



CS1002 – Programming Fundamentals

Lecture # 15
Monday, October 17, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Muhammad Yousaf



Outline

- Control Structures (**Repetition**/Loops)
- Types of Repetition Structures
 - Pre-test (while, for)
 - Post-test (do-while)
- While Loop
 - Counter controlled While loop
 - Sentinel controlled While loop
 - Flag controlled While loop
 - EOF (end of file) controlled While loop
 -
- For Loop
- Do While Loop
- Break & Continue Statements

Why Is Repetition Needed?

- Repetition allows you to **efficiently** use **variables**
- Can input, add, and take average of multiple numbers using a limited number of variables
- For example, to add five numbers:
 - Declare a variable for each number, input the numbers and add the variables together
 - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

while Looping (Repetition) Structure

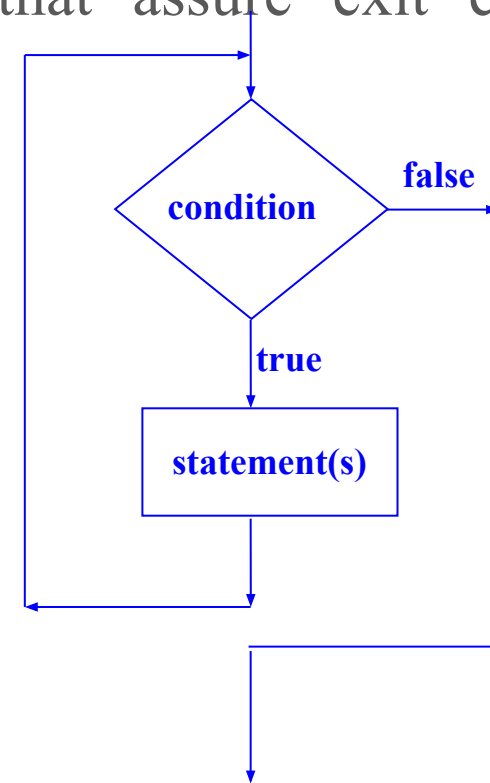
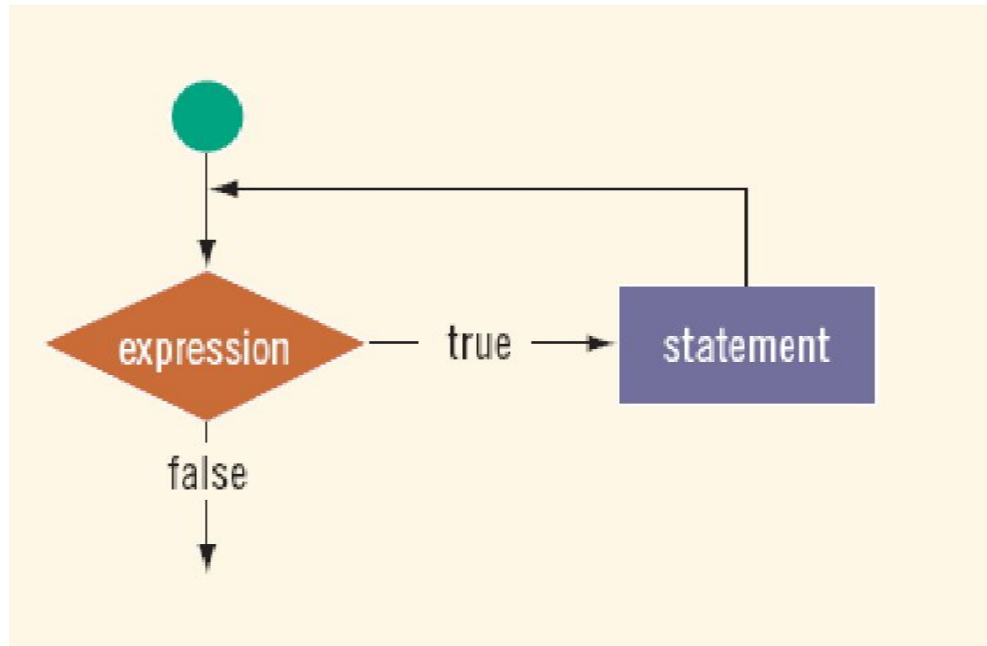
- The general form of the **while** statement is:

```
while (expression)  
    statement
```

