Loops and Files

CS 1: Problem Solving & Program Design Using C++

Objectives

- Go around and around with the while statement
- Get input via cin within a while loop
- Go through increments of the for statement
- Also get input via cin within a for loop
- Keeping doing the do with the do statement
- Get a handle on handling files
- Discover common programming errors

The while Statement

- A general repetition statement
- Format:

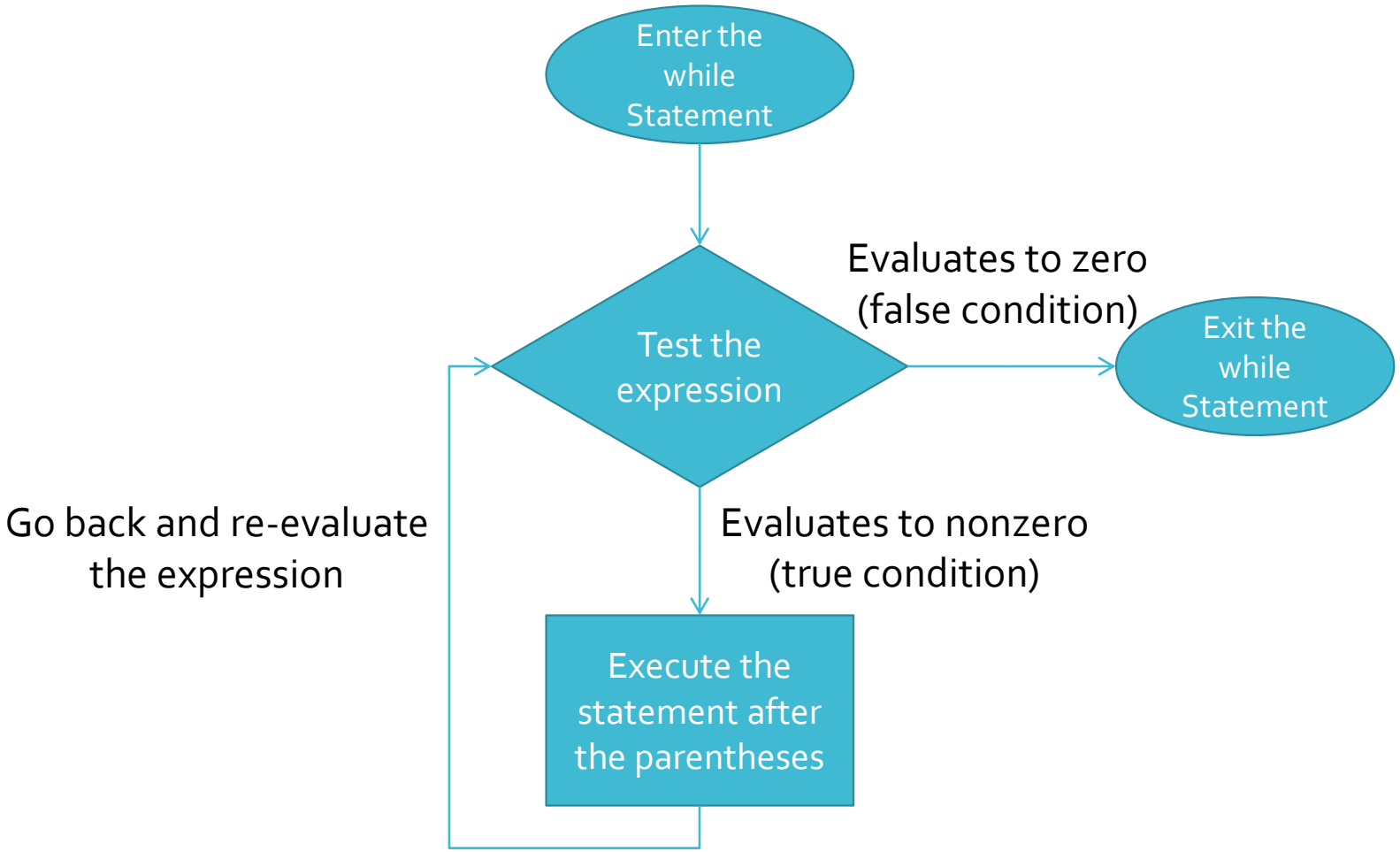
```
while (expression)
{
        statement;
}
```

