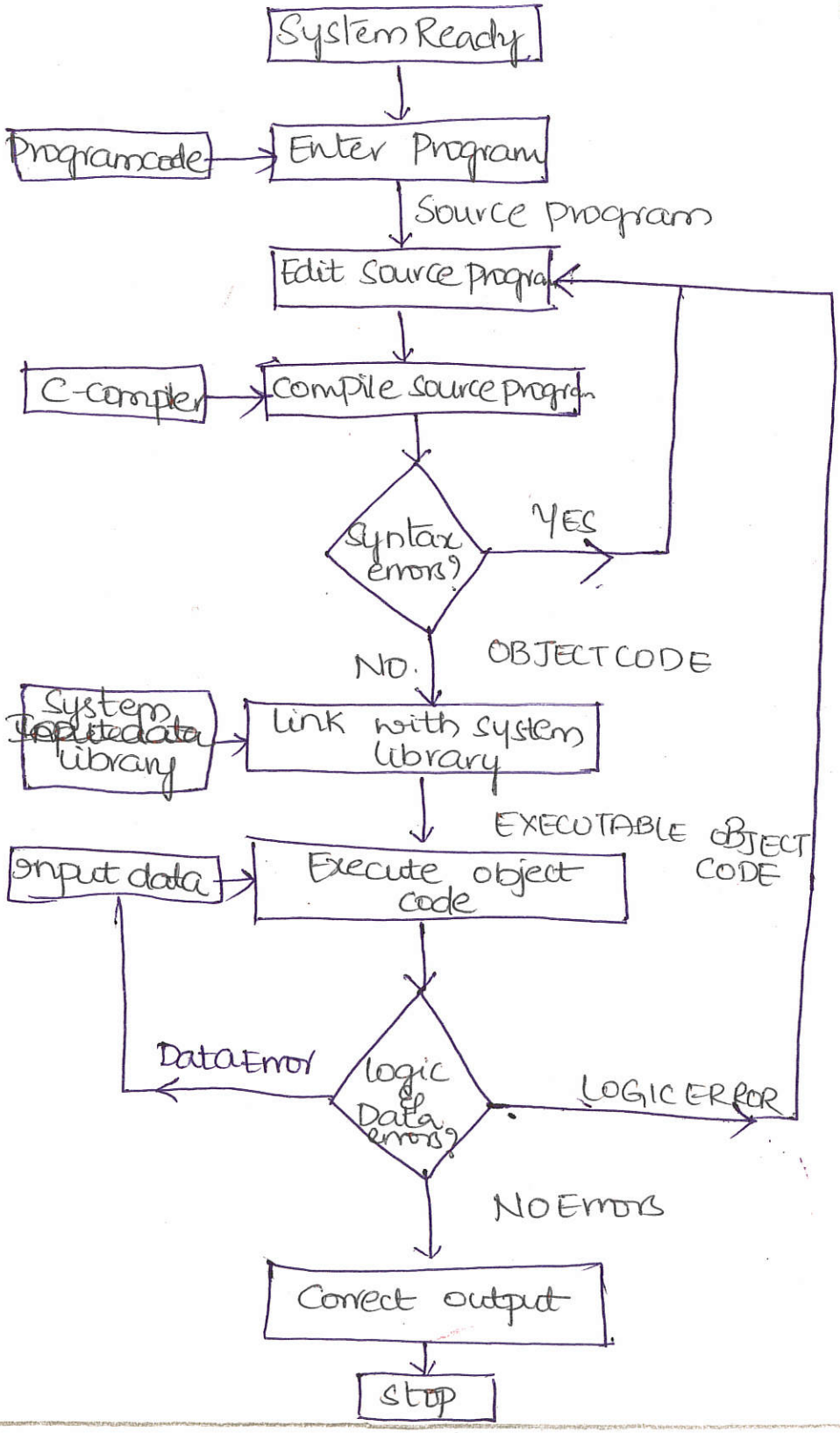
→ The "sub-program section" contains all the user-defined functions that are called in the-main-function

→ Generally user-defined functions are placed immediately after main-function.

EXECUTING A 'C' PROGRAM:-

Executing a 'c' program involves series of steps (5) they are:

1. creating the program
2. compiling the program
3. linking the program
4. Executing



Creating the program:-

→ The program must be entered into a file.

→ The file name can consist of letters, digits and special characters, end with extension (.c)

Example:- HELLO.C
- HELLO → Program name
- .C → Extension name

→ Every 'C' program ~~file~~ file name should end with extension .C

→ When the editing is over, the file is saved on the disk. It therefore can be refered anytime.

→ The program that is entered into file is known as a Source file.

COMPILING:-

→ Now the source file is ready for the execution.

→ Then the source program instructions are translated into a form that is suitable for execution by the computer.

→ The translation is done after instructions are verified with error-free, if it is then the translated program is stored on an another file.

i.e. .C file is converted to .obj

i.e., Example: HELLO.C ⟷ HELLO.obj

↓ machine code/ object code.

LINKING:-

→ "Linking" is process of putting together other program files and functions that are required by the program.

→ If any mistakes in the syntax and semantics of the language are listed the compilation stops, the errors then should be rectified and compilation has to be done again.

→ The compiled and linked program is called executable object code and is saved on another file as .exe
↓
executable code

Example :- Hello.exe.

EXECUTING PROGRAM:-

→ Now the program is ready for execution, during this process the program may request for some data to be entered through the terminal (keyboard).

→ There are some times possibility that desired results are not produced, it means the program is wrong with logic (or) data.

→ Hence, If the process goes perfectly with all the steps perfectly the desired result is automatically generated.

