Control structures

Ms Kanwal Lodhi

Conditional or Ternary Operator

- The conditional operator is an operator used in C and C++ (as well as other languages, such as C#). The ?: operator returns one of two values depending on the result of an expression.
- It is an alternative of simple if-else

Syntax

variable=(expression1-condition) ? expression 2(true) : expression 3(false)

- The condition is specified as relational or logical expression
- If *expression 1* evaluates to true, then *expression 2* is evaluated.
- If *expression 1* evaluates to false, then *expression 3* is evaluated instead.

Example

```
c=(a>b)?a:b;
```

This statement means if value of a is greater than b ,assign a to c else assign b.

The above statement is equivalent to:

```
If(a>b)
    c=a;
else
    c=b;
```

3. Repetitive structure / Loops

- Any statement or set of statements that is executed repeatedly is called loop. The statement in a loop are executed for a specified no. of times until some given condition remains true.
- Types:
  - The “while” loop
  - The “for” loop
  - The “do-while” loop.
- **<u>The “while” loop</u>**

  The syntax of the while loop is:

  while (condition) { // body of the loop }

  Here,

  A while loop evaluates the condition

  If the condition evaluates to true, the code inside the while loop is executed.

  The condition is evaluated again.

  This process continues until the condition is false.

  When the condition evaluates to false, the loop terminates.

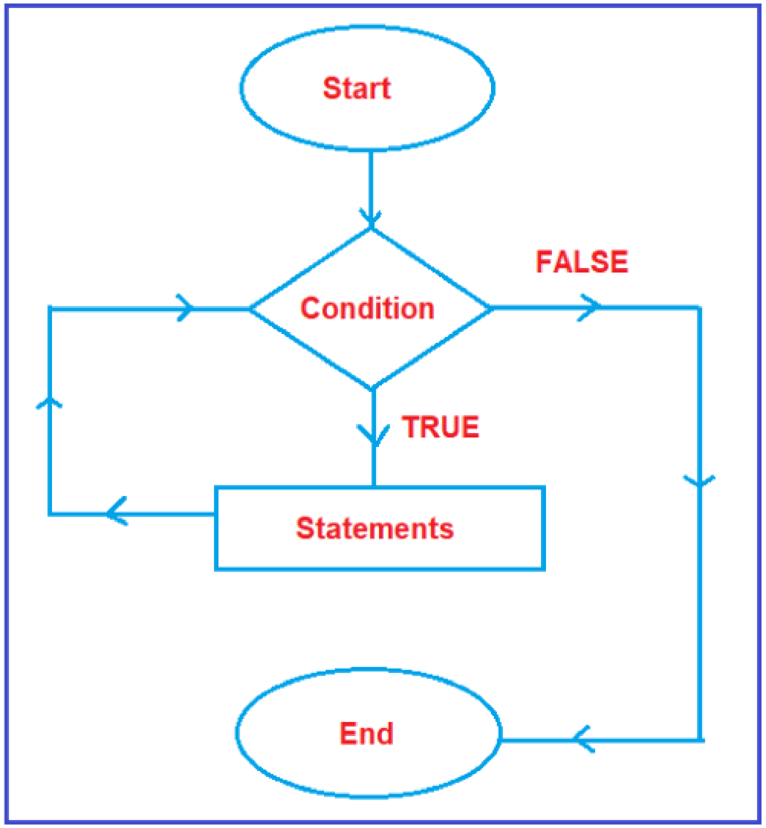
while loop

Example

C++ Program to print numbers from 1 to 5

```
#include <iostream>
 using namespace std;
 int main()
 {
 int i = 1; // while loop from 1 to 5
 while (i <= 5)
{
cout << i << " ";
++i;
}
return 0;
 }
```

