Control Structures

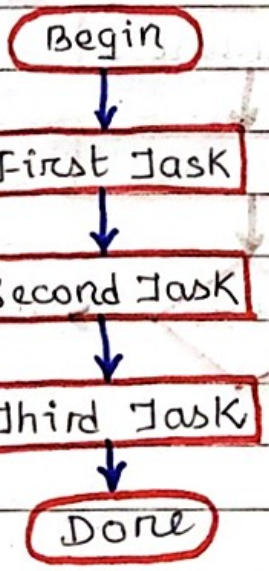
Introduction:

C is a structural and procedural programming language. Program execute in top down approach. C program tends to flow from top left to right bottom sequentially. But always sequential flow of program execution is not useful. Depending upon the requirement of the user, programmer can change the flow of program execution.
Thus, the statements which are used to change the flow of program execution are known as control structures. It controls the flow of program execution. There ar

Types of control structure:

1.) Sequence 2) Branching 3.) Looping

1.) **Sequence:** This is the type of control structure in which program flows from top to bottom sequentially without changing the flow of program execution. None of the program statements are skipped or repeated. Every statements are executed one after another.



e.g:

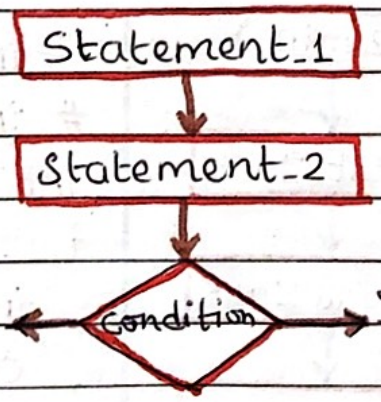
```
#include <stdio.h>
# include <conio.h>
int main()
{
  int a=5, b=3, c;
  c=a+b;
  printf("Value is %d", c);
  return 0;
}
```

2.) Branching / Selection :

It is the type of control structure where we can change the flow of program execution based on requirements of the user. The flow of program can be changed with or without condition. Branching statements are also called selection. These statements are used if we want to skip some statement depending upon the requirement. Basically, there are two categories in Branching :

a) Conditional Branching :

As name suggest, these types of branching changes the flow of program execution depending upon the condition or criteria supplied by the user. The condition supplied by the user plays key role to change the program flow. It can execute several statements depending upon whether the condition is true or false.



There are various types of conditional branching in C programming :

- if statement
- if else statement
- else ladder statement
- nested if statement
- switch statement

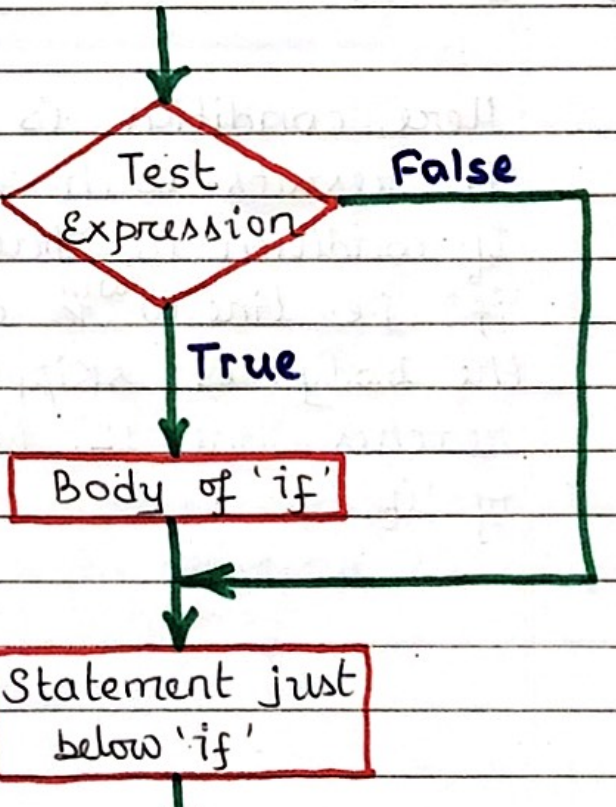
(i) if statement :

This is the most simple of the branching statements. The 'if' statement is a powerful decision making statement and it is used to control the flow of execution of statement.

It takes an expression in parenthesis and a statement or block of statements. If the expression is true, the statement or block of statements gets executed otherwise these statements are skipped.

Syntax :

```
if (condition)
{
  True statement/s;
}
Statements;
```



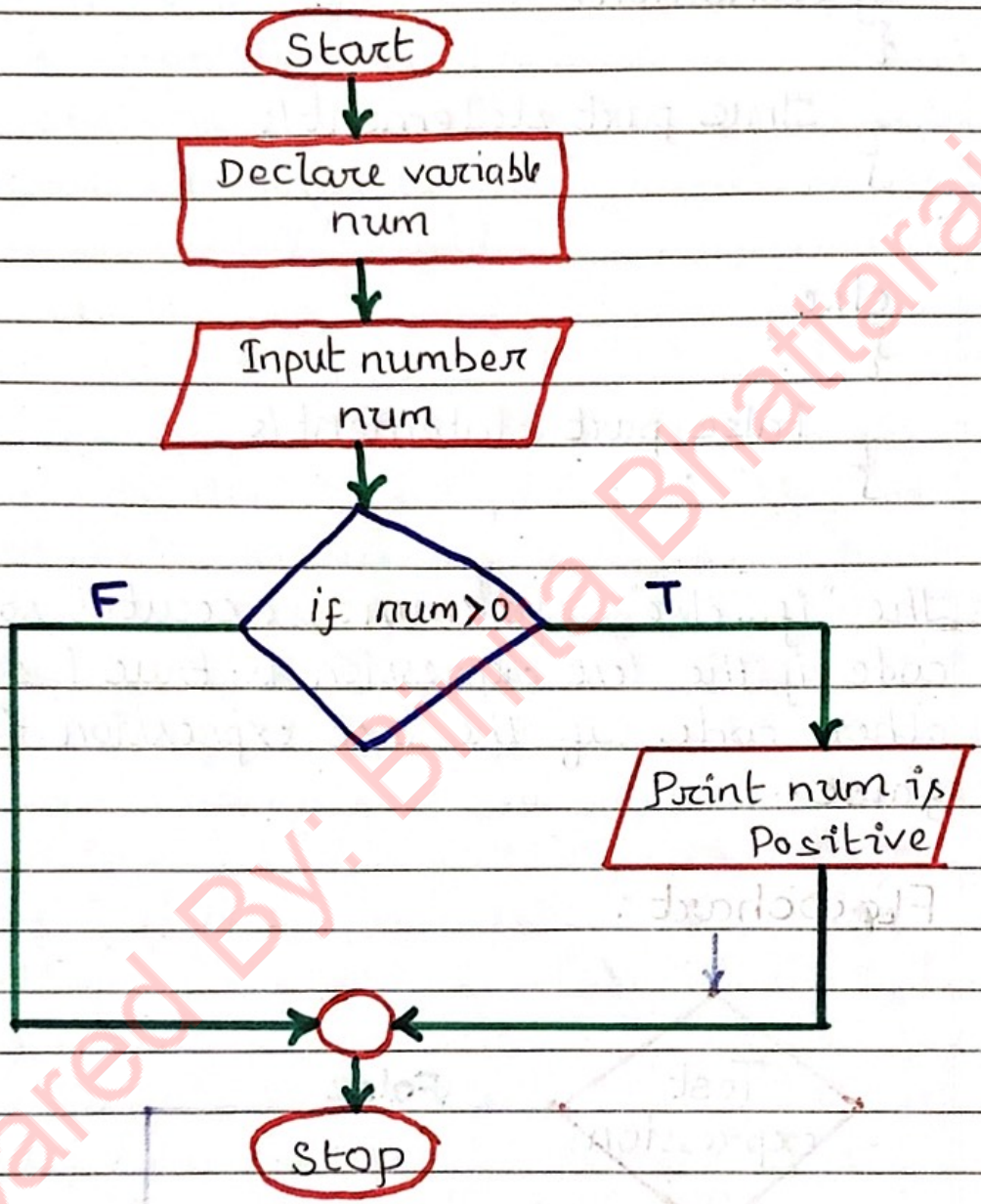
Flowchart of 'if' statement

Program to find given number is +ve:

