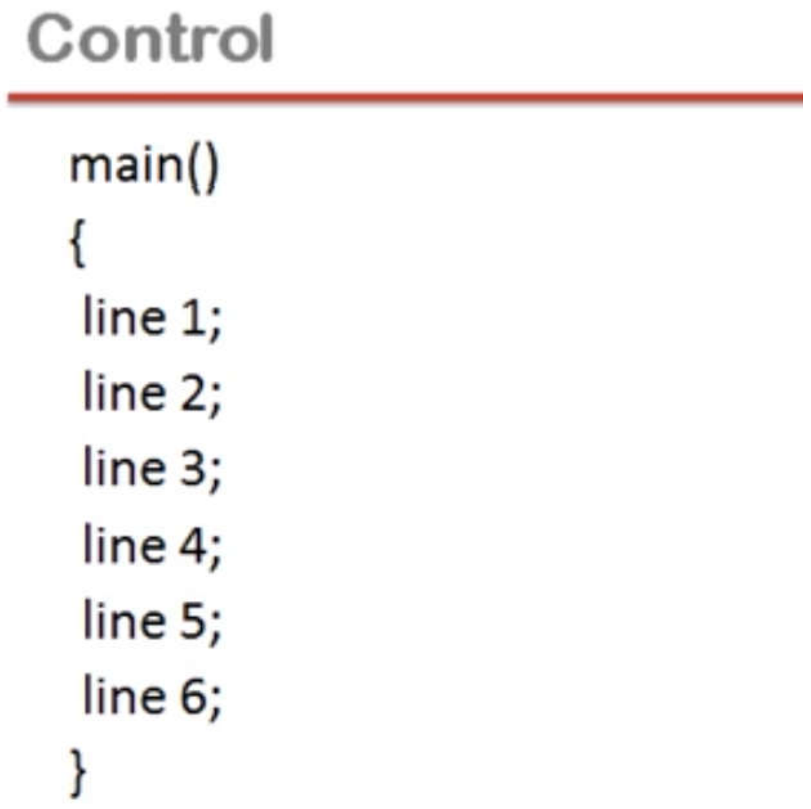
Unit – III

- Conditional statements (if statement, if-else statement, ternary statement or ternary operator, nested if-else statement, switch statement),
- Difference between performance of if else and switch, Advantages of if else and switch over each other
- Loops – ‘for’ loops, ‘while’ loops, ‘do while’ loops, entry control and exit control, break and continue, nested loops

Control Statements

In C programs, statements are executed sequentially in the order in which they appear in the program.



Control

```
main()
{
 line 1;
 line 2;
 line 3;
 line 4;
 line 5;
 line 6;
}
```

Control Statements

- But sometimes we may want to use a condition for executing only a part of program.
- Also many situations arise where we may want to execute some statements several times.
- Control statements enable us to specify the order in which the various instructions in the program are to be executed.
- This determines the flow of control. Control statements define how the control is transferred to other parts of the program.

Decision Control Instruction

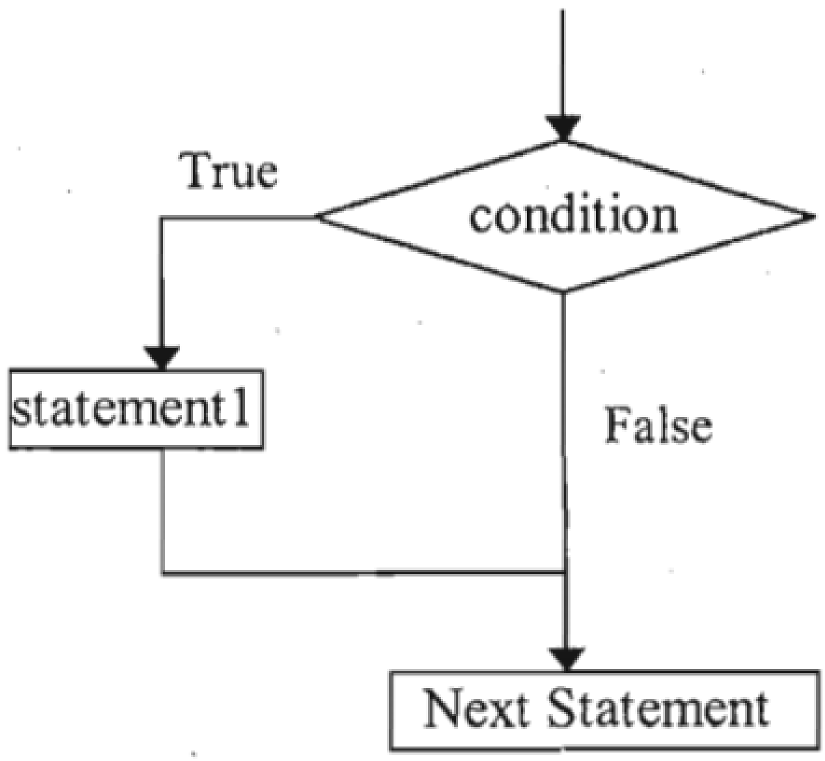
- If
- If-else
- Conditional Operator (?:)

if

```
if(condition)
    statement1;
```

```
if(condition)
     {
        statement;
       ---------------
     }
```

