

Control Structure

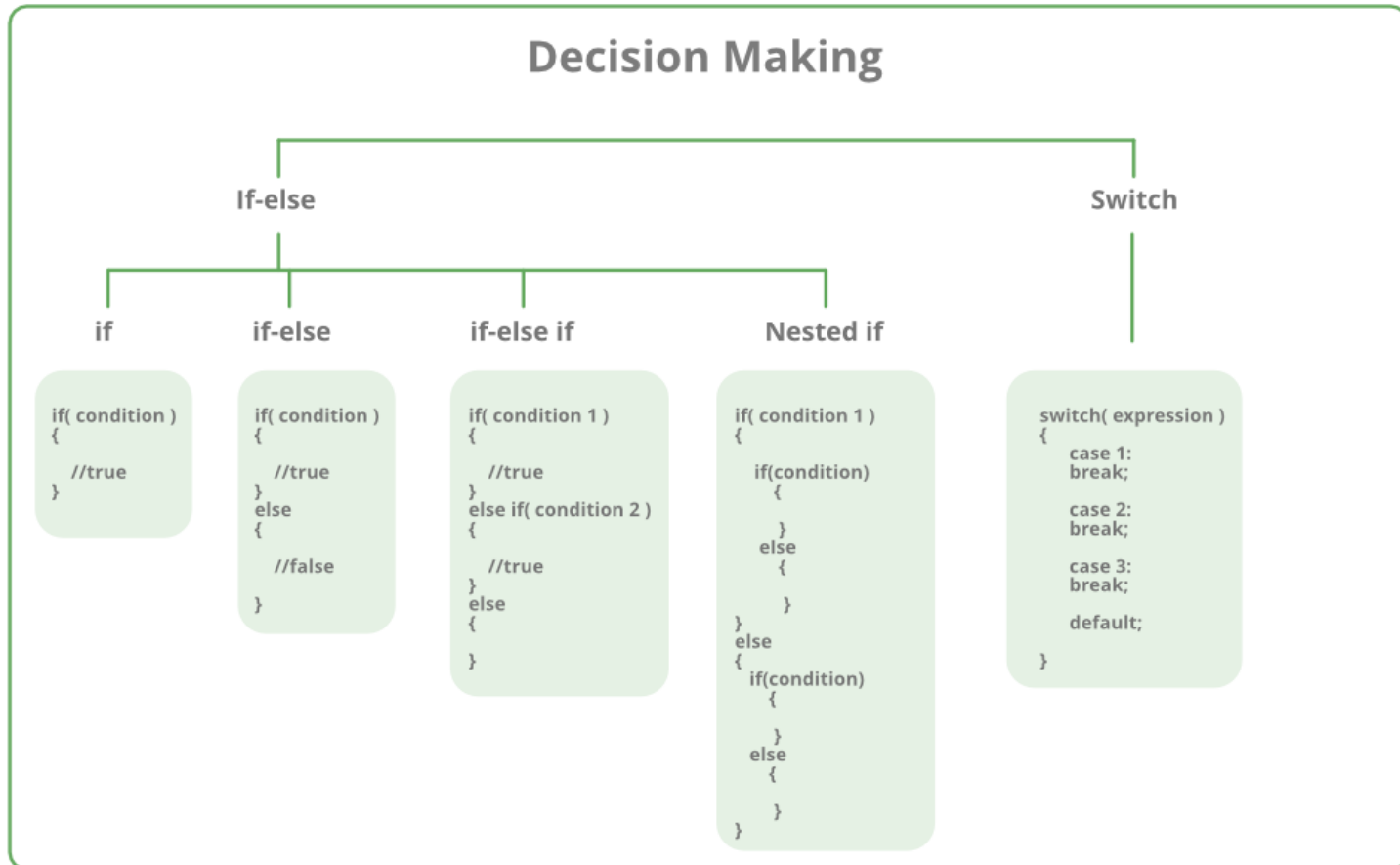
Unit 5

Er.Nipun Thapa

Decision Making Statement

- There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next.
- Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.
- For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else-if is one of the many ways of importing multiple conditions.

Decision Making Statement



if statement

- The *if* statement is a two way decision statement and is used together with an expression, i.e. test condition.
- The *if* statement evaluates the ***test expression*** first and then, if the value of the expression is ***true***, it execute the ***statement(s)*** within its block.
- Otherwise, it skips the statements within its block and continues from the first statement outside the *if* block.

if statement

- The general syntax is :

```
if(condition)
```

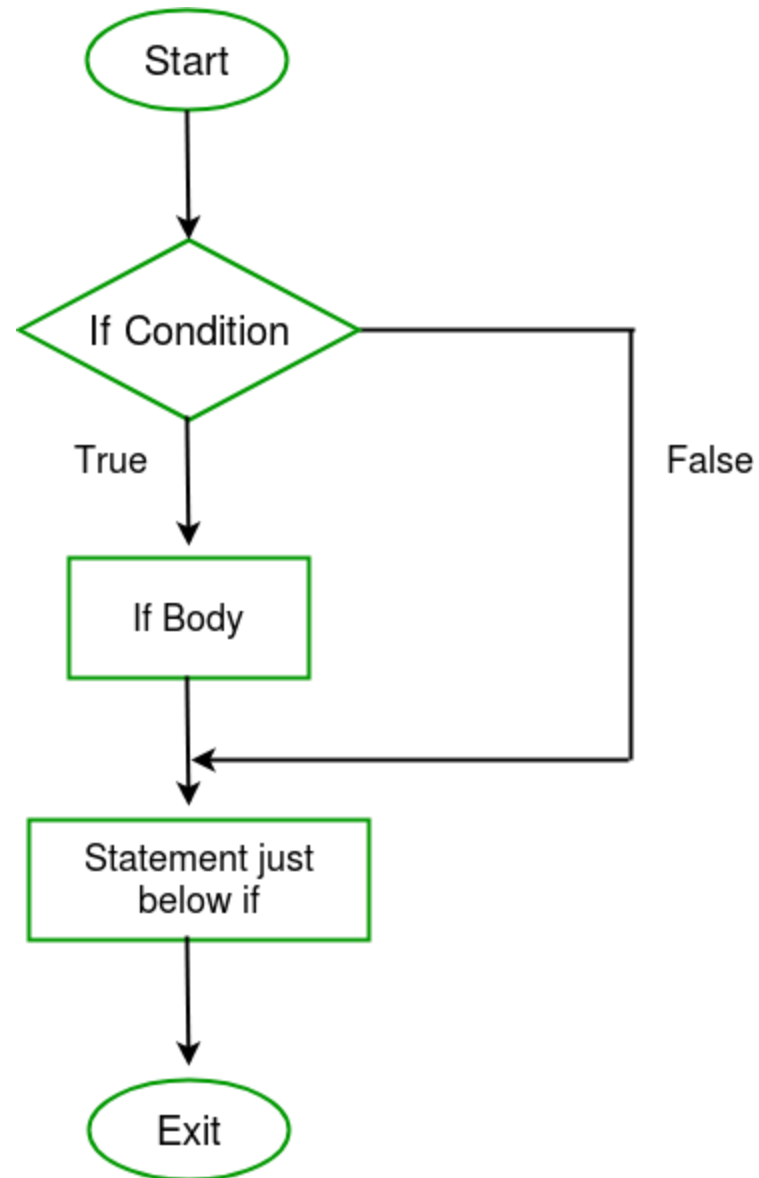
```
{
```

```
statement(s);
```

```
}
```

if statement

- The flowchart



if statement

```
// LAB 3 Q.N.1 : Program to check whether entered number is negative.(using if statement)
#include<stdio.h>
int main()
{
    int number;
    printf("Entered number:");
    scanf("%d",&number);
    //decision making statement
    if(number<0)
    {
        printf("The given number is negative");
        printf("\n The number %d is negative",number);

    }
    return 0;
}
```

if else statement

- The *if...else* statement is an execution of the simple *if* statement.
- It is used when there are two possible action – one when a condition is *true*, and the other when the condition is *false*.
 - If expression evaluates to *true*, true block statement is executed
 - If expression evaluates to *false*, false block statement is executed

