PRINCIPLES OF PROGRAMMING USING C

Course Objectives:

- To learn the fundamentals of computers.
- To understand the various steps in program development.
- To learn the syntax and semantics of C programming language.
- To learn the usage of structured programming approach in solving problems.

Course Outcomes:

At the end of the Course the students will be able to:

- Apply the fundamentals of computer and programming language, to draw flow chart, algorithm to solve given program.
- Comprehend the general structure of C program using control structures, functions, recursion to support reusability.
- Apply searching and sorting algorithms for the given list of elements
- Design an application to solve real world problem**.**

Module- 1: INTRODUCTION

Introduction to components of a computer: Memory, processor, I/O Devices, storage, operating system; Concept of assembler, compiler, interpreter, loader and linker.
Idea of Algorithms: Algorithms, Flowcharts, Pseudo code with examples, from algorithm to Programs and Source Code.
Introduction to C Programming Language: History of C, Basic structure of a C program, Process of compiling and running a C program; C Tokens: Keywords, Identifiers, Constants, Strings, Special symbols, Variables, Data types; Operators, Precedence of Operators, Expression evaluation, Formatted Input/Output functions, Type Conversion and type casting.

Module-2: CONTROL STRUCTURES

Decision Making Statements: Simple if, if-else, else if ladder, Nested if, switch case statement;
Unconditional Control Structures: break, continue and goto statements.
Loop control statements: for, while and do while loops, nested loops.

Module-3: ARRAYS AND FUNCTIONS

Arrays: Introduction, Single dimensional array and multi-dimensional array: declaration, initialization, accessing elements of an array; Operations on arrays: traversal, reverse, insertion, deletion, merge, search; **Strings:** Arrays of characters, Reading and writing strings, String handling functions, Operations on strings; array of strings.
Functions: Concept of user defined functions, Function declaration, return statement, Function prototype, Types of functions, Inter function communication, Function calls, Parameter passing mechanisms; Recursion; Passing arrays to functions, passing strings to functions; Storage classes.

Module-4: POINTER AND STRUCTURES

Pointers: Basics of pointers, Pointer arithmetic, pointer to pointers, array of pointers, Generic pointers, Null pointers, Pointers as functions arguments, Functions returning pointers; Dynamic memory allocation.
Structures: Structure definition, initialization, structure members, nested structures, arrays of structures, structures and functions, structures and pointers, self-referential structures; Unions: Union definition, initialization, accessing union members; bit fields, typedef, enumerations, Preprocessor directives.

Module-5: FILE HANDLING AND APPLICATIONS IN C

Preprocessor: Commonly used Preprocessor commands like include, define, undef, if, ifdef, ifndef
File Handling: Concept of a file, text files and binary files, streams, standard I/O, formatted I/O, file I/O operations, error handling, Line I/O, miscellaneous functions; Applications in C.

TEXT BOOKS:

1. Byron Gottfried, “Programming with C”, Schaum's Outlines Series, McGraw Hill Education, 3rd Edition, 2017.

2. Reeema Thareja, “Programming in C”, Oxford university press, 2nd Edition, 2016.

UNIT -I

INTRODUTION TO COMPONENTS OF A COMPUTER SYSTEM

INTRODUCTION TO PROGRAME

A Programe is a set of instructions given to the computer to do various tasks what the user wishes too. The instructions can be in the form of commands also. A Program(or) Instructions (or) commands are given by the user to computer to fulfill the desired task. As computer cannot understand human language so it needs the translators like compiler, interpreter, assembler to convert the human language of instructions to binary.

1.1 Introduction to Components of a Computer Systems:

Definition Of Computer:

- It is a high speed electronic device .
- It helps to store large volumes of data safely and securely & retrieves it whenever required by the user.
- It receives input from user through input devices, process it, and sends output through output devices to user again

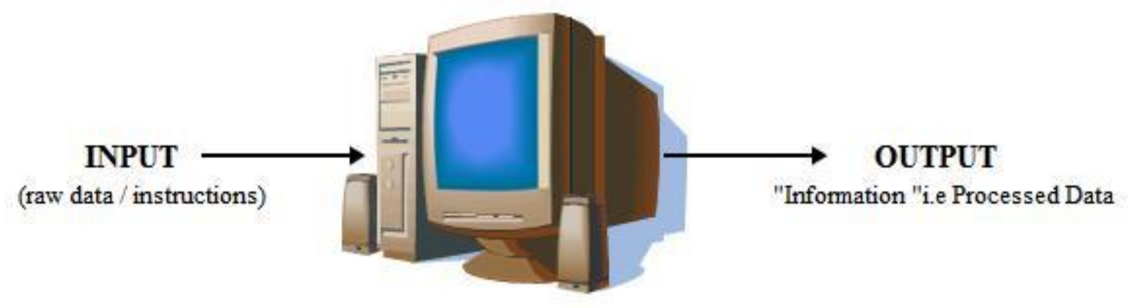


Figure 1.1: Computer System

Computer Systems :

- It is made up of 2 major components They are:

  1) Hardware 2) Software

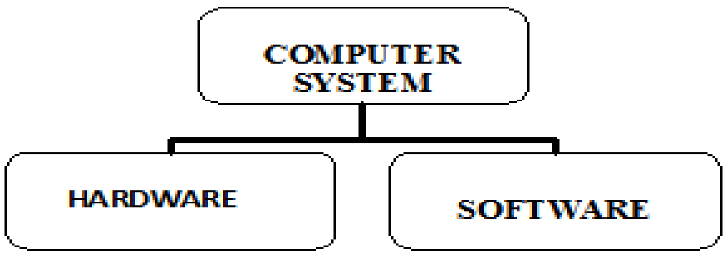


Figure 1.2: Components of Computer System

Computer Hardware :

- These are the physical components of a computer.

- It consists of 5 Parts
    1. Input Devices
    2. Output Devices
    3. CPU
    4. Primary Storage
    5. Secondary Storage

- These are the physical components of a computer It consists of 5 Parts
    1. Input Devices
    2. Output Devices
    3. CPU
    4. Primary Storage
    5. Secondary Storage

Block diagram of a Computer :

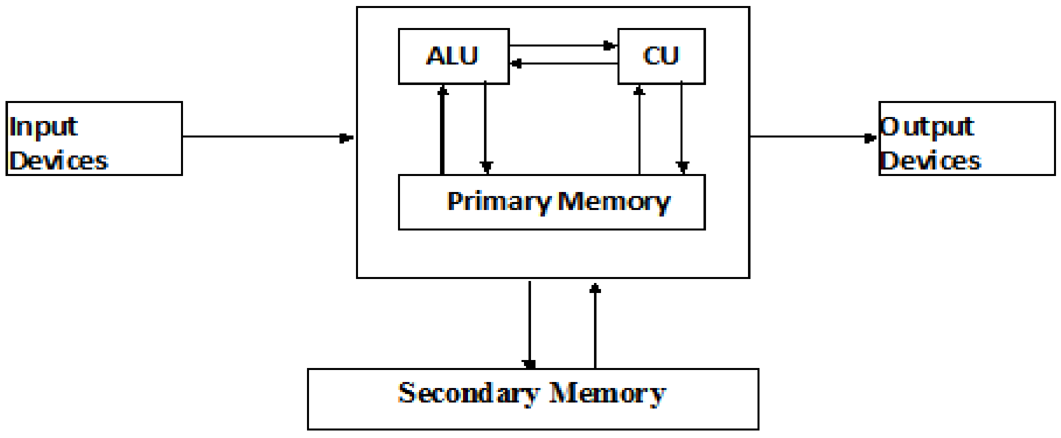


Figure 1.3: Block Diagram of Computer

Input Devices

- These are used to enter data and programs into the computer
- These are for man to machine communication

Ex: Keyboard, mouse, scanner, touch screen, audio input, Digitizer, light pen, joysticks, Track ball, Space ball, z-mouse, Digital Camera, Microphone, finger print reader, barcode reader, Data Glove, web cam,

Output Devices

- These are used to get the desired output from the computer
- These are for machine to man communication
- Ex: Printer, Monitor, Speakers, projector, disk drive, flash drive
- If the output is shown on monitor then it is called “Soft copy”
- If the output is printed on a paper using printer then it is called “**hard copy**”

CPU (Central Processing Unit)

- It is responsible for processing the instructions
- It consists of 3 parts
  1. ALU – Arithmetic & Logic Unit
  2. CU- Control Unit
  3. Memory
- **ALU** performs arithmetic operations like addition, subtraction, multiplication, division and logical operations like comparisons among data.
- **CU** is responsible for movement of data inside the system

1.1.1 Disks

Figure 1.4: Disks

A **disk** is a hard or floppy round, flat, and magnetic platter capable of having information read from and written to it. The most commonly found disks with a computer are the hard disks and floppy disks (floppy diskette) shown in the picture to the right.

Hard drives

A hard-disk drive (HDD) is a non-volatile device used for storage, located inside the computer case. Like the floppy drive, it holds its data on rotating platters with a magnetic upper exterior which are changed or read by electromagnetic tipped arms that move over the disk as it spins.

1.1.2 Memory

A memory is just like a human brain. It is used to store data and instructions. The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory locations. The address of these locations varies from 0 to 65535.

- **Memory** is used for storage of data and programs. It is divided into 2 parts.

