

Loops

The Increment and Decrement Operators



- ❖ `++` is the increment operator.
It adds one to a variable.

`val++;` is the same as `val = val + 1;`

- ❖ `++` can be used before (prefix) or after (postfix) a variable:

`++val;` `val++;`

The Increment and Decrement Operators



- ❖ `--` is the decrement operator.

It subtracts one from a variable.

`val--;` is the same as `val = val - 1;`

- ❖ `--` can be also used before (prefix) or after (postfix) a variable:

`--val;` `val--;`

Increment and Decrement Operators

<i>Operator</i>	<i>Name</i>	<i>Description</i>	<i>Example (assume i = 1)</i>
++var	preincrement	Increment var by 1 , and use the new var value in the statement	int j = ++i; // j is 2, i is 2
var++	postincrement	Increment var by 1 , but use the original var value in the statement	int j = i++; // j is 1, i is 2
--var	predecrement	Decrement var by 1 , and use the new var value in the statement	int j = --i; // j is 0, i is 0
var--	postdecrement	Decrement var by 1 , and use the original var value in the statement	int j = i--; // j is 1, i is 0

Increment and Decrement Operators

```
int i = 10;
```

```
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

```
int i = 10;
```

```
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

Increment and Decrement Operators in Program 5-1



Program 5-1

```
1 // This program demonstrates the ++ and -- operators.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int num = 4;    // num starts out with 4.
8
9     // Display the value in num.
10    cout << "The variable num is " << num << endl;
11    cout << "I will now increment num.\n\n";
12
13    // Use postfix ++ to increment num.
14    num++;
15    cout << "Now the variable num is " << num << endl;
16    cout << "I will increment num again.\n\n";
17
18    // Use prefix ++ to increment num.
19    ++num;
20    cout << "Now the variable num is " << num << endl;
21    cout << "I will now decrement num.\n\n";
22
23    // Use postfix -- to decrement num.
24    num--;
25    cout << "Now the variable num is " << num << endl;
26    cout << "I will decrement num again.\n\n";
27
```

Continued...

Increment and Decrement Operators in Program 5-1



Program 5-1 *(continued)*

```
28      // Use prefix -- to increment num.
29      --num;
30      cout << "Now the variable num is " << num << endl;
31      return 0;
32  }
```

Program Output

```
The variable num is 4
I will now increment num.

Now the variable num is 5
I will increment num again.

Now the variable num is 6
I will now decrement num.

Now the variable num is 5
I will decrement num again.

Now the variable num is 4
```

Prefix vs. Postfix



- ❖ ++ and -- operators can be used in complex statements and expressions
- ❖ In prefix mode (++val, --val) the operator increments or decrements, *then* returns the value of the variable
- ❖ In postfix mode (val++, val--) the operator returns the value of the variable, *then* increments or decrements

Prefix vs. Postfix - Examples



```
int num, val = 12;
```

```
cout << val++; // displays 12,  
               // val is now 13;  
cout << ++val; // sets val to 14,  
               // then displays it  
num = --val;   // sets val to 13,  
               // stores 13 in num  
num = val--;   // stores 13 in num,  
               // sets val to 12
```

Notes on Increment and Decrement



- ❖ Can be used in expressions:

```
result = num1++ + --num2;
```

- ❖ Must be applied to something that has a location in memory. Cannot have:

```
result = (num1 + num2)++;
```

- ❖ Can be used in relational expressions:

```
if (++num > limit)
```

pre- and post-operations will cause different comparisons

5.2

Introduction to Loops: The `while` Loop

Motivations



- Suppose that you need to print a string (e.g., "Welcome to C++!") a **hundred times**. It would be tedious to have to write the following statement a hundred times:

```
Cout << ("Welcome to C++!");
```

Motivations



How do you solve this problem?

Motivations



while loop

do-while loop

for loop

Motivations



while loop

Introducing **while** Loops



```
int count = 0;
while (count < 100) {
    cout << "Welcome to C++";
    count++;
}
```


Introducing **while** Loops



```
int count = 0;
while (count < 100) {
    cout << "Welcome to C++";
    count++;
}
```

```
int count = 0;
if (count < 100) {
    cout << "Welcome to C++";
    count++;
}
```

- ✦ **if** statement executes only once if the condition is true.
- ✦ **while** statement keeps on iterating until the condition becomes false.