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

3   void main()
4   {
5       int num;
6       printf("Enter a number");
7       scanf("%d", &num);
8       if (num > 0)
9       {
10          printf("%d is positive," num);
11      }

12      getch();

13  }
```

Here, condition is mentioned at line 8. It results with either TRUE or FALSE. If condition is True, the body within 'if', i.e. line 10 will execute, if it's False, the body is skipped, & program control reaches line 12, i.e. just below the body of 'if'.

Flowchart of above program:



(ii) if else statement:

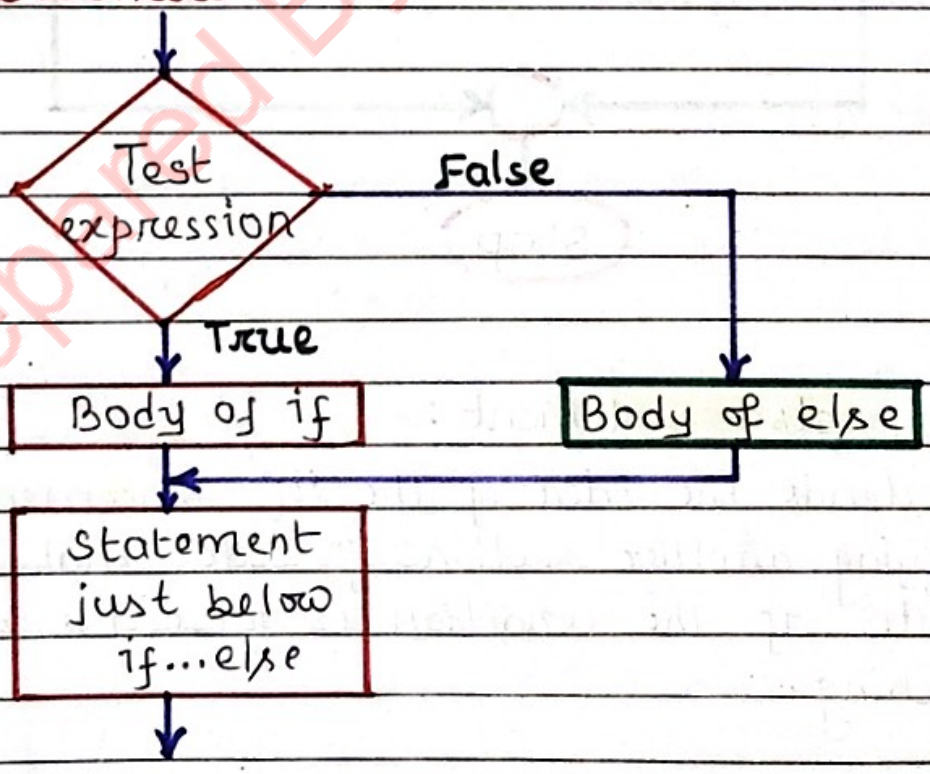
It extends the idea of the 'if' statement by specifying another section of code that should be executed if the condition is false, i.e. conditional branching.

Syntax:

```
if (condition)
{
    True part statement/s
}

else
{
    False part statement/s
}
```

The "if...else" statement executes some code if the test expression is true & some other code if the test expression is false.

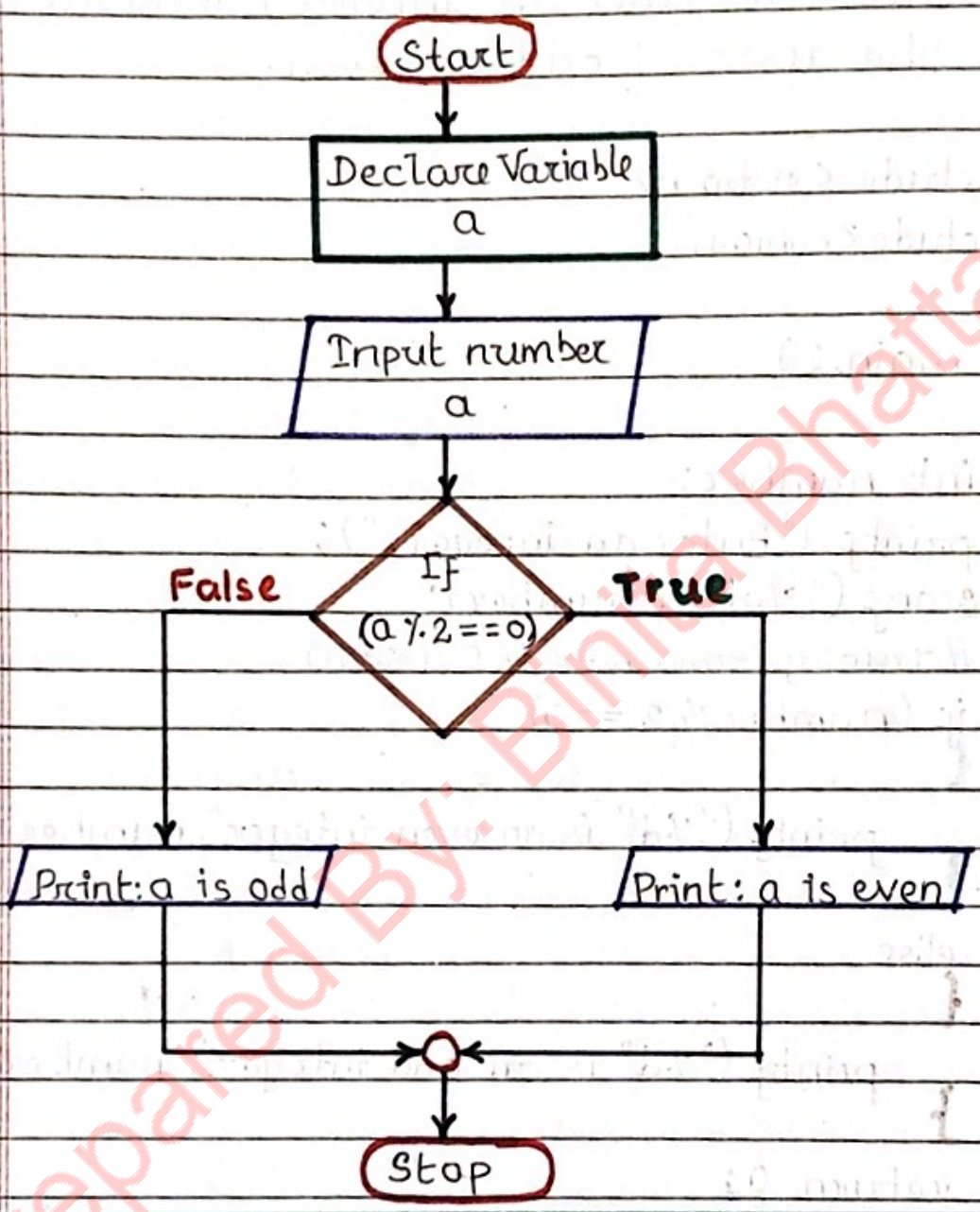
Flowchart:



Example:
To check whether an integer numbered entered by the user is odd or even.

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

int main()
{
    int number;
    printf("Enter an integer:");
    scanf("%d", &number);
    // True if remainder is 0 (even)
    if (number % 2 == 0)
    {
        printf("%d is an even integer", number);
    }
    else
    {
        printf("%d is an odd integer", number);
    }
    return 0;
}
```

Flowchart of above program:



Sample program to find given number is +ve or -ve.

