SUBJECT : PROGRAMMING FOR PROBLEM SOLVING (PPS)

FACULTY : PROF. SHIVA KUMAR. R. NAIK
SCHOOL OF COMPUTING & IT

Unit - 1

Fundamentals of problem Solving and Introduction to C Language

① Algorithm

Before writing any sequence of instructions or a program for a problem, we need to know how the solution can be expressed to solve a problem using Algorithms and flowcharts.

Definition: An algorithm is defined as unambiguous, step by step procedure to solve a given problem in finite number of steps by accepting a set of inputs and producing the desired answer for the given problem.

An algorithm is a procedural solution to a problem. This procedural solution is not the answer for a problem, but step by step procedure to get the answer.

Characteristics/properties of Algorithm

- There should not be ambiguity in any of the instructions.
- There should not be any uncertainity about which statement or instruction has to be executed next.
- The instructions should be simple.
- There should be finite sequence of instructions or statements to transform the given input to the desired output
- The algorithm should always produce correct output
- Once the onput is obtained, the algorithm should terminate

Advantages of writing Algorithms and Flowcharts

- Effective communication: Since Algorithm is written using English like language, algorithm is better way of communicating the logic to the concerned people.
- Effective Analysis: With the help of an algorithm, problem can be analyzed effectively. There can be number of algorithms for solving a given problem. Based on the analysis, a best and efficient algorithm can be selected.
- Proper Documentation: Algorithms serve as a good documentation which is required to identify the logical errors.
- Easy and Efficient coding: By looking at the algorithm, we can easily write programs using C/C++ or any programming language.
- Program Debugging: Algorithms help in debugging so that we can identify the logical errors easily.
- Program Maintenance: Maintaining the software becomes much easier.

* Design an Algorithm to find Area of Rectangle

Step 1: Start

Step 2: [Input the values of length and breadth]
read length, breadth

Step 3: [Find the area of Rectangle]
area <- length * breadth

Step 4: [output the result]
print area

Step 5: [Finished]
Stop

* Design an algorithm to find Area of circle

Step 1: Start

Step 2: [Input the value of radius]
read radius

Step 3: [Find the area of circle]
area <— 3.14 * r * r

Step 4: [output the result]
print area

Step 5: [Finished]
Stop

* Design an algorithm to convert degree in fahrenheit to celsius

Step 1: Start

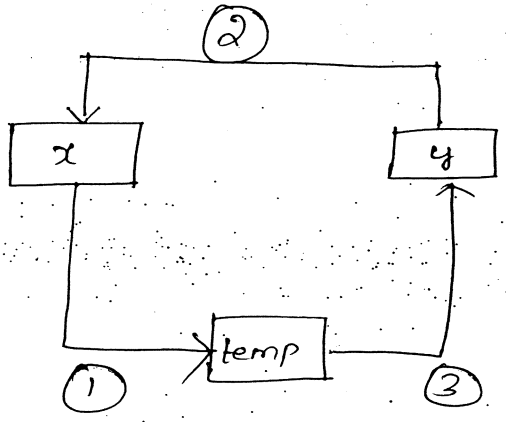
Step 2: [Input the degrees in fahrenheit]
read f

Step 3: [Compute degrees in Celsius]
$c = (5/9)(f - 32)$

Step 4: [Display degrees in celsius]
print c

Step 5: [Finished]
Stop.

* Design an Algorithm to exchange the contents of two variables using temporary variable



- temp = x
- x = y
- y = temp

Step1: Start

Step2: [Input the values of x and y]
read x, y

Step3: [Exchange x and y]
temp = x
x = y
y = temp

Step4: [output the result]
print x, y

Step5: [Finished]
Stop

* Design an Algorithm to exchange the contents of two variable without using temporary variable

Step1 : Start

Step2 : [Input the values of x and y]

read x, y

Step3 : [Exchange x and y]

$x = x + y$

$y = x - y$

$x = x - y$

Step4 : [output the result]

print x, y

Step5 : [Finished]

Stop

* Design an algorithm to find Simple Interest

Step1 : Start

Step2 : [Input the values of p, t and r]

read p, t and r

Step3 : [Compute Simple Interest]

$SI \leftarrow (p * t * r)/100$

Step4 : [output the result]

print SI

Step5 : [Finished]

Stop

* Design an algorithm to find Area of triangle when 3 sides are given

area of triangle can be computed using the formula

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

Where a, b and c are the sides of the triangle and s is half the perimeter of the triangle which is given by

$$s = \frac{(a+b+c)}{2}$$

Step1: Start

Step2: [Read three sides of a triangle]

read a, b, c

Step3: [Compute half the perimeter]

s ← (a+b+c)/2

Step4: [Find the area of triangle]

area ← Sqrt(s * (s-a) * (s-b) * (s-c))

Step5: [output the result]

print area

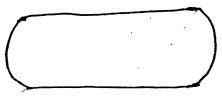
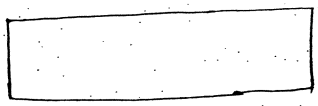
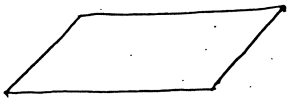
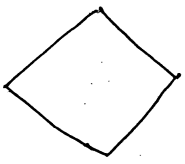
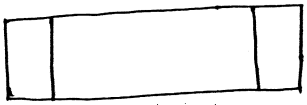
Step6: [Finished]

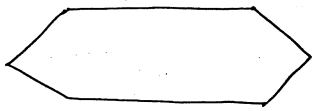
Stop

② Flowchart

Definition: A flowchart is a pictorial representation of an algorithm/program. That is, flowchart consists of sequence of instructions that are carried out in an algorithm. All the steps are drawn in the form of different shapes of boxes, circles and connecting arrows.

The various types of geometric shapes, arrows and symbols used while drawing the flowchart are called flowchart symbols.

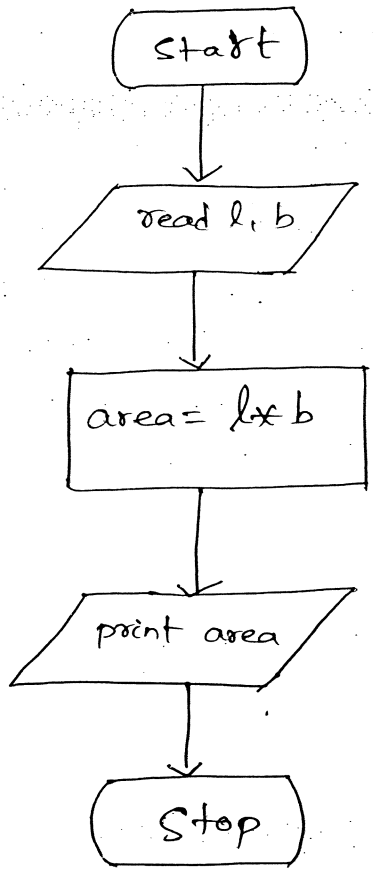
Symbols used	Meaning associated with symbol
	Start or end of the algorithm (program)
	Computational steps (used while assigning and calculating more results)
	Input or output operation
	Decision making and branching
	predefined computation or process (used when functions are used)
	Connector (used to show the continuity of the algorithm in the next page).



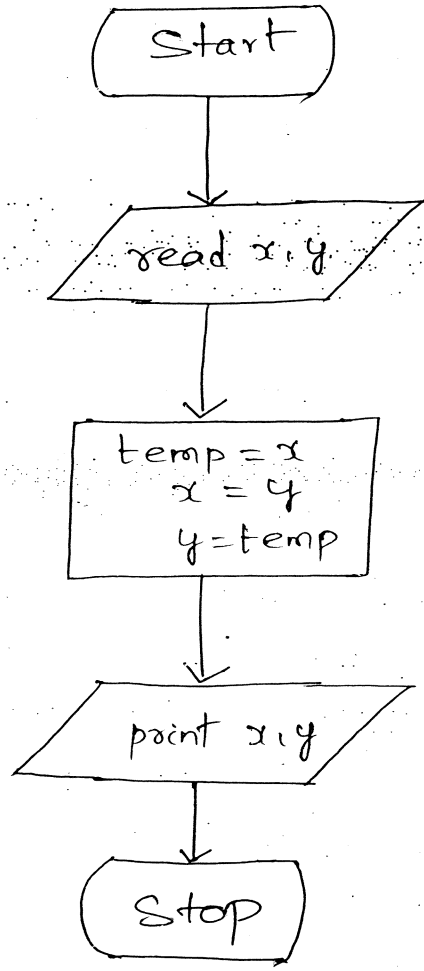
Repitition or looping Statements

Flow of control

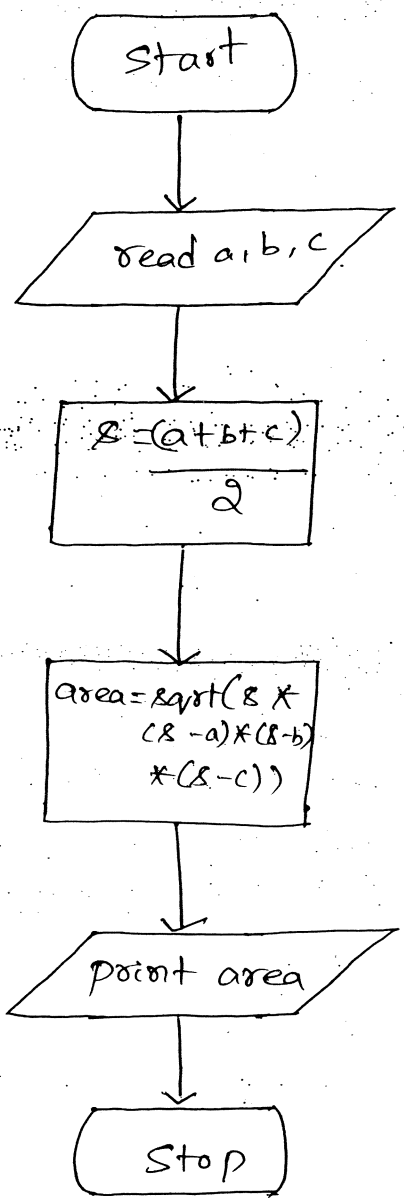
* Design a flowchart to find Area of Rectangle



* Design a flowchart to swap contents of two variables using temporary variable



* Design a flowchart to find Area of triangle when 3 sides (a, b, c) are given

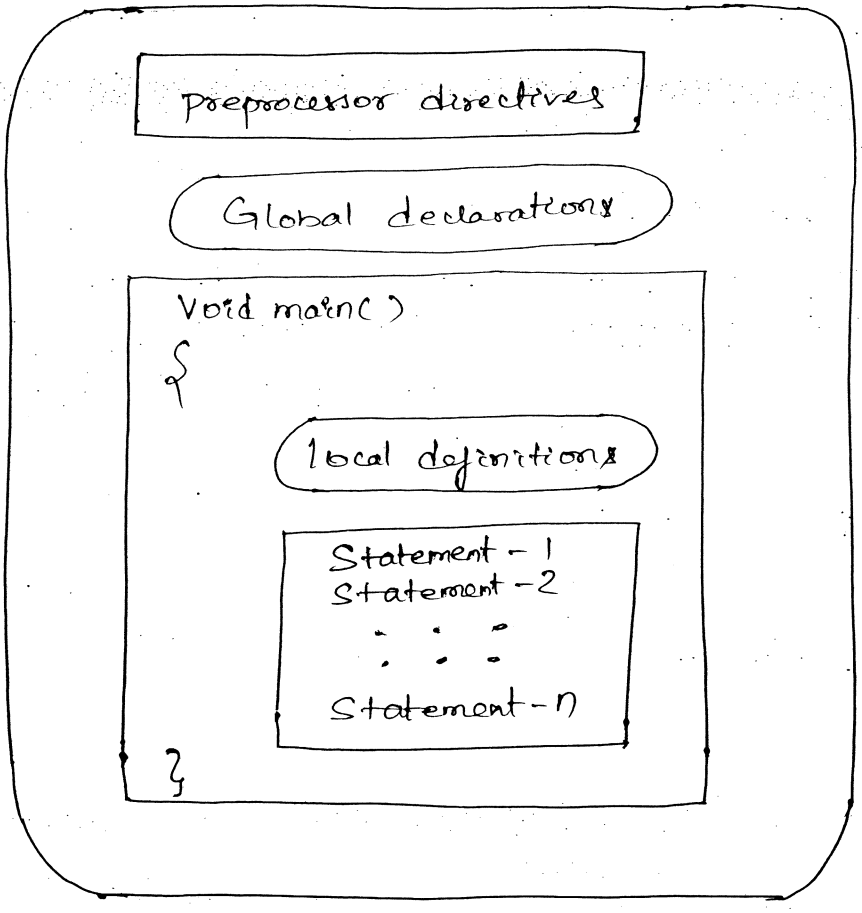


* Differences between Algorithm and Flowchart

Flowchart	Algorithm
• Graphical or pictorial representation along with instructions	• Algorithm are expressed in English like language along with mathematical expressions
• For complex problems, flowcharts become complex and clumsy and hence are not used	• Algorithms can be used for simple or complex problems

Flowchart	Algorithm
• Modification of flowchart is difficult	• Algorithm can be modified easily
• Since it is pictorial representation, we can easily understand the logic for a given problem	• Since, it is not pictorial representation, understanding may be slightly difficult

(3) Structure of C program



- **Preprocessor directives** : preprocessor directives are also called pre-compiler directives. preprocessor accepts the source program and prepare the source program for compilation. The preprocessor statements start with symbol #. The normal preprocessor used in all programs

is include. The preprocessor directives contains the list of header files. Definition of library functions are included in a file called header file.

Example: if you use printf() and scanf() function then we have to include "stdio.h" header file within which definition of printf() and scanf() are found.