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

Steps in Executing a while Statement

- Test the expression
- If the expression has a nonzero (true) value
  - Execute the statement following the parentheses
  - Go back and test the expression again
- Else
  - Exit the while statement

Flowchart of the while Statement



Example of the while Statement

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

int main()
{
        int count;

        count = 1;
        while (count <= 10)
        {
                cout << count << " ";
                count++;
        }
        cout << endl;

        return 0;
}
```

- Output of the program: 1 2 3 4 5 6 7 8 9 10

More on the while Statement

- 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 #2 of the while Statement

- Celsius-to-Fahrenheit temperature-conversion
  - Display Fahrenheit and Celsius temperatures, from 5-50 degrees C, in 5 degree increments

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

cin Within a while Loop Example #1

- while statement accepts and displays four user-entered numbers
- Numbers accepted and displayed one at a time

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

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

        cout << "This program will ask you to enter "
             << MAXNUMS << " numbers." << endl;
        count = 1;
```

cin Within a while Loop Example #1 (2)

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

        return 0;
}
```

Sample Run of cin Within a while Loop Example #1

```
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
```

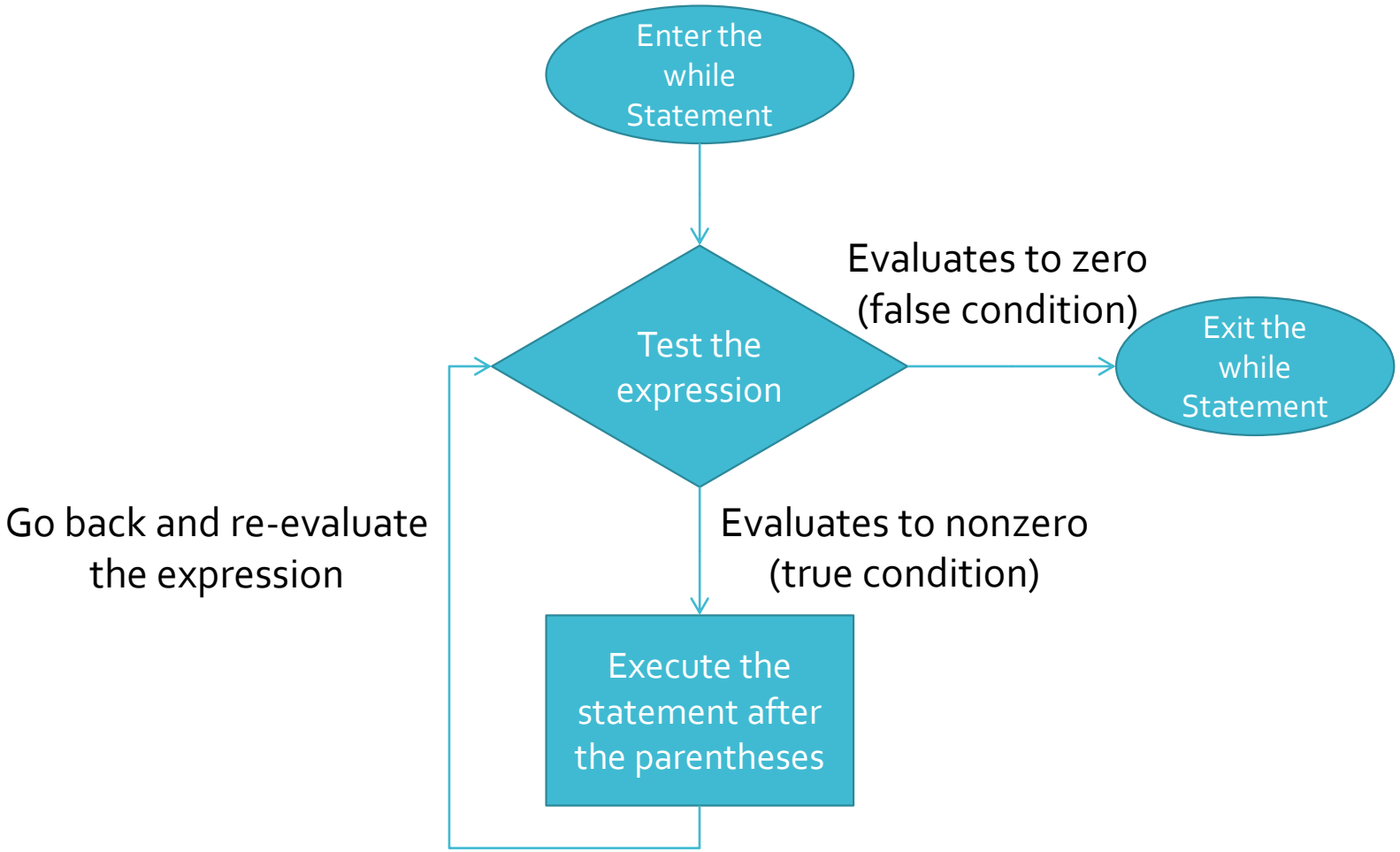
cin Within a while Loop Example #2

- 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

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

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

        cout << "This program will ask you to enter "
             << MAXNUMS << " numbers." << endl;
        count = 1;
```