Flow Chart

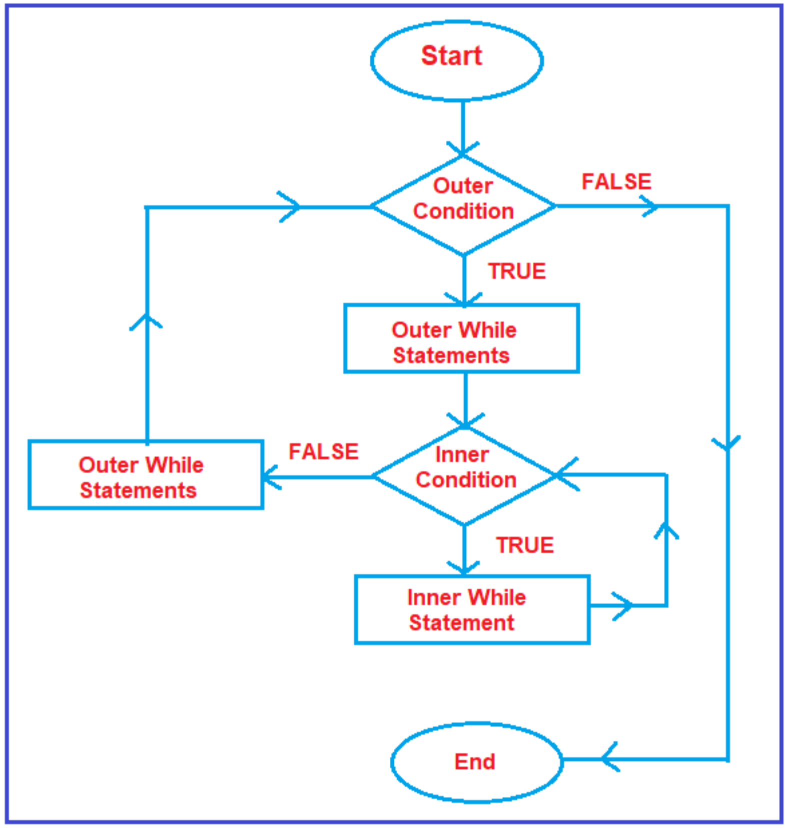


Nested while loop

Syntax:

```
while(test_expression_1)
{
statement(s)
    while(test_expression_2)
{
        statement(s)
}
 }
```

Flow chart



Example of nested while loop

code

```
•    #include <iostream>
•    #include <conio.h>
•    using namespace std;
•    int main()
•    {
•    int i=0;
•    while(i<=3){
•    cout<<"outer loop executes"<< endl;
•    int j=0;
•    while(j<=4){
•    cout<<"inner loop executes";
•    cout << "i = "<<i<<" and j="<<j << endl;
•    j++;
•    }
•    i++;
•    }
•    getch();
•    return 0;
•    }
```

output

```
•    Outer loop executes
•    Inner loop executes i=0 and j=0
•     Inner loop executes i=0 and j=1
•    Inner loop executes i=0 and j=2
•     Inner loop executes i=0 and j=3
•     Inner loop executes i=0 and j=4
•     Outer loop executes
•     Inner loop executes i=0 and j=0
•    Inner loop executes i=0 and j=1
•    Inner loop executes i=0 and j=2
•    Inner loop executes i=0 and j=3
•    Inner loop executes i=0 and j=4
•    Outer loop executes
•     Inner loop executes i=0 and j=0
•     Inner loop executes i=0 and j=1
•     Inner loop executes i=0 and j=2
•     Inner loop executes i=0 and j=3
•     Inner loop executes i=0 and j=4
•    Outer loop executes
•     Inner loop executes i=0 and j=0
•     Inner loop executes i=0 and j=1
•     Inner loop executes i=0 and j=2
•     Inner loop executes i=0 and j=3
•     Inner loop executes i=0 and j=4
```

for loop

The “for’ loop

Syntax

for (initialization; condition; update)
{ // statement(s) }

At the start of for loop execution, initialization statement is executed and then the condition is checked.

If the condition is true, statement(s) inside for block are executed.