  1. Primary Memory/ Main Memory
  2. Secondary Memory/ Auxiliary Memory
  3. Cache Memory

1. **Primary Memory :**
   - It is also called main memory or volatile memory.
   - Data is stored temporarily i.e. data gets erased when computer is turned off.
   - Faster than secondary memories.
   - A computer cannot run without the primary memory
   - Ex: RAM , ROM

Figure 1.5: RAM (Random Access Memory)

i) RAM (Random Access Memory)

- It is the internal memory of the CPU for storing data, program, and program result.
- It is a read/write memory which stores data until the machine is working.
- once the machine is switched off, data is erased.

RAM is of two types −

a) Static RAM (SRAM)
b) Dynamic RAM (DRAM)

a) SRAM (static RAM) : The word **static** indicates that the memory retains its contents as long as power is being supplied.

Characteristic of Static RAM

- Long life
- No need to refresh
- Faster
- Used as cache memory
- Large size
- Expensive
- High power consumption

b) DRAM (Dynamic Random Access Memory)

- DRAM is used for most system memory as it is cheap and small.
- All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

Characteristics of Dynamic RAM

- Short data lifetime
- Needs to be refreshed continuously
- Slower as compared to SRAM
- Used as RAM
- Smaller in size
- Less expensive
- Less power consumption

ii) **ROM (READ ONLY MEMORY)**

- The memory from which we can only read but cannot write on it.
- This type of memory is non-volatile.

- The information is stored permanently in such memories during manufacture.
- A ROM stores such instructions that are required to start a computer.
- This operation is referred to as **bootstrap**.
- ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.

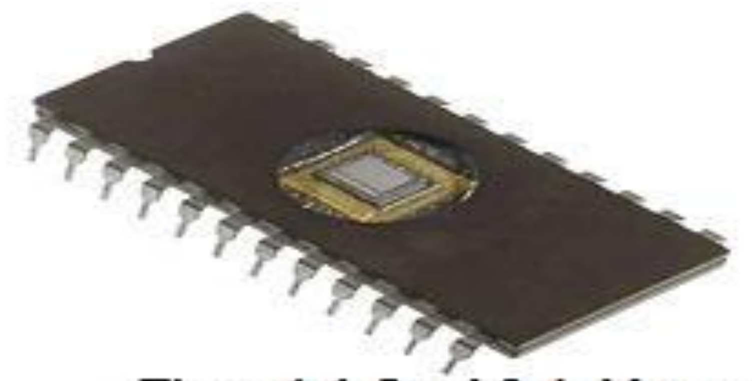


Figure 1.6: *Read-Only Memory*

Types of ROM Chips :

1. **PROM** - (Programmable Read only memory).
2. **EPROM** - (Erasable Programmable Read only Memory).
3. **EEPROM -** (Electrically Erasable Programmable Read only Memory).

Advantages of ROM :

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

Difference between RAM & ROM

S.no	RAM	ROM
1.	Stands for *Random-Access Memory*	Stands for *Read-only memory*
2.	RAM is a read and write memory	ROM is read only memory and it can not be overwritten. EPROMs can be reprogrammed
3.	RAM is faster	ROM is relatively slower than RAM
4.	RAM is a **volatile memory**. It means that the data in RAM will be lost if power supply is cut-off	ROM is permanent memory. Data in ROM will stay as it is even if we remove the power-supply
5.	There are mainly two types of RAM; *static RAM* and *Dynamic RAM*	There are several types of ROM; Erasable ROM, Programmable ROM, EPROM etc.
6.	RAM stores all the applications and data when the computer is up and running	ROM usually stores instructions that are required for starting (booting) the computer
7.	Price of RAM is comparatively high	ROM chips are comparatively cheaper
8.	RAM chips are bigger in size	ROM chips are smaller in size
9.	Processor can directly access the content of RAM	Content of ROM are usually first transferred to RAM and then accessed by processor. This is done in order to be able to access ROM content at a faster speed.
10.	RAM is often installed with large storage.	Storage capacity of ROM installed in a computer is much lesser than RAM

Table 1.1: *Difference between RAM and ROM*

2. Secondary Memory

- It is also called as auxiliary memory or non volatile memory.
- Data is stored permanently so that user can reuse the data even after power loss.
- It is known as the backup memory.
- Data is permanently stored even if power is switched off.
- Slower than primary memories

  Ex: Hard disk, CD, DVD, Floppy etc.

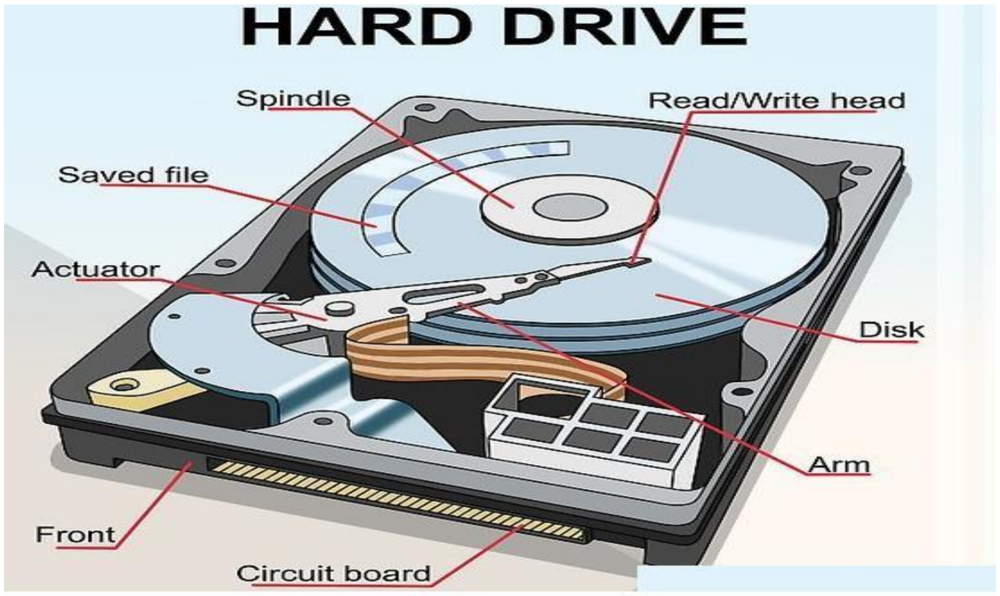


Figure 1.7: HardDisk

3. Cache memory

- Cache memory is a fastest & cheapest memory.
- Cache memory is the smallest volatile memory
- It is placed between the CPU and main memory, to operate at the speed of the CPU.
- It is also called as CPU's memory.
- It stores or fetches the frequently repeated instructions or data.

Cache memory is located in two general locations

✓ inside the processor (internal cache) and

✓ on the motherboard (external cache)

Advantages

The advantages of cache memory are as follows −

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.

Disadvantages

The disadvantages of cache memory are as follows −

- Cache memory has limited capacity.
- It is very expensive.

Difference between Primary Memory & Secondary Memory

Subject	**Primary Memory**	**Secondary Memory**
Volatility	This is volatile	This is non volatile
Access time	Access time is higher than secondary memory	Access time is lower than primary memory
Memory status	This is Temporary memory	This is Permanent memory
Capacity	Storage capacity is less	Storage capacity is more
Price	Higher than HDD or secondary memory	Lower than primary memory
Connection	Connected via slots	Connected via cables
Accessible	It is directly accessible to the CPU	It is not directly accessible to the CPU

Table 1.2: *Difference between Primary Memory and secondary memory*

1.1.3 PROCESSOR :

- A processor is an intExrated electronic circuit that performs the calculations that run a computer.
- A processor performs arithmetical, logical, input/output (I/O) and other basic instructions that are passed from an operating system (OS).
- The processor is sometimes referred to as the **Central Processing Unit (CPU).**

1.1.4 COMPUTER SOFTWARE:

- Software is defined as intangible parts of computer.

- Software is a collection of programs.
- Software makes the hardware of the computer to function properly
- Computer software is divided into 2 catExories.
    1. System software
    2. Application software

1. **System Software*:***
    - They constitute set of programs that manage the hardware resources of a computer
    - It is divided into 3 types.

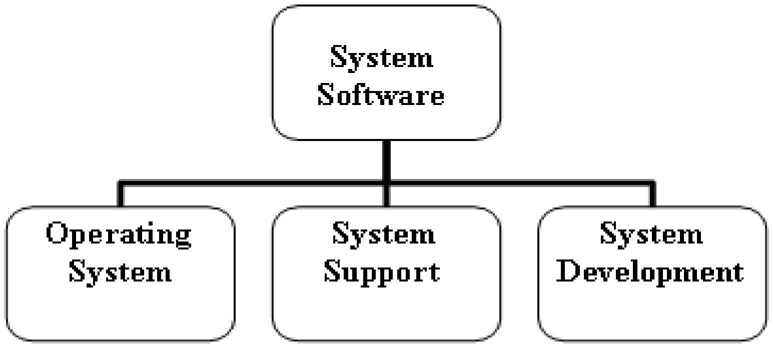


Figure 1.8 :*Classification of System Software*

Operating System

- It acts as an interface between the user and the hardware
- It makes the system to operate in an efficient manner. Ex: MS DOS, Windows, UNIX, LINUX, etc.

