Loop - Repetition structure

Loop - Repetition structure

Topics to cover here:

- **Introduction to Repetition**
- **FOR** .. **DO** Statement
- **WHILE** .. **DO** Statement
- **DO .. WHILE** Statement
- **Nested Loops**
- **BREAK** and **CONTINUE** Statements

Repetition: An Introduction

- You can repeat many steps in an algorithm.
- Repetition of a sequence of steps in an algorithm is called a **loop**.

- Types of loops:

  - Counter-controlled loops

    (e.g. WHILE..DO, FOR..DO, DO..WHILE)

  - Conditional loops (e.g. WHILE..DO, DO..WHILE)

Example

```
int sum ;
sum = 1+2+3+4+5+……..+10 ;
cout << sum ;
```

Find the Sum of the first 100 Integer starting from 1

Counter-Controlled Loop

- The repetition of this loop is controlled by a **loop control variable (lcv)** whose value represents a count.
- This type of loops is used when you can determine prior to loop execution exactly how many loop repetitions will be needed to solve a problem.
- The **lcv** should be incremented as the final statement of the loop.

- **NOTE:**

  All types of repetition statements could be used as counter-controlled loops. The **FOR** statement is based suited for this type of looping.

FOR Statement

- It could be an increasing loop or decreasing loop.

- **Syntax for the increasing loop:**

FOR (lcv ← initial_value **TO** final_value [**BY** increment_value]) **DO**

Statements

END FOR

where, lcv is the loop control variable, and the part [**BY** increment] is optional (it is omitted if it is 1).

FOR .. DO Statement .. Cont.

- **Semantics:**

  The execution of this statement is as follows

  1- The lcv is set to initial_value

  2- lcv is checked with final_value

  - if it is less than or equal, then

  * ***statements*** are executed

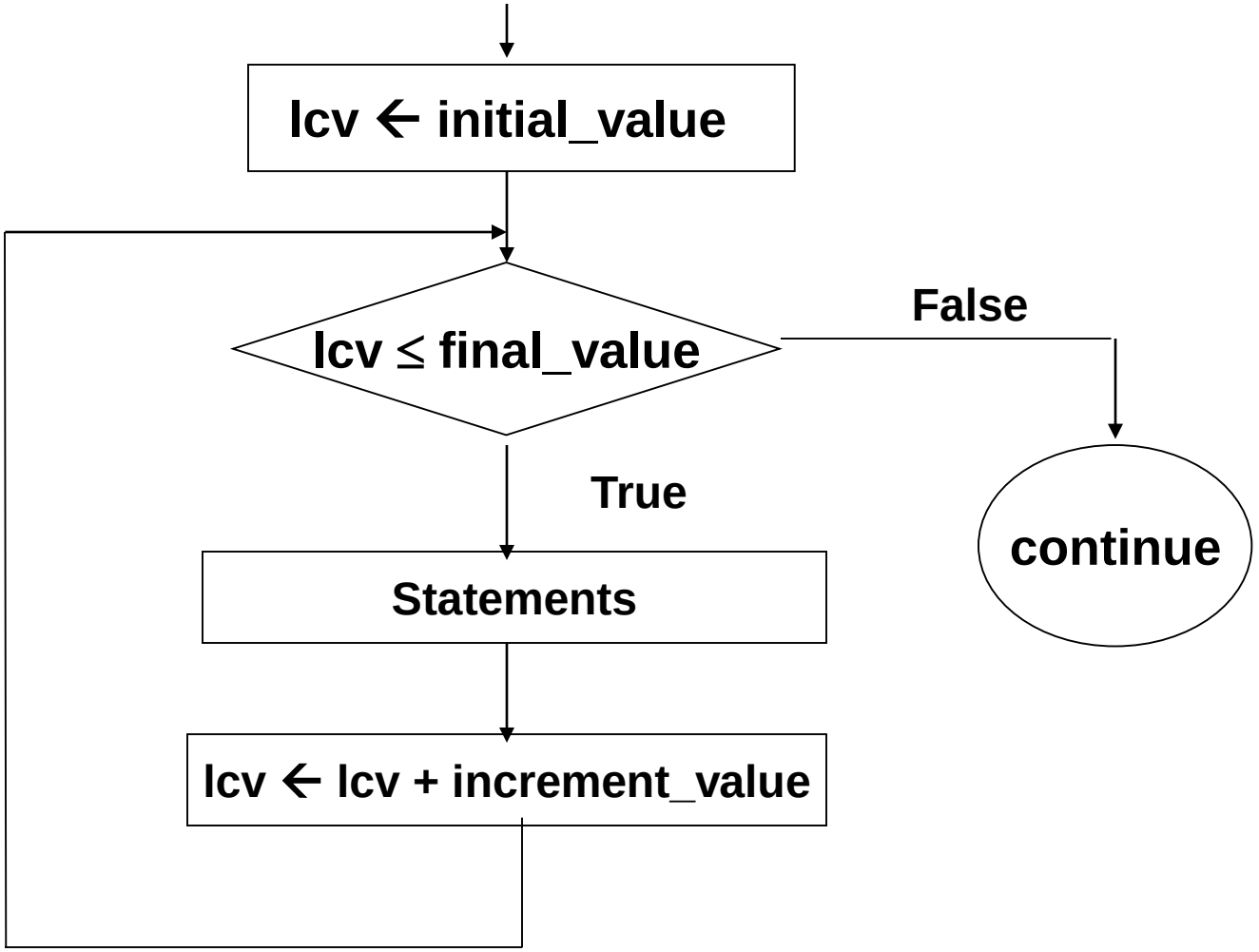
  * the lcv is incremented by increment_value