Global declarations (Global variables): These are the variables that are defined immediately after preprocessor directives. These variables are visible to all parts of the program.

main(): Every 'C' program will have one function main() and it is the first function which will be always executed and there should be only one main() function within the program and it is case sensitive. main() functions calls all other functions. The statements enclosed within left and right brace is called body of the function.

main() function is divided into two parts

① Declaration section: The variables that are used within the function. main() should be declared in the declaration section only. The variables declared inside any function are called local variables

② Executable section: Consists of set of statements which are required in order to express the solution to the problem statement and are the building blocks of a program.

(4) Introduction to programming languages

- Set of instructions given to the computer to achieve a specific task is called a program. The process of providing instructions to solve a specific problem is called programming
- The person who writes the program is called programmer. The language that is used to write a program is called programming Language.

Types of programming Languages

(a) Machine level Language (MLL)

Machine Language is nothing but the set of instructions given to the computes in the form of 0's and 1's.

(b) Assembly level language (ALL)

Instead of using 0's and 1's to represent an instruction, in assembly language we use symbolic names

Ex: ADD R1, R2

This instruction tells that the values in R1 and R2 are to be added.

(c) High level Language (HLL)

HLL is one, which is written using symbols and words just like English language. HLL enable the programmer to write machine independant code. The main advantage of HLL over low lowel languages is that they are easy to read, write and maintain

Examples: C, C++, java etc.

(5) Introduction to 'C' programming language

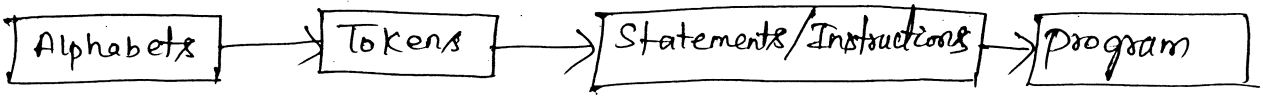
* Why C Language ?

C is very popular language because of the following features

- C is a structured programming language
- It is considered a high-level language because it allows the programmer to solve a problem without worrying about machine details
- It has wide variety of operators using which a program can be written easily to solve a given problem.
- Helps us to develop efficient programs.
- C is machine independent
- A C program can be executed on many different hardware platforms.

* Steps to learn C Language

Dennis Ritchie developed C language in 1972. Dennis Ritchie is called father of C language. C language inherits many features from ALGOL, BCPL and B Language.



(a) Character set (Alphabets of C)

Any symbol that is used while writing a C program is called a Character or an alphabet. A character can be a letter, digit or any special symbol. All those characters that can be used while writing C programs are called alphabets or character set. Alphabets of C Language can be

letters : lower case letters a b c ... z
Upper case letters A B C ... Z

digits : from 0 to 9

Symbols such as : ; : , # ( ) { } [ ] etc

Whitespaces : Characters such as tab (\t), space, newline (\n), Carriage return (\r) etc are also alphabets of C Language and are called white spaces.

(b) Tokens

A token is a smallest or basic unit of a C program. One or more characters are grouped in sequence to form meaningful words called tokens.

Tokens in C language are classified into 5 types.

(1) Keywords (Reserved words)

The tokens which have predefined meaning in C Language are called keywords. They are reserved for specific purpose in C language, hence they are also called reserved words,

Examples: int, float, double, if, else, while, do, char, long etc

The rules to be followed while using Keywords

- Keywords should not be used as variable names, function names, array names etc.
- Meaning of Keywords cannot be changed by end user.
- All Keywords should be written in lowercase letters.

② Identifiers:

In the computer memory we can store the data as well as instructions. Each piece of data or the instruction is stored at unique address in the memory of computer. It is very difficult for us to:

- Access the data or instructions using these addresses
- Manipulate the data using these addresses.
- Keep track of ~~data~~ ~~stored~~ addresses of data stored in the memory.

All these disadvantages can be overcome using identifiers

Definition: Identifiers are the names given to various program elements such as Variables, Constants, function names, array names etc. Identifier is a word consisting of sequence of one or more letters or digits along with "_" (underscore) Symbol.

* Rules to be followed while framing an identifier

- The first character in the identifier should be a letter or underscore and can be followed by any number of letters or digits or underscores.

- No extra symbols are allowed other than letters, digits and "_"
- Length of the identifier can be upto maximum of 31 characters.
- Keywords cannot be used as identifiers.
- Identifiers are case sensitive.

Differences between Keywords and identifiers

Keywords	Identifiers
• Have pre-defined meaning	• Do not have pre-defined meaning
• Keywords are the instructions given to the compiler	• Identifiers are not instructions to the compiler. They are the names introduced into a program.
• Examples: if, for, while, do, int, Char etc	Examples: sum, length, a, b, c, area etc

(5) Constants

Constant is a data item which will not change its value during the execution of a program.

Constants in 'C' Language are classified into 3 types.

- Numeric constant → Integer
  → Floating point
- Character constant
- String constant

* Numeric constants

① **Integer constant**: An Integer is a whole number without any fraction part. No extra characters are allowed other than + and - sign. If + and - are present, they should precede the number.

Integer constant is classified into 3 types.

- **Decimal**: Combination of digits '0' to '9'.
  Ex: 10, 100, -67, +989 etc.
- **Octal**: Combination of digits '0' to '7' with a prefix 0 (digit zero).
  Ex: 010, 0777, -065 etc.
- **Hexadecimal**: Combination of digits from '0' to '9' along with the letters 'A' to 'F' or 'a' to 'f' ~~and~~ with a prefix 0x or 0X
  Ex: 0x8A, 0XAB, 0XA123 etc.

② **Floating point**: Floating point Constants are base 10 numbers with fraction part. No extra characters are allowed other than + and - sign.

Floating point numbers can be represented in 2 forms

- **Fractional form**: In this form the number has an integer part followed by a dot and fractional part.
  Ex: 0.5, -0.99, -.6, +0.9

- Exponent form (Scientific notation)

	mantissa	e/E	exponent	
Ex:	9.86	E	3	imply 9.86×10^3
	9.86	e	3	imply 9.86×10^3

* Character constant

A symbol enclosed within a pair of single quotes is called a character constant. Each character is associated with a unique value called ASCII value.

Backslash constants (Escape sequence characters)

Character	Escape character	Meaning
Bell	\a	Beep sound
backspace	\b	Cursor moves towards left one position
horizontal tab	\t	Cursor moves towards right by 8 positions
new line	\n	Cursor moves to the beginning of next line
Carriage return	\r	Cursor moves to the beginning of current line.
quotation mark	\"	double quote
apostrophe	\'	single quote
question mark	\?	question mark
backslash	\\	backslash
NULL	\0	Null character

* String constant

A sequence of characters enclosed within a pair of double quotes is called a string constant. The string always ends with NULL character (\0)

Ex: "shiva", " ", "CIVIL" etc.

(4) operators

operator is a symbol that specifies ~~various type of~~ operation being performed on various types of data.

Ex: +, -, <<, >>, !, <, > etc.

(5) Special symbols

Ex: {, }, (,), [,] etc.

* Range of integer

Integers are classified into 2 types

(1) unsigned integer: only positive numbers are considered.

Range of unsigned number will be 0 to $2^n - 1$ in an n-bit machine

Ex: Range of 3-bit unsigned number = 0 to $2^3 - 1$

= 0 to 8-1

= 0 to 7

(2) signed integer : Both positive and negative numbers are considered

Range of signed number is -2^{n-1} to $+2^{n-1}-1$

Example : Range of 4-bit signed number = -2^{4-1} to $+2^{4-1}-1$

$= -2^3$ to $+2^3 - 1$

$= -8$ to $+8 - 1$

$= -8$ to $+7$

* Data types

Datatype defines the type of data stored in a memory location. The data type determines how much memory should be allotted or allocated for a variable.

Datatypes in C Language are classified into two types

(1) primitive data type

- Data types that can be manipulated by machine instructions are called primitive data types. The various primitive data types supported by C language are shown below

(a) int

An int is a keyword which is used to define integers in C Language. The size of int is machine dependent. C supports three different sizes of integer data type namely

- short int – Keyword used is short, format specifier is %h
- int – Keyword used is int, format specifier is %d
- long int – Keyword used is long, format specifier is %ld

To know the size of any data type, we use the operator sizeof.

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

No. of bits	Type and Size	Range of unsigned int 0 to $2^n - 1$	Range of signed int -2^{n-1} to $+2^{n-1} - 1$
16-bit machine	short int 2 bytes	0 to $2^{16} - 1$	-2^{15} to $+2^{15} - 1$
	int 2 bytes	0 to $2^{16} - 1$	-2^{15} to $+2^{15} - 1$
	long int 4 bytes	0 to $2^{32} - 1$	-2^{31} to $+2^{31} - 1$
32-bit machine	int 4 bytes	0 to $2^{32} - 1$	-2^{31} to $+2^{31} - 1$
	long int 4 bytes	0 to $2^{32} - 1$	-2^{31} to $+2^{31} - 1$

Advantages: can be used for
- counting
- indexing

Disadvantages:
- Higher range numbers cannot be represented
- Numbers with fractional part cannot be represented and stored.

(b) float (%f)

float is a keyword which is used to define floating point number in C language.

n-bit machine	Size of float	Range of float
16-bit machine	4 bytes	3.4E-38 to 3.4E38
32-bit machine	8 bytes	1.7E-308 to 1.7E308

Advantages

- Numbers with fractional part or decimal points can be stored
- Higher precision and range when compared to integers

Disadvantages

- Cannot be used for counting, indexing
- precision of more than 6 digits after decimal point is not possible.
- Represented approximately by most computers.

(c) double (%lf)

double is a keyword which is used to define long floating point numbers in C language

n-bit machine	Size of double	Range of double
16-bit	8 bytes	1.7E-308 to 1.7E308
32-bit	16 bytes	3.4E-4932 to 1.1E4932

Advantages

- Higher precision and range when compared to floating point numbers

Disadvantages

- Cannot be used for counting, indexing
- Numbers are represented approximately by most computers

(d) Char

- format specifier used is %c
- A char is a keyword which is used to define single character or a sequence of characters called string in C Language.

n-bit machine	Size of char	Range of unsigned char 0 to 2^n-1	Range of signed char -2^{n-1} to $+2^{n-1}-1$
16/32-bit	1 byte	0 to 2^8-1 or 0 to 255	-2^{7-1} to 2^7-1 or -128 to 127

Advantages

- Used to represent characters or strings.

(e) void

Empty data type. No value is allocated hence does not occupy any space in the memory

n-bit machine	Size of void	Range
16/32-bit	0	No value

③ Derived data types

Derived data types are aggregates of one or more types of basic data types

Ex: structures, unions, pointers, arrays etc.

✱ Variables

Definition: A variable is a name given to a memory location where the data can be stored.

① Why we use variables

The computer memory is divided into a number of locations called storage cells. Each location can hold one byte of information and each location is associated with address. Any type of data such as integer, floating point, character and string can be stored in these memory locations using the addresses. But it is not possible to remember addresses of every data stored in memory. For this we use variables.

As we identify a person or a place by a name, the data stored in the memory can be identified by giving names. These names that we associate with various data in the memory are called variables.

Points to remember when we use variables

- Every variable should be associated with type, size and value
- Whenever a new value is placed into a variable, it replaces the previous value.
- Reading variables from memory does not change them.

- The value stored in the variable may change during execution of the program.

(2) Rules for defining a variable

- The first character in the variable should be a letter or an underscore
- The first character can be followed by any number of letters or digits or underscores.
- No extra symbols are allowed other than letters, digits and underscore
- The length of the variable can be up to a maximum of 31 characters.
- C Keywords should not be used as variable names.

(3) Syntax for Declaring a variable

Syntax:

```
type   v1, v2, ... vn;
```

Where

- type is type of variables v1, v2, ... vn. The type of variables may be int, float, char, double etc.
- v1, v2, ... vn are variable names. All the variables should be separated by commas and must end with semicolon.

④ Syntax for initializing a variable

Syntax: `type var_name = data;`

- type indicates the data type
- var_name indicates name of the variable
- = is the assignment operator
- data indicates the value to be stored in memory associated with variable var_name

Ex:

```
int a=10;
float b=20.20;
```

Variable initialization can be done using three different ways:

ⓐ Initializing only one variable

Syntax:

`type var_name = data;`

Ex:

```
double b = 20.6070;
```

ⓑ Initializing more than one variable while defining

Ex:

```
int count=0, sum=10;
```

ⓒ Initialization at appropriate place

To avoid most of the programming errors, it is always better to initialize the variables at the appropriate place in the body of the program.

Ex:
```
int i;
int j;
int k;
. . . .
. . . .
i=100;
j=200;
k=300;
```

(5) Types of variables

Based on the type of data stored in the memory, the variables are classified as

- Integer variable: if integer value has to be stored in the memory, then we have to define a variable of type int.

  Ex: `int a;`

  Integer variable occupy 2 bytes of memory.

- Floating point variable: if floating point value has to be stored in the memory, then we have to define a variable of type float. Floating point variable occupy 4 bytes of memory.

  Ex:
  ```
  float a;
  float b= 3.1416
  ```

- Character variable: if character value has to be stored in the memory, then we have to define a variable of type char. A character variable occupies one byte of memory.

  Ex: `Char c;`

- Double variable: if long floating point data has to be stored in the memory, then we have to define a variable of type double. A long float or double occupy 8 bytes of memory.