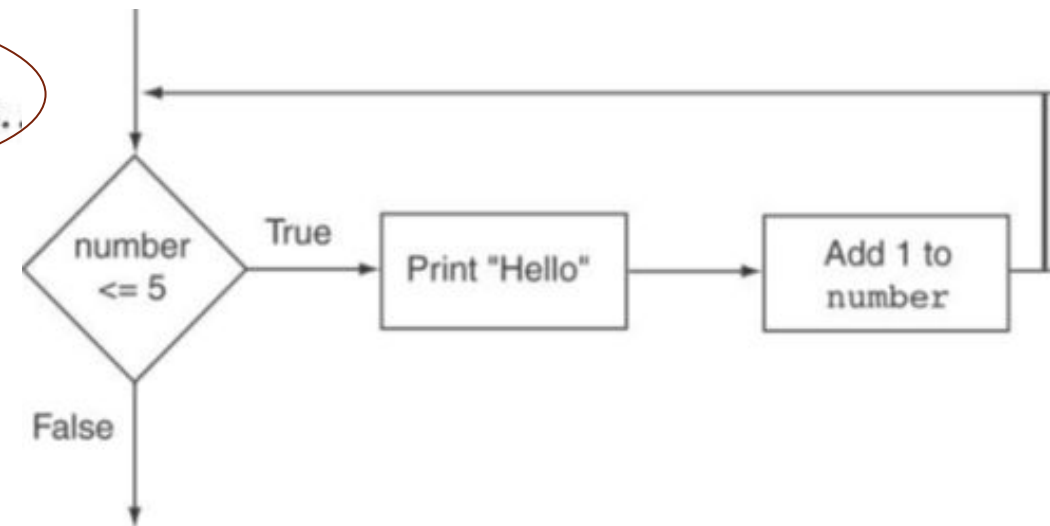
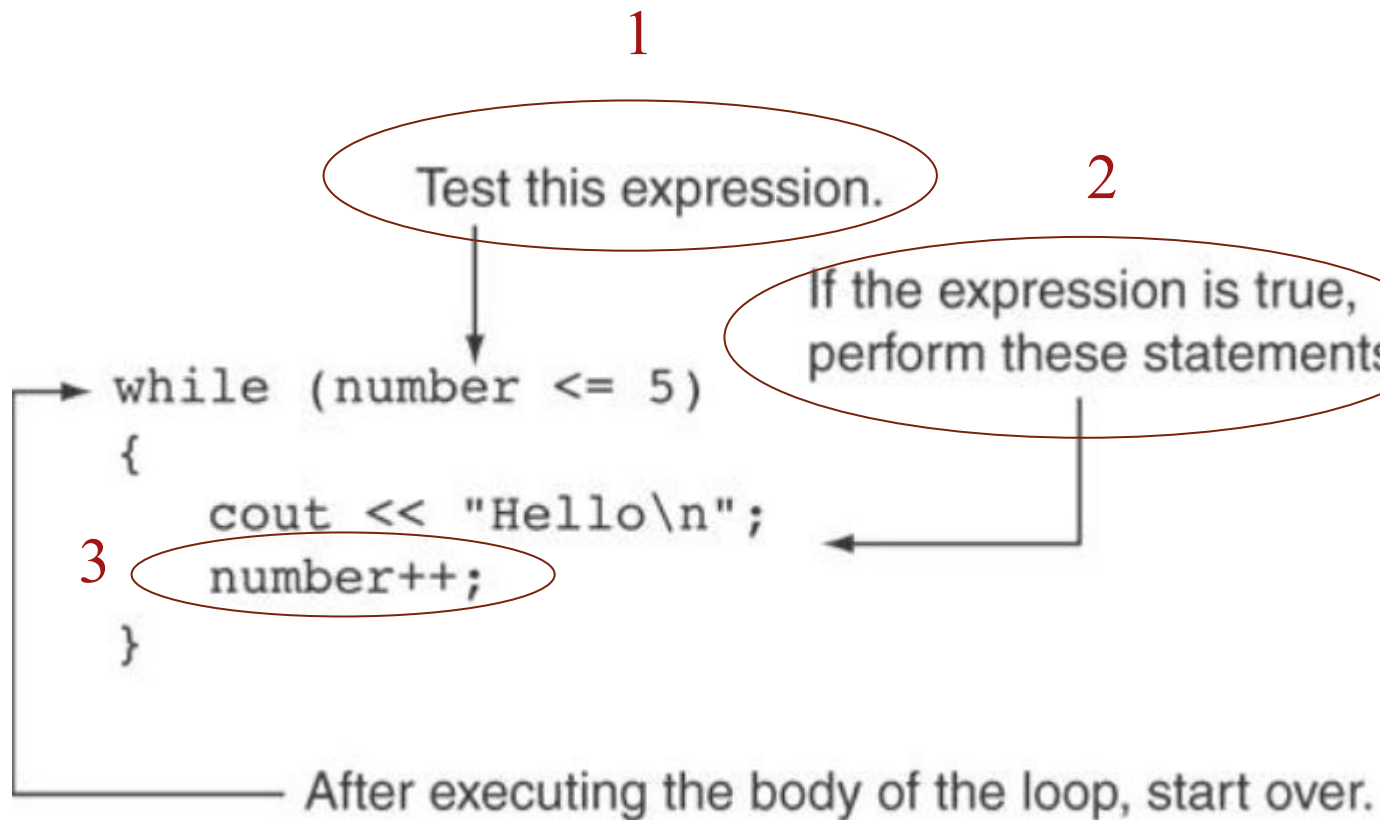
- **while** is a **reserved** word
- Statement can be simple or compound
- Expression acts as a decision maker and is usually a logical expression
- **while** is a **pre-test** loop
- Statement is called the **body** of the loop
- The parentheses are part of the syntax