Introduction to Loops: The `while` Loop



- ❖ Loop: a control structure that causes a statement or statements to repeat
- ❖ General format of the `while` loop:

```
while (expression)  
    statement;
```
- ❖ *statement*; can also be a block of statements enclosed in { }

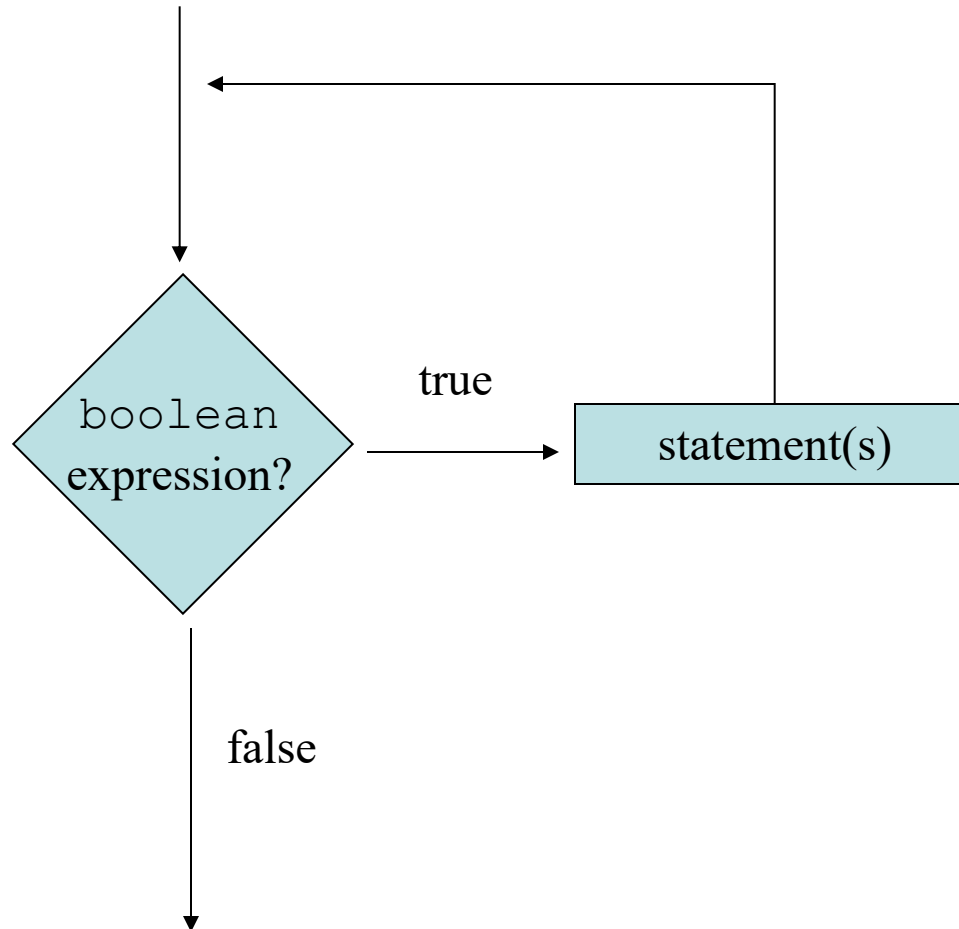
The **while** Loop – How It Works



```
while (expression)  
    statement;
```

- ❖ *expression* is evaluated
 - if true, then *statement* is executed, and *expression* is evaluated again
 - if false, then the loop is finished and program statements following *statement* execute

