

THIRD EDITION

A First Book of C++

From Here to There

CHAPTER 5

Repetition

Objectives

You should be able to describe:

- The `while` Statement
- `cin` within a `while` Loop
- The `for` Statement
- The `do` Statement
- Common Programming Errors

The `while` Statement

- A general repetition statement

- Format:

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

- Function

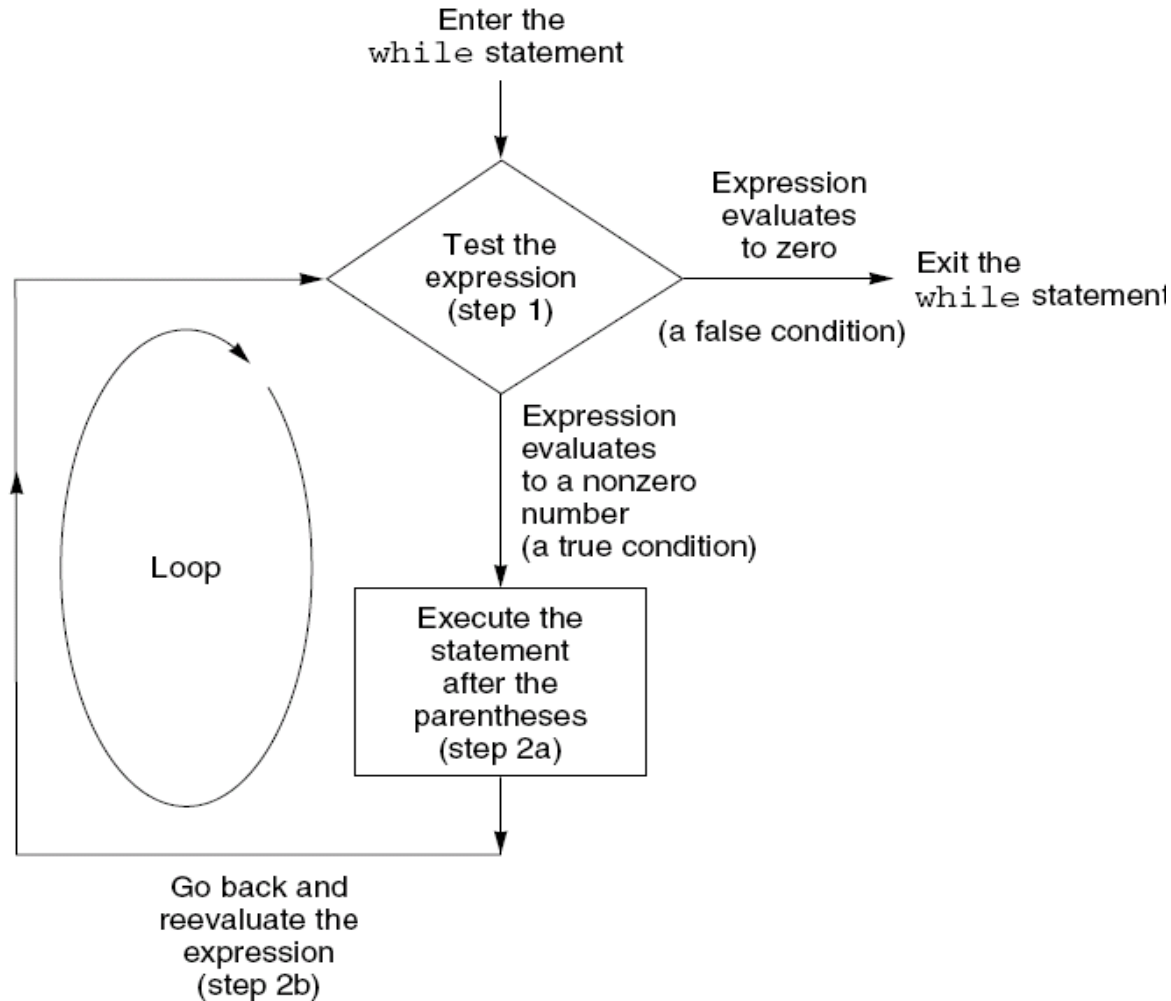
- Expression is evaluated the same as an `if-else` expression
- Statement following expression executed repeatedly as long as expression has nonzero value

The `while` Statement (continued)

- Steps in execution of `while` statement:
 1. Test the expression
 2. If the expression has a nonzero (true) value
 - Execute the statement following the parentheses
 - Go back to step 1
 - `else`
 - Exit the `while` statement