C-character set:-

→ A programming language is designed to help process certain kind of data consisting of numbers, characters and strings and to provide useful output known as information.

Program:- The task of processing of data is accomplished by executing a sequence of precise instructions called program.

→ These instructions are formed using the [character set].

~~Token: the fundamental unit (or) smallest individual unit of~~

The character set in 'C' includes

→ Letters
→ Digits
→ Special characters (and)
→ White spaces.

CHARACTERS	NOTATIONS
⊛ Letters	⊙ A-Z (uppercase), a-z (lower case)
⊛ Digits	⊙ 0-9 (all decimals)
⊛ Special_characters	⊙ comma (,) ampersand (&) period (.) caret (^) semicolon (;) asterisk (*) colon (:) minus sign (-) Questionmark (?) plus sign (+) apostrophe (') open angle bracket (<) (or) less than Quotation mark (") number sign (#) Exclamation (!) close angle bracket (or) (>) greaterthan vertical bar (|) slash (/) left parenthesis (() blackslash (\)

Page 12

right parenthesis (')')
tilde (~)
left bracket ([)
underscore (_)
right bracket (])
dollar sign ($)
left brace ({)
percent sign (%)
right brace (})
Blank space
Horizontal tab
Carriage return
new line
Form feed

(*) White spaces

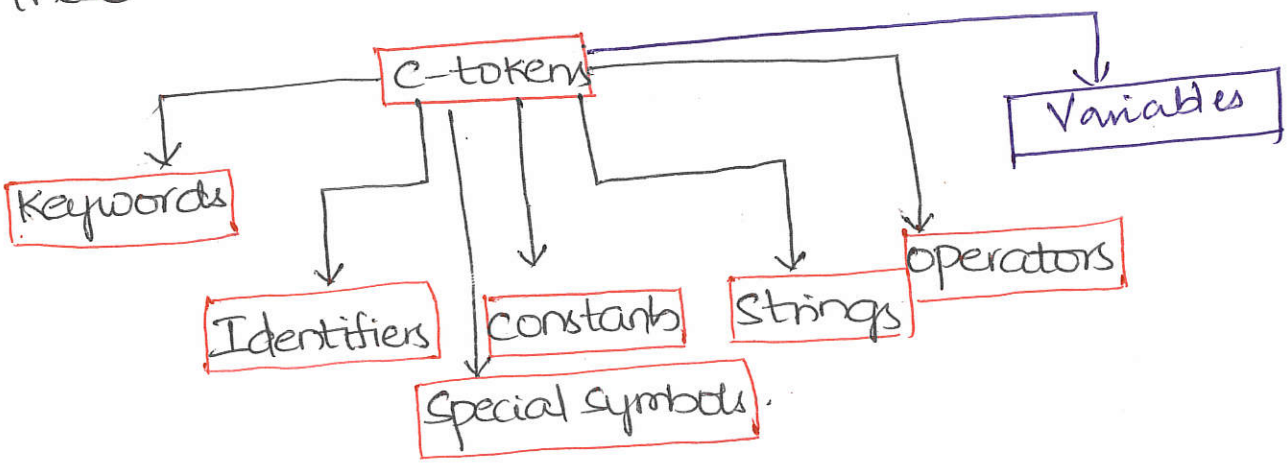
(7)

'C' TOKENS:-

Definition of token:-

The smallest individual unit (or) the fundamental unit of a 'C' program is called as a token.

→ These 'C' tokens are classified into '6' types.



KEYWORDS & IDENTIFIERS:-

→ Every 'C' word is classified as either a keyword (or) an identifier

→ Keywords have fixed meanings and they cannot be changed.

→ Keywords serve as the building blocks for the Program statements, they must be in lower case.

→ Examples :- int, float, if, etc---- called

→ "Identifiers" refers to names of variables, functions and arrays.

→ Rules for Identifiers :-

① The first character must be an alphabet (or) underscore.

Eg:- A23; } valid
_A23;
_1A; }
1A; } invalid.
1_A;

abc; abc123; abc_123 [valid]
123_abc [invalid]

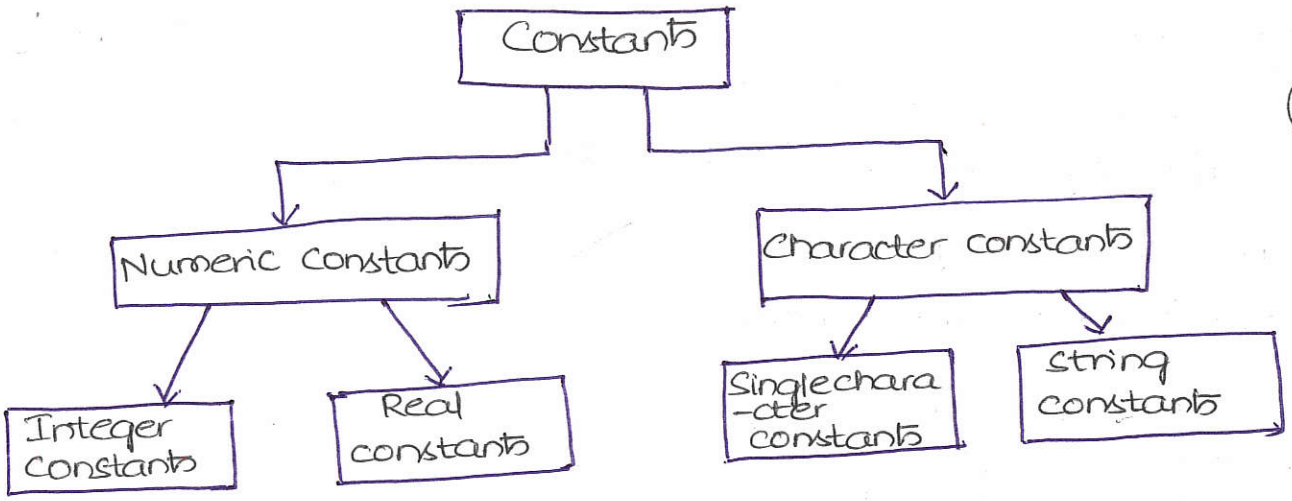
② It must consist of only letters, digits (or) underscore.
③ Only 31 characters' name is allowed
④ It cannot be a keyword.
⑤ No white spaces are included.

page 14

CONSTANTS:-

→ A constant is fixed-value that do not change during the execution of a Program.