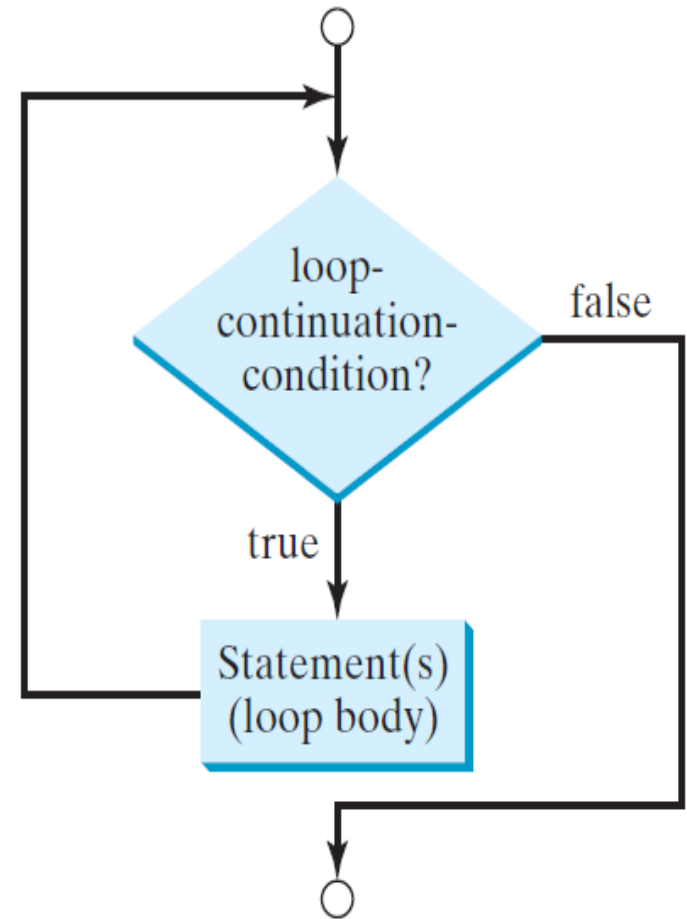
The while loop Flowchart



while Loop Flow Chart



```
while (loop-continuation-condition)
{
    // loop-body;
    Statement(s);
}
```



The while Loop (1 of 2)



- ❖ The while loop has the form:

```
while(condition)  
{  
  statements;  
}
```

- ❖ While the condition is true, the statements will execute repeatedly.
- ❖ The while loop is a *pretest* loop, which means that it will test the value of the condition prior to executing the loop.

The while Loop (2 of 2)



- ❖ Care must be taken to set the condition to false somewhere in the loop so the loop will end.
- ❖ Loops that do not end are called *infinite loops*.
- ❖ A while loop executes 0 or more times. If the condition is false, the loop will not execute.

Infinite Loops (1 of 3)



- ❖ In order for a while loop to end, the condition must become false. The following loop will not end:

```
int x = 20;  
while(x > 0)  
{  
    cout << "x is greater than 0";  
}
```

- ❖ How do you fix this?

Infinite Loops (2 of 3)



- ❖ In order for a while loop to end, the condition must become false. The following loop will not end:

```
int x = 20;
while(x > 0)
{
    cout << "x is greater than 0";

}
```

- ❖ The variable x never gets decremented so it will always be greater than 0.
- ❖ Adding the **x--** above fixes the problem.

Infinite Loops (3 of 3)



- ❖ This version of the loop decrements x during each iteration:

```
int x = 20;
while(x > 0)
{
    cout << "x is greater than 0";
    x--;
}
```

Block Statements in Loops



- ❖ Curly braces are required to enclose block statement while loops. (like block if statements)

```
while (condition)  
{  
    statement;  
    statement;  
    statement;  
}
```

The while loop in Program 5-3



Program 5-3

```
1 // This program demonstrates a simple while loop.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int number = 1;
8
9     while (number <= 5)
10    {
11        cout << "Hello\n";
12        number++;
13    }
14    cout << "That's all!\n";
15    return 0;
16 }
```

Program Output

```
Hello
Hello
Hello
Hello
Hello
That's all!
```



Trace while Loop

```
int count = 0;
```

Initialize count

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

(count < 2) is true

Trace while Loop, cont.



```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

Print Welcome to C++



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

Increase count by 1
count is 1 now

A light blue callout box with a black border and rounded corners. A black arrow points from the box to the `count++;` line of code. The box contains the text "Increase count by 1" and "count is 1 now".



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

(count < 2) is still true since count
is 1



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

Print Welcome to C++



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```

```
}
```

Increase count by 1
count is 2 now

A light blue callout box with a black border and a tail pointing to the `count++;` line of code. The box contains the text "Increase count by 1" and "count is 2 now".



Trace while Loop, cont.

