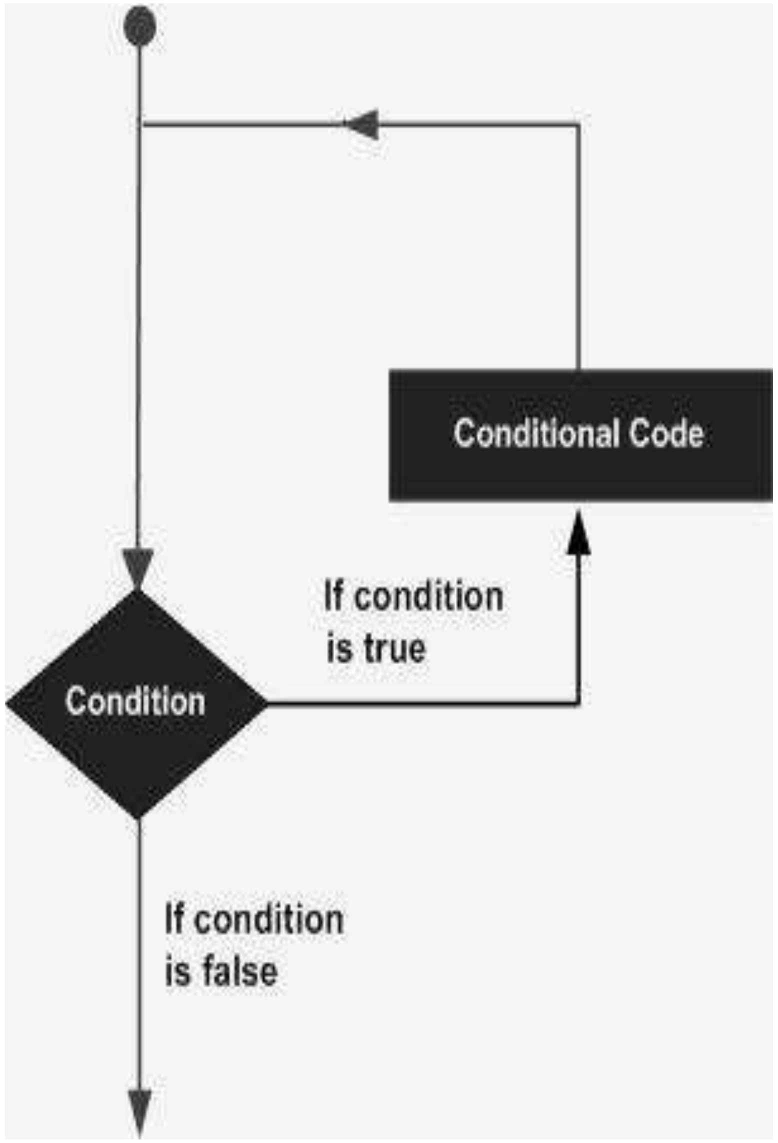
JAVA LOOPS

CONTENTS

1. Loop Statements
2. Parts of a loop
3. Types of Loops
   - While Loop
   - For Loop
   - Do-while Loop
4. Nested Loops
5. Exercises

LOOP STATEMENTS

The loop statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. Following is the general from of a loop statement in most of the programming languages:



PARTS OF A LOOP

- **Initialization Expression(s)** initialize(s) the loop
  variables in the beginning of the loop.
- **Test Expression** decides whether the loop will be executed (if test expression is true) or not (if test expression is false).
- **Update Expression(s)** update(s) the values of loop variables after every iteration of the loop.
- **The Body-of-the-Loop** contains statements to be executed repeatedly.