→ 'C' supports several constants.



Numeric constants:-

Integer constants:-

An integer constant refers to a sequence of digits.

→ There are three types of integers like

- decimal integer
- octal integer (and)
- Hexa-decimal integer.

Page 15

"Decimal-integers" consist a set of digits (0 to 9) either positive (or) negative. [Eg 123 (or) -321]

→ Spaces, commas (or) special characters are not permitted between digits [Eg ($123), (20,000) are invalid]

"octal-integer" consits any combination of digits from set (0 to 7) [Eg: 037, 0435, 0551] with a leading "0".

"Hexa-decimal integer" consits any combination of digits from set [(0 to 9) and (10-15) represented as A through F] with leading 0x.

0-0	5-5	10-A
1-1	6-6	11-B
2-2	7-7	12-C
3-3	8-8	13-D
4-4	9-9	14-E
		15-F

(0 to F)

↳ Representation of a hexa-decimal numbers.

Real-Constants:-

→ Integer numbers are inadequate to represent quantites that vary continuosly, such as distances, heights, temperatures etc..,

→ Numbers containing fractional parts are called as the real (or) floating point constants.

Eg:- 0.0083, -0.75 - - -

Character constants:-

Single-character constants:-

→ A single-character constant (or) a character constant, contains a single-character enclosed within a pair of single quote marks.

Eg:- '5', 'x', ...

Note:- 5 is different from '5'

5 → number

'5' → character constant

BACK SLASH-CHARACTER-CONSTANTS:-

→ 'C' supports some special back-slash characters constants that are used in output functions.

→ That each one of them represents one character, consist of two characters.

→ These character combinations known as escape sequences.

Constant	Meaning
'\a'	audible alert(bell)
'\b'	back space
'\f'	form-feed
'\n'	new-line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical-tab
'\''	singlequote
'\"'	double quote
'\?'	Question mark
'\\'	back slash
'\0'	null

page no

String constants:-

(9)

→ A string constant is sequence of double characters enclosed in double quotes.

→ The characters may be letters, numbers, special characters and blankspaces.

Examples:- "Hello!", "1987", "x"...

→ It is important to note that a character constant (eg:'x') is not equivalent to the single character constant (eg: "x").

Single character

VARIABLES:-

→ A variable is a dataname that may be used to store a data value.

→ A variable may take different values at different times during execution

→ A variable name can be chosen by the programmer in a meaningful way.

Conditions for declaring a variable:-

page 14

→ They must begin with a character. some system permits underscore as first character.

→ Length of variable is 31-characters.

→ It should not be a keyword, white spaces are also not allowed.

char	Invalid (char is a keyword)
First_tag	Valid
1a	Invalid (starting letter should be character)
price$	Invalid (no special symbols other than under score(_))
int_type	Valid

Variable as a constant (compile time)

Syntax:-

<datatype> <variablename> = value

Eg:-

```
int a=10;
```

Declaring variable as volatile (runtime)

Syntax:

<datatype> <variablename>

```
int a;
```

Operators and expressions:-

→ Operator is a symbol that tells the computer to perform certain mathematical (or) logical manipulations.

→ operators are used in programs to manipulate data and variables.

→ 'C' operators are classified into number of categories.

1. Arithematic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment & Decrement operators.
6. conditional operators
7. Bitwise operators (and)
8. Special operators.

Eg:- Unary
++a
i/p operand

Eg:- Binary
a + b
a, b → are operands
'+' → is operator

Arithematic operators:-

'C' provides all basic arithematic operators, they can operate on any built-in data type allowed in C.

→ Integer divison truncates any fractional part; the modulo division operation produces the remainder of an integer divison.

Eg: a-b, a+b, a*b, a/b, a%b

→ Here a, b are called as operands, the modulus operator cannot be used on floating point data.

Integer Arithematic:-

When both the operands in a single arithematic expression such as a+b integers, the expression is called as an integer expression., and operation is called integer arithematic which yields an integer value.

→ During modulo operation the sign of result is always sign of first operand.

Real Arithematic:-

(10)

→ An arithematic operation involving only real operands is called real arithematic, real operand may assume the values either in decimal (or) exponential notation.

→ The final value is an approximation of the correct result.

Eg:- $x = 6.0/7.0 = 0.857143$

$y = 1.0/3.0 = 0.333$

$z = -2.0/3.0 = -0.666$

→ The operator (%) cannot be used with real operands.

Mixed-mode arithematic:-

→ When one of the operands is real and the other is an integer, the expression is called mixed-mode arithematic

→ Either of operand is realtype, then the real operation is performed and result is always real number.

Eg:- $15/10.00 = 1.5$.

(and)

$15/10 = 1$

RELATIONAL OPERATORS:-

→ When comparison of two (or) more quantites to be evaluated with help of relational operators.

