UNIT – 5
Control Statements

C Programming (CSC115)
BSc CSIT First Semester (TU)

Prepared by:
Dabbal Singh Mahara
ASCOL (2025)

CONTROL STATEMENTS

- Control structures in C are constructs that dictate the flow of control in a program.
- They allow programmers to specify the order in which instructions are executed, enabling decision-making, repetition, and branching.
- The statements which control or alter the flow of execution of the program are known as control statements.
- In the absence of control statements, the instructions or statements are executed in the same order in which they appear in the program (called sequential constructs).

SEQUENTIAL CONTROL STRUCTURE

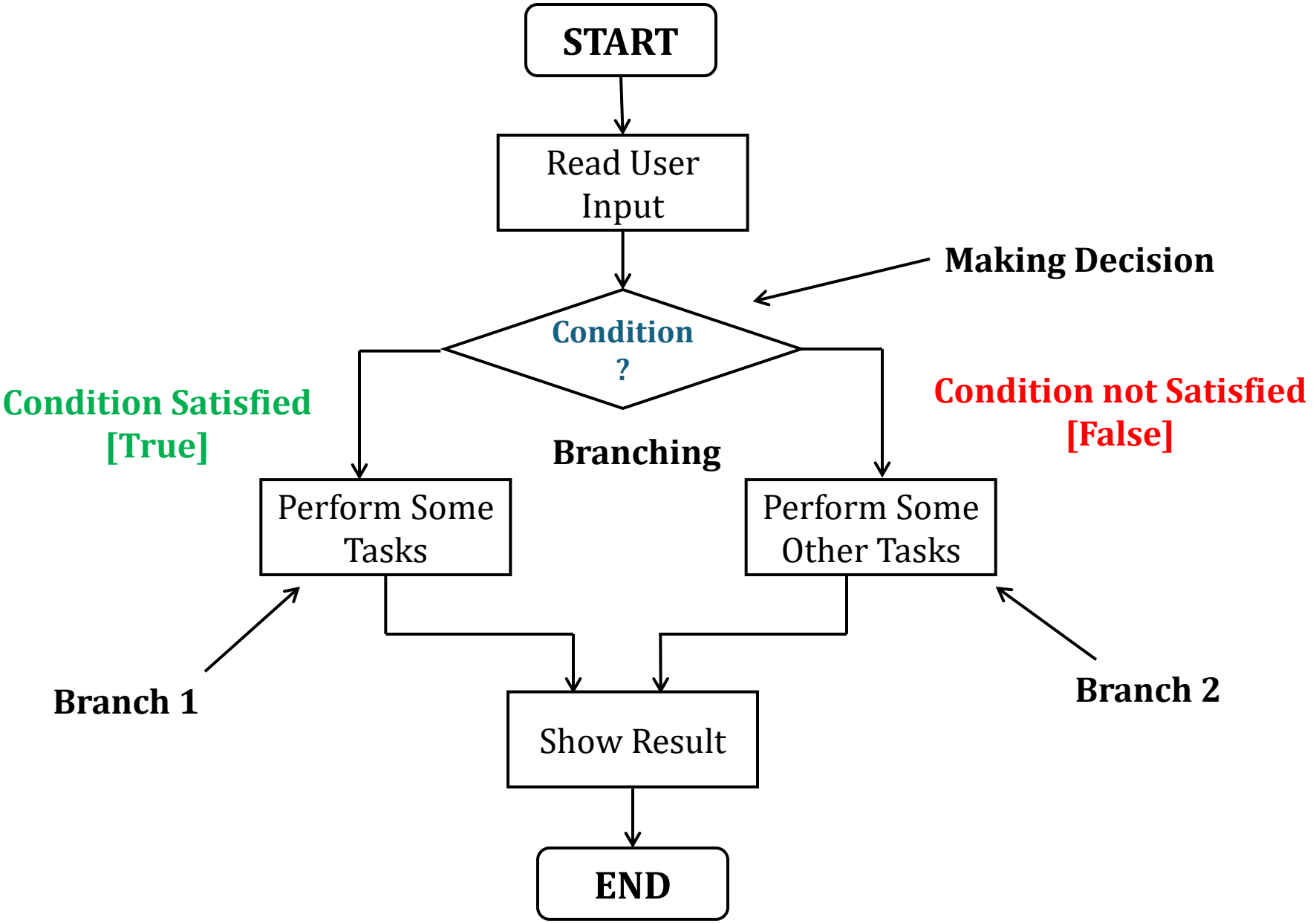
- The default mode of execution in C, where statements are executed one after the other in the order they appear.

Example:

```
int x = 5;
int y = 10;
int sum = x + y;
printf("Sum: %d\n", sum);
```

DECISION-MAKING STATEMENTS

- These structures allow the program to choose between different paths based on a condition.
- Selection statements in C allow programs to make decisions and execute specific blocks of code based on certain conditions.
- These statements enable branching, where the flow of execution deviates based on the outcome of a condition.


START
Read User Input
Making Decision
Condition ?
Condition Satisfied [True]
Condition not Satisfied [False]
Branching
Perform Some Tasks
Perform Some Other Tasks
Branch 1
Branch 2
Show Result
END

DECISION MAKING & BRANCHING IN C

- C language possesses decision making and branching capabilities by supporting the following statements:

1. if Statement
2. if-else Statement
3. if-else if-else Ladder
4. switch Statement

IF STATEMENT

- The if statement evaluates a condition and executes a block of code if the condition is true.

Syntax:

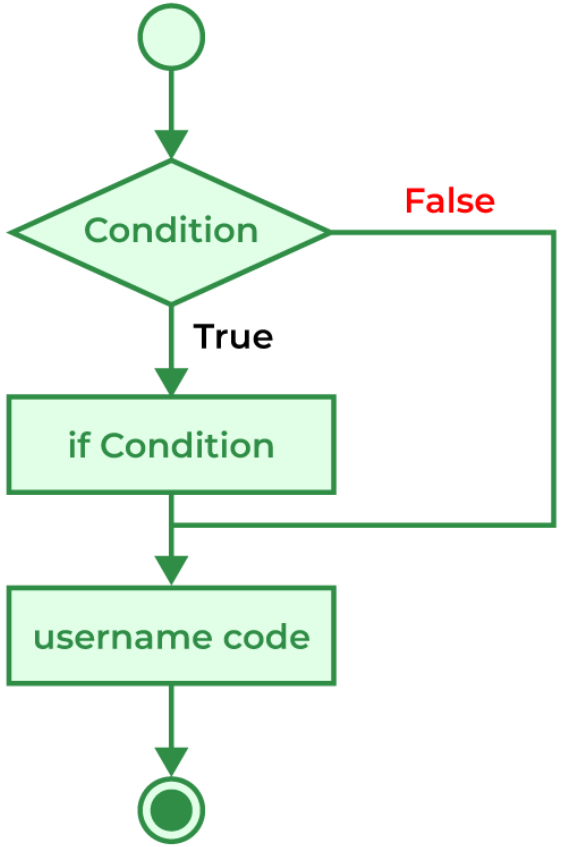
```
if (condition) {

    // Code to execute if condition is
true
}
```

Example:

```
int age = 20;
if (age >= 18) {
    printf("You are eligible to vote.\n");
}
```

