C.P. PATEL & F.H. SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME
PGDCA SEM I
PS01CDCA22: C AND DATA STRUCTURE
UNIT 2: LOGIC DEVELOPMENT

Algorithm:

Def: - An Algorithm is finite sequence of step written in a simple English word.

OR

An algorithm is a set of sequence of instruction designed in such a way that if the instruction are execute in specific sequence the desired result will be obtain.

Rules for writing an Algorithm:

1) Start – Stop

- Start indicate begin of an Algorithm.
- Stop indicate end of an Algorithm.
- The first step an Algorithm must be start.
- The last step of an Algorithm must be stop.

2) Input – Output

- Input is indicated by the Read statement.
- Output is indicated by the Write statement.

For above example a & b are the input to an algorithm and it is write with the use of

Read statement.

Like: Read a, b

And the result will be store in 'c' and 'c' is the output of the Algorithm means it is displayed using

write statement

Like: Write c

3) **Condition**:-

In some case we can get more than one output and we select any one from them at that time Condition will be used. Condition is either YES or TRUE and NO or FALSE.

4) **Loop**:-

If you want to repeat the some step for number of time at the time Loop will be used.

Repeat……..Until

➢ **Basic format for writing an Algorithm**

1) Start
2) Read
3) Calculation
4) Output
5) Stop

Examples of Algorithm

1. **Write an Algorithm for an Algorithm for addition of two numbers.**

Step 1 = START
Step 2 = Read a, b
Step 3 = Compute Sum = a + b
Step 4 = Write Sum
Step 5 = STOP

2. **Write an Algorithm for subtraction of three numbers.**

Step 1 = START
Step 2 = Read no1, no2, no3
Step 3 = Compute sub = no1-no2-no3
Step 4 = Write sub
Step 5 = STOP

3. **Write an Algorithm for addition and subtraction.**

Step 1 = START
Step 2 = Read no1.no2
Step 3 = Compute sum = no1+no2
Step 4 = Compute sub = no1-no2
Step 5 = Write sum, sub
Step 6 = STOP

4. **Write an Algorithm for simple interest.**

Step 1 = START
Step 2 = Read P, R, N
Step 3 = Let I = (P*R*N)/100
Step 4 = Write I
Step 5 = STOP

5. **Write an Algorithm for find maximum number of given two number.**

Step 1 = START
Step 2 = Read a, b
Step 3 = if a > b , go to the Step 6
Step 4 = Write ‘a’ is max
Step 5 = go to Step 7
Step 6 = Write b is max
Step 7 = STOP

6. **Write an Algorithm for Read on number from user and find the numbers is odd or even.**

Step 1 = START
Step 2 = Read a, b, c
Step 3 = if a / 2=0 , go to the Step 6
Step 4 = Write ‘a’ is even number
Step 5 = go to the last Step

Step 6 = Write 'a' is odd number
Step 7 = STOP

7. Write an Algorithm for finding max number of given tree number.

Step 1 = START
Step 2 = Read a, b, c
Step 3 = if a > b && a > c ,go to the Step 6
Step 4 = Write a is max
Step 5 = go to the last Step
Step 6 = if b > a && b > c, go to the Step 9
Step 7 = Write b is max
Step 8 = go to the last Step
Step 9 = if c > a && c > b
Step 10= Write c is max
Step 11= STOP

- **Characteristics of an Algorithm :**
  - Every instruction should be precise & unambiguous, (Not double meaning)
  - Each and every instruction should be performed or executed and finite time.
  - One or more instruction can be
  - After performing the instruction the desired result is obtain.

- **Flow – Chart** :-

Definition :-

Flow-chart is a pictorial representation of an Algorithm using standered set of symbol.

Or

Flow-chart is a diagrammatic representation of an Algorithm using standered set of symbol.

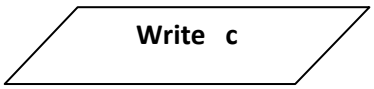
Following symbols are used to draw the flow-chart

Start / Stop:-

Start indicate begin of an flowchart

- Stop indicate end of an flowchart.
- The first step any flowchart must be start.
- The last step of any flowchart must be stop.

Input – Output:-

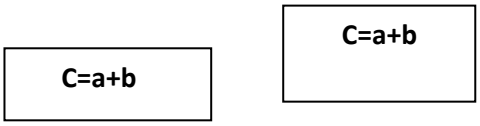


- Input indicates by the Read statement.

- Output indicate by the Write statement.

The point at which value of some data item have to be read or some result written are indicate by input and output box

Process:-

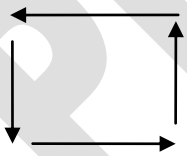


- If we use mathematical symbol for drawing the Flow-Chart the process symbol will be used.
- Process symbol is use to indicate straight forward computation of certain quantity.

Decision (Condition) box:-

- In same situation we get more the one o/p and we select any on from them at that time condition box will be use.
- This box indicate that comparison is to be made based on the result of Comparison.
- The control passes to the one of the two phases **YES** or **NO** and **TRUE** OR **FALSE.**

Flow - Line

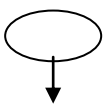


- Flow – line is use to connect different symbol at drawing the Flow-Chart.