Flow chart of if control statement



Flow chart of if control statement

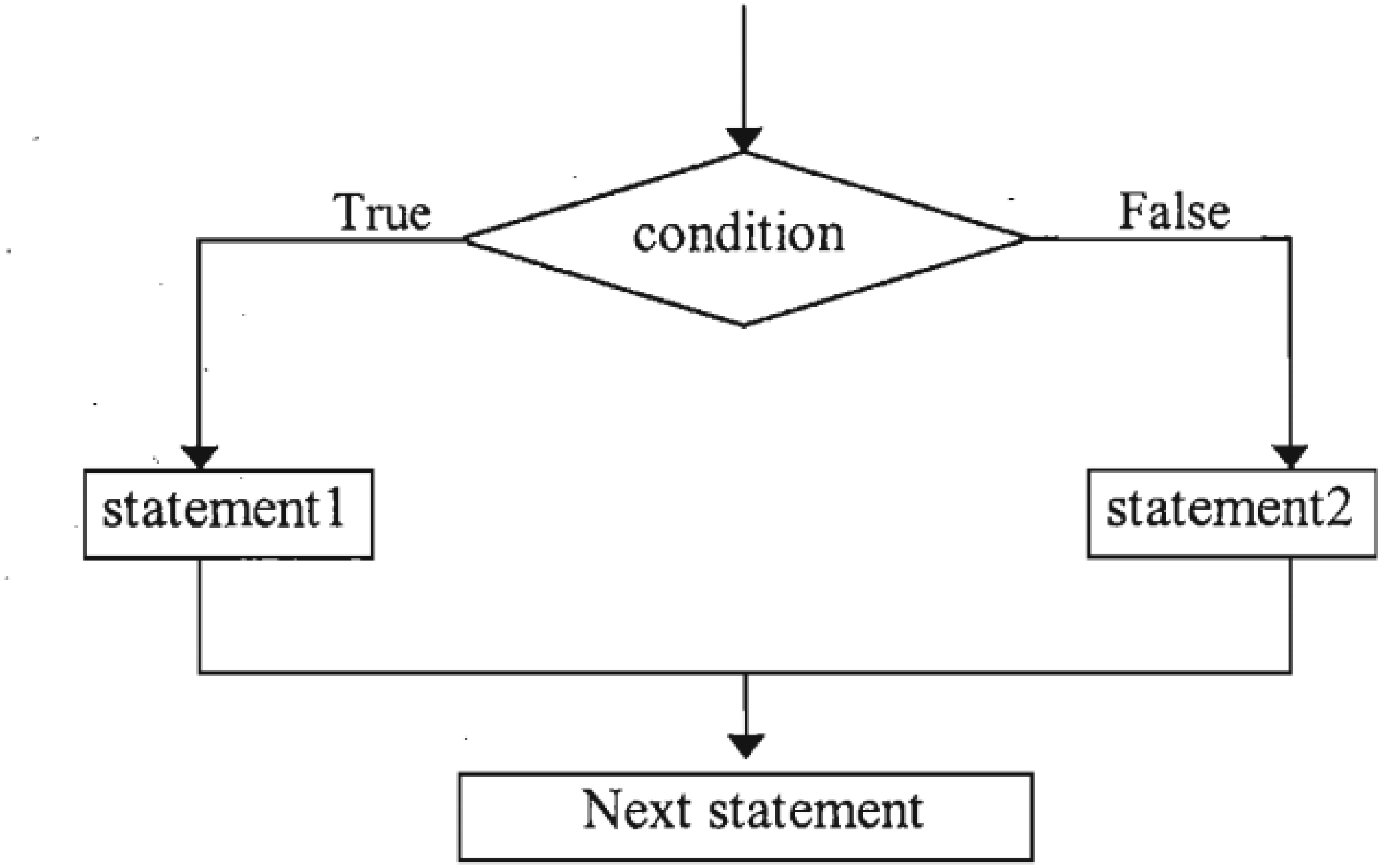
if...else

- This is a bi-directional conditional control statement. This statement is used to test a condition and take one of the two possible actions.
- If the condition is true then a single statement or a block of statements is executed, otherwise another single statement or a block of statements is executed.
- Recall that in C, any **nonzero** value is regarded as **true** while **zero** regarded as **false**.

If else

```
if( condition)
{
 line 1;
 line 2;
 ...
}
else
{
 line 1;
 line 2;
 ...
}
```

Flow chart of if....else control statement



/ * Program to print a message if negative number is entered* /

```
#include<stdio.h>
int main( )
{
int num;
printf( "Enter a number ");
scanf("%d",&num);
if (num<0)
printf("Number entered is negative");
printf ("Value of num is %d", num);
return 0;
}
```

Output

1st **run:**

```
Enter a number   -6
Number entered is negative
Value of num is -6
```

2nd **run:**·

```
Enter a number    8
Value of num is 8
```

Program to print the larger and smaller of the two numbers

```
#include<stdio.h>
int main( )
{
int a,b;
printf("Enter the first number: ");
scanf("%d",&a);
printf ("Enter the Second Number:");
scanf ("%d",&b);
if (a>b)
printf("larger number is: %d”,a);

else
printf("larger number is: %d”,b);
return 0;
}
```

If else

```
if( condition)
{
 line 1;
 line 2;
 ...
}
else
{
 line 1;
 line 2;
 ...
}
```

```
Condition ? Statement : Statement;
```

Nesting of if...else

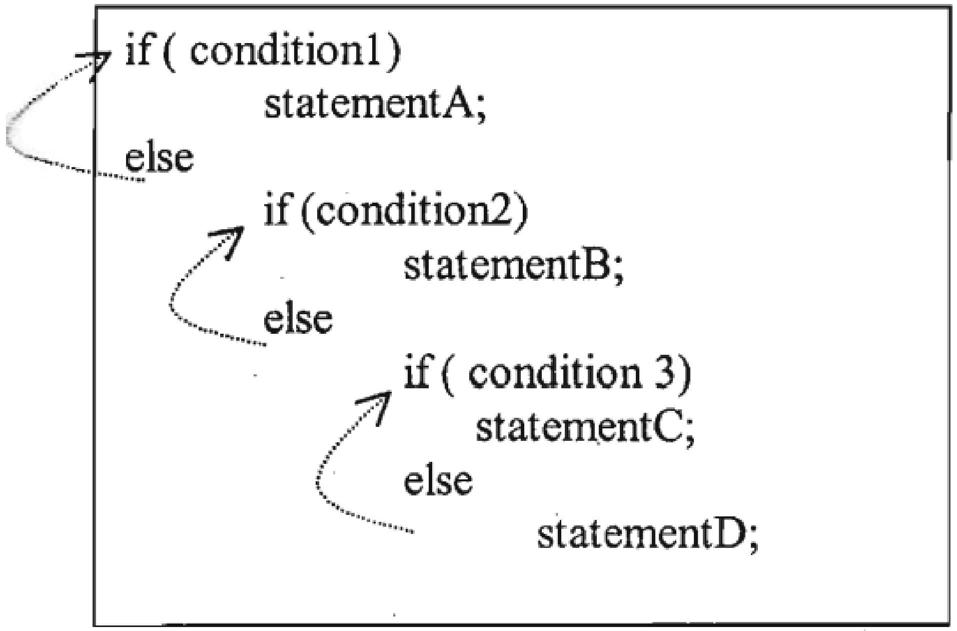
- We can have another if. .. else statement in the if block or the else block. This is called nesting of if... else statements.

```
if (condition 1)
{
        if (condition 2)
        statementA1;

        else
        statementA2;
}
else
{
        if(condition 3)
        statementB1;

        else
        statementB2
}
```

else if Ladder

- This is a type of nesting in which there is an if. .. else statement in every else part except the last else part.