The while Statement (continued)

FIGURE 5.1 Anatomy of a while Loop



The `while` Statement (continued)



Program 5.1

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

int main()
{
    int count;

    count = 1;                // initialize count
    while (count <= 10)
    {
        cout << count << " ";
        count++;             // increment count
    }

    return 0;
}
```

The output produced by Program 5.1 is

1 2 3 4 5 6 7 8 9 10

The `while` Statement (continued)

- **Fixed-count Loop:** tested expression is a counter that checks for a fixed number of repetitions
- **Variation:** Counter is incremented by a value other than 1
- **Example:**
 - Celsius-to-Fahrenheit temperature-conversion
 - Display Fahrenheit and Celsius temperatures, from 5-50 degrees C, in 5 degree increments

The `while` Statement (continued)

```
celsius = 5;           // starting Celsius value
while (celsius <= 50)
{
    fahrenheit = (9.0/5.0) * celsius + 32.0;
    cout << setw(4) << celsius
         << setw(13) << fahrenheit << endl;
    celsius = celsius + 5;
}
```

`cin` within a `while` Loop

- Combines interactive data entry with the repetition of a `while` statement
 - Produces very powerful and adaptable programs
- Example (program 5.5): `while` statement accepts and displays four user-entered numbers
 - Numbers accepted and displayed one at a time

cin within a while Loop (continued)



Program 5.5

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

int main()
{
    const int MAXNUMS = 4;
    int count;
    double num;

    cout << "\nThis program will ask you to enter "
         << MAXNUMS << " numbers.\n";
    count = 1;

    while (count <= MAXNUMS)
    {
        cout << "\nEnter a number: ";
        cin >> num;
        cout << "The number entered is " << num;
        count++;
    }
    cout << endl;

    return 0;
}
```

`cin` within a `while` Loop (continued)

Sample run of program 5.5:

This program will ask you to enter 4 numbers.

Enter a number: 26.2

The number entered is 26.2

Enter a number: 5

The number entered is 5

Enter a number: 103.456

The number entered is 103.456

Enter a number: 1267.89

The number entered is 1267.89

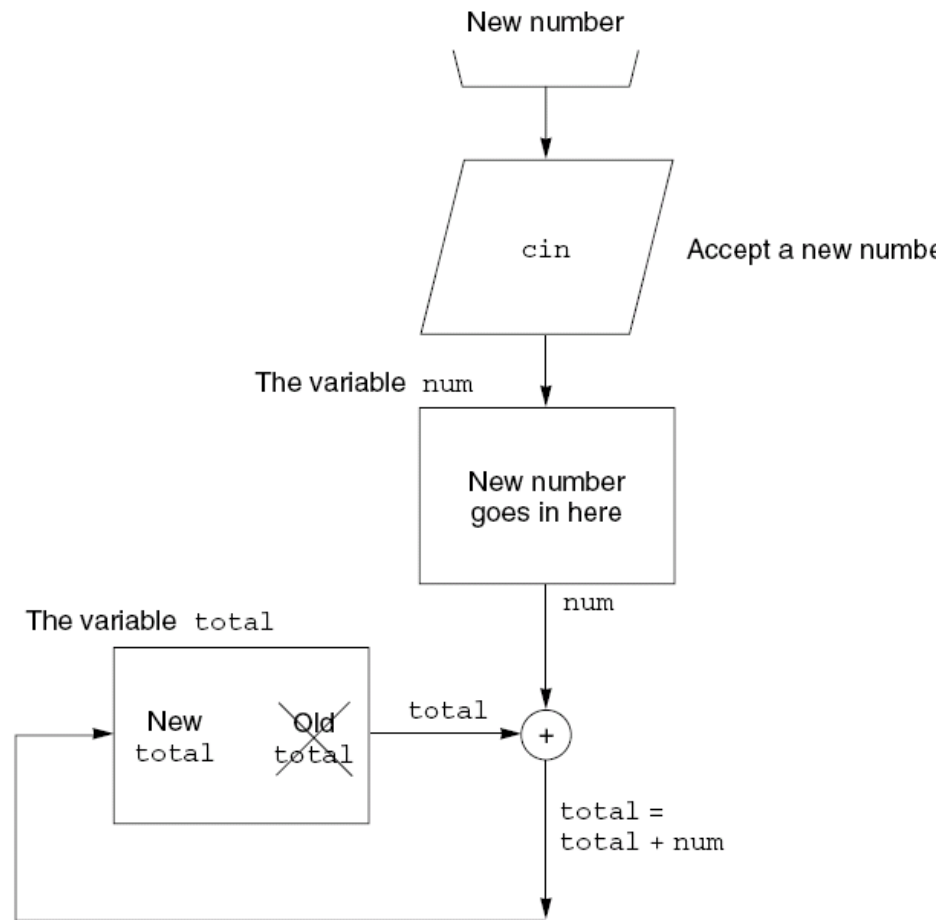
cin within a while Loop (continued)