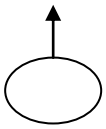
Connector:-

- In some situation the Flow-Chart are not fit in one on page it & if you want to continue
- the same flowchart in second page at that time connector will be use.
- Two type of connecter is use.

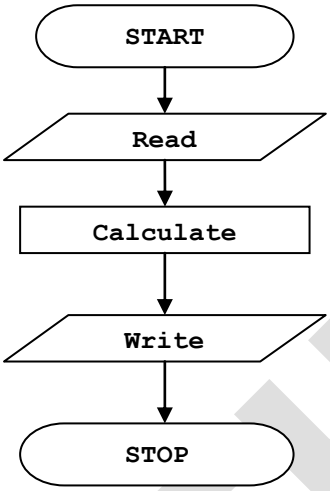
Off page connector :



On page connector

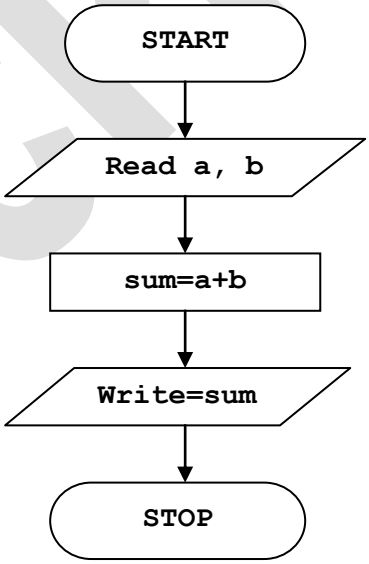


Basic format for drawing flow chart

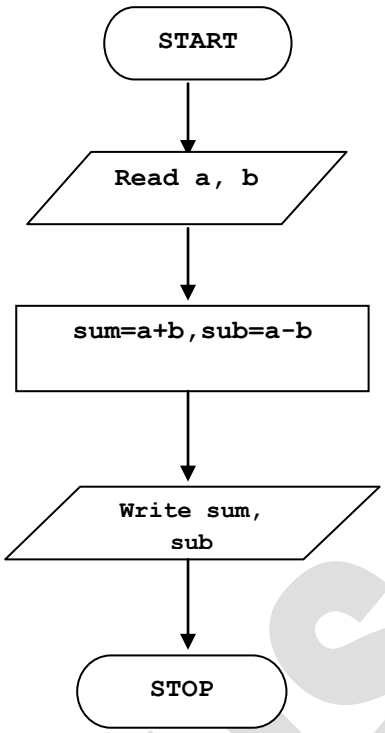


Example of Flow Chart:

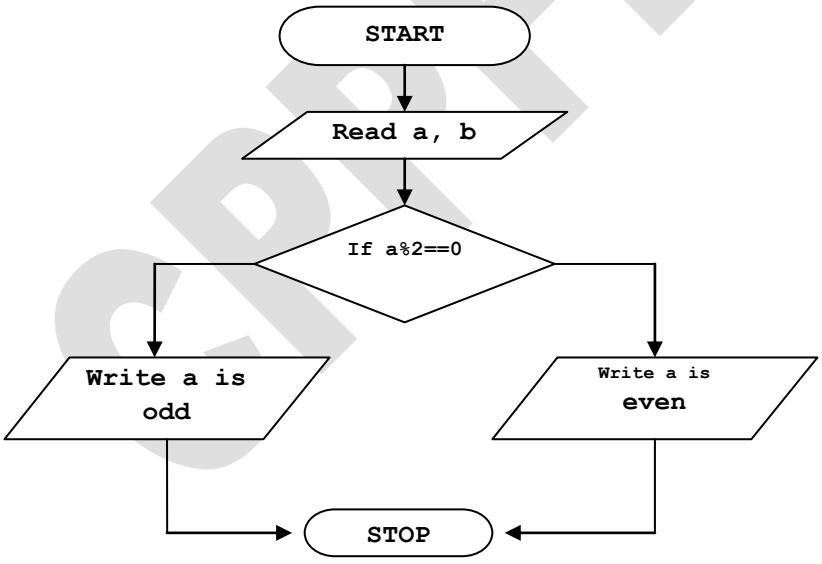
1) Draw flow-chart for addition of two numbers.