System Support

- They provide some utilities and other operating services
- Ex: Sort utilities, disk formatting utilities, Security monitors

System Development

It includes

a) Language translators – Used for converting programs into machine language
b) Debuggers – for error free programs
c) CASE tools – Computer Assisted Software Engineering Tools

2. Application Software

- It contains programs for performing tasks
- It is divided into 2 classes

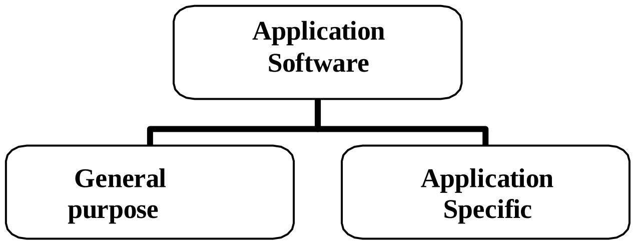


Figure 1.9: *Classification of Application Software*

General purpose software

- These are purchased from software developers
- They can be used for more than one application
- Ex: word processors, DBMS etc…

Application Specific Software

- It can be used only for intended purpose i.e. for which they were designed
- Ex: Accounting system, hospital management system, banking system. Etc.

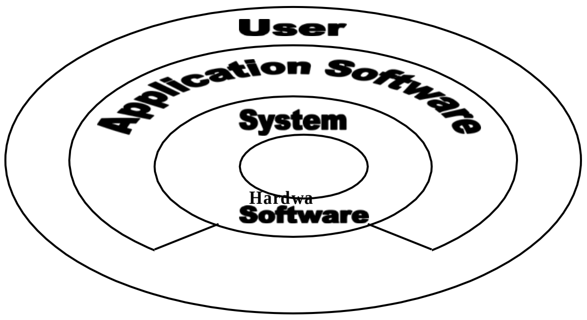


Figure 1.10: *Relationship among Application and System Software*

1.1.5. COMPILERS

- It is a Translator.
- A compiler is a software program which is used to convert human readable code into machine understandable code.(Or)
- It’s a computer program which reads source code and outputs assembly code or executable code is

called compiler.

- It Scans the entire program once and translates it as a whole into machine code.
- It reduces the execution time, it is the fastest translator.
- It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
- Programming language like C, C++ use compilers.

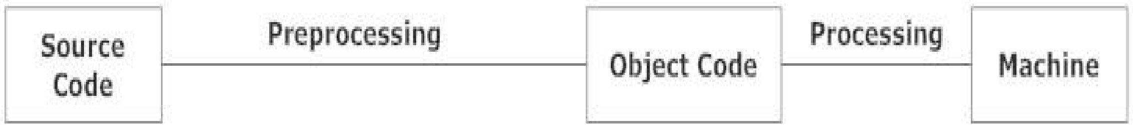


Figure 1.11: *Compiler*

1.1.6 COMPILING AND EXECUTING A PROGRAM / CREATING AND RUNNING PROGRAMS

- Program consists of set of instructions written in a programming language
- The job of a programmer is to write and test the program.
- There are **4 steps** for converting a 'C' program into machine language.
  1) Writing and editing the program
  2) Compiling the program
  3) Linking the program
  4) Executing the program

1) Writing and editing the program

- '**Text editors**' are used to write programs.
- Users can enter, change and store character data using text editors
- Special text editor is often included with a compiler
- After writing the program, the file is saved to disk. It is known as **'sourcefile'**
- This file is input to the compiler

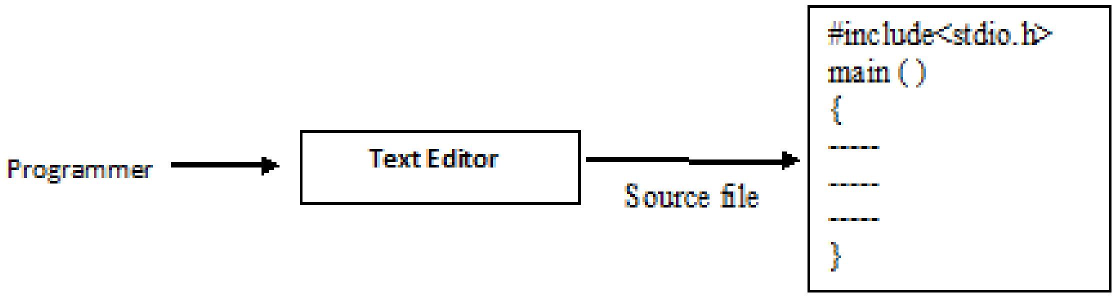


Figure 1.12: *Writing and editing the program*

2) Compiling the program

- "**Compiler**" is a software that translates the source file into machine language
- The 'C' compiler is actually 2 separate programs

  a) Preprocessor

  b) Translator

a) Preprocessor

- It reads the source code and prepares it for the translator
- It scans for special instructions known as 'preprocessor' commands which start with ' #' symbol
- These commands tell the preprocessor to look for special code libraries and make substitutions
- The result of preprocessing is called 'translation' unit

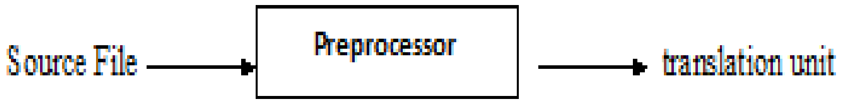


Figure 1.13: Preprocessor

b) Translator

- It does the actual work of converting the program into machine language
- It reads the translation unit and results in 'object module' i.e., code in machine language
- But it is not yet executable because it does not have the 'C' and other functions included.

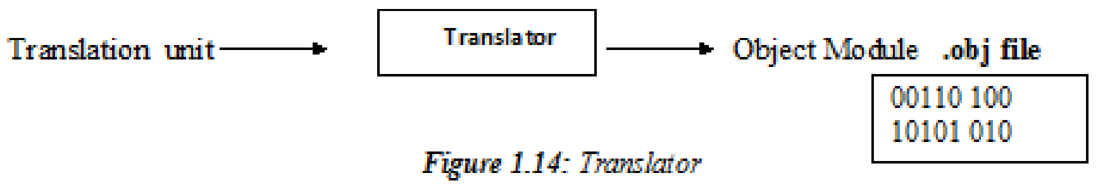


Figure 1.14: Translator

3) Linking programs

- 'Linker' assembles input /output functions, mathematical library functions and some of the functions that are part of source program into final executable program. It is called as executable file that it is ready for execution.

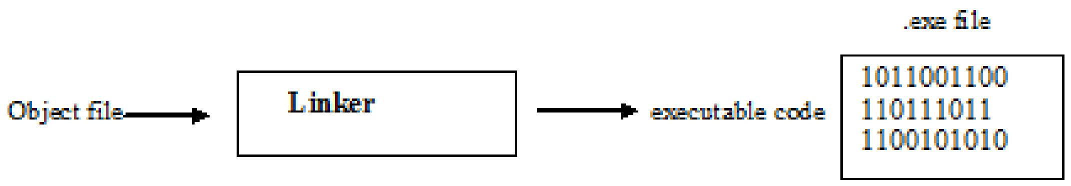


Figure 1.15: Linker

4) Executing Programs

- '**Loader**' is the software that gets the program that is ready for execution into the memory
- When everything is loaded, the program takes control and the '**Runner**' bExins its execution.
- In the process of execution, the program reads the data from the user, processes the data and prepares the output.

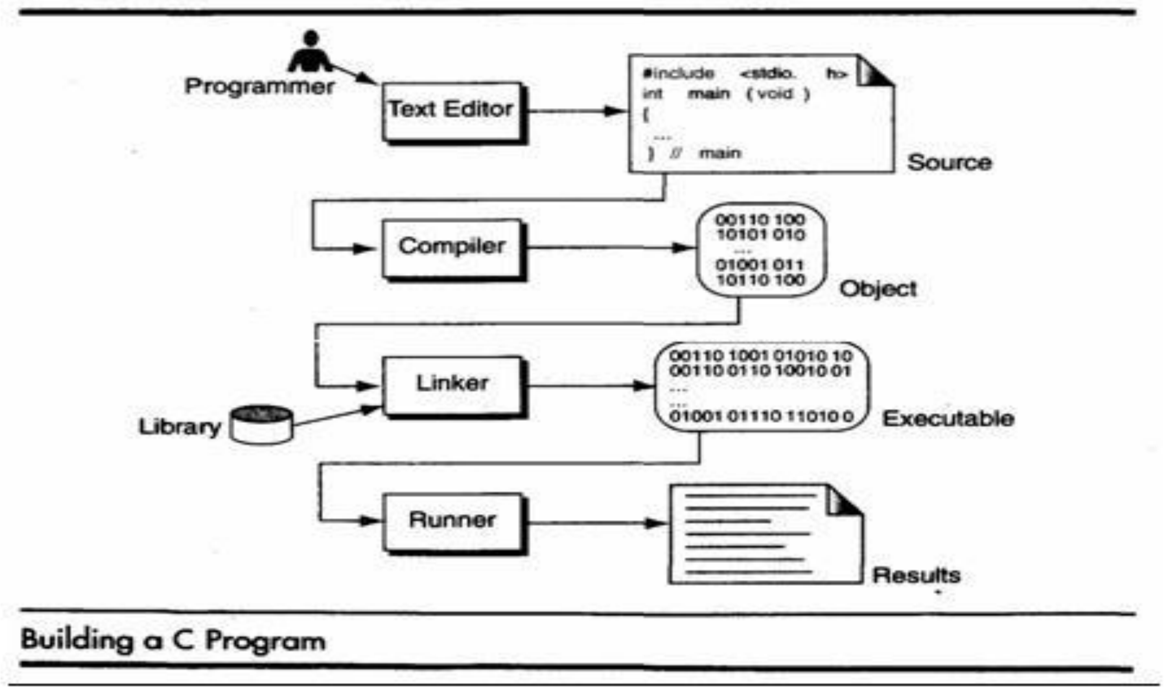


Figure 1.16: *Creating and Running a Program*

1.2 Introduction to Algorithms:

- It is a step – by – step procedure for solving a problem
- If algorithm is written in English like sentences then it is called as ‘PSEUDO CODE’