- Example (program 5.6): adding a single number to a total
 - A number is entered by the user
 - Accumulating statement adds the number to total

```
total = total + num;
```
 - A `while` statement repeats the process

cin within a while Loop (continued)

FIGURE 5.3 *Accepting and Adding a Number to a Total*



cin within a while Loop (continued)



Program 5.6

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

int main()
{
    const int MAXNUMS = 4;
    int count;
    double num, total;

    cout << "\nThis program will ask you to enter "
         << MAXNUMS << " numbers.\n";
    count = 1;
    total = 0;

    while (count <= MAXNUMS)
    {
        cout << "\nEnter a number: ";
        cin >> num;
        total = total + num;
        cout << "The total is now " << total;
        count++;
    }

    cout << "\n\nThe final total is " << total << endl;

    return 0;
}
```

cin within a while Loop (continued)

Sample run of program 5.6:

```
This program will ask you to enter 4 numbers.
```

```
Enter a number: 26.2
```

```
The total is now 26.2
```

```
Enter a number: 5
```

```
The total is now 31.2
```

```
Enter a number: 103.456
```

```
The total is now 134.656
```

```
Enter a number: 1267.89
```

```
The total is now 1402.546
```

```
The final total is 1402.546
```

break and continue statements

- **break:** forces immediate exit from structures:
 - Use in `switch` statements:
 - The desired case has been detected and processed
 - Use in `while`, `for` and `do-while` statements:
 - An unusual condition is detected

- **Format:**

```
break;
```

break and continue statements (continued)

- **continue:** causes the next iteration of the loop to begin immediately
 - Execution transferred to the top of the loop
 - Applies only to `while`, `do-while` and `for` statements
- **Format:**
`continue;`

The Null Statement

- Used where a statement is syntactically required but no action is called for
 - A do-nothing statement
 - Typically used with `while` or `for` statements

The `for` Statement

- Same function as the `while` statement but in different form

```
for (initializing list; expression;  
    altering list)  
    statement;
```

- **Function:** statement executed while expression has nonzero (true) value
- **Components:**
 - **Initializing list:** Initial value of expression
 - **Expression:** a valid C++ expression
 - **Altering list:** statements executed at end of each `for` loop to alter value of expression

The `for` Statement (continued)

- Recall placement of statements in `while` loop

```
initializing statements;
while (expression)
{
    loop statements;
    expression-altering statements;
}
```
- `for` statement format differences
 - All initialization statements grouped as first set of items within the `for`'s parentheses
 - Expression and loop statements: no change
 - Expression-altering statements combined as last set of items within `for`'s parentheses

The `for` Statement (continued)

- Components of `for` statement correspond to operations performed in `while` statement
 - Initialization
 - Expression evaluation
 - Altering of expression values
- Components of `for` statement are optional but semicolons must always be present
- **Example:**
(`; count <= 20 ;`) is valid content of `for` statement parentheses

The `for` Statement (continued)



Program 5.9

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

int main()
{
    int count;

    for (count = 2; count <= 20; count = count + 2)
        cout << count << " ";

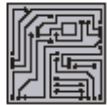
    return 0;
}
```

The output of Program 5.9 is

2 4 6 8 10 12 14 16 18 20

The `for` Statement (continued)

Program 5.9 modified: initializer outside for loop



Program 5.9a

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

int main()
{
    int count;

    count = 2;    // initializer outside for statement
    for ( ; count <= 20; count = count + 2)
        cout << count << " ";

    return 0;
}
```

The for Statement (continued)



Program 5.10

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

int main()
{
    const int MAXNUMS = 10;
    int num;
    cout << endl;          // print a blank line
    cout << "NUMBER    SQUARE    CUBE\n"
         << "-----    -       -\n";

    for (num = 1; num <= MAXNUMS; num++)
        cout << setw(3) << num << "    "
             << setw(3) << num * num << "    "
             << setw(4) << num * num * num << endl;

    return 0;
}
```