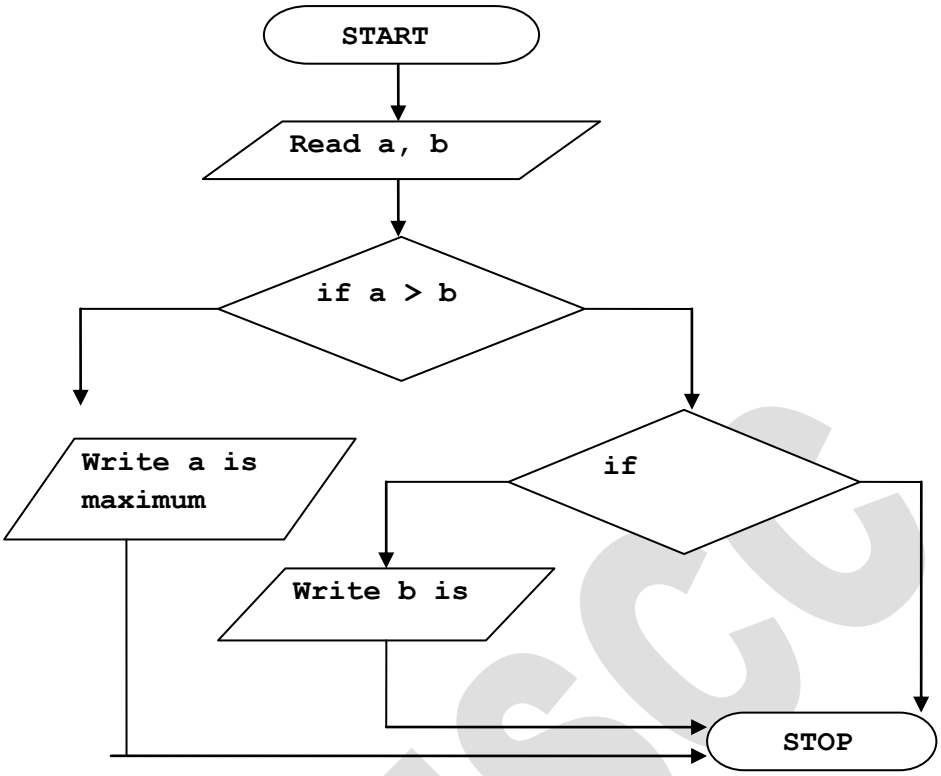
2) Draw a flow-chart for addition and subtraction of two number.



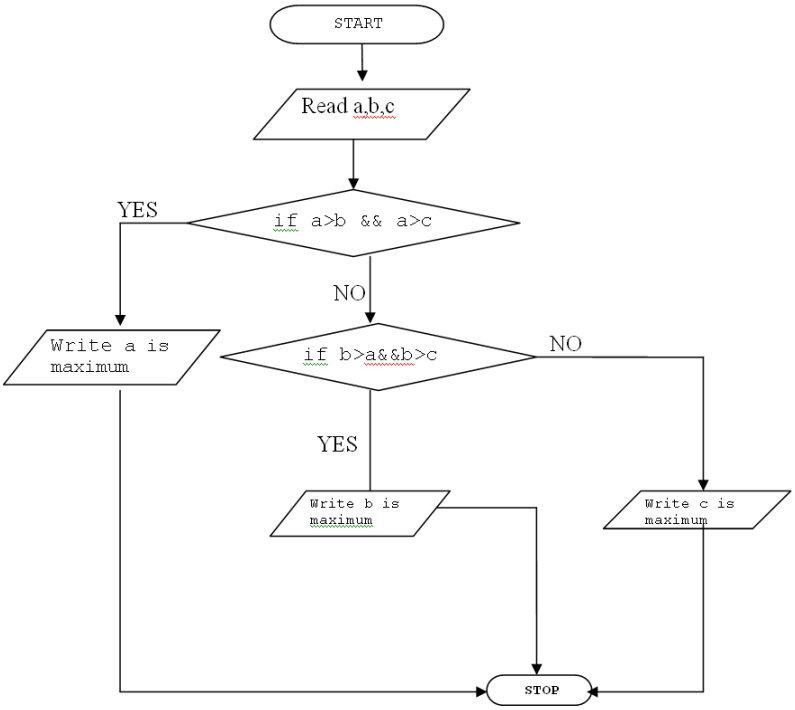
3) Draw a flow-chart for read one number from user and find given no is Odd or Even.



4) Draw a flow-chart to read two number from user and find maximum number them



5) Write a flow-chart to three number from user and find maximum of them.



- **Problem analysis :-**

Before we think of a solution procedure to the problem, we must fully understand the nature of the

Problem and what we want the program to do. Without the comprehension and definition of the problem at hand, program design might turn into a hit or miss approach .we must carefully decide the

Following at this stage,

1) What kind of data will go on.
2) What kind of output are needed
3) What are the constrain and conditions under the program has to operate?

- **Basic Structure of C Program**

The basic **structure of C program** is as follow:

Document Section,
Links Section (File),
Definition Section,
Global variable declaration Section,

void main()
{
declaration part
executable part
}

Sub Program Section Function 1 Function 2 . . Function n

Document Section :

- This section contains the set of comment lines which include name of a program author name, creation date and other information.
- Comment line can be given in two way.
  1) Single comment line //
  2) Multiple comment line /*……
     ……*/

1. **Links Section (File)**

- It is used to link the required system libraries or header files to execute a program.
- The link section provide the instruction to the compiler to link function from system library
- Generally **#include** statements comes into this section.

2. **Definition Section** :

- It is used to define or set values to variables.
- This section defines all the symbolic constant.