while Loop Flow of Control

- **Infinite loop:** continues to execute endlessly
- Avoided by including statements in loop body that assure exit condition is eventually false



Understanding while



while Looping (Repetition) Structure (cont'd.)

Initialize
before
using

EXAMPLE 5-1

Consider the following C++ program segment:

```
i = 0; //Line 1
while (i <= 20) //Line 2
{
    cout << i << " "; //Line 3
    i = i + 5; //Line 4
}
```

Loop Control
Variable

cout << endl;

Sample Run:

0 5 10 15 20

Omit
this

25 is not printed Why?

Examples

```
i = 0;
while (i <= 20)
{
    cout << i << " ";
    i = i + 5 ;
}
cout << endl ;
```

Here is the output

0 5 10 15 20

```
i = 0;
while (i <= 20)
{
    i = i + 5 ;
    cout << i << " ";
}
cout << endl ;
```

//What will be the output

5 10 15 20 25

Designing while Loops

EXAMPLE 5-2

Consider the following C++ program segment:

```
i = 20; //Line 1
while (i < 20) //Line 2
{
    cout << i << " "; //Line 3
    i = i + 5; //Line 4
}
cout << endl; //Line 5
```

It is easy to overlook the difference between this example and Example 5-1. In this example, in Line 1, *i* is set to 20. Because *i* is 20, the expression *i* < 20 in the **while** statement (Line 2) evaluates to **false**. Because initially the loop entry condition, *i* < 20, is **false**, the body of the **while** loop never executes. Hence, no values are output and the value of *i* remains 20.



Types of while loops

- Counter-controlled while loop
- Sentinel-controlled while loop
- Flag-Controlled while loop