  * repeat the process from step (2)

  - else, goto the rest of the algorithm

These steps are shown in the following flowchart:

FOR .. DO Statement .. Cont.



FOR .. DO Statement .. Cont.

- **Syntax for the decreasing loop:**

FOR (lcv ← final_value **DOWNTO** initial_value [**BY** decrement_value]) **DO**

Statements

END FOR

- The lcv is initialized to the final_value and consequentially will take values up to the initial_value. The lcv is decremented in each step by the decrement_value.

Examples

- **Example 1**

Write an algorithm that print the first 10 positive integer numbers

- **Analysis Stage:**
  - ***Problem Input:***

    The 10 integers are generated by the algorithm
  - ***Problem Output:***

    The first 10 positive integers

Example 1 .. Cont.

- **Algorithm Design:**

ALGORITHM Print

Begin

FOR (I ← 1 **TO 10**) **DO** // here, increment by 1

OUTPUT I, “ “

END FOR

END print

Example 1.. Cont.

- **Testing the Algorithm**

I	(I ≤ 5)	The output:
1	True	12345678910
2	True	
3	True	
4	True	
5	True	
6	True	
7	True	
8	True	
9	True	
10	True	
11	False (Stop)	

For Statement in C++

- **Syntax**

  for (initialization; test expression; update)

  Statements

  where,

  - initialization is to initialize the loop control variable (lcv)
  - test expression is the condition that will stop the loop
  - update is the increment to the lcv (in case of increasing loop) or decrement to the lcv (in case of decreasing loop)

Example 1: C++ Program

```
#include <iostream>
using namespace std;
void main ( )
{
int I;
for( I = 1;I<= 10;I++ )
        cout << I<<“ “ endl;
}
```

- **Exercise**

  Modify the above Example so that it prints the numbers from 1 to n.
- Modify the above Example so that it prints the numbers from m to n.

Example2

Write an algorithm that finds the sum of the first 5 positive integer numbers

- **Analysis Stage:**
  - ***Problem Input:***

    The 5 integers are generated by the algorithm
  - ***Problem Output:***

    The sum of the first 5 positive integers

Example 2 .. Cont.