FLOWCHART OF IF STATEMENT

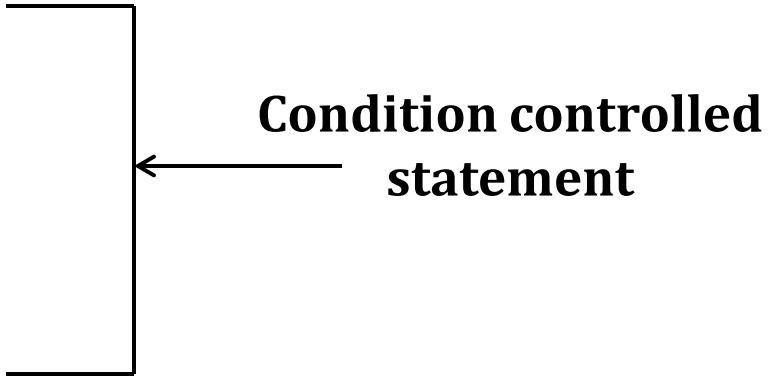


Simple if Statement-Example

- Example with block of statements:

```
if (marks>=90)
{
        marks = marks + bonus_marks;
        grade="A+";
}

printf("The mark achieved: %d" , marks);
```



```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num;
    printf("Enter a number to be tested:");
    scanf("%d", &num);
    if(num < 0)
    {
       printf("\nThe number is negative");
    }
    printf("\nDon't use FACEBOOK while studying.");
    getch();
}
```

IF…ELSE STATEMENT

- The if-else statement provides an alternative action if the condition is false.
- It is used when there are two possible actions- one when a condition is true, and the other when the condition is false.

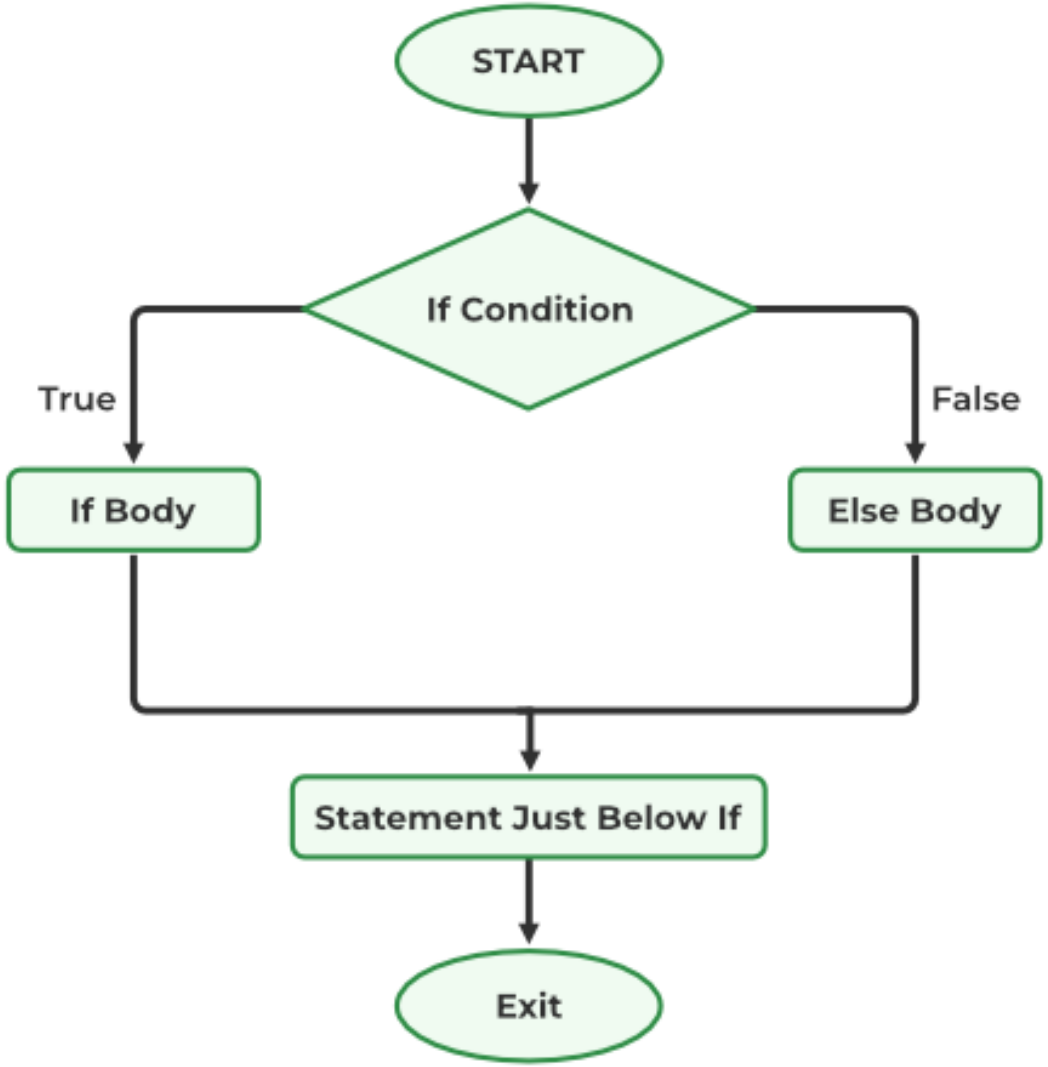
Syntax:

```
if(test expression is True)  {
        true-block statement(s);
}
else  {
        false-block statement(s);
}
statement-x;
```

Example:

```
int num = -5;
if (num > 0) {
    printf("Positive number\n");
} else {
    printf("Non-positive number\n");
}
```

Flowchart of if statement



```
#include <stdio.h>
#include <conio.h>
int main()
{
	int num, rem;
	printf("Enter a number:");
	scanf("%d",&num);
	rem = num%2;
	if(rem == 0)
	{
		printf("\nThe number is even.");
	}
	else
	{
		printf("\nThe number is odd");
	}
	printf("\nDon't use MOBILE while studying.");
	getch();
}
```

EXAMPLE

- Write a C program to read the values of coefficients a, b and c of a quadratic equation and find the real roots of the equation. Use if…else statement and If the roots are imaginary, display the message “roots are imaginary”. Calculate imaginary roots too.

```
#include <stdio.h>
#include <conio.h>
#include <cmath>
int main()
{
	float a, b, c, d, root1, root2, real, img;
	printf("\nEnter values a, b and c of the quadratic equation:");
	scanf("%f %f %f", &a, &b, &c);
	d = b*b-4*a*c;

	if(d<0)
	{
		printf("\nImaginary Roots.");
		d = sqrt(fabs(d));    //compute absolute value of discriminant
		real = -b/(2*a);
		img = d/(2*a);
		printf("\nRoot1 = %.2f +i %.2f", real, img);
		printf("\nRoot2 = %.2f -i %.2f", real, img);
	}
	else
	{
		printf("\nRoots are real.");
		d = sqrt(d);
		root1 = (-b+d)/(2*a);
		root2 = (-b-d)/(2*a);
		printf("\nRoot1 = %.2f \t Root2 = %.2f", root1, root2);
	}
	getch();
	return 0;
}
```

Class Work

1) Write a program to read two integers from keyboard and check whether the first integer is exactly divisible by the second or not.
2) Write a program that asks a number from keyboard and tests whether it is a multiple of 7 or not.