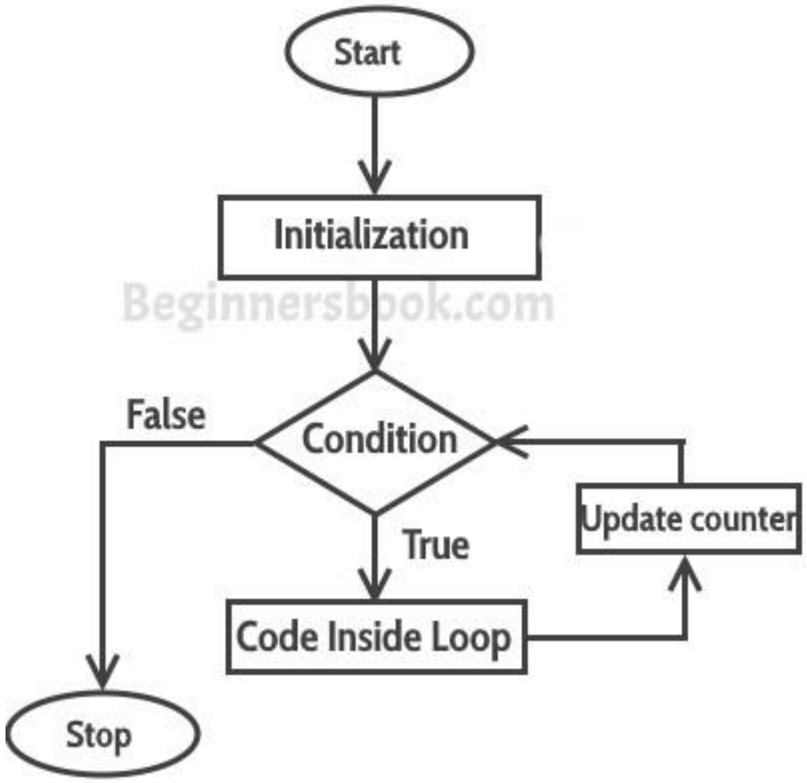
And the update statement is executed. The condition is checked again.

If it evaluates to true, the statement(s) inside the for loop are executed. This cycle goes on.

If at all, the condition evaluates to false, for loop execution is deemed completed and the program control comes out of the loop.

And the program continues with the execution of statements after for loop if any.

Flow chart



for loop

Structure of for loop

- The loop structure has following parts.
  - Initialization
  - Condition
  - Increment or decrement
  - Body of loop

Example 1

A program to print the first 10 natural numbers.

```
#include <iostream>
using namespace std;
 int main()
 {
int i;
 cout << " The natural numbers are: \n";
for (i = 1; i <= 10; i++)
 {
cout << i << " ";
}
cout << endl;
 }
```

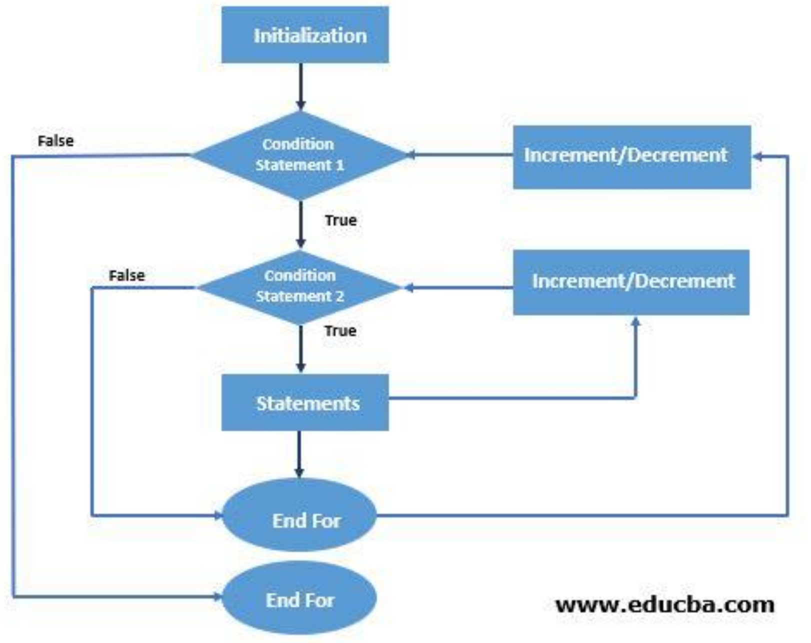
Nested for loop

Syntax:

- **Syntax for Nested For loop:**

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

Flowchart



Example of nested for loop

code

```
// C++ program to display a pattern
// with 5 rows and 3 columns
#include <iostream>
using namespace std;
int main()
{
int rows = 5;
int columns = 3;
for (int i = 1; i <= rows; ++i)
 {
 for (int j = 1; j <= columns; ++j)
 {
cout << "* ";
 }
 cout << endl;
 }
return 0;
 }
```

output

```
* * *

* * *

* * *

* * *

* * *
```

The "do-while" loop

The do-while loop

It's a variant of while loop.