Counter-Controlled **while** Loops

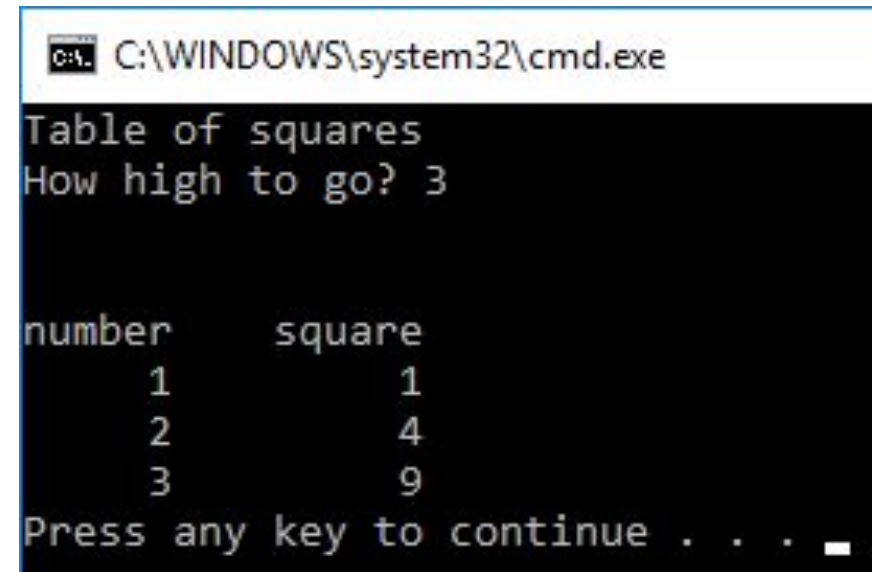
- If you know exactly how many pieces of data need to be read.
While loop becomes a counter-controlled loop with **loop control variable (LCV)**

```
int counter = 0 ; //initialize the loop control variable
while (counter < N) //test the loop control variable
{
    .
    .
    .
    counter++ ; //update the loop variable
    .
    .
    .
}
```

User Controls the Loop Example

```
int num, limit;
std::cout << "Table of squares\n";
std::cout << "How high to go? ";
std::cin  >> limit;
std::cout << "\n\n number" << std::setw(10) << "square" << std::endl;
num = 1;
while (num <= limit)
{
    std::cout << std::setw(6) << num << std::setw(10)
               << num*num << std::endl;

    num++;
}
```



```
C:\WINDOWS\system32\cmd.exe
Table of squares
How high to go? 3

number      square
    1         1
    2         4
    3         9
Press any key to continue . . .
```

Case 2: Sentinel-Controlled while Loops

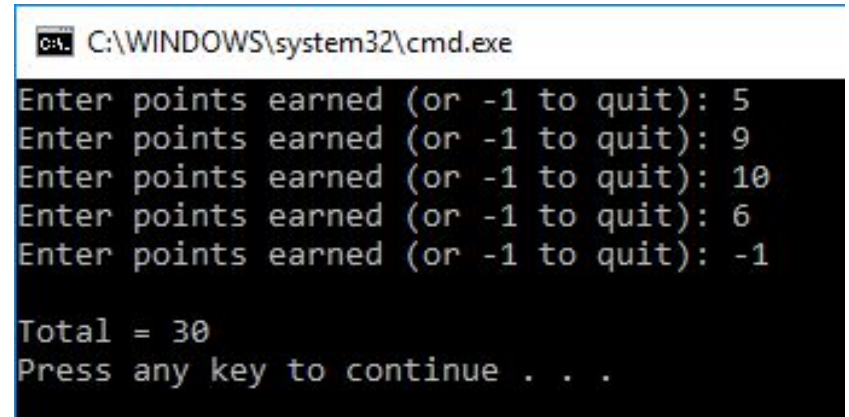
- The number of iterations is **not known** before the loop starts executing
- **Sentinel** is a special value of variable which changes the loop condition true to false
- Loop ends when **sentinel** value is encountered
 - Reading from a file till end-of-file eof()
 - Get input from user until a -ve number is input

When to Use: Sentinel controlled loop is useful when we don't know in advance how many times the loop will be executed.

```
std::cin >> variable ;           //initialize the loop variable
while(variable != sentinel) //test the loop control variable
{
    .
    .
    std::cin >> variable ; //update the loop control variable
    .
    .
}
```