IF … ELSE IF STATEMENT

- The *else if* statement is used when there are more than two possible actions depending upon the outcome of test.
- When one action is taken, no others can be executed or taken.
- In such situation, if…else if…else if…else structure is used.
- The general structure is given in coming slide:

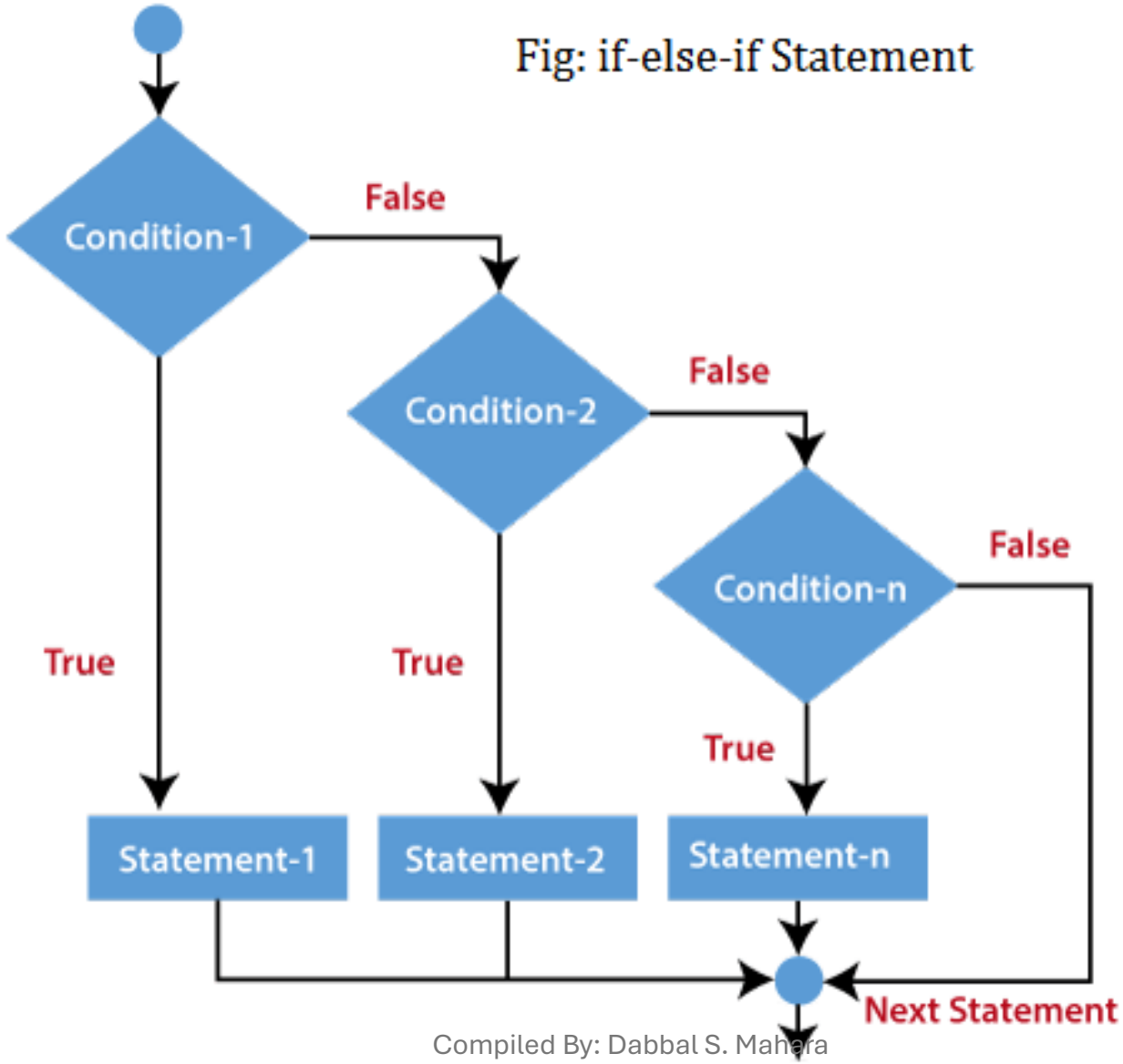
Syntax:

```
if(condition 1)
{
        statement-1;
}
else if(condition 2)
{
        statement-2;
}
… … … … … … … … … …
else if(condition n)
{
        statement-n;
}
else
{
        default-statement;
}
statement-x;
```

If …else if statement…

- The conditions are evaluated from top to bottom until a true condition is found.
- Whenever any One of the conditions becomes True, the statements associated with it are executed and then the control is transferred directly to statement-x; skipping the rest of the structure.
- When all the n conditions become false, then the final else containing the default-statement will be executed.

Fig: if-else-if Statement



```
#include <stdio.h>
#include <conio.h>
int main()
{
    int n1,n2,n3;
    printf("\nEnter 3 numbers:\t");
    scanf("%d %d %d",&n1,&n2,&n3);
    if(n1>n2 && n1>n3)
    {
            printf("\n%d is the greatest number.",n1);
    }
    else if(n2>n1 && n2>n3)
    {
            printf("\n%d is the greatest number.",n2);
    }
    else
    {
            printf("\n%d is the greatest number.",n3);
    }
    getch();
}
```

Exercise

Write an algorithm, draw a flowchart and write a C program using *else if* statement to read the marks of a student in various subjects of First semester, then calculate the percentage obtained by them and output the division. The conditions to be used are:

- % greater than or equal to 80 => Distinction
- % between 70 and 79 => First Division
- % between 60 and 69 => Second Division
- % between 50 and 59 => Third Division
- % less than 50 => Fail