→ An expression such as a<b (or) K≥0 containing a relational operator is termed as relational expression.

→ If the specified relation is true it yields value '1' and '0' if expression is false.

Categories of relational operators:-

Operator	Meaning
<	is less-than
<=	is less than (or) equal to
>	is greater-than
>=	is greater-than(or) equal to
==	is equal to.
!=	is not equal to.

→ These are generally used in decision-making statements as if and while.

LOGICAL OPERATORS:-

→ ~~logical expressions~~ These combines two (or) more relational expressions termed as logical expressions (or) compound relational expression.

→ These expression also yields a value either (one) (or) zero.

Categories of logical operators:-

&& (logical-And)
|| (logical-or) (and)
! (logical-not)

Truth table for logical-And, logical-or

operand-1	operand-2	(logical-And) operand-1 (&&) operand-2	(logical-OR) operand-1 (||) operand-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Eg:- ① if (age > 55 && salary < 1000)

On this above expression if both the conditions are satisfied then only yields a value '1' else either of condition fails (or) both fails yields a value '0'.

② if (number < 0 || number > 100)

On this above expression any of statements is true entire expression is true yields a value('1') and if both expression are false ~~its~~ entire expression is false.

ASSIGNMENT-OPERATORS:-

These operators are used to assign the value (or) result of an expression in a variable.

→ In addition to these we also have short-hand assignment operators.

Syntax for short-hand assignment operator

```
V op = exp
```

'V' is a variable
'op' is operator
'exp' is expression.

Page 21

Eg:- x += y+1 is similar as x = x+(y+1)

'x' is variable
'y+1' is expression
'+' is operator

Uses of short-hand assignment operators:-

→ No need of repeating same variable appeared on left-side.
→ Statements are efficient and easy to read.

INCREMENT AND DECREMENT OPERATORS:-

→ 'C' language supports two special and useful operators as increment and decrement operators.
(++) & (--)

→ The increment and decrement operators are classified in two ways as (post and pre)

Pre-increment and post-increment:-

→ A pre-increment (or) prefix notation first adds '1' to the operand and the result is assigned to the variable.

Eg:- m=5
y=++m

Here value of m=6 and y=6.

→ A post-increment (or) postfix notation first the value is assigned then the operand value is increased.

Eg:- m=5
y=m++

Here value of y=5, and m=6.

Post-decrement and pre-decrement:-

→ A post-decrement (or) postfix notation first value is assigned then the operation decrement is performed.

Eg:- m=5, y=m--

Here value of y=5, and m=4.

→ The pre decrement (or) prefix notation first decrements the value and then value is assigned to variable.

Eg:- m=5, y=--m Here value of y=4 and m=4

Some of the rules for using (++) and (--) operators are:- (12)

⊙ Increment and Decrement operators are unary operators and require variable as their operands.

⊙ When postfix ++ (or) -- is used with a variable in an expression, the expression is evaluated first using the original value of variable and then variable is (incremented (or) decremented)) by '1'. (Eg:- a++/a--)

⊙ When prefix --(or)++ is used with a variable the variable is either incremented (or) decremented then the value is assigned. (Eg:- ++a/--a)

⊙ precedence and associativity of ++ and -- operators are same as unary (+) and unary(-)

CONDITIONAL OPERATOR (OR) TERNARY OPERATOR:-

A ternary operator '?:' is available in 'C' to construct the conditional expression.

Syntax:-

```
<exp1>?<exp2>:<exp3>
```


→ exp1, exp2, exp3 are expression

working of conditional operator:-

→ First the "exp-1" is evaluated first, if exp-1 is true(non-zero) then exp2 is evaluated and becomes the value of expression.

→ If "exp-1" is false, the exp3 is evaluated and becomes value of expression.

Eg:-
```
a=10, b=5, x;
x= (a>b)? a:b;
```
In the above example 'x' is assigned with value 'b'.

Bit-wise operators:-

→ For manipulation of data at a bit-level we make use of bitwise operators.

→ with help of bitwise operators testing of bits is done (or) shifting of bits is done.

→ These are not applied on float (or) double.

Categories of Bit-wise operators:-

Operators	Meaning
&	Bitwise-AND
|	Bitwise-OR
^	exclusive OR
<<	Leftshift
>>	Rightshift
~	one's complement

Bitwise And (&) & Bitwise OR (|):-

These operators work similar to logical-and/or.

→ Bitwise-And(&) evaluates true only if the entire expression is true else, the entire expression is false.

→ Bitwise-OR(|) evaluates true if either of expression is true.

Bitwise Exclusive (or):- (^)

Page 24

Truthtable

input-1	input-2	input1 (^) inp-2
1	1	0
1	0	1
0	1	1
0	0	0

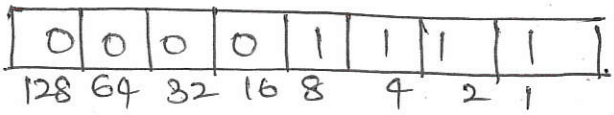
«Left-Shift and Right-Shift» -

13

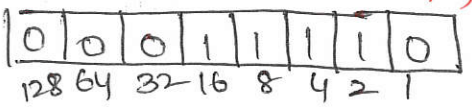
→ The left-shift(<<) operator shift the position of bits towards left, and where as right-shift(>>) operator shift the position of bits towards right

Eg:- a = 15, m;

Leftshift operation m = a<<1



m=30 (after left-shift)



→ In above example the value 'a' is shifted towards one left only one-time if operation is (a<<2) shifts (or) apply shifting twice

Formula for left-shift

$$\text{Variable} = (\text{value} * 2^n)$$

n - no of shifts
value - any integer value

Page 25

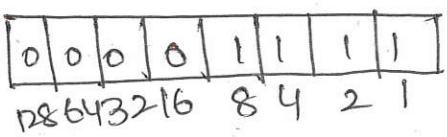
Eg:- a = 5 (a: Variable, 5: value) a<<2 (2: value of n = 2 (no. of shifts))

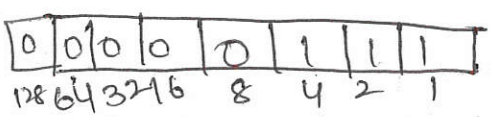
Hence ⇒ $a = 5 * 2^2$

$= 5 * 4 = 20$

→ The Right-shift(>>) operator shift the position of bits towards right.