TYPES OF LOOPS

C++ programming language provides following types of loop to handle looping requirements:

Loop Type	Description
while loop	Repeats a statement or group of statements until a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

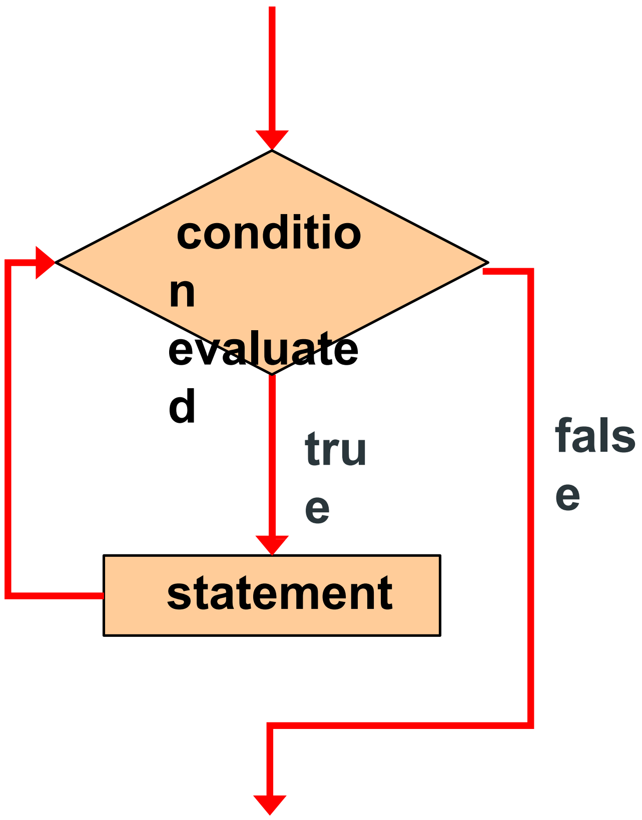
WHILE LOOP

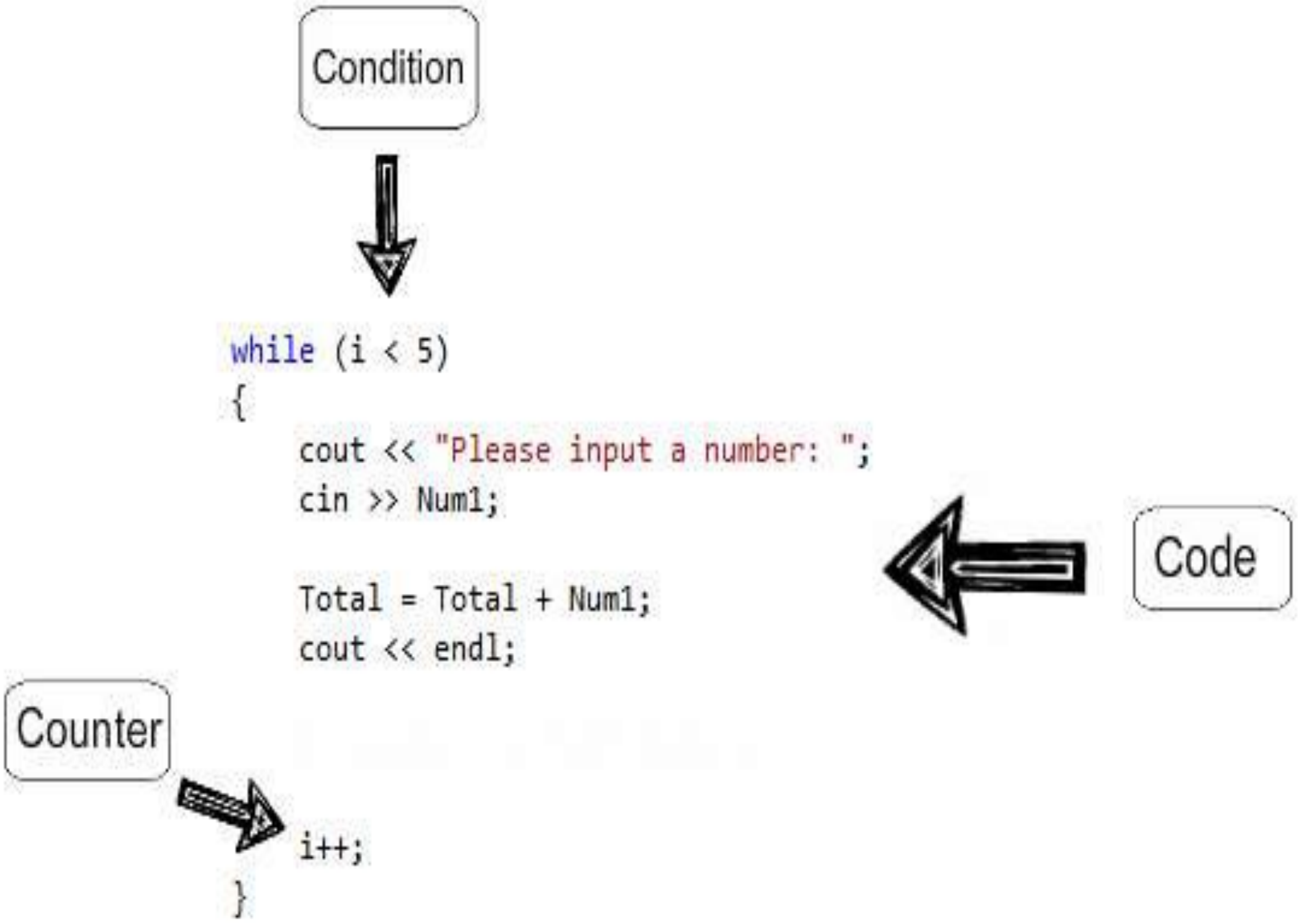
- The syntax of **while** statement :

```
        while (loop repetition condition)
                statement
```