Sentinel Example

```
int total = 0;
int points;
std::cout << "Enter points earned (or -1 to quit): ";
std::cin  >> points;
while (points != -1) // -1 is the sentinel
{
    total += points;
    std::cout << "Enter points earned (or -1 to quit): ";
    std::cin  >> points;
}
std::cout << "\nTotal = " << total << std::endl;
```



```
C:\WINDOWS\system32\cmd.exe
Enter points earned (or -1 to quit): 5
Enter points earned (or -1 to quit): 9
Enter points earned (or -1 to quit): 10
Enter points earned (or -1 to quit): 6
Enter points earned (or -1 to quit): -1

Total = 30
Press any key to continue . . .
```

Comparison of Sentinel and Counter Loops

S.NO	BASIS OF COMPARISON	SENTINEL CONTROLLED LOOP	COUNTER CONTROLLED LOOP
1.	Definition	A sentinel controlled loop is the infinite repetition loop as the number of repetitions is not known before the loop begins executing	A counter controlled loop is the finite repetition loop as the number of repetitions is known before the loop begins executing
2.	Controlling variable	Controlled variable used is known as sentinel variable.	Controlled variable used is known as counter.
3.	Number of iteration	Unknown number of iteration	Known number of iteration.
4.	Value of variable	The value of the variable is not strict i.e. any value	The value of the variable is strict i.e. $num < N$
5.	Limitation of variable	The Limitation of the variable is strict.	The Limitation of the variable is strict also.
6.	Example	A do....while loop is an example of sentinel controlled loop.	A while loop is an example of counter controlled loop.

Case 3: Flag-Controlled while Loops

- A flag-controlled **while** loop uses a bool variable to control the loop
- The flag-controlled **while** loop takes the form:

```
found = false ; //initialize the loop control variable
while( !found) //test the loop control variable
{
    .
    .
    .
    if (expression)
        found = true;
    .
    .
    .
}
```


Example Number guessing game

```
#include "iostream"
#include "cstdlib"
#include "ctime"
using namespace std;
int main() {
    int num;           //Variable to store the random number
    int guess ;        //Variable to store the number guessed by the user
    bool isGuessed;    //Boolean variable to control the loop

    srand(time(0));    //Sets the random number's seed to current time
    num = rand()%100;   //Generates a random number between 0-99
    isGuessed = false;  //Initializes flag to false
    while(!isGuessed) {
        cout << "Enter an integer in the range 0-99 : " ;
        cin >> guess ;
        cout << endl;
        if(guess == num) {
            cout << "You entered the correct number" << endl;
            isGuessed = true;
        }
        else if(guess < num) {
            cout << "Your guess is lower than the number\nGuess again!" <<endl ;
        }
        else {
            cout << "Your guess is higher than the number\nGuess again!" <<endl ;
        }
    }
}
```

More on Expressions in **while** Statements

The expression in a while statement can be complex

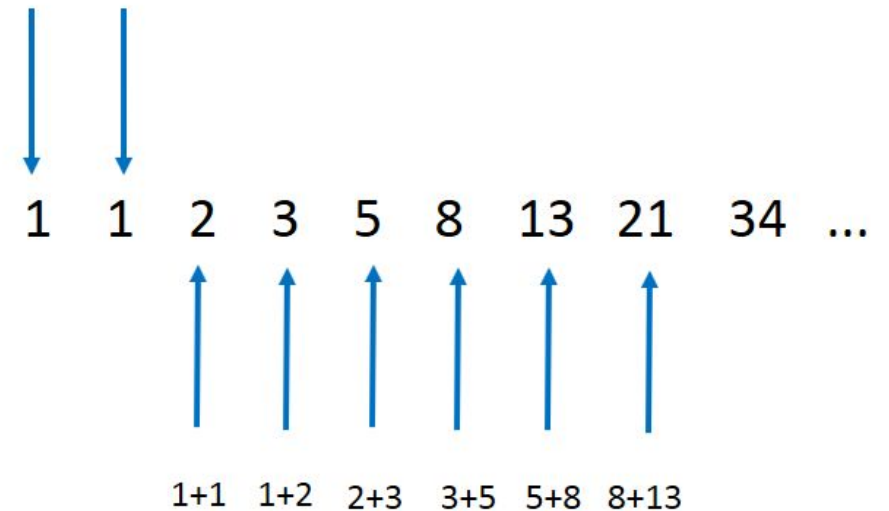
For example:

```
while ((noOfGuesses < 5) && (!isGuessed))  
{  
    ...  
}
```

Example

1. Get the first two Fibonacci numbers.
2. Get the desired Fibonacci number. That is, get the position, n , of the Fibonacci number in the sequence.
3. Calculate the next Fibonacci number by adding the previous two elements of the Fibonacci sequence.
4. Repeat Step 3 until the n th Fibonacci number is found.
5. Output the n^{th} Fibonacci number.