NESTED *IF…ELSE* STATEMENT

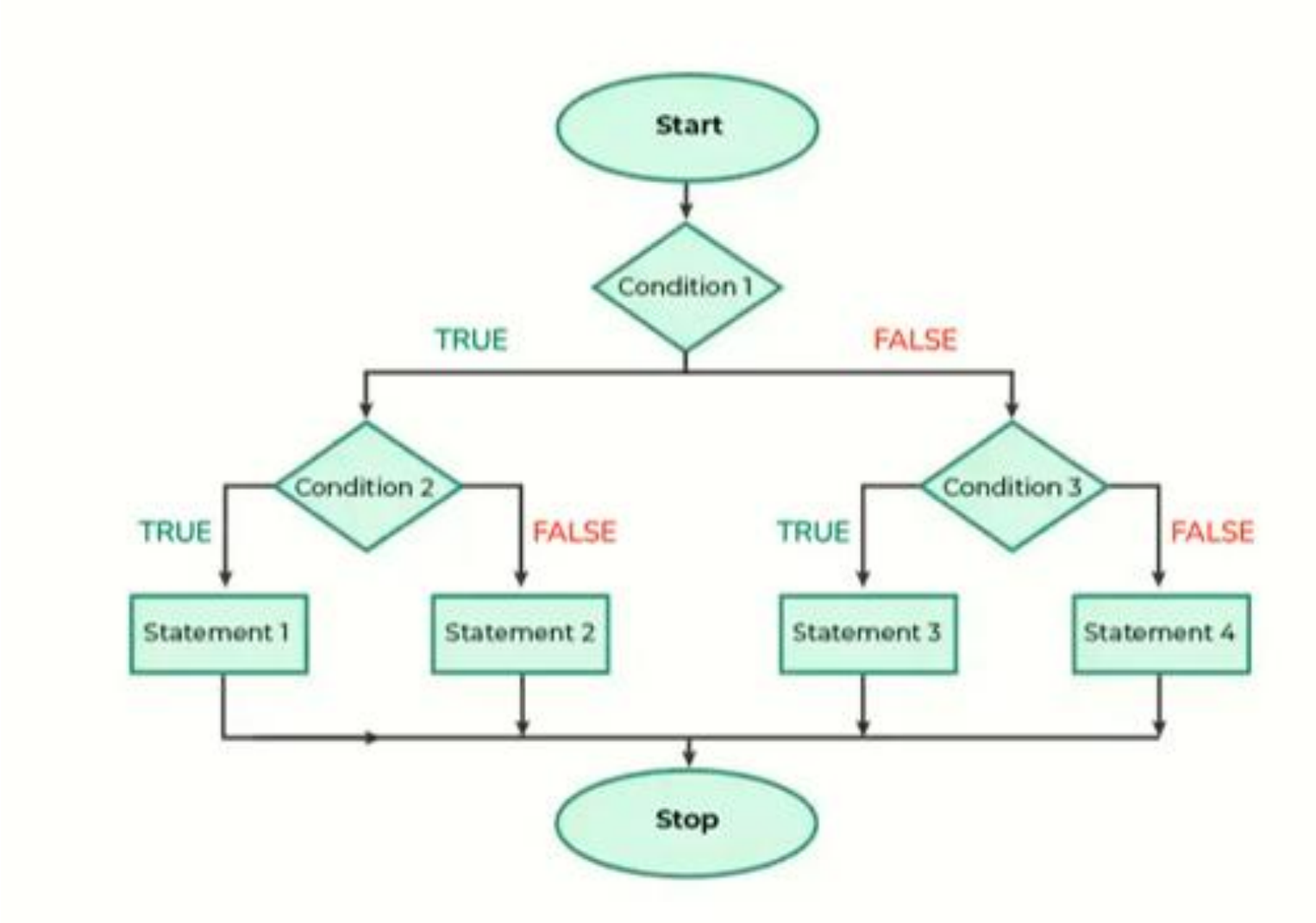
- If an entire *if…else* construct is written under either the body of an *if* statement or the body of an *else* statement, then such type of construct is called nested *if…else* statement.
- In nested form, the condition for *inner if* is evaluated only if the condition for *outer if* is satisfied; otherwise, it is skipped and the *else* part of the *outer if* is executed.
- The general form is given in coming slide:

```
if(condition 1)
        {
                if(condition 2)
                {
                        statement-1;
                }
                else
                {
                        statement-2;
                }
        }
else
        {
                if(condition 3)
                {
                        statement-3;
                }
                else
                {
                        statement-4;
                }
        }
statement-x;
```

- If condition 1 is true, then the second test condition 2 is evaluated and statement-1 or statement-2 is executed, depending on the result of condition-2.
- If condition 1 is false, then the else portion is evaluated, which contains another test condition 3 => so that either statement-3 or statement-4 is executed depending on the result of condition 3.
- Thus, only one of the four statements will be executed will be executed at a time, and after that the control is transferred to the statement-x.

Note: There can be other constructs of nested *if…else* depending on the presence or absence of *if…else* inside the outer *if…else* statements.

Flowchart of nested if statement



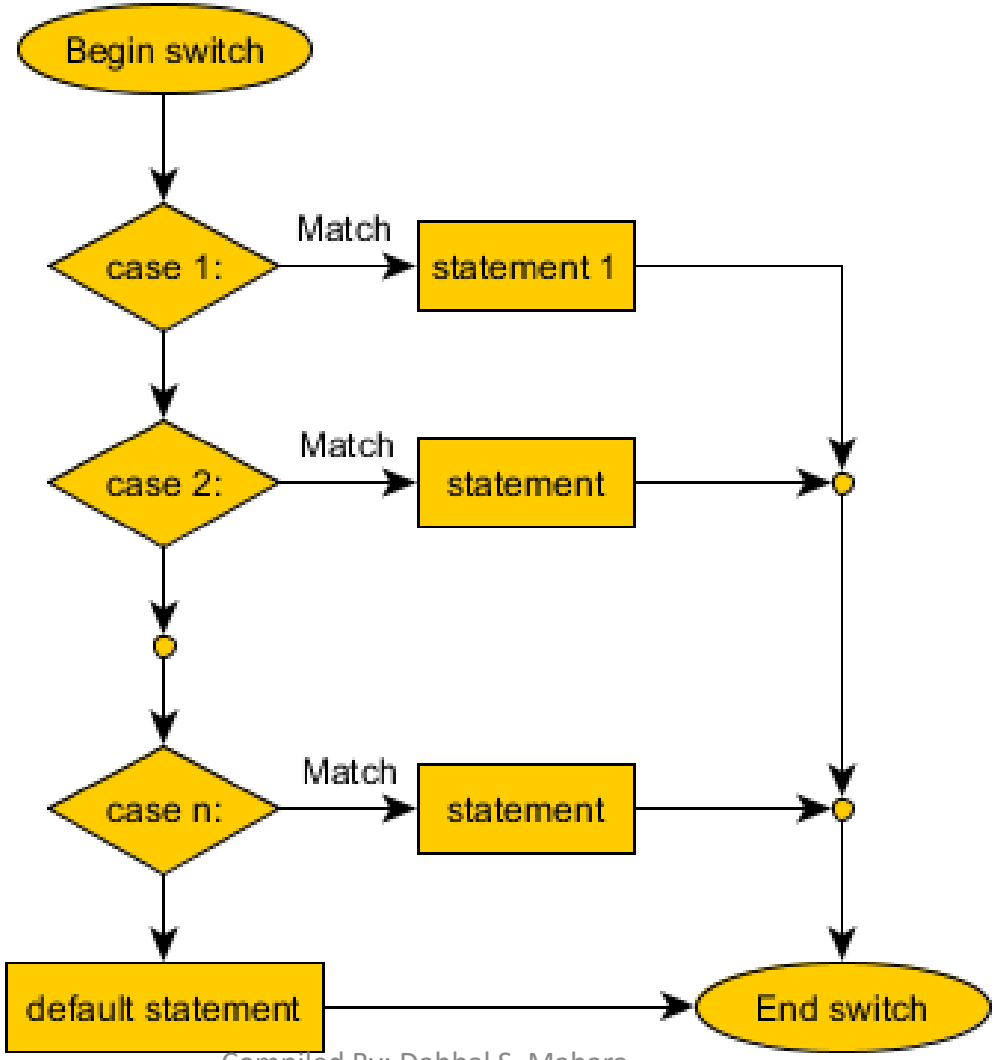
```
#include <stdio.h>
#include <conio.h>
int main()
{
        int n1,n2,n3;
        printf("\nEnter 3 numbers:\t");
        scanf("%d %d %d",&n1,&n2,&n3);
        if(n1>n2)
            {
            if(n1>n3)
                {
                printf("\n%d is the greatest number.",n1);
                }
            else
                {
                printf("\n%d is the greatest number.",n3);
                }
            }
        else
            {
                if(n2>n3)
                {
                printf("\n%d is the greatest number.",n2);
                }
                else
                {
                printf("\n%d is the greatest number.",n3);
                }
            }
        getch();
}
```

