Fundamentals of Computer Programming

Chapter 3
Flow of Control Part II
(Loop Statements)

Chere L. (M.Tech)
Lecturer, SWEG, AASTU

Outline

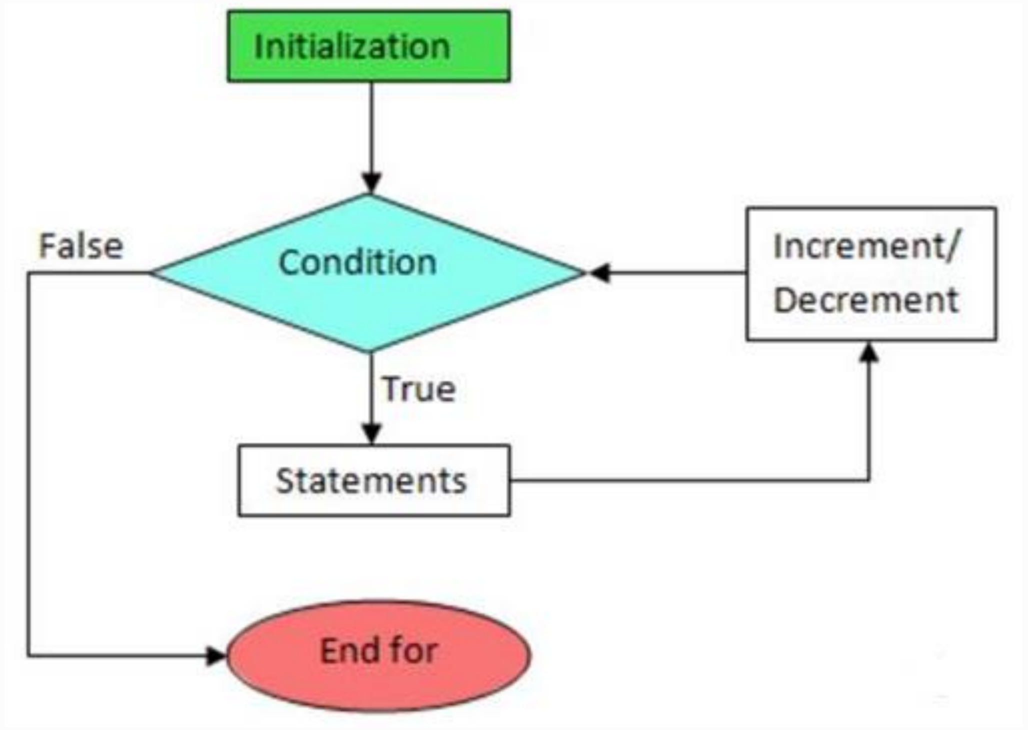
- Introduction to iterative flow control
- Iterative flow controls (Looping statements)
  - ✓ for loop
  - ✓ while loop
  - ✓ do . . . while loop
- Jumping statements
  - ✓ break, continue, goto
- Program termination statements
  - ✓ return, exit, abort

Objectives

- Learn how to use iterative flow control
- Learn how to form Boolean expressions and examine relational and logical operators
- Design and develop program using loop statements

1. Introduction to looping

- 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



1. Introduction to looping (cont'd)

Part of 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.

1. Introduction to looping (cont'd)

Types of loop

- Most programming language provides the 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.

1. Introduction to looping (cont'd)

Category of loops

1) Pretest and Posttest loops

- **Pretest loops (while loop & for loop) -** the loop condition checked first, if **false,** statements in the loop body never executed.
- **Posttest loop (do .. while loop) -** the loop condition is checked/tested after the loop body statements are executed.
- Loop body always executed at least once

2) Count-controlled and Event-Controlled loops

- **Count-controlled (for loop) –** also called fixed count loop
  - ✓ Repeat a statement or block a specified number of times
  - ✓ *Used when exactly how many loops want to made*
- **Event-controlled (while and do-while loop) –** also called variable condition loop
  - ✓ Repeat a statement or block until a condition within the loop body changes that cause the repetition to stop.

1. Introduction to looping (cont'd)

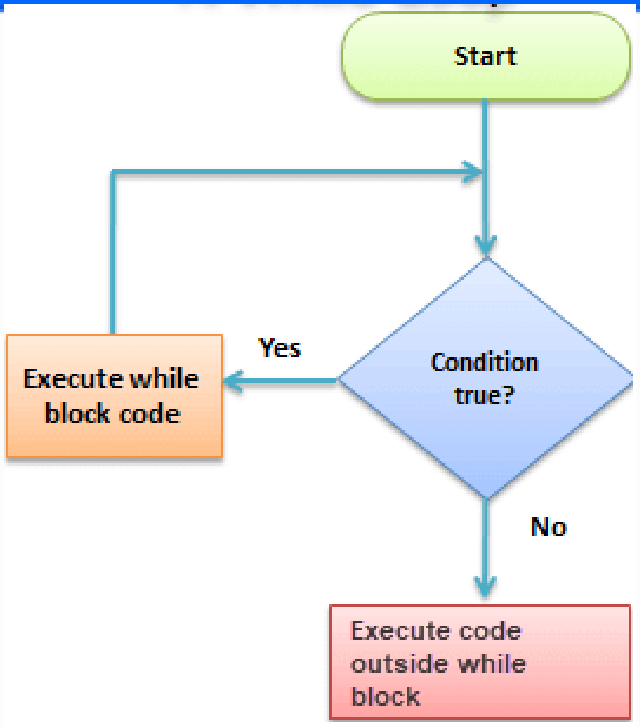
Types of Event-Controlled Loops

- **Sentinel controlled**
  - ✓Keep processing data until a special value (*sentinel value*) that is not a possible data value is entered to indicate that processing should stop.
- **End-of-file controlled**
  - ✓Keep processing data or executing statement(s) as long as there is more data in the file.
- **Flag controlled**
  - ✓Keep processing data until the value of a flag changes in the loop body

2. while loop

- **Syntax**

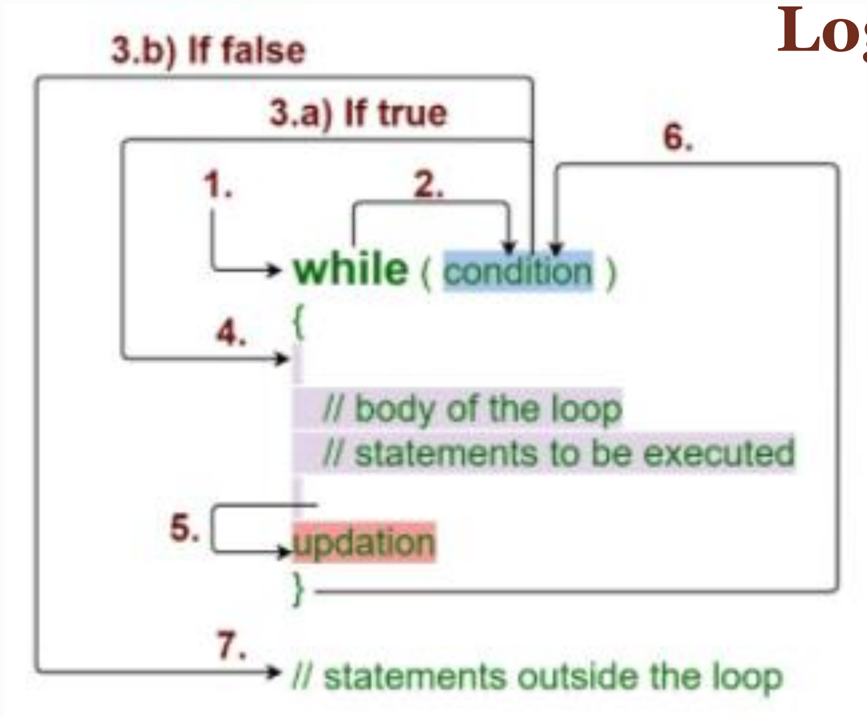
```
while (repetition condition) {
      statement (s);
  }
next statement(s);
```