```
int count = 0;
```

```
while (count < 2)
```

```
{
```

```
    cout << "Welcome to C++!";
```

```
    count++;
```


```
}
```

(count < 2) is false since count is 2
now



Trace while Loop

```
int count = 0;
while (count < 2)
{
    cout << "Welcome to C++!";
    count++;
}
```



The loop exits. Execute the next statement after the loop.

A blue rectangular box is positioned at the bottom of the code block, with a blue arrow pointing from its right side to the text box above.

How the while Loop in Program 5-3 Lines 9 through 13 Works



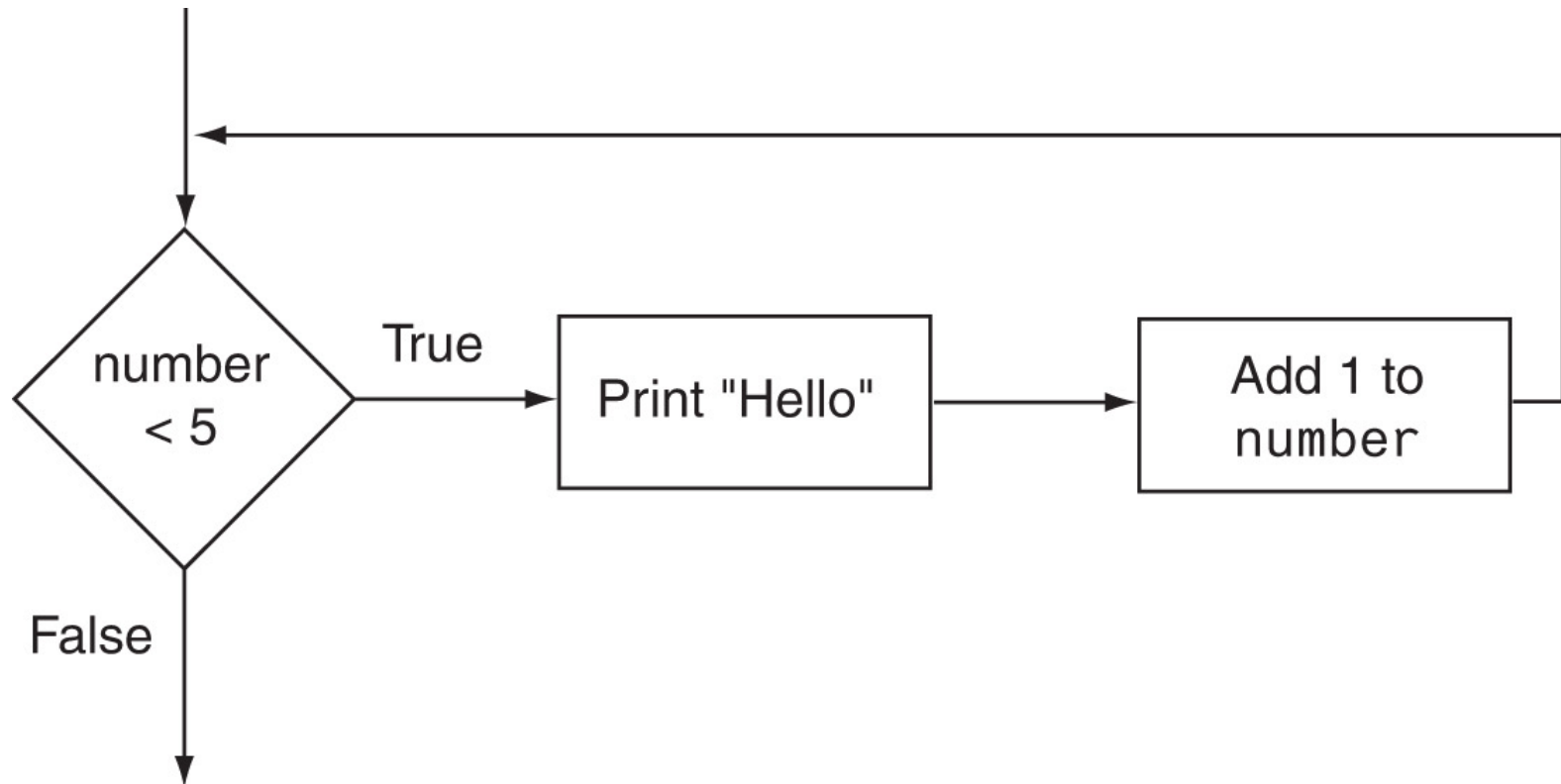
Test this expression.

If the expression is true,
perform these statements.