SWITCH STATEMENT

- When there are a number of else alternatives, *switch* statement is another way of representing this multi-way selection. (What was one other way???)
- The *switch* statement is useful when a variable is to be compared with different constants, and in case it is equal to a constant, a set of statements are to be executed.
- The constants in the case statement may be either *char* or *int* type only.
- Syntax is provided in coming slide:

```
switch(variable_name)
{
      case   caseConstant1:
                statements;
                break;
     case      caseConstant2:
                 statements;
                   break;
              … … … … … … … … … …
    default:
           statements;
}
```

Switch statement Flowchart



```
int  main()
{
      int choice;
      printf("\nWhich of these websites you visit the most?");
      printf("\nSelect 1 for FACEBOOK, 2 for YAHOO! and 3 for GOOGLE.");
      printf("\n1=>FACEBOOK \n2=>YAHOO! \n3=>GOOGLE\n");
      scanf("%d",&choice);

      switch(choice)
          {
          case 1:
              printf("\nYou use FACEBOOK.");
              break;
          case 2:
              printf("\nYou use YAHOO!.");
              break;
          case 3:
              printf("\nYou use GOOGLE.");
              break;
          default:
              printf("\nYou have entered an invalid option.");
          }
      getch();
      return 0;
}
```

```
/*Program to add, subtract, multiply and divide two complex numbers using
   switch statement*/
#include <stdio.h>
#include <conio.h>
int  main()
{
      int a,b,x,y,real,img;
      char opr;
      printf("\nEnter first complex number of the form (a+ib):");
      scanf("%d+i%d", &a, &b);
      printf("\nEnter second complex number of the form (x+iy):");
      scanf("%d+i%d", &x, &y);
      printf("\nEnter one of the operators among +, -, *, /:\t");
      scanf(" %c",&opr);
      switch(opr)       {
          case '+':
              real = a+x;
              img = b+y;
              printf("\nThe addition is:%d+i%d.", real, img);
              break;
```

```
    case '-':
                real = a-x;
                img = b-y;
                printf("\nThe subtraction is:%d+i%d.", real, img);
                break;
    case '*':
                real = a*x-b*y;
                img = a*y+b*x;
                printf("\nThe multiplication is:%d+i%d", real, img);
                break;
case '/':
                real = (a*x+b*y)/(x*x+y*y);
                img = (b*x-a*y)/(x*x+y*y);
                printf("\nThe division is:%d+i%d", real, img);
                break;
default:
                printf("\nInvalid Operator.");
}
getch();
return 0;
}
```

GOTO STATEMENT

- The *goto* statement is used to alter the normal sequence of program execution by unconditionally transferring control to some other part of the program.
- The *goto* statement transfers the control to a labeled statement somewhere in the current function using syntax:

```
goto label;
```

- Here, label is an identifier used to label the target statement to which the control would be transferred.
- The target statement must be labeled using syntax:

```
label: statements;
```

/* Program to ask two numbers and display message “Either number is negative”, if one of the numbers is negative; otherwise display message “Both numbers are positive” */

```
int main()
{
    int i, num1, num2;
    printf("Enter first no:");
    scanf("%d", &num1);
        if(num1<0)
            {
            goto negative;
            }
    printf("\nEnter second number:");
    scanf("%d", &num2);
        if(num2<0)
            {
            goto negative;
            }
    printf("\nBoth numbers are positive");
    getch();
    return;
    negative: printf("\nEither number is negative");
    getch();
}
```

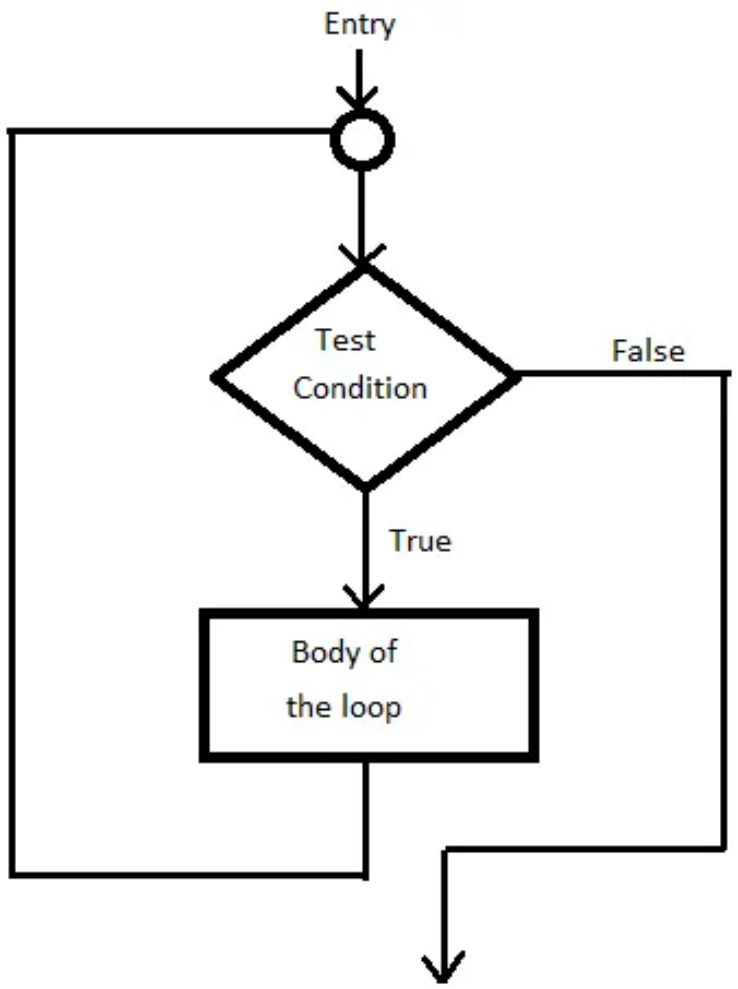
Loop or Iterative Statements

- Loop is defined as a block of statements which are repeatedly executed for a certain number of times till a particular condition is satisfied.
- When the condition becomes false, the loop terminates and the control is passed to the statement immediately following the loop.
- A loop consists of two segments: *control statement* and *body of the loop*.
- The control statement in loop decides whether the body is to be executed or not.