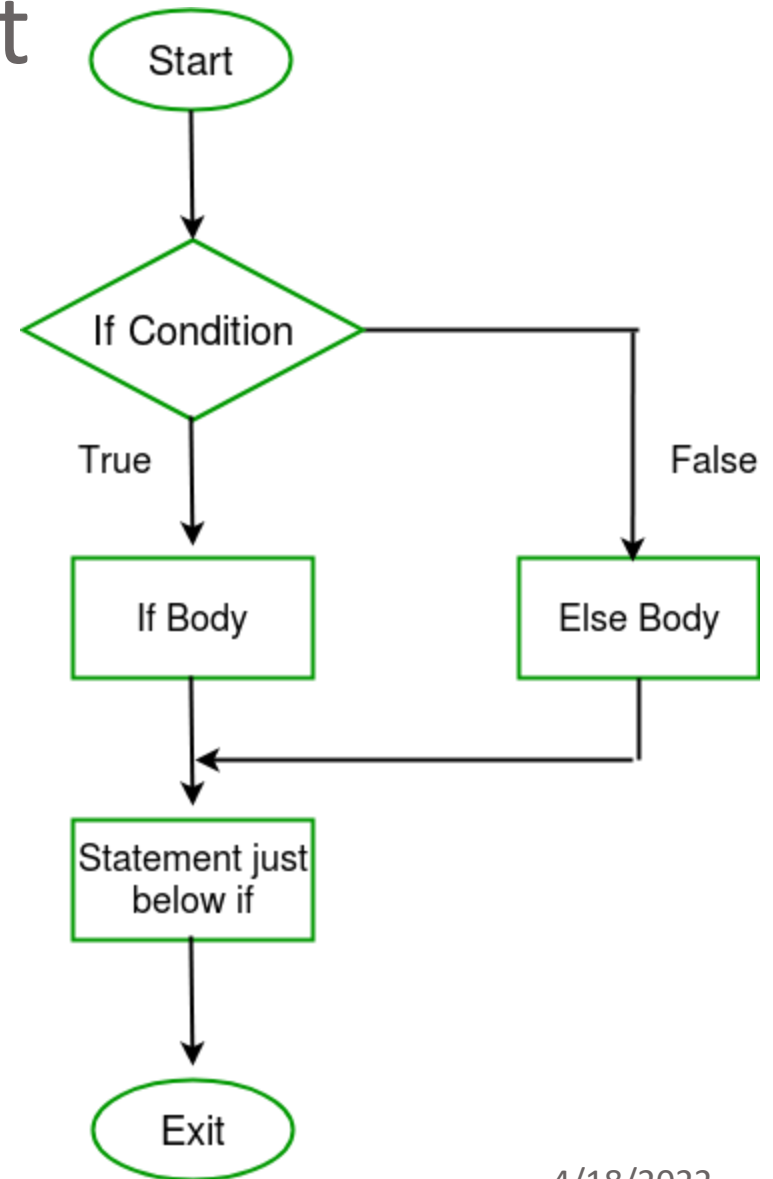
if else statement

- Syntax

```
If( condition )  
{  
    true_statement(s) ;  
}  
else  
{  
    false_statement(s);  
}
```

if else statement

- Flowchart



if else statement

//lab 3 QN 2: Check given number is positive or negative.

```
#include<stdio.h>
int main()
{
    int number;
    printf("Enter the number:");
    scanf("%d",&number);
    //decision making
    if(number>=0) // true condtion
    {
        printf("Your entered number is positive");
    }
    else // else(number<0), false
    {
        printf("Your entered number is negative");
    }
    return 0;
}
```

if else statement

```
//LAB 3 QN3: Program to find the number is even or odd.(using If..else statement)
#include<stdio.h>
int main()
{
    int number;
    printf("Enter number:");
    scanf("%d",&number);

    if(number%2==0) //decision making
    {
        printf("Your number is even");
    }
    else
    {
        printf("Your number is odd");
    }
    return 0;
}
```

if else statement

```
/*LAB 3 QN4: Program to find the profit or loss using selling price and
cost price.(using If..else statement)*/
#include<stdio.h>
int main()
{
    int SP,CP,P,L;
    printf("Enter selling price(SP) and Cost price(CP):");
    scanf("%d%d",&SP,&CP);
    if(SP>CP)
        {
            P=SP-CP;
            printf("Your profit amount is = %d",P);
        }
    else
        {
            L=CP-SP;
            printf("Your loss amount is = %d",L);
        }
    return 0;
}
```

if else statement

```
/*LAB 3 QN5: Program to find the largest number among two number.
(using If..else statement)*/
#include<stdio.h>
int main()
{
    int n1,n2;
    printf("Enter two numbers:");
    scanf("%d%d",&n1,&n2);
    //decision making
    if(n1>n2)
    {
        printf("%d is greater than %d",n1,n2);
    }
    else
    {
        printf("%d is greater than %d",n2,n1);
    }
    return 0;
}
```

if else statement

```
/*LAB 3 QN6: Program to check the number is divisible  
by 5 or not.  
(using If..else statement)*/
```

Nested If statement

- When a series of decisions are involved, we may have to use more than one ***if else*** statement in nested.
- A ***if*** statement contains another if else statement is called ***nested if*** statement.
- The condition are execute from top to bottom checking each condition whether it meets the condition criteria or not
- if it found the condition is true then it executes the block associated statements of true part else it goes to next condition to execute.

Nested If statement

- **Format**

If (condition1)

```
{
```

```
    if (condition 2)
```

```
        {
```

```
            statement_1;
```

```
        }
```

```
    else
```

```
        {
```

```
            statement_2;
```

```
        }
```

```
    }
```

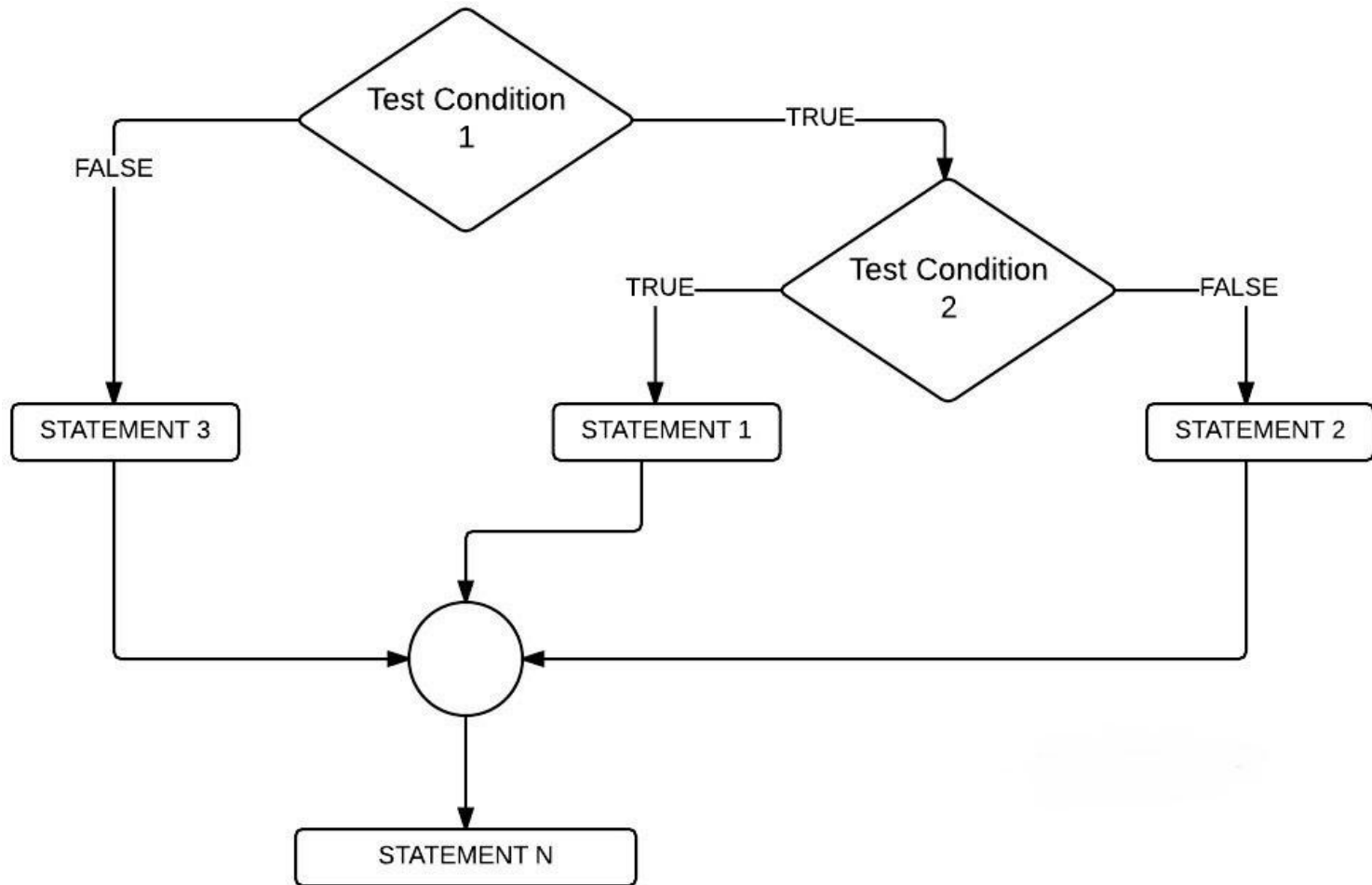
```
else
```

```
{
```

```
    statement_3;
```

```
}
```

Nested If statement



Nested **if..else** statement

- Similar to **nested if** statement, **if..else** statements shall also be written inside the body of another **if..else** body called nested **if..else** statement.
- An entire **if..else** construct is written under either the body of an **if** statement or the body of an **else** statement.
- The **if..else** statement shall be declared within either anyone of **if** or **else** body statement or both.