3. **Global variable declaration Section** :

- It is used to declare global or public variable.
- The variables are use in more then one function such variable are called global variable. and they are defined in global declaration section.

4. **void main()** :

- Every C program has one main function.
- Main function used to start of actual C program.
- It includes two parts:
  1) Declaration Part
  2) Executable Part

1) Declaration Part

- This part declares all the variable use in the program.

2) Executable Part

- There is at least one statement in the executable part for the output.
- This two part must be appear between opening and closing curly { } brackets.
- The program execution stating form the **"{"** and closing by **"}"**.

5. **Sub-Program Function section**:

- This section contain all the user define function that are call from the main( ).

Variables:

- A variable is a data name which is used to store data and may change during program execution.
- It is opposite to constant.
- Variable name is a name given to memory cells location of a computer where data is stored.

Rules for declaration of variables:

- First character should be letter or alphabet.
  - int @a; //invalid variable
  - int b; //valid variable
- Keywords are not allowed to use as a variable name.
  - int for; //invalid variable name
- White space is not allowed.
  - float area of circle; //invalid variable
- C is case sensitive i.e. UPPER and LOWER case are significant.
  - int TOTAL,total // valid variable here we can store value in both the variable
- Underscore is allowed between two characters.
  - Float area_of_circle, //valid variable
- The length of identifier may be up to 31 characters but only the first 8 characters are significant by compiler.

Syntax for variable Declaration:

```
data_type identifier;
        Or
data_type identifier_1, identifier_2……, identifier_n;
```

Variables Example:

`int a;` (An integer variable)
`char b;` (A character variable)
`float m ,n,o,p;` (multiple float variables)

Initialization of the Variables:

- The assignment of the values to the variables is known as the initialization of variables.
- The variable in the C can be initialized at the declaration time for example:

```
int a=100;
float d=12.56;
char c='m';
```

Keywords:

- Keywords are the system defined identifiers.
- All keywords have fixed meanings that do not change.
- White spaces are not allowed in keywords.
- Keyword may not be used as an identifier.
- It is strongly recommended that keywords should be in lower case letters.

There are totally **32(Thirty Two) keywords** used in a C programming.

int	float	Double	Long
short	signed	Unsigned	const
if	else	Switch	break
default	do	While	For
register	extern	static	struct
typedef	enum	return	sizeof
goto	union	auto	Case
void	char	continue	volatile

Escape Sequence Characters (Backslash Character Constants) in C:

- C supports some special escape sequence characters that are used to do special tasks.
- These are also called as 'Backslash characters'.
- Some of the escape sequence characters are as follow:

Character Constant	Meaning
\n	New line (Line break)
\b	Backspace
\t	Horizontal Tab
\f	Form feed
\a	Alert (alerts a bell)
\r	Carriage Return
\v	Vertical Tab
\?	Question Mark
\'	Single Quote
\''	Double Quote
\\	Backslash
\0	Null

Expressions & Manipulation

- An expression is combination of variable, constant and operator arrange as per the type of the Language.
- C language can handle any complex expression.
- Some examples are shown in below table.

Algebraic expression	C expression
A × B – C	A * B – C
(M + N)(X + Y)	(M + N) * (X + Y)
$3X^2 + 2X + 1$	3*X*X +2*X+1

- **Evaluation of Expression**:-
- Expression can be evaluated using the assignment operator.
- Variable = expression;
- Variable is any valid c variable name. When the statement is encountered the expression is evaluated first.
- All the variable used in the expression must be assigned value before they are use.
- **Ex**: - 1) X = A * B – C;
  2) Y = B / C * A;
  3) Z = A – B / C + D
- Expression can be evaluated form LEFT to RIGHT using the rules of the precedence of operator..
- Two priory levels are use in expression.
  1) High Priority (*, /, %)
  2) Low Priority (+,-)

Ex: - x = a-b/3+c*2-1 Where a =9 ,b = 12, and c = 3

Step 1:- x = 9 – 12/3 +3*2 – 1

Step 2:- x = 9 – 4 +3*2 – 1

Step 3:- x = 9 – 4 + 6 – 1

Step 4:- x = 5 + 6 – 1

Step 5:- x = 11 – 1

Step 6:- x = 10.

Data Types in C:

"Data type can be defined as the type of data of variable or constant store."
When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.
Followings are the most commonly used data types in C.

- **`char`:**
  - The size of a char variable is a single byte.

- The control string used for it is %c.
- It can hold one character in the local character set.

- **`int`**:
  - The size of an integer variable is 2 bytes.
  - The control string used for it is %c.
  - The range of it is -32768 to +32767
  - It can store non floating values
    - Ex: 120, 16500, 9 etc.
- **`float`***:*
  - It can store a single-precision floating point values.
  - Its size is 4 bytes.
  - The control string for float is %f.
  - The range for float is $-3.4e^{38}$ to $+3.4e^{38}$

Keyword	Format Specifier	Size	Data Range
`char`	`%c`	1 Byte	-128 to +127
`unsigned char`	`<-- -- >`	8 Bytes	0 to 255
`int`	`%d`	2 Bytes	-32768 to +32767
`long int`	`%ld`	4 Bytes	-2^{31} to $+2^{31}$
`unsigned int`	`%u`	2 Bytes	0 to 65535
`float`	`%f`	4 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$
`double`	`%lf`	8 Bytes	$-1.7e^{38}$ to $+1.7e^{38}$
`long double`	`%Lf`	12-16 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$

Type Qualifiers:

When qualifier is applied to the data type then it changes its size or its size.