Eg:- a=15, m; Right-shift operation m=a>>1





m=7 (after right-shift)

→ In above example value of a is shifted toward right only once if operation is (a>>2) shifts (or) apply shifting twice.

Formula for right-shift operation:-

$$\text{variable} = \text{value} / 2^n$$

n - no. of shifts
value - any integer values

→ Eg:- num = 10

num >> 1

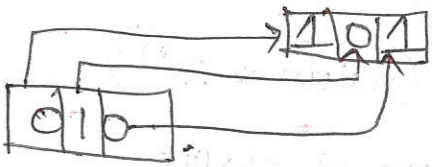
num = $10/2^1$

= 5.

One's compliment (~):-

This operator converts the bits 1-0 and 0-1

Eg:- a = 5 represented as in binary form

~a = 

Special-operators:-

In 'C'-language we have two basic important special operators (comma(,) and sizeof operator).

→ A comma operator is used to seperated two variables, if they have relation they are linked together.

→ A sizeof operator() gives the variable data-type size. i.e., data-type size of given operand.

Example:-

```
main()
{int a;
 float b;
 char c;
 Printf("%d", sizeof(a));
 Printf("%f", sizeof(b));
 Printf("%c", sizeof(c));
}
```

Page 26

output:-

2
4
1

Explination:-
as a is integer hence it is 2 bytes
→ b is float hence it is 4 bytes
→ c is char hence it is 1 byte.

DATA TYPES:-

'C' language is rich in data-types. It supports variety (14) of datatypes.

There are three classes of datatypes

- Primary datatypes
- Derived datatypes (and)
- user-defined datatypes

→ 'C' supports five fundamental data-types likely integer(int), character(char), floating point(float), double and void.

Datatype	Range
char	-128 to 127
int	-32,768 to 32,767
float	$3.4e^{-38}$ to $3.4e^{38}$
double	$1.7e^{-308}$ to $1.7e^{308}$

Page 22

Character Type:-

→ A single character can be defined as a character(char) type data.

→ characters are usually stored in 8 bits (one byte) of internal storage.

Void Types:-

→ The void does not have no values, these generally specify the type of functions.

→ A type of a function when it does not return any value.

DECISION MAKING & BRANCHING:-

→ 'C' program is set of statements that are normally executed sequentially in order they appear.

→ 'C' language possesses such decision-making

- if statement
- Switch statement (and)
- goto statement.

Decision making with if-statement:-

→ If statements and is used to control the flow of execution statements.

→ It is basically two-way decision statement and is used in conjuction.

→ These expressions evaluate test-conditions and then depending on that value is returned for true block (or) false block

These are basically categorized as

→ Simple-if

→ if-else statements

→ Nested if-else (and)

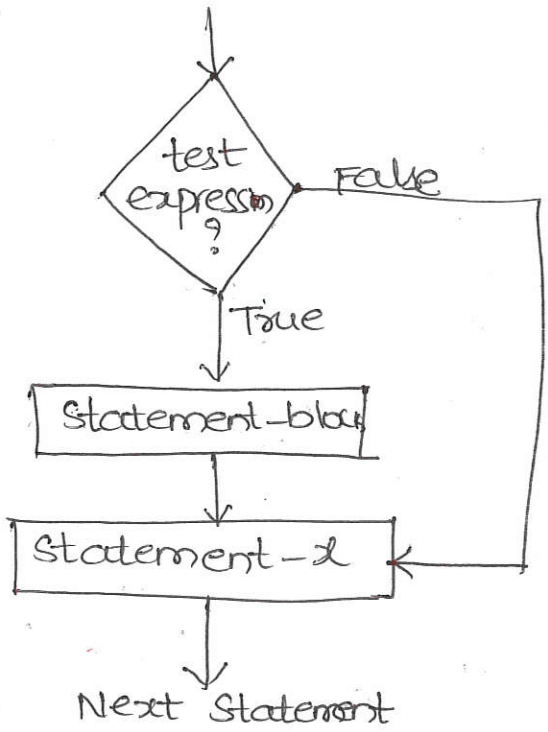
→ else if ladder.