- **Loop repetition condition** is the condition which controls the loop.
- The ***statement*** is repeated as long as the loop repetition condition is **true**.
- A loop is called an **infinite loop** if the loop repetition condition is always true.

Logic of a while Loop





```
while (i < 5)
{
    cout << "Please input a number: ";
    cin >> Num1;

    Total = Total + Num1;
    cout << endl;


    i++;
}
```

FOR LOOP

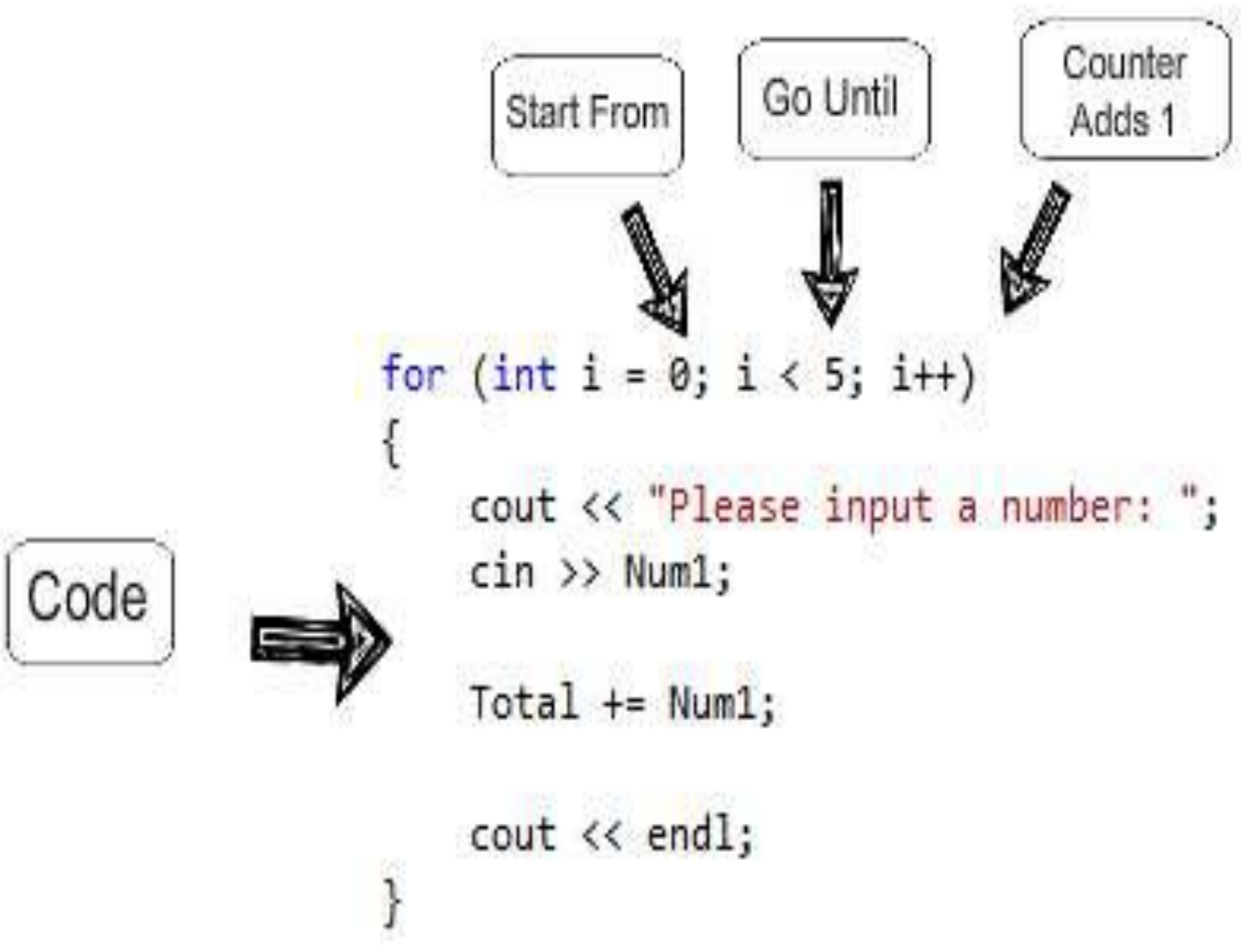
A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
{
    statement;
}
```

The *increment* portion is executed at the end of each iteration

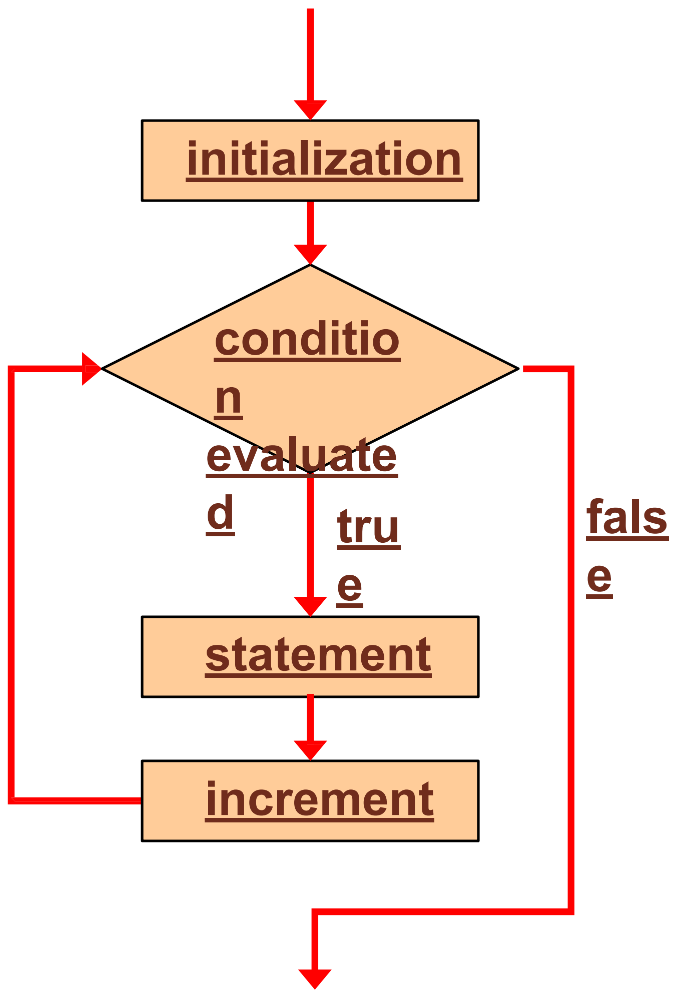


```
for (int i = 0; i < 5; i++)
{
    cout << "Please input a number: ";
    cin >> Num1;

    Total += Num1;

    cout << endl;
}
```

Logic of a for loop



EXAMPLE:

//program to display table of a given number using for loop.

```
#include<iostream.h
>  void main()
{
int n;
cout<<“\n Enter number:”;
cin>>n;
//for loop
for(int i=1;i<11;++i)
cout<<“\n”<<n<<“*”<<i<<“=“<<n*i;
}
```

OUTPUT

```
Enter number: 3
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
3*10=30
```

THE FOR LOOP VARIATIONS

- **Multiple initialization and update expressions**

A for loop may contain multiple initialization and/or multiple update expressions. These multiple expressions must be separated by commas.

e.g.