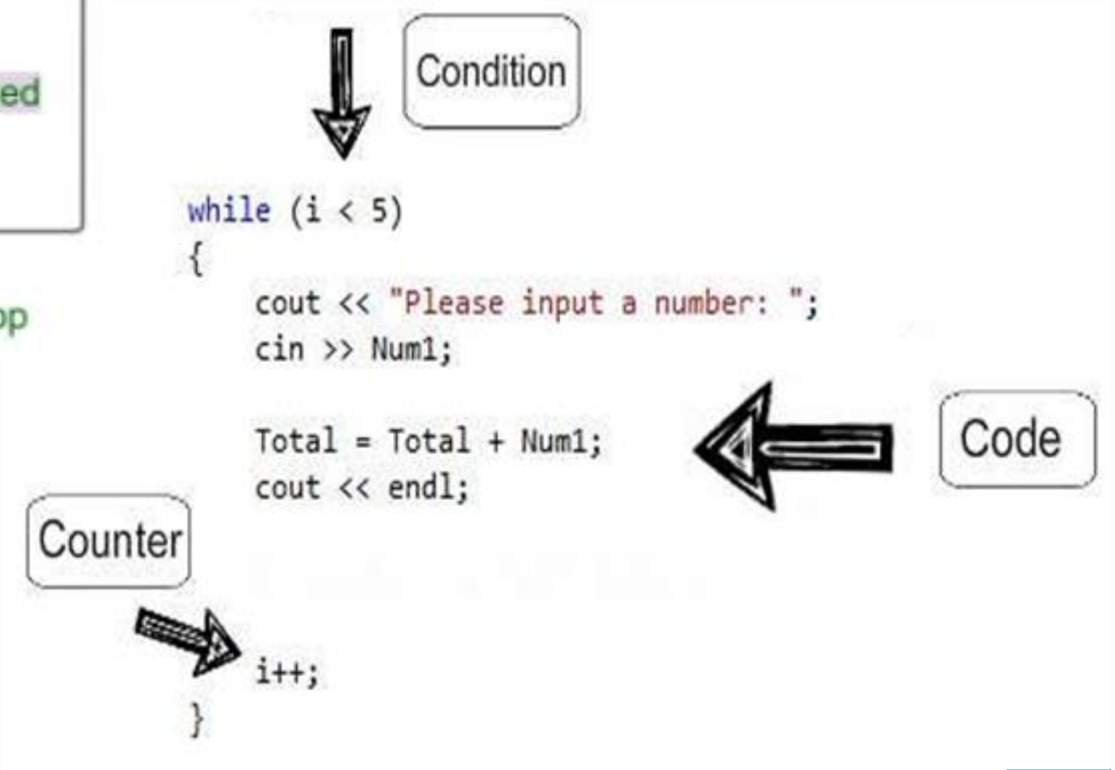


- **Repetition condition**
  - ✓ It is the condition which controls the loop
  - ✓ Must evaluated to true/false (i.e. Boolean expression)
  - ✓ Can be formed by combining two or more relational expression with logical operators
- The ***statement*** is repeated as long as the loop repetition condition is **true**.
- **infinite loop -** if the loop repetition condition is always true.

2. while loop (cont'd)

Logic of a while loop



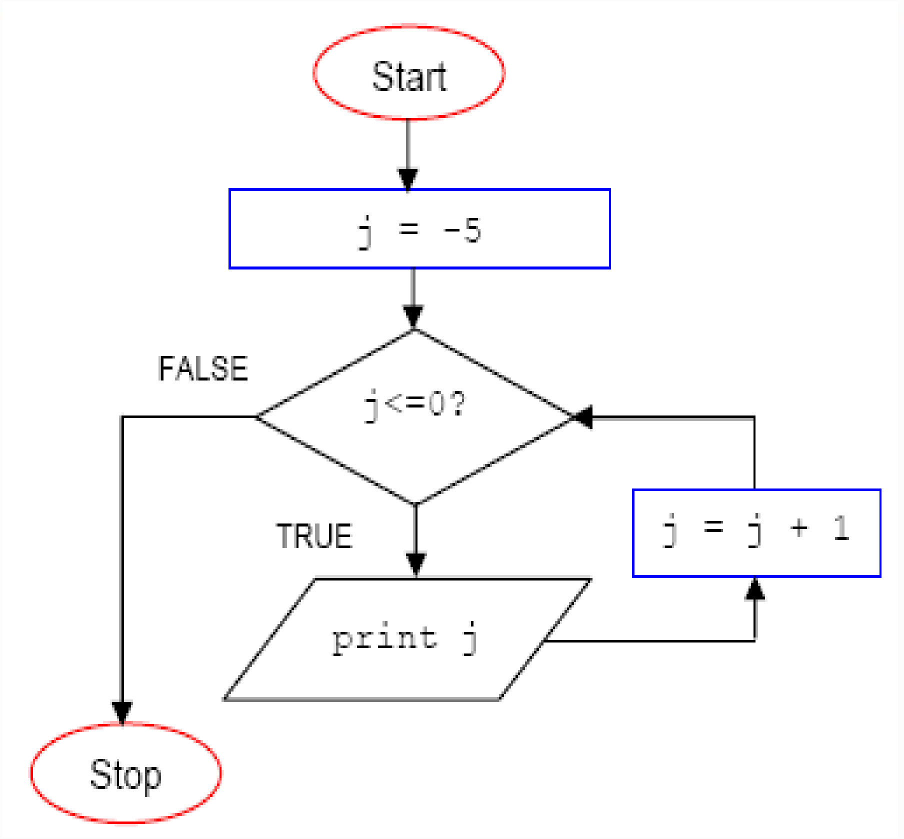


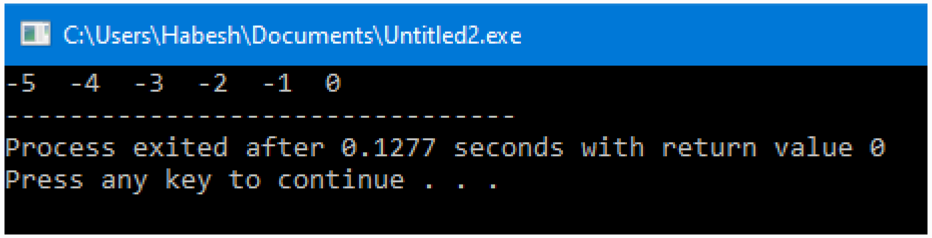
```
while (i < 5)
{
     cout << "Please input a number: ";
     cin >> Num1;

     Total = Total + Num1;
     cout << endl;


     i++;
}
```

2. while loop (cont'd)

EXAMPLE:

```cpp
#include <iostream>
using namespace std;

int main(){
    int j;
    j = -5;

    while(j <= 0)
    {
        cout<<j<<"  ";
        j = j + 1;
    }
    return 0;
}
```




C:\Users\Habesh\Documents\Untitled2.exe

```
-5  -4  -3  -2  -1  0
--------------------------------
Process exited after 0.1277 seconds with return value 0
Press any key to continue . . .
```