  Ex: double d;

* Differences between Keywords and variables

Keywords	variables
• Keywords have pre-defined meaning in a language	• Variables do not have pre-defined meaning
• Keywords are the instructions given to the compiler	• Compiler determines the size of the data to be stored in the variable using the keywords.
• Keywords are not used to store the data	• Variables are used to store the data
• Examples: int, float, double, if, for etc	Examples: sum, add, div, length, area etc

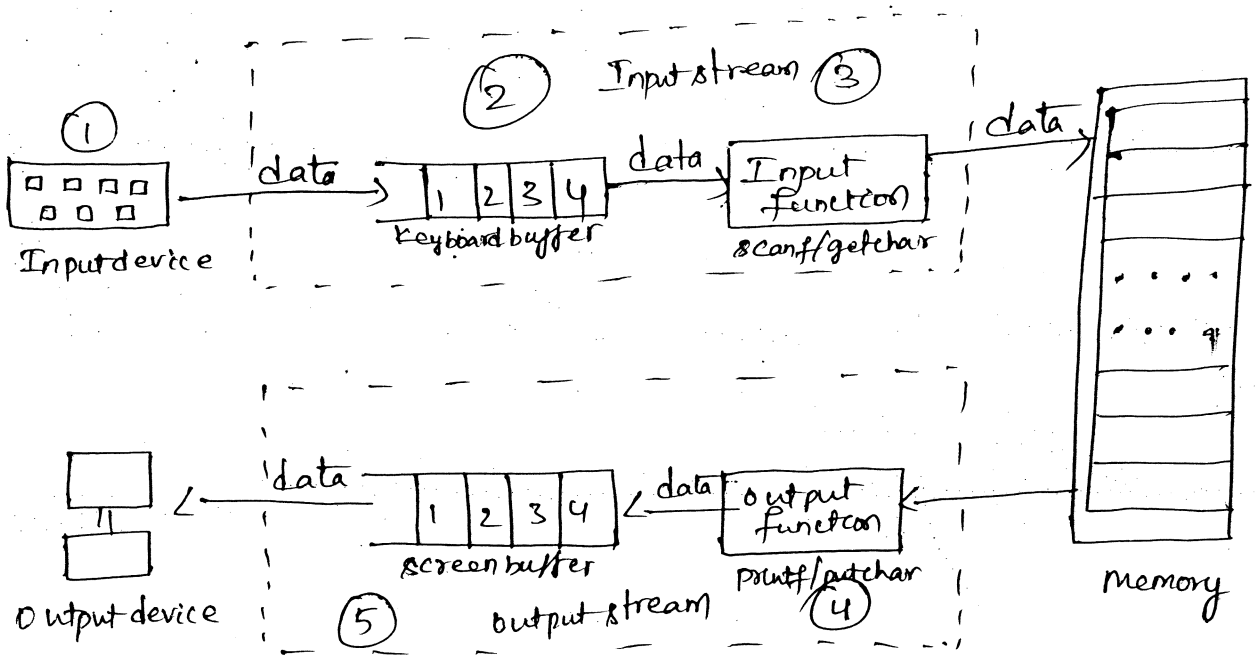
Input and Output Functions

Input Functions: The input functions accept the data from the keyboard and store in memory locations. These functions that help the user to input the data from input devices such as keyboard and transfer the data into memory locations are called input functions

Examples: scanf(), getch(), getchar(), getche(), gets()

Output functions: output functions receive the data from memory locations and display on the monitor. These functions that help the user to send the data stored in memory locations to output devices are called output functions

Examples: printf(), putch(), putchar(), puts()



Activities done when input function is executed are shown below

(1). data is entered from the keyboard

(2). data entered from the key board will be stored in temporary storage area called keyboard buffer. The buffer holds the characters until we press enter.

(3). using input function such as scanf/getche, the data stored in the keyboard buffer is read and the data is converted into appropriated data type and will be stored in memory location using variables.

Activities done when output function is executed are.

(4). using output function, the data is read from memory and stored in temporary storage area called screen buffer.

(5). The data available in the screen buffer is visible on the screen.

* Types of Input/output Functions

(1) Unformatted Input/output Functions

* Unformatted Input Functions

- getchar() → Reads a character from keyboard and waits for the user to press enter key.
- getche() → Reads a character from keyboard and does not wait for the user to type enter key. Character entered will not be visible.
- getche() → Reads a character from keyboard and does not wait for the user to type enter key. character entered will be visible.

- gets() ⟶ Reads a string from keyboard and waits for the user to type Enter key.

* Unformatted output Functions

- putchar() ⟶ Displays a character on the screen
- putch() ⟶ Displays a character on the screen
- puts() ⟶ Displays a string on the screen

Advantages

- Very easy to use
- Sequence of characters can be read and displayed

Disadvantage

- It is not possible to read/print any other data except Characters.

(2) Formatted Input/output Functions

* Formatted Input function scanf()

- ### Need for scanf()

Sequence of character entered from the keyboard may have to be converted into appropriate data type such as integer or floating point number and this can be done in C language using scanf() function.

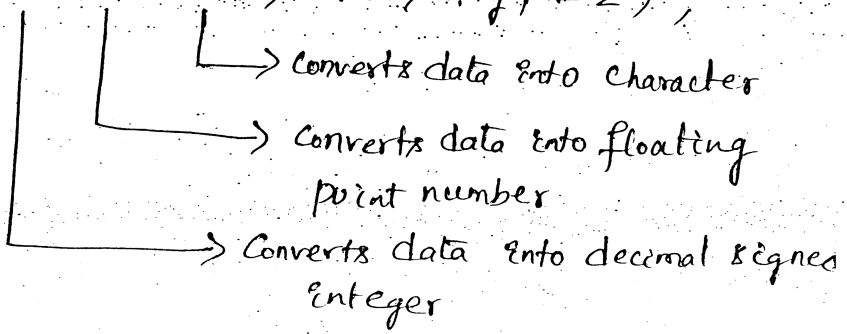
scanf(scan+f)

- scan : scans the data entered from the keyboard
- format : Data is converted into appropriate data type

Syntax

scanf("format string/specifier", address list);

Ex: scanf("%d%f%c", &x, &y, &z);



Number of format specifier = number of variables

Rules while using scanf()

- Variables present in scanf() must represent address of memory locations
- type conversion specified in each format specifier must match with corresponding type of variable present.
- Input data items must be separated by appropriate delimiters such as spaces, commas etc and all the delimiters must be in the same order in format string.
- White space characters other than space should not be used i.e., \t, \b and \n characters should not be used in scanf
- scanf("%d%d ", &a, &b); → space should not be there at end of format string

* Formatted output function printf()

Need for printf()

The data stored in memory locations (0's and 1's) have to be converted into appropriate data type such as integer or floating point number and the above activity is done in C language by the function printf().

- Printf() is used for 2 purpose - printf() function can be used for only printing the message on the terminal

Syntax: Printf ("Message");

- Second purpose or functionality of printf() is to print the value of variables stored in memory locations onto the terminal

Printf (print + f)

- print: prints the data stored in specified memory locations
- format: data will be converted into appropriate data type

Syntax: printf ("format string", list of variables);

Example: Suppose if you want to print the value of two integer variables x and y on the terminal. then the printf() function is written as

printf ("%d %d", x, y);

Number of format specifiers = number of variables.

But always whenever we are printing the value of any variables we only don't Print the value but we will Print ~~the~~ along with a message.

printf("value of x = %d \n value of y = %d \n", x, y);

* Format specifiers used in scanf()

Data type	Format specifier	Description
int	%d	Converts data into decimal signed integer
	%o	Converts data into octal integer
	%x	Converts data into hexadecimal integer
	%i	Converts the data into decimal signed or octal or hexadecimal integer
	%u	converts the data into unsigned integer
	%h	Converts the data into short integer
float	%e	Converts the data into floating point value
	%f	Converts the data into floating point value
	%g	Converts the data into floating point value
char	%c	Converts the data into character
	%s	Convert sequence of characters into a string
double	%lf	Convert the data into long floating point number or double
long int	%ld	Convert the data into long integer value

* Format specifiers used in printf( )

Data type	Format Specifier	Description
int	%d	data is displayed as decimal signed integer
	%o	data is displayed as octal integer
	%x	data is displayed as hexadecimal integer (lowercase hexadecimal digits a to f are printed along with digits 0 to 9)
	%X	data is displayed as hexadecimal integer (uppercase hexadecimal digits A to F are printed along with digits 0 to 9)
	%i	data is displayed as decimal or octal or hexadecimal integer
	%u	data is displayed as unsigned integer
	%h	data is displayed as short integer
float	%f	data is displayed as floating point value without exponent
	%e	data is displayed as floating point value with exponent (e is used while printing)
	%E	data is displayed as floating point value with exponent (E is used while printing)
	%g	data is displayed as floating point value with or without exponent (Trailing zeros will not be displayed)

Data type	Format specifier	Description
Char	%c	data is displayed as character
	%s	data is displayed as a string
double	%lf	data is displayed as long float
long int	%ld	data is displayed as long integer

PROGRAMMING EXAMPLES

(1) Develop a 'C' program to print message "HELLO WORLD" on the terminal

```
#include<stdio.h>
main()
{
        printf("HELLO WORLD\n");
}
```

(2) Develop a 'C' program to print "USN Number" of the student on the terminal screen

```
#include<stdio.h>
main()
{
        printf("1DA07IS048\n");
}
```

(3) Develop a 'C' program to print Student Name, College Name, USN Number and College Address on the terminal using single printf() statement

```
#include<stdio.h>
main()
{
        printf("Shiva\n AIT\n 1DA07IS048\n
                Bengaluru\n");
}
```

(4) Develop a 'C' program to find Area of Rectangle

```
#include<stdio.h>
main()
{
        int length, breadth, area;
        printf(" Enter the values of length and breadth\n");
        &Canf("%d%d", &length, &breadth);
        area = length * breadth;
        printf(" Area of rectangle = %d\n", area);
}
```

(5) Develop a 'C' program to find Area of circle

```
#include<stdio.h>
main()
{
    int radius, area;
    printf("Enter the Radius of the circle\n");
    scanf("%d", &radius);
    area= 3.14 * radius * radius;
    Printf("Area of circle =%d\n", area);
}
```

(6) Develop a 'C' program to convert degree in fahrenheit to celsius

```
#include<stdio.h>
main()
{
    float f, c;
    printf("Enter the degree in fahrenheit\n");
    scanf("%f", &f);
    c = (5/9) * (f-32);
    printf("Degree in celsius = %f\n", c);
}
```

(7) Develop a 'C' program to exchange the contents of 2 variables using temporary variable

```
#include<stdio.h>
main()
{
    int x, y, temp;
    printf("Enter the values of x and y\n");
    scanf("%d%d", &x, &y);
    printf("Value of x before swapping=%d\n
            value of y before swapping=%d\n", x, y);
    temp=x;
    x=y;
    y=temp;
    printf("Value of x after swapping=%d\n
            Value of y after swapping=%d\n", x, y);
}
```

(8) Develop a 'C' program to find Simple Interest

```
#include<stdio.h>
main()
{
    float p, t, r, si;
    printf("Enter the values of p, t and r\n");
```

```
scanf("%f%f%f", &p, &t, &r);
si = (p*t*r) / 100;
printf("Simple Interest=%f\n", si);
}
```

(9) Develop a 'C' program to find Area of triangle when 3 sides a, b, c are given

- Formula used is $area = \sqrt{s(s-a)(s-b)(s-c)}$

- But we cannot use $\sqrt{\ }$ symbol directly in the program, but we can make use of mathematical function called sqrt() which is replacement for $\sqrt{\ }$ symbol defined in header file "math.h". So we can write the formula as `area=sqrt(s*(s-a)*(s-b)*(s-c))`

and $s = \frac{(a+b+c)}{2}$

program:

```
#include<stdio.h>
main()
{
    float a, b, c, area, s;
    printf("Enter the values of three sides of triangle
            a, b and c\n");
    scanf("%f%f%f", &a, &b, &c);
```

```
s = (a+b+c) / 2;
area = sqrt(s*(s-a)*(s-b)*(s-c));
printf("Area of triangle = %f\n", area);
}
```

(10) Develop a 'C' program to read and print size of different data types (LAB PROGRAM-1a)

```
#include<stdio.h>
main()
{
    printf("Size of character data type = %lu\n", sizeof(char));
    printf("Size of short integer datatype = %lu\n", sizeof(short));
    printf("Size of integer data type = %lu\n", sizeof(int));
    printf("Size of long integer data type = %lu\n", sizeof(long));
    printf("Size of float data type = %lu\n", sizeof(float));
    printf("Size of double data type = %lu\n", sizeof(double));
    printf("Size of long double datatype = %lu\n", sizeof(long double));
}
```

(11) Develop a 'C' program to perform Arithmetic operations Addition, Subtraction, multiplication, modulo division and division operations (LAB PROGRAM - 1B)

```
#include<stdio.h>
main()
{
    int a, b, add, sub, mult, mod;
    float div;
    printf("Enter two input numbers a and b\n");
    scanf("%d %d", &a, &b);
    add = a+b;
    sub = a-b;
    mult = a*b;
    mod = a%b;
    div = a/b;
    printf("Addition = %d\n subtraction = %d\n
        multiplication = %d\n modulus = %d\n
        division = %f\n", add, sub, mult, mod, div);
}
```

(12) Develop a 'C' program to read and print an integer number

```
#include<stdio.h>
main()
{
    int num;
```

```
    printf(" Enter the integer number\n");
    scanf("%d", &num);
    printf(" integer number entered by user = %d\n", num);
}
```

* Operators and Expressions

- **Operator**: An operator is a symbol (or a token) that specifies the operation to be performed on various types of data.
- **Operand**: A constant or a variable or a function which returns a value is an operand.
- **Expression**: A sequence of operands and operators that reduces to a single value is an expression.

```
a + b
a - b
a * b
a / b
a % b
```

Operands ← Operators → operands

Using the expression, the programmer can inform the compiler about the operation to be performed on the operands. By looking at the operators and operands, the compiler generates necessary machine instructions. When these machine instructions are executed, we get a single value as the result.