The **first** and **second** numbers in the Fibonacci sequence are 1



```

#include <iostream>
using namespace std;
int main()
{
    //Declare variables
    int previous1, previous2, current;
    int counter, nthFibonacci;
    cout << "Enter the first two Fibonacci
"
    << "numbers: ";
    cin >> previous1 >> previous2;
    cout << endl;
    cout << "The first two Fibonacci "
    << "numbers are "
    << previous1 << " and "
    << previous2 << endl;
    cout << "Enter the position of the "
    << "desired Fibonacci number: ";
    cin >> nthFibonacci;
    cout << endl;
    if (nthFibonacci == 1)
        current = previous1;
    else if (nthFibonacci == 2)
        current = previous2;

```

```

else
{
    counter = 3;
    while (counter <= nthFibonacci)
    {
        current = previous2 + previous1;
        previous1 = previous2;
        previous2 = current;
        counter++;
    } //end while
} //end else
cout << "The Fibonacci number at "
    << "position " << nthFibonacci
    << " is " << current << endl;
    return 0;
} //end main

```

Fibonacci Number Sample Runs

Sample Run 1:

Enter the first two Fibonacci numbers: 12 16

The first two Fibonacci numbers are 12 and 16

Enter the position of the desired Fibonacci number: 10

The Fibonacci number at position 10 is 796

Sample Run 2:

Enter the first two Fibonacci numbers: 1 1

The first two Fibonacci numbers are 1 and 1

Enter the position of the desired Fibonacci number: 15

The Fibonacci number at position 15 is 610

Increment and Decrement Operators

- **Increment operator(++):** increment variable by 1
 - **Pre-increment:** ++variable
 - **Post-increment:** variable++
 - `int j = i++;` // j will contain i, i will be incremented.
 - `int j = ++i;` // i will be incremented, and j will contain i+1.
- **Decrement operator (--):** decrement variable by 1
 - **Pre-decrement:** --variable
 - **Post-decrement:** variable--
- What is the difference between the following?

```
x = 9;  
y = ++x;
```

```
x = 9;  
y = x++;
```

for Looping (Repetition) Structure

- The general form of the for statement is:

```
for (initial statement; loop condition; update statement)  
statement
```

- The **initial statement**, **loop condition**, and **update statement** are called **for** loop control statements
 - **initial statement** usually initializes a variable (called the **for loop control**, or **for indexed variable**)
- In C++, **for** is a reserved word

for Looping (Repetition) Structure (cont'd.)