```
while (number < 5)
{
    cout << "Hello\n";
    number++;
}
```

After executing the body of the loop, start over.

Flowchart of the while Loop in Program 5-3



The `while` Loop is a Pretest Loop



expression is evaluated *before* the loop executes. The following loop will never execute:

```
int number = 6;
while (number <= 5)
{
    cout << "Hello\n";
    number++;
}
```


Repeat Additions - Class exercise



- ❖ Generate 2 random integers and ask the user to input the correct added value of 2 integers. If the answer is wrong display with a comment, “Your answer is wrong. What should the value be ?”.

- ❖ **Output:**

What is $23 + 100$? 23

Wrong answer. Try again. What is $23 + 100$? 112

Wrong answer. Try again. What is $23 + 100$? 123

You got it!

Repeat Additions - Class exercise



- ❖ Generate 2 random integers(one within 50 and the next one within 100) and ask the user to input the correct added value of 2 integers. Try to generate 10 questions randomly. Display the correct count. the If the answer is wrong display the correct answer with a comment, “Your answer is wrong. The correct answer is”.

Repeat Additions - Output:



What is $30 + 79$? 109

What is $4 + 81$? 85

What is $23 + 50$? 73

What is $48 + 39$? 34

Wrong answer.

Your correct answer is 87

What is $25 + 54$? 79

What is $6 + 16$? 22

What is $29 + 47$? 65

Wrong answer.

Your correct answer is 76

What is $41 + 67$? 34

Wrong answer.

Your correct answer is 108

What is $10 + 63$? 73

What is $42 + 86$? 128

Your have answered 7 questions correctly.

Case Study: Guessing Numbers



Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

Case Study: Guessing Numbers



Guess a magic number between 0 and 100

Enter your guess: 56

Your guess is too low

Enter your guess: 78

Your guess is too high

Enter your guess: 72

Your guess is too high

Enter your guess: 62

Your guess is too high

Enter your guess: 58

Your guess is too low

Enter your guess: 60

Your guess is too low

Enter your guess: 61

Yes, the number is 61

Watch Out for Infinite Loops



- ❖ The loop must contain code to make *expression* become `false`
- ❖ Otherwise, the loop will have no way of stopping
- ❖ Such a loop is called an *infinite loop*, because it will repeat an infinite number of times

Example of an Infinite Loop



```
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```

5.3

Using the `while` Loop for Input Validation

Using the `while` Loop for Input Validation



- ❖ Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.
- ❖ The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Using the **while** Loop for Input Validation



- ❖ Here's the general approach, in pseudocode:

Read an item of input.

While the input is invalid

Display an error message.

Read the input again.