(1) Classification of operators

Operators in C can be classified based on

(a) The number of operands an operator has

The operators are classified into four major categories based on the number of operands as shown below.

- Unary operator : An operator which acts on only one operand to produce the result is called unary operator. In the expressions involving unary operators, the operators precede the operand.

  Ex: ++, --, ~ etc

- Binary operator : An operator which acts on two operands to produce the result is called a binary operator. In an expression involving a binary operator, the operator is in between two operands.

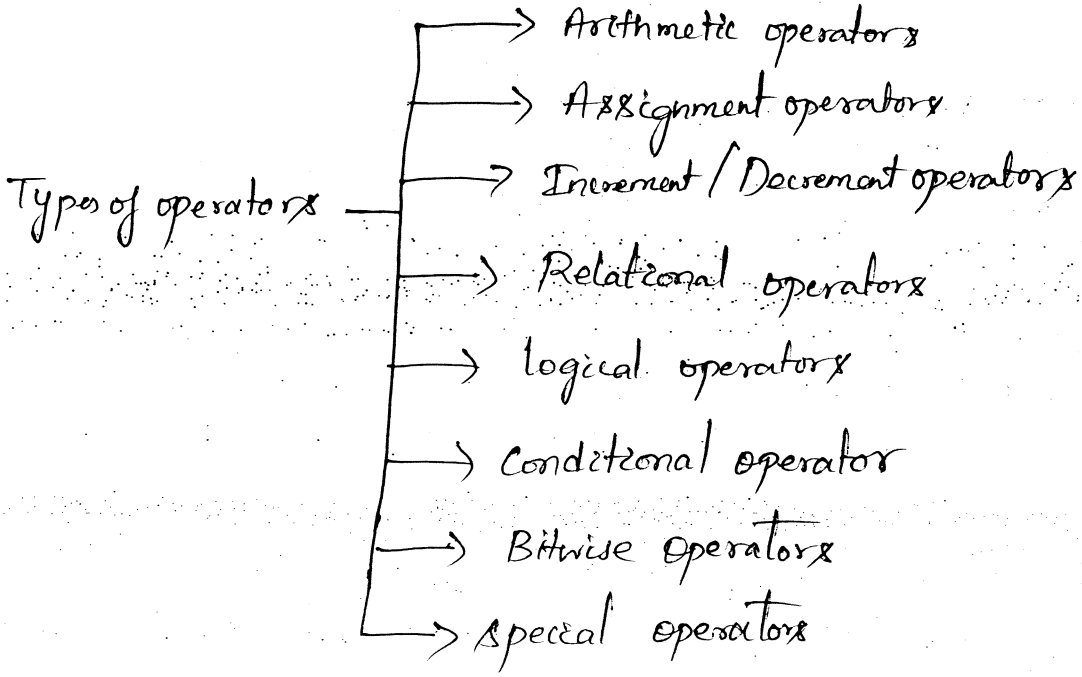
  Ex: +, -, *, %, / etc

- Ternary operator : An operator which acts on three operands to produce a result is called a ternary operator

  Ex: a ? b : c

Since there are three operands a, b and c that are associated with operators ? and : it is called ternary operator. Ternary operator is also called conditional operator.

(b) Based on the type of operation being performed



(i) Arithmetic operators

. The operators that are used to perform arithmetic operations such as addition, subtraction, multiplication, division and modulus operations are called arithmetic operators. These operator perform operations on two operands and hence they are binary operators. Examples: +, −, *, /, %

. In division (/) operation quotient is the result. In modulus operation both the operands must be integer and remainder is the result.

Examples:

- 3+9 = 12
- 10-3 = 7
- 10*5 = 50
- 20/2 = 10
- 20%2 = 0

* Arithmetic expressions

The expressions consisting of only arithmetic operators are called arithmetic expressions. There are two types of arithmetic expressions

(a) Additive expressions: if addition and subtraction operations are performed, then the expression is called additive expression Ex: a+b, a-b

(b) Multiplicative expressions: if multiplication, division and modulus operations are performed, then the expression is called multiplicative expression Ex: a*b, a/b, a%b

* Conversion of expressions

The expressions that we use in mathematics are different from the expression that we use in C Language. So, all mathematical expressions have to be converted suitably into equivalent C expressions.

The following rules have to be remembered while converting mathematical expression into C expressions

- if numerator has more than one operand, enclose the entire numerator inside the parentheses

	Mathematical expression	C expression
Ex:	$s = \frac{a+b+c}{2}$	s = (a+b+c)/2

- if denominator has more than one operand, enclose the entire denominator inside parentheses.

	Mathematical expression	C expression
Ex:	$x = \frac{-b}{2a}$	x = -b/(2*a)

- The argument value of a function must be enclosed inside the parentheses

Ex:	$x = \sqrt{2\pi n}$	x = Sqrt(2*3.1416*n)

- In mathematical expressions the braces such as '{' and '}', the brackets such as '[' and ']' can be used. But, they cannot be used in C expressions.

Ex: (a+b) * c (valid)

{a+b} * c (invalid)

* Type conversion

The process of converting the data from one data type to another data type is called type conversion. There are two types of type conversion

(a) implicit type conversion or promotion

The process of conversion of data from lower rank to higher rank automatically by the C compiler is called implict conversion or promotion.

The promotion hierarchy is

Char → ~~short~~ int → int → unsigned int → long int → unsigned long int → float → double

It is clear from the above hierarchy that the following relationship holds

sizeof(char) < sizeof(short) < sizeof(int) < sizeof(unsigned int) < sizeof(long) < sizeof(unsigned long) < sizeof(float) < sizeof(double)

- If both the operands are of the same type then no conversion takes place.

Examples: (1) 5 + 3

Both are of same data type so no Conversion takes plae 5 + 3 = 8

(2) int float
5 + 3.5
↓
5.0 + 3.5 = 8.5

(3) 4.0/3

4.0/3.0 = 1.333333

(4)
```
int p = 7;
float q = 2;
float r;
r = p/q
```

r = 7/2.0 => r = 7.0/2.0 = 3.5

(5)
```
int x = 7;
int y = 2;
int z;
z = x/y
z = 7/2
```
z = 3

Advantage

- Compiler automatically does conversion from lower rank to higher rank. So, programmer need not worry about conversion procedure or the syntax.

Disadvantage

- if both operands are of the same data type, implicit type conversion is not possible.
- Conversion from higher rank to lower rank is not performed automatically by the Compiler.

(b) Explicit type conversion

Implicit type conversion is possible only if the data type of two operands are different. In a situation where the data type of two operands are same, still conversion is required, then we have to go for explicit type conversion.

Example: Suppose, we want to find the ratio of girls to boys in the college. The ratio is given by:

```
ratio = no_of_girls/no_of_boys ;
```

Here, number of girls and number of boys will be of type integer. So, the compiler will not do any implicit type conversion because, both operands are of the same data type and hence the result will be of type integer. But, to get the ratio which is a floating point number, we are forced to convert one of the operand to float so that the result is also float. This is the place where we require explicit type conversion.

The forcible conversion from one data type to another data type is called explicit type conversion or type casting.

Syntax: (type) expression

Where

- type is the required type of the expression.
- expression can be an operand such as variable or a constant.

Examples:

(1) `i = (int) 8.9999 = 8`

(2) `i = (float) 1/2`

=> `i = 1.0/2`

`i = 1.0/2.0`

`i = 0.500000`

(3) Explain how the evaluation is done for the expression 4/3 to get the answer 1.333333 using explicit type Conversion

Soln: 4/3

We can convert 4 to 4.0 by using explicit type conversion. (float) 4/3

4.0/3 ⇒ 4.0/3.0 = 1.333333.

(4) Assume a = 3, evaluate the following expression

- (float) (a/10)

(float) (3/10) = (float) (0) = 0.0

10) 3 (0 — Quotient.
0
3

(float) (a/10) = 0.000000

- (float) a/10

(float) 3/10.

3.0/10

3.0/10.0

0.333333.

* Arithmetic operator's precedence (precedence of operators).

To evaluate mathematical expressions we will use "BODMAS" rule.

Example: Based on the BODMAS rule evaluate the following expression $6 \times (2+3)/5$

$$6 \times (2+3)/5$$

$$6 \times 5/5$$

$$\frac{6 \times 1}{6}$$

But the BODMAS rule cannot be used for solving C language expressions Instead we use precedence of operators or hierarchy of operators.

The order in which different operators are used to evaluate an expression is called precedence of operators or hierarchy of operators.

	Description	operator	priority	Associativity
Arithmetic operators →	Multiplication	*	1	left to Right
	Division	/	1	left to Right
	Mod	%	1	left to Right
	Addition	+	2	left to Right
	Subtraction	-	2	left to Right

Ex: $8+4*3$ Multiplication has highest precedence

$8+12$ Addition has least precedence.

20

* Associativity of operators

When two or more operators have the same precedence, then precedence rules are not applicable. Associativity determines how operators with the same precedence are evaluated in the expression. So, precedence is applied before associativity and associativity is applied only if operators have the same precedence.

Two types of operator associativity

- Left associativity: In an expression, if two or more operators having the same priority are evaluated from left-to-right then the operators are called left to Right associative operators denoted by L -> R and the process of evaluating from left to Right is called left associativity.
- Right associativity: In an expression, if two or more operators having the same priority are evaluated from Right-to-left then the operators are called Right to left associative operators denoted by R -> L and the process of evaluating from Right to left is called Right associativity.

Examples:

(1) 8/4*16

2 *16

32

(2) 8*4/16

32/16

2

(3) Assume a=8, b=15, c=4.

2 * ((a%5) * (4+(b-3)/(c+2)))

2 * ((8%5) * (4+(15-3)/(4+2)))

• Parentheses will have the highest priority and then next priority is for the operators.

2 * (3 * (4+ (15-3)/(4+2)))

2 * (3 * (4 + 12 / (4+2)))

2 * (3 * (4 + 12/ 6))

2 * (3 * (4 + 2))

2 * (3 * 6)

2 * 18

36

(4) Assume a=1, b=-5, c=6. Evaluate the following expression x1 = (-b+sqrt(b*b-4*a*c))/(2*a)

Soln: After Substituting

x1 = (-(-5) + sqrt((-5)*(-5)-4*1*6))/(2*1)

x1 = (5 + sqrt((-5)*(-5)-4*1*6))/(2*1)

x1 = (5 + sqrt(25 - 4*1*6))/(2*1)

x1 = (5 + sqrt(25 - 4*6))/(2*1)

x1 = (5 + sqrt(25 - 24))/(2*1)

x1 = (5 + sqrt(1))/(2*1)

x1 = (5 + 1.0)/(2*1)

x1 = 6.0/(2*1)

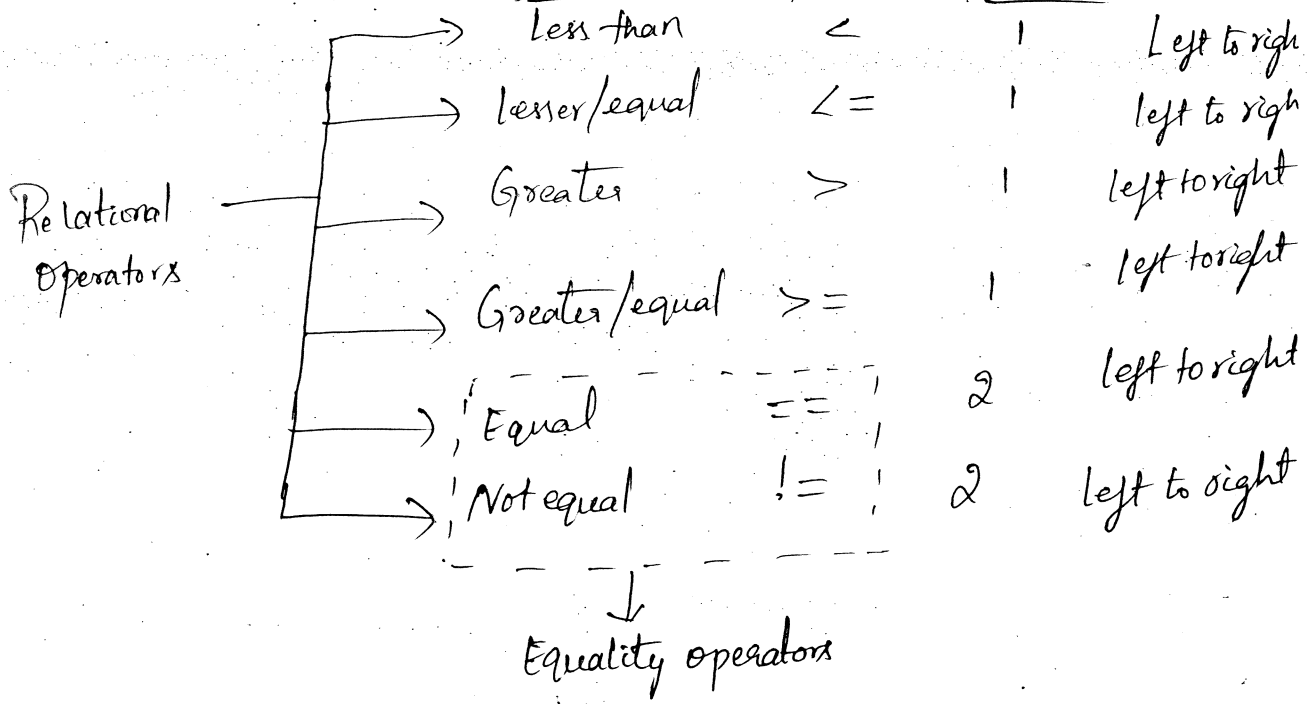
x1 = 6.0/2

x1 = 3.0

Relational operators

In real world, we may have to compare two quantities which results in either true/false or yes/no outputs. In such situations we use relational operators. In C Language, instead of using yes/no answers, we use the terms true/false. So, output of relational expression is true denoted by 1 or false denoted by 0 (zero).