1.2.1 Properties of an Algorithm

An algorithm must possess the following 5 properties. They are

1. Input
2. Output
3. Finiteness
4. Definiteness
5. Effectiveness

1. **Input :** An algorithm must have zero (or) more number of inputs
2. **Output:** Algorithm must produce one (or) more number of outputs
3. **Finiteness :** An algorithm must terminate in countable number of steps
4. **Definiteness:** Each step of the algorithm must be stated clearly
5. **Effectiveness:** Each step of the algorithm must be easily convertible into program statements

1.2.2 Frequently used mnemonics in algorithm:

- Start = this label in algorithm indicates starting the task
- Repeat =do the step, logic or process for some time till condition gets failed
- goto=it states that control should go to a specific label and then start its execution
- break = it’s a termination point i.e control has to come out of the block or loop where the break is specified
- Continue = When a continue statement is encountered inside a loop, control jumps to the bExinning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.
- If-then-else= this decision making statement is used when we have two choices for selection
- Stop=This indicates end of algorithm

1.2.3 Examples on Algorithms

Example-1

Algorithm for finding the average of 3 numbers

1. start
2. Read 3 numbers a,b,c
3. Compute sum = a+b+c
4. compute avg = sum/3
5. Print avg value
6. Stop

Example 2: *Convert Temperature from Fahrenheit (℉) to Celsius (℃)*

Algorithm:

Step 1: Read temperature in Fahrenheit,

Step 2: Calculate temperature with formula C=5/9*(F-32),

Step 3: Print C,

1.2.4 Introduction to Flowchart

Diagrammatic or pictorial representation of an algorithm is called flow chart

1.2.4.1 Advantages and Disadvantages of Flowchart

➢ It is very easy to understand because the reader follows the process quickly from the flowchart instead of going through the text.

➢ It is the best way of representing the sequence of steps in an algorithm

➢ It gives a clear idea about the problem

➢ Various symbols are used for representing different operations

➢ Arrows are used for connecting the symbols and show the flow of execution

Disadvantage :

1. Drawing flowchart for complex logic programs is difficult.

2. If alterations are required the flowchart may require re-drawing completely. This will usually waste valuable time.

1.2.4.2 Symbols used in flowchart

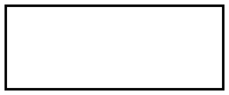
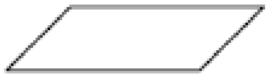
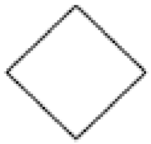
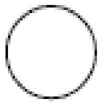
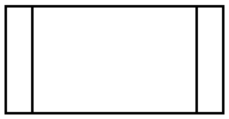
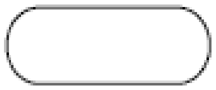
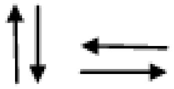
Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

Table 1.17: *List of symbols used in drawing flowchart*

1.2.4.3 Examples on Flowcharts

Example 1:

Flowchart for finding the average of 3 numbers

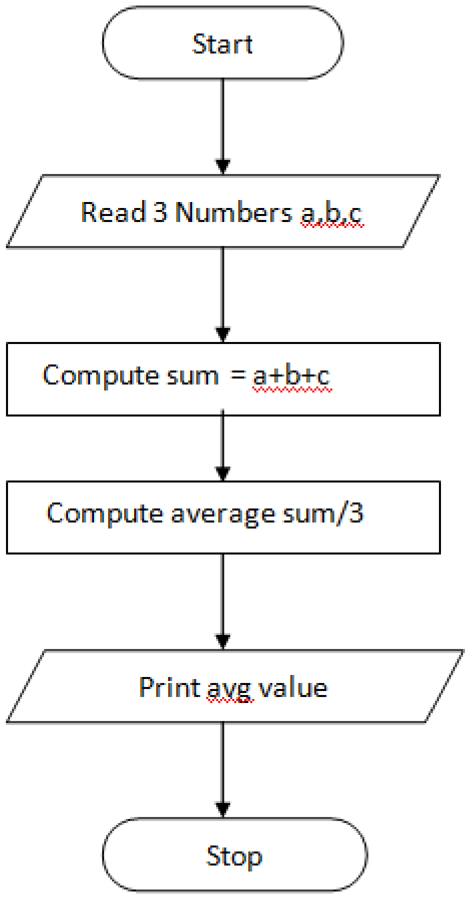


Figure 1.17: *Flowchart for finding the average of 3 numbers*

Example 2: *Flow chart for converting foreign heat to centigrade :*

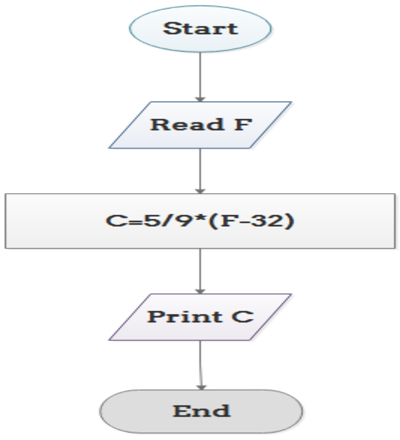


Figure 1.18: *Flowchart for converting foreign heat to centigrade*

1.2.5 Introduction to Pseudo code :

Pseudo code is an artificial and informal language that helps programmers develops algorithms. Pseudo code is similar to everyday English, it is convenient and user friendly although it is not an actual computer programming language. Pseudo code programs are not actually executed on computer rather they merely help the programmer "think out" a program before attempting to write it in a programming language such as C.

1.2.5.1 Advantages and Disadventages of Pseudocode

Advantages:

- It can be written easily.
- It can be read and understood easily.

Disadvantages:

- It is not visual.
- We do not get a picture of the design

1.2.5.2 Example on Pseudocode

Example 1 : Pseudo Code to print Fibonacci series upto 50 terms

```
1. Declare an integer variable called n
2. Declare an integer variable sum
3. Declare an integer variable f1
4. Declare an integer variable f2
5. set sum to 0
6. set f1 and f2 to 1
7. set n to 50
8. repeat n times
      a. sum = f1 + f2
      b. f2 = f1
      c. f1 = sum
      d. print sum
9. end loop
```

1.2.6 Program design and structured programming

- Programming is a problem solving task / activity

- Programmers use the software development method for solving problems
- It consists of the following **6 phases**
    1) Requirements
    2) Analysis
    3) Design
    4) Coding
    5) Testing
    6) Maintenance

❖ Requirements

- Information about the problem must be stated clearly and unambiguously
- Main objective of this phase is to eliminate unimportant aspects and identify the root problem

❖ Analysis

- It involves identifying the problem inputs, outputs, that the program must produce
- It also determines the required format in which results should be displayed

❖ Design

- It involves designing algorithms, flowcharts (or) GUI's (Graphical User Interfaces)
- Designing the 'algorithm' is to develop a list of steps called algorithm to solve the problem and then verify that the algorithm solves the problem intended.
- "Top – down design" is followed i.e. list the major steps (or) sub problems that need to be solved
- "Flow charts" are used to get the pictorial representation of the algorithm.
- Algorithm for a programming problem consists of at least the following sub problems
    1. Get the data
    2. Perform the computations
    3. Display the results

❖ Coding / Implementation

- This step involves writing algorithm as a program by selecting any one of the high – level languages that is suitable for the problem.

- Each step of the algorithm is converted into one (or) more statements in a programming language.

❖ Testing

- Checking / verifying whether the completed program works as desired is called " Testing"
- Running the program several times using different sets of data verifies whether a program works correctly for every situation provided in the algorithm.
- After testing, the program must be free from the following errors.

  a) Syntax errors

  b) Logical errors

  c) Run-time errors

❖ **Maintenance**

- It involves modifying a program to remove previously undetected errors and to keep it up-to-date as government rExulations (or) company polices change.
- Many organizations maintain a program for some period of time i.e. 5 years.