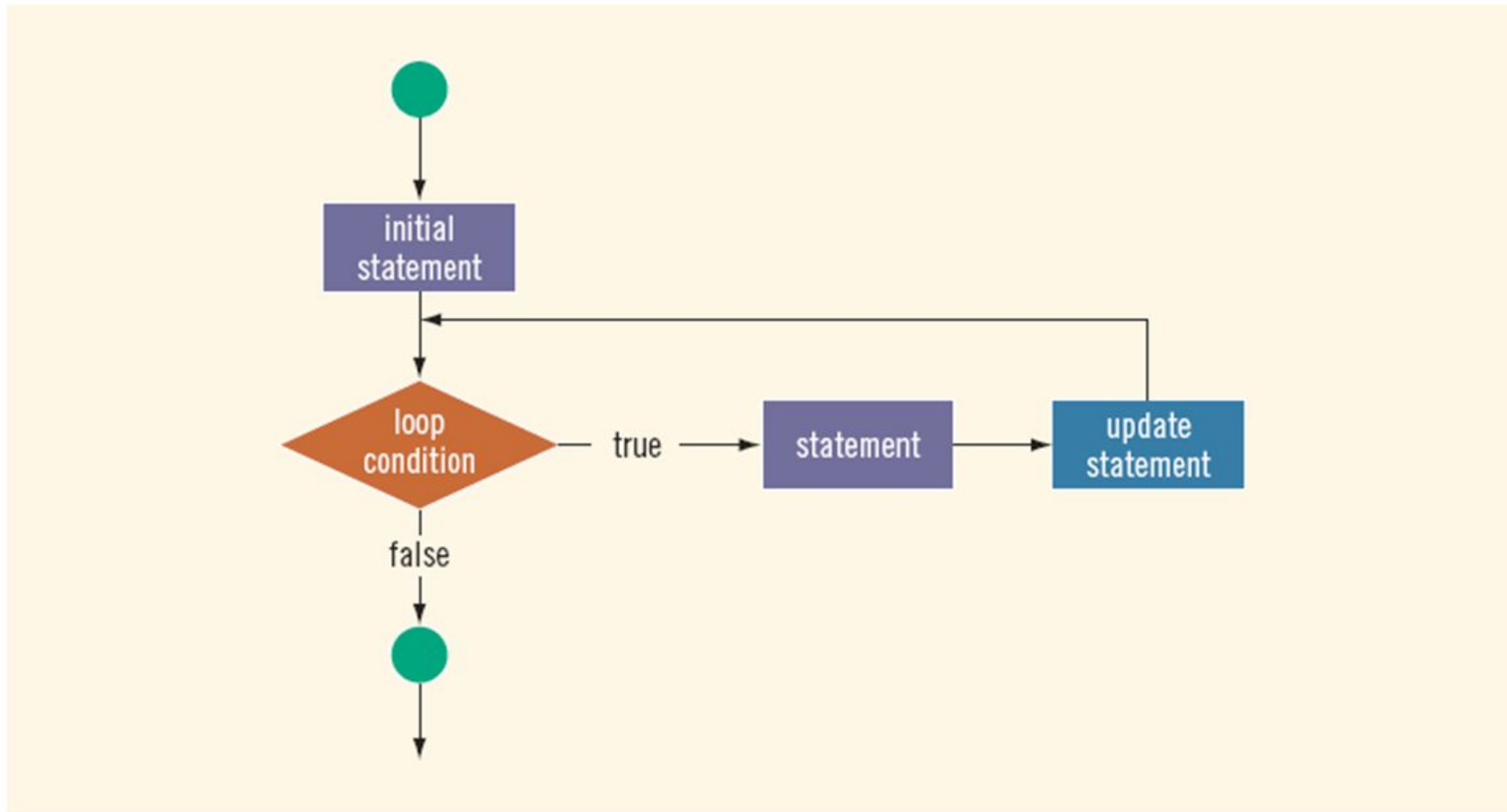


FIGURE 5-2 `for` loop

for Loop Example

```
int sum = 0;
```

```
for ( int num = 1; num <= 10; num++ )  
{  
    sum += num;  
}
```

Diagram illustrating the steps of the for loop:

- Step:1** (only once): `int num = 1;`
- Step:2**: `num <= 10;`
- Step:4**: `num++`
- Step:3** (only if condition is true): `sum += num;`

```
cout << "Sum of numbers 1 - 10 is " << sum << endl;
```

More for Loop

1. The following for loop output Hello! And a star (on separate lines) five times

```
for (i = 1; i <= 5; i++)  
{  
    cout << "Hello!" << endl;  
    cout << "*" << endl;  
}
```

2. Consider the following for loop

```
for (i = 1; i <= 5; i++)  
    cout << "Hello!" << endl;  
    cout << "*" << endl;
```

This loop outputs Hello! Five times and the star only once. Note that the for loop controls only the first output statement because the two output statements are not made into a compound statement. Therefore, the first output statement executes five times because the for loop body executes five times. After the for loop executes, the second output statement executes only once. The indentation, which is ignored by the compiler, is nevertheless misleading.