The operators that are used to find the relationship between two operands are called relational operators. The two operands may be constants, variables or expressions. The relationship between the two operand values results in true (1) or false (0).



An expression that contains relational operators is called relational expression.

The rules to be followed while evaluating the relational ~~operat~~ expressions are shown below:

- Evaluate expression within parenthesis
- Evaluate unary operators
- Evaluate arithmetic expressions
- Evaluate relational expressions

Evaluate the following expression

100 /20 <= 10 - 5 + 100 % 10 - 20 == 5 >= 1 ! = 20

- No expression inside parentheses and no unary operators.
- Arithmetic operators are evaluated first next relational

5 <= 10 - 5 + 100 % 10 - 20 == 5 >= 1 ! = 20

5 <= 10 - 5 + 0 - 20 == 5 >= 1 ! = 20

5 <= 5 + 0 - 20 == 5 >= 1 ! = 20

5 <= 5 - 20 == 5 >= 1 ! = 20

5 <= -15 == 5 >= 1 ! = 20

0 == 5 >= 1 ! = 20

0 == 1 ! = 20

0 ! = 20

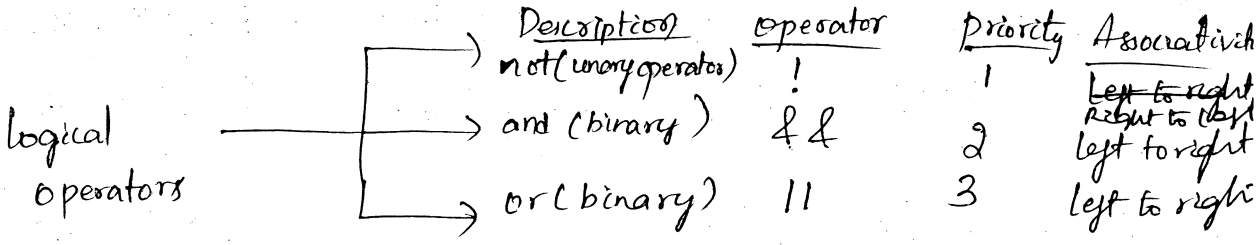
1

Result = 1

(iii) Logical operators

In real world, we may have to combine two or more relational operations to get either true/false or yes/no outputs. In such situations we use logical operators. In C Language 0 is taken as false and any non-zero value is considered as true.

The operators that are used to combine two or more relational expressions are called logical operators. Output of the logical expression will be either true or false. The expression that contains only logical operators are called logical expression.



logical NOT(!)

operand	!operand
True(1)	False(0)
False(0)	True(1)

logical AND(&&)

Operand 1	AND	operand 2		Result
True(1)	&&	True(1)	=	True(1)
True(1)	&&	False(0)	=	False(0)
False(0)	&&	True(1)	=	False(0)
False(0)	&&	False(0)	=	False(0)

The output of logical AND operator will be true if and only if both the operands are evaluated to true

logical OR(||)

Operand 1	OR	operand 2		Result
False(0)	||	False(0)	=	False(0)
False(0)	||	True(1)	=	True(1)
True(1)	||	False(0)	=	True(1)
True(1)	||	True(1)	=	True(1)

The output of logical OR operation will be ~~true~~ false if and only if both the operands are false otherwise it will be true.

Rules to be followed while evaluating logical expression

- Evaluate expression inside parenthesis
- Evaluate unary operators
- Evaluate Arithmetic expressions
- Evaluate Relational expressions
- Evaluate Logical expressions

Evaluate the expression

$11 + 2 > 6 \;\&\&\; !0 \;||\; 11 != 7 \;\&\&\; 11 - 2 <= 5$

- No Parenthesis
- Evaluate unary operator !

$11 + 2 > 6 \;\&\&\; 1 \;||\; 11 != 7 \;\&\&\; 11 - 2 <= 5$

Evaluate Arithmetic operators

$13 > 6 \;\&\&\; 1 \;||\; 11 != 7 \;\&\&\; 11 - 2 <= 5$

$13 > 6 \;\&\&\; 1 \;||\; 11 != 7 \;\&\&\; 9 <= 5$

Evaluate Relational operators

$1 \;\&\&\; 1 \;||\; 11 != 7 \;\&\&\; 9 <= 5$

$1 \;\&\&\; 1 \;||\; 11 != 7 \;\&\&\; 0$

$1 \;\&\&\; 1 \;||\; 1 \;\&\&\; 0$

Evaluate logical operators

$1 \;||\; 1 \;\&\&\; 0$

$1 \;||\; 0$

1

Result = 1

Evaluate the expression

$$10 != 10 \,||\, 5 < 4 \,\&\&\, 8$$

$$10 != 10 \,||\, 0 \,\&\&\, 8$$

$$10 != 10 \,||\, 0 \,\&\&\, 8$$

$$0 \,||\, 0 \,\&\&\, 8$$

$$0 \,||\, 0$$

$$0$$

Result = 0

(iv)

* Assignment operator (=)

An operator which is used to assign the data or result of an expression into a variable is called an assignment operator. A statement with assignment operator is called assignment statement or assignment expression

Types of assignment statements

① Simple assignment statement

Syntax: variable = expression;

Ex:

```
a = 10;
a = b;
x = y;
z = x+y;
```

Assignment operators are Right to left associative operators so whenever the assignment operators are there in the expression we will evaluate from right to left.

② Shorthand assignment statement (Compound assignment)

The operators such as +=, -=, *=, /= and %= are called shorthand assignment operators

Ex: $a = a + 10$ can be written as $a+ = 10$

$a+ = 2 \Rightarrow a = a + 2$

$a- = 2 \Rightarrow a = a - 2$

$a* = 10 \Rightarrow a = a * 10$

$a/ = 50 \Rightarrow a = a / 50$

In the above statements, the expression on the right hand side of assignment operator is evaluated and the result is assigned to variable a on the LHS.

Evaluate the expression

(a) $x* = y+3$ when $x = 20$ and $y = 7$

$x* = (y+3)$

$x = x * (y+3)$

$x = 20 * (7+3)$

$x = 20 * 10$

$\boxed{x = 200}$

(b) $a+ = b* = c-- = 5$ when $a = 1$, $b = 3$ and $c = 7$

$\boxed{a+ = \boxed{b* = \boxed{c-- = 5}}}$

$a+ = b* = \boxed{c-- = 5}$

$\downarrow$

$c = c - 5$

$c = 7 - 5$

$c = 2$

$\downarrow$

$a+ = \boxed{b* = 2}$

$a+ = b = b * 2$

$b = 3 * 2$

$b = 6$

$\downarrow$

$a+ = 6$

$a = a + 6$

$a = 1 + 6 = 7$

So, $\boxed{a = 7, b = 6, c = 2}$

③ Multiple assignment statement

Assing a value or a set of values to different variables in one statement is called multiple assignment statement.

Ex: `i=j=k=10;`

Here the value 10 is copied into variable k, j and i from right to right.

(V) Increment and Decrement operators

Increment and decrement operators are very compact and powerful operator available especially in C language. These two operators are unary operators and have only one operand. All unary operators are right associative.

① Increment Operator (++)

++ is an increment operator. This is a unary operator. It increments the value of a variable by one. The increment operator is classified into two categories as shown below

	Type	Example	Description
Increment operator	post increment	a++	increments value of a by 1
	pre increment	++a	increments value of a by 1

post increment	Description	Equivalent statements	Example
			let a = 20
b = a++;	current value of a used	b = a	// b = 20
	a is incremented by 1	a = a+1	// a = 21

In post increment operator (a++) Operand value is used first and then the operand value is encremented by 1 and the encremented value is used in the next step.

Pre increment		let a = 20
b = ++a; → a is incremented by 1	a = a + 1	// a = 21
b = ++a; → incremented value of a is used	b = a	// b = 21

In pre-increment operator (++a) operand value is incremented by 1 first and this incremented value is used in the expression.

(2) Decrement operator (--)

-- is a decrement operator. This is a unary operator. It decrements the value of a variable by one.

Type	Example	Description
decrement operator → post decrement	a--	decrements value of a by 1
decrement operator → pre decrement	--a	decrements value of a by 1

Description	Equivalent statements	Example let a = 10
post decrement: b = a--; → current value of a used	b = a	// b = 10
post decrement: b = a--; → a is decremented by 1	a = a - 1	// a = 9
pre decrement: b = --a; → a is decremented by 1	a = a - 1	// a = 9
pre decrement: b = --a; → decremented value of a is used	b = a	// b = 9

In post decrement operator, Current value of the variable (operand) is used in the expression and then the value of variable is decremented by 1.

In predecrement operator, value of variable is decremented by 1 first and then decremented value is used in the expression.

- Evaluate the following expression when a = 4

  b = a - ++a

Soln: Increment operators are right associative so solve the expression from RHS to LHS.

b = a - 5 // a = 5

b = 5 - 5

b = 0

Result a = 5, b = 0

- Evaluate the following expressions when a = 10

(i) b = a++ - ++a + a++

b = a++ - ++a + 10 // a = 11

b = a++ - 12 + 10 // a = 12

b = 12 - 12 + 10 // a = 13

Result b = 10, a = 13

(ii) b = --a - a--

b = --a - 10 // a = 9

b = 8 - 10 // a = 8

b = -2

- Evaluate the following expressions when a = 5

(i) a+ = (a++) + (++a)

a = a + (a++) + (++a)

a = a + (a++) + 6 // a = 6

a = a + 6 + 6 // a = 7

a = 7 + 6 + 6

a = 19

(ii) a = (--a) - (a--)

a = (--a) - 5 // a = 4

a = 3 - 5 // a = 3

a = -2

(vi) Conditional operator / ternary operator

As the name indicates, an operator that operates on three operands is called ternary operator. The ternary operators are ? and :

Syntax : (exp1) ? exp2 : exp3 ;

Where

- exp1 is an expression evaluated to true or false
- if exp1 is evaluated to true, exp2 is executed
- if exp1 is evaluated to false, exp3 is executed.

C program to find largest of two numbers using conditional operator

```
#include<stdio.h>
main()
{
    int a, b, max;
    printf("Enter the value of a and b\n");
    scanf("%d %d", &a, &b);
    max = (a>b) ? a : b;
    printf("Largest = %d\n", max);
}
```

C program to find largest of 3 numbers using Conditional Operator

```
#include<stdio.h>
main()
{
    int a, b, c, max;
    printf("Enter values of a, b and c\n");
    scanf("%d %d %d", &a, &b, &c);
    max = (a>b) ? (a>c) ? a : c : (b>c) ? b : c;
    printf("Largest = %d\n", max);
}
```

(vii) Bitwise operators

The operators that are used to manipulate the bits of given data are called bitwise operators.

	Description	Operator	precedence	Associativity
Bitwise operators	Bitwise negate	~	1	L → R
	left shift	<<	2	L → R
	right shift	>>	2	L → R
	Bit-wise and	&	3	L → R
	Bit-wise xor	^	4	L → R
	Bit-wise or	|	5	L → R

(a) Bitwise negate (~) / one's complement

The operator that is used to change every bit from 0 to 1 and 1 to 0 in the specified operand is called one's complement operator.

Example: $a = 10$

$b = \sim a$

First we will represent a in binary in terms of 8 bits

$a = 0000\ 1010$

$b = \sim a$

$\therefore \boxed{b = 1111\ 0101}$

(b) Bit-wise AND (&)

If the corresponding bit positions in both the operands are 1, the AND operation results in 1; otherwise, AND operation results in 0.

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

$\Rightarrow$

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

Example: $a = 10,\ b = 6,\ c = a \& b$

$a = 0000\ 1010$

$b = 0000\ 0110$

$c = a \& b = 0000\ 0010 = 2$

(c) Bit-wise OR (|)

If the corresponding bit positions in both the operands are 0, then OR operation results in 0; otherwise, OR operation results in 1.

1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0

$\Rightarrow$

1 | 1 = 1
1 | 0 = 1
0 | 1 = 1
0 | 0 = 0

Example: a = 10, b = 6, c = a | b

a = 0000 1010

b = 0000 0110

c = a|b = 0000 1110 = 14

(d) Bit-wise XOR (^)

If the corresponding bit positions in both the operands are different, then ex-OR operation results in 1; otherwise, the result is 0.

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

(e) left-shift operator (<<)

The operator that is used to shift the data by a specified number of bit positions towards left is called left shift operator.

Syntax: b = a << num;

num: second operand (Constant or variable)

<<: left shift operator

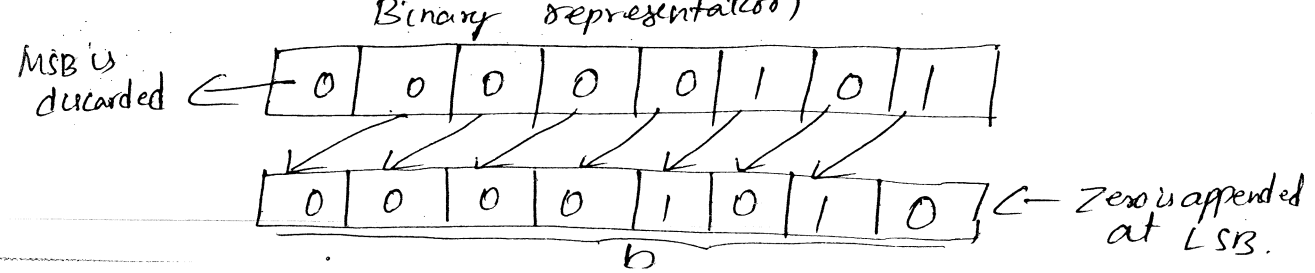
a: first operand (Constant or variable)

Where,

- The first operand is the value to be shifted
- The second operand specifies the number of bits to be shifted.