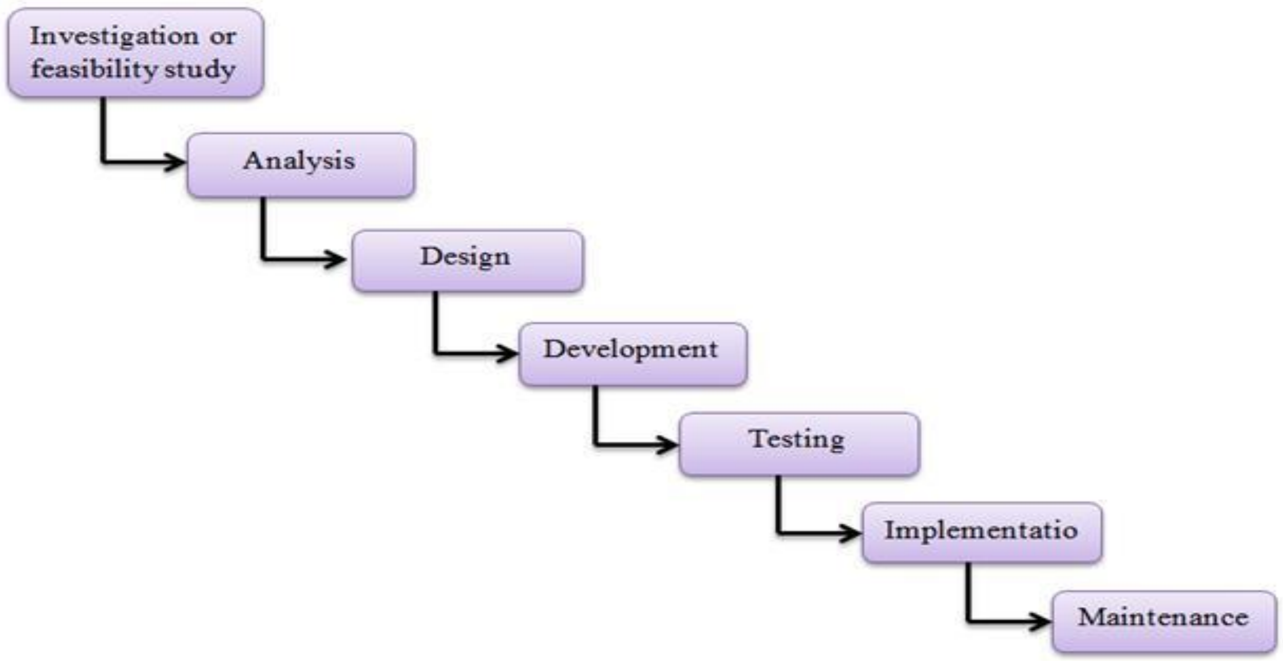


Figure 1.19: *Water fall model*

1.3 INTRODUCTION TO 'C' PROGRAMMING LANGUAGE

- 'C' is a high –level programming language developed in **1972 by Dennis Ritchie at AT & T Bell Laboratories**

- It was primarily developed for systems programming i.e. for designing operating systems, compilers etc

1.3.1 Importance of ’C’ Language

1. It is a **robust language**, whose rich set of built-in functions and operations can be used to write any complex program
2. The ‘C’ compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages.
3. ‘C’ Programs are **efficient and fast, this is because it has variety of data types and powerful operators**
4. C is highly **portable**, that is c programs written for one computer can be run on another with little (or) no modification.
5. ‘C’ Language is well suited for **structured programming**, thus requiring the user to think of a problem in terms of function modules (or) blocks. The proper collection of these module make a complete program. This modular structure makes program debugging, testing and maintenance easier.
6. ‘C’ program has the **ability to extend itself**. A 'c' program is basically a collection of functions that are supported by the 'C' library. we can add our own functions to 'C' library..
7. 'C' is highly portable .I.e. 'C' programs written for one computer can be run on another with little or no modifications .portability is important if we to run a program on a new computer with different operating system. I.E "C " is a machine independent language.
8. 'C' is high-level language which allows the programmer to concentrate on the problem with out worry about the machine that program will be using.

1.3.2 Data Types

Data Types

❖ Data type specifies the set of values and the type of data that can be stored in a variable.

❖ They allow the programmer to select the type appropriate to the needs of application.

Types: ANSI supports 3 classes of data types namely

Primary data type/fundamental data type/ Basic data type

1) Derived data type
2) User-defined data type

1. Primary data types

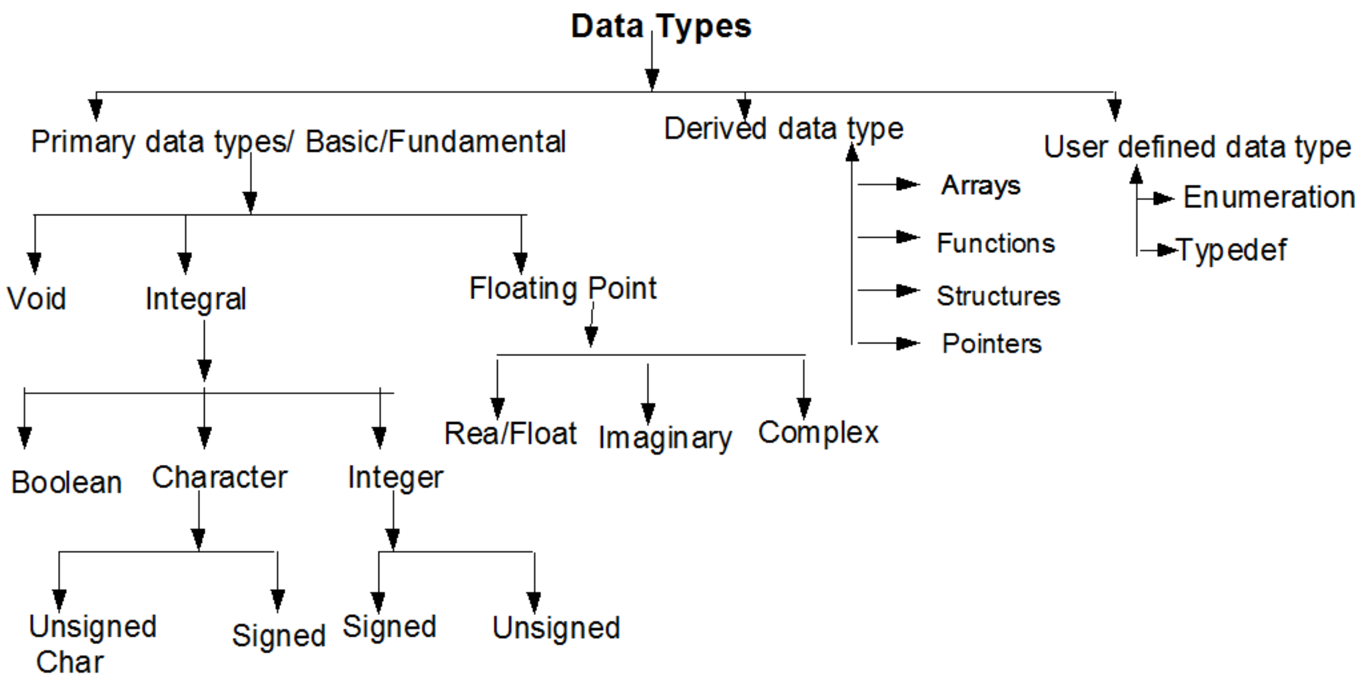


Figure 1.20: *Data Types*

Void:

- Designated by the keyword ' void '. The void type has no values.
- It is used to specify the type of a functions. The type of functions will be void if it not returning any value to the calling functions.
- It is also used to define a pointer to generic data.
- It is used to designate that a function has no parameters as in main function.

Integral type:

Integral data types are used to store whole numbers and characters

C have 3 Integral types namely

i) Boolean

ii) Character

iii) Integer

i) Boolean:

- This is named after French mathematician "George Boole".
- It can represent two values namely true and false. It can be in memory as 0(false) and 1(true).

ii) Character data type

- A single character can be defined as a character (Char)type data.

- These characters are internally stored as integers
- Each character has an equivalent ASCII value
- This data type is used to store character

iii) Integer:

- An integer type is a number without a fractional part or a whole number.
- In order to provide some control over the range of numbers and storage spaces C supports 4 different size of integers.

Integer Data type			
Type	**size (in bytes)**	**Range**	**Control string**
Short int (or) signed short int	1	-128 to 127	% h
Unsigned short int	1	0 to 255	%uh
int (or) signed int	2	-32768 to 32767	% d or %i
unsigned int	2	0 to 65535	% u
Long int (or) signed long int	4	-2147483648 to 2147483647	%ld
Unsigned long int	4	0 to 4294967295	%lu
long long int	8		

- short int, int and long int are in both signed and unsigned forms

Table 1.18: *Integer Data Types*

-- the function size of (data type) gives the size of the data type in bytes.
C follows the followings relation ship

sizeof(short) < sizeof(int) < sizeof(long int) < sizeof(long long int)

Floating – point Data types

'C' standard recognizes 3 floating point types

i) Real
ii) Imaginary
iii) Complex

i) Real:

- ✓ It is used to store real numbers (i.e., decimal point numbers).
- ✓ C has 3 different sizes of real types
- ✓ For 6 digits of accuracy/precision, ‘float’ is used.
- ✓ For 14 digits of accuracy, ‘double' is used.
- ✓ For more than 14 digits of accuracy, ‘long double’ is used..

the relation among the real type

sizeof (float) < sizeof (double) < sizeof (long double)

ii) **Imaginary**:

It is real number multiplied by root of -1

It has 3 different size of imaginary types

a. float imaginary
b. double imaginary
c. long double imaginary

iii) **Complex type**

--It is combination of real and an imaginary number

-- It have 3 different sizes

a. float complex
b. double complex
c. long double complex

Floating point data type			
Type	**Size (in bytes)**	**Range**	**Control string**
float	4	3.4 E – 38 to 3.4 E + 38	%f
double	8	1.7 E – 308 to 1.7 E +308	%lf
long double	10	3.4 E – 4932 to 1.1 E +4932	%Lf

Table 1.19: *Floating point Data Types*

1.3.3. Types of Errors

While writing c programs, errors (also known as bugs) may occur unwillingly which may prevent the program to compile and run correctly as per the expectation of the programmer.