```
        for( i=1, sum=0; i<=n; sum+=i, ++i)
        cout<<“\n”<<i;
```

- **Infinite loop**

An infinite loop can be created by omitting the test expression as shown:

```
for(j=25;   ;--j)
cout<<“an infinite for loop”;
```

An infinite loop can also be created

```
as:  for( ;  ; )
cout<<“endless for loop”;
```

Empty loop

If a loop does not contain any statement in its loop-body, it is said to be an empty loop:

If we put a semicolon after for's parenthesis it repeats only for counting the control variable. And if we put a block of statements after such a loop, it is not a part of for loop.

e.g.

```
for(i=0;i<10;++i);
{
        cout<<"i="<<i<<endl;
}
```

The semicolon ends the loop here only

This is not the body of the for loop. For loop is an empty loop

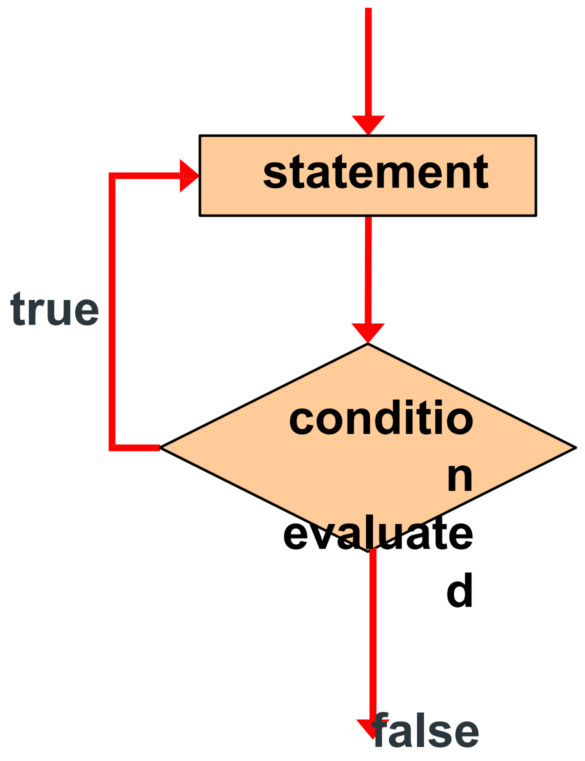
DO…WHILE LOOP

- The syntax of **do-while** statement in
  C: do

  statement

  while (**loop repetition condition**);
- The ***statement*** is first executed.
- If the **loop repetition condition** is true, the ***statement*** is repeated.
- Otherwise, the loop is exited.

Logic of a do…while loop



EXAMPLE:

//program to display counting from 1 to 10 using do-while loop.

```
#include<iostream.h
>  void main()
{
int i=1;
 //do-while loop
do
{
cout<<"\n"<<i;
i++;
}while(i<=10);
}
```

OUTPUT

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

NESTED LOOPS

- Nested loops consist of an **outer loop** with one or more **inner loops**.

e.g.,

```
for (i=1;i<=5;i++){
        for(j=1;j<=3;j++){
                ...
        }
}
```

Outer loop

Inner loop

- The above loop will run for 5*3 iterations.

EXAMPLE:

//program to display a pattern of a given character using nested loop.