Example: b = 5 << 1;

the data 5 is shifted towards left by 1 bit-position



- If we are shifting towards left by 1 bit it is equivalent to multiplying by 2.

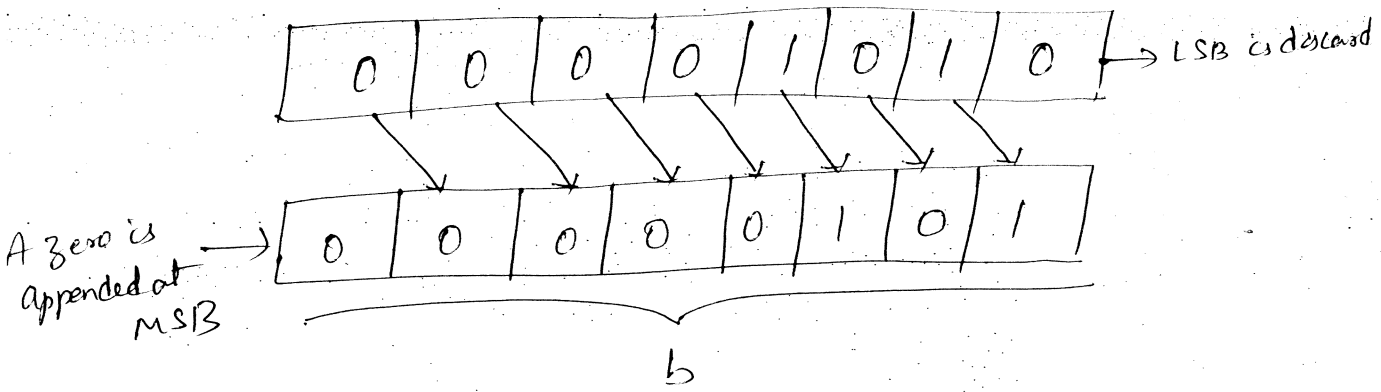
(e) Right shift operator (>>)

The operator that is used to shift the data by a specified number of bit positions towards right is called right shift operator.

Syntax: `b = a >> num;`

- num: second operand (constant or variable)
- >>: right shift operator
- a: first operand (constant or variable)

Example: `b = 10 >> 1`.



- Right shift by 1-bit is divide by 2

(viii) Special operators

(a) Comma operator: The comma denoted by symbol ',' is normally used in the declaration to separate the variables.

Ex: `int a, b, c;`

In the above declaration comma is not used as an operator. Instead, it is used as delimiter to separate variables.

The comma operator has least precedence among all the operators and it is left associative operator.

Evaluate the expression

a = 12, 345, 678

There are two operators - assignment and comma operator. Assignment operator has higher precedence, integer 12 is copied into variable a.

a = 12, 345, 678;
discarded

(b) sizeof() operator

Even though it looks like a function call, sizeof is an operator in C language. This operator is used to determine the number of bytes occupied by a variable or constant in the memory.

Syntax: `sizeof(operand);`

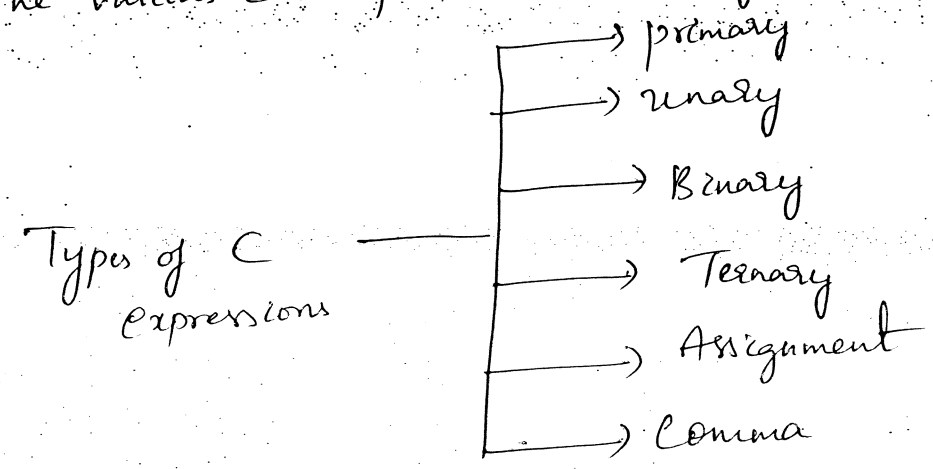
* Precedence of all the Operators (Hierarchy of all operators)

Operator category	operators in order of precedence (Highest to lowest)	Associativity
( ) [ ]	inner most brackets/functioncalls array element reference	L → R
Unary operators	++, --, !, sizeof, ~, +, -, &, *	R → L
member access	* or →	L → R
arithmetic	*, /, %	L → R
arithmetic	-, +	L → R
Shift operators	<< , >>	L → R
Relational	<, <=, >, >=	L → R
equality	==, !=	L → R
Bitwise AND	&	L → R
Bitwise XOR	^	L → R
Bitwise OR	|	L → R

Logical AND	&&	L → R
Logical OR	||	L → R
Conditional	? :	R → L
assignment	=, +=, -=, *=, /=, %=	R → L
Comma	,	L → R

* Expression and Statements (C Expression formats)

The various C expressions are classified as shown below



(1) Primary expressions : An expression with only one operand but without any operator is called primary expression.

The various types of primary expressions are :

. Names : A name can be a variable or a function or any object in the language such as named constant etc.

Ex :

```
const int MAX_LENGTH = 1024;
int a;
float sum;
.
.
.
```

In the above examples all are names.

- **Constants:** A piece of data whose value cannot be changed during the execution of the program is called a constant.

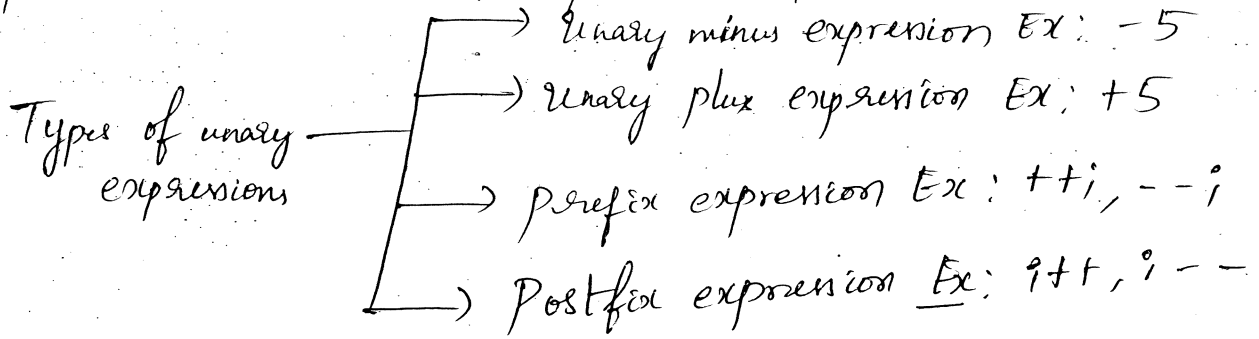
  Examples:

5	integer constant
3.1816	floating point constant
'A'	character constant
"Hello"	String constant

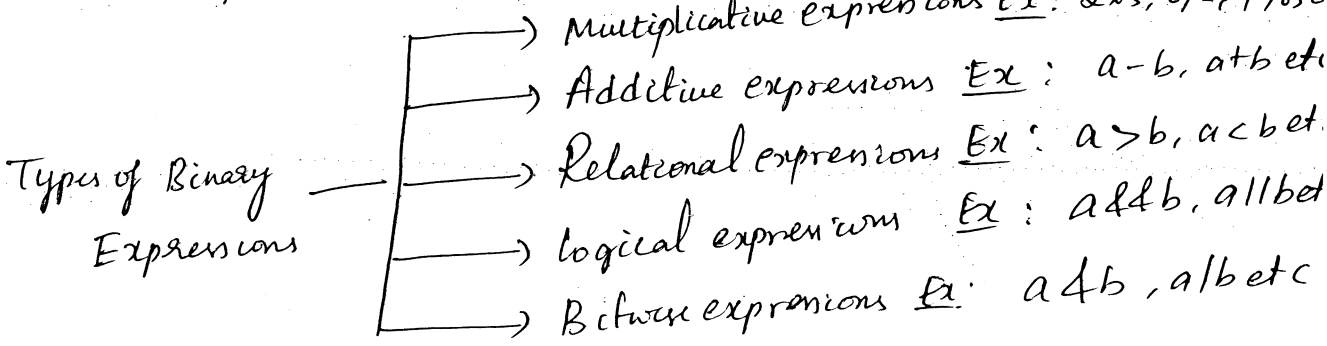
- **Parenthesized expressions:** Any expression or a value enclosed in parantheris must be reducible to a single value and is therefore a primary expression.

  Ex: (2+4*3(4-2))

(2) **Unary expressions:** An expression with only one operand and one operator is called unary expression. The unary operators act on a single operand to produce a value.



(3) **Binary expressions:** An expression containing two operators and an operator is called Binary expression. A binary operator act on two operands to produce a value



4) Ternary expressions: An expression containing three operands and two operators are called ternary expression. Here, the two operators act on three operands.

Ex: a ? b : c

5) Assignment expressions: A statement with assignment operator is called assignment expression. The assignment statements are often referred to as assignment expressions.

Ex:

```
a = 10;
b = a + c;
```

6) Comma expressions: A set of statements separated by commas are evaluated from left to right one after the other. Such statements are called statements with comma operators

Ex: a = 10, b = 30, c = 40;

Unit-2 : Branching Constructs, looping Constructs and Arrays

* Statements

A Statement is a programming construct That is used to inform the computer to perform an action when a program is executed.

Statements are classified into 3 types

(1) Expression statement

An expression followed by Semicolon is called an expression statement. An expression statement includes

- Statements with assignment operator

Examples

```
area = length * breadth ;
a = 3 ;
Sum = Sum + i ;
```

- Statements consisting of only expressions

Ex:
```
j++ ;
a++ ;
++b ;
--b ;
```

- Statements that invoke the functions

Ex:
```
printf("Enter the number") ;
Scanf("%d", &num) ;
```

(2) Compound statement

Sequence of Statements (one or more statements) enclosed within a pair of braces '{' and '}' is called a compound statement. The compound statement is also called block statement.

(3) Control statement

The order in which the statements are executed is called control flow. The statements that are used to control the flow of execution of the program are called control statement. Control statement are of 3 types

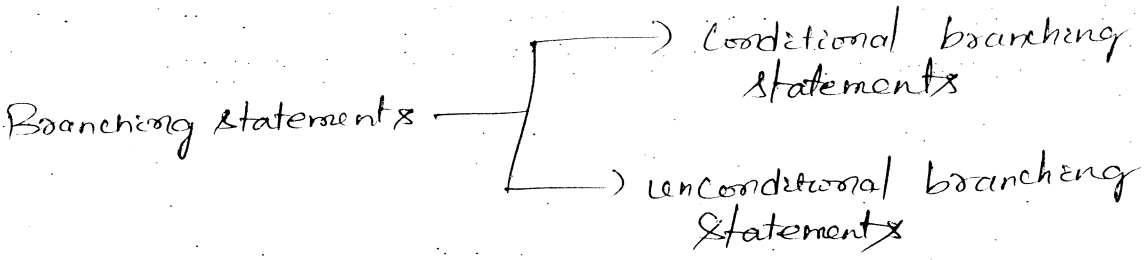
(i) Sequential statements: The programming statements that are executed sequentially.

(ii) Branching statements: Ex: if, if-else etc.

(iii) Looping statements Ex: for, while, do-while

* Branching Constructs/Statements

The statements that transfer the control from one place to other place in the program with or without any condition are called branching statements



* Conditional Branching statements

Normally, all the statements are executed one after the other. However, sometimes the user wants to execute certain statements when some condition is met or the user may skip the execution of some statements when some other condition is met. These statements that transfer the control from one place to other place so as to execute a set of ~~statem~~ instructions if some condition is met or to skip the execution of some statements if the condition is not met are called Conditional branching statements. They are also called selection statements or decision statements.

Conditional branching statements
- simple-if (single selection)
- if-else (two way selection)
- nested-if
- else-if-ladder (multiway selection)
- switch (multi-way selection)

(1) The if-statement (One way selection / Decision statement)

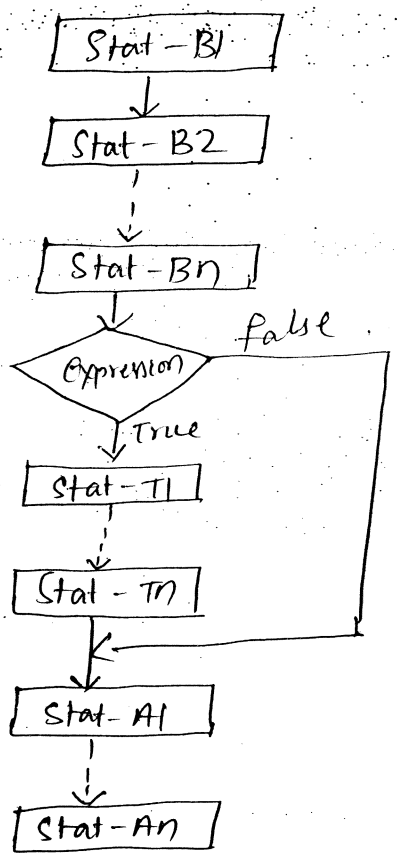
C Syntax

```
Stat-B1;
Stat-B2;
  :
Stat-Bn;
```
Sequential statements

```
if (expression)
{
    Stat-T1;
      :
    Stat-Tn;
}
```
executed if the condition is true

```
Stat-A1;
  :
Stat-An;
```

Flow-chart



Definition: When a set of statements have to be executed or skipped when an expression is evaluated to true (non-zero value) or false (zero), then if-statement is used. It is used when we have only one alternative. Hence, it is also called one-way decision / selection statement.