Basically there are three types of errors in c programming:

1. Runtime Errors

2. Compile Errors
3. Logical Errors

1. Runtime Errors:

C runtime errors are those errors that occur during the execution of a c program and generally occur due to some illExal operation performed in the program.

Examples of some illExal operations that may produce runtime errors are:

- Dividing a number by zero
- Trying to open a file which is not created
- Lack of free memory space
  - ✓ When you instruct your computer to find logarithm of a nExative number.
  - ✓ When you instruct your computer to find the square root of a nExative integer.

Compile Errors: Compile errors are those errors that occur at the time of compilation of the program. C compile errors may be further classified as:

- o **Syntax Errors**
- o **Semantic Errors**

- **Syntax Errors:**

A collection of rules for writing programs in a programming language is known as syntax.When the rules of the c programming language are not followed, the compiler will show syntax errors.A program containing syntax error cannot be compiled successfully.

For example, consider the statement,

```
int a,b:
```

The above statement will produce syntax error as the statement is terminated with : rather than ;

Ex: A variable is used without declaration.
Example, in C, if you don’t place a semi-colon after the statement (as shown below), it results in a syntax error.
printf(“Hello,world”) – error in syntax (semicolon missing)
printf("Hello,world"); - This statement is syntactically correct.

- **Semantic Errors**

Semantic errors are reported by the compiler when the statements written in the c program are not meaningful to the compiler.

For example, consider the statement,

1b+c=a;

In the above statement we are trying to assign value of a in the value obtained by summation of b and c which has no meaning in c. The correct statement will be

1a=b+c;

- **Logical Errors**

Logical errors are the errors in the output of the program. The presence of logical errors leads to undesired or incorrect output and are caused due to error in the logic applied in the program to produce the desired output. Also, logical errors could not be detected by the compiler, and thus, programmers have to check the entire coding of a c program line by line.

1.3.4 Operators & Types of Operators:

Operators:

A operator is a symbol that tells the computer to perform certain mathematical (or) Logical manipulations.
An operator indicates the operation to be performed on data that yields a value.
C Operators can be classifieds into

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment & Decrement Operators
6. Conditional Operators
7. Bitwise Operators &
8. Special Operators

1) Arithmetic operators:

❖ These operators are used for numerical calculations (or) to perform arithmetic operations like addition, subtraction etc.

- ❖ Arithmetic Operator involving only real operands is called " Real Arithmetic"
- ❖ When one of the operands is real and other is integer then expansion is called as " Mixed –Mode arithmetic Expression ""

Operator	Description	Example	a =20, b=10	output
+	Addition	a+b	20+10	30
-	Subtraction	a-b	20-10	10
*	multiplication	a*b	20*10	200
/	Division	a/b	20/10	2 (quotient)
%	Modular division	a%b	20%10	0 (remainder)

Table 1.20: *Arithmetic Operators*

Note :

- ❖ C does not have exponentiation.

  The modulo Operator (%) can't be used on floating point data

- ❖ **Program on Arithmetic Operators:**

```
main ( )
{                                   Output
        int a= 20, b = 10;
                                      30
        printf (" %d", a+b);
                                      10
        printf (" %d", a-b);
                                      200
        printf (" %d", a*b);
                                      2
        printf (" %d", a/b);
        printf (" %d", a%b);
}
```

2) **Relational operators:**

The comparisons can be done with relational operators .

C supports six relational operators .

The output of relational expression is either one or zero (i.,e True or False).

Relational expression syntax:

```
arg1    relational_operator    arg2
```

here a**rg1 & arg2 are** **<u>arithmetic expressions</u>** which may be simple constants variables or combination of them .

The output of a relational expression is either true (1) (or) false (0)

Operator	Description	Examble	a =10, b=20	output
<	less than	a<b	10<20	1
<=	less than (or) equal to	a<=b	10< = 20	1
>	greater than	a>b	10>20	0
>=	greater than (or) equal to	a>=b	10> =20	0
= =	equal to	a= =b	10 = = 20	0
! =	not equal to	a! = b	10 ! =20	1

Table 1.21: *Relational Operators*

❖ **<u>Program to demonstrate relational operators</u>**

```
main ( )                                  Output
{

        int a= 10, b = 20;
        printf (" %d", a<b);                1
        printf (" %d", a<=b);               1
        printf (" %d", a>b);                0
        printf (" %d", a>=b);               0
        printf (" %d", a = =b);
        printf (" %d", a ! =b);
        }
```

Note:

In above six operators each one is complement of another operator

> is Complement of <=

< is complement of >=

== is complement of !=

3) Logical operators:

C has three logical operators namely

- Logical AND (&&)
- Logical OR (||)
- Logical NOT(!)

Logical Operators && and || used when we want to test more than one condition

Logical expression will yield a value i.e. either one or zero (true or false)

❖ These are used to combine 2 (or) more expressions logically

Logical AND (&&)

exp1	**exp2**	**exp1&&exp2**
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.22: *Logical AND (&&)*

exp1	**exp2**	**exp1&&exp2**
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.23: *Logical OR(||)*

exp	**!(exp)**
T	F
F	T

Table 1.24: *Logical NOT(!)*

Program To Demonstrate logical operators :

```
main ( )
{
        int a= 10, b = 20, c= 30;
        printf (" %d", (a>b) && (a<c));
        printf (" %d", (a>b) | | (a<c));
         printf (" %d", ! (a>b));
 }
```

Output

```
0
1
1
```

4) Assignment operators:

It is used to assign the result of an expression to a variable.

Types :

- Simple Assignment
- Compound Assignment

Operator	Description	Example
=	Simple assignment	a=10

+ =, - =,
* =, / =, %=
	Compound assignment	a+=10 □ a=a+10
		a-=10 □ a=a-10

***Table 1.25** Assignment operators*

General assignment operator (=) and syntax:

v operator= exp;

$\hookrightarrow$ Same as v =v operator (exp);

here v --> variable

exp--> expression

Operator--> Binary operation

example 1: x+=y

$\hookrightarrow$ x= x+y

2: x-=y

$\hookrightarrow$ x= x-y

3: x*=y+1

$\hookrightarrow$ x= x*(y+1)

4: x/=y+1

$\longrightarrow$ x= x/(y+1)

5: x%=y

$\longrightarrow$ x= x%y

Increment & Decrement operators:

The increment and decrement operators are ++ and -- respectively.

a) **Increment operator (++):**

- ❖ It is used to increment the value of a variable by 1
- ❖ 2 types : i) pre increment ii) post increment
- ❖ Increment operator is placed before the operand in preincrement and the value is first incremented and then operation is performed on it.

Ex: z = ++a; first: a=
a+1 second:
z=a

- ❖ Increment operator is placed after the operand in post increment and the value is incremented after the operation is performed

Ex: z = a++;
first: z=a
second: a= a+1

Program pre increment

```
main ( )          Program
main ( )
{
int a= 10, z;
z= ++a
printf (“z= %d”, z);
printf (“ a=%d”, a);
}
```

Program post increment

main ( ) **Program**

```
main ( )
{
        int a= 10, z;
        z= a++
        printf (“z= %d”, z);
        printf (“ a=%d”, a);
}
```

b) Decrement operator: (- -)

- ❖ It is used to decrement the values of a variable by 1

  2 types : i) pre decrement

  ii) post decrement
- ❖ decrement operator is placed before the operand in predecrement and the value is first decremented and then operation is performed on it.

Ex: z = - - a; $<$ first: a= a-1;

second: z=a ;

- ❖ decrement operator is placed after the operand in post decrement and the value is decremented after the operation is performed

Ex: z = a--; $<$ first: z=a;

second: a= a-1;

Program:

```
main ( )
{
        int a= 10, z;
        z= --a
        printf (“z= %d”, z);
        printf (“ a=%d”, a);
}
```

Output

```
z=9
a =9
```

```
main  ( )
{
        int a= 10, z;
        z= a--;
        printf (“z= %d”, z);
        printf (“a=%d”, a);
}
```

Output

```
z=10
a = 9
```

Conditional Operators:

A ternary operator pair " ? : " is used to construct conditional expressions.

Syntax:

```
exp1 ? exp2 : exp3;
```

where exp1,exp2,exp3 are expressions .

-->If exp1 is true,then exp2 is evaluated.

-->If exp1 is false,the exp3 is evaluated.

Note: Only one expression either exp2 or exp3 is evaluated.

Example: a=10 b=5

x=(a>b)?a:b;

x is assign with 'a' value since the exp1 is true.