- **Size qualifiers**
- **short, long for example:**
  - `long int`
  - `long double`
- **Sign qualifiers:**
- **signed, unsigned for example:**
  - `signed int`
  - `unsigned int`

Operators

"An operator is special symbol that tells the computer which operation is to be performed

either mathematical or logical."

C operators can be classified into a number of categories. They include

1. Arithmetic Operator
2. Relational Operator
3. Logical Operator
4. Assignment Operator
5. Increment and Decrement Operator
6. Conditional Operator
7. Bitwise Operator
8. Special Operator

1. Arithmetic Operator:

C provides all the basic arithmetic operators. These can operate on any built-in numeric data type of C. C has unary arithmetic operators.

The arithmetic operators and their meaning are shown below:

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Integer Arithmetic: When both the operands in a single arithmetic expression (such as a + b) are integers, the expression is called integer expression, and the operation is called integer arithmetic. Integer arithmetic always results an integer value.
Example: If a & b are integer & its value is a = 14 and b=4 then we have the following results:
a -b results 10, a + b results 18;
a*b results 56,
a/b results 3 (decimal part truncated) and
a % b results 2 (remainder).

Real Arithmetic: An arithmetic operation involving only real operands is called real arithmetic.
Example: If a, b and c are floats then we will have
a = 20.5, b=6.4
Then a+b becomes 26.9,
a-b becomes 14.1,
a*b becomes 131.2 and
a/b becomes 3.20313.

Mixed-Mode Arithmetic: When one of the operand is real and the other is integer the expression is called Mixed-mode arithmetic expression.
If either operand is of the real type, then only the real operation is performed and the result is always a real number. Thus 15/10.0 gives 1.5 while 15/10 gives 1.

This is the example of Arithmetic operators in C:

```
#include <stdio.h>
 void main()
 {
        int a = 100;
        int b = 3;
        int c;
        c = a + b;
        printf( "a + b = %d\n", c );
        c = a - b;
        printf( "a - b = %d\n", c );
        /* multiplication performed before call to printf() */
        printf( "a * b = %d\n", a * b );
        c = a / b;
        printf( "a / b = %d\n", c );
        c = 100 % 3;
        printf( "a % b = %d\n", c );
 }
```

This program gives the following output:

```
a + b = 103
a - b = 97
a * b = 300
a / b = 33
a % b = 1
```

2. Relational Operator

A relation operator is used to make comparisons between two expressions.
C language supports six relational operators. These operators and their meaning are shown below.

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

An expression containing a relational operator is termed as relational expressions. The value of relational expression is either true or false.
Example:

A = 5;
B = 10;

The condition (A < B) evaluates as a True While (A==B) evaluates as a False.

- **Here is an example of how to use these operators.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b;
      clrscr();
      printf("enter the a");
      scanf ("%d",&a);
      printf("enter the b");
      scanf ("%d",&b);
      if(a!=b)
            {
            printf("A and B are not equal\n");
            if(a>b)
            {
            printf("a is big");///true
            }
            else if(a<b)
            {
            printf("a is small");//false
            }
            }
      else
      {
      printf("a and b are equal");
      }
```

```
    getch();
      }
```

3. Logical Operator :-

Logical operators are useful in combining one or more conditions. C language has the following three logical operators.

Operator	Meaning
&&	Logical AND(True only if all conditions are true, otherwise false)
||	Logical OR(False only if all conditions are false, otherwise true)
!	Logical NOT(True if exp. is false or vice - versa)

Example on Logical AND:

A =5; B=10; X =10;

The condition (A < B && X == 10) evaluates true.
The condition (A < B && X!= 10) evaluates false.
The condition (A > B && X ==10) evaluates false.
The condition (A > B && X!= 10) evaluates false.

Example on Logical OR:

A =5; B =10; X = 10;

The condition (A < B || X ==10) evaluates true.
The condition (A < B || X != 10) evaluates true.
The condition (A > B || X ==10) evaluates true.
The condition (A > B || X != 10) evaluates false.

Example on Logical NOT:

The condition !(A <B) evaluates false and !(A> B)evaluates true

4. Assignment Operator(=)

Assignment operators are used to assign the result of an expression to a variable. The general form of assignment operator is

```
Variable = expression;
```

C has a set of "**Shorthand"** assignment operators of the form

```
V op = exp ;
```

Where V is a variable, exp is an expression and op is a binary arithmetic operator.
The operator op= is known as the shorthand assignment operator.

Example:

A += 1; same as A = A + 1;
B+= 1; same as B= B+ 1;
A += B + 3; same as A = A + (B + 3);
A *= B; same as A = A * B;
A /= B; same as A = A / B;

The use of shorthand assignment operator has three advantages:

- What appears on the left hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- The statement is more efficient

5. Increment and Decrement Operator :-

The C language uses two operators for incrementing and decrementing variables.
These are ++ and – operators.
The operator ++ adds 1 to the operand and -- subtract 1 from the operand.

Syntax:

Prefix	Postfix
++VariableName	VariableName++
--VariableName	VariableName—

When used a prefix, the value of a variable is incremented/ decremented before used in expression. But when used as a postfix, its value is first used in expression and then the Value is incremented/ decremented.

Example:

```
A=5
A++
```
The value of a is 6.