Working:

- The statements B1, ... Bn before if-statement are executed one after the other.

- If the expression is evaluated to non-zero value, it is considered as true and the statements T1, T2, ... Tn are executed. If more than one statement has to be executed, all those statements must be enclosed within braces i.e., within '{' and '}'.
- If the expression is evaluated to zero, the statements T1, T2, ... are skipped and control comes out of the if statement
- The statements A1, A2, ... An following if-statement are execu

Rules for if-statement

- The keyword if must be followed by an expression and the expressi must be enclosed within parentheses.

  Ex:
  ```
  if(a>b)
      printf("A is greater\n");
  ```
- The expression may have sideEffect i.e., the value of a variables used while evaluating the expression in if-statement may change during the process.

  Ex:
  ```
  if(++count>10)
  {
      count = 0;
  }
  ```
  The ~~value~~ incremented value of count is compared with 10.
- If multiple statements have to be executed when the expression is true, then all those statements must be enclosed within braces. If only one statement has to be executed, then usage of braces is optional.
- No semicolon is required for an if-statement.

C program to check for even number

Algorithm:

Step1: Start
Step2: read num
Step3: Check whether num%2==0, then go to step 4

Step 4: print "even number"

Step 5: Stop

Program:

```
#include<stdio.h>
main()
{
    int num;
    printf("Enter the number\n");
    scanf("%d", &num);
    if(num%2==0)
        printf("Even number\n");
}
```

C program to check whether a person is eligible to vote

```
#include<stdio.h>
main()
{
    int age;
    printf("Enter the age of person\n");
    scanf("%d", &age);
    if(age>=18)
        printf("Eligible to vote\n");
}
```

Advantages: if statement can be used in the following situations

- When a set of statements have to be executed when a condition is satisfied
- When a set of statements have to be skipped when a condition is satisfied.

In general, whenever a set of actions have to be performed or whe a set of actions have to be skipped when a single condition is satisfied, then if-statement is used.

Disadvantages

- if one action has to be performed when the condition is true and another action has to be performed when the condition is false, then if-statement is not recommended. This disadvantage we can overcome using if-else statement.

(2) if-else statement

Definition: If one set of activities have to be performed when the expression is evaluated to true and another set of activities have to be performed when the expression is evaluated to false, then if-else statement is used. if-else statement is used when we have two alternatives.

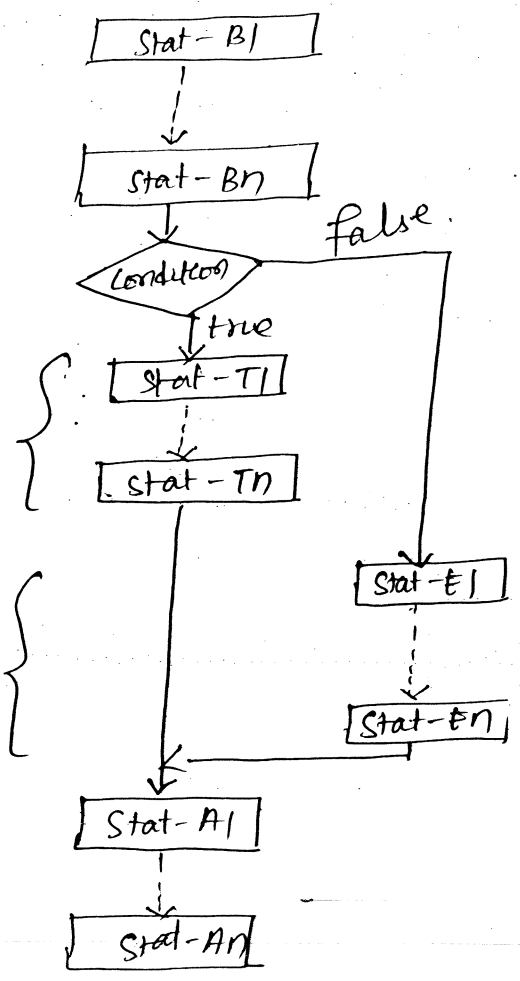
C Syntax

```
Stat-B1;
  :
Stat-Bn;
if(condition)
{
    Stat-T1;
      :
    Stat-Tn;
}
else
{
    Stat-E1;
      :
    Stat-En;
}
Stat-A1;
  :
stat-An;
```

if condition is true execute statements T1 to Tn

if condition is false execute statements E1 to En

Flow-Chart



Working:

- Statements B1, . . . Bn before the if-else statement are executed one after the other
- If expression is evaluated to non-zero value, it is considered as true and T1, ... Tn are executed followed by A1 to An.
- If expression is evaluated to zero value, it is considered as false and the statements E1, . . . En are executed followed by A1 to

Rules for if-else statement

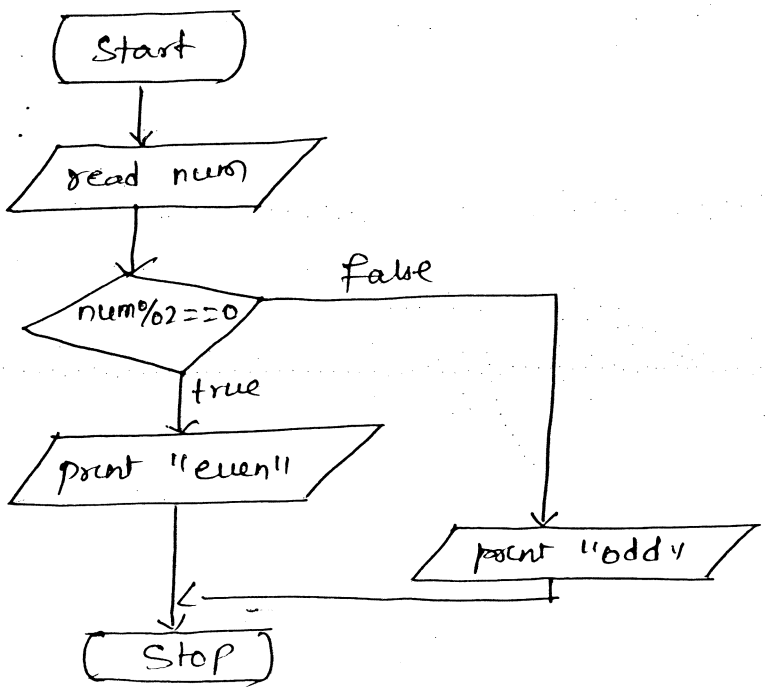
- Same rules as if-statement

Write a C program to check whether a number is odd or even

Algorithm:

Step1: Start
Step 2: read num
Step3: Check whether num%2 is equal to zero, then go to step 4 else step 5
Step4: print "even number"
Step5: print "odd number"
Step 6: Stop

Flowchart:



Program :

```
#include <stdio.h>
main()
{
    int num;
    printf("Enter the number\n");
    scanf("%d", &num);
    if(num%2==0)
        printf("even number\n");
    else
        printf("odd number\n");
}
```

Write a C program to find laogest of 2 numbers

Algorithm :

Step 1: start
Step 2: read a, b
Step 3: Check whether a > b, then
goto step 4 else step 5
Step 4: print "A is greatest"
Step 5: print "B is greatest"
Step 6: Stop

Program :

```
#include<stdio.h>
main()
{
    int a, b;
    printf("Enter two numbers\n");
    scanf("%d %d", &a, &b);
    if(a>b)
        printf("A is greatest\n");
```

```
else
    printf("B is greatest\n");
}
```

C program to check for leap year

A year which satisfies one of the following two cases is a leap year:

- It is divisible by 4 and should not be divisible by 100
- Divisible by 400

If one of the condition is true, it is a leap year. But, if both conditions are false, the year is not a leap year.

Examples:

Year	Case 1 satisfied (true/false)		Case 2 satisfied (true/false)	Remark
	Divisible by 4	not divisible by 100	Divisible by 400	
2000	True	False	True	2nd case satisfied. So, leap year
2004	True	True	—	1st case satisfied. So, leap year
3000	True	False	False	Both condition fail. So, not Leap year

The code for checking leap year can be written as

```
if ((year%4==0) && (year%100!=0)) || (year%400==0)
```

Program

```
#include<stdio.h>
main()
{
    int year;
    printf("Enter the year which needs to be checked\n");
    Scanf("%d", &year);
    if((year%4==0) && (year%100!=0) || (year%400==0))
        printf("%d is a leap year\n", year);
    else
        printf("%d is not a leap year\n", year);
}
```

Disadvantages

- if-else statement cannot be used when we have three or more alternatives.

3) nested-if statement

Definition: An if or if-else statement within another if or if-else statement is called nested if statement. When an action has to be performed based on many decisions involving various types of expressions and variables, then this statement is used. So, it is called multi-way decision statement.

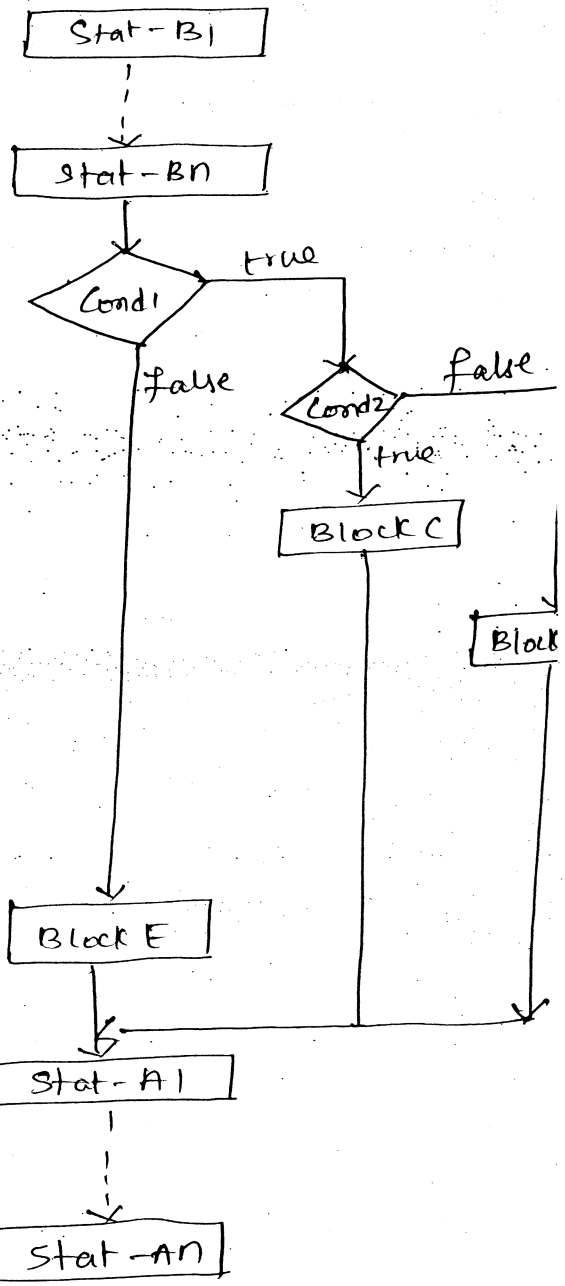
Advantages:

- There are situations involving series of decisions where we are forced to use an if or if-else statement in another if or if-el statement. In such situations, nested if-statements are used