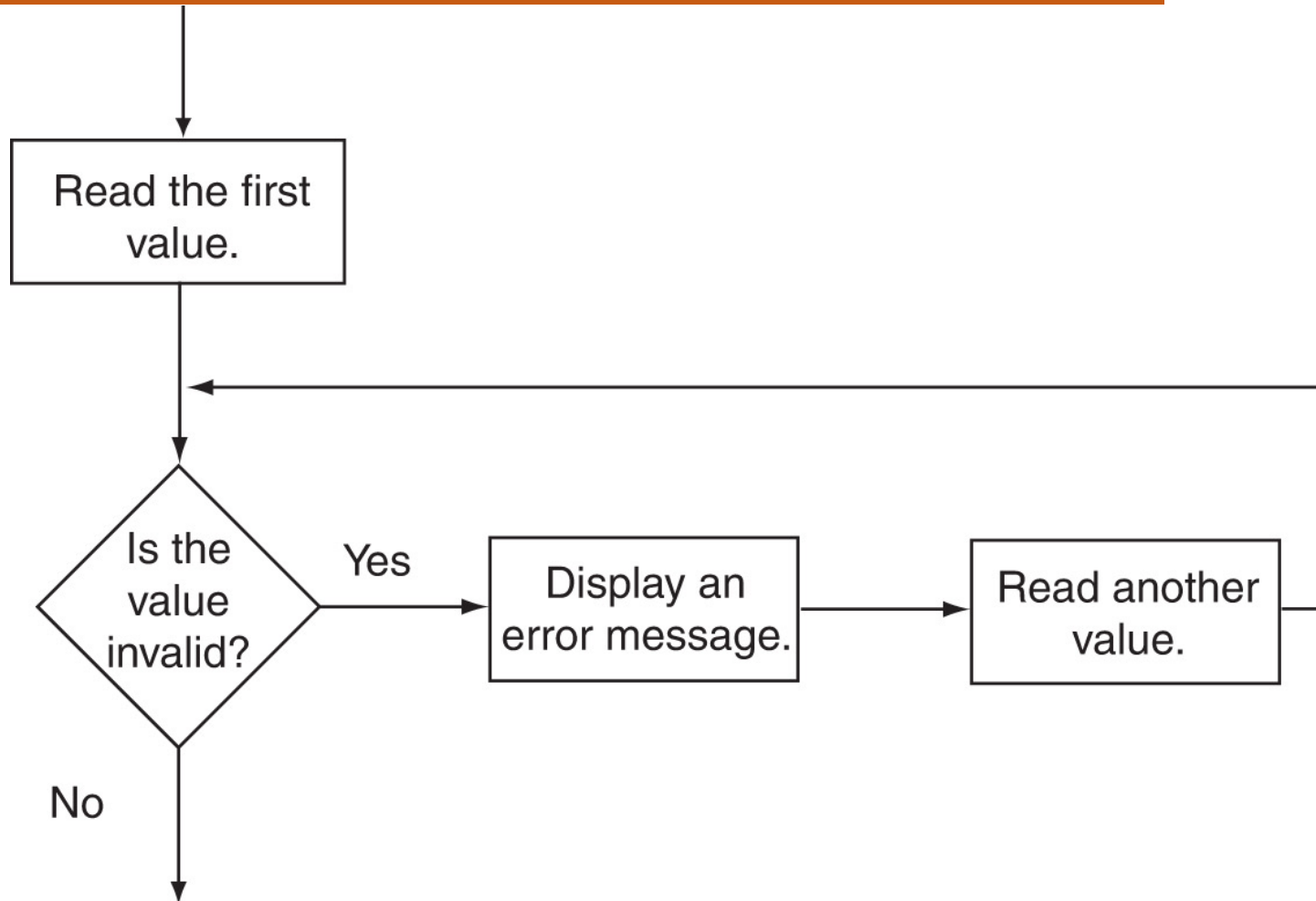
End While

Input Validation Example



```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
    cin >> number;
}
```

Flowchart for Input Validation



Input Validation in Program 5-5



```
20 // Get the number of players per team.
21 cout << "How many players do you wish per team? ";
22 cin >> teamPlayers;
23
24 // Validate the input.
25 while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)
26 {
27     // Explain the error.
28     cout << "You should have at least " << MIN_PLAYERS
29         << " but no more than " << MAX_PLAYERS << " per team.\n";
30
31     // Get the input again.
32     cout << "How many players do you wish per team? ";
33     cin >> teamPlayers;
34 }
35
36 // Get the number of players available.
37 cout << "How many players are available? ";
38 cin >> players;
39
40 // Validate the input.
41 while (players <= 0)
42 {
43     // Get the input again.
44     cout << "Please enter 0 or greater: ";
45     cin >> players;
46 }
```

Bagel problem



- ❖ What happens if someone inputs an invalid number?
- ❖ With if?
- ❖ While for input validation? `BagelShopInputValidation.cpp`
- ❖ `BagelShopWithWhile`



5.4

Counters

Counters



- ❖ Counter: a variable that is incremented or decremented each time a loop repeats
- ❖ Can be used to control execution of the loop (also known as the loop control variable)
- ❖ Must be initialized before entering loop

A Counter Variable Controls the Loop in Prog 5-6



Program 5-6

```
1 // This program displays a list of numbers and
2 // their squares.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     const int MIN_NUMBER = 1,    // Starting number to square
9           MAX_NUMBER = 10;    // Maximum number to square
10
11     int num = MIN_NUMBER;        // Counter
12
13     cout << "Number Number Squared\n";
14     cout << "-----\n";
```

Continued...