The `for` Statement (continued)

When Program 5.10 is run, the display produced is

NUMBER	SQUARE	CUBE
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

`cin` within a `for` Loop

- Same effect as using `cin` object within a `while` loop
- Provides interactive user input

cin within a for Loop (continued)



Program 5.11

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

// This program calculates the average
// of MAXCOUNT user-entered numbers
int main()
{
    const int MAXCOUNT = 5;
    int count;
    double num, total, average;

    total = 0.0;

    for (count = 0; count < MAXCOUNT; count++)
    {
        cout << "Enter a number: ";
        cin >> num;
        total = total + num;
    }

    average = total / count;
    cout << "The average of the data entered is " << average
        << endl;

    return 0;
}
```

`cin` within a `for` Loop (continued)

- **Program 5.11:** `for` statement creates a loop
 - Loop executed five times
- Actions performed in each loop
 - User prompted to enter a number
 - Number added to the total

cin within a for Loop (continued)

- **Initialization variations:**

- **Alternative 1:** initialize `total` outside the loop and `count` inside the loop as in program 5.11

- **Alternative 2:** initialize both `total` and `count` inside loop

```
for (total = 0.0, count = 0; count <
    MAXCOUNT; count++)
```

- **Alternative 3:** initialize and declare both `total` and `count` inside loop

```
for (double total = 0.0, int count = 0;
    count < MAXCOUNT; count++)
```

Nested Loops

- A loop contained within another loop
- **Example:**

```
for(i = 1; i <= 5; i++)          // start of outer loop
{
    cout << "\ni is now " << i << endl;

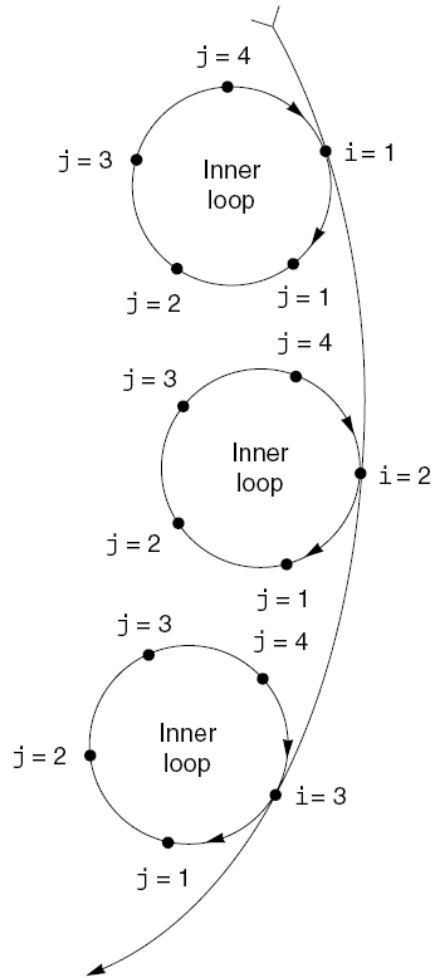
    for(j = 1; j <= 4; j++)      // start of inner loop
    {
        cout << " j = " << j;
    }                            // end of inner loop
}                                 // end of outer loop
```

Nested Loops (continued)

- **Outer (first) Loop:**
 - Controlled by value of i
- **Inner (second) Loop:**
 - Controlled by value of j
- **Rules:**
 - For each single trip through outer loop, inner loop runs through its entire sequence
 - Different variable to control each loop
 - Inner loop statements contained within outer loop

Nested Loops (continued)

FIGURE 5.6 For Each i, j Loop



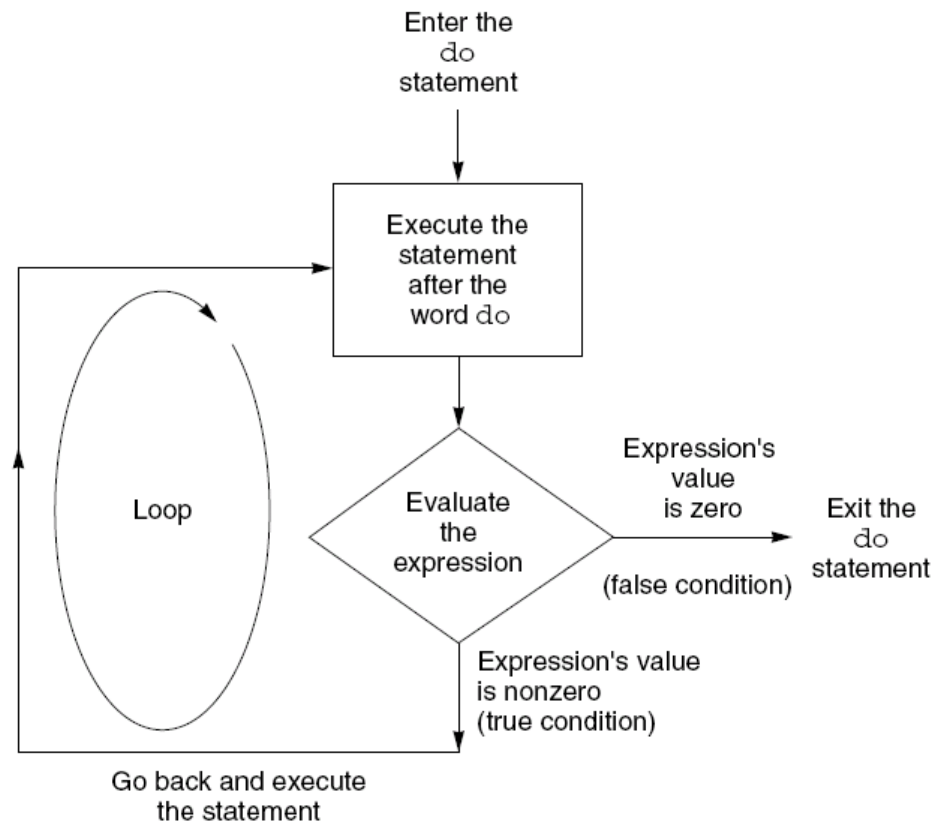
The do Statement

- A repetition statement that evaluates an expression at the end of the statement
 - Allows some statements to be executed before an expression is evaluated
 - `for` and `while` evaluate an expression at the beginning of the statement
- **Format:**