3. for loop

- **Syntax**

The *initialization* is executed once before the loopbegins

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

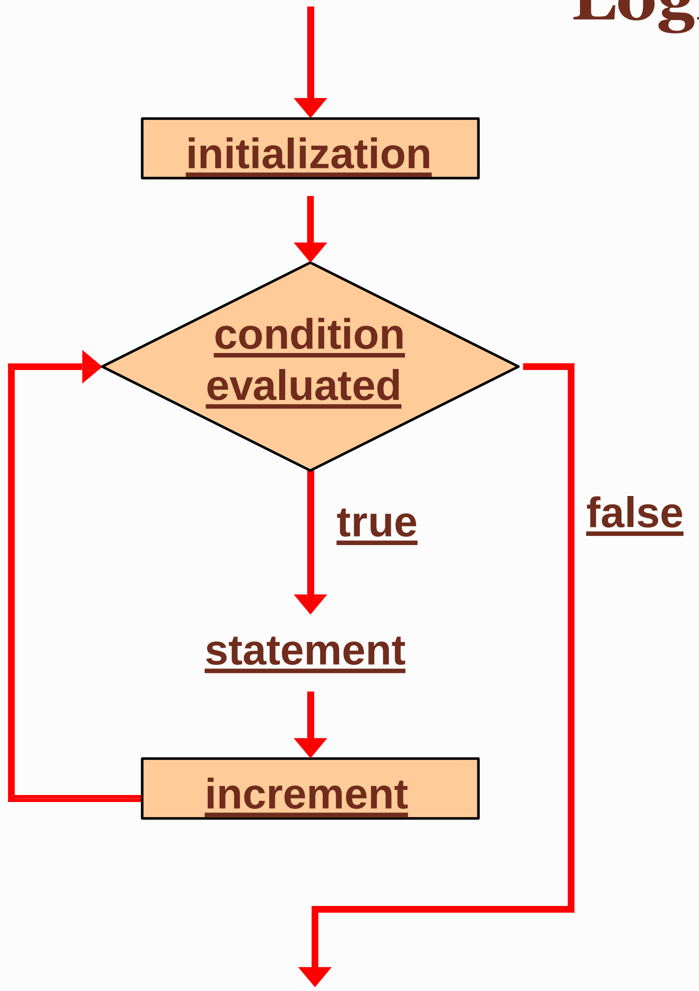
```
for ( initialization ; condition ; increment )
{
    statement;
}
```

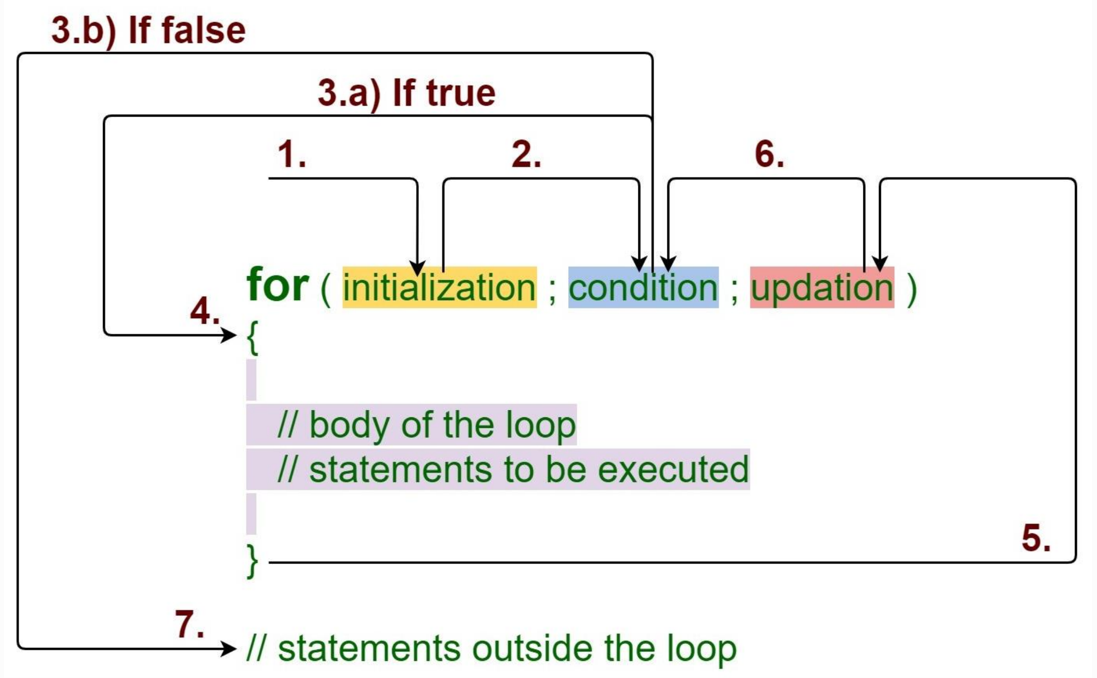
The *increment* portion is executed at the end of each iteration

- **Condition**
  - ✓ controls the loop and must evaluated to true/false
  - ✓ Can be formed by combining two or more relational expression with logical operators
- The ***statement*** is repeated as long as the loop repetition condition is **true**

3. for loop (cont'd)

Logic of a for loop



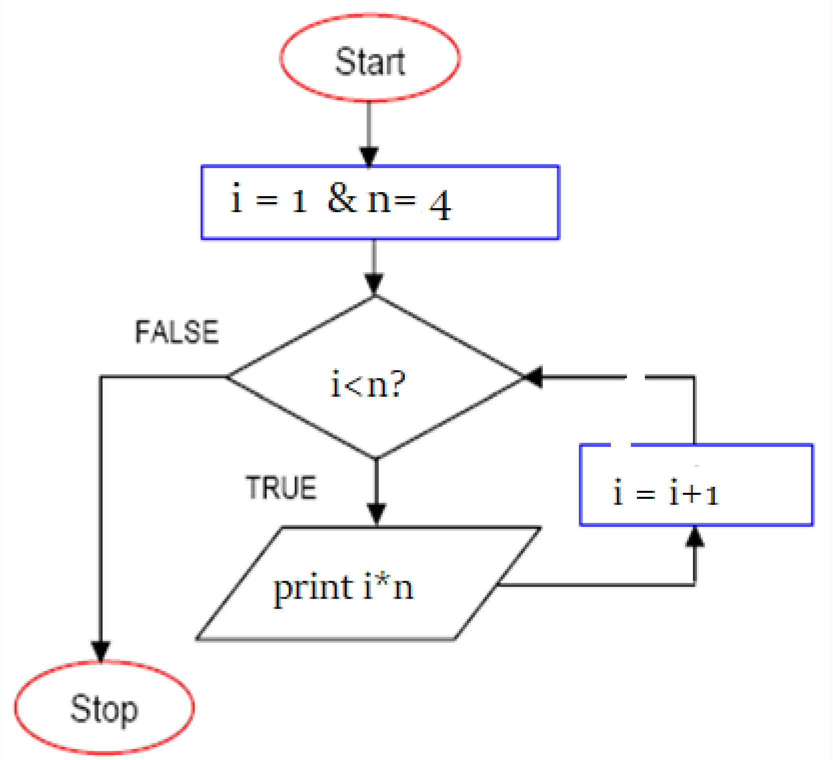


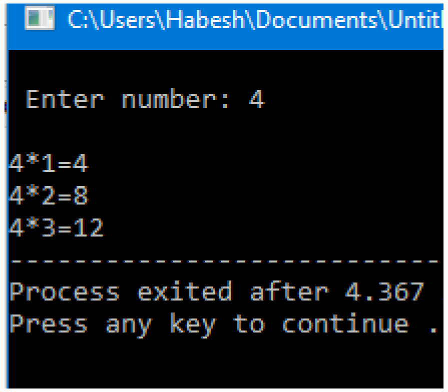
```
for ( initialization ; condition ; updation )
{

  // body of the loop
  // statements to be executed

}
// statements outside the loop
```

3. for loop (cont'd)