More for Loop

```
for (i = 0; i < 5; i++);      //Line1  
    cout << "*" << endl;    //Line2
```

The above **for** loop executes five empty statements:

The semicolon at the end of the **for** statement (before the output statement, Line) terminates the **for** loop. The action of this **for** loop empty, that is, null

for Loop Notes

- If **test** is **false** the first time it is evaluated, the body of the loop will not be executed
- The **update** expression can **increment** or **decrement** by any amount
- Variables used in the initialization section **should not be modified** in the body of the loop

for Looping (Repetition) Structure (cont'd.)

- C++ allows you to use fractional values for loop control variables of the double type
- Results may differ
- The following is a semantic error:

EXAMPLE 5-11

The following `for` loop executes five empty statements:

```
for (i = 0; i < 5; i++);    //Line 1
    cout << "*" << endl;    //Line 2
```

The semicolon at the end of the `for` statement (before the output statement, Line 1) terminates the `for` loop. The action of this `for` loop is empty, that is, null.

- The following is a le

```
for(;;)
```

```
    cout << "Hello " << endl;
```

More for Loop

1. Consider the following `for` loop

```
for(int i = 10 ; i <= 9 ; i++)  
    cout << i << " " ;  
cout << endl;
```

In this `for` loop initial statement sets `i` to 10. Because initially the loop condition (`i <= 9`) is `false`. Nothing happens.

2. Consider the following `for` loop

```
for(int i = 9 ; i >= 10 ; i--)  
    cout << i << " " ;  
cout << endl;
```

In this `for` loop initial statement sets `i` to 9. Because initially the loop condition (`i >= 10`) is `false`. Nothing happens.

3. Consider the following `for` loop

```
for(int i = 10 ; i <= 10 ; i++)    //Line 1  
    cout << i << " " ;           //Line 2  
cout << endl;                     //Line 3
```

In this `for` loop initial statement sets `i` to 10. Because initially the loop condition (`i <= 10`) is `true`, so



4. Consider the following `for` loop

```
int i;  
  
for(i = 1 ; i <= 10 ; i++);    //Line 1  
    cout << i << " " ;        //Line 2  
    cout << endl;              //Line 3
```

This `for` loop has no effect on the output statement in **line 2**. The semicolon at the end of the `for` loop ends the `for` loop; the action of the `for` loop is thus empty. The output statement is all by itself and hence runs only once.

5. Consider the following `for` loop

```
for(int i = 1 ; ; i++)  
    cout << i << " " ;  
    cout << endl;
```

In this `for` loop because the loop condition is omitted from the `for` statement, the loop statement is always **true**. This is an infinite loop.

for Loop Modifications

- Can **define variables in initialization code**
 - Their scope is the **for** loop
- Initialization code, test, or update code can **contain more than one statement**
 - Separate statements with commas

Example:

```
for (int sum = 0, num = 1; num <= 10; num++)  
    sum += num;
```


More for Loop Modifications (These are NOT Recommended)

- Can omit initialization if already done

```
int sum = 0, num = 1;  
for (; num <= 10; num++)  
    sum += num;
```

- Can omit update if done in loop

```
for (sum = 0, num = 1; num <= 10;)   
    sum += num++;
```

- Can omit test – may cause an infinite loop

```
for (sum = 0, num = 1;; num++)  
    sum += num;
```

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-12

You can count backward using a `for` loop if the `for` loop control expressions are set correctly.

For example, consider the following `for` loop:

```
for (i = 10; i >= 1; i--)  
    cout << " " << i;  
cout << endl;
```

EXAMPLE 5-13

You can increment (or decrement) the loop control variable by any fixed number. In the following `for` loop, the variable is initialized to 1; at the end of the `for` loop, `i` is incremented by 2. This `for` loop outputs the first 10 positive odd integers.

```
for (i = 1; i <= 20; i = i + 2)  
    cout << " " << i;  
cout << endl;
```



Example

- Revise the Fibonacci numbers using **for** loop.

Example

```
for (int previous1 = 0, previous2 = 1, counter = 3, current ; counter <=
nthFibonacci ; counter++)
{
    current = previous2 + previous1;
    previous1 = previous2;
    previous2 = current;
} //end for
```

for Vs while

```
for (initial expression; logical expression; update expression)
    statement
```

is functionally equivalent to the following `while` statement:

```
initial expression
while (expression)
{
    statement
    update expression
}
```

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

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

Questions