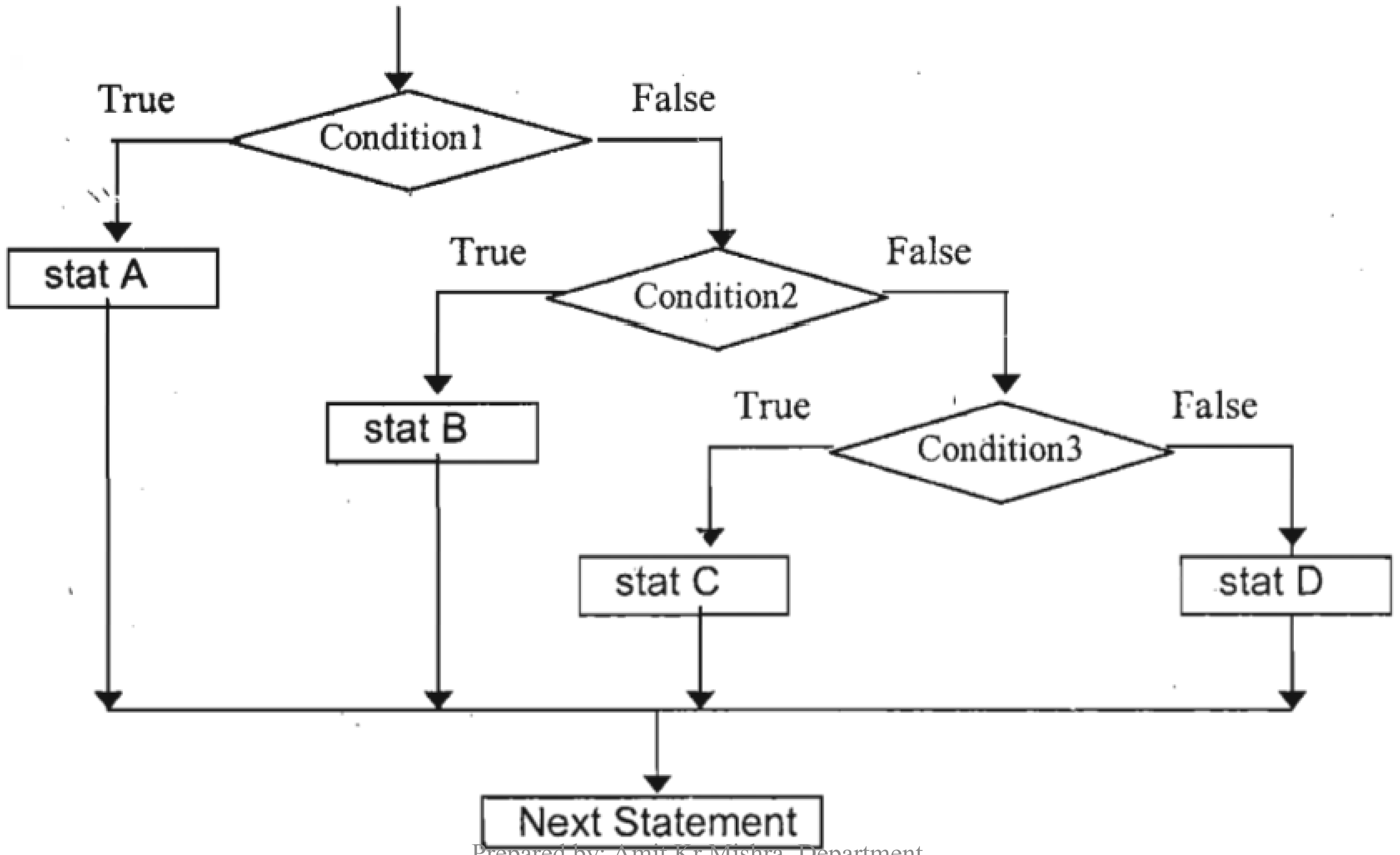
This type of nesting is frequently used in programs and is also known as else if ladder.



```
if ( condition1)
        statementA;
else
        if (condition2)
                statementB;
        else
                if ( condition 3)
                    statementC;
                else
                        statementD;
```

```
if ( condition1)
        statementA;
else if (condition2)
        statementB;
else if ( condition 3)
        statementC;
else
        statementD;
```

Flow chart of else.. .if ladder



/* Write a program to display the examination result.*/

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

    if ( marks >= 75)
    printf("\n DISTINCTION");

    else if ( marks >= 60)
    printf("\n FIRST DIVISION");

    else if ( marks >= 50)
    printf("\n SECOND DIVISION");

    else if ( marks >= 40)
    printf("\n THIRD DIVISION");

    else
    printf("\n FAIL");

return 0;
}
```

/* Write a program to display the examination result.*/ (without else)

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

        if ( marks >= 75)
        printf("\n DISTINCTION");

        if ( marks >= 60)
        printf("\n FIRST DIVISION");

        if ( marks >= 50)
        printf("\n SECOND DIVISION");

        if ( marks >= 40)
        printf("\n THIRD DIVISION");

         if ( marks >= 0)
        printf("\n FAIL");

return 0;
}
```

```
/* Write a program to display the examination result.*/
    #include <stdio.h>
    main()
    {
    int marks;
    printf("\n Enter the marks obtained: ");
    scanf("%d", &marks);
            if ( marks >= 75)
            printf("\n DISTINCTION");

            if ( marks >= 60 && marks <75)
            printf("\n FIRST DIVISION");

            if ( marks >= 50 && marks < 60)
            printf("\n SECOND DIVISION");

            if ( marks >= 40 && marks < 50)
            printf("\n THIRD DIVISION");

            if ( marks >= 0 && marks < 40)
            printf("\n FAIL");

    return 0;
    }
```

- In else.. .if ladder   each condition is checked, and when a condition is found to be true, the statements corresponding to that are executed, and the control comes out of the nested structure without checking remaining conditions.
- If none of conditions is true then the last else part is executed.
- In If…else ladder whenever a condition is found true other conditions are not checked.
- In if…. all the conditions will always be checked wasting a lot of time, and moreover the conditions here are more lengthy.

Iterative Control Instructions

- while
- do while
- for

Loops

Loops are used when we want to execute a part of the program or a block of statements several times.

- For example, suppose we want to print “**I am the best**" 10 times.
- One way to get the desired output is- we write 10 printf statements, which is not preferable.
- Other way out is - use loop.
- Using loop we can write one loop statement and only one printf Statement, and this approach is definitely better the first one.
- **With the help of loop we can execute a part of the program repeatedly till some condition is true.**
- There are three loop statements in C-
  - **While**
  - **do while**
  - **for**

while loop

The while statement can be written as:

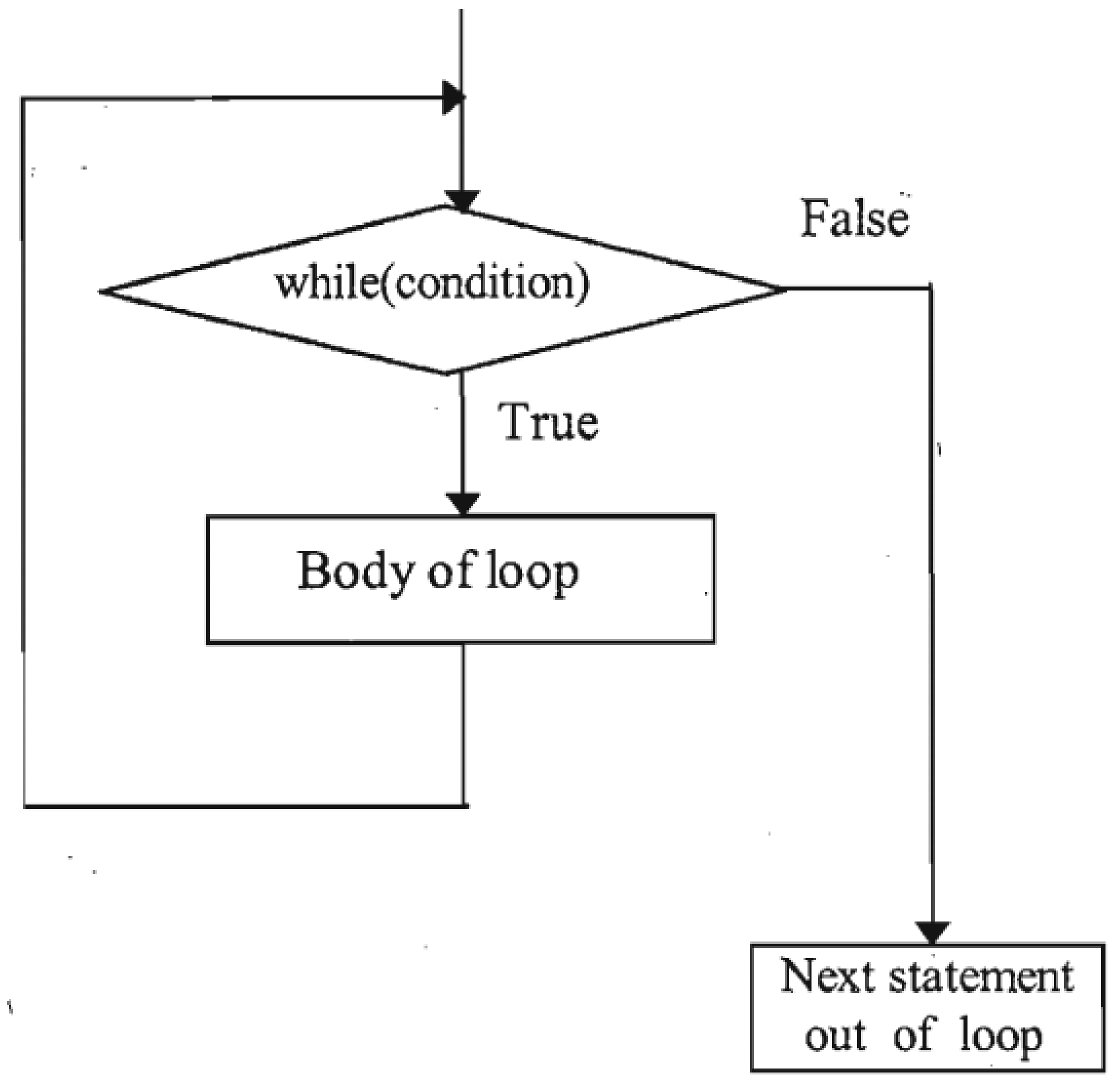
```
while(condition)
statement;
```

or

```
while(condition)
{
statement1;
statement2;
----------------
}
```

Like if-else statement here also we can have either a single statement or a block of statements, and here it is known as the body of the loop.

Flow chart of while loop

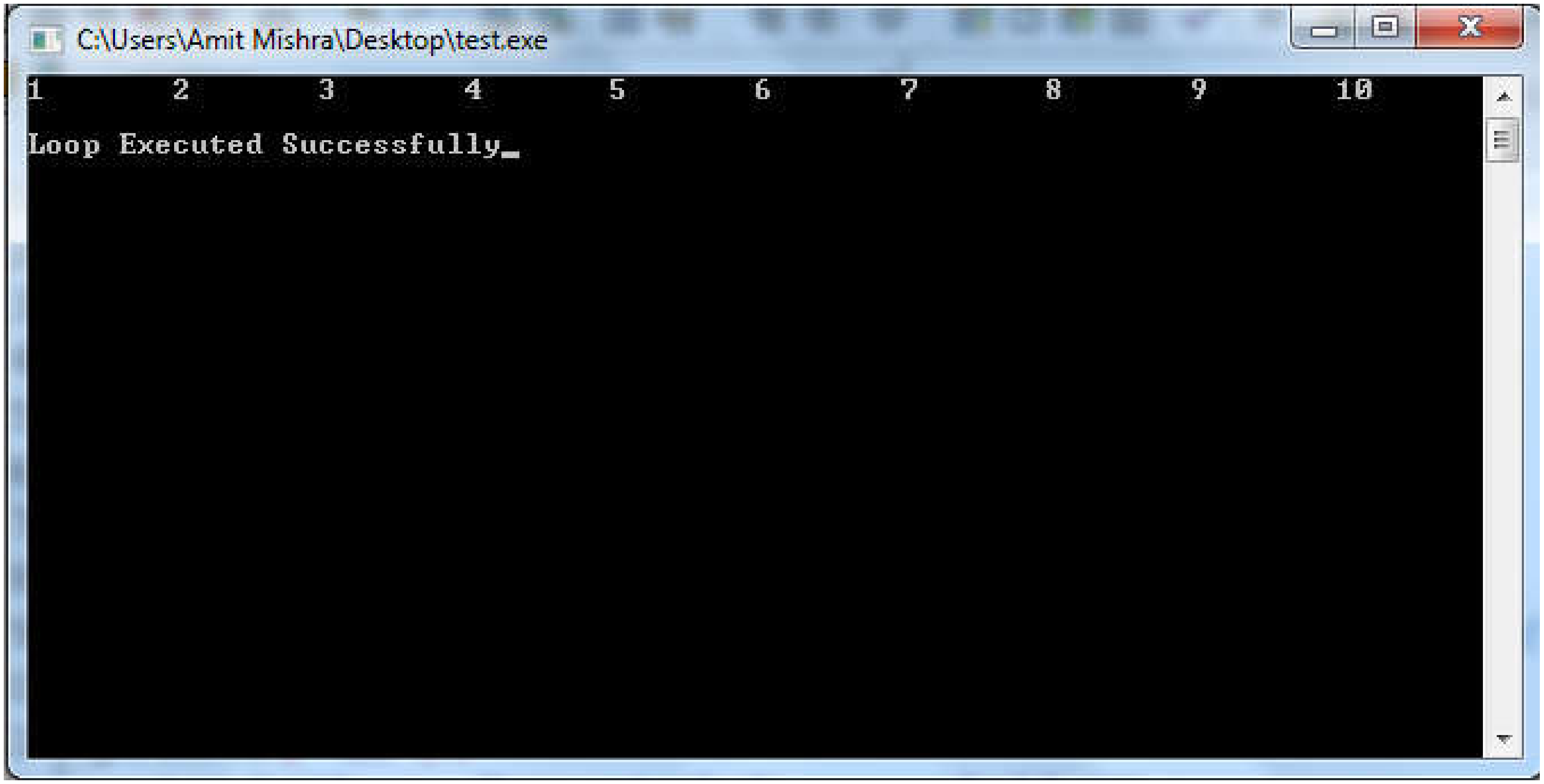


let's see how this -loop works

- First the condition is evaluated; if it is true then the statements in the body of loop are executed.
- After the execution, again the condition is checked and if it is found to be true then again the statements
  in the body of loop are executed. This means that these statements are executed continuously till the
  condition is true and when it becomes false, the loop terminates and the control comes out of the loop.
- Each execution of the loop body is known as iteration.

//Program to print the numbers from 1 to 10 using while loop

```
#include<stdio.h>
int main( )
{
int i=1;
    while(i<=10)
    {
    printf("%d\t",i) ;
    i++;  /*Statement that changes the value of condition*/
    }
printf("\n");
printf("Loop Executed Successfully");
getch();
return 0;
}
```



C:\Users\Amit Mishra\Desktop\test.exe

```
1       2       3       4       5       6       7       8       9       10

Loop Executed Successfully_
```

do...while loop

The 'do...while' statement is also used for looping. The body of this loop may contain a single statement or a block of statements.