- **Algorithm Design:**

ALGORITHM SumOfIntegers

Begin

sum ← 0

FOR (I ← 1 **TO** 5) **DO** // here, increment by 1

sum ← sum + I

END FOR

OUTPUT " Sum = ", sum

END SumOfIntegers

Example 2.. Cont.

- **Testing the Algorithm**

sum	I	(I ≤ 5)	The output:
0			
	1	True	
1			
	2	True	
3			
	3	True	
6			
	4	True	
10			
	5	True	
15			
	6	False (Stop)	
			sum = 15

Example 2: C++ Program

```
#include <iostream>
using namespace std;
void main ( )
{ int I, sum = 0;
   for ( I = 1; I < =5; I++ )    // here, increment by 1
         sum = sum + I;
  cout << " Sum = " <<  sum << endl;
}
```

- **Exercise**

  Modify the above example so that it finds the sum of any 10 integers.

WHILE Statement

- **Syntax:**

In pseudo code	In C++

lcv ← initial value
WHILE (logical expression) DO
  Statements
  lcv ← lcv+1
END WHILE
	Lcv=initial value;
	WHILE (logical expression)
	{
	  Statements;
	  lcv = lcv+1;
	}

Example 3

Write an algorithm that finds the product of odd numbers among the first 6 positive integers.

- **Analysis Stage:**
  - ***Problem Input:***

    The first 6 integers are generated by the algorithm
  - ***Problem Output:***

    The product of the odd integers

Example 3 .. Cont.

- **Algorithm Design:**

ALGORITHM product_Odd

Begin

p ← 1

I ← 1

while (I≤ 6) do

p← p * I

I=I+2

END while

OUTPUT “ product = “, p

END product_Odd

Example 3 .. Cont.

- **Testing the Algorithm**

p	I	(I ≤ 6)	The output:
1			
	1		
		True	
1			
	3		
		True	
3			
	5		
		True	
15			
	7		
		False (Stop)	
			product = 15

Example 3: C++ Program

```
/* The program finds the sum of the odd numbers among the first 6
    positive integers. */
#include <iostream>
using namespace std;
void main ( )
{ int i, p = 1 ;
  i =1
  while (i <=6)
   {
      p *= i ;
      i += 2;
  }
   cout << " product =  " << p << endl;
}
```

Example 4

Write an algorithm that reads 10 numbers and finds the maximum and minimum numbers among them.

- **Analysis Stage:**

Use a loop to read the 10 numbers one by one and test it for maximum and minimum.

- ***Problem Input:***

  Ten numbers to be read repeatedly.

- ***Problem Output:***

  maximum, minimum

- ***Criteria***

  Let max and min equal the first number in the sequence

Example 4 .. Cont.

- **Algorithm Design:**

ALGORITHM MaxMin
Begin
OUTPUT " Enter a number: "
INPUT n
max ← n
min ← n
I ← 2
while (I≤10) **DO**
OUTPUT " Enter a number: "
INPUT n
IF (n > max) **THEN**
max ← n
END IF
IF (n < min) **THEN**
min ← n
END IF
I←I+1
END while
OUTPUT " Max = " , max, " Min = " , min
END MaxMin

Example 4 .. Cont.

- **Testing the Algorithm (for 4 numbers only)**

I	(I ≤4)	n	max	min	(n>max)	(n<min)	The output:
							Enter a number:
		5					
			5				
				5			
2	True						
							Enter a number:
		7					
					True		
			7				
						False	
3	True						
							Enter a number:
		2					
					False		
						True	
				2			
4	True						
							Enter a number:
		9					
					True		
			9				
						False	
5	False (stop)						
							Max = 9 Min = 2