A Counter Variable Controls the Loop in Prog 5-6



```
15     while (num <= MAX_NUMBER)
16     {
17         cout << num << "\t\t" << (num * num) << endl;
18         num++; //Increment the counter.
19     }
20     return 0;
21 }
```

Program Output

Number Number Squared

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

5.5

The do-while Loop

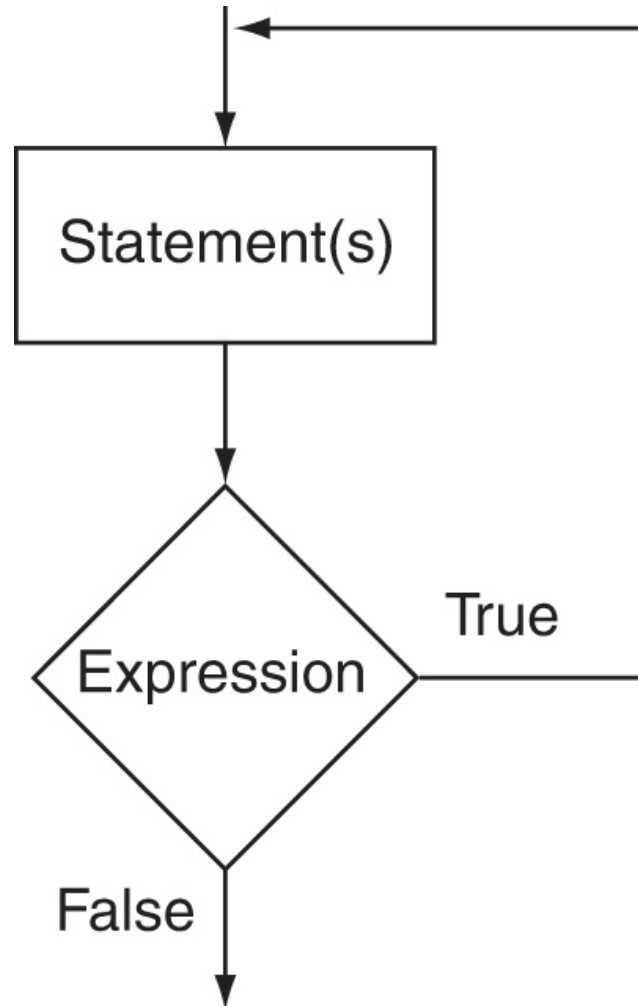
The do-while Loop



- ❖ do-while: a posttest loop – execute the loop, then test the expression
- ❖ General Format:

```
do
    statement; // or block in { }
while (expression);
```
- ❖ Note that a semicolon is required after (*expression*)

The Logic of a do-while Loop



An Example do-while Loop



```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because `do-while` is a posttest loop.

A do-while Loop in Program 5-7



Program 5-7

```
1 // This program averages 3 test scores. It repeats as
2 // many times as the user wishes.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int score1, score2, score3; // Three scores
9     double average;             // Average score
10    char again;                  // To hold Y or N input
11
12    do
13    {
14        // Get three scores.
15        cout << "Enter 3 scores and I will average them: ";
16        cin >> score1 >> score2 >> score3;
17
18        // Calculate and display the average.
19        average = (score1 + score2 + score3) / 3.0;
20        cout << "The average is " << average << ".\n";
21
22        // Does the user want to average another set?
23        cout << "Do you want to average another set? (Y/N) ";
24        cin >> again;
25    } while (again == 'Y' || again == 'y');
26    return 0;
27 }
```

Continued...

A do-while Loop in Program 5-7



Program Output with Example Input Shown in Bold

Enter 3 scores and I will average them: **80 90 70** [Enter]

The average is 80.

Do you want to average another set? (Y/N) **y** [Enter]

Enter 3 scores and I will average them: **60 75 88** [Enter]

The average is 74.3333.

Do you want to average another set? (Y/N) **n** [Enter]

do-while Loop Notes



- ❖ Loop always executes at least once
- ❖ Execution continues as long as *expression* is true, stops repetition when *expression* becomes false
- ❖ Useful in menu-driven programs to bring user back to menu to make another choice (*see Program 5-8 on pages 245-246*)
- ❖ 5.8

Bagel Shop Checkout Counter



- ❖ Write a program to calculate the amount a customer should pay in a self checkout counter for his purchases in a bagel shop. The products sold are everything bagels, garlic bagels, blueberry bagels, cream cheese, and coffee. Using doWhile loops write the program to display the menu of the Bagel shop. Make the user buy multiple items of different choices. Finally display the total amount for the purchases
- ❖ Paste your output using different choices and different counts. Also choose a choice not in Menu
- ❖ [BagelShopDoWhile.java](#) p7