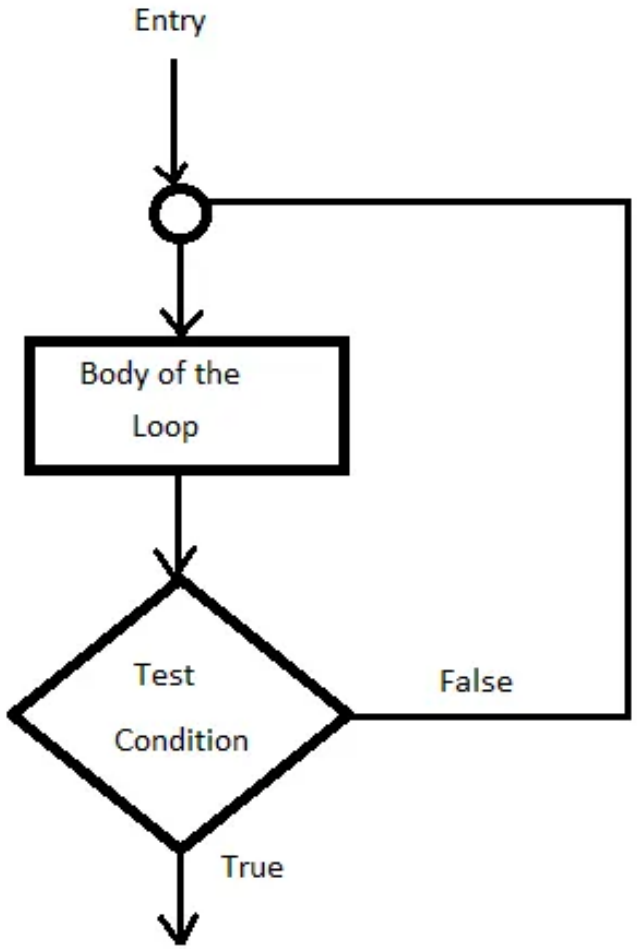
Entry Controlled and Exit Controlled loop

- Depending on the position of the test condition in the loop, a loop may be classified either as the entry controlled loop or exit controlled loop.
- In the entry controlled loops, conditions are tested before the start of the loop execution. If the conditions are satisfied, then the body of the loop will be executed.
- **while loop** and **for loop** are examples of entry controlled loop.
- In the case of an exit-controlled loop, the test is performed at the end of the body of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.
- **Do While loop** is an example of Exit controlled loop.

Entry Controlled and Exit Controlled loop



Entry Controlled Loop



Exit Controlled Loop

TYPES OF LOOP

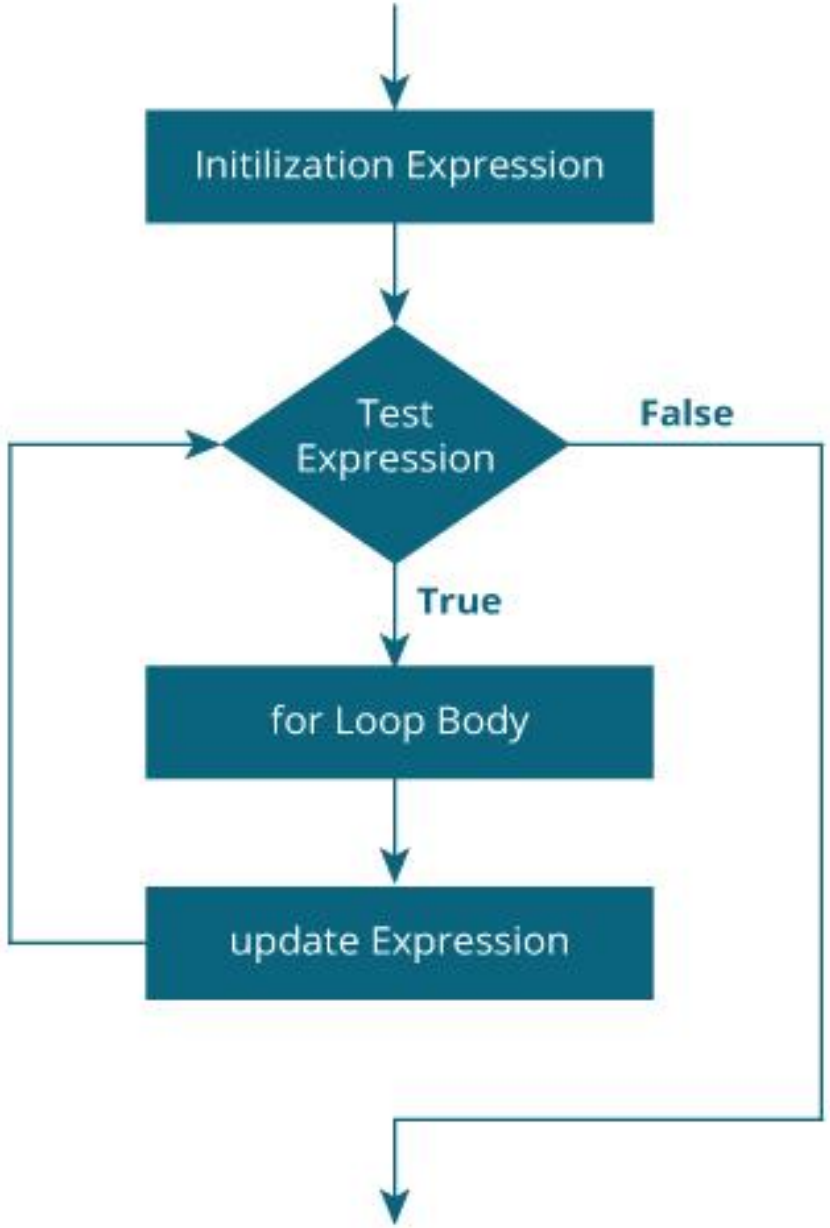
- *for* loop
- *while* loop
- *do while* loop

FOR LOOP

- When the number of repetitions is known in advance, the use of this loop is more efficient and so is also called *determinate* or *definite loop*.
- The general syntax is:

```
for(initialization; test condition; increment or decrement)
    {
        statement(s);       //body of the loop
    }
```

Flow Chart of for loop



HOW FOR LOOP WORKS?

- The **initialization expression** is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

Example : for loop

```
// Displaying your name and country 10 times each
#include <stdio.h>
#include <conio.h>
int main()
{
        int i;
        for(i=0; i<10; i++)
          {
            printf("\n My name is Prabin Kumar.");
            printf("\n I live in Nepal.");
          }
       getch();
       return 0;
}
```

Example: 2 Printing Multiplication Table

```
//Multiplication table
#include <stdio.h>
#include <conio.h>
int main()
{
   int n,t;
   printf(“Enter a number” );
   scanf(“%d”,&n);
   for(j=1;j<=10;j++)
   {
            t = n *i;
            printf("%d * %d = %d\n",n,i,t);
   }
   getch();
   return 0;
}
```

Practice Time

1) Write a program to read an integer number n from keyboard and display the message "Get Well Soon" n times.
2) Write a program to calculate the factorial of a number using for loop.
3) Write a program that asks an integer number n and calculate sum of all natural numbers from 1 to n.
4) Compute $1^2+2^2+3^2+\ldots+n^2$ using for loop (Take n from user).