Example 4: C++ Program

```
#include <iostream>
using namespace std;
void main ( )
{ int I, n, max, min;
   cout << " Enter a number: " ;
   cin >>  n ;
    max = n ;        min =  n ;
    I = 2;
    for  (I <= 10)
    { cout <<  " Enter a number: "  ;
       cin >>  n ;
       if  ( n > max )
            max =  n ;
       if  ( n < min )
           min = n ;
      I++
     }
    cout << " Max = "<< max<< " Min = "<< min<< endl;
}
```

Conditional Loops

- Such loops are used when you cannot determine the exact number of loop repetitions before loop execution begins.
- The number of repetitions may depend on some aspects of the data that is not known before the loop is entered but that usually can be stated by a **condition**.
- The condition value should be modified inside the loop to ensure loop termination.

- **NOTE:**

  The **WHILE .. DO** statement and **DO .. WHILE** statement are best suited for this type of looping.

WHILE Statement

- **Syntax:**

  WHILE (logical expression) **DO**

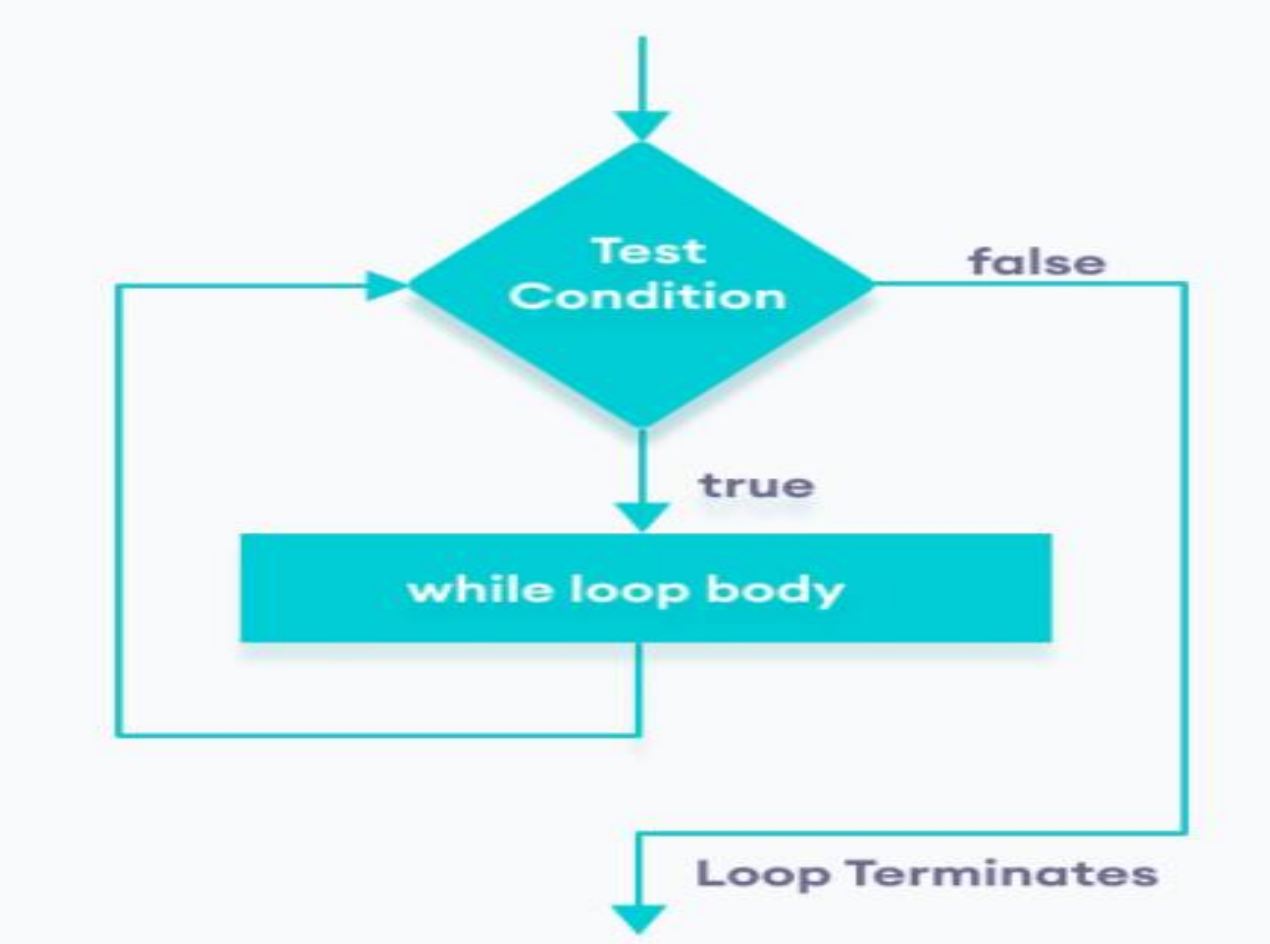
  Statements

  END WHILE

- **Semantics:**

  The execution of this statement is shown as in the following flowchart:

WHILE Statement Execution



While Statement in C++

Syntax:

while (*logical expression*)
statements;

where *statements* could be one or more statements enclosed by braces { and }

Semantics:

This statement has the same meaning as in the algorithmic language.

An Example on Conditional Loops

- **Example 1:**

  Write an algorithm that reads a sequence of integer numbers and finds the product of the numbers as long they are positive.

- **Analysis Stage:**

  - ***Problem Input:***

    a sequence of integers

  - ***Problem Output:***

    The product of the positive integers

  - ***Criteria:***

    Any negative number will stop looping

Example 1 .. Cont.

- **Algorithm Design:**

ALGORITHM Multiplying
Begin
    product ← 1
    OUTPUT " Enter first number: "
    INPUT number
    WHILE (number > 0) **DO**
        product ← product * number
        OUTPUT " Enter next number: "