Special operators:

C supports special operators such as:

--comma operator

--size of operator

--pointer operator(& and *)

--member selection operator(. And →)

Comma operator-

Link the related expressions together

Evaluated left to righ

Ex: value=(x=10,4=5,x+y);

Size of operator

It is a compile time operator with operand

It returns numbers of bites the operand occupies

Ex: m=size of (sum);

n=size of (int)

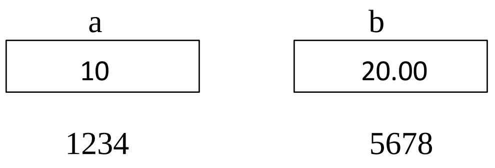
Note: to print the character %, by placing it immediately after another % character in control statement.

Ex: printf("a%%b=%d",a%b);

out put: a%b=5

Program:

```
main ( )
{
     int   a=10;
     float  b=20 ;
    printf (“ a= %d      b=%f’, a,b );
      printf (“ Adderss  of a =%u “ &a ) ;
    printf (“ Adderss of b =%u” &b ) ;
    printf (“Size of a = %d ” , sizeof (a) ) ;
    printf ( “ Size of b = %d ” . sizeof (b) ) ;
   }
```



Output :

```
int a=10   float b=20.00
Adderss  of a  =1 2 3 4
Adderss of b  = 5 6 7 8
Only for this example
Size of a = 2  bytes    Size of b = 4 bytes
```

1.3.5 Introduction to Expressions

EXPRESSIONS

An expression is sequence of operands and operators that reduces to single value.

- An operand is a data item on which an operation is performed.
- An operator indicates an operation to be performed on data

  Ex:10+5 ,this is an expression whose value is 15

1.3.5.1 Types of Expression

An expression can be a

- **Simple Expression:** Expression which contain single operator is called as simple expression Ex: 11+5; -a
- **Complex Expression:** Expression which contain more than one operator is called as complex expression. Ex:2*3+5-1

  Expression is of 6 catExories namely

1.3.5.2 Precedence And Associativity

Precedence:

The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as precedence.

Associativity :

Associativity is used to determine the order in which operators **with same precedence** are evaluated in complex expression

Associativity can be i) Left to Right. ii) Right to left .

i) **Left to Right associativity** evaluates the expression by starting on left and moving to the right side of expression.

ii) **Right to Left associativity** evaluates the expression by starting on right and moving to the left side of expression.

Side effect:

A side effect is an action that results from the evaluation of a expression..

```
Ex:    int   x=3;
             x = x+ 4;
```

here initial value of x is 3 and the value of the expression on right side of assignment is 7 (ie 3+4).so the final value of x is 7 this is side effect.

1.3.5.3 Rules for Evaluation of Expression

- ✓ First, parenthesized sub-expression from left to right are evaluated
- ✓ If parentheses are nested, the evaluation bExins with the innermost sub-expression.
- ✓ The precedence rule is applied in find the order of operators in evaluating subexpression
- ✓ The associativity rule is applied when two or more operators have same precedence in sub expression.
- ✓ Arithmetic expressions are evaluated from left to right using the rules of precedence
- ✓ When parenthesis are used, the expression with parenthesis assume highest priority.

Examples on Precedence and Associativity:

Ex 1: evaluate 2+3*4

Since * has high precedence than the + so multiplication is performed first than followed by addition

i.e. 2+ 12----□ 14

Ex 2: evaluate –b++ where b=5;

In the above expression both – and ++ are having same precedence i.e 2nd so we consider associativity for these operators ,which is left to right…hence given expression is evaluated as –(b++) which yield to 5 and value of b after completion of expression is 6(since b++ is post increment)

Ex 3: evaluate 3 * 8 / 4 % 4 * 5

In the given expression all three operators(* / %) are belonging to same precedence and their associativity is from left to right. so given expression should be evaluated from left to right

i.e. ((((3*8)/4)%4)*5)

 (((24/4)%4)*5)

((6%4)*5)

- 2*5
- 10

Ex 4: Evaluate a+=b*=c-=5 where a=2,b=3 and c=7

In the given expression all assignment operators i.e += ,*=,-= are with same precedence and have associativity from right to left . So the rightmost expression is evaluated first and the value is assigned to operand on the left of the assignment operator and next expression is evaluated.

Given expression is a+=b*=c -=5

- (a+=(b*=(c-=5)))
- (a+=(b*=(c=c-5)))
- (a+=(b*=(c=7-5)))
- (a+=(b*=2))
- (a+=(b=b*2))
- (a+=(b=3*2))
- (a+=6)
- a=a+6
- a= 2+6
- 8

After the side effect a=8,b=6 and c=2

Ex 5: Evaluate - -a * (3+b) / 2 - c++ * b, where a =3,b=4 and c=5

In the given expression () is having higher precedence so it is evaluated fist

--a*(3+4)/2-c++ *b

- -- a *7/2 -c ++ * b (in this expression - - and ++ have higher precedence than * / - and associativity is right to left.)
- --a*7/2 -5*b // post increment takes place after evaluating expression i.e after these 'c' value is 6
- 2*7/2 -5 *b // --a is pre decrement so a is decreased by 1

In the above expression operators * / are having same and higher precedence than' –' and their associativity is left to right so

- (2*7)/2 –5 *b
- 14/2 – 5*b
- 7- 5 *4 // in this expression * has high precedence than – so it is evaluate first

- ❖ 7-20
- ❖ 13

After the side effect a=2 ;b=4 and c=6

Ex6: evaluate x = a - b / 3 + c * 2 -1,where a=9,b=12,c=3

9 - 12 / 3 + 3 * 2 – 1 since / and * have same and high precedence than + and – with left to right assoiciativity .

- ❖ Step1: x=9 - 4 + 3* 2 -1
- ❖ Step2: x= 9 -4 + 6 -1 // - and + have same precedence and left to right Associativity
- ❖ Step3: x= 5+6-1
- ❖ Step4:x=11-1
- ❖ Stept5: x=10

1.3.6 Storage classes

There are 4 storage classes (or) storage class specifiers supported in 'C' they are:

- auto
- extern
- static
- rExister

1. Automatic variables / Local variables.

Automatic variables are declared inside a function in which they are to be utilized.

They are created when the function is called and destroyed automatically when the function is exited, hence the name automatic.

Automatic variables are therefore private(or local) to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

A variable declared inside a function without storage class specification is, by default, an automatic variable. For instance, the storage class of the variable **number** in the example below is automatic.

```
main()
{
        int number;
        -----------
        ----------
}
```

We may also use the keyword auto to declare automatic variables explicitly.

```
main()
{
        auto int nimber;
        -------------
        -------------
}
```

One important feature of automatic variables is that their value cannot be changed accidentally by what happens in some other functions in the program. This assures that we may declare and use the same variable in different functions in the same program.

1. External variables

They are also known as global variables. Unlike local variables, global variables can be accessed by any function in the program. External variables are declared outside a function.

Note: External variables should not be initialized.

For example

```
        main()
        {
                y = 5;
                -------
                -------
        }
        int y;      //global declaration
        fun1( )
        {
                y=y+1;
        }
```

In the above example as far as main is concerned, y is not defined. So, the compiler will issue an error message. Unlike local variables, global variables are initialized to zero by default. The statement
y = y + 1; in fun1 will therefore assign 1 to y.

External declaration

In the program sExment above, the main cannot access the variable y as it has been declared after the main function. This problem can be solved by declaring the variable with the storage class **extern.**

```
        main()
        {
                extern  int  y ;//external declaration
                y = y + 5;
                printf("%d\n", y);
                func1( );
    }
    int y;
    func1()
    {
                y = y + 1;
                printf("%d", y );
        }
```

Output:

```
5
6
```

Ex:2:

```
#include<stdio.h>
void display();
void main()
{
extern int i; i=10
printf("i=%d",i);
display();
getch();
return;
}
int i;
void display()
{
printf("i=%d",i);
return;
}
```

Static variables

The value of a static variable persists until the end of the program. A variable can be declared static using the keyword static like static int x;

A static variable may be either an internal type or an external type depending on the place of declaration.

Internal static variables are those which are declared inside a function. The scope of internal static variables extends up to the end of the function in which they are defined. Therefore, internal static variables are similar to auto variables, except that they remain in existence (alive) throughout the remainder of the program. Therefore, internal static variables can be used to retain values between function calls. For example it can be used to count the number of calls made to a function.

Ex:

```
void func1 ( void ) ;    //function prototpye
void main( )
```

```
{
  int i;
  for( i = 0; i < 3 ; i ++ )
  func1( );          //function call
}
void func1 ( void )
{
  static int x = 0;
  x = x  + 1 ;
  printf("x = %d \n ", x );
}
```

<u>Output</u>

```
x = 1
x = 2
x = 3
```

A static variable is initialized only once, when the program is compiled. It is never initialized again. During the first call to **func1**, x is incremented to 1. Because x is static, this value persists and therefore, the next call adds another 1 to x giving it a value of 2. The values of x becomes 3 when the third call is made.

If we declared x as an **auto** variable, the output would have been:

```
x = 1
x = 1
x = 1
```

This is because each time func1 is called, the auto variable x is initialized to zero. When the function terminates, its value of 1 is lost.

An **external static variable** is declared outside of all functions and is available to all functions in that program. The difference between a static external variable and a simple external variable is that the static external variable is available only within the file where it is defined while the simple external variable can be accessed by other files.