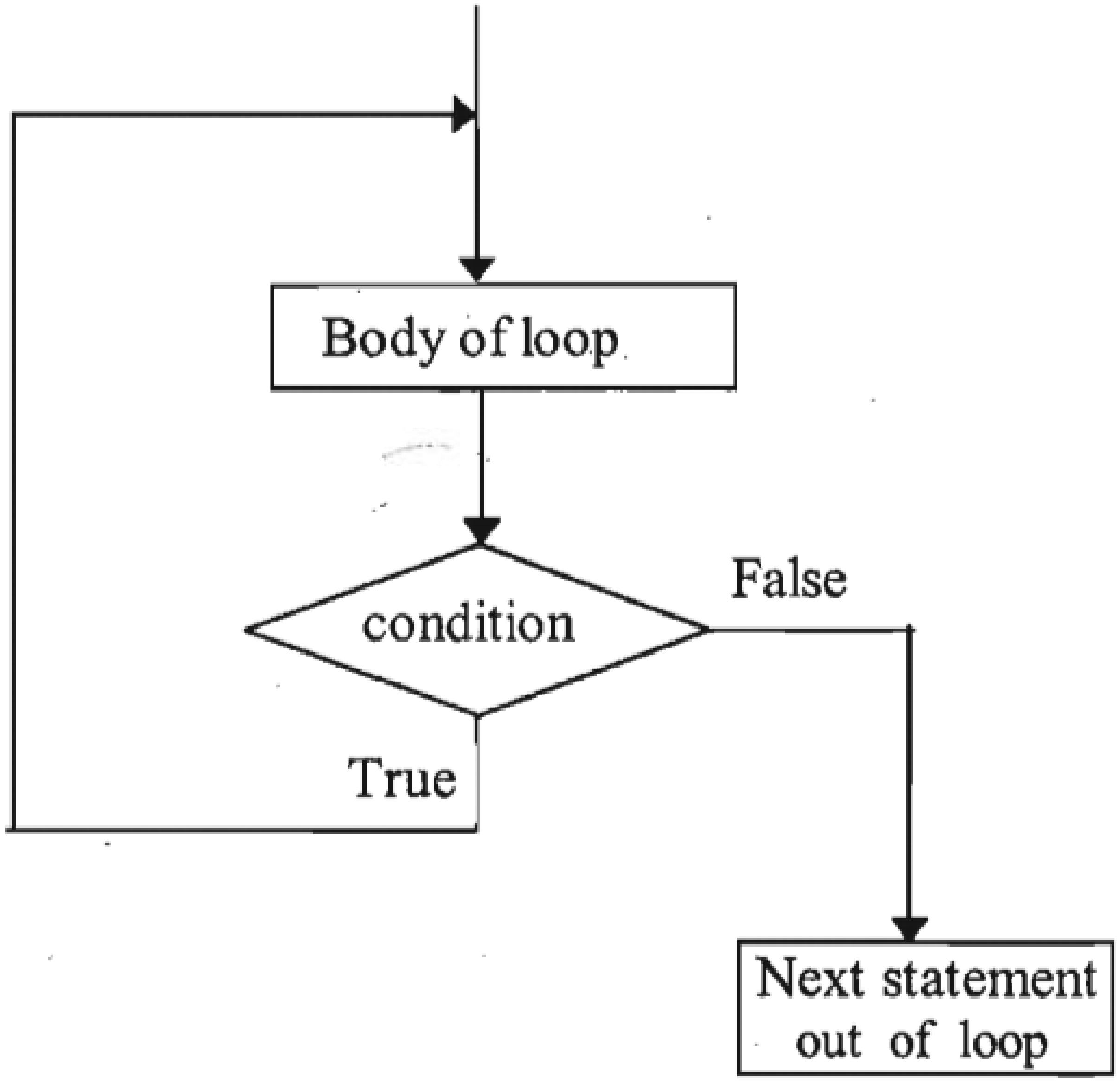
The syntax for writing this loop is:

```
do
    statement;
while(condition) ;
```

Or

```
do
{
statement1;
statement2;
--------------
}while(condition) ;
```

Flow chart of do … while loop

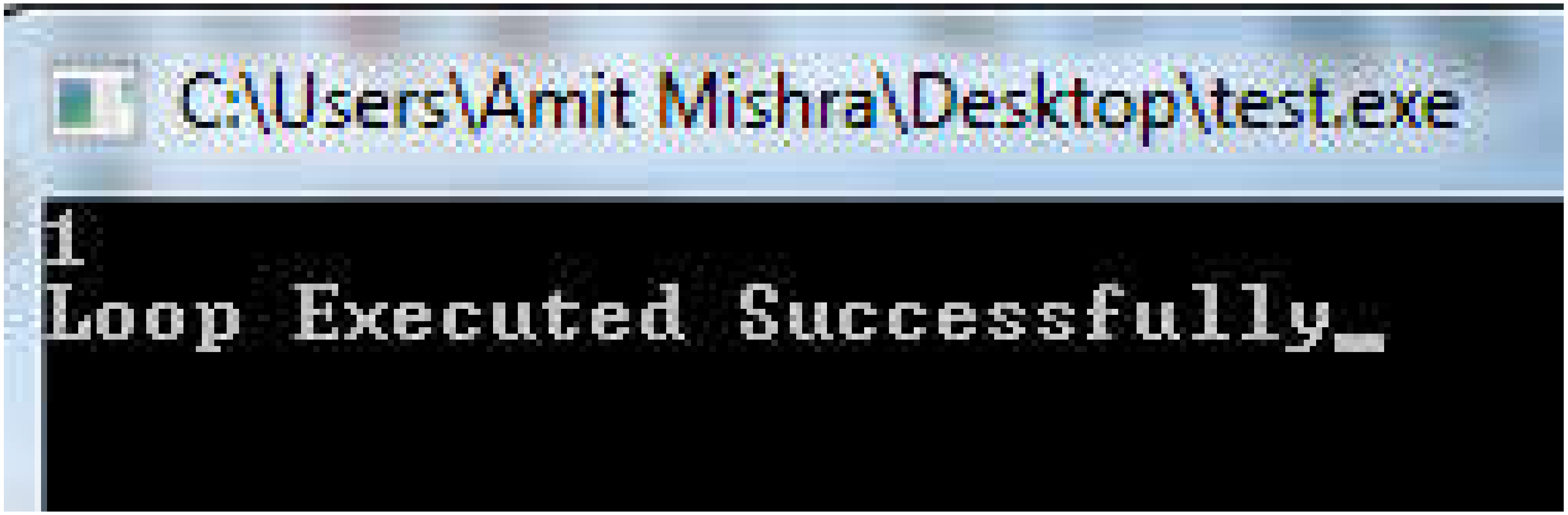


- Here firstly the statements inside loop body are executed and then the condition is evaluated.
- If the condition is true, then again the loop body is executed and this process continues until the condition becomes false.
- Note that unlike while loop, here a semicolon is placed after the condition.

  In a 'while' loop, first the condition is evaluated and then the statements are executed
- whereas in a do…while loop, first the statements are executed and then the condition is evaluated.
- So if initially the condition is false the while loop will not execute at all, whereas the do while loop will always execute at least once.

//Program to print the numbers from 1 to 10 using do….while loop

```
#include<stdio.h>
int main( )
{
int i=1;
    do
    {
    printf("%d\t",i) ;
    i=i+1;  /*Statement that changes the value of condition*/
    }while(i>=10);
printf("\n");
printf("Loop Executed Successfully");
getch();
return 0;
}
```


C:\Users\Amit Mishra\Desktop\test.exe

```
1
Loop Executed Successfully_
```

for loop

- The 'for' statement is very useful while programming in ‘C’ language.
- It has three expressions , and
- Semicolons are used for separating these expressions.

The loop body can be a single statement or block of statement. The 'for' statement can be written as-

```
for(expression1; expression2; expression3)
{
Statement 1;
Statement 2;
----------------
}
```