```
do
    statement;
while (expression); // don't forget final ;
```

The do Statement (continued)

FIGURE 5.7 *The do Statement's Flow of Control*



Validity Checks

- Provided by `do` statement through filtering of user-entered input
- **Example:**

```
do
{
    cout << "\nEnter an identification number: ";
    cin >> idNum;
    if (idNum < 100 || idNum > 1999)
    {
        cout << "\n An invalid number was just entered"
              << "\nPlease check the ID number and re-
                enter";
    }
    else
        break; // break if a valid id number was
              entered
} while(1); // this expression is always true
```

Common Programming Errors

- “**One-off**” errors: loop executes one time too many or one time too few
 - Initial and tested conditions to control loop must be carefully constructed
- Inadvertent use of assignment operator, = in place of the equality operator, ==
 - This error is not detected by the compiler

Common Programming Errors (continued)

- Using the equality operator when testing double-precision operands
 - Do not test expression `(fnum == 0.01)`
 - Replace by a test requiring absolute value of `(fnum - 0.01) < epsilon` for very small `epsilon`
- Placing a semicolon at end of the `for` statement parentheses:

```
for (count = 0; count < 10; count ++);  
total = total + num;
```

 - Creates a loop that executes 10 times and does nothing but increment `count`

Common Programming Errors (continued)

- Using commas instead of semicolons to separate items in a for statement:

```
for (count = 1, count < 10, count ++)  
// incorrect
```

- Commas should be used to separate items within the separating and initializing lists

- Omitting the final semicolon from the *do* statement

```
do  
    statement;  
while (expression) ← don't forget the final ;
```

Summary

- `while`, `for` and `do` statements create loops
 - These statements evaluate an expression
 - On the basis of the expression value, either terminate the loop or continue with it
 - Each pass through the loop is called a repetition or iteration
- `while` checks expression before any other statement in the loop
 - Variables in the tested expression must have values assigned before `while` is encountered

Summary (continued)

- The `for` statement: fixed-count loops
 - Included in parentheses at top of loop:
 - Initializing expressions
 - Tested expression
 - Expressions that affect the tested expression
 - Other loop statements can also be included as part of the altering list

Summary (continued)

- The `do` statement checks its expression at the end of the loop
 - Body of the loop must execute at least once
 - `do` loop must contain statement(s) that either:
 - Alter the tested expression's value or
 - Force a break from the loop