Rexister variables

Keyword : rExister

- RExister variables values are stored in CPU rExisters rather than in memory where normal variables are stored.
- RExisters are temporary storage units in CPU .

❖ The time required to access rExister value is less than values stored in memory.

Restriction on RExister variables:

i) RExister variable address is not available to user.

ii) We can not use address operator(&) and indirect operator (*) on rExister variable.

iii) RExistered variables does not allow implicit conversions.

Ex:

```
main ( )
{
   rExister int i;
   for (i=1; i< =5; i++)
   printf (”%d  ”,i);
}
```

Output: 1 2 3 4 5

1.3.7 Type conversions

Converting one data type into another is the concept of type conversion

2 types

1. Implicit type conversion
2. Explicit type conversion

1) Implicit type conversion

❖ ‘C ‘ compiler converts one data type to another automatically so that expression can be evaluated without losing any significance. This automatic conversion is known as “implicit type conversion”

❖ Implicit type conversion is automatically done by the compiler by converting smaller data type into a larger data type.

❖ During the evaluation , ‘C’ uses a rule that in all expressions except in assignment, any implicit type conversion are made from lower size type to higher size type as shown below

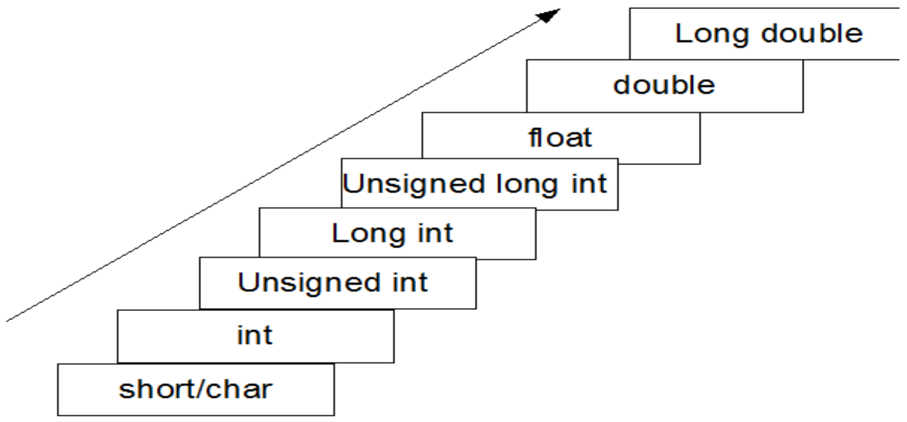


Figure 1.21: *Implicit type conversion*

Here, the above expression finally evaluates to a 'double' value

The sequence of rules that are applied while **implicit type conversion** is as follows:

All short and char are automatically converted to int; then

1. 1 if one of the operands is long double, the other will be converted to long double and the result will be long double
2. else, if one of the operands is double, the other will be converted to double and the result will be double.
3. else, if one of the operands is float, the other will be converted to float and the result will be float.
4. else, if one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be unsigned long int .
5. else, if one of the operands is long int, the other will be converted to long int and the result will be long int .
6. else, if one of the operands is unsigned int, the other will be converted to unsigned int and the result will be unsigned int.

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However the following changes are introduced during the final assignment.

Float to int causes truncation of the fractional part.

Double to float causes rounding of digits.

Long int to int causes dropping of the excess higher order bits.

Example for Implicit conversion :

```
 int i,x;
float f;
double d;
long int l;
```

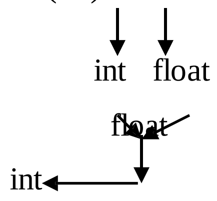
Here, the above expression finally evaluates to a ’double’ value

<u>2)Explicit type conversion</u>

It is done by the user by using (type) operator

Ex:
```
int a,c;
float b;
```

c = (int) a + b



Here, the resultant of ‘a+b’ is converted into ‘into’ explicitty and then assigned to ‘c’

<u>Example for Explicit conversion :</u>

```
main ( )
{
        printf (“%d”, 5/2); printf
        (“%f”, 5.5/2); printf
        (“%f”, (float) 5/2);
}
```

<u>Output</u>

```
2
2.75
2.5
```

1.3.8 The main method and command line arguments

Command line arguments in C:

The most important function of C is main() function. It is mostly defined with a return type of int and without parameters : int main() { /* ... */ }

We can also give command-line arguments in C Command-line arguments are given after the name of the program in command-line shell of Operating Systems.
To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.

```
int main(int argc, char *argv[]) { /* ... */ }
```

or

```
int main(int argc, char **argv) { /* ... */ }
```

- **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non nExative.
- **argv(ARGument Vector)** is array of character pointers listing all the arguments.
- If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

Example:

```
// Name of program mainreturn.c
#include <stdio.h>

int  main(int argc, char* argv)
{
   Printf (“no.of arguments passed through command prompt is %d”,argc);
   Printf (“arguments are:\n”);
   for (int i = 0; i < argc; ++i)
       printf(“%s”, argv[i]);

   return 0;

}
```

1.4 Bitwise operators:

This operators are used to manipulate the data at bit level.

Used to test the bits , shift the bits to right or left.

Note:

Bitwise operators may not be applied to float or double.

Unlike other operators, bitwise operators operate on bits (i.e. on binary values of on operand)

1.4.1 Bitwise AND, OR, XOR and NOT operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
~	One's complement

Table 1.27 :*List of Bitwise operators*

Bitwise AND		
a	b	a &b
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.28: *Bitwise AND operator*

Bitwise OR		
a	b	a | b
0	0	0
0	1	1
1	0	1

1	1	1

Table 1.29: *Bitwise OR operator*

Bitwise XOR		
a	b	a ^b
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.30: *Bitwise XOR operator*

let let a= 12, b=10

	8	4	2	1
a =12	1	1	0	0
b =10	1	0	1	0
a &b	1	0	0	0

a&b = 8

	8	4	2	1
a =12	1	1	0	0
b =10	1	0	1	0
a | b	1	0	1	0

a | b = 14

	8	4	2	1
a =12	1	1	0	0
b =10	1	0	1	0
a ^b	0	1	1	0

a ^ b

a ^ b = 6

Program

```
main ( )
{
 int a= 12, b = 10;
 printf (" %d", a&b);
          printf (" %d", a| b);
 printf (" %d", a ^ b);
```

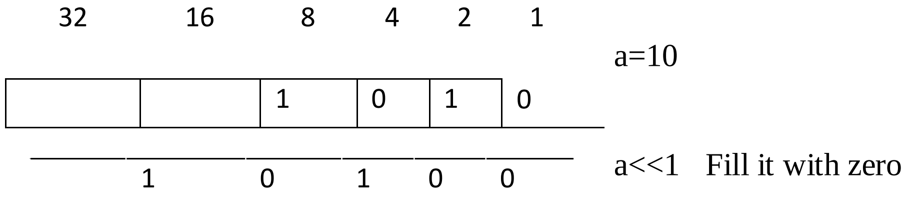
Output

```
8

14

6
```

}

Left Shift

- ❖ If the value of a variable is left shifted one time, then its value gets doubled
- ❖ eg: a = 10 then a<<1 = 20



a<<1 = 20

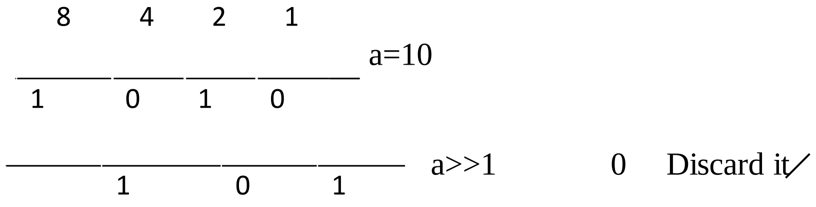
Formula: n * 2^s where n is the given number and s in the number of shifts.

Example: left shift 4 by 2 (i.e 4<<2),here n=4 and s=2 then result is 16 i.e(4* 2^2)

Right shift :

If the value of a variable is right shifted one time, then its value becomes half the original value

- ❖ eg: a = 10 then a>>1 = 5



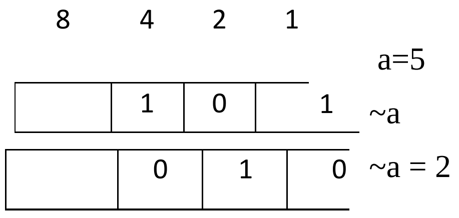
a>>1 = 5

formula: n / 2^s where n is the given number and s in the number of shifts.

Example: **Right** shift 4 by 2 (i.e. 4>>2),here n=4 and s=2 then result is 1 i.e.(4 / 2^2)

Ones complement

- ❖ If converts all ones to zeros and zeros to ones
- ❖ Ex: a = 5 then ~a=2 [only if 4 bits are considered]



Program

```
main ( )
{
 int a= 20, b = 10,c=10;
 printf (" %d", a<<1);
        printf (" %d", b>>1);
        printf (" %d", ~c);

}
```

Output

```
40

5

11
```

Signed

1's complement = - [give no +1]

Eg : ~10 = - [10+1] = -11

~-10 = - [-10+1] = 9

unsigned

1's complement = [65535 – given no]