        INPUT number
    END WHILE
    OUTPUT " The product is " , product
END Multiplying

Example 1 .. Cont.

- **Testing the Algorithm**

product	number	(number > 0)	The output:
1			Enter first number:
	2		
		True	
2			
			Enter next number:
	5		
		True	
10			
			Enter next number:
	7		
		True	
70			
			Enter next number:
	-3	False (stop)	
			The product is 70

Example 1: C++ Program

```
#include <iostream>
using namespace std;
void main ( )
{
   int number, product;
   product = 1 ;
   cout << " Enter first number: " ;
   cin >> number
   while ( number > 0 )
     { product = product * number ;
         cout << " Enter next number; to end enter any negative number " ;
         cin >> number ;
       }
     cout << " The product is " << product << endl;
}
```

Sentinel-Controlled Loops

- This type of loops is used when you don't know exactly how many data items a loop will process before it begins execution.
- One way to handle this situation is to instruct the user to enter a unique data value, called a sentinel value, as the last data item.
- The sentinel value should be carefully chosen and must be a value that cannot possibly occur data.

- **NOTE:**

  The **WHILE .. DO** statement and **DO .. WHILE** statement are can be used for this type of looping.

An Example on Sentinel-Controlled Loops

- **Example 2:**

  Write an algorithm that sums up the student's exam scores by using sentinel-controlled loop.

- **Analysis Stage:**

  You must choose a sentinel value that could not be a student's score (say, -1)

  - ***Problem Input:***

    a sequence of student's exam scores that ends with -1

  - ***Problem Output:***

    The sum of the scores

  - ***Criteria:***

    the input -1 will stop looping

Example 2 .. Cont.

- **Algorithm Design:**

ALGORITHM Scores
Begin
    sum ← 0
    OUTPUT “Enter a score: “
    INPUT score
    WHILE (score ≠ -1) **DO**
        sum ← sum + score
        OUTPUT “Enter the next score, to end enter -1: “
        INPUT score
    END WHILE
    OUTPUT “ The sum is “ , sum
END Score

Example 2 .. Cont.