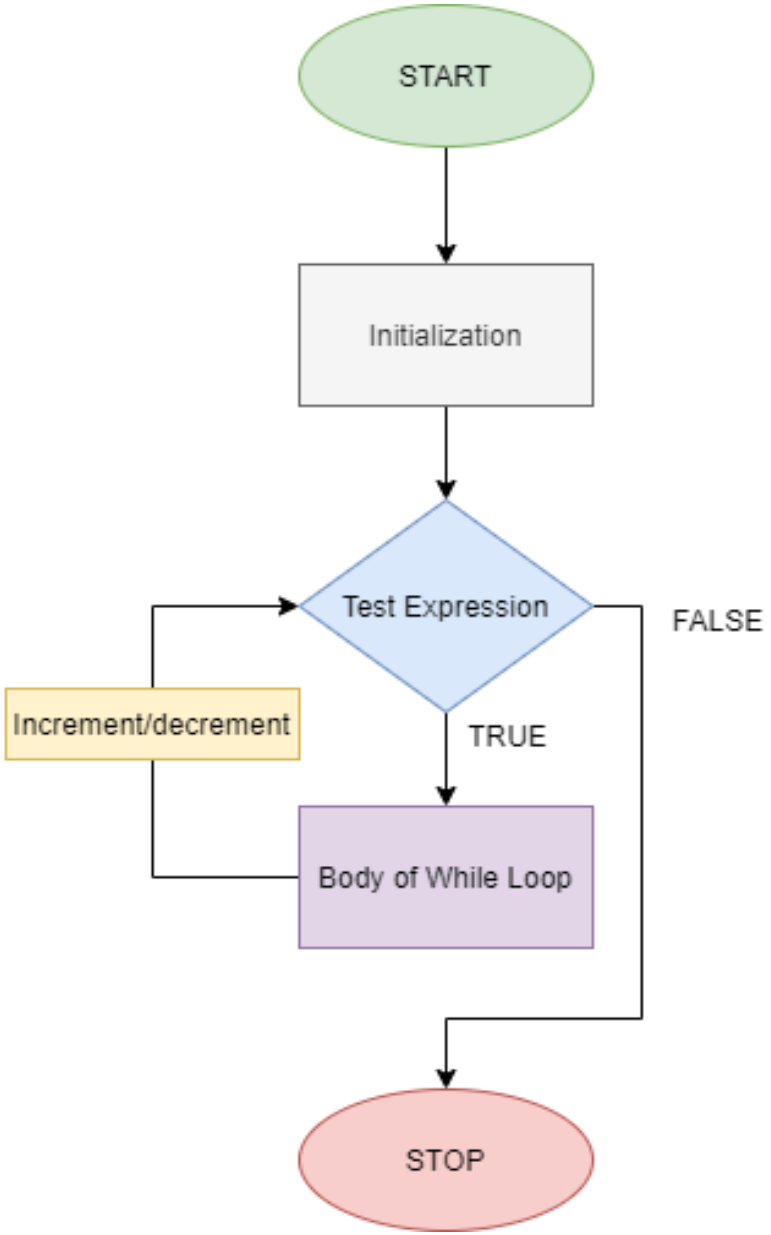
while loop

- The syntax is:

```
initialization
while(test condition)
{
        body of loop;
}
```

- The test condition is evaluated at first and if the condition is true, then the body of the loop is executed.
- After execution of the body once, the test condition is again evaluated and if it is true, the body is executed once again.
- This process repeats until the test condition finally becomes false and then the control is transferred out of the loop to the statement immediately following the loop.

Flow Chart of while loop



Example: while loop

```
// Print numbers from 1 to 5
#include <stdio.h>
int main() {
  int i = 1;
   while (i <= 5)
   {
      printf("%d\n", i);
      ++i;
    }
   return 0;
}
```

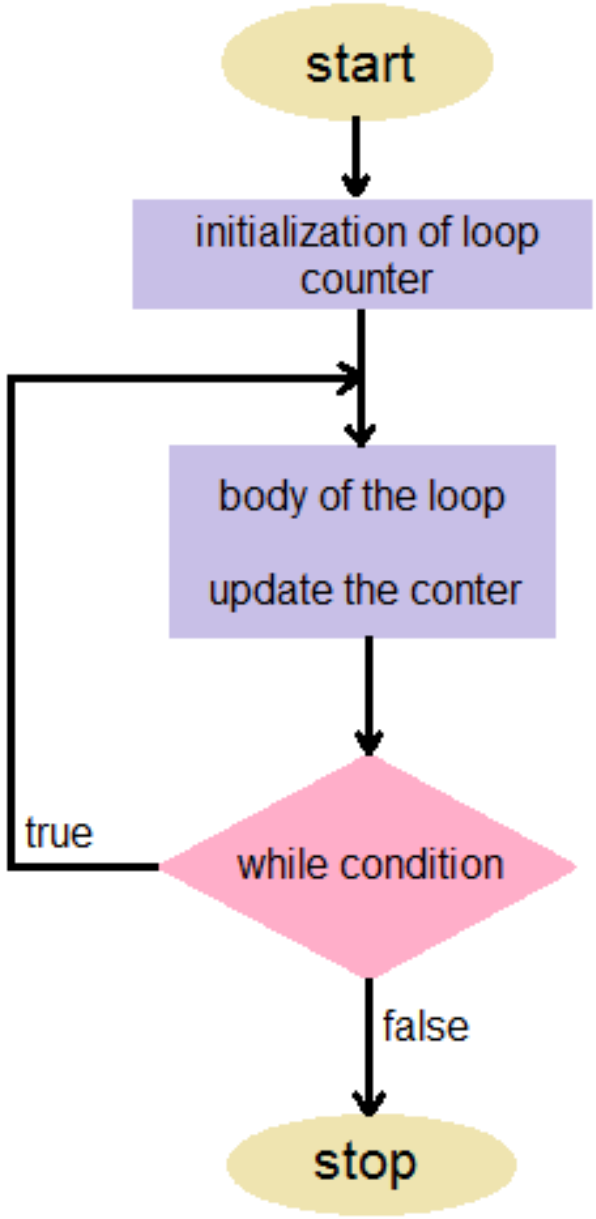
DO WHILE LOOP

- The syntax is:

```
do
{
    statement(s) or body of loop;
} while(test condition);
```

- In *do while* loop, the body of the loop is executed at first without testing the condition.
- At the end of the loop, test condition is evaluated and if the condition is true, the body is executed; otherwise the loop gets terminated and control is passed to the statement immediately following the loop.
- Important to Note: The body of the loop is executed at least once in a *do while* loop regardless of the test condition.

Flowchart of do while loop



Example:

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

int main()
{
    int i=1;
    do{
        printf("%i\n",i);
         i++;
    }
    while(i<=10);

    return 0;
}
```

```
F:\ctutorial\dowhilec\bin\Debu...
1
2
3
4
5
6
7
8
9
10

Process returned 0 (0x0)   execution time
Press any key to continue.
```

NESTED LOOP

- When the body part of a loop contains another loop, then the inner loop is said to be nested within the outer loop.
- There is no limit of the number of loops that can be nested.
- In case of nested loops, for each value or pass of the outer loop, the inner loop is executed completely. Therefore, inner loop operates fast and outer loop operates slow.

Nested Loop

The nested for loop means any type of loop which is defined inside the 'for' loop.