Sample program to check whether a person can vote or not (age >18)

Sample program to check whether a person is married or not (based on input character)

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

void main()
{
    char ch;
    printf("Enter M for married & U for unmarried");
    scanf("%c", &ch);

    if (ch=='M' || ch=='m')
    {
        printf("Person is married");
    }

    else
    {
        printf("Person is unmarried");
    }

    getch();
}
```

(iii) Nested if

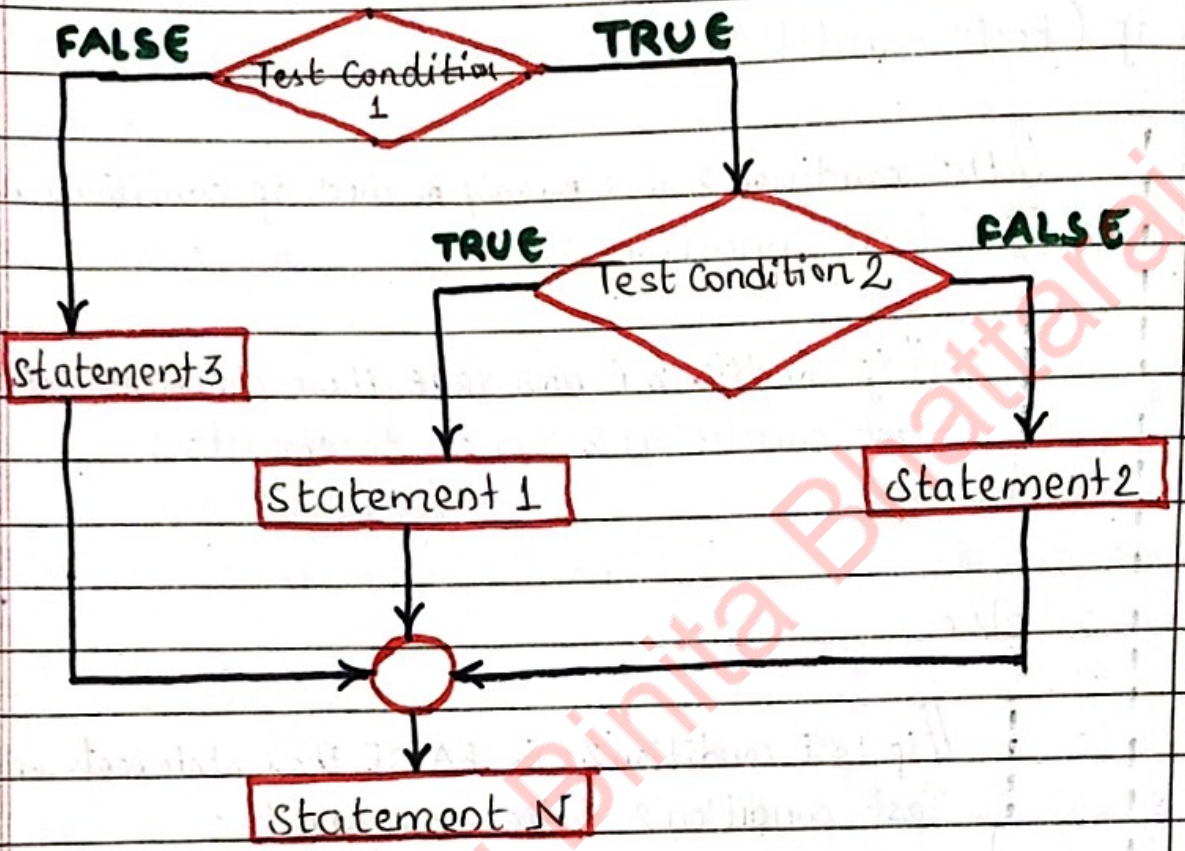
It means placing 'if' statement inside another 'if' statement. It is helpful in C if we want to check condition inside a condition.
'If...else' statement prints different statements based on the expression result (TRUE, FALSE). Sometimes we have to check even further when the condition is TRUE. In these situations we can use these 'nested if' statements.

For example, let's say, every person is eligible (योग्य) to work **if** he/she is 18 years old or above, **else** not eligible. If this much only is the scenario, then **if...else** is sufficient.
But, just think of the case that, being 18+ is not sufficient to get the job, & their academic qualification, experience &/or any other specific company requirements are also required to be fulfilled. This is the scenario in which, first check is for the age to be 18+. If this is satisfied, then further testing are to be performed (i.e. 'if' within 'if').

Syntax:

```
if (test condition 1)
{
    // the condition 2 will be only be check if condition1 is TRUE.
    if ( test condition 2)
    {
        // if condition2 is now TRUE, these statements will execute
        Test condition 2 True statements;
    }

    else
    {
        // if test condition2 is FALSE, these statements will execute
        Test condition 2 False statements;
    }
}
else
{
    // if condition1 is FALSE, execute these statements
    Test condition 1 False statements.
}
```

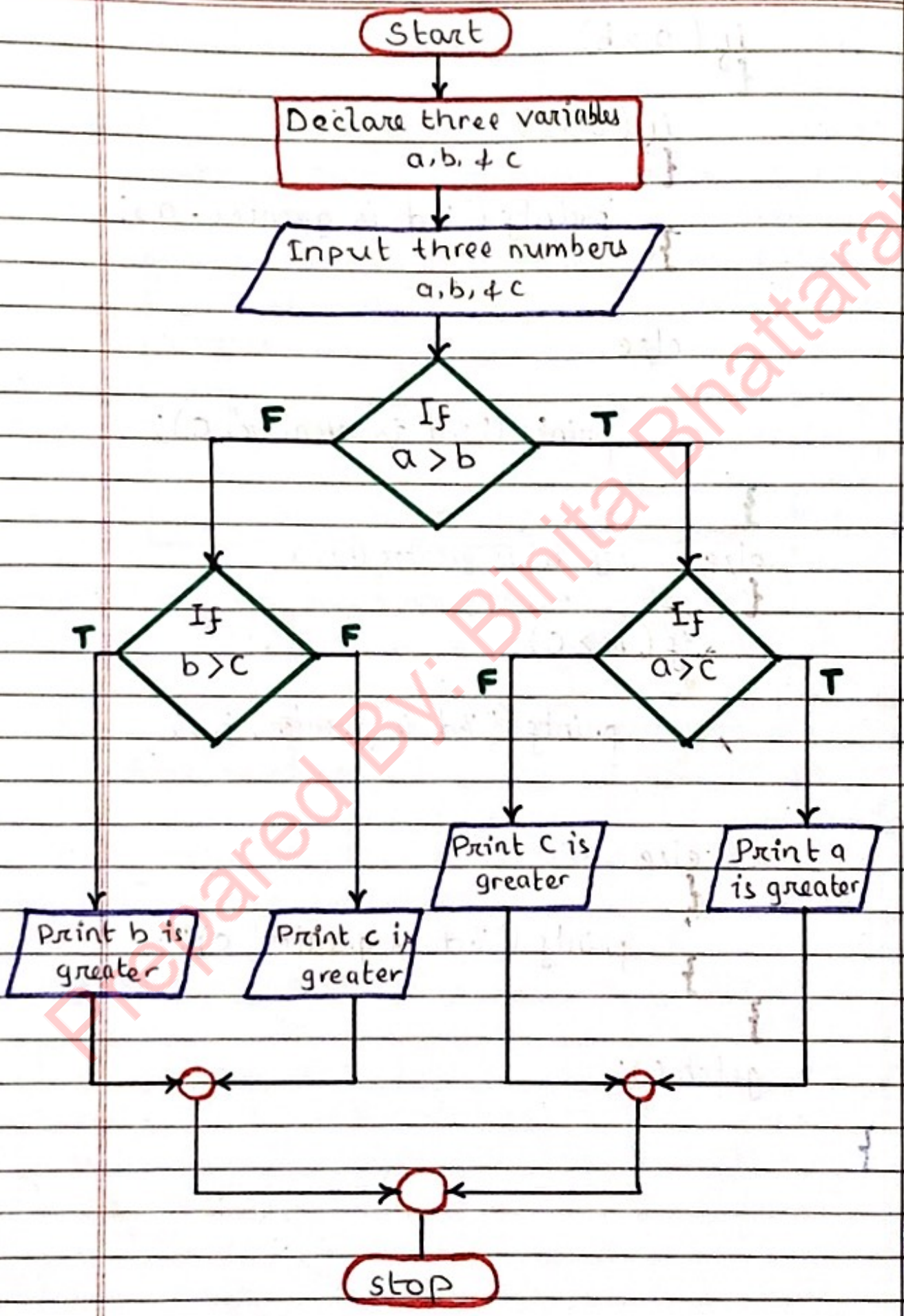
Flowchart:



★ Sample program to find maximum of three numbers.

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

void main()
{
    int a,b,c;
    printf("enter three number");
    scanf("%d%d%d",&a,&b,&c);
```

```
if (a>b)
{
    if (a>c)
    {
        printf ("%d is greater", a);
    }

    else
    {
        printf ("%d is greater", c);
    }
}
else    // if b is greater than a
{
    if (b>c)
    {
        printf ("%d is greater", b);
    }

    else
    {
        printf ("%d is greater", c);
    }
}
getch();
}
```