EXAMPLE:

```cpp
#include <iostream>
using namespace std;

int main(){
	//program to display table of a
	//given number using for loop.

	int n;
	cout<<"\n Enter number: ";
	cin>>n;

	for(int i=1;i<n;++i)
		cout<<"\n"<<n<<"*"<<i<<"="<<n*i;

	return 0;
}
```




C:\Users\Habesh\Documents\Untit

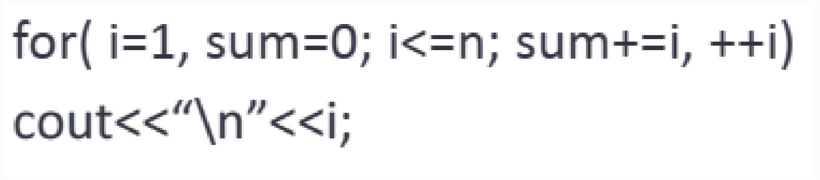
```
 Enter number: 4

4*1=4
4*2=8
4*3=12
--------------------------------
Process exited after 4.367
Press any key to continue .
```

3. for loop (cont'd)

The for loop Variations

a) 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.
- ✓ Example:



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

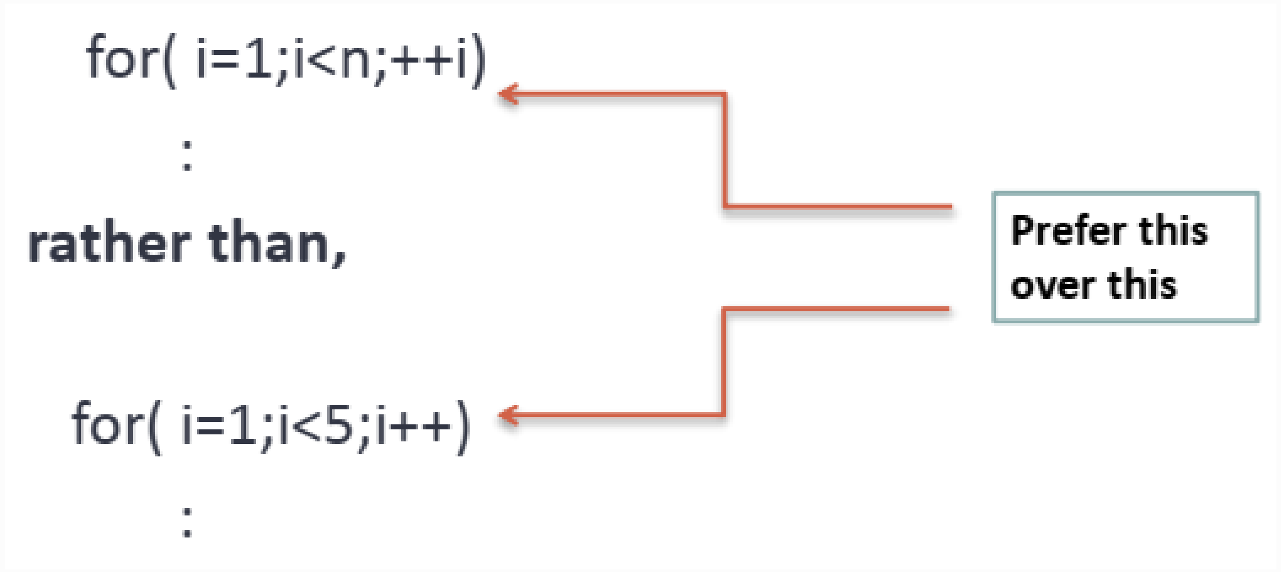
b) Other for loop forms

for (;n < 10;)	if we wanted to specify no initialization and no update expression
for (; n<10; n++)	if we wanted to include an update expression but no initialization (maybe because the variable was already initialized before).

for (;;)
for(j=25; ;--j)
	infinite loop:- Removing either all the expressions or missing condition or using condition that never get false gives us an infinite loop

3. for loop (cont'd)

Prefix or postfix increment/decrement



```
   for( i=1;i<n;++i)
           :
rather than,

  for( i=1;i<5;i++)
           :
```

Prefer this over this

- ✓ Reason being that when used alone, prefix operators are faster executed than postfix

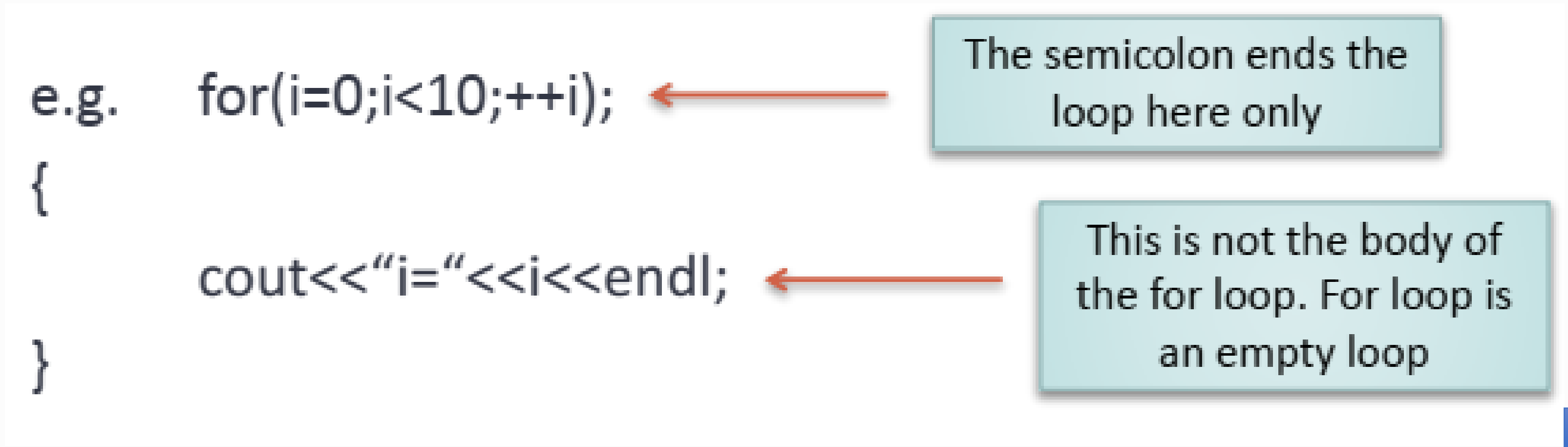
3. for loop (cont'd)

Empty loop

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

```
for(j=25; (j);--j)   //(j) tests for non zero value of j.
```