- **Testing the Algorithm**

sum	score	(score ≠ -1)	The output:
0			
			Enter a score:
	60		
		True	
60			
			Enter the next score, to end enter -1:
	75		
		True	
135			
			Enter the next score, to end enter -1:
	80		
		True	
215			
			Enter the next score, to end enter -1:
	-1		
		False (stop)	
			The sum is 215

Example 2: C++ Program

```
#include <iostream>
using namespace std;
void main ( )
{ int sum = 0 , score ;
   cout << "Enter a score: " ;
    cin >>  score ;
    while ( score  !=  -1 )
      {
        sum =- sum + score ;
        cout << "Enter the next score, to end enter -1: " ;
         cin >>  score ;
       }
     cout <<  " The sum is "  <<  sum << endl ;
}
```

- **Exercise**

  Modify the above example so that it calculates the average of the exam scores.

For and While loop executes zero or more times.

What if we want the loop to execute at least one time?

do-while

Do while loop execute on or more times

do-while loop

Syntax:

In pseudo code	In C++

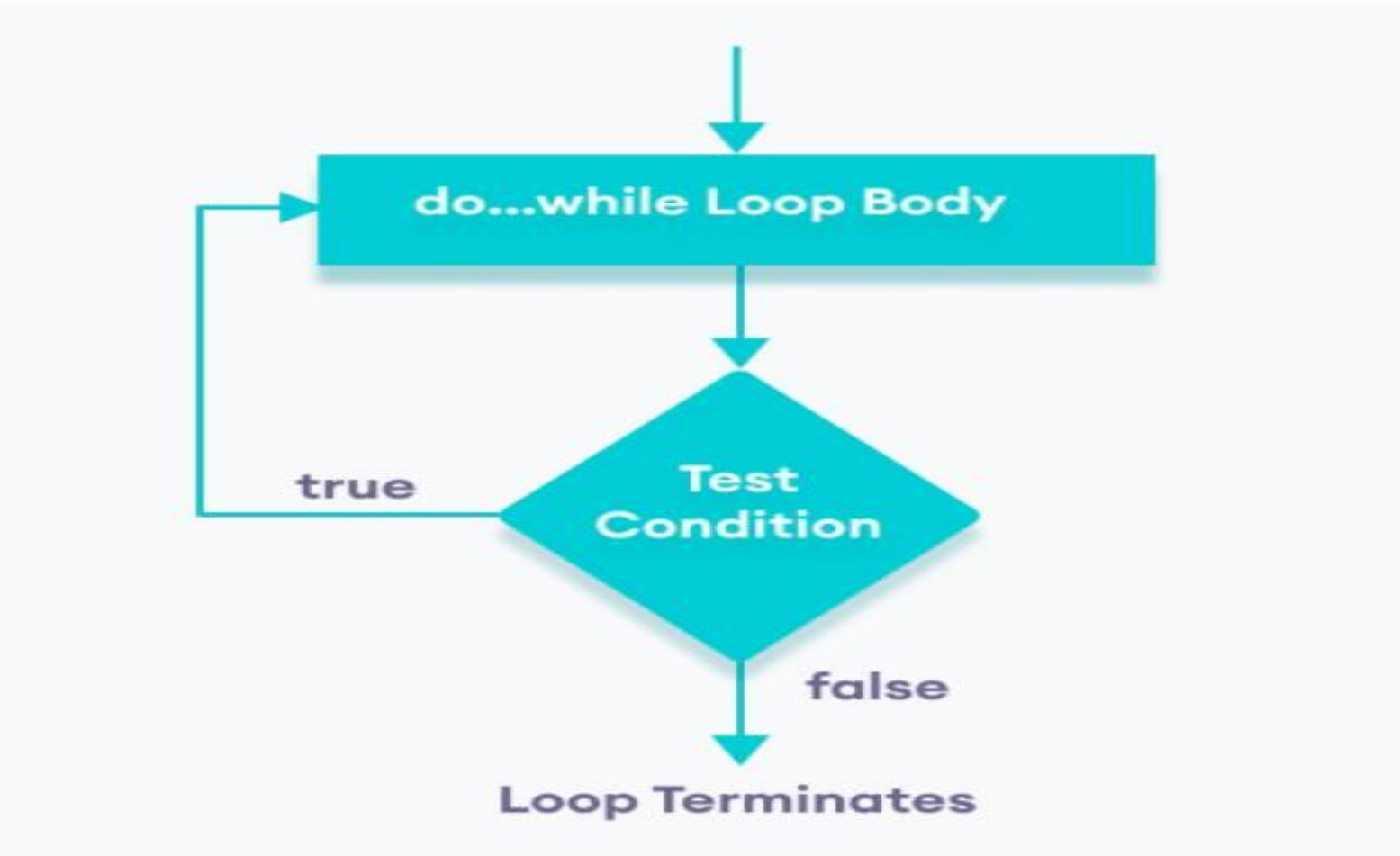
DO
Statements
WHILE (condition)
	do
	{
	statements ;
	}
	while (condition) ;