```
for (initialization; condition; update)
{
   for(initialization; condition; update)
   {
        // inner loop statements.
   }
   // outer loop statements.
}
```

Example: Nested Loop

```
#include <stdio.h>
int main()
{
   int i, j, rows;

   printf("Enter number of rows: ");
   scanf("%d",&rows);

   for(i=rows; i>=1; --i)
   {
      for(j=1; j<=i; ++j)
      {
         printf("%d ",j);
      }
      printf("\n");
   }

   return 0;
}
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

break, continue, return and exit

break - The break statement is used to jump out of loop. After the break statement control passes to the immediate statement after the loop.

continue - Using continue we can go to the next iteration in loop.

return - Exits the function.

exit - It is used to exit the execution of program.

note: break and continue are *statements,* exit is *function.*

BREAK STATEMENT

- The *break* statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop.
- As we know, a loop is terminated when its condition becomes false, however if we have to terminate the loop without testing loop the termination condition, then *break* statement is useful.
- Syntax:

```
break;
```

break statement

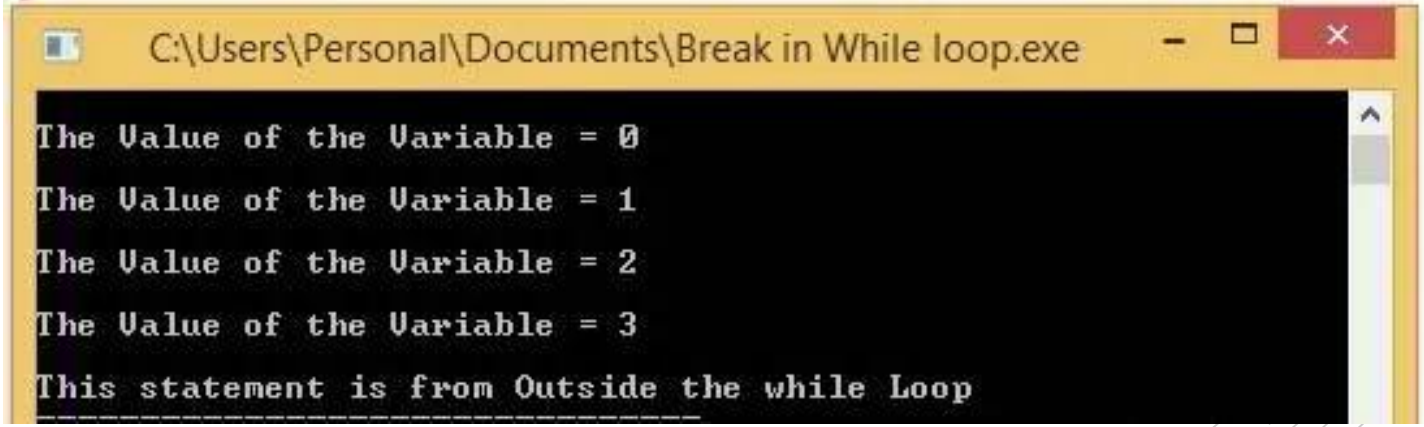
```
while (test Expression)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

```
for (init, condition, update)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

Example: break statement

```c
#include <stdio.h>
int main()
{
  int i =0;
  while(i<=10)
   {
     printf("\nThe Value of the Variable = %d\n", i);
     i++;

     if (i==4)
      {
       break;
      }
   }
  printf("\nThis statement is from Outside the while Loop");
  return 0;
}
```

```
// Program to test whether an entered number is prime or not
#include <stdio.h>
#include <conio.h>
 int main()
{
      int i,num;
      printf("\nEnter a number:");
      scanf("%d",&num);
      for(i=2;i<num;i++)
          {
                if(num%i==0)
            {
            printf("\nNot Prime.");
            break;
            }
          }
      if(i==num)
          {
          printf("\nPrime Number.");
          }
      getch();
}
```

continue Statement

- The *continue* statement is used to bypass the remainder of the current pass through the loop.
- The loop does not terminate when a *continue* statement is encountered.
- "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION"
- Syntax:

```
continue;
```

continue Statement

```
while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

```
for (init, condition, update)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
int i, num;
printf("\nEnter a number:");
scanf("%d", &num);
printf("\nThe even numbers from 2 to %d are:\n",
   num);
for(i=1;i<=num;i++)
   {
   if(i%2!=0)
    {
    continue;
    }
    printf("%d\t", i);
   }
getch();
}
```

goto Statement

- The *goto* statement is used to alter the normal sequence of program execution by unconditionally transferring control to some other part of the program.
- The *goto* statement transfers the control to a labeled statement somewhere in the current function using syntax:

```
goto label;
```

- Here, label is an identifier used to label the target statement to which the control would be transferred.
- The target statement must be labeled using syntax:

```
label: statements;
```

Example goto statement

```c
#include<stdio.h>
int main()
{
  int Totalmarks;
  printf("\n Please Enter your Subject Marks\n");
  scanf("%d", & Totalmarks);
  if(Totalmarks >= 50)
   {
     goto pass;
   }
  else
  {
    goto fail;
  }
  pass:
     printf("\n Congratulation! You made it\n");
  fail:
     printf("\n Better Luck Next Time\n");

  return 0;
}
```


// Program for Armstrong number

```
int main()
  {
      int num,digit,sum=0;
      int temp;

      printf("\nEnter number to be checked:\t");
      scanf("%d", &num);
      temp=num;

      while(num!=0)
      {
          digit = num % 10;
          sum += digit*digit*digit;
          num /= 10;
      }
      if(temp==sum)
          printf("\nArmstrong Number!!!");
      else
          printf("Not Armstrong Number.");
      getch();
      return 0;
}
```

//Program for reversing a number

```
int main()
{
    int n, digit;
    printf("\n Enter number you want to reverse:");
    scanf("%d", &n);
    printf("\n The reverse of the entered number is:");
    while(n!=0)
    {
       digit=n%10;
       printf("%d", digit);
       n=n/10;
    }
    getch();
    return 0;
}
```

//Program to compute sum of digits of an integer

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int num, rem, q, sum=0;
        printf("\nEnter number to be checked:\t");
        scanf("%d", &num);
        do
            {
            q = num / 10;
            rem = num % 10;
            num = q;
            sum += rem;
            }while(q!=0);

        printf("\nThe sum of digits is:\t%d", sum);
        getch();
        return 0;
}
```

//Program to reverse a number

```
int main()
{
    long int num,rev=0;
    int digit;

    printf("\nEnter number to be reversed:\t");
    scanf("%ld", &num);

    while(num!=0)
      {
          digit = num%10;
          rev = rev*10+digit;
          num = num/10;
      }
    printf("\nThe reversed number is:%ld",rev);
    getch();
}
```

```
/*Program to read a number from keyboard and check whether it is a palindrome or not*/
int main()
{
        int num, rev=0, digit, temp;
        printf("\nEnter number to be checked:\t");
        scanf("%d", &num);
        temp = num;
        while(num!=0)
                {
                digit = num%10;
                rev = rev*10 + digit;
                num = num/10;
                }
        if(temp==rev)
            {
            printf("\nThe number is a palindrome");
            }
        else
            {
            printf("\nThe number is not a palindrome");
            }
        getch();
        }
```

//Program to convert decimal no. to binary

```
int main()
{
    long int decnum, binnum, rev=0, q=1, rem, i=1;

    printf("\nEnter decimal number:\t");
    scanf("%ld", &decnum);

      while(q!=0)
        {
        q = decnum / 2;
        rem = decnum % 2;
        decnum = q;
        rev = rev + rem*i;
        i = i * 10;
        }
    printf("\nThe corresponding binary number is:%ld", rev);
    getch();
}
```

//Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

```
int main()
{
      int a,b ,c, num;
      a=1;
      b=1;
      printf("\nEnter number upto which you want Fibonacci sequence:\t");
      scanf("%d", &num);
      printf("%d %d”,a,b);
      do
         {
            c = a + b;
            a = b;
            b = c;
            printf(",%d", c);
         }  while(i<=num);
        return 0;
}
```


Thank You