or

```
for(expressionl;expression2;expression3)
statement;
```

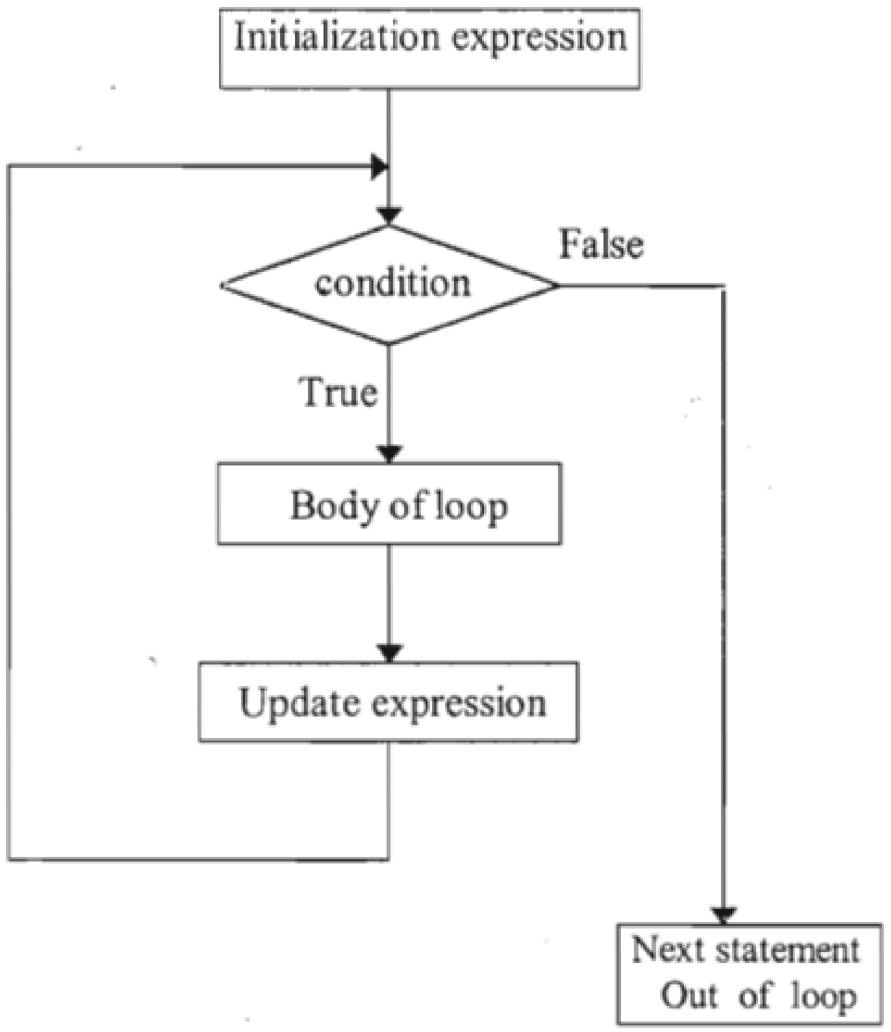
Expression 1 is an initialization expression,
Expression2 is a test expression or condition and
Expression3 is an update expression.

expression1 is executed only once when the loop starts and is used to initialize the loop variables. This expression is generally an assignment expression.

expression2 is a condition and is tested before each iteration of the loop. This condition generally uses relational and logical operators.

expression3 is an update expression and is executed each time after the body of the loop is executed.

Flow chart of for loop



/* Program to print the numbers from 1 to 10 using for loop*/

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

The work done by the for loop can be performed by writing a while loop as :

```
expression 1;
    while(expression 2)
    {
    statement;
    --------------
    --------------
    expression 3;
    }
```

Although the task of while and for loops is same, the for loop is generally used when the number of iterations are known in advance and while loop is used where number of iterations are not known.

/* Program to print the numbers from 1 to 10 using for loop*/

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

/*Program to print· numbers in reverse order with a difference of 2*/

```
#include<stdio.h>
int main()
{
int k;
    for(k=10;k>=2;k=k-2)
    printf("%d \t",k);
printf("\n");
getch();
return 0;
}
```

k=k-2 can be written as k-=2

We can also have any number of expressions separated by commas.
For example, we may want to initialize more than one variable or take more than one variable as loop variable.

/ * Program to print numbers using for loop* /

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

Nesting of Loops

When a loop is written inside the body of another loop, then it is known as nesting of loops.

- Any type of loop can be nested inside any other type of loop.
- For example a for loop may be nested inside another for loop or inside a while or do while loop.
- Similarly while and do while loops can be nested.

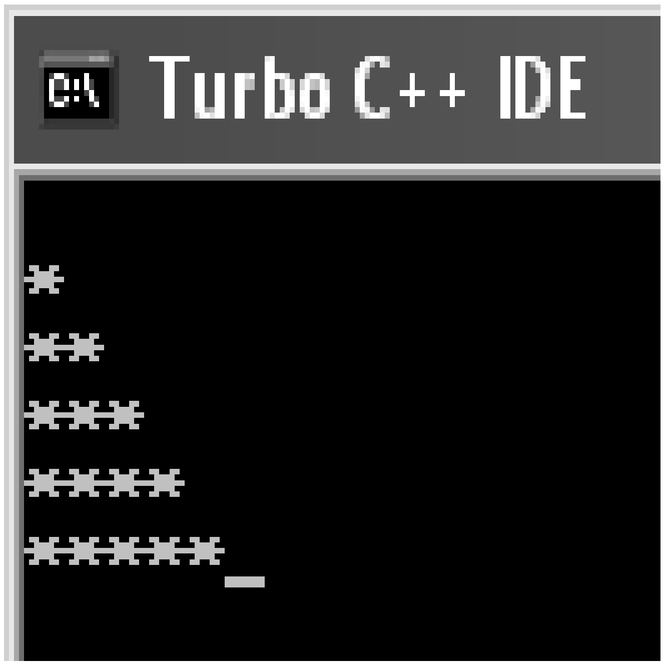
/* Program to understand nesting in for loop*/

```
#include<stdio.h>
int main()
{
int i,j;
    for(i=1;i<=3;i++)        //outer loop
    {
    printf("i=%d\n",i);
            for(j=1;j<=3;  j++)           //inner loop
            {
            printf ("j=%d\t",j);
            printf("\t");
            }
    printf("\n");
    }
return 0;
}
```

write a program for printing the pattern as follows:

```
*
**
***
****
*****
```

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



```

*
**
***
****
*****_
```

break statement

break statement is used inside, loops and switch statements. Sometimes it becomes necessary to come out of the loop even before the loop condition becomes false.

In such a situation, break statement is used to terminate the loop. This statement causes an immediate exit from that loop.