(iv) Else if Ladder :

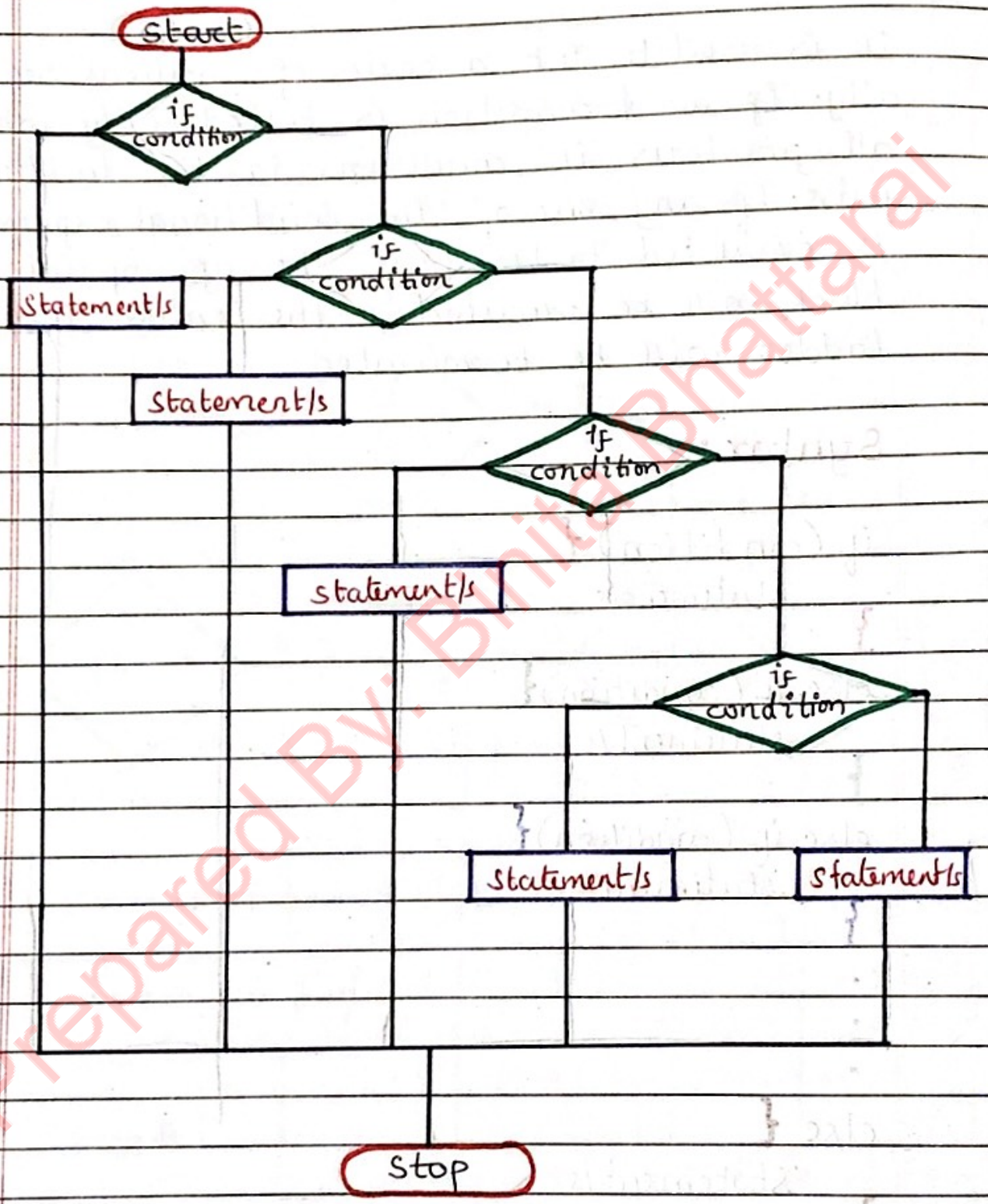
It is used to test a series of conditions sequentially. If a condition is tested only when all previous 'if' conditions in the ladder are false. If any one of the conditional expressions is evaluated to be true, the appropriate code block will be executed & the entire if-else ladder will be terminated.

Syntax :

```
if (condition) {
    statement/s
}
else if (condition){
    statement/s
}
else if (condition){
    statement/s
}
.
.
.
else {
    statement/s
}
```

Date
Page

Flowchart:



Example: To rank obtained marks.

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

void main {
    int a;
        printf ("Enter your mark");
        scanf ("%d",&a);

        if (a>=80){
            printf ("Excellent");
        }

        elseif (a>=70){
            printf ("Very good");
        }

        elseif (a>=60){
            printf ("Good");
        }

        elseif (a>=45){
            printf ("satisfactory");
        }

        else{
            printf ("Fail");
        }
getch();
}
```

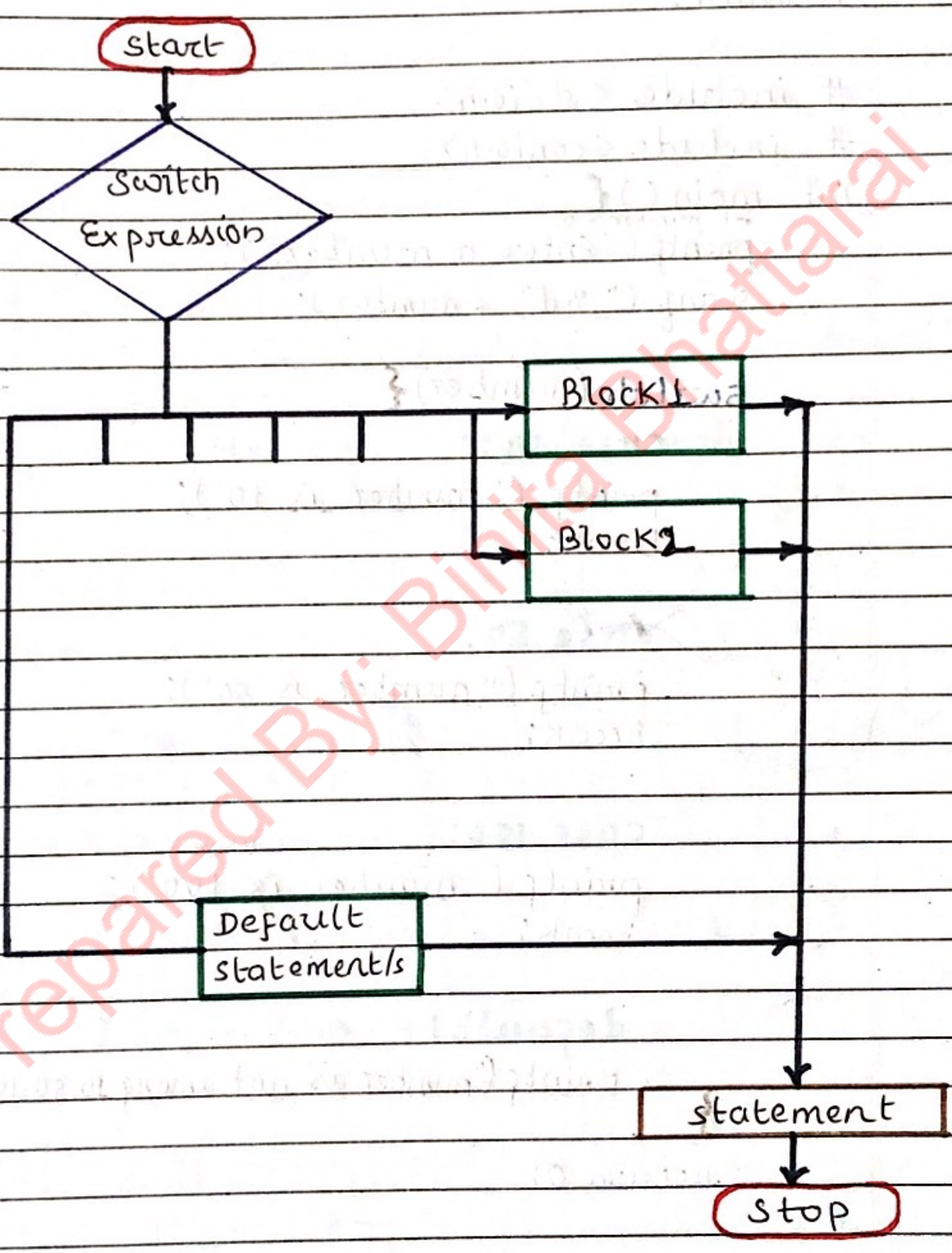
v) Switch (case) statement:

When one of the many alternatives is to be selected, we can design a program using if statements to control the selection. However, the complexity of such a program increases dramatically when the number of alternatives increases.
C has built-in multiway decision statement known as **switch**. It successfully tests the value of an expression against a list of case values (integer or character constants). When a match is found, the statement associated with that case is executed.

Syntax:

```
switch (expression)
{
   Case constant-1
          statement/s
          break;
   Case constant-2
          statement/s
          break;
   Default: statement/s
}
```

Flowchart:



Note: Default statement/s are the codes that gets executed if none of the cases is matched.

* Sample program to identify entered number:

```
# include <stdio.h>
# include <conio.h>
int main(){
    int number = 0;
    printf("Enter a number:");
    scanf("%d", &number);

    switch (number){
        case 10:
        printf("number is 10");
        break;

        case 50:
        printf("number is 50");
        break;

        case 100:
        printf("number is 100");
        break;

        default:
        printf("number is not among 10,50,100");
    }
    return 0;
}
```

⇒ end of branching/selection

3.) Looping

Loops in programming are used to repeat a block of code until the specified condition is met. A loop statement allows programmers to execute a statement or group of statements multiple times without writing the code repeatedly. The condition (test) may be :

either, to determine whether the loop has repeated the specified no. of times

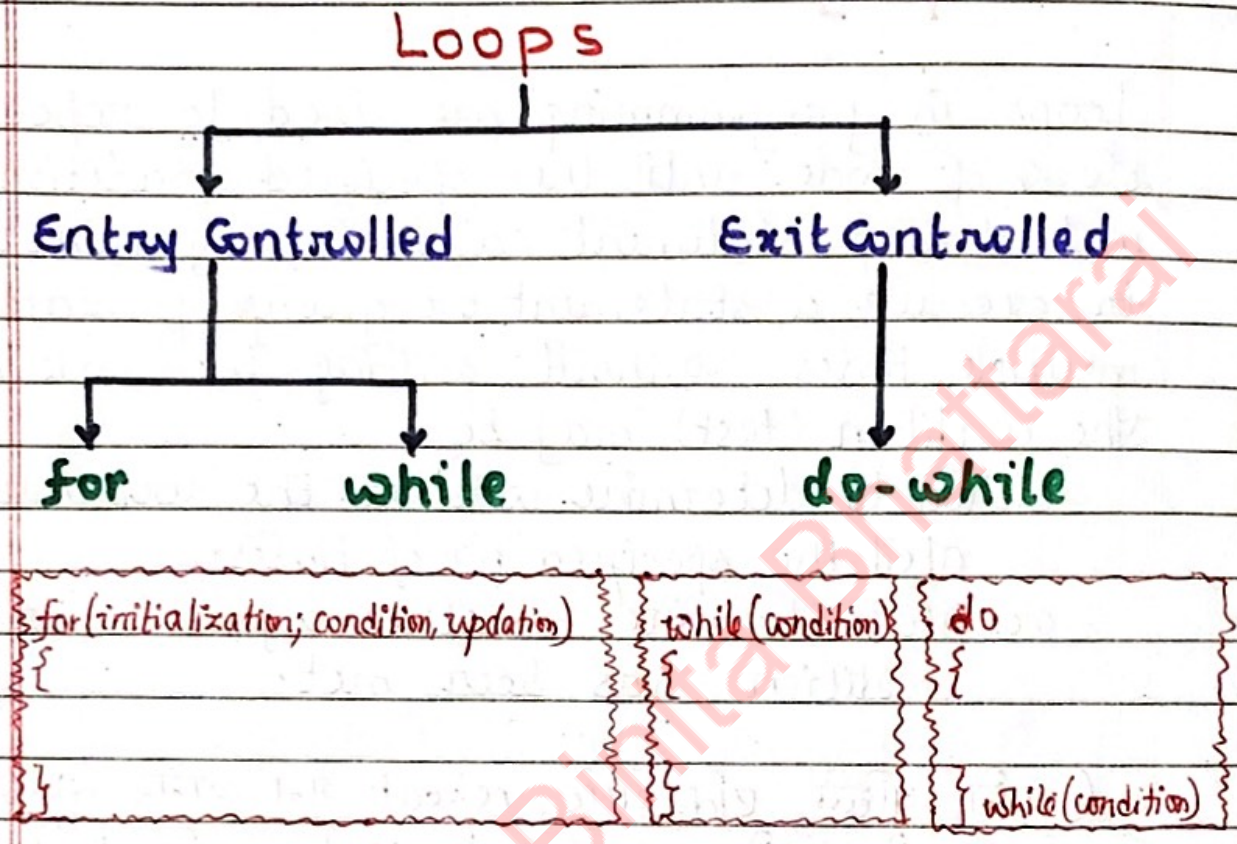
or, to determine whether a particular condition has been met.

(यानि, पहिलें एति चोटि repeat गर्ने भनेर तोकेको र त्यति चोटि वा चल्यो कि चलेन भनि test गर्ने अथवा,
कुनै condition fulfil नभए सम्म चलाउने, repeat गराइरहने)

There are mainly two categories of loops in C:

1) **Entry controlled loops**: Here, the test condition is checked before entering the main body of the loop.
e.g. 'for' & 'while' loop are examples.

2) **Exit controlled loops**: Here, the test condition is evaluated at the end of the loop body. The loop body will execute at least once, irrespective of whether the condition is True or False. e.g: 'do-while' loop.



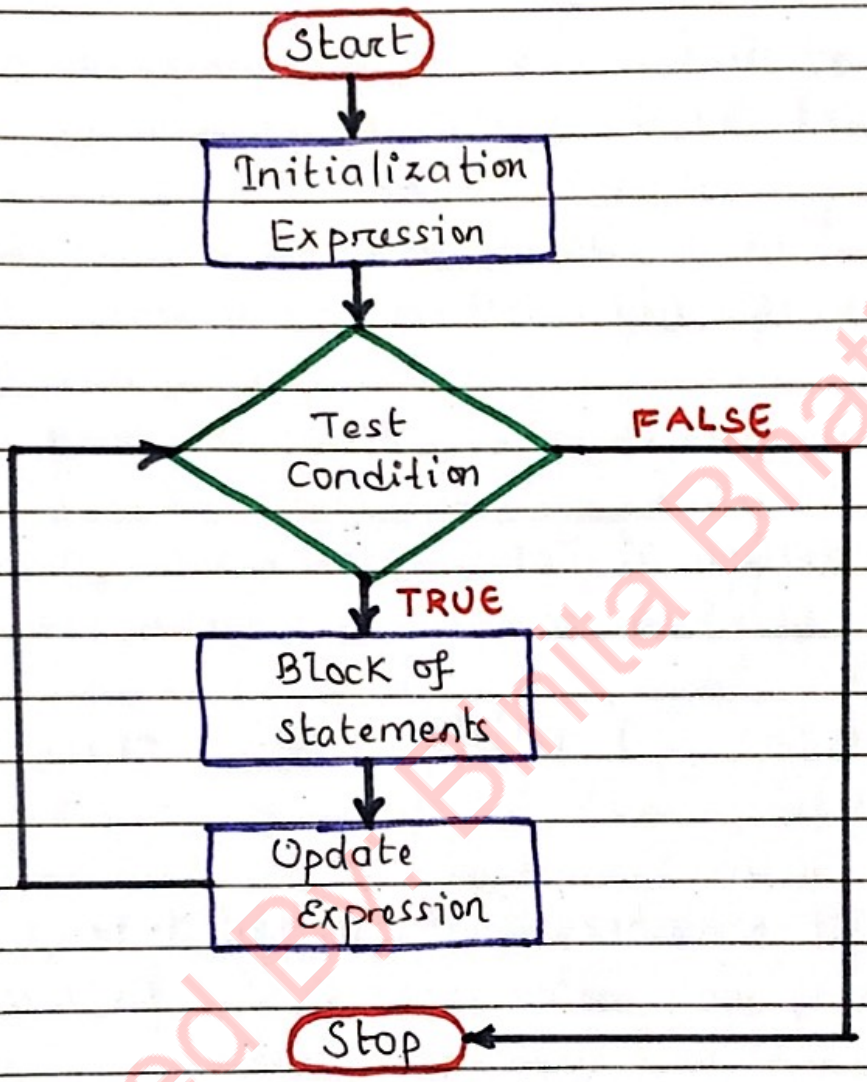
1.) for loop:

It is repetition control structure that allows programmers to write a loop that will be executed a specific number of times.

Syntax:

```
for(initialize expression; test expression; update expression)
{
    body of 'for' loop
}
```

Flowchart:



Sample program:

```
int main(){
    int i;
    for(i=0; i<10; i++){
        printf("Hello World\n");
    }
    return 0;
}
```

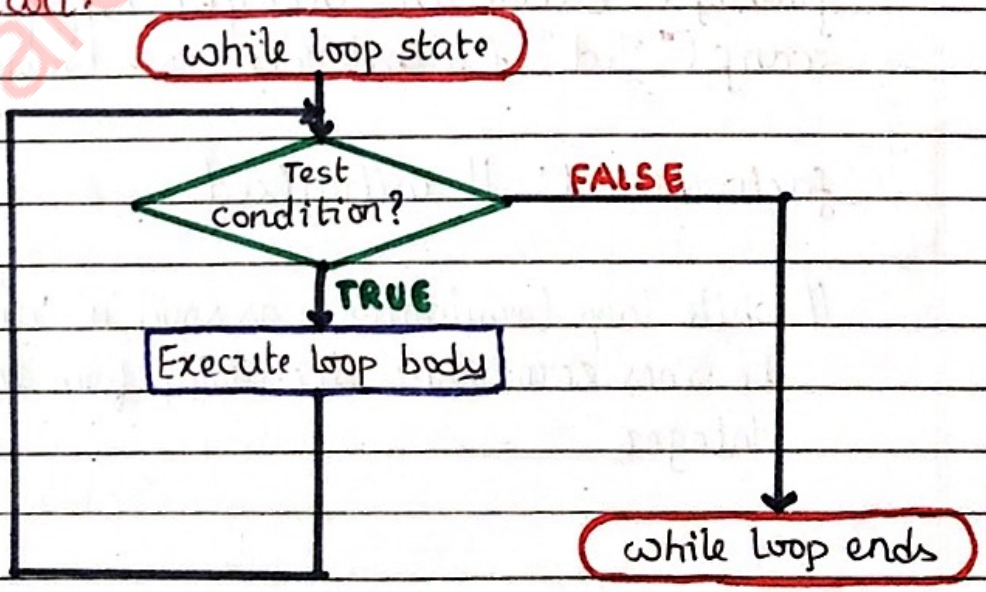
2) while loop

In 'for' loop, the number of iterations was previously known to us but in the 'while' loop, the execution is terminated on the basis of the test condition. If the test condition will become false, then it will break from the 'while' loop, else body will be executed.

Syntax:

```
counter variable (within 'while') declaration
counter variable initialization

while (test condition) {
    body of the while loop (statement/s)

    update expression (counter variable increment/decrement)
}
```

Flowchart:



Sample program:

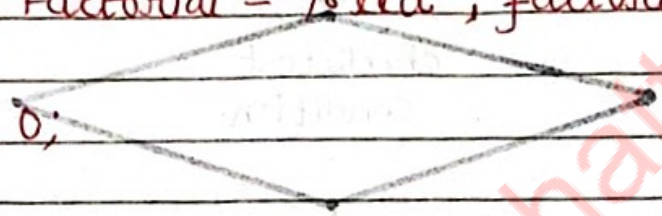
```
int main(){
    int i=0;  // counter variable initialization

    while(i<10){
        printf("Hello World\n");

        i++;
    }

    return 0;
}
```

Sample program to find factorial

```
int main(){
    int number; // whose fact to be found
    long long factorial; // to store the answer

    printf(" Enter an integer ");
    scanf("%d", &number);

    factorial=1; // initialized

    // while loop terminates as soon as 'number' becomes zero while decreasing from any +ve integer
```

asking user to provide a number

```
while (number > 0){
    factorial *= number; // factorial = factorial * number

    --number; // decreasing 'number' by 1
}
printf ("Factorial = %lld", factorial);

return 0;
}
```



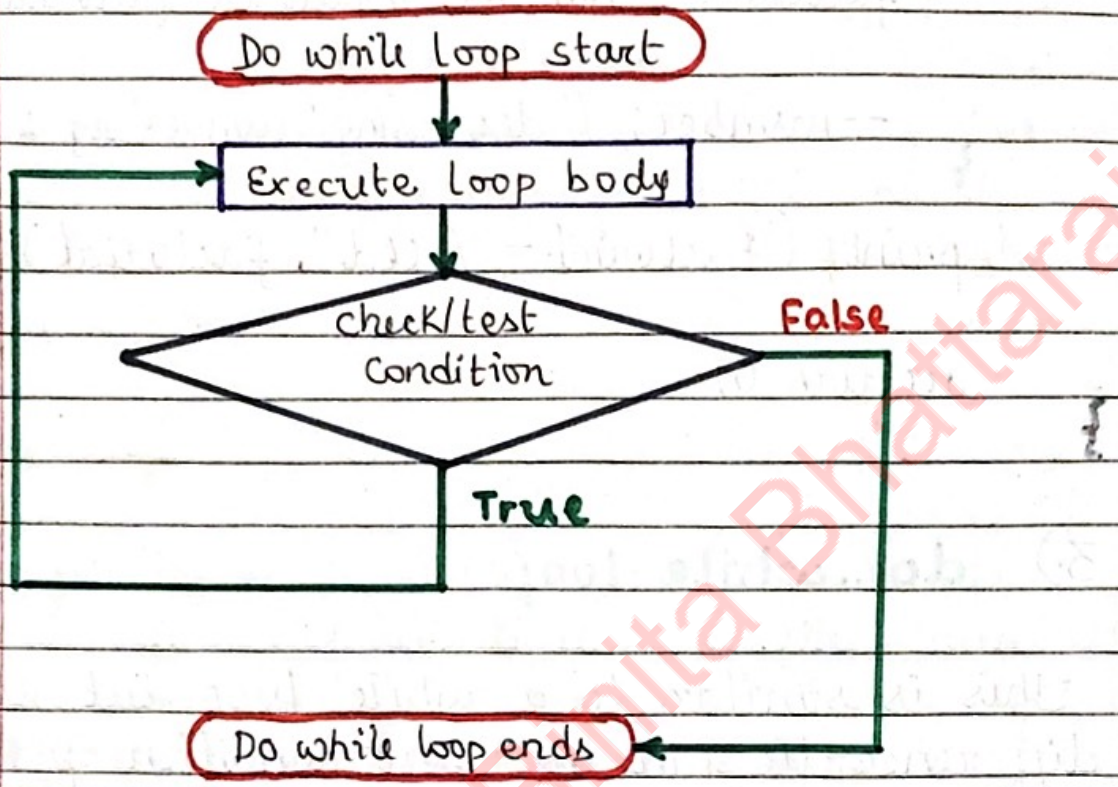
3.) do...while loop

This is similar to a 'while' loop but the only difference lies in the test condition of the do-while, which is tested at the end of the body (whereas, in 'while' loop, test condition is tested at first as soon as program execution enters 'while' statement & then only 'body' gets executed).
In the do-while loop, the loop body will execute at least once irrespective of the test condition.

Syntax:

```
initialization expression;
do {
    // body of do-while loop

    update expression;
}
while (test condition);
```

Flowchart:



Sample program

```
int i=1;
int main(){
do do{
        printf("Hello World\n");
        i++;
```

```
int main(){
    int i=5;

    do{
        printf("Hello World\n");

        i++;
    }
    while(i<1);

    return 0;
}
```

Note: 1st time when test condition is met, it is already false, yet, this program will be executed once.

Sample program to add numbers until user enters zero.