Nested If..else statement

- **Format**

If (condition1)

{

 if (condition 2)

 {

 statement_1;

 }

 else

 {

 statement_2;

 }

}

else

{

 if (condition 3)

 {

 statement_3;

 }

 else

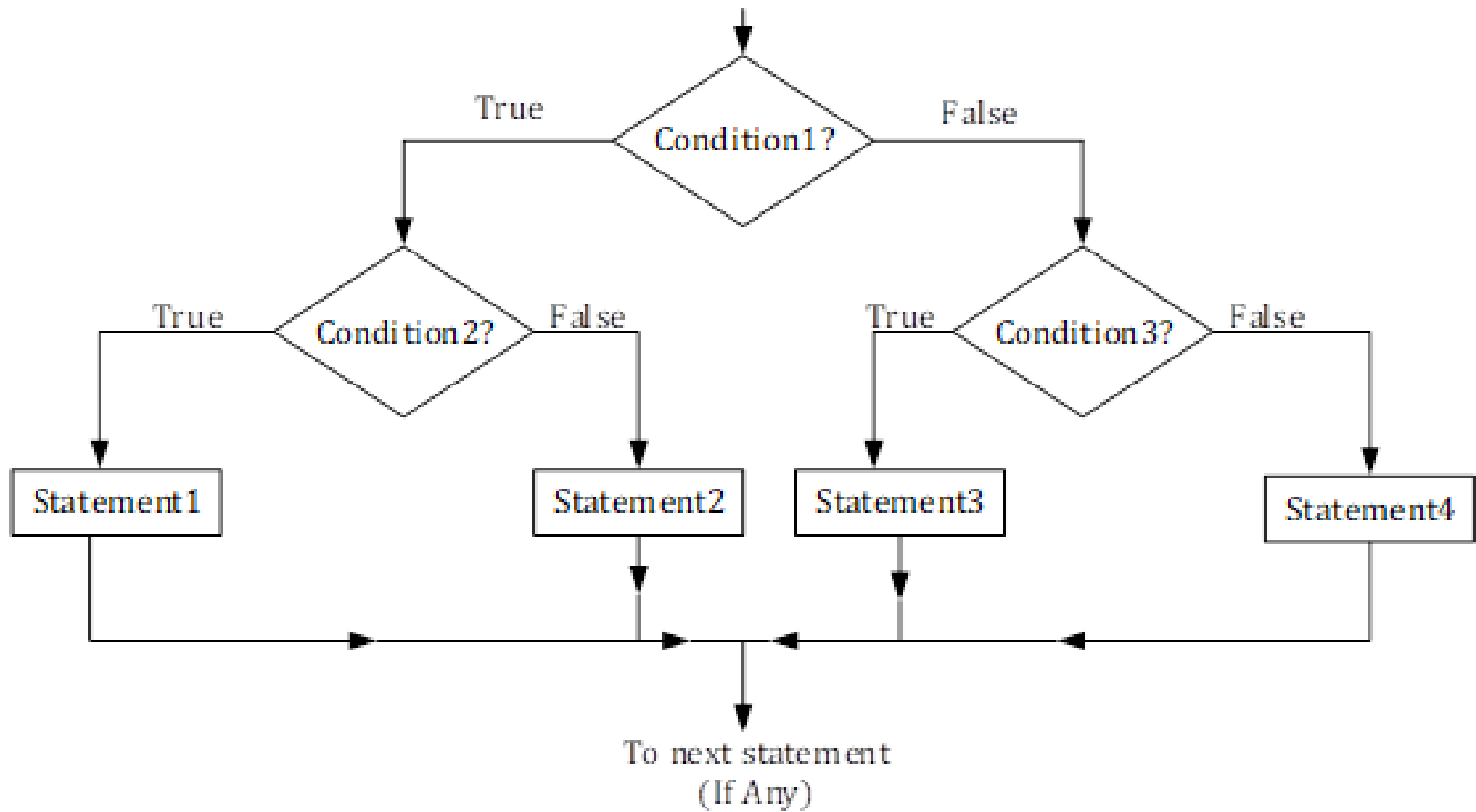
 {

 statement_4;

 }

}

Nested If..else statement



Nested If statement

/*LAB 3 QN7: Program to find largest number among the three numbers using nested if..else condition.ex:

1,3,5 :- 5 is the largest number*/

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

```
int main()
```

```
{
```

```
int n1,n2,n3;
```

```
printf("Enter three numbers:");
```

```
scanf("%d%d%d",&n1,&n2,&n3); //5,2,1
```

```
if(n1>n2)
```

```
{
```

```
if(n1>n3)
```

```
    printf("%d is the largest number",n1);
```

```
    else
```

```
        printf("%d is the largest number",n3);
```

```
    }
```

```
else // (n2<n1)
```

```
{
```

```
if(n2>n3)
```

```
    printf("%d is largest number",n2);
```

```
else
```

```
    printf("%d is the largest number",n3);
```

```
}
```

```
return 0;
```

```
}
```

If-else-if ladder statement

- This is a type of nesting in which there is an if...else statement in every part except the last else part.
- This type of nesting is frequently used in programs and also known as else if ladder .
- This construct is if-else-if ladder because of its appearance

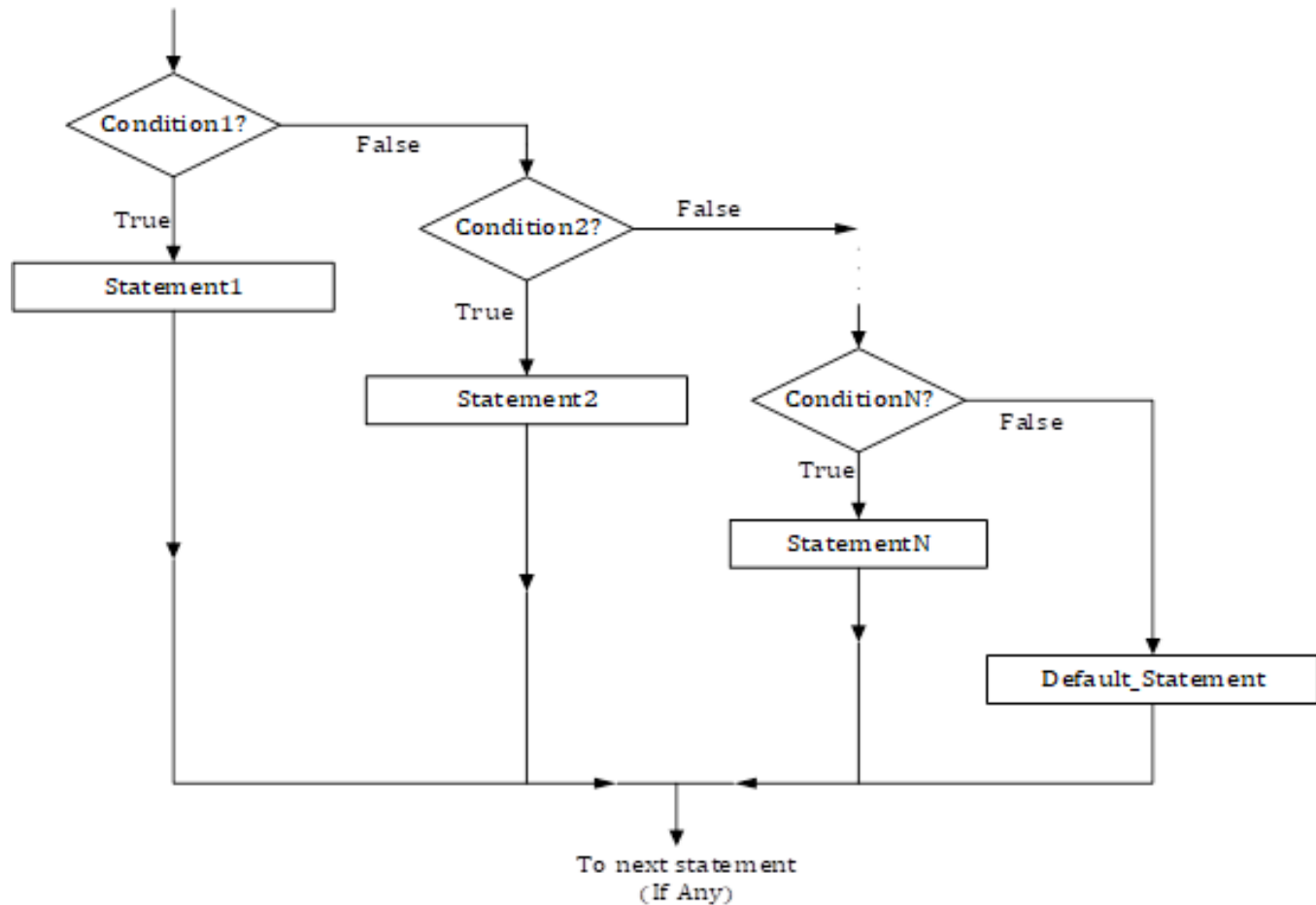
If-else-if ladder statement

- Syntax

```
If(condition 1)
{
    statement_1;
}
else if(condition 2)
{
    statement_2;
}
else if(condition 3)
{
    statement_3;
}
.....
else
{
    Final statement;)
}
```


If-else-if ladder statement

flowchart



If-else-if ladder statement

//Lab 3.QN 8 Program to relate two integers using =, > or < symbol

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

```
int main()
```

```
{ int number1, number2;
```

```
    printf("Enter two integers: ");
```

```
    scanf("%d %d", &number1, &number2);
```

```
    if(number1 == number2)
```

```
        {           printf("Result: %d = %d",number1,number2); }
```

```
    else if (number1 > number2)
```

```
        {           printf("Result: %d > %d",number1,number2); }
```

```
    else
```

```
        {           printf("Result: %d < %d",number1,number2); }
```

```
    return 0;
```

```
}
```

If-else-if ladder statement

//Lab 3.QN9: C program to find largest number from four given numbers.

```
int main()
{
    float a,b,c,d, lg;
    printf("Enter four numbers:\n");
    scanf("%f%f%f%f", &a, &b, &c, &d);

    if(a>b && a>c && a>d)
    {
        lg = a;
    }
    else if(b>a && b>c && b>d)
    {
        lg = b;
    }
    else if(c>a && c>b && c>d)
    {
        lg = c;
    }
    else
    {
        lg = d;
    }
    printf("Largest = %f", lg);
    return(0);
}
```

The **Switch** statement

- The switch is a multiple branch selection statement that successively test the value of an expression against a list of integer or character constant.
- When a match is found, the statements associated with that constant are executed.
- Useful when there are a number of else alternatives and one of them is to be selected on the basis of some criteria.
- It allows a user to choose a statement among several alternatives.
- The constants in switch statement may be either **char** or **int** type only.