Its syntax is:

```
do {
// body of loop;
 }
 while (condition);
```

Here,

The body of the loop is executed at first. Then the condition is evaluated.

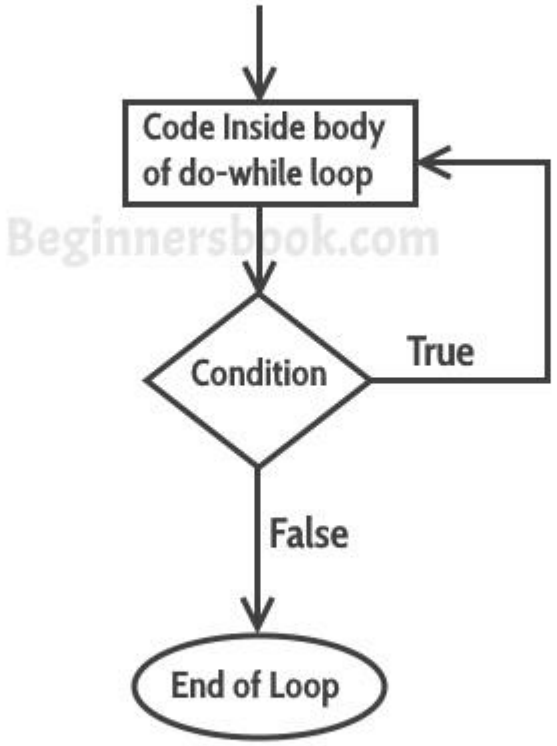
If the condition evaluates to true, the body of the loop inside the do statement is executed again.

The condition is evaluated once again.

If the condition evaluates to true, the body of the loop inside the do statement is executed again.

This process continues until the condition evaluates to false. Then the loop stops.

Flow chart



The “do-while” loop

- Example

C++ Program to print numbers from 1 to 5 #include <iostream>

```
using namespace std;
int main()
{
int i = 1;
// do...while loop from 1 to 5
do
{
cout << i << " ";
++i;
 }
while (i <= 5);
return 0;
 }
```

Nested do-while loop

Syntax:

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

Flowchart

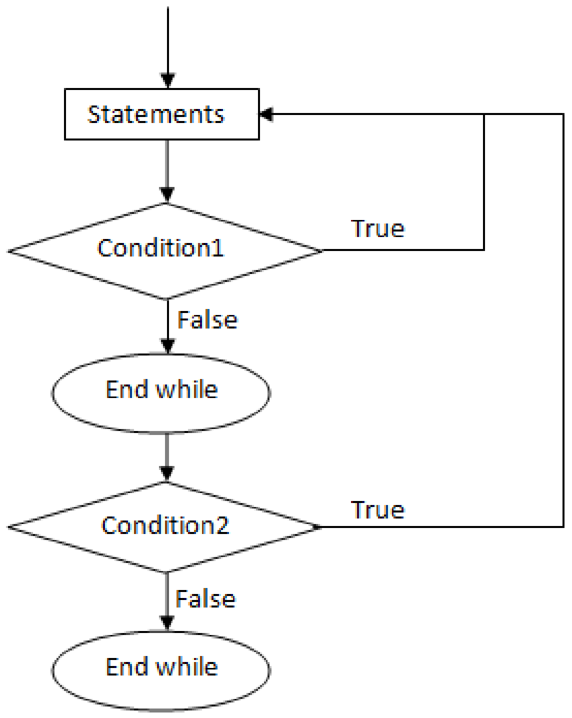


Fig: Flowchart for nested do-while loop

Nested do-while loop

code

```
#include <iostream>
using namespace std;
 int main ()
{
 int i = 0;
 do
 {
int j = 0;
 do
 {
cout << "i = " << i << " and j = " << j << endl;
j++;
}
while(j < 5);
 i++;
 }
while(i < 3);
 return 0;
}
```

Output

```
i = 0 and j = 0
i = 0 and j = 1
i = 0 and j = 2
i = 0 and j = 3
i = 0 and j = 4
i = 1 and j = 0
i = 1 and j = 1
i = 1 and j = 2
i = 1 and j = 3
i = 1 and j = 4
i = 2 and j = 0
i = 2 and j = 1
i = 2 and j = 2
i = 2 and j = 3
i = 2 and j = 4
```

continue statement

- continue statement

This statement is used to skip the current iteration of the loop and control of the program goes to the next iteration.