```
int main(){
    double number, sum=0;

    do{
        printf("Enter a number:");
        scanf("%lf", &number);
        sum += number;
    }
    while(number != 0.0);

    printf("Sum = %.2lf", sum);
    return 0;
}
```

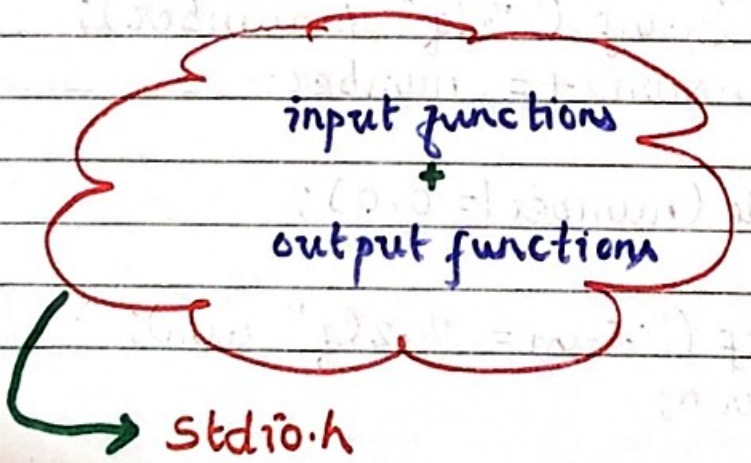
Input / Output instruction:

Input means to provide the program with some data to be used in it and **Output** means to display data on the screen or write the data to a printer or a file.

The C programming language provides standard library functions to read any given input & display output on the console. While dealing with input-output operations in C, we use the following two streams (files):

* Standard input (stdin)
* Standard output (stdout)

Standard input or **stdin** is used for taking input and **Standard output** or **stdout** is used for giving output. The functions used for input & output are present in the **stdio.h** header file.



Hence, to use those functions, we need to include the **stdio.h** header file in our program, as shown below

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

C language offers several built-in functions for performing input/output operations. Following are the functions used for standard input & output:

1. printf() function- Take Show Output
2. scanf() function- Take Input
3. getchar() & putchar() function
4. gets() & puts() function
5. getch() & getche() function

Note:
C language मा, output device हरू जस्तै monitor, printer etc लाई पनि file सरह नै व्यवहार (treat) गरिन्छ। त्यसैले, C मा, कुनै output लाई file मा राख्ने (write) जुन process छ, त्यहि process नै अपनाइन्छ जब कुनै o/p लाई o/p device मा ल्याउन (देखाउन) पर्छ।

Note:
1.) printf() & 2.) scanf() belongs to Formatted input/o/p & remaining to Unformatted.

1.) The printf() function:

This function is defined in the stdio.h header file & is used to show output on the console (standard output). printf() is formatted output. This function is used to print a simple text sentence or value of any variable which can be of int, char, float or any other datatype.

printf() Code Example:

a) Print a Sentence:

```
# include <stdio.h>

int main (){
    printf("I am studying C programming");
    return 0;
}
```

b) Print an Integer Value:

We can use the printf() function to print integer value coming from a variable using the %d format specifier.

```
#include <stdio.h>

int main(){
    int x = 10;
    printf("Value of x is: %d", x);

    return 0;
}
```

Note: **%d** and **%i** are used for integer value

(c) Print a Character value

The **%c** format specifier is used to print character variable value using the printf() function.

```
#include <stdio.h>

int main(){
    char gender = 'M';
    printf("John's gender is : %c", gender);

    return 0;
}
```

d) Print a Float & a Double value:

* For 'float' value, we use %f format specifier.
* For 'double' value, we use %lf.
⇒ Also note the use of \n for new line.
⇒ Also note

```
#include <stdio.h>

int main(){
    float num1 = 15.50;
    double num2 = 15556522.097868;

    printf("Value of num1 is: %f \n", num1);
    printf("Value of num2 is: %lf", num2);

    return 0;
}
```

Output:

```
Value of num1 is: 15.500000
Value of num2 is: 15556522.097868
```

e) Print multiple outputs:

We can use single printf() function to display values of multiple variables

```c
#include <stdio.h>
int main(){
    int day = 20;
    int month = 11;
    int year = 2015;

    printf("The date is: %d-%d-%d", day, month, year);

    return 0;
}
```

Output:

The date is: 20-11-2015

Note: We did formatting (%d-%d-%d) while printing values.

(f) Calculations inside printf():

```c
#include <stdio.h>

int main(){
    int a=5, b=6;
    printf("%d", a+b);
    return 0;
}
```

Output:

11

Date |
Page |

2.) The scanf() function:

When we want to take input from the user, we use the **scanf()** function and store the input value into a variable. It is defined in the C stdio.h library.
It can be used to take any datatype input from user, all we have to take care is that the variable in which we store the value should be of the same datatype.

Syntax:

```
scanf ("%x", &variable);
```

where, %x is the format specifier. Using the format specifier, we tell the compiler what type of data is in a variable & and '&' is the address operator which tells the compiler the address of the variable so that the compiler can assign the variable with the value entered by the user.

Examples:

1) **Input Integer value:**

```
# include <stdio.h>
int main(){
    int user_input;
    printf("Please enter a number: ");
    scanf("%d", &user_input);
    printf("You entered: %d", user_input);
    return 0;
}
```

[Similarly, float value → %f

Date |
Page |

2) Input character value:

```
#include <stdio.h>

int main(){
    char gender;
    printf("Please enter your gender (M, F or O): ");
    scanf("%c", &gender);
    printf("Your gender: %c", gender);

    return 0;
}
```

Return value of printf() & scanf():

The printf() function returns the number of characters printed by it, and scanf() function returns the number of characters read by it.

```
int i = printf("studytonight");
printf("value of i is: %d", i);
```

Output:

```
studytonightValue of i is: 12
```

(New syllabus doesn't include Unformatted input/output)

Date |
Page |

Break, continue & goto statements:

These statements are used to alter the normal flow of a program.

Loops perform a set of repetitive task until test expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break & continue statements are used.

break statement

In C programming, 'break' statement is used with conditional 'if' statement. The 'break' is used in terminating the loop immediately after it is encountered. It is also used in 'switch... case' statement.

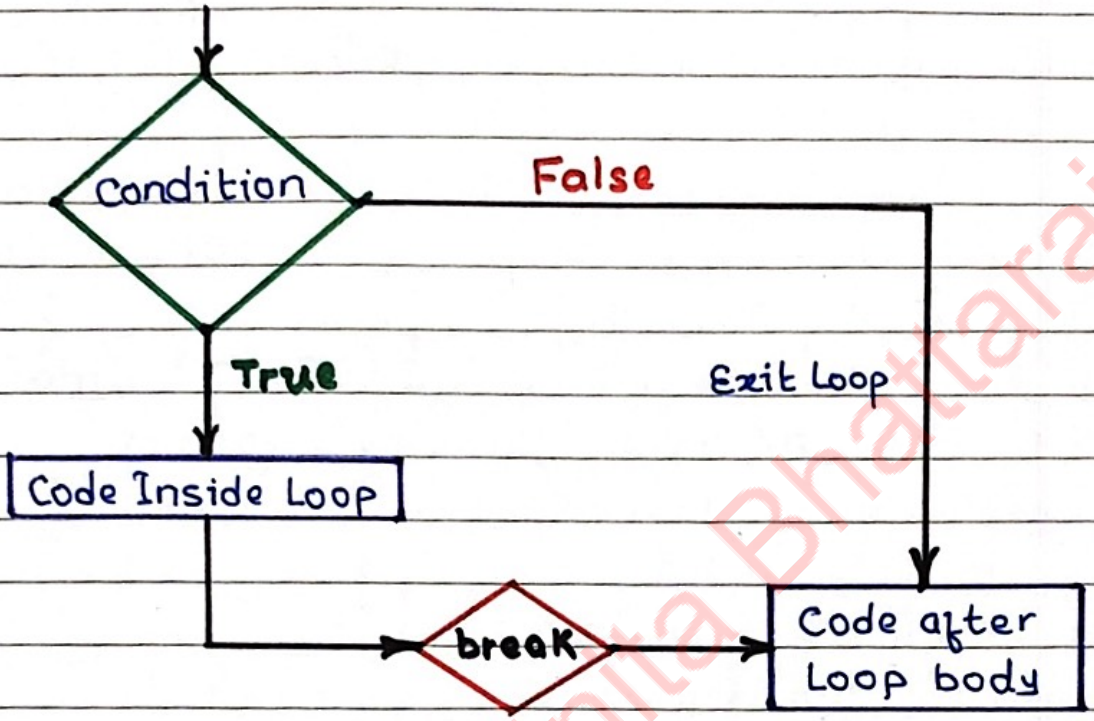
Syntax:

```
break;
```

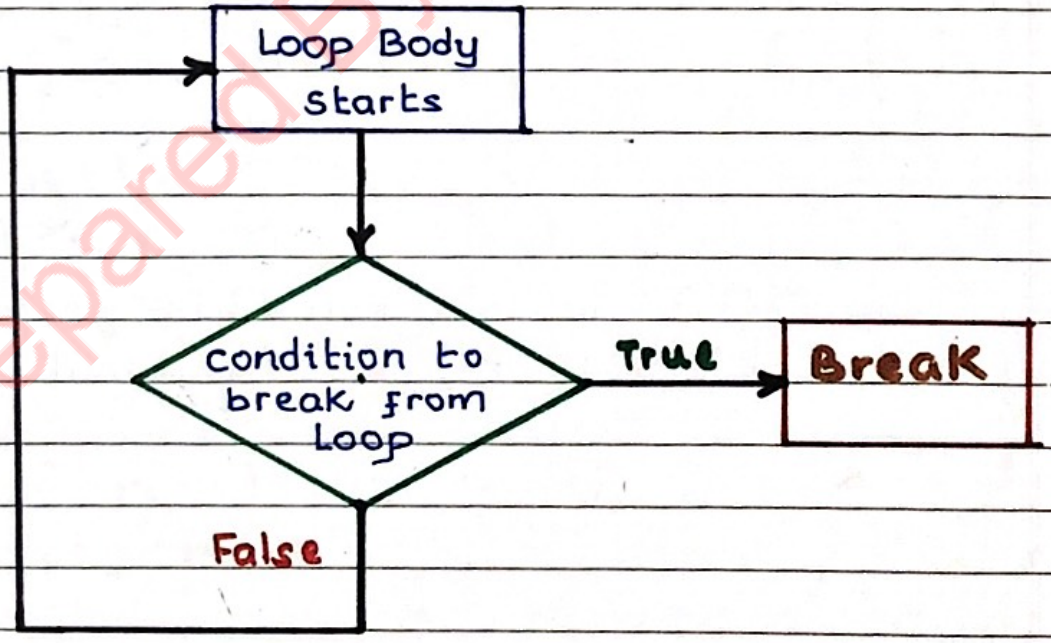
The 'break' statement can be used in terminating loops like 'for', 'while' & 'do...while'.

Date |
Page |

Flow diagram:



Method 1



Method 2

Date |
Page |

Example 1 :

```
#include <stdio.h>

int main(){
    int a=10;

    while (a<20){
        printf ("value of a: %d \n", a);
        a++;

        if(a>15){ // condition for 'break'; if
                  true, gets out of while loop.
            break;
        }
    }
    return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Note: while loop was supposed to iterate until a becomes 19, but when it reached 16, the 'if' condition within while becomes true & 'break' executes due to which program took exit from loop.

Date
Page

Example 2 (with 'for' loop)

```
#include <stdio.h>

void main(){
    int i;

    for(i=0; i<10; i++){
        printf("%d", i);

        if(i==5)
            break;
    }
    printf("came outside of loop i = %d", i);
}
```

Example 3 : (Asking user to take exit)

```
int num, total=0;
while(1){    // always true
    printf("Enter an integer : press 0 to stop entering");
    scanf("%d", &num);

    if(num==0)
        break;

    else{
        total = total + num;
    }
}
printf("Sum of all numbers you entered is : %d", total);
```

Date |
Page |

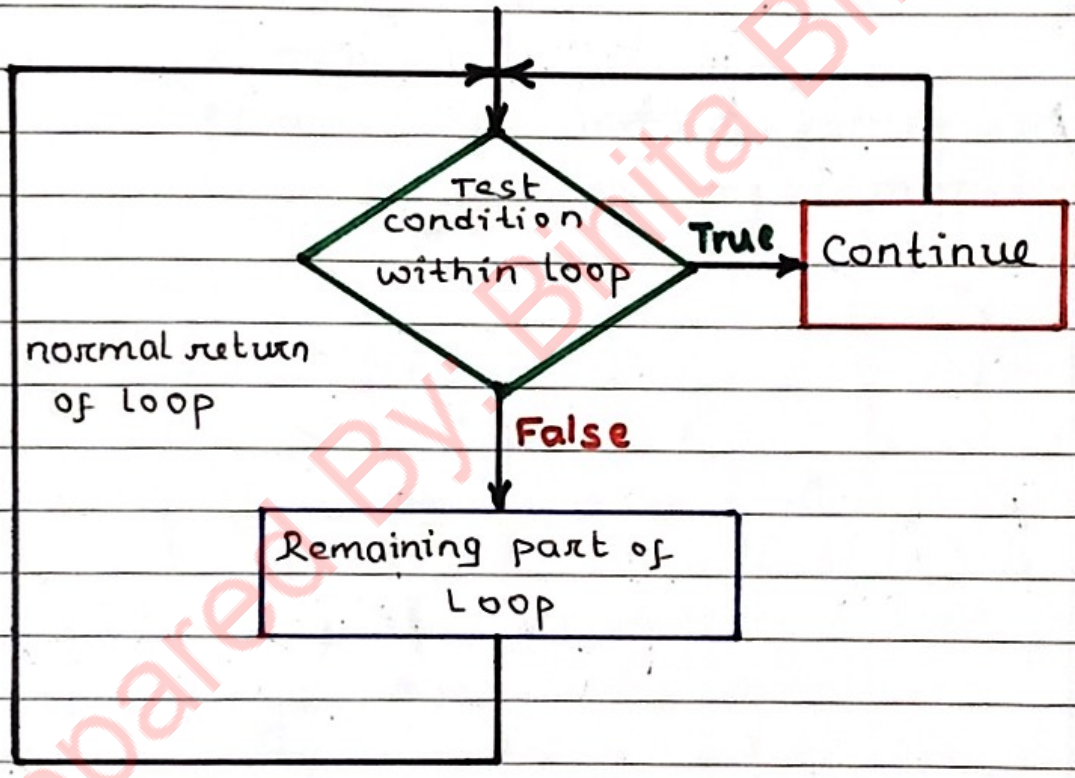
Continue statement:

It is sometimes desirable to skip some statements inside the loop. In such cases, 'continue' statement is used.

Syntax:

```
continue;
```

Flow diagram:



Date |
Page |

Example:

```
include <stdio.h>

void main ( ){
    int i, n=20;

    for(i=1; i<=20; ++i){

        if (i%5 == 0){
            printf ("pass \n");
            continue; // तलको कोड execute नभई सिधै
                      माथि for loop मा गई 'i' increment
                      हुन्छ
        }
        printf ("%d \n", i);
    }
    getch();
}
```

OUTPUT:

```
1
2
3
4
pass
6
7
8
9
pass
11
12
13
14
pass
16
17
18
19
pass
```

Date |
Page |

goto statement

In C programming, 'goto' statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax:

```
goto label;
.....
.....
.....
label:
statement;
```

In this syntax, when 'label' is an identifier. When the control of program reaches to 'goto' statement, the control of the program will jump to the label: and executes the code below it.

Example: (Printing 1 to 10 numbers using goto)

```
# include <stdio.h>
void main ( ){
    int i=1;
    count:        // label defined here

    printf ("%d", i);
    i++;
    if (i<=10){
        goto count;
    }
    getch ( );
}
```

Date |
Page |

Note:

Though goto statement is included in ANSI standard of C, use of goto should be reduced as much as possible in a program.

Reason:

* Though, using goto statement give power to jump to any part of program using goto statement makes the logic of the program complex & tangled.

* In modern programming, goto statement is considered a harmful construct and a bad programming practice.

* The goto statement can be replaced in most of C program with the use of break & continue statements.

* In fact, any program in C can be perfectly written without the use of goto.