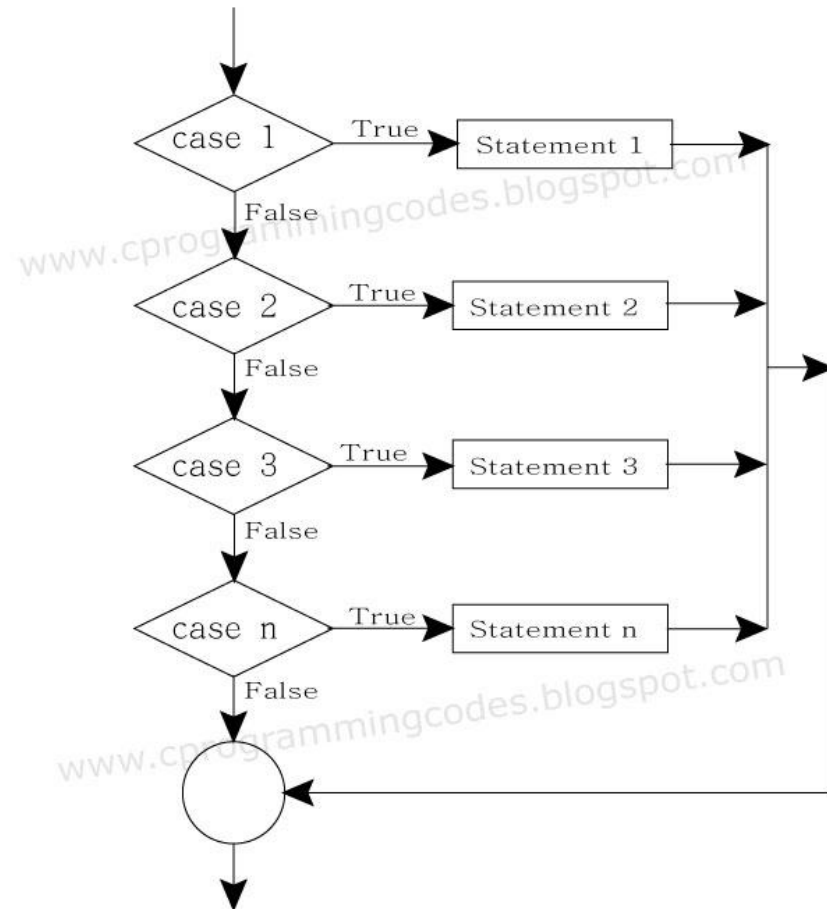
The Switch statement

- Syntax

```
switch(expression)
{
    case constant1:
        statement_1;
        break;
    case constant2:
        statement_2
        break;
    .....
    case constantN;
        statement_N;
        break;
    default:
        default_statement;
}
```

The Switch statement

Flowchart



The Switch statement

```
//LAB 3 QN10:Program to choose days from weeks using switch statement  
// 1.sunday 2.monday ..... to 7.saturday
```

```
#include<stdio.h>  
int main()  
{  
int day;  
printf("1.SUN\n2.MON\n3.TUE\n4.WED\n5.THU\n6.FRI\n7.SAT:");  
scanf("%d",&day);  
switch(day)  
{  
case 1:  
    printf("Its SUnDay!!!");  
    break;  
case 2:  
    printf("Its MOnday!!!");  
    break;  
case 3:  
    printf("Its tuesday!!!");  
    break;  
case 4:  
    printf("Its Wednesday!!!");  
    break;  
case 5:  
    printf("Its Thursday!!");  
    break;  
case 6:  
    printf("Its Friday!!");  
    break;  
case 7:  
    printf("Its saturday!!");  
    break;  
default:  
    printf("Please insert from 1 to 7");  
}  
return 0;}
```

The Switch statement

//LAB 3 QN11:Program to perform the arithmetic operation using switch statement.

```
#include<stdio.h>
int main()
{
int a,b;
char ch;
printf("\nChoose:");
scanf(" %c",&ch);

printf("Enter two number:");
scanf("%d%d",&a,&b);

switch(ch)
{
case '+':
    printf("%d+%d=%d",a,b,a+b);
    break;
case '-':
    printf("%d-%d=%d",a,b,a-b);
    break;
case '*':
    printf("%d*%d=%d",a,b,a*b);
    break;
case '/':
    printf("%d/%d=%d",a,b,a/b);
    break;
default:
    printf("ERRRRRRRRRR");
}
return 0;
}
```

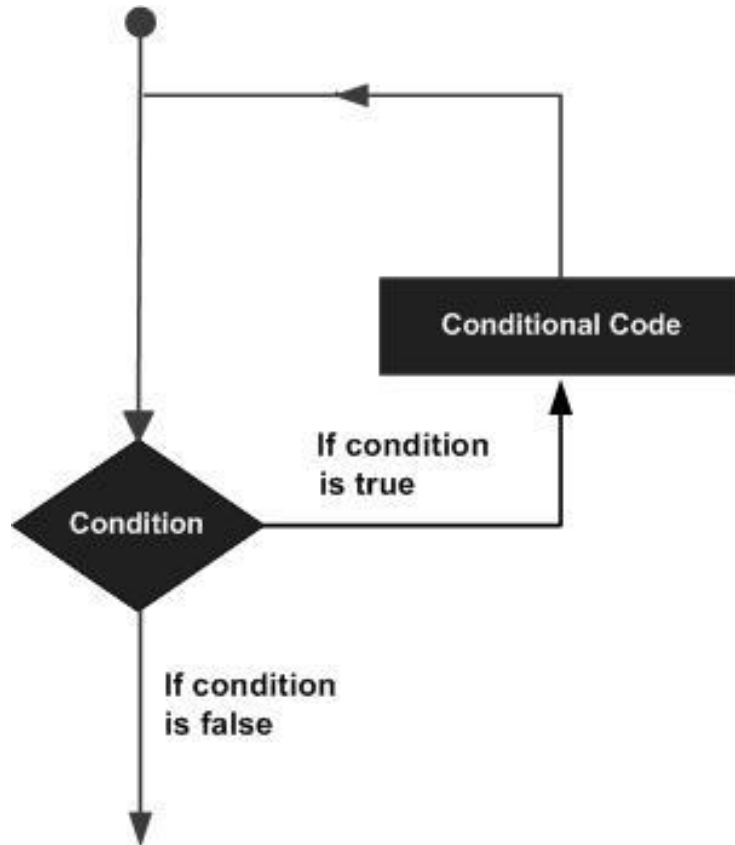

Loops

Er. Nipun Thapa

Introduction

- You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages

Introduction



What are Loops?

- A **Loop** executes the sequence of statements many times until the stated condition becomes false.
- A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.
- The purpose of the loop is to **repeat** the same code a number of times.

Types of Loops in C

Depending upon the position of a control statement in a program, looping in C is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

- In an **entry controlled loop**, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.
- In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

Types of Loops in C

- 'C' programming language provides us with three types of loop constructs:
 1. The while loop
 2. The do-while loop
 3. The for loop

1. While Loop

- It is an entry-controlled loop.
- In while loop, a condition is evaluated before processing a body of the loop.
 - If a condition is true then and only then the body of a loop is executed.
- After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false.
- Once the condition becomes false, the control goes out of the loop.

1. While Loop

- After exiting the loop, the control goes to the statements which are immediately after the loop.
- The body of a loop can contain more than one statement. If it contains only one statement, then the curly braces are not compulsory.
- In while loop, if the condition is not true, then the body of a loop will not be executed, not even once.
- It is different in do while loop.

1. While Loop

Syntax :

```
while (test condition)  
{  
    Body of the loop  
}
```

1. While Loop

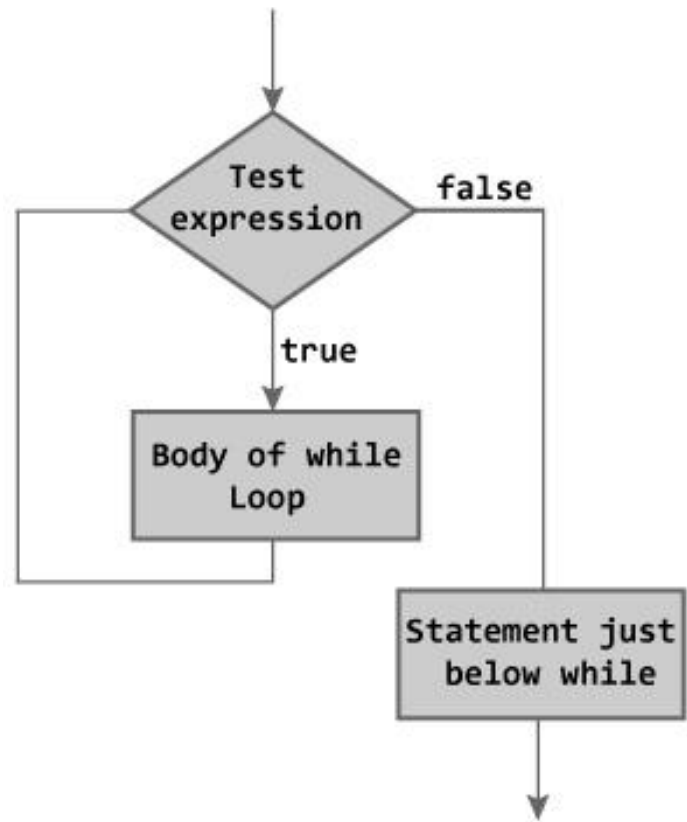


Figure: Flowchart of while Loop

1. While Loop

//LAB 4: Q.n.1 : Program to display 1 to 10 using while loop

Source code:

```
#include<stdio.h>
#include<conio.h>
    int main()
{
    int num=1; //initializing the variable
    while(num<=10) //while loop with condition
    {
        printf("%d\n",num);
        num++; //incrementing operation
    }
    return 0;
}
```

```

#include<stdio.h>
#include<conio.h>
int main()
{
    1 int num=1; //initializing the variab
    while (num<=10) 2 //while loop with con
    {
        printf ("%d\n", num);
        num++; //incremesting operat
    }
    return 0;
}

```

- We have initialized a variable called num with value 1. We are going to print from 1 to 10 hence the variable is initialized with value 1. If you want to print from 0, then assign the value 0 during initialization.
- In a while loop, we have provided a condition (num<=10), which means the loop will execute the body until the value of num becomes 10. After that, the loop will be terminated, and control will fall outside the loop.
- In the body of a loop, we have a print function to print our number and an increment operation to increment the value per execution of a loop. An initial value of num is 1, after the execution, it will become 2, and during the next execution, it will become 3. This process will continue until the value becomes 10 and then it will print the series on console and terminate the loop.