```
#include<iostream.h
> void main()
{
int i,j;
for( i=1;i<5;++i)
{
cout<<“\n”;
for(j=1;j<=i;++j)
cout<<“*”;
}
}
```

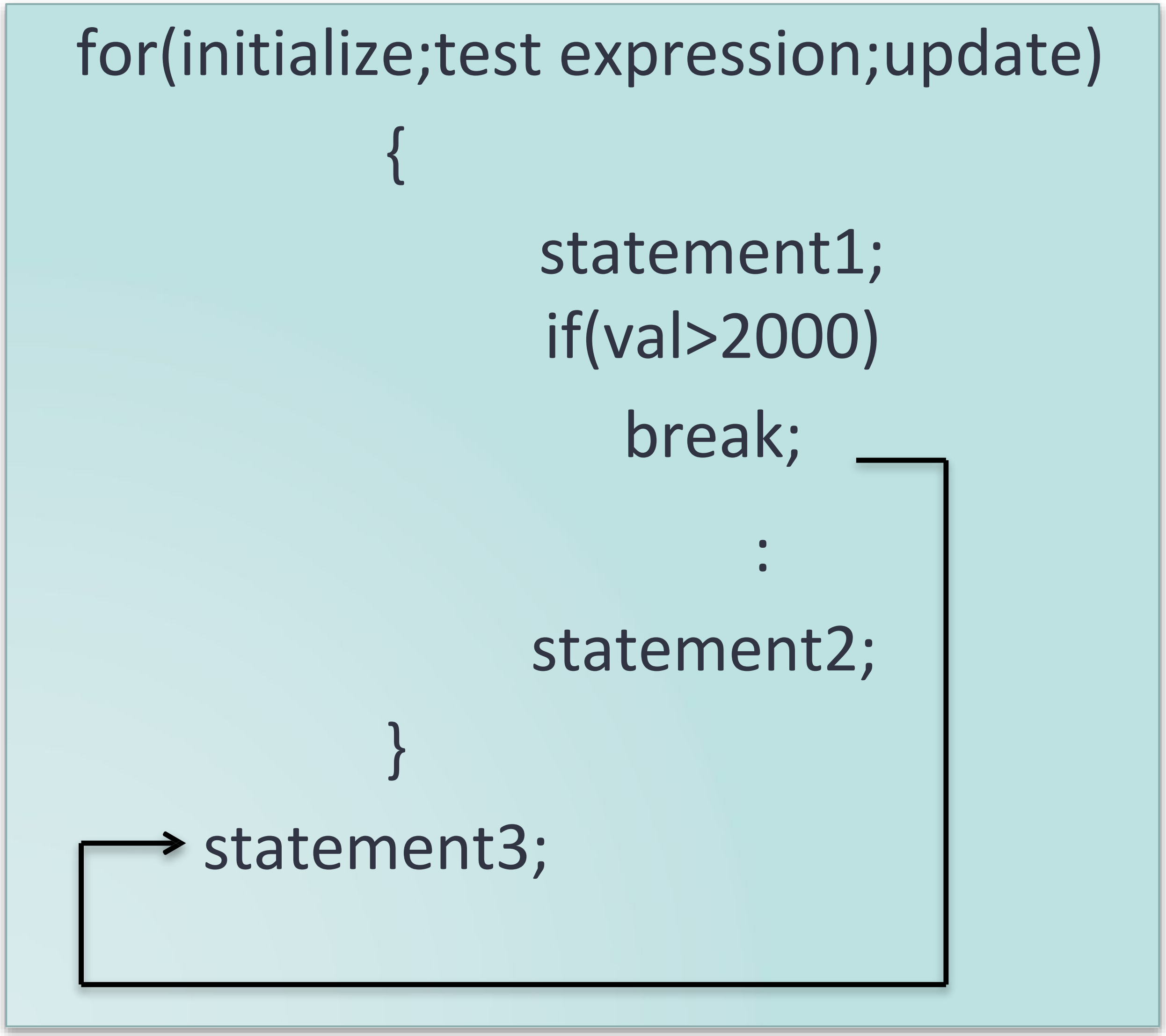
OUTPUT

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

2. The break statement

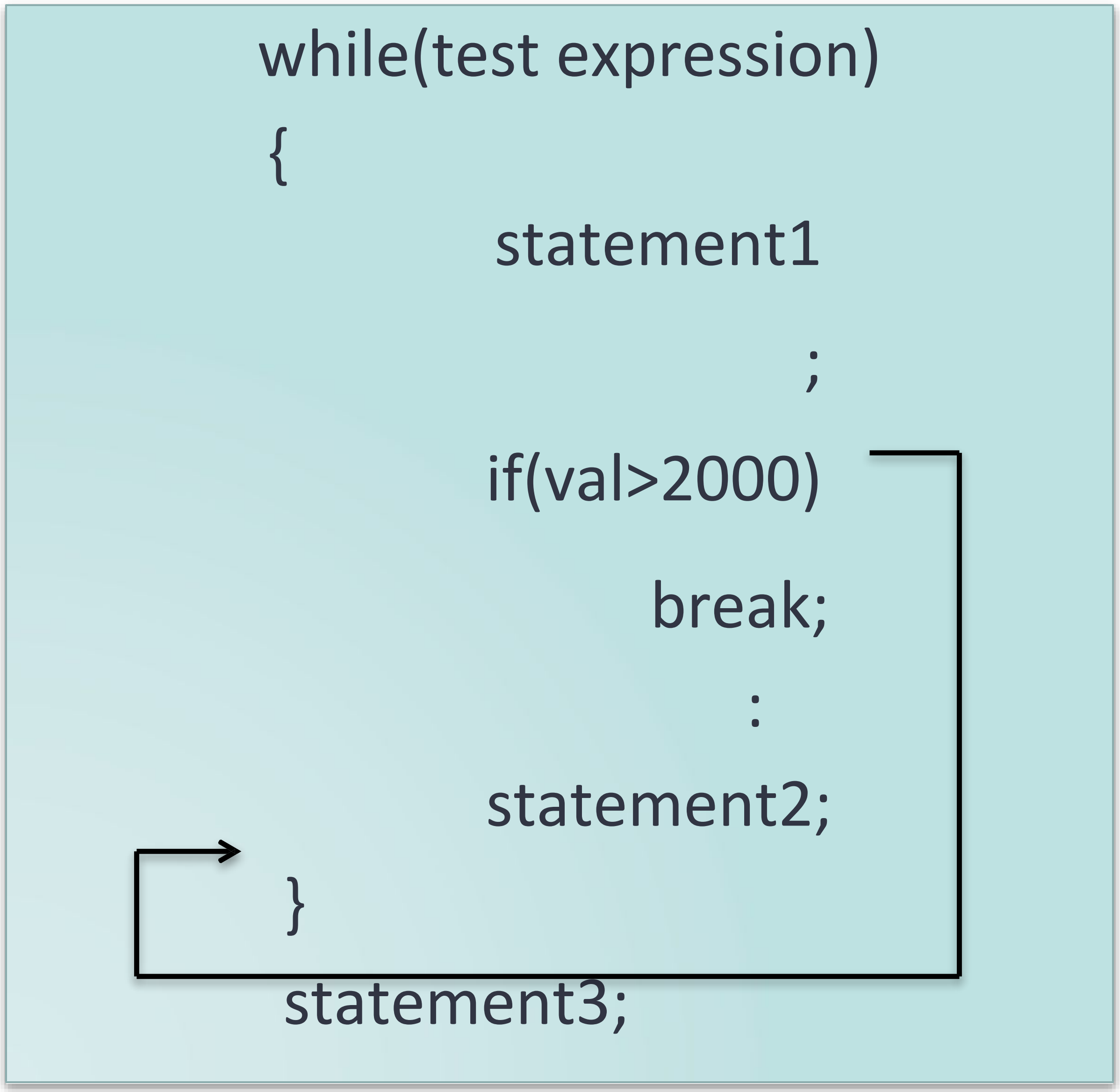
- The **break** statement enables a program to skip over part of the code.
- A **break** statement terminates the smallest enclosing while, do-while and for statements.
- A **break** statement skips the rest of the loop and jumps over to the statement following the loop.

The following figures explains the working of a break statement :



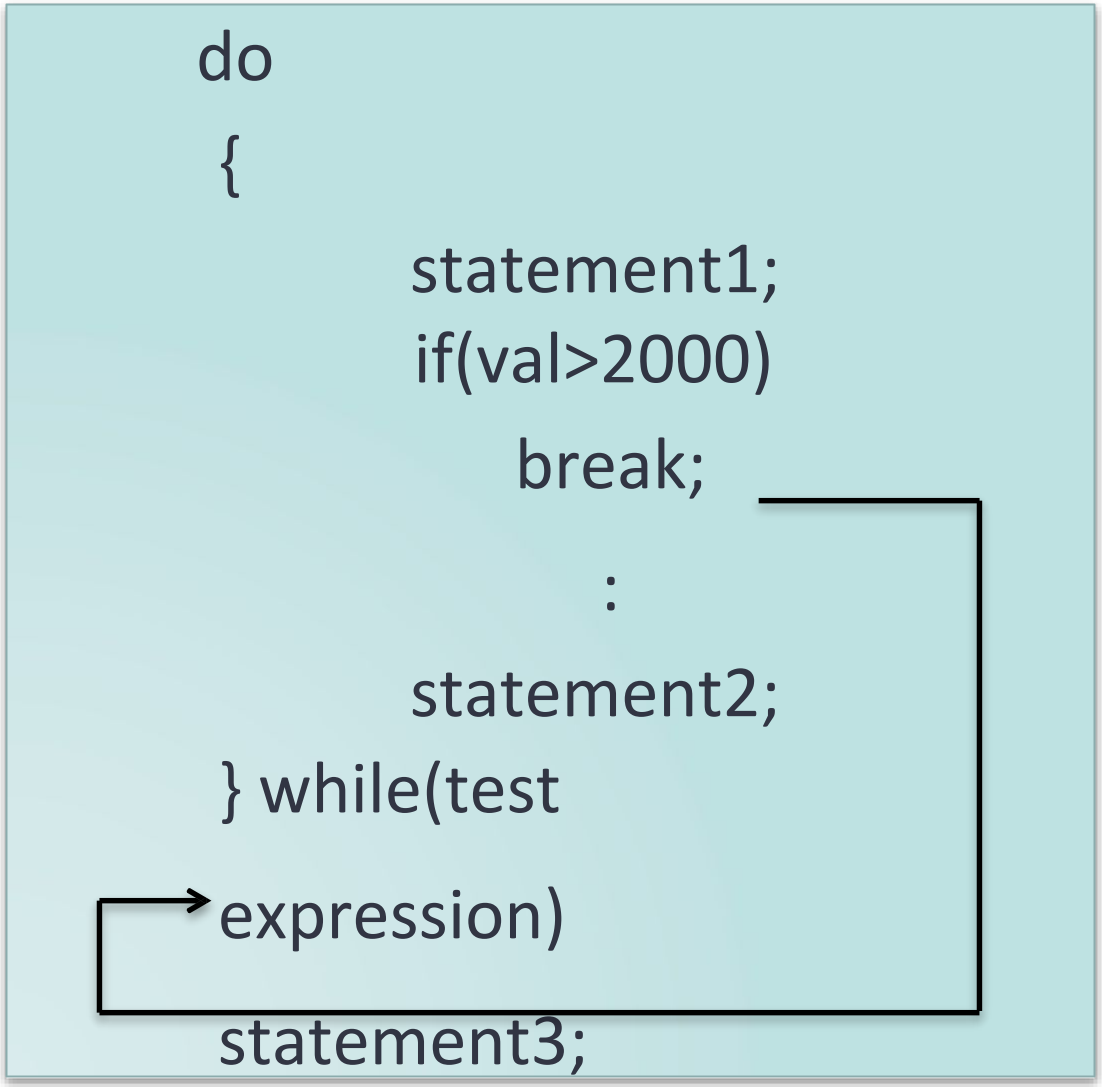
```