C Syntax

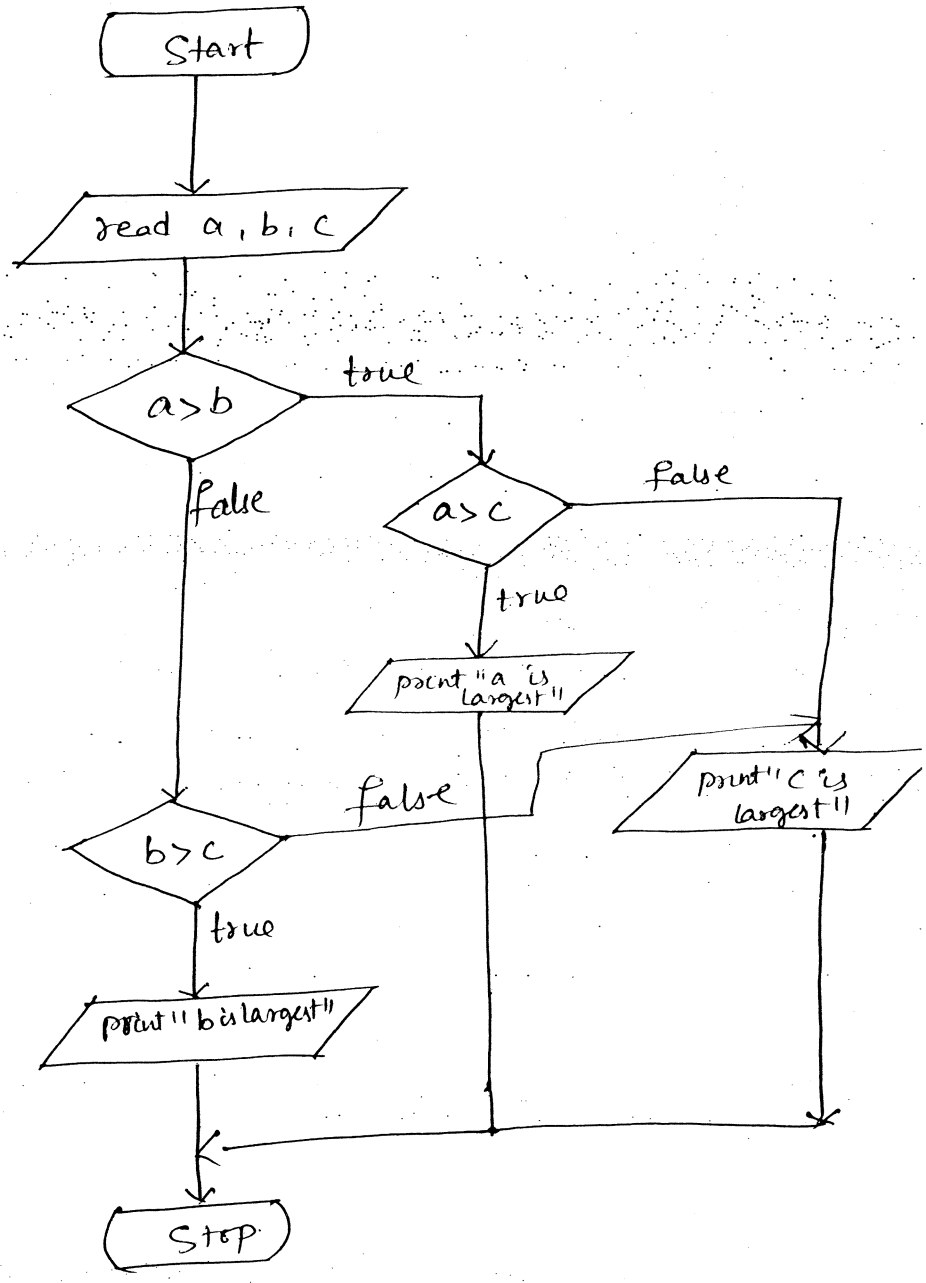
```
Stat - B1;
   :
Stat - Bn;

if(cond-1)
{
    Stat 1;
    if(cond 2)
        Block C
    else
        Block D
}
else
{
        Block E
}
Stat - A1;
   :
Stat - An;
```

Flow-chart



Write a C program to find largest of three numbers using nested - if statement

Flowchart:



Program:

```
#include<stdio.h>
main()
{
        int a, b, c;
        Printf("Enter three numbers\n");
```

```
scanf("%d %d %d", &a, &b, &c);
if(a > b)
{
    if(a > c)
        printf("A is largest\n");
    else
        printf("C is largest\n");
}
else
{
    if(b > c)
        printf("B is largest\n");
    else
        printf("C is largest\n");
}
```

Disadvantages:

- nested if's are difficult to understand and modify
- As depth of nesting increases, the readability of the program decreases.

(4) else-if ladder

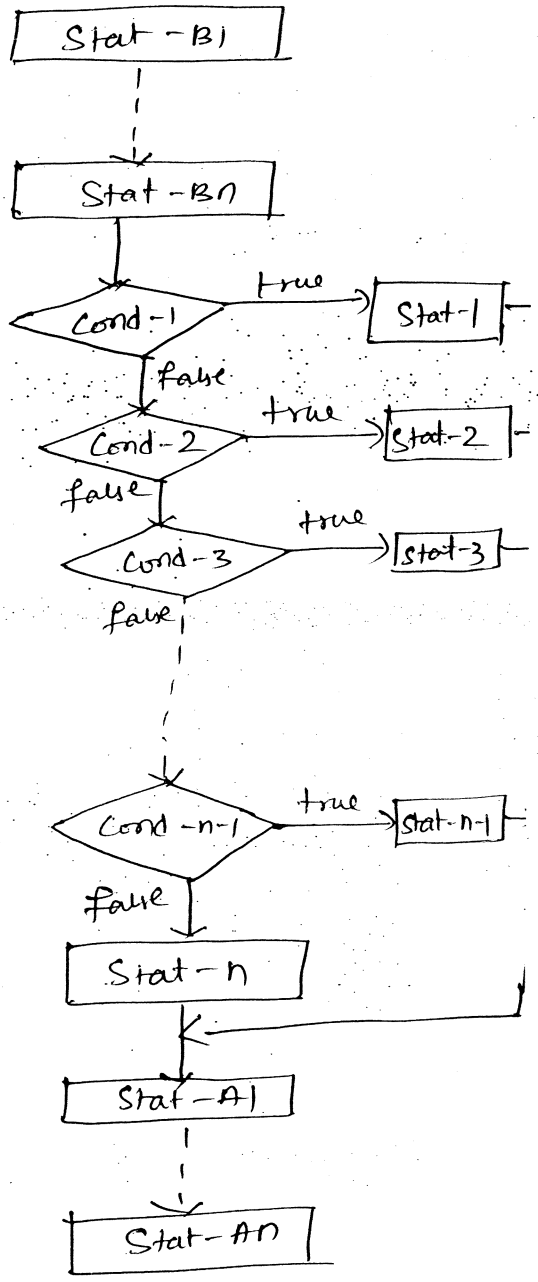
Definition: An else-if ladder is a special case of nested-if statement where nesting takes place only in the else part. When an action has to be selected based on range of values, then this statement is used. So, it is called multi-way decision/selection statement. The orderly nesting of if-els

Statement only in the else part is called else-if ladder.

C Syntax

```
Stat-B1;
   :
Stat-Bn;
if(cond-1)
      Stat-1;
else if(cond-2)
      Stat-2;
else if(cond-3)
      Stat-3;
   :
else if(cond-n-1)
      Stat-n-1;
else
      Stat-n;
Stat-A1;
   :
Stat-An;
```

Flowchart



C program to check whether a number is positive, negative or zero

```
#include<stdio.h>
main()
{
        int num;
        printf("Enter the number\n");
        scanf("%d", &num);
```

```
if (num > 0)
        printf ("positive\n");
else if (num < 0)
        printf ("Negative\n");
else
        printf ("Zero\n");
}
```

C program to find largest of 3 numbers

```
#include<stdio.h>
main()
{
    int a, b, c;
    printf ("Enter three numbers\n");
    scanf ("%d %d %d", &a, &b, &c);
    if (a>b && a>c)
        printf ("A is largest\n");
    else if (b>a && b>c)
        printf ("B is largest\n");
    else
        printf ("C is largest\n");
}
```

* C program to find the roots of Quadratic Equation

Here we can have 3 types of roots.

- if(disc==0) → roots are real and equal
- else if (disc>0) → roots are real and distinct
- else → roots are complex and imaginary

As we need to check for 3 cases/alternatives we will make use of else-if ladder.

Program:

```
#include<stdio.h>
#include<math.h>
main()
{
    float a, b, c, root1, root2, realp, imgp, disc;
    printf("Enter the co-efficients a, b and c\n");
    scanf("%f%f%f", &a, &b, &c);
    disc=(b*b-4*a*c);
    if(disc==0)
    {
        printf("Roots are real & equal\n");
        root1=root2= -b/(2.0*a);
        printf("Root1=%f\n Root2=%f\n",
                root1, root2);
    }
    else if(disc>0)
    {
        printf("Roots are real and distinct\n");
        root1=(-b+sqrt(disc))/(2.0*a);
        root2=(-b-sqrt(disc))/(2.0*a);
        Printf("Root1=%f\n", root1);
        printf("Root2=%f\n", root2);
    }
```

```
else
{
        printf("Roots are complex and imaginary\n");
        realp= -b/(2.0*a);
        imgp= sqrt(fabs(disc))/(2.0*a);
        printf("Root1= %f+i %f\n", realp, imgp);
        printf("Root2= %f- i %f\n", realp, imgp);
}
}
```

C program to display the Grade based on the marks obtained by the student in examination

Range/Marks	Grade
<0 || >100	"Invalid"
0 to 39	F
40 to 49	E
50 to 59	D
60 to 69	C
70 to 79	B
80 to 89	A
90 to 100	O (Outstanding)

```
#include<stdio.h>
main()
{
        int marks;
        printf("Enter the marks\n");
        scanf("%d", &marks);
        if(marks<0 || marks>100)
                printf("Invalid\n");
        else if(marks<=39)
                printf("F Grade\n");
```

```
else if (marks <= 49)
        printf("E Grade\n");
else if (marks <= 59)
        printf("D Grade\n");
else if (marks <= 69)
        printf("C Grade\n");
else if (marks <= 79)
        printf("B Grade\n");
else if (marks <= 89)
        printf("A Grade\n");
else
        printf("O Grade - Outstanding\n");
}
```

(5) Switch statement

The switch statement is used in the following scenarios

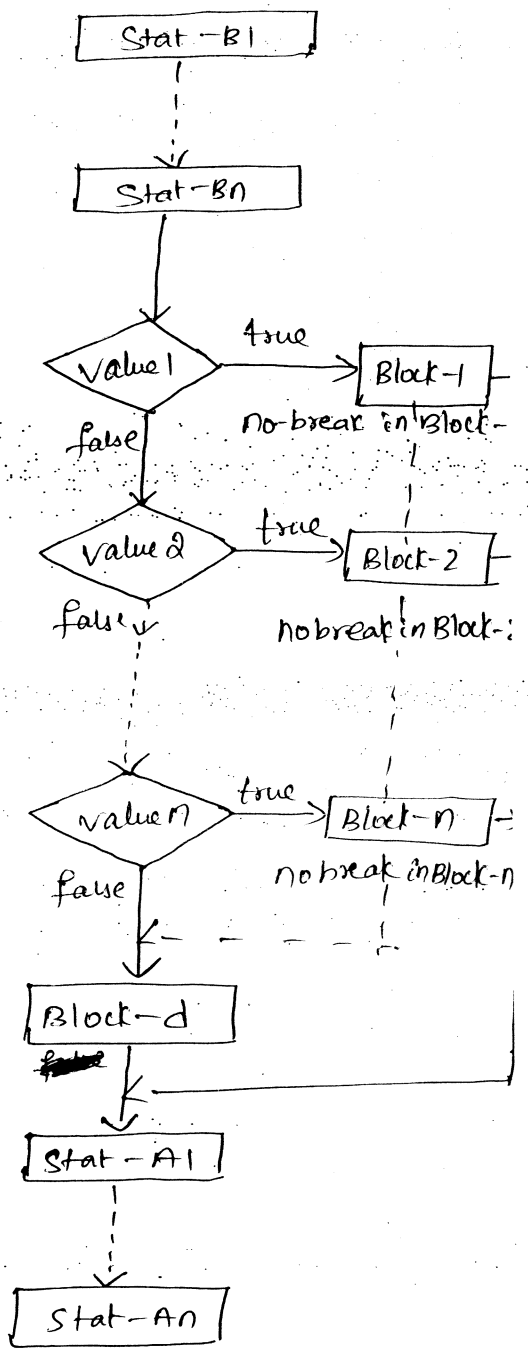
- When a decision has to be made between many alternatives
- When the selection condition reduces to an integer value

Switch statement cannot be used when a series of decision/conditi involve a logical or relational expression. The choice within switch statement can either be any integer value or a character. The choice can also be an expression which results in an integ value. The choice can be integer expression or character as every character also will return a integer value because, every character is associated with unique value called 'ASCII value' and ASCII value is an integer value. Based on this integer value, the control is transferred to a particular case value where necessary statements are executed.

C Syntax

```
Stat-B1;
   :
Stat-Bn;
Switch(choice/expression)
{
    case value-1:
        Block-1;
        break;
    case value2:
        Block-2;
        break;
        :
    case valuen:
        Block-n;
        break;
    default:
        Block-d
}
Stat-A1;
   :
Stat-An;
```

Flowchart



Working:

- After executing the statements B1, ...Bn sequentially, the expression in switch is evaluated to integer value.
- The integer value thus obtained if it matches with any of the values value1, value2, ... valuen, the control is transferred to the appropriate block.

- During execution, if break statement is executed, then the control comes out of the switch statement and the statements A1, A2.. which comes after the switch statement are executed.
- During execution of a particular case, if break is not encountered then control goes to the subsequent cases and the statements under those cases will be executed till the break statement is encountered.
- If the value of the expression does not match with any of the case values value1, value2, ... valuen then, control goes to default label.
- If the value of the expression does not match with any of the case values value1, ... valuen and default label is not there, then, Control comes out of the switch statement and statements following switch will be executed.

Rules to be followed while using switch statement

- The expression that follows the keywork switch must be evaluated to an integer value.
- The expression that follows the keyword case should not contain any variables

  Ex: case 10+2: //valid
  case i+2: //invalid

- Two or more case labels with same value not allowed.
- Two or more case labels can be associated with same statements

  Ex: case 1:
  case 2:
  case 3: printf("Hello\n"); //valid

Advantages:

- Improves readability of the program
- More structured way of writing the program

Disadvantages:

- Used only if the expressions used for checking the condition results in integer value (char is also allowed). If the result of expression is not integer value, switch statement cannot be used.
- Cannot be used when a selection is based on a range of values.

C program to show the simulation of simple calculator

```
#include <stdio.h>
main()
{
    float a, b, res;
    char op;
    printf("Enter operand1, operator and operand2\n");
    scanf("%f %c %f", &a, &op, &b);
    switch(op)
    {
        case '+':
                res=a+b;
                break;
        Case '-':
                res=a-b;
                break;
        Case '*':
                res = a*b;
                break;
        Case '/':
                if(b!=0)
                    res= a/b;
                else
                {
                    printf("Divide by zero exception\n");
                    [redacted];
                }
                break;
        default: printf("Illegal operator\n");
    }
    printf("%f%c%f=%f\n", a, op, b, res);
}
```