\n is used for formatting purposes which means the value will be printed on a new line.

1. While Loop

LAB 4 Q.n.2: Program to sum all integers from 1 to 100 using while loop

```
#include<stdio.h>
int main()
{
    int sum = 0 , i = 1;
    while (i<=100)
    {
        sum +=i;
        i++;
    }
    printf("Sum is %d \n",sum);
return 0;
}
```

1. While Loop

LAB 4 Q.n.3: Program to find the sum and average of the mark of five subjects using while loop

```
#include<stdio.h>
int main()
{
    int marks,total,i;
    float average;
    total = 0;
    i=1;
    while(i<=5)
    {
        printf("\n Enter marks of %d subject:",i);
        scanf("%d",&marks);
        total=total + marks;
        i++;
    }
    average = (float) total / 5;
    printf("\n The sum =%d\t and average of marks of five subject is: %f",total,average);
    return 0;
}
```

1. While Loop

LAB 4 .Qn4: Program to find the sum of digits of any num supplied by the user.

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

```
int main()
```

```
{
```

```
    int n, sum=0,rem;
```

```
    printf("Enter the number:");
```

```
    scanf("%d",&n);
```

```
    while(n>0)
```

```
    {
```

```
        rem=n%10;
```

```
        sum+=rem;
```

```
        n/=10;
```

```
    }
```

```
    printf("Sum of digits = %d\n",sum);
```

```
    return 0;
```

```
}
```

1. While Loop

LAB 4 QN5 : Program that check whether the entered number is Armstrong number.

```
#include<stdio.h>
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("\n Armstrong Number !!!");
    else
        printf("not Armstrong number.");
    return 0;
}
```


1. While Loop

LAB 4 QN6: Program to read a number and find and display its reverse.

```
#include<stdio.h>
int main()
{
    long int num, rev=0;
    int digit;
    printf("\n Enter number to be reversed:\t");
    scanf("%ld",&num);
    while(num!=0)
    {
        digit=num%10;
        rev=rev*10+digit;
        num=num/10;
    }
    printf("\n The reversed number is:%ld",rev);

    return 0;
}
```

1. While Loop

LAB 4 QN 7: Program to read a number from keyboard and check whether it is a palindrome or not

```
#include<stdio.h>
int main()
{
    int num,rev=0,digit,temp;
    printf("\n Enter 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");
    return 0;
}
```

1. While Loop

LAB 4 QN 8: Program to convert decimal number into binary

```
#include<stdio.h>
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("\n The corresponding binary number is:%ld",rev);
    return 0;
}
```

2. do-while loop

- A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop.
- The test is performed at the end of the loop rather than the beginning.
- It is also called an exit-controlled loop.

2. do-while loop

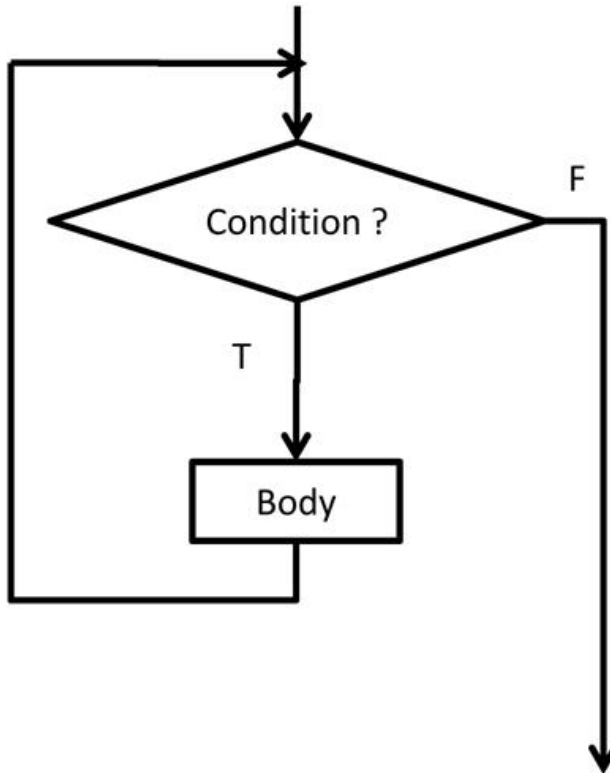
Syntax:

```
do {  
    statements  
} while (expression);
```

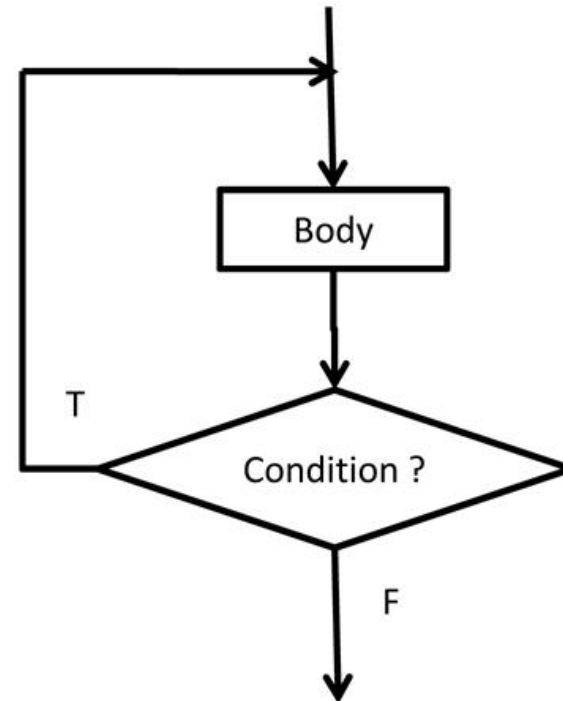
2. do-while loop

While versus Do-While Loops

while(condition)
body;



do {
body;
} while(condition);



2. do-while loop

- As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.
- In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.
- Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.
- The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

2. do-while loop

LAB 4 QN 9: Program to print a table of number 2.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num=1;    //initializing the variable
    do    //do-while loop
    {
        printf("%d\n",2*num);
        num++;    //incrementing operation
    }while(num<=10);
    return 0;
}
```



```

#include<stdio.h>
#include<conio.h>
int main()
{
    int num=1; 1 initializing t
    do //do-while loop
    {
        printf("%d\n", 2*num) 2
        num++; //incrementi
    }while (num<=10); 4
    return 0;
}

```

- First, we have initialized a variable 'num' with value 1. Then we have written a do-while loop.
- In a loop, we have a print function that will print the series by multiplying the value of num with 2.
- After each increment, the value of num will increase by 1, and it will be printed on the screen.
- Initially, the value of num is 1. In a body of a loop, the print function will be executed in this way: $2 * \text{num}$ where $\text{num}=1$, then $2 * 1 = 2$ hence the value two will be printed. This will go on until the value of num becomes 10. After that loop will be terminated and a statement which is immediately after the loop will be executed. In this case return 0.

2. do-while loop

LAB 4 QN 10: Program to print out all numbers from 1 to 10 using for do-while loop.

```
#include<stdio.h>
int main()
{
    int x=1;
    do
    {
        printf("%d\t",x);
        x++;
    } while(x<=10);
    return 0;
}
```

2. do-while loop

LAB 4 QN 11: Program to find the Fibonacci sequence: 1,1,2,3,5,8,13.....