- ✓ 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;
}
```

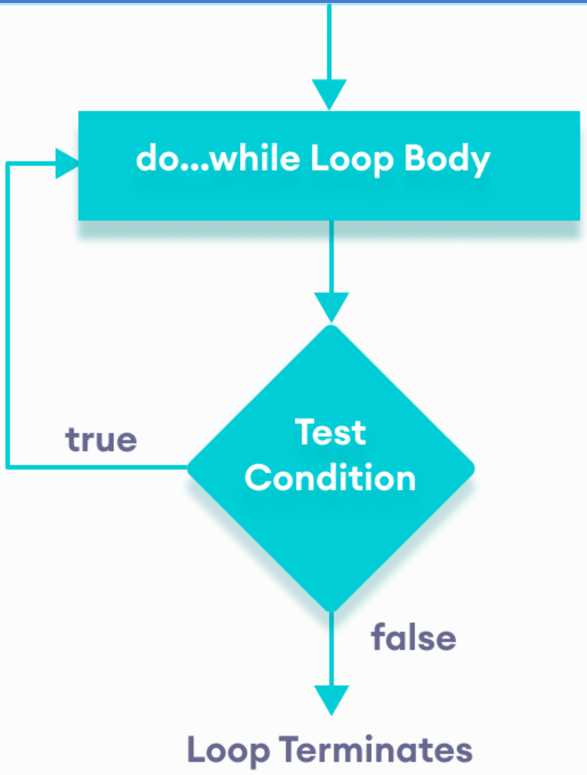
The semicolon ends the loop here only

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

4. do . . . while loop

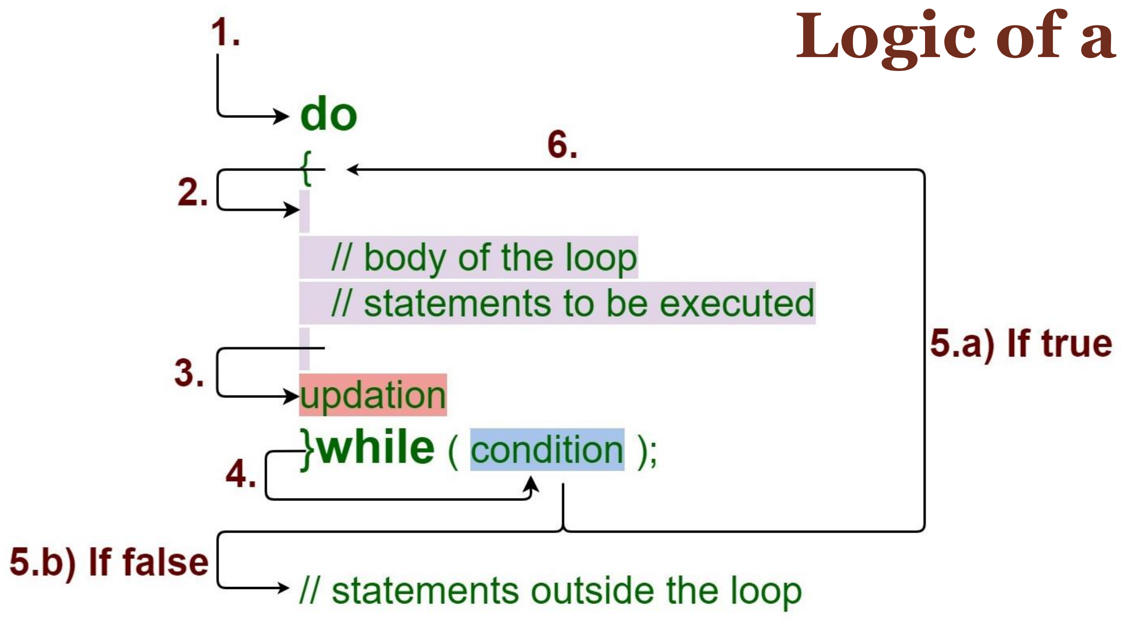
- **Syntax**

```
do {
      statement (s);
  } while (repetition condition)
 next statement(s);
```

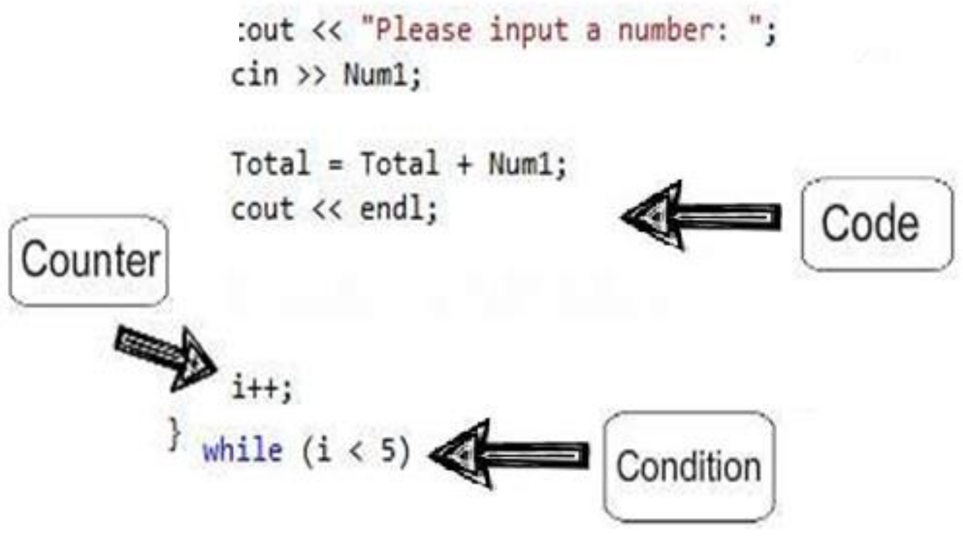


- It is similar to while loop except it is **posttest loop**
- The ***statement*** is first executed.
- If the **loop repetition condition** is true, the ***statement*** is repeated.
- Otherwise, the loop is exited.
- The repetition condition should be Boolean expression
- Used when your program need to be executed at least one iteration

4. do . . . while loop (cont'd)



Logic of a do . . while loop
1.
do
{
2.
// body of the loop
// statements to be executed
3.
updation
}while (condition);
4.
6.
5.a) If true
5.b) If false
// statements outside the loop

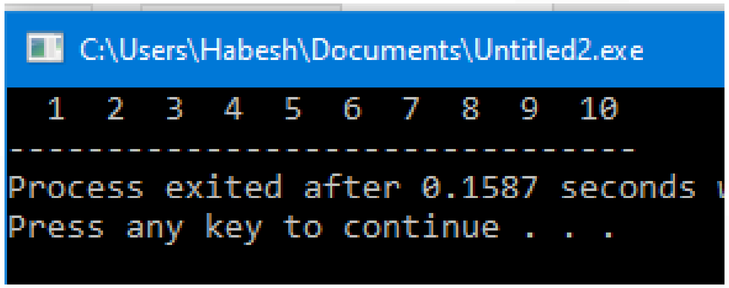


```
:out << "Please input a number: ";
cin >> Num1;

Total = Total + Num1;
cout << endl;

i++;
} while (i < 5)
```

4. do . . . while loop (cont'd)

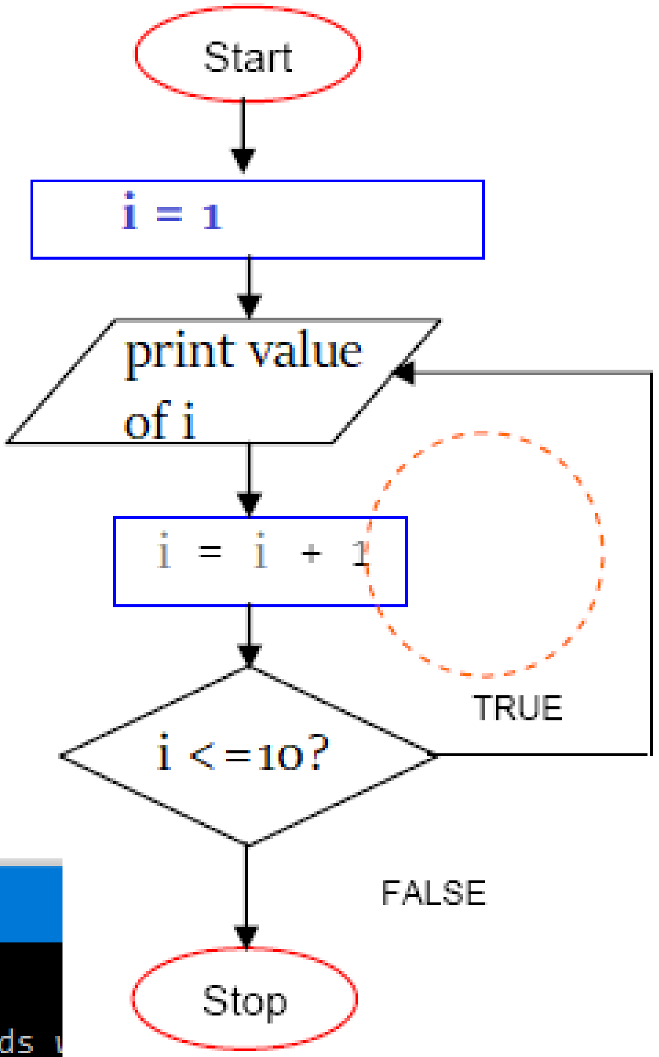
EXAMPLE:

```cpp
#include <iostream>
using namespace std;

int main(){
    //program to display counting
    //from 1 to 10 using do-while loop.

    int i=1;
    do{
        cout<<"  "<<i;
        i++;
    }while(i<=10);

    return 0;
}
```



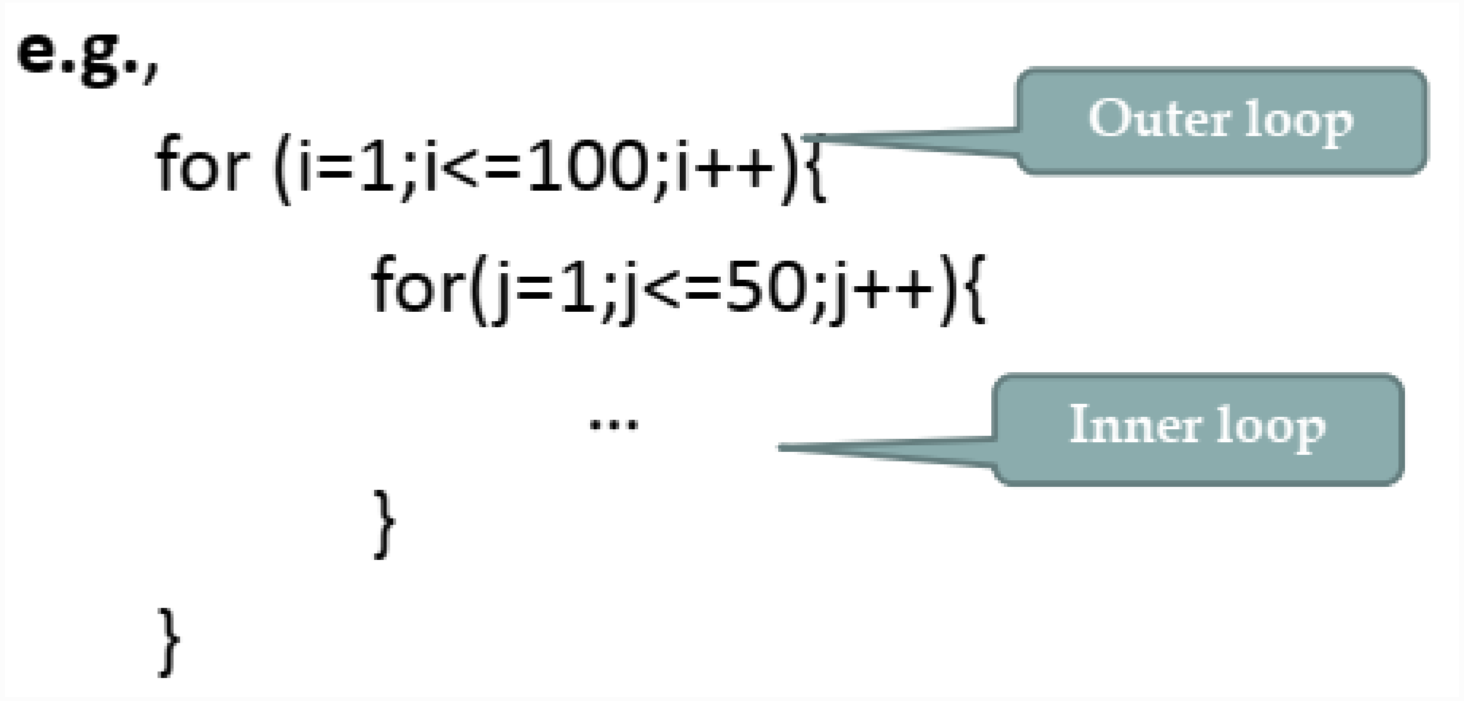
C:\Users\Habesh\Documents\Untitled2.exe

```
 1  2  3  4  5  6  7  8  9  10
--------------------------------
Process exited after 0.1587 seconds w
Press any key to continue . . .
```



5. Nested loop

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



e.g.,

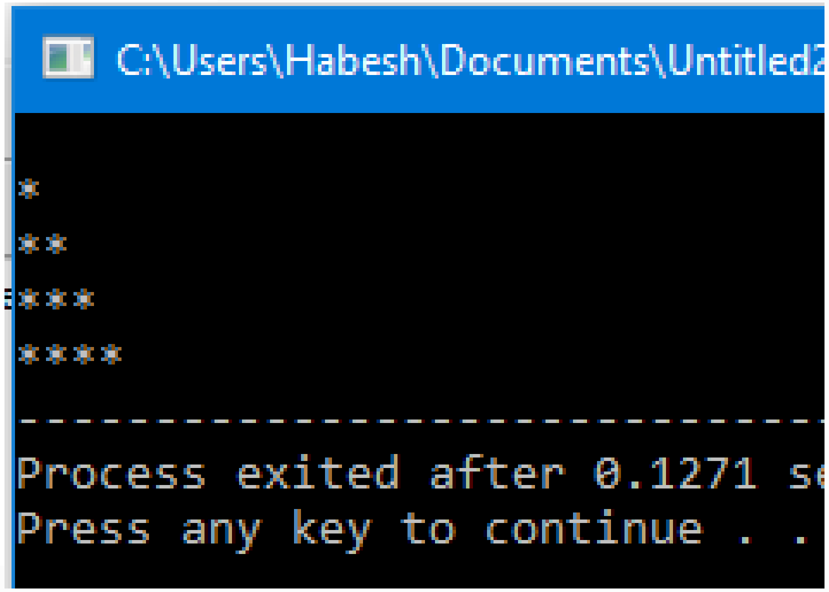
```
    for (i=1;i<=100;i++){
            for(j=1;j<=50;j++){
                    …
            }
    }
```

The above loop will run for 100*50 iterations

5. Nested loop (cont'd)

EXAMPLE:

```cpp
#include <iostream>
using namespace std;

int main(){

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

    int i,j;
    for( i=1;i<5;++i)
    {
        cout<<endl;
        for(j=1;j<=i;++j)
            cout<<"*";
    }

    return 0;
}
```



C:\Users\Habesh\Documents\Untitled2

```

*
**
***
****

--------------------------------
Process exited after 0.1271 se
Press any key to continue . .
```

6. Jumping Statements

(a) The goto statement

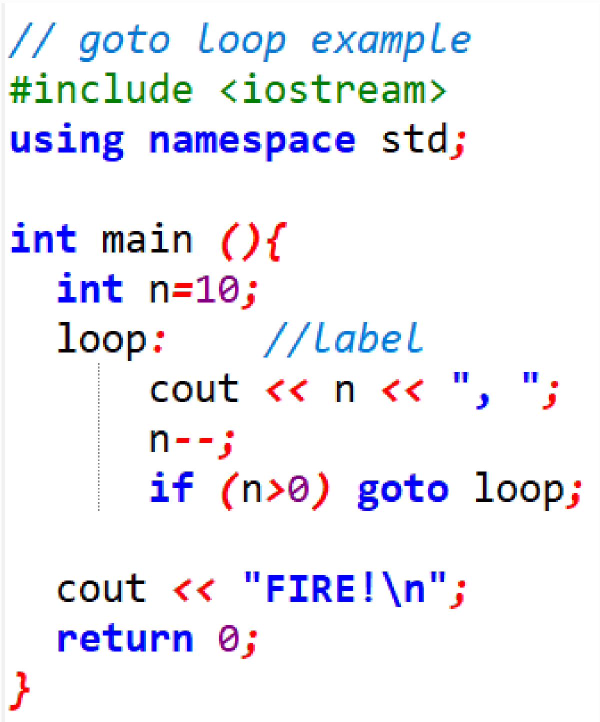
- ✓ It can transfer the program control anywhere in the program.
- ✓ The target destination is marked by a ***label***.
- ✓ The target ***label*** and **goto** must appear in the same statement.
- ✓ The syntax:

goto *label*;

………

………

label:



```
// goto loop example
#include <iostream>
using namespace std;

int main (){
  int n=10;
  loop:     //label
      cout << n << ", ";
      n--;
      if (n>0) goto loop;

  cout << "FIRE!\n";
  return 0;
}
```

6. Jumping Statements (cont'd)

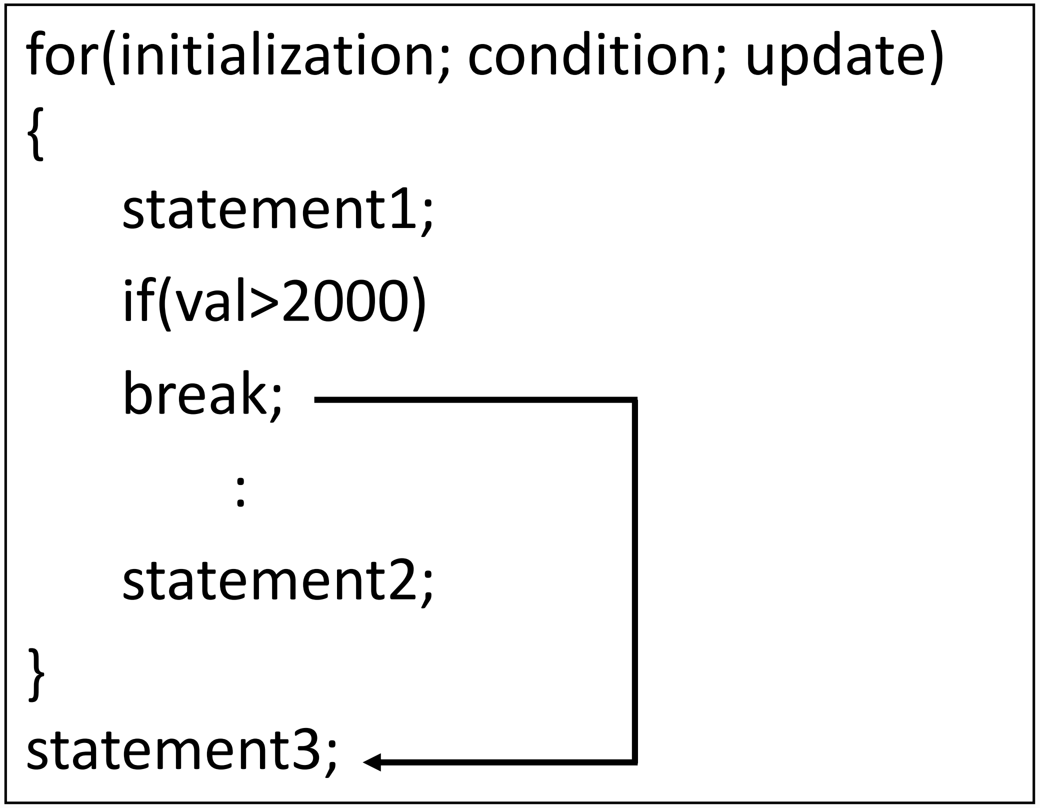
(b) The break statement

- ✓ Enables a program to skip over part of the code.
- ✓ It terminates the smallest enclosing while, do-while and for loop statements.
  - ➢ It skips the rest of the loop and jumps over to the statement following the loop.
- ✓ The figures on the next slide explains the working of a break statement :
- ✓ Aslo use along with switch as discussed under the selection control section
- ✓ **Syntax**:

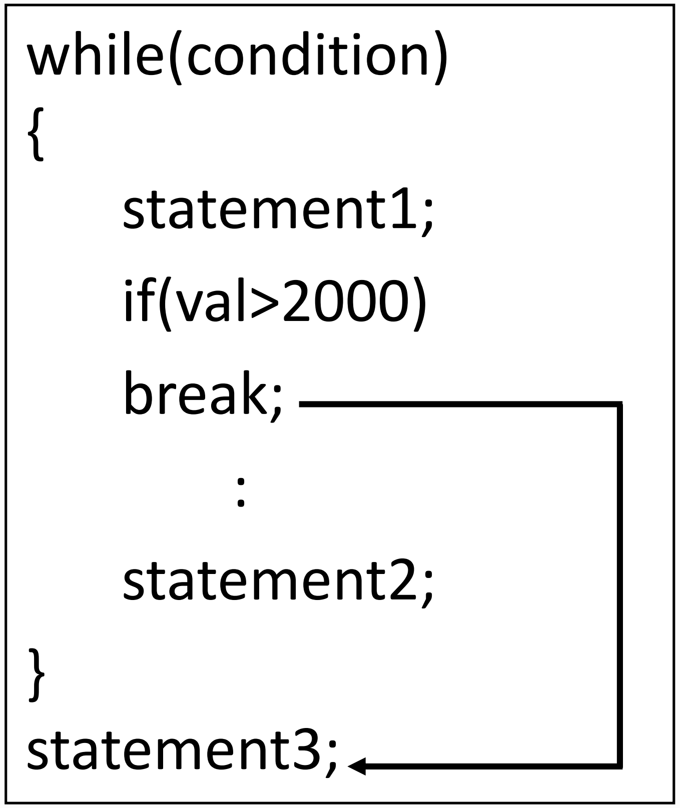
  break;

6. Jumping Statements (cont’d)

How break statement works with loops



```
for(initialization; condition; update)
{
      statement1;
      if(val>2000)
      break;
             :
      statement2;
}
statement3;
```



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

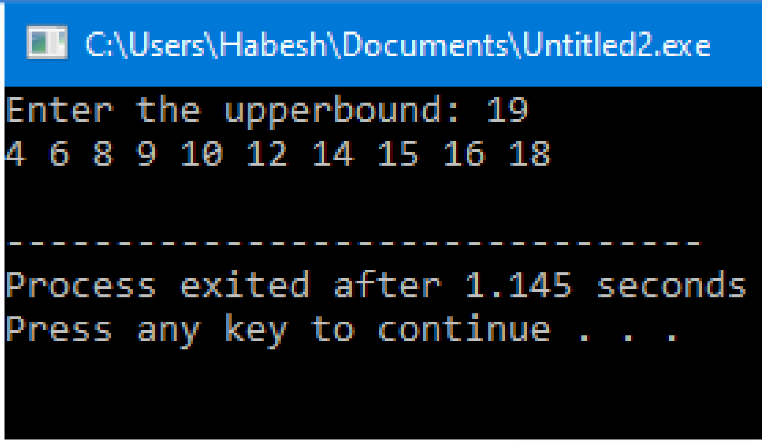
Note:

- **The break statement can be used in similar fashion with do…while loop also**

6. Jumping Statements (cont'd)

Example of break statement