```
int a = 0, b = 10;
a = ++b;
a = b++;
```
The value of a becomes 11 &
The Value of a becomes 10 &
b is 11.

6. <u>Conditional Operator</u> :-

It is also known as **Ternary Operator**.
The general form of ternary operator is

```
exp 1 ? exp2 : exp3 ;
```
Where exp1, exp2, exp3 are expressions.

The operator **? :** Works as follows:
exp1 is evaluated first.

If it is non-zero (true), then the expression exp2 is evaluated and becomes the value of the expression.

If it is zero (false), then the expression exp3 is evaluated and becomes the value of the expression.

Example:

```
A = 10;
B = 15;
X = (A>B) ? A : B;
```

In this example, X will be assigned the value of B. This can be achieved using the if...else statements as follows.

```
if ( A > B)
     X=A;
else
     X = B;
```

- **<u>Input Output Statements in C:</u>**
  - The `printf()` and `scanf()` statements are used to manage the inputs and output in the C programming language.

  - **<u>The printf():</u>**
    - This statement is used to give the formatted output to the user on the output screen.
    - This statement can print one line of string on the screen.
    - The format for printf() statement is as below:

```
printf("control string",arg1,arg2,arg3…..);
```

    - The control string is also known as format specifier.
    - And arg1….. etc are the variable names containing the values.
    - The general meaning for *control strings* are as below:

Code	Format
%c	Character.
%d	Signed decimal integers.
%i	Signed decimal integers.
%f	Decimal floating point.
%g	Uses %e or %f, whichever is shorter.
%G	Uses %E or %F, whichever is shorter.
%s	String of characters.
%%	Prints a % sign.

- Example of the printf() statement:
  1. `printf("Hello all");`
     - Use of string constant.
  2. `int x=190;`
     `printf("%d",x);`
     - This will print the number 190 on the screen.
  3. `int a=190,b=200,c;`
     `c=a+b;`
     `printf("The sum is %d",c);`
     - This will print the value of sum.
- The format specifier can take the following format:
  - `% w d`
    - Where w is minimum field width of the output.
    - The number will be written into the right justification.
  - For example:
  - `printf("%d",1234);` will display output as:

1	2	3	4

  - `printf("%6d",1234);` will display output as:

		1	2	3	4

  - `printf("%-6d",1234);` will display output as:

1	2	3	4		

- **The `scanf()` statement:**
  - This statement is used to get formatted input from the user.
  - The general format for this statement is as below:
    - `scanf("control string",&var1,&var2,.....&varn);`
  - The control string contains the format for data being received from the user.
  - The ampersand (**&**) symbol is the operator used for specifying the address of the variable.
  - The example of the scanf statement is as below:
    - `scanf("%d",&a);`
  - This statement takes the value of the variable a from the user.
  - **The formatted input:**
  - We can get the formatted input from the user using the scanf statement.
  - The syntax for this is:
    - `%wd`
  - In this **w** is the maximum width of the value a variable can contain.
  - And **d** is the data type specification of the variable.
  - For example:
    - `scanf("%2d %4d",&a,&b);`
    - User passes the value as 50 1000 then 50 will be assigned to a and 1000 will be assigned to b.

- For the same statement if user passes values 1532 and 50 then 15 will be assigned to a and 32 will be assigned to b. This can create the problems in the program.
- To avoid this problem the use of simple format specification is advisable. For example:
  - `scanf("%d %d",&a,&b);`
- This statement will assign the values to the variables as user has passed.

o The general meaning for *control strings* and *format specifications* are as below:

Code	Format
%c	Character.
%d	Signed decimal integers.
%i	Signed decimal integers.
%f	Decimal floating point.
%g	Uses %e or %f, whichever is shorter.
%G	Uses %E or %F, whichever is shorter.
%s	String of characters.
%%	Prints a % sign.

Assignment Statement :-

Value can be assigned to variable using the assignment operator (=) as follow.