Simple-if-Statement:-

General-form:- (or) syntax:-

```
if (test_expression)
{
  statement_block;
}
Statement_x.
```

Flow-chart



15

Page 29

Explination:-

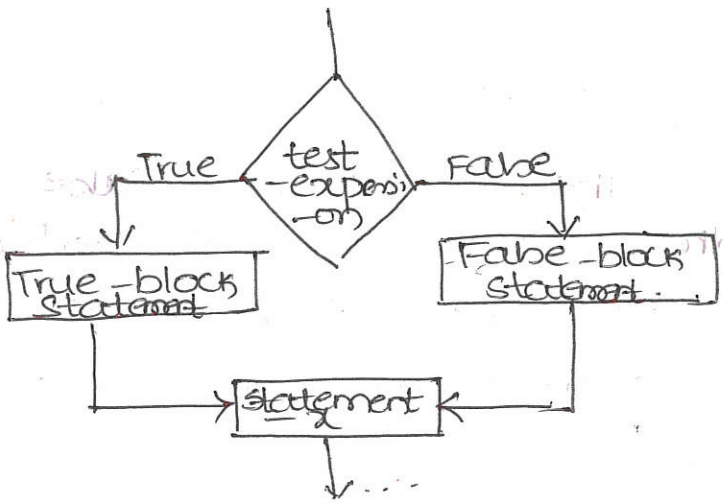
On the above if the test expression is true the Statement-block is executed along with Statement-x. But if test expression is false only statement-x is generated

If-else statements:-

Syntax:-

```
if (test-expression)
{
  true-block statements
}
else
{
  false-block statements
}
statement-x.
```

Flowchart:-



Explination:-

If the test expression is true, then true block statement(s) evaluated, else false block is executed.

→ In this either of true (or) false is generated but not both.

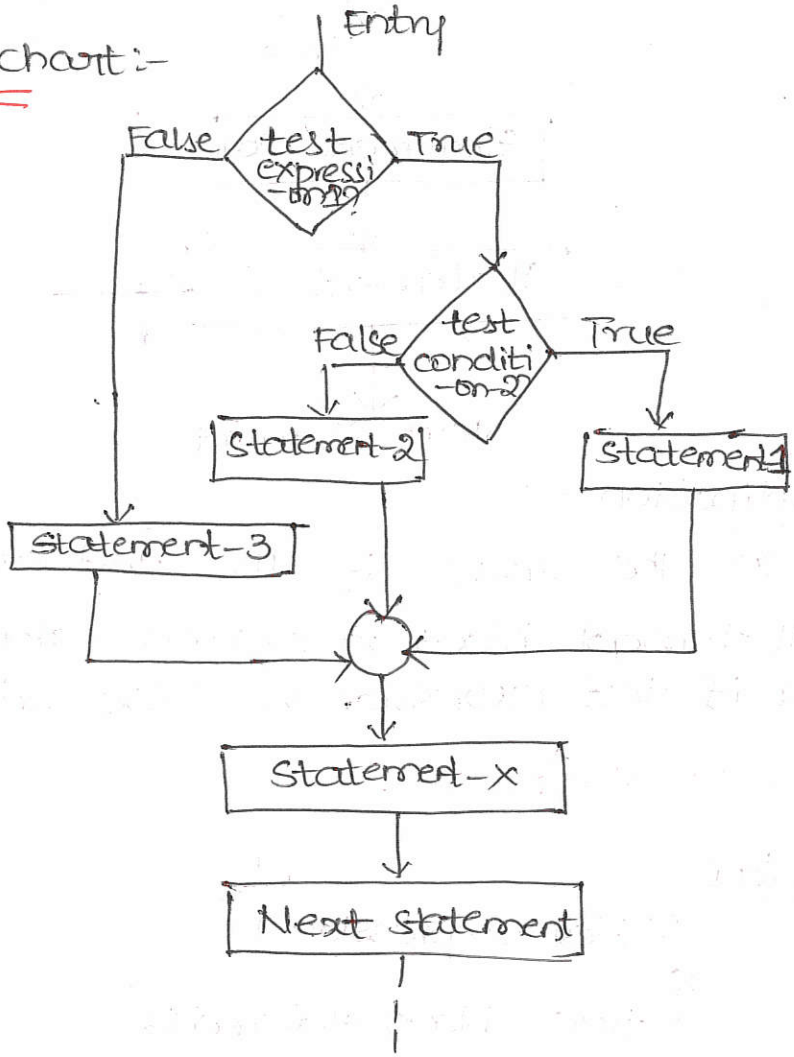
Nesting of if-else statements:-

→ When a series of decisions are involved, we can have more than one if-else statements.

Syntax:-

```
if (test_condition1)
{
  if (test_condition2)
   { statement-1;
   }
  else
   {statement-2;
   }
}
else
{statement-3;
}
statement-x;
```

Flowchart:-



Explination:-

If the condition-1 is false statement-3 is executed, otherwise it continues to perform second-test, if condition-2 is true statement-1 is executed otherwise statement-2 is executed and control is transfered to statement-x;