```
#include<stdio.h>

int main()
{
    int fib1,fib2,prev,next,num;
    fib1=1;
    fib2=1;
    prev=fib1;
    printf("\nEnter number:");
    scanf("%d",&num);
    printf("%d",fib1);
    do
    { next=fib2+prev;
      prev=fib2;
      fib2=next;
      printf(",%d",prev);
    }while(num>next);
    return 0;
}
```

Comparison of while and do-while loop

while	do-while
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);

3. The for loop

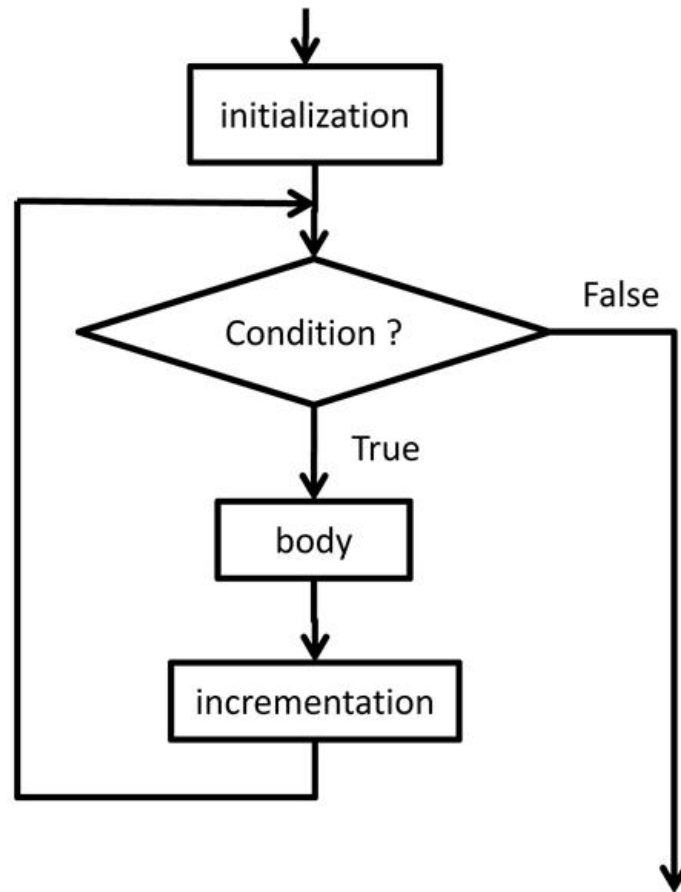
- A **for loop** is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.
- **Syntax :**

```
for(initial value; condition; incrementation or decrementation )  
{  
    statements;  
}
```

3. The for loop

- The **initial value** of the for loop is performed only once.
- The **condition** is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The **incrementation/decrementation** increases (or decreases) the counter by a set value.

3. The for loop



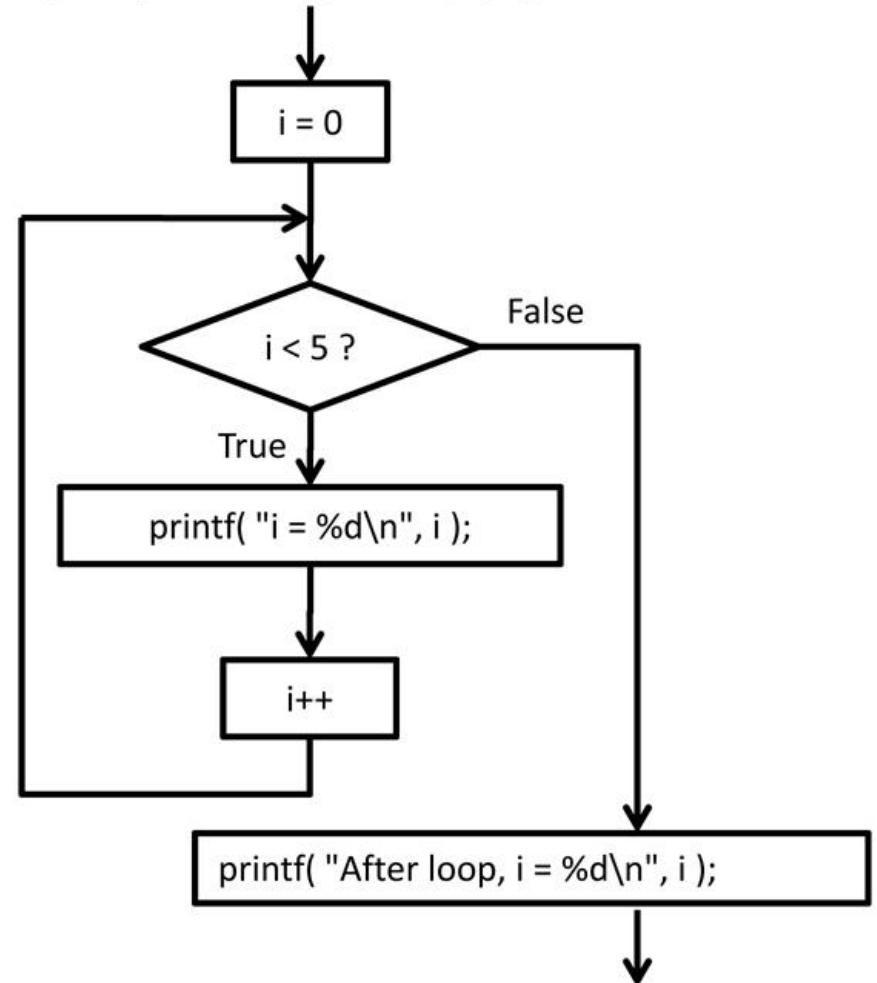
3. The for loop

- The initialization step occurs one time only, before the loop begins.
- The condition is tested at the beginning of each iteration of the loop.
 - If the condition is true (non-zero), then the body of the loop is executed next.
 - If the condition is false (zero), then the body is not executed, and execution continues with the code following the loop.
- The incrementation happens **AFTER** the execution of the body, and only when the body is executed.

3. The for loop

```
int i;
```

```
for( i = 0; i < 5; i++ )  
    printf( "i = %d\n", i );  
printf( "After loop, i = %d\n", i );
```



3. The for loop

LAB 4 QN 12: Program to print out all numbers from 1 to 10 using for loop

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int x;
    for(x=1;x<=10;x++)
        printf("%d\t",x);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int number; 1
    2 for (number=1; number<=10; number++)
    {
        printf("%d\n", number) 3
    }
    return 0;
}
```

- We have declared a variable of an int data type to store values.
- In for loop, in the initialization part, we have assigned value 1 to the variable number. In the condition part, we have specified our condition and then the increment part.
- In the body of a loop, we have a print function to print the numbers on a new line in the console. We have the value one stored in number, after the first iteration the value will be incremented, and it will become 2. Now the variable number has the value 2. The condition will be rechecked and since the condition is true loop will be executed, and it will print two on the screen. This loop will keep on executing until the value of the variable becomes 10. After that, the loop will be terminated, and a series of 1-10 will be printed on the screen.

3. The for loop

The comma operator

- C has a comma operator, that basically combines two statements so that they can be considered as a single statement.
- About the only place this is ever used is in for loops, to either provide multiple initializations or to allow for multiple incrementations.

For example:

```
int i, j = 10, sum;  
for( i = 0, sum = 0; i < 5; i++, j-- )  
    sum += i * j;
```

3. The for loop

- Also, we can skip the initial value expression, condition and/or increment by adding a semicolon.

For example:

```
int i=0; int max = 10;
for (; i < max; i++)
{
    printf("%d\n", i);
}
```

3. The for loop

LAB 4 QN 13 : Program to calculate the factorial of a positive number read from user.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int num,i;
    long fact=1;
    printf("\n ENter number:");
    scanf("%d",&num);
    for(i=1;i<=num;i++)
        fact=fact*i;
    printf("\nThe factorial is:%ld",fact);
    return 0;
}
```

Nested Loop

- The code inside a loop can be any valid C code, including other loops.
- Any kind of loop can be nested inside of any other kind of loop.
- **Nested loop** means a loop statement inside another loop statement. That is why nested loops are also called as “**loop inside loop**”.

Nested Loop

- **Syntax for Nested For loop:**

```
for ( initialization; condition; increment )  
{  
    for ( initialization; condition; increment )  
        {  
            // statement of inside loop  
        }  
    // statement of outer loop  
}
```


Nested Loop

- **Syntax for Nested While loop:**

```
while(condition)
{
    while(condition)
    {
        // statement of inside loop
    }
    // statement of outer loop
}
```

Nested Loop

- **Syntax for Nested Do-While loop:**

```
do{
```

```
    do{
```

```
        // statement of inside loop
```

```
    }while(condition);
```

```
    // statement of outer loop
```

```
}while(condition);
```

Nested Loop

LAB 4 QN 14: Program to show multiplication table .