```cpp
//pogram to list non-prime from 1 to an upperbound
#include <iostream>
#include <cmath>
using namespace std;

int main(){
   int upperbound;
   cout << "Enter the upperbound: ";
   cin >> upperbound;
   for (int number = 2; number <= upperbound; ++number)
     {
      // Not a prime, if there is a factor between 2 and sqrt(number)
      int maxFactor = (int)sqrt(number);
      for (int factor = 2; factor <= maxFactor; ++factor)
       {
         if (number % factor == 0)  // Factor?
        {
            cout << number << " ";
            break;   // A factor found, no need to search for more factors
         }
      }
   }
   cout << endl;
   return 0;
}
```



C:\Users\Habesh\Documents\Untitled2.exe

```
Enter the upperbound: 19
4 6 8 9 10 12 14 15 16 18


--------------------------------
Process exited after 1.145 seconds
Press any key to continue . . .
```

6. Jumping Statements (cont'd)

(c) The continue statement

- ✓ Enables a program to skip over part of the code.
- ✓ works somewhat like the **break** statement.
- ✓ For the for loop, continue causes the conditional test and increment portions of the loop to execute.
- ✓ For the while and do...while loops, program control passes to the conditional tests.
- ✓ **Syntax**:

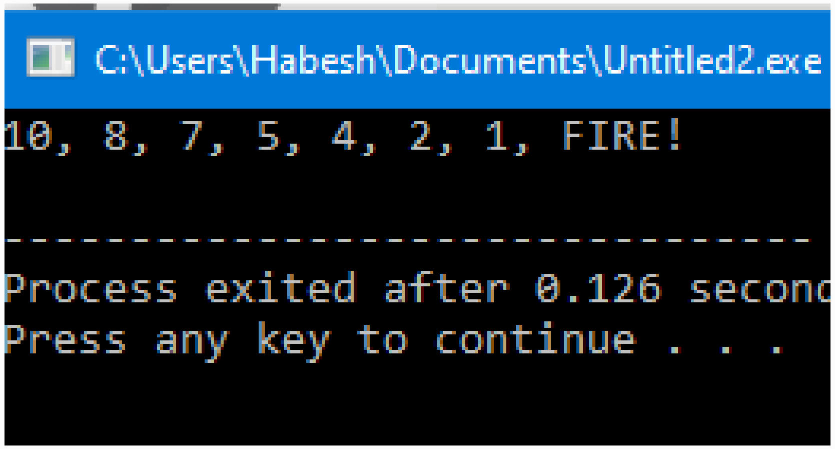
```
continue;
```

6. Jumping Statements (cont'd)

Example of continue statement

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n%3 == 0)
       continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

As you can see on the output below the program jumps printing 3 & 6 which are factor of 3



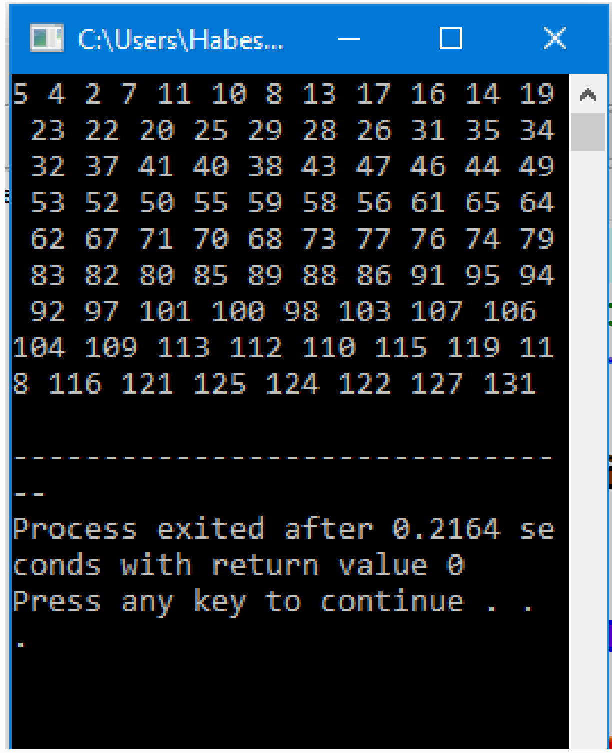
C:\Users\Habesh\Documents\Untitled2.exe

```
10, 8, 7, 5, 4, 2, 1, FIRE!


--------------------------------
Process exited after 0.126 seconc
Press any key to continue . . .
```

6. Jumping Statements (cont'd)

Examples (break and continue

```
#include <iostream>
using namespace std;

int main()
 {
   int number = 1;
   while (true)
    {
      ++number;
      if ((number % 3) == 0) continue;
      if (number == 133) break;
      if ((number % 2) == 0) { number += 3; }
      else {
         number -= 3;
       }
      cout<<number<<" ";
   }
   cout << endl;
   return 0;
}
```



```
5 4 2 7 11 10 8 13 17 16 14 19
 23 22 20 25 29 28 26 31 35 34
 32 37 41 40 38 43 47 46 44 49
 53 52 50 55 59 58 56 61 65 64
 62 67 71 70 68 73 77 76 74 79
 83 82 80 85 89 88 86 91 95 94
 92 97 101 100 98 103 107 106
104 109 113 112 110 115 119 11
8 116 121 125 124 122 127 131


------------------------------
--
Process exited after 0.2164 se
conds with return value 0
Press any key to continue . .
.
```

7. Terminating Program

(a) The return statement

- ✓ As you seen in the main() function it terminate the program and return control back to the Operating System
- ✓ **Syntax:** return returnValue;

(b) The exit() function

- ✓ Used to terminate the program normally and return the control to the Operating System.
- ✓ **Syntax:** exit(int exitCode);
- ✓ Avaliable in <cstdlib> library (ported from C's "stdlib.h")

(c) The abort() function

- ✓ The same as exit() function but except it used to terminate the program *abnormally.*
- ✓ **Syntax:** abort(int exitCode);

7. Terminating Program

Example

```
if (errorCount > 10)
{
   cout << "too many errors" << endl;
   return 1;
}
```

```
if (errorCount > 10)
{
   cout << "too many errors" << endl;
   exit(-1);  // Terminate the program
              // also you can use abort(-1);  instead to
               terminate the program abnormally
}
```

Exercises (MCQ)

(1) The statement i++; is equivalent to

(a) i = i + i; (b) i = i + 1; (c) i = i - 1; (d) i --;

(2) What's wrong? for (int k = 2, k <=12, k++)

(a) the increment should always be ++k
(b) the variable must always be the letter i when using a for loop
(c) there should be a semicolon at the end of the statement
(c) the commas should be semicolons

(3) A looping process that checks the test condition at the end of loop?

(a) for while (b) do-while (c) while (d) none

(4) A looping process is best used when the number of iterations is known

(a) for while (b) do-while (c) while (d) all are require

Exercises (MCQ)

(5) A **continue** statement causes execution to skip to

(a) The return 0; statement

(b) The first statement after the loop

(c) The statement following the continue statement

(d) The next iteration of the loop

(6) A **break** statement causes execution to skip to

(a) The return 0; statement

(b) The first statement after the loop

(c) The statement following the continue statement

(d) The next iteration of the loop

(e) The statement outside the loop

Reading Resources/Materials

Chapter 7 & 8:

- ✓ **Diane Zak**; An Introduction to Programming with C++ [8th Edition], 2016 Cengage Learning

Chapter 5:

- ✓ **Gary J. Bronson**; C++ For Engineers and Scientists [3rd edition], Course Technology, Cengage Learning, 2010

Chapter 2 (section 2.4):

- ✓ **Walter Savitch**; Problem Solving With C++ [10th edition], University of California, San Diego, 2018

Chapter 4 & 5:

- ✓ **P. Deitel , H. Deitel**; C++ how to program [10th edition], Global Edition (2017)

Thank You
For Your Attention!!