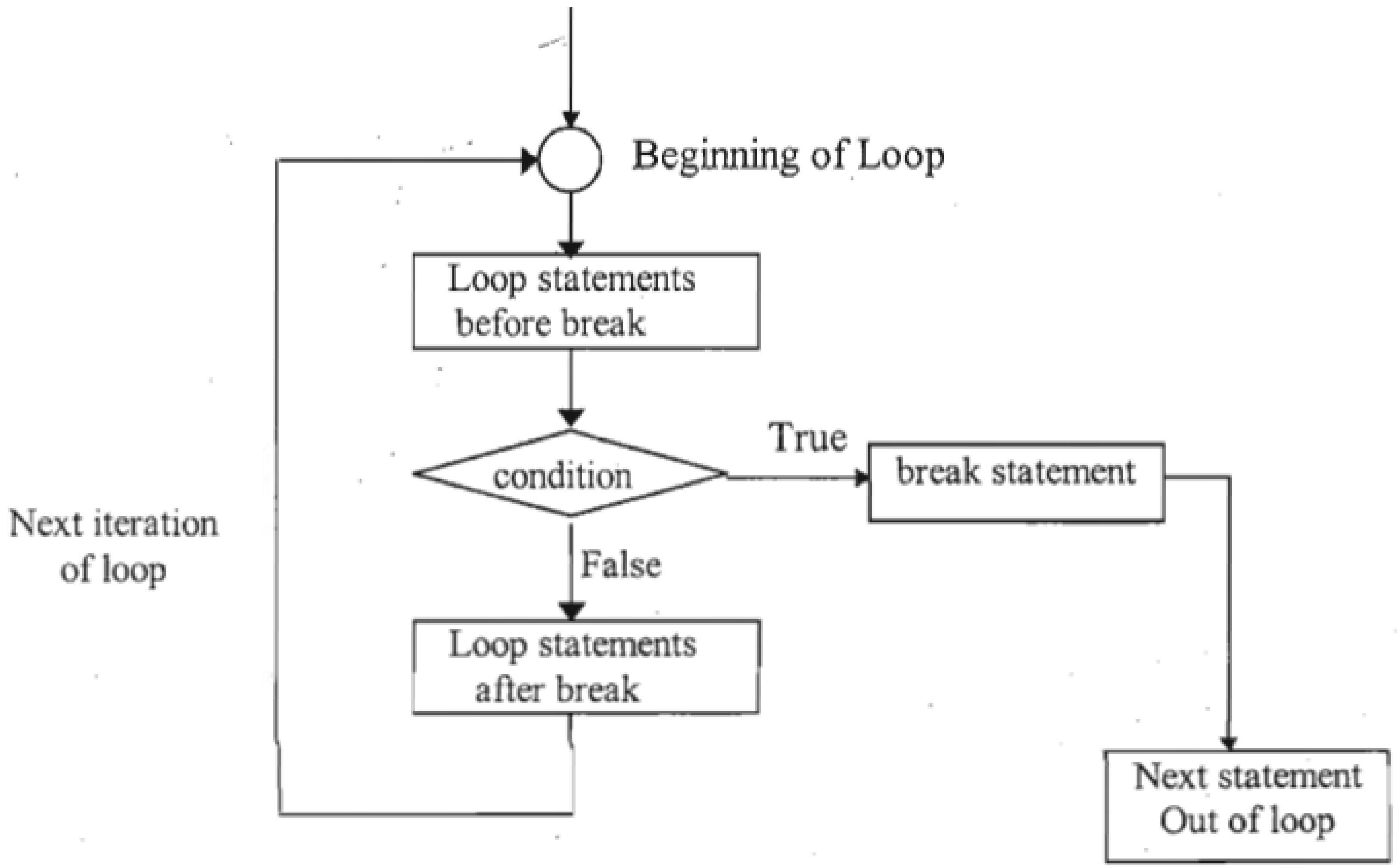
It can be written as

break;

break;

- When break statement is encountered, loop is terminated and the control is transferred to the statement immediately after the loop.
- The break statement is generally written along with a condition.
- If break is written inside a nested loop structure then it causes exit from the innermost loop.

break (control statement)



Beginning of Loop
Loop statements before break
condition
True
break statement
False
Loop statements after break
Next iteration of loop
Next statement Out of loop

Program to understand the use of break

```
#include<stdio.h>
int main()
{
int n;
    for(n=1;n<=5;n++)
    {
            if (n==3)
            {
            printf ("I understand the use of break\n");
            break;
            }
    printf ("Nulmber %d\n", n);
    }
printf("Out of for loop \n ");
return 0;
}
```

continue statement

- The continue statement is used when we want to go to the next iteration of the loop after skipping some statements of the loop.

- This continue statement can be written simply as

```
continue;
```

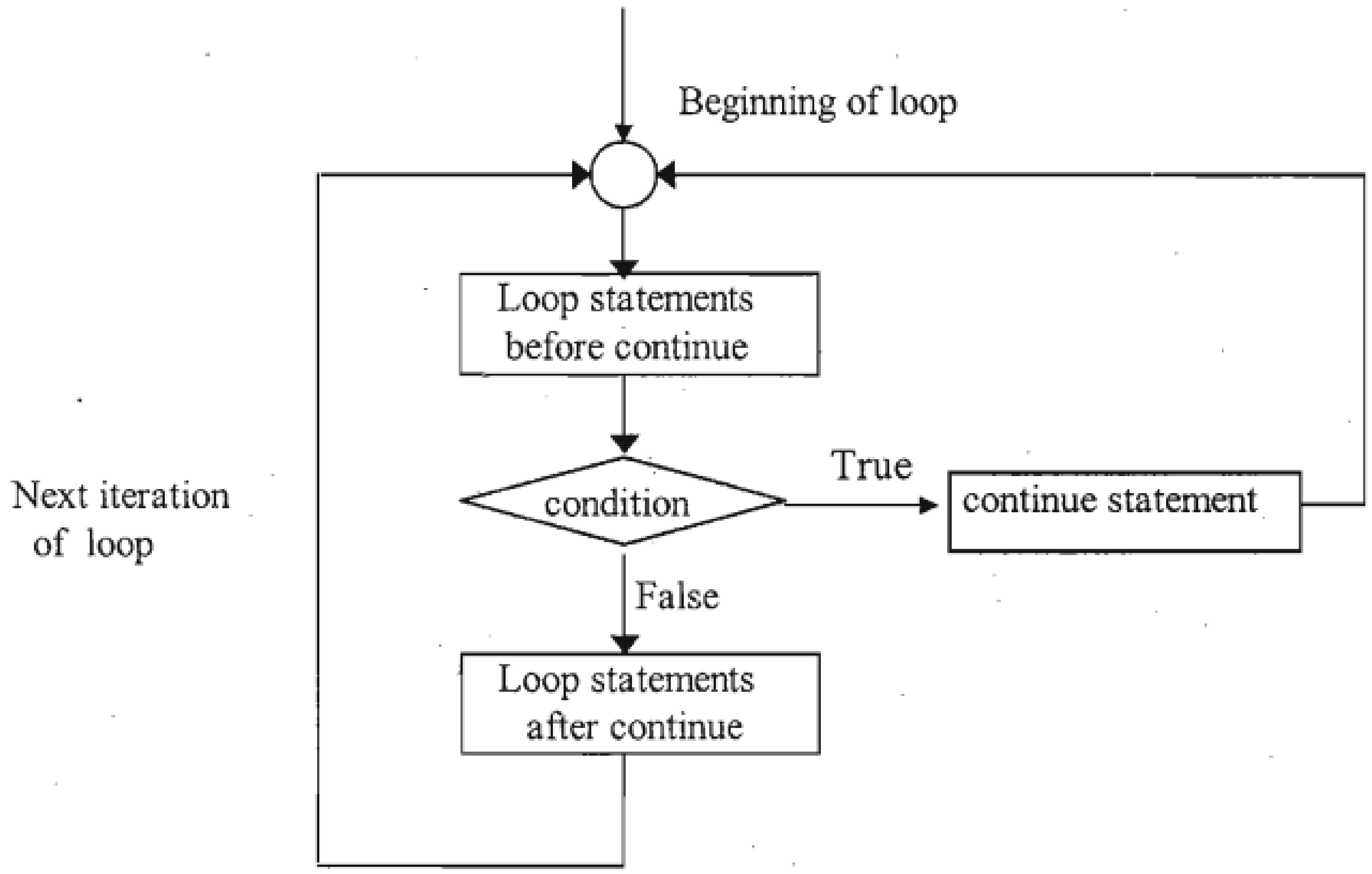
- It is generally used with a condition.
- When continue statement is encountered all the remaining statements (statements after continue) in the current iteration are not executed and the loop continues with the next iteration.

The difference between break and continue

The difference between break and continue is that when break is encountered the loop terminates and the control is transferred to the next statement following the loop.

When a continue statement is encountered the loop is not terminated and the control is transferred to the beginning of the loop.

continue (control statement)



Program to understand the use of continue statement

```
#include<stdio.h>
int main()
{
int n;
    for(n=1;n<=5;n++)
    {
            if (n==3)
            {
            printf (“I understand the use of continue \n");
            continue;
            }
    printf (“This is round:  %d\n", n);
    }
printf("Out of for loop \n ");
return 0;
}
```

goto

This is an unconditional control statement that transfers the flow of control to another part of the program.