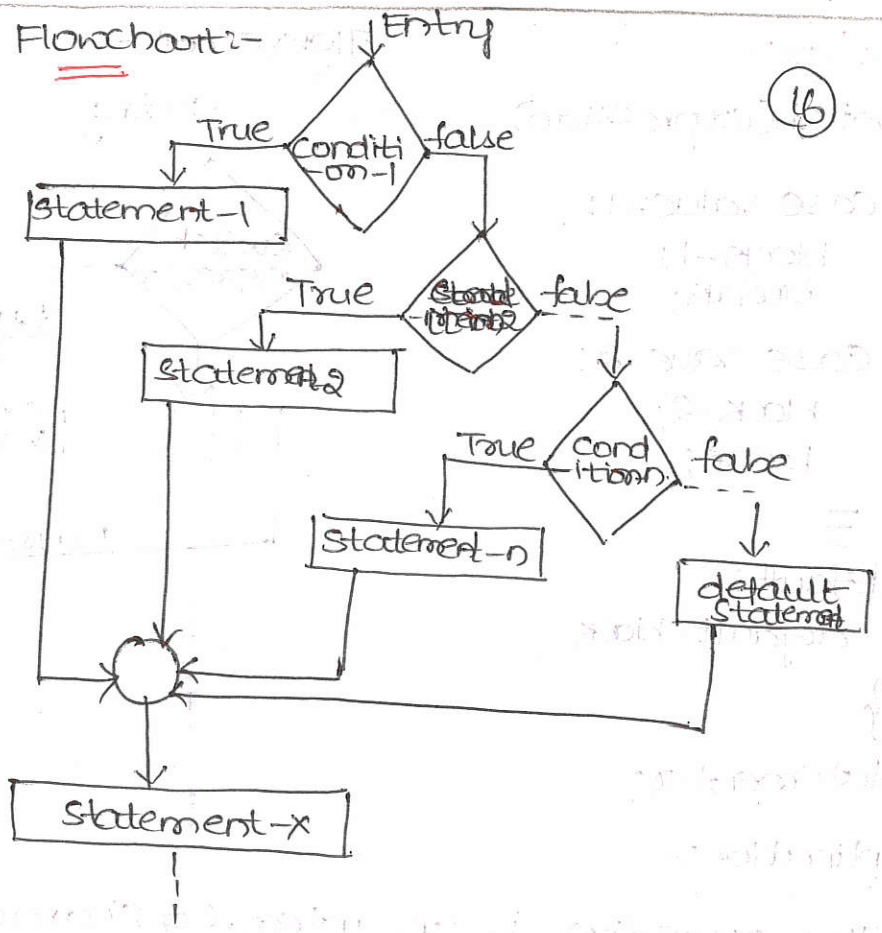
ELSE-if-ladder:-

The another way of putting ifs together when multipath decisions are involved.

Syntax:-

```
if(condition-1)
Statement-1;
else if(condition-2)
Statement-2;
else if(condition-n)
Statement-n;
else
default_statement;
Statement-x;
```

Flowchart:-



16

Explination:-

→ In this statements as soon as condition is evaluated the statements associated with it is excuted by skipping rest of ladder along statement-x.

SWITCH STATEMENTS:-

→ When one of the many alternatives is to be selected, if-statements are used with this the program becomes difficult to read and follow.

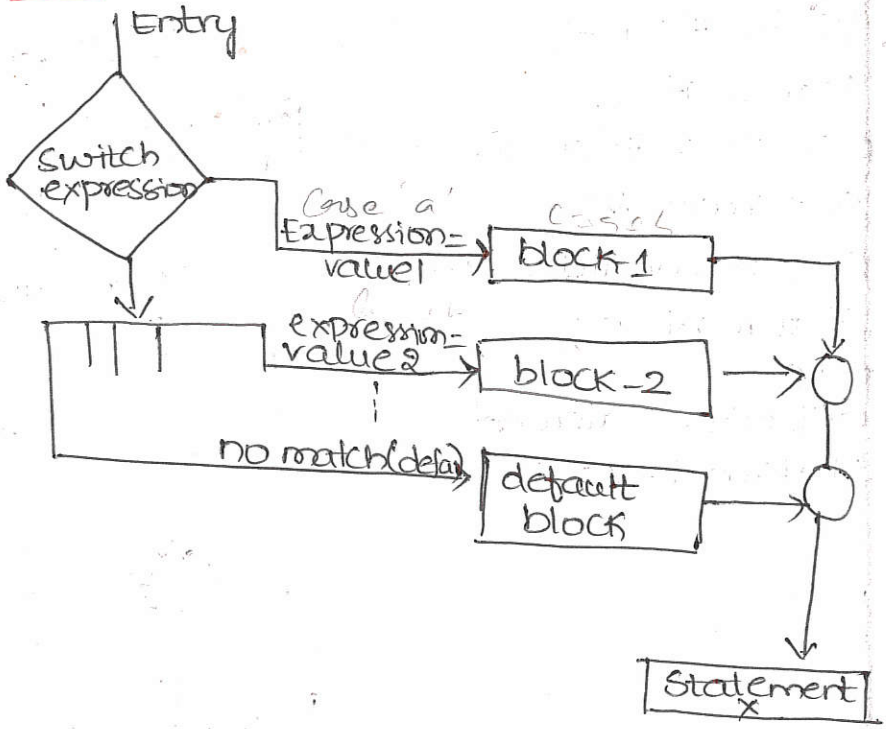
→ 'C' has a built-in multi-way decision statements known as switch.

→ These statements test the value of a given variable against a list of case-values and when match is found a block of statements associated with that case is executed.

Syntax:-

```
switch(expression)
{
 case value-1:
    block-1;
    break;
  case value-2:
    block-2;
    break;
   =
  default:
    default-block
 }
statement-x;
```

Flow-chart:-



Explination:-

→ The expression is an integer(or)characters, values(or)constants (or) constant expressions. and are known as case-labels.

→ When switch is executed, value of expression is succefully compared against case-value and particular block is executed otherwise default block is executed.

Rules for Switch-case:

→ Switch expression must be an integral type.

→ case labels must be unique(no two labels have same values) and must be constant (integer/character).

→ case labels end with colon(:)

→ A break statement is used to control the switch statements.

→ The default-statement is optional, we may also have nested switch statements.

DECISION MAKING AND LOOPING:-

In the looping control, statements are executed until some conditions for termination of the loop are satisfied.