for(initialize;test expression;update)
        {
                statement1;
                if(val>2000)
                    break;
                        :
                statement2;
        }
    statement3;
```

<u>WORKING OF BREAK STATEMENT IN FOR LOOP</u>



```
while(test expression)
{
        statement1
                  ;
        if(val>2000)
              break;
                :
        statement2;
}
statement3;
```

<u>WORKING OF BREAK STATEMENT IN WHILE LOOP</u>



```
do
 {
          statement1;
          if(val>2000)
               break;
                  :
          statement2;
 } while(test
 expression)
 statement3;
```

WORKING OF BREAK STATEMENT IN DO-WHILE LOOP

PROGRAM BASED QUESTIONS…

1. Write a program to print first n natural numbers and their sum.
2. Write a program to calculate the factorial of an integer.
3. Write a program that prints 1 2 4 8 16 32 64 128.

4. Write a program to generate divisors of an integer.

5. Write a program to find whether a given number is odd or even. The program should continue as long as the user wants.

6. Write a program to print Fibonacci series i.e.,0 1 1 2 3 5 8 entered by user.

7. Write a program to calculate average of 10 numbers.

8. Write programs to produce the following designs:

```
       A
    A      B
    A      B      C
    A      B      C      D
    A      B      C      D    E
```

THANK YOU