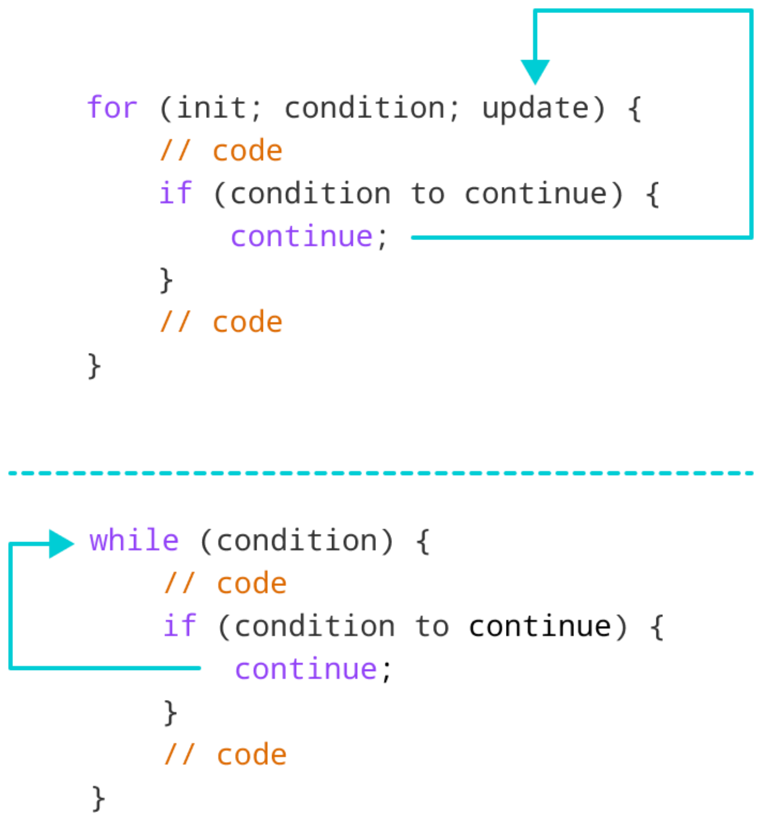
- syntax:

  continue;

In a for loop, continue skips the current iteration and the control flow jumps to the update expression.

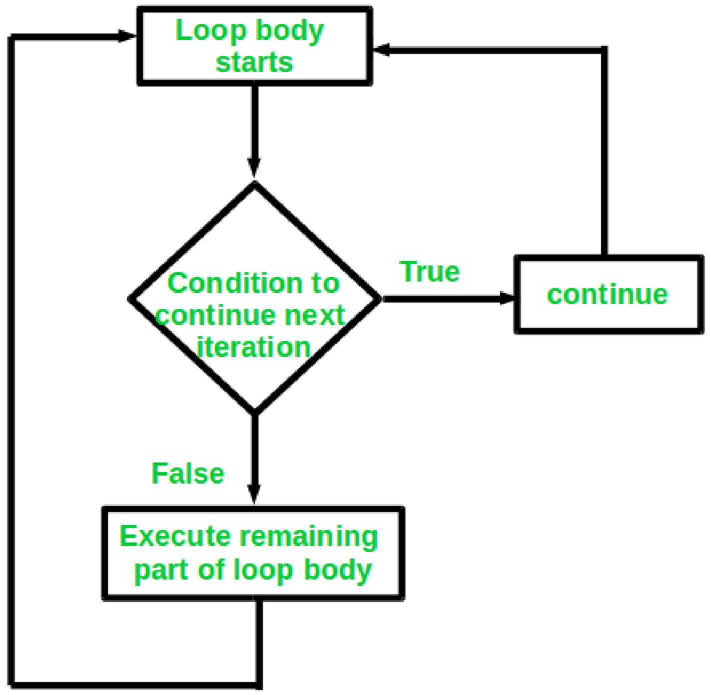
```
// program to print the value of i
#include <iostream>
using namespace std;
int main()
{
 for (int i = 1; i <= 5; i++)
 {
 // condition to continue
if (i == 3)
 {
 continue;
 }
cout << i << endl;
 }
return 0;
}
```

continue statement



```
for (init; condition; update) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```


Loop body starts
Condition to continue next iteration
True
continue
False
Execute remaining part of loop body