```
#include <stdio.h>
int main()
{
    int i, j; int table = 2;
    int max = 10;
    for (i = 1; i <= table; i++)
    {
        for (j = 0; j <= max; j++)
        {
            printf("%d x %d = %d\n", i, j, i*j);
        }
        printf("\n"); /* blank line between tables */
    }
    return 0;
}
```

Jumping Statements

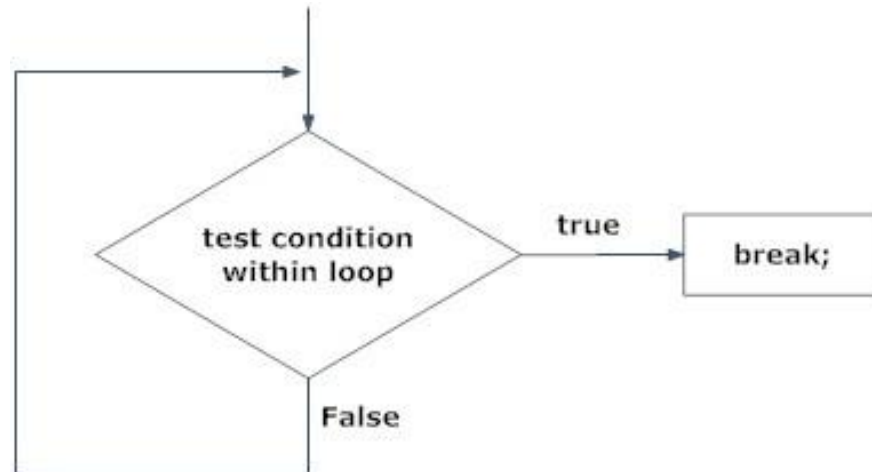
- The **break**; **continue**; and **goto**; statements are used to alter the normal flow of a program.
- Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

break statement

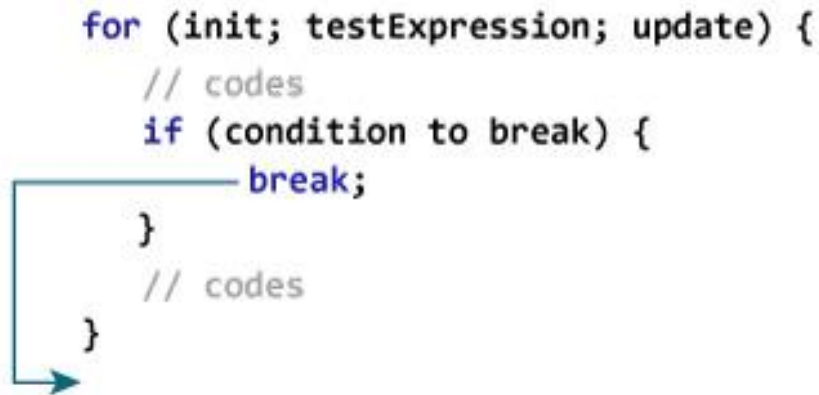
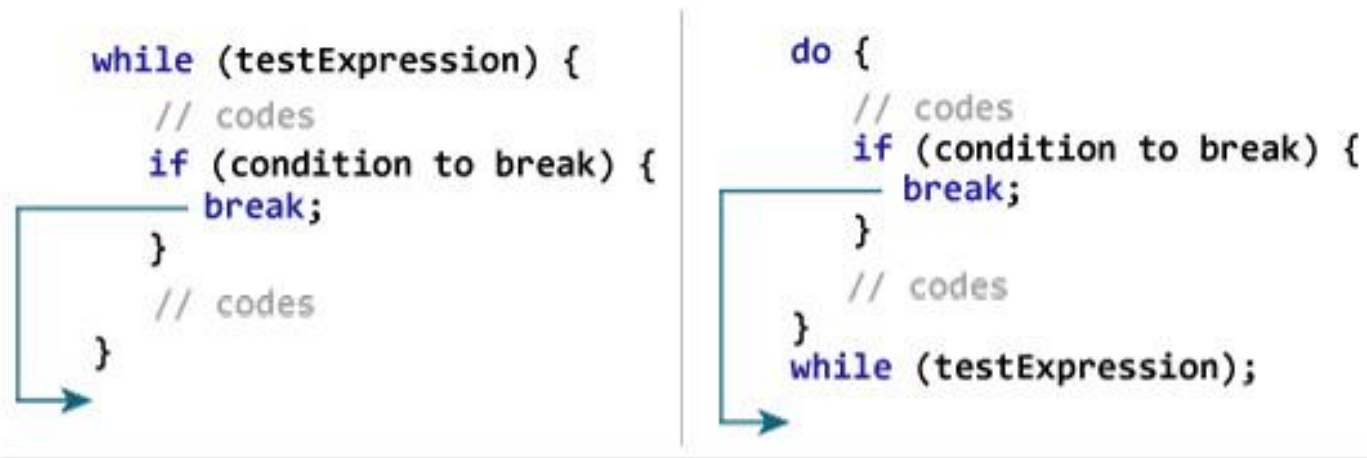
- break statement is used with conditional if statement.
- The break is used in terminating the loop immediately after it is encountered.
- It is also used in switch...case statement. which is explained in next topic.
- The break statement is almost always used with if...else statement inside the loop.

Syntax :

break ;



How break statement works?



Example 1: break statement

LAB 4 QN 15: Program to illustrate the use of break within loop

```
#include<stdio.h>
int main()
{
    int x;
    for(x=1;x<=10;x++)
    {
        if(x>4)
            break;
        printf("%d\t",x);
    }
    printf("\nEnd");
    return 0;
}
```

o/p : 1 2 3 4
 End

LAB 4 QN16 : Program to calculate the sum of numbers (10 numbers max) .If the user enters a negative number, the loop terminates.

```
#include <stdio.h>
int main() {
    int i;
    double number, sum = 0.0;
    for (i = 1; i <= 10; ++i) {
        printf("Enter a n%d: ", i);
        scanf("%lf", &number);
        if (number < 0.0) { // if the user enters a negative number, break the
            loop
            break;
        }
        sum += number; // sum = sum + number;
    }
    printf("Sum = %.2lf", sum);
    return 0;
}
```


LAB 4 QN17: Write a C Program Which use of break statement.

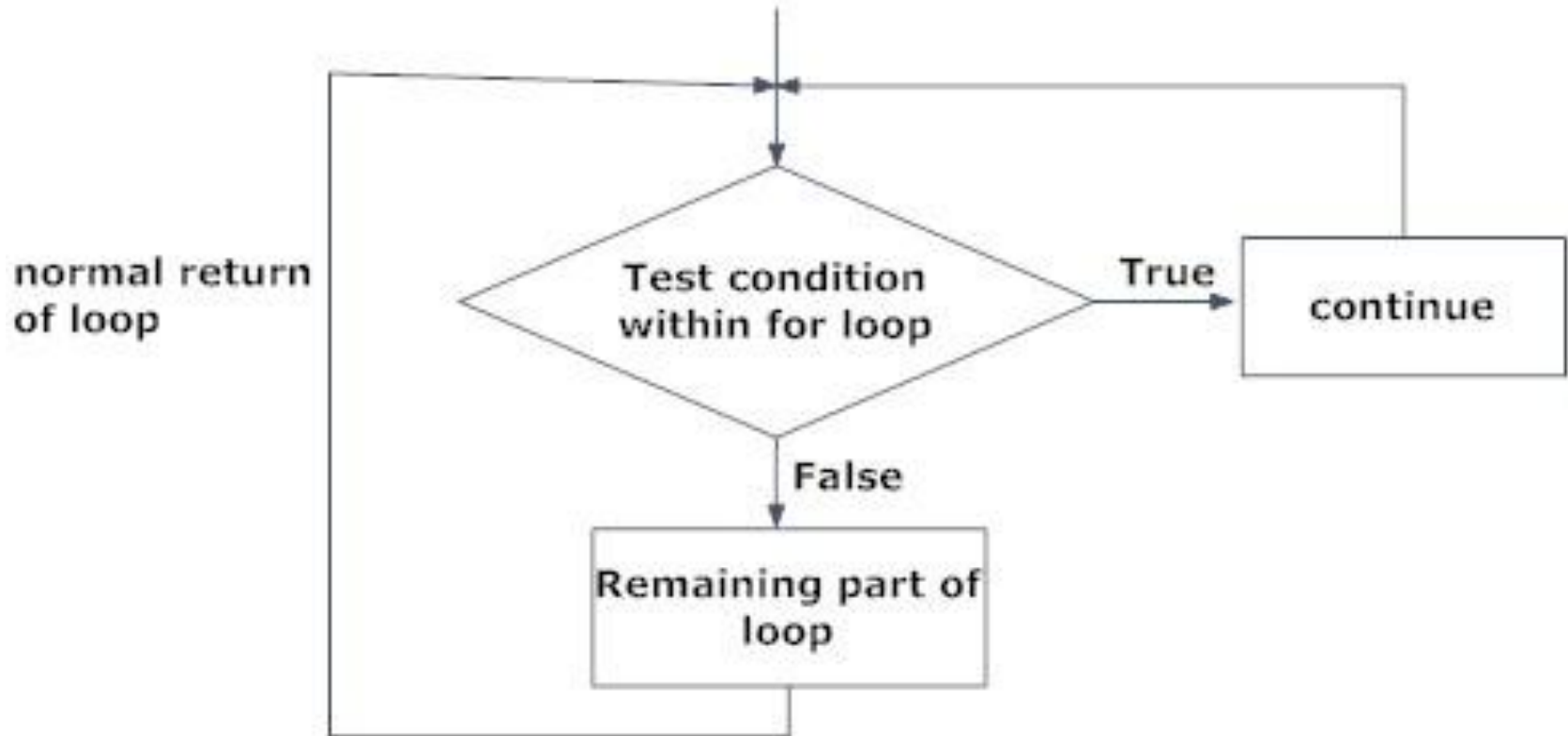
```
#include<stdio.h>
int main()
{
    int num, sum=0;
    int i,n;
    printf("Note: Enter Zero for break loop!\n");
    printf("Enter Number of inputs\n");
    scanf("%d",&n);
    for(i=1;i<=n;++i){
        printf("Enter num%d: ",i);
        scanf("%d",&num);
        if(num==0) {
            break;    /*this breaks loop if num == 0 */
            printf("Loop Broken\n");
        }
        sum=sum+num;
    }
    printf("Total is %d",sum);
    return 0;
}
```

continue statement

- The **Continue** statement like any other jump statements interrupts or changes the flow of control during the execution of a program.
- **Continue** is mostly used in loops. Rather than terminating the loop it stops the execution of the statements underneath and takes the control to the next iteration.
- Similar to a **break** statement, in case of nested loop, the **continue** passes the control to the next iteration of the inner loop where it is present and not to any of the outer loops.

Syntax:
continue;

continue statement




continue statement


- The continue statement skips the current iteration of the loop and continues with the next iteration.
- The continue statement is almost always used with the if...else statement.

How continue statement works?