The goto statement can be used as

```
goto label;
----------
----------
    label:
    statement;
----------
----------
```

Here label is any valid C identifier and it is followed by a .colon.

Program to print whether the number is even or odd

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

        else
        goto odd;

even:
printf ("Number is even");
goto end;

odd:
printf ("Number is odd");
goto end;

end:
printf("\n ***** END*****") ;
return 0;
}
```

- The use of 'goto' should be avoided, as it is difficult to understand where the control is being transferred.
- We can always perform all our jobs without using goto, and the use of goto is not preferred in structured programming.

switch

This is a multi-directional conditional control statement. Sometimes there is a need in program to make choice among number of alternatives. For making this choice, we use the switch statement. This can be written as:

```
switch(expression)
{
case constant1 :
statement

case constant2 :
statement
---------
---------
case constantN :
statement

default :
Statement
}
```

- Here switch, case and default are keywords.
- The "expression" following the switch keyword can be any C expression such as an integer value.
- Each case can be followed by any number of statements. The statements under case can be any valid ‘C’ statements.
- If the value of expression matches with any case constant, then all statements under that particular case are executed.
- If none of the case matches with the value of the expression then the block of statements under default is executed. 'default' is optional, if it is not present and no case matches then no action takes place.

Program to understand the switch control statement .

```c
#include <stdio.h>
int main()
{
int choice;
printf("Enter your choice ");
scanf("%d",choice);

        switch(choice)
        {
        case 1:
        printf("First\n");

        case 2:
        printf("Second\n");

        case 3:
        printf("Third\n");

        default:
        printf ("Wrong choice\n");
        }
getch();
return 0;
}
```

Output:

```
    Enter your choice : 2
    Second
    Third
    Wrong choice
```

Here value of choice matches with second case so all the statements after case 2 are executed sequentially. This is known as falling through cases.

Suppose we don't want the control to fall through the statements of all the cases under the matching case, then we can use break statement.

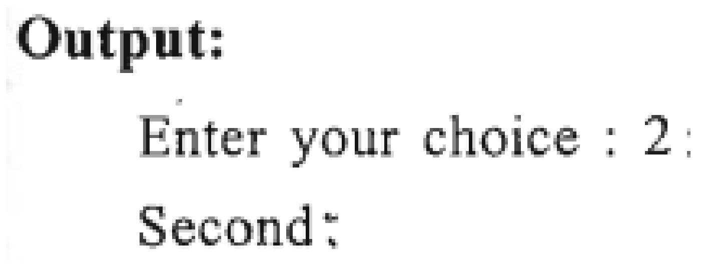
If a break statement is encountered inside a switch, then all the statements following break are not executed and the control jumps out of the switch.

So the use of break inside switch is optional, it may or may not be used depending on the requirement and logic of the program.

The use of continue statement is not valid inside switch.

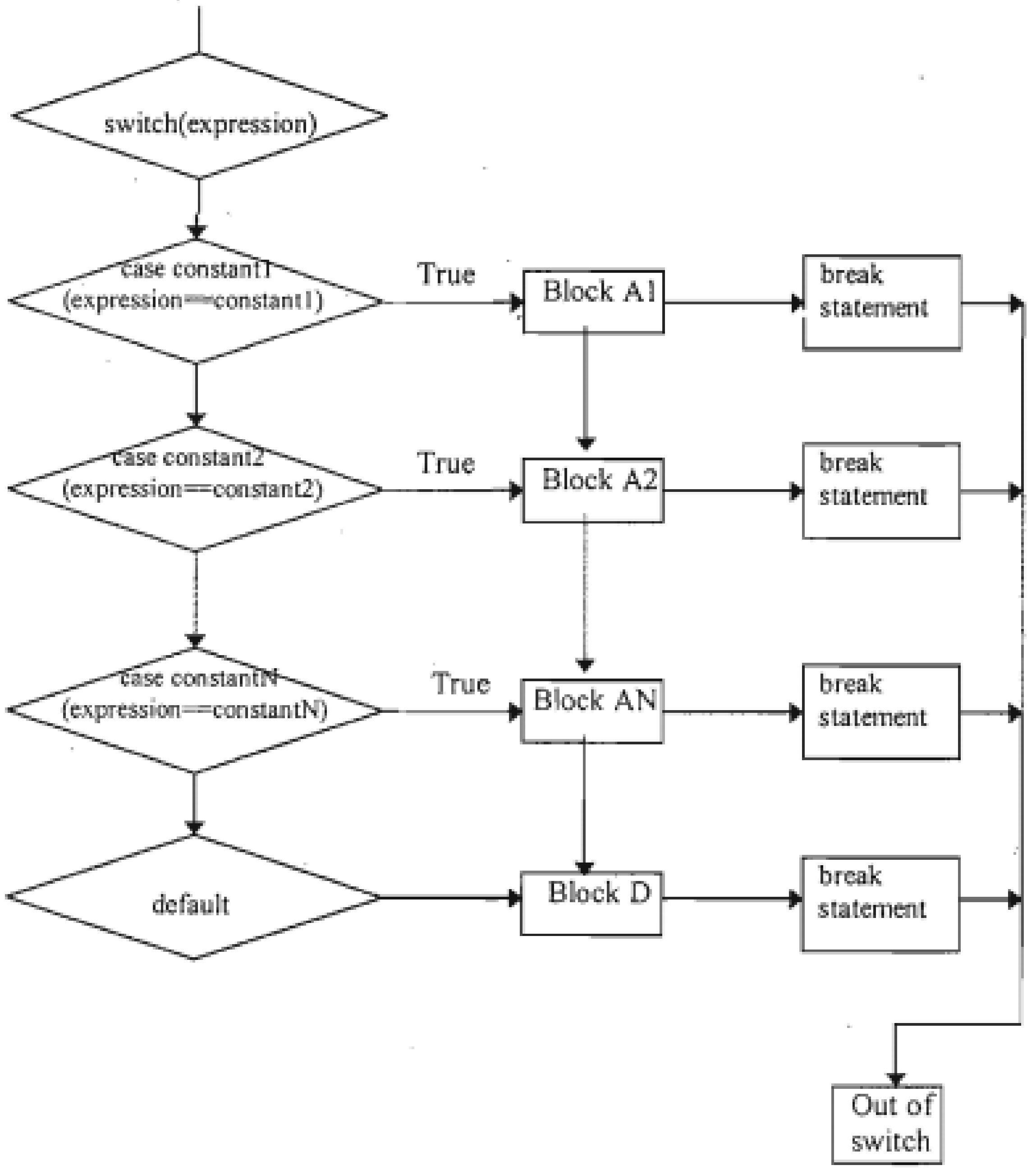
Program to understand the switch with break statement.

```c
#include <stdio.h>
int main()
{
int choice;
printf ("Enter your choice ") ;
scanf("%d",&choice);
switch(choice)
{
case 1:
        printf("First\n");
break;
case 2:
        printf("Second\n");
break;
case 3:
        printf("Third\n");
break;
default:
        printf("Wrong choice\n" );
}
return 0;
}
```



Output:

```
Enter your choice : 2
Second
```


switch(expression)
case constant1 (expression==constant1)
True
Block A1
break statement
case constant2 (expression==constant2)
True
Block A2
break statement
case constantN (expression==constantN)
True
Block AN
break statement
default
Block D
break statement
Out of switch

END OF UNIT III