The semantics (execution) of this statement:

The statements are evaluated first, then the condition is tested. If the condition is true, the process is repeated until the condition become false.

•This statement is executed at least once.

Flow chart for do-while loop



do-while loop

Example 1:

Write an algorithm that reads a sequence of integer numbers and finds the product of the numbers as long they are positive.

Example

```
#include <iostream>
using namespace std;
void main ( )
{
   int number, product;
   product = 1 ;
do
   {
    cout << " Enter positive number" ;
    cin >> number
    product = product * number ;
   } while ( number > 0 ) ;

cout << " The product is " << product << endl;
}
```

Nested Loops

- When you write a loop inside another loop, then it is called a nested loop.
- You can use FOR loop inside another FOR loop
- You can use WHILE loop inside a FOR loop or vise versa.

Example of Nested Loops

- **Example:**

Write an algorithm that prints the following figure by printing one "*" each time. Use nested loops.

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

Example of Nested Loops

- **Algorithm Design:**

ALGORITHM Stars

Begin

```
    FOR ( I ← 1  TO   5 )   DO
        FOR (  J ←  1  TO  I )  DO
            OUTPUT " * " , "  "
        END FOR
        OUTPUT "\n"
    END FOR
```

END Stars

Example : C++ Program

/* The program prints a figure of stars by using nested loops to print one “*” at a time. */

```
#include <iostream>
using namespace std;
void main ( )
{ int I, J;
   for  ( I = 1; I <=  5 ; I++)
       for  (  J =  1; J <=  I ; J++ )
              cout << “ * “  <<  “  “ ;
  cout << endl ;
}
```

The BREAK and CONTINUE Statements in Loops

- **The BREAK Statement**
- Use **BREAK** to leave the loop even if the condition for its end is not achieved.
- **Example**

```
FOR ( n ← 10  DOWNTO  0  BY  -1 ) DO
    OUTPUT  n * 2 , “ , “
    IF ( n = 4 ) THEN
        OUTPUT “ Loop Aborted”
        BREAK
    END IF
END FOR
```

The output is: 20, 18, 16, 14, 12, 10, 8, Loop Aborted

The BREAK and CONTINUE Statements in Loops

- **The CONTINUE Statement**
- This statement causes the program to skip the rest of the loop in the present iteration as if the end of the statement block would have been reached, causing it to jump to the following iteration.
- **Example**

```
FOR ( n ← 1  TO  10 )  DO
    IF ( n = 5 ) THEN
        CONTINUE
    END IF
    OUTPUT   n ,  “,  ”
END FOR
OUTPUT “ Finished “
```

- The output is: 1, 2, 3, 4, 6, 7, 8, 9, 10, Finished