```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```



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



Example 2: continue statement

LAB 4 QN 18: Program to illustrate the use of continue statement

```
#include<stdio.h>
int main()
{
    int x;
    for(x=1;x<=5;x++)
    {
        if(x==3)
            continue;
        printf("%d\t",x);
    }
    printf("\nFinished Loop:");
    return 0;
}
```

o/p: 1 2 4 5

LAB 4 QN 19 Program to calculate the sum of numbers (10 numbers max). If the user enters a negative number, it's not added to the result.

```
#include <stdio.h>
int main() {
    int i;
    double number, sum = 0.0;
    for (i = 1; i <= 10; ++i) {
        printf("Enter a n%d: ", i);
        scanf("%lf", &number);
        if (number < 0.0) {
            continue;
        }
        sum += number; // sum = sum + number;
    }
    printf("Sum = %.2lf", sum);
    return 0;
}
```

Return

- This jump statement is usually used at the end of a function to end or terminate it with or without a value.
- It takes the control from the calling function back to the main function(main function itself can also have a **return**).
- An important point to be taken into consideration is that **return** can only be used in functions that is declared with a **return** type such as *int*, *float*, *double*, *char*, etc.
- The functions declared with *void* type does not return any value. Also, the function returns the value that belongs to the same data type as it is declared.

Syntax:

return expression;

Return

Eg:

```
int findmax(int a,int b)
{
    if(a>b)
        return a;
    else
        return b;
}
```

Goto

- **goto** statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

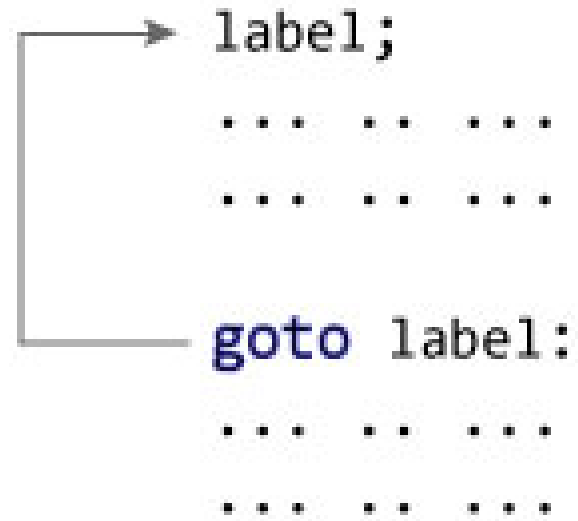
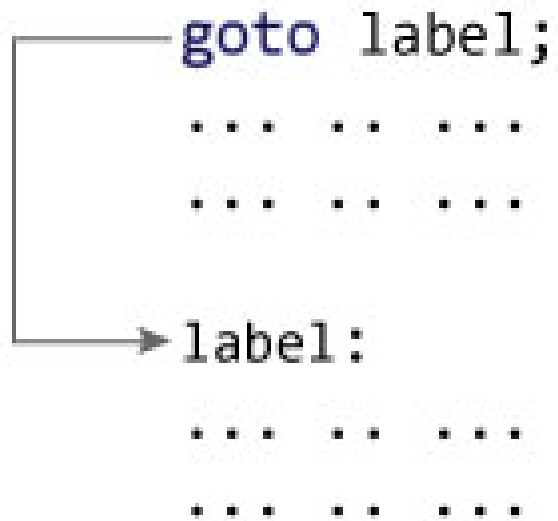
Syntax:

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

In this syntax, 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.

Goto



LAB 4 QN 20 : Program to print numbers 1 to 10 using goto statement (without using loop) and label.

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

```
int main()
```

```
{
```

```
    int x=1;
```

```
    loop1:
```

```
    printf("%d\t",x);
```

```
    x++;
```

```
    if(x<10)
```

```
        goto loop1;
```

```
    return 0;
```

```
}
```

o/p: 1 2 3 4 5 6 7 8 9

LAB 4 QN 21: Program that display the temperatures from 0 to 100 degree Celsius and their Fahrenheit equivalent.

```
#include <stdio.h>

int main()
{
    int c; float f;
    printf("Temperature in fahrenheit:");
    for(c=0;c<=25;c++)
    {
        f=(float)c *9/5+32;
        printf("%.2f\t",f);
    }
    return 0;
}
```

LAB 4 QN21 : Program to calculate sum of first 10 even numbers.

```
#include <stdio.h>
int main()
{
    int i,sum=0;
    printf("Sum of first 10 even number:\n");
    for(i=0;i<=10;i=i+2)
    {
        sum=sum+i;
    }
    printf("Sum=%d",sum);
    return 0;
}
```

LAB 4 QN 22: Program to find out sum of all number completely divisible by 5 among n number given by user.

```
#include <stdio.h>
int main()
{
    int n,i,sum=0;
    printf("Enter n:");
    scanf("%d",&n);
    for(i=5;i<=n;i=i+5)
    {
        sum=sum+i;
    }
    printf("Sum of number exactly divisible by 5=%d",sum);
    return 0;
}
```

LAB 4 QN 23: Program to determine the sum of the harmonic series :

($1+1/2+1/3+1/4+\dots+1/n$).

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

```
int main()
```

```
{
```

```
    int n,i;
```

```
    float sum=0;
```

```
    printf("Enter n:");
```

```
    scanf("%d",&n);
```

```
    for(i=1;i<=n;i=i+1)
```

```
    {
```

```
        sum=sum+1.0/i;
```

```
    }
```

```
    printf("Sum=%f",sum);
```

```
    return 0;
```

```
}
```


LAB 4 QN 24: Program to find the sum of the series $1+x^2+3x^2+4x^2+\dots+nx^2$

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

```
int main()
```

```
{
```

```
    int n,x,i,sum;
```

```
    printf("Enter value of x and n:");
```

```
    scanf("%d%d",&x,&n);
```

```
    sum=1+x*x;
```

```
    for(i=3;i<=n;i=i+1)
```

```
    {
```

```
        sum=sum+i*x*x;
```

```
    }
```

```
    printf("Sum=%d",sum);
```

```
    return 0;
```

```
}
```

LAB 4 QN 25:

Program to find X of the following series for the given value of a and N.

$$X = a - a^2/2 + a^3/3 - a^4/4 \dots \dots \dots \text{up to N}$$

```
#include<stdio.h>
#include<math.h>
int main()
{
    int a, n, i;
    float x = 0;
    printf("Enter values of a and n:\n");
    scanf("%d%d", &a, &n);
    for(i = 1; i <= n; i++)
    {
        if(i%2 == 0)
            x = x - pow(a, i) / i;
        else
            x = x + pow(a, i) / i;
    }
    printf("X = %f", x);
    return 0;
}
```

LAB 4 QN 26

Program to display all prime numbers less than 100 */

```
#include<stdio.h>
int main()
{
    int i, j, prime;
    printf("Prime numbers from 1 to 100 are:\n");
    for(i = 2; i < 100; i++)
    {
        prime = 1;
        for(j = 2; j < i; j++)
        {
            if(i % j == 0)
            {
                prime = 0;
                break;
            }
        }
        if(prime)
            printf("%d\t", i);
    }
}
```

LAB 4 QN 27

Program that calculates sum of the sequence $1/1! + 2/2! + 3/3! + \dots + n/n!$, where n is the number of input by the user */

```
#include<stdio.h>
int main()
{
    int n, i, j, fact;
    float sum = 0;
    printf("Enter n:");
    scanf("%d", &n);
    for(i = 1; i <= n; i++)
    {
        fact = 1;
        for(j = 1; j <= i; j++)
            fact = fact * j;
        sum = sum + (float)i / fact;
    }
    printf("Sum = %f", sum);
    return 0;
}
```

LAB 4 QN 28

Program to display sum of the following series up to n terms:

$$\text{Sum} = x - x^2 + x^3 - x^4 + \dots$$

```
#include<stdio.h>
#include<math.h>
int main()
{
    int x, n, sum = 0, i;
    printf("Enter values of x and n:\n");
    scanf("%d%d", &x, &n);
    for(i = 1; i <= n; i++)
    {
        if(i%2 == 0)
            sum = sum - pow(x, i);
        else
            sum = sum + pow(x, i);
    }
    printf("Sum = %d", sum);
    return 0;
}
```


LAB 4 QN 29

```
Execution time : 39.299 s
Program to print the following outputs using for loops
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5*/

#include<stdio.h>
int main()
{
    int i, j;
    for(i = 1; i <= 5; i++)
    {
        for(j = 1; j <= i; j++)
            printf("%d\t", i);
        printf("\n");
    }
    return 0;
}
```

LAB 4 QN 30

Write a program to display following

```
1
2   3
3   4   5
4   5   6   7
5   6   7   8   9
```

```
#include<stdio.h>
int main()
{
    int i,j,p;
    for(i=0;i<=4;i++)
    {
        for(j=0;j<=i;j++)
        {
            p=(i+j)+1;
            printf("%d\t",p);
        }
        printf("\n");
    }
    return 0;
}
```

LAB 4 QN 31

```
15          20          25          30          35          40          45
Program to display the following:
1
1  1
1  1  1
1  1  1  1
1  1  1  1  1

#include<stdio.h>
void main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1;j<=i;j++)
        {
            printf("%d\t",1);
        }
        printf("\n");
    }
    return 0;
}
```


LAB 4 QN 32

Write program to display following:

```

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

```
#include<stdio.h>
void main()
{
    int i,j,k,l,s=4;
    for(i=1;i<=4;i++)
    {
        for(k=1;k<=s;k++)
        {
            printf(" ");
        }
    }
}
```

```
for(j=i;j>=1;j--)
{
    printf("%d",j);
}
for(l=2;l<=i;l++)
{
    printf("%d",l);
}
printf("\n");
s--;
}
```

Which loop to Select?

Selection of a loop is always a tough task for a programmer, to select a loop do the following steps:

- Analyze the problem and check whether it requires a pre-test or a post-test loop.
- If pre-test is required, use a while or for a loop.
- If post-test is required, use a do-while loop.

Finished

Chapter 5