```
Variable_name = constant ;
```

This statement is use to assign the value to variable .

Ex;- 1) age = 12;
2) balance =13444;
3) sum = 0 ;

C language also support the multiple assignment statement in one line

4) balance =13444; sum = 0 ;

An assignment statement implies that the value of the variable on left of the "equal sign" Is set to the value of the variable on right side.

5) Year = Year + 1 ;

It is also possible to assign a value to a variable at the time the variable is declared. This takes the following form

```
Data_type Variable_name = constant;
```

6) **int** age = 12;
7) **int** balance =13444
8) **int** sum = 0;

Control strategies, Conditions

- C program executes program sequentially.
- Sometimes, a program requires checking of certain conditions in program execution.
- C provides various key condition statements to check condition and execute statements according conditional criteria. This is known as *'Decision Making Statements'* or 'Conditional Statements.'

  Followings are the different conditional statements **used in C.**

  1. If Statement
  2. If-Else Statement
  3. Nested If-Else Statement
  4. Switch Case

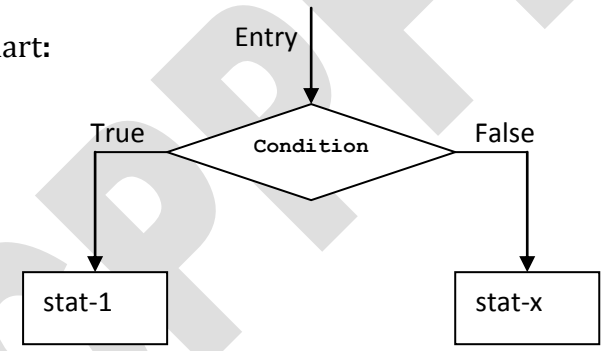
If Statement:

- This is a conditional statement used in C to check condition or to control the flow of execution of statements.
- This is also called as 'decision making statement or control statement.
- The execution of a whole program is done in one direction only.

  Syntax:

```
if (condition)
{
        Stat-1;
}
Stat-X;
```

Flowchart:



- In above syntax, the condition is checked first.
- If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it.
- If it returns false, then program skips the braces.
- If there are more than 1 (one) statements in if statement then use { } braces else it is not necessary to use.
- The example of the if statement is:

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

```
   int num1;
   clrscr();
   printf("\n\t\tenter the year:");
   scanf("%d",num1);
   if (num1%4==0)
{
   printf("\n\t\tenter the year is leap year");
   }
   getch();
}
```

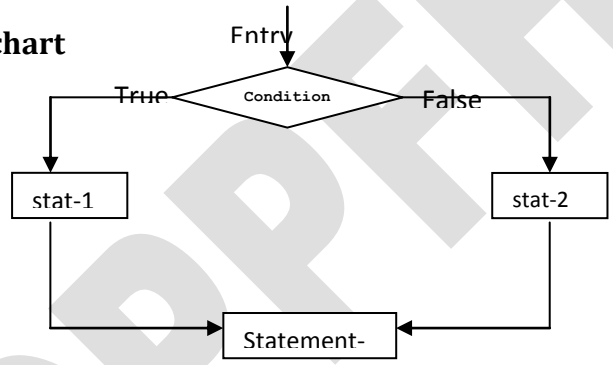
If-Else Statement:

- This is also one of the most useful conditional statements used in C to check conditions.

Syntax:

```
if (condition)
{
   true stat-1;
}
else
{
   false stat-2;
}
statement-x
```

Flowchart



- In above syntax, the condition is checked first.
- If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it.
- If it returns false, then it executes the else part of a program.
- The example of this statement is as below:

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int num1;
   clrscr();
   printf("\n\t\t enter the year:");
   scanf("%d",&num1);
   if (num1%4==0)
   printf("\n\t\t enter the year is leap year");
```

```
    else
    printf("\n\t\t enter the year is not leap year");
    getch();
}
```

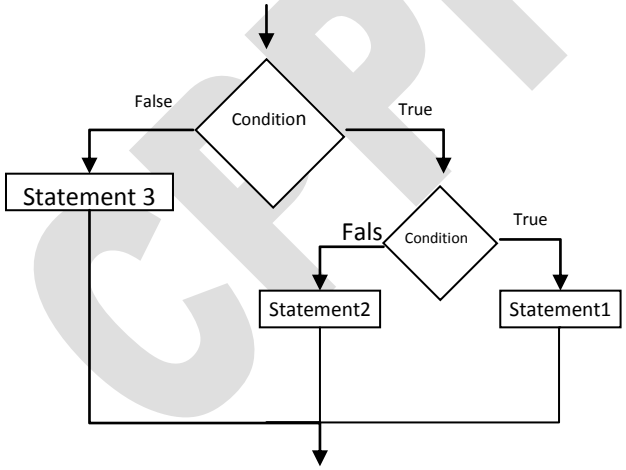
Nested If-Else Statement:

- It is a conditional statement which is used when we want to check more than 1 condition at a time in a same program.
- The conditions are executed from top to bottom checking each condition whether it meets the conditional criteria or not.
- If it found the condition is true then it executes the block of associated statements of true part else it goes to next condition to execute.