→ Depending on position of control statements in a loop, control structures are classified into

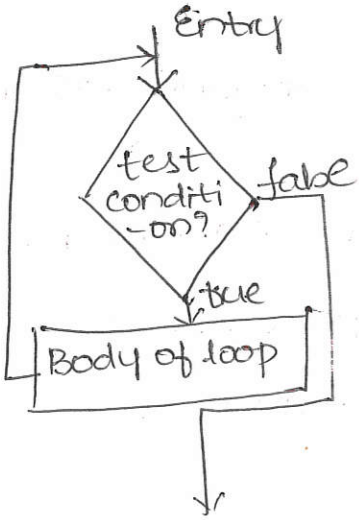
① Entry-controlled type
② Exit-controlled type (and)

Entry-controlled and Exit-controlled structures:-

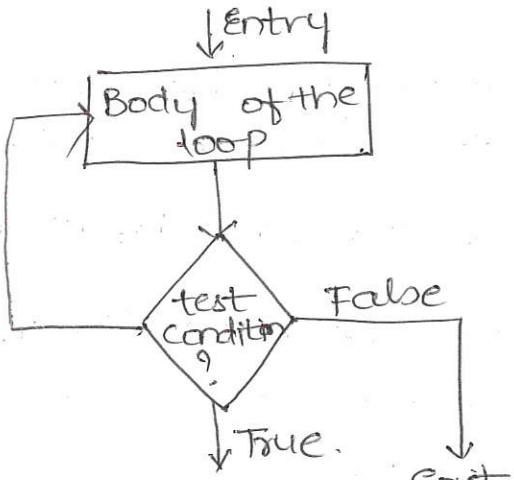
The entry-controlled structures and exit control structures are known as pre-test loops (and) post-test-loop.

→ In entry-controlled structures, the control conditions are tested before the start of the loop execution, if conditions are not satisfied body of loop will not be executed.

→ In exit-controlled structures, the body of loop is executed atleast once and then test condition is performed



Entry-controlled loop



Exit-controlled loop

Looping statements in general would include:-

① Setting and intialisation of a condition variable
② Execution of statements in loop.
③ Test for specified value of condition variable
④ Incrementing (or) updating the condition variable.

The goto statement:- (unconditional branching statements)

(17)

→ Goto statements are used to branch unconditionally from one-point to another point in a program.

→ It is not essential to use gotostatement in a highly structured language.

→ The goto requires a label in order to identify the place where branch is to be made.

→ A label is any valid variable name, and must be followed by colon.

→ The label is placed immediately at statement where control is to be transfered.

```
goto label;
  - - -
  - - -
label:
statement;
```

(forward jump)

```
label:
Statement;
- - -
- - -
goto label;
```

(backward jump)

→ The label: can be anywhere in the program either before (or) after goto label; statement.

→ The goto statement break normal execution of program.

→ If label: is before the statement goto label; a loop is formed and statements executed repeatedly, called as a backward jump.

→ If label: is placed after the statement goto label; some statements are skipped this is called forward jump.

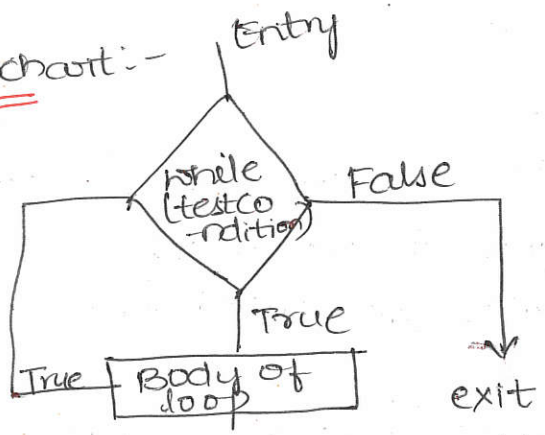
C- language provides for three constructs for performing loop operations. They are

(18)

(i) The while-statement
(ii) The do-statement (and)
(iii) The for-statement

The while-statement:-

Flowchart:-



Syntax:-

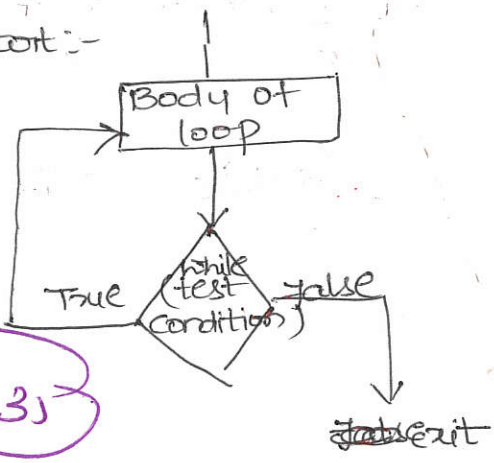
```
while (test_condition)
{ body of loop;
}
```

Explination:-

→ The while is an entry-controlled loop statements.
→ In this the test condition is evaluated and if condition is true, then the body of the loop is executed.
→ This process of execution is repeated till the test-conditions are failed.
→ Loop may have one(or) more statements.

The do-statement:-

Flow-chart:-



Syntax:-

```
do
{ body of the loop;
}
while (test-condition);
```

Page 3)

Explination:-

→ The do-while is an exit-controlled loop statements.
→ In this the body of loop is executed atleast once, and the test condition is performed.
→ This process is repeated until conditions are failed.

The for-Statements:-

→ The for-Statements are another entry-controlled structures

Syntax:-

for(intilization; test-condition; re-evaluation parameter).

Explination:-

Step1:-

— Intilisation of the control variables is done first, using assignment statements such as [i=1, - -] and these variables are called loop-control variables.

Step2:-

— The value of control variable is tested using test-condition if condition is satisfied loop is executed, otherwise loop is terminated.

Step3:-

— When body of loop is executed the control is transfered back to the for statement and reevaluation parameter is generated (either increment/decrement) then the new value is assigned to control variable, and again procedure is followed

the loop is executed till condition fails.

/* We can also do nesting of for-statements */