Syntax:

```
if (condition)
{
    if (condition)
    {
          statement1;
    }
    else
    {
          statement2;
    }
}
else
{
    Statement3;
}
```

Flowchart:



- In above syntax, the condition is checked first.
- If it is true, then the program control flow goes inside the braces and again checks the next condition.
- If it is true then it executes the block of statements associated with it.
- If it is false the else part will be executed.
- **The program of this statement is as below:**

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

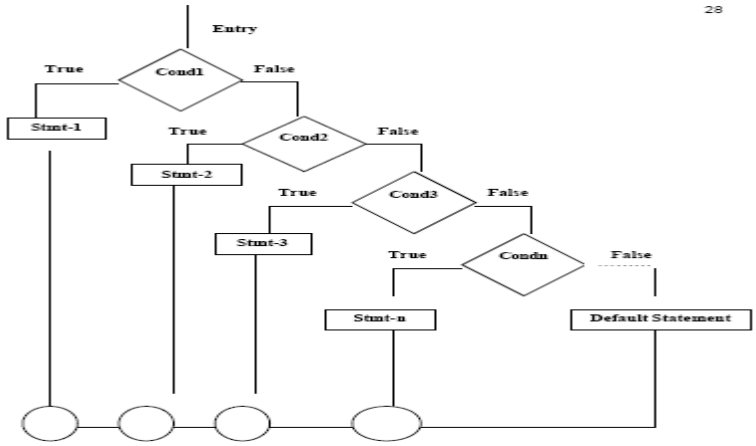
```
{
   int num1;
   clrscr();
   printf("\n\t\t enter the year:");
   scanf("%d",&num1);
   if (num>0)
{
   if((num1%2)==0)
   printf("The number is even");
   else
   printf("The number is odd");
}
else
{
   if(num==0)
   printf("The number is zero");
   else
   printf("Number entered is a negative number");
   }
   getch();
}
```

Else-If Ladder:

- When there is the need for checking so many conditions, the else if ladder is used to decrease the complexity of the program.
- The construction syntax for this statement is:

```
if (expression)
{
       statement
}
else if (expression)
{
       statement
}
else if (expression)
{
       statement
}
else
{
       statement
}
```

Flowchart:



- This sequence of if statements is the most general way of writing a multi-way decision.
- The *expressions* are evaluated in order; if an *expression* is true, the *statement* associated with it is executed, and this terminates the whole chain.
- As always, the code for each *statement* is either a single statement, or a group of statements.
- The last else part handles the ``none of the above'' or default case where none of the other conditions is satisfied.
- Sometimes there is no explicit action for the default; in that case the trailing else statement can be omitted.
- **The program for else if statement is as below:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
   float s1, s2, s3,tot;
   int no;
   clrscr();
   printf("\t\tEnter Roll Number: ");
   scanf("%d",&no);
   printf("\n Enter Marks of IT: ");
   scanf("%f",&s1);
   printf("\n Enter Marks of BS: ");
   scanf("%f",&s2);
   printf("\n Enter Marks of wpd: ");
   scanf("%f",&s3);
   tot=s1+s2+s3;
   per=tot/3;
   if (per>=75)
   printf("\t ---A+ GRAD---");
   else if((per>=68) && (per<75))
   printf("\n ---A GRAD---");
   else if ((per>=60) && (per<68))
   printf("\n ---B GRAD---");
   else if ((per>=50) && (per<60))
   printf("\n ---C GRAD---");
   else if ((per>=40) && (per<50))
   printf("\n ---D GRAD---");
   else
   printf("\n ---FAIL ---");
```

```
      getch();
}
```

Switch case Statement:

- This is a multiple or multi way branching decision making statement.
- When we use nested if-else statement to check more than 1 condition then the complexity of a program increases in case of a lot of conditions.
- Thus, the program is difficult to read and maintain.
- So to overcome (solve) this problem, C provides 'switch case'.
- Switch case checks the value of a expression against a case values.
- If condition matches the case values then the control is transferred to that point.

Syntax:

```
switch (expression)
{
   case expr1:
        statements;
        break;
   case expr2:
        statements;
        break;

             --------------------
             ------------------
   case expr n:
        statements;
        break;
   default:
        statements;
}
```

- In above syntax, switch, case, break are keywords.
- expr1, expr2 are known as 'case labels.'
- Statements inside case expression need not to be closed in braces.
- Break statement causes an exit from switch statement.
- Default case is optional case. When neither any match found, it executes.

The sample program for switch case is as below:

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
char ch;
clrscr();
printf("1. Check Spelling\n");
printf("2. Correct Spelling Errors\n");
printf("3. Display Spelling Errors\n");
printf("Strike Any Other Key to Skip\n");
printf(" Enter your choice: ");
ch = getchar(); /* read the selection from the keyboard */
switch(ch)
{
case '1':
```

```
printf("Wrong Spelling");
break;
case '2':
printf("Logic Errors");
break;
case '3':
printf("The errors");
break;
default:
printf
("No option selected");
}
getch();
}
```

Rules for declaring switch case:

- The case label should be integer or character constant.
- Each compound statement of a switch case should contain break statement to exit from case.
- Case labels must end with (:) colon.

Advantages of switch case:

- Easy to use.
- Easy to find out errors.
- Debugging is made easy in switch case.
- Complexity of a program is minimized.