Flowchart for cin Within a while Loop Example #2



Code of cin Within a while Loop Example #2

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

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

        cout << "This program will ask you to enter "
             << MAXNUMS << " numbers." << endl;
        count = 1;
        total = 0;
```

Code of cin Within a while Loop Example #2 (2)

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

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

        return 0;
}
```

Sample Run of cin Within a while Loop Example #2

```
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 Statement

- 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;
```

continue Statement

- 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 (2)

- 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 (3)

- 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 Example #1

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

int main()
{
        int count;

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

        return 0;
}
```

- Output of the program: 2 4 6 8 10 12 14 16 18 20

The for Statement Example #1 Modified

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

int main()
{
        int count;

        count = 2; // Initializer outside for statement
        for ( ; count < 20; count += 2)
        {
                cout << count << " ";
        }
        cout << endl;

        return 0;
}
```

The for Statement Example #2

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

int main()
{
        const int MAXNUMS = 10;
        int num;

        // Print the header row
        cout << endl; // Print a blank line
        cout << "NUMBER  SQUARE  CUBE\n"
             << "------  ------  ----\n";
```

The for Statement Example #2 (2)

```
        // Print the first 10 numbers, squares, and cubes
        for (num = 1; num <= 10; num++)
        {
                cout << setw(6) << num
                        << setw(8) << num * num
                        << setw(6) << num * num * num << endl;
        }

        return 0;
}
```

The for Statement Example #2 Sample Run

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 Example

- 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 Example (2)

```
#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;
```

cin Within a for Loop Example (3)

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

		average = total / count;
		cout << "\n\nThe average of the data entered is " << average
<< endl;

		return 0;
}
```

cin Within a for Loop Example Initialization Variations

- Alternative 1: initialize total outside the loop and count inside the loop
- 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

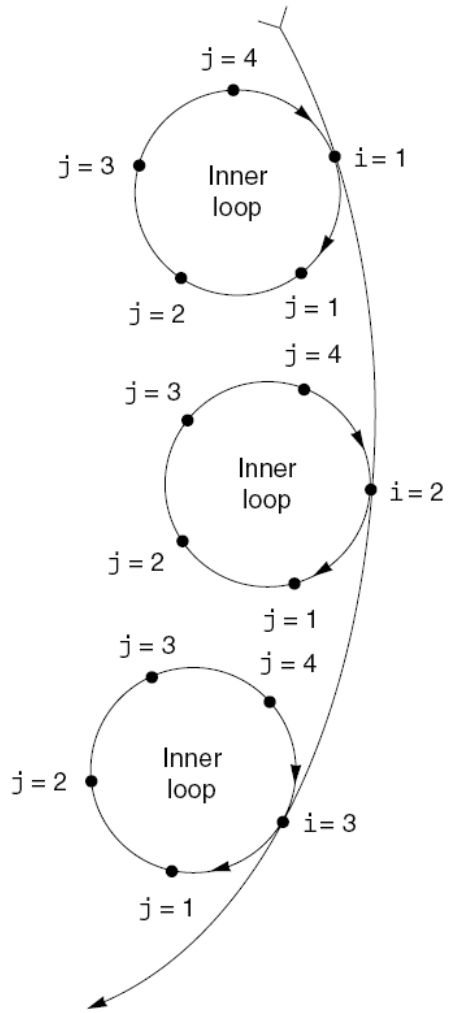
```
for (i = 1; i <= 5; i++) // start of outer loop
{
        cout << "\ni is now " << i << endl;

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

Nested Loops (2)

- 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 (3)

FIGURE 5. 6 *For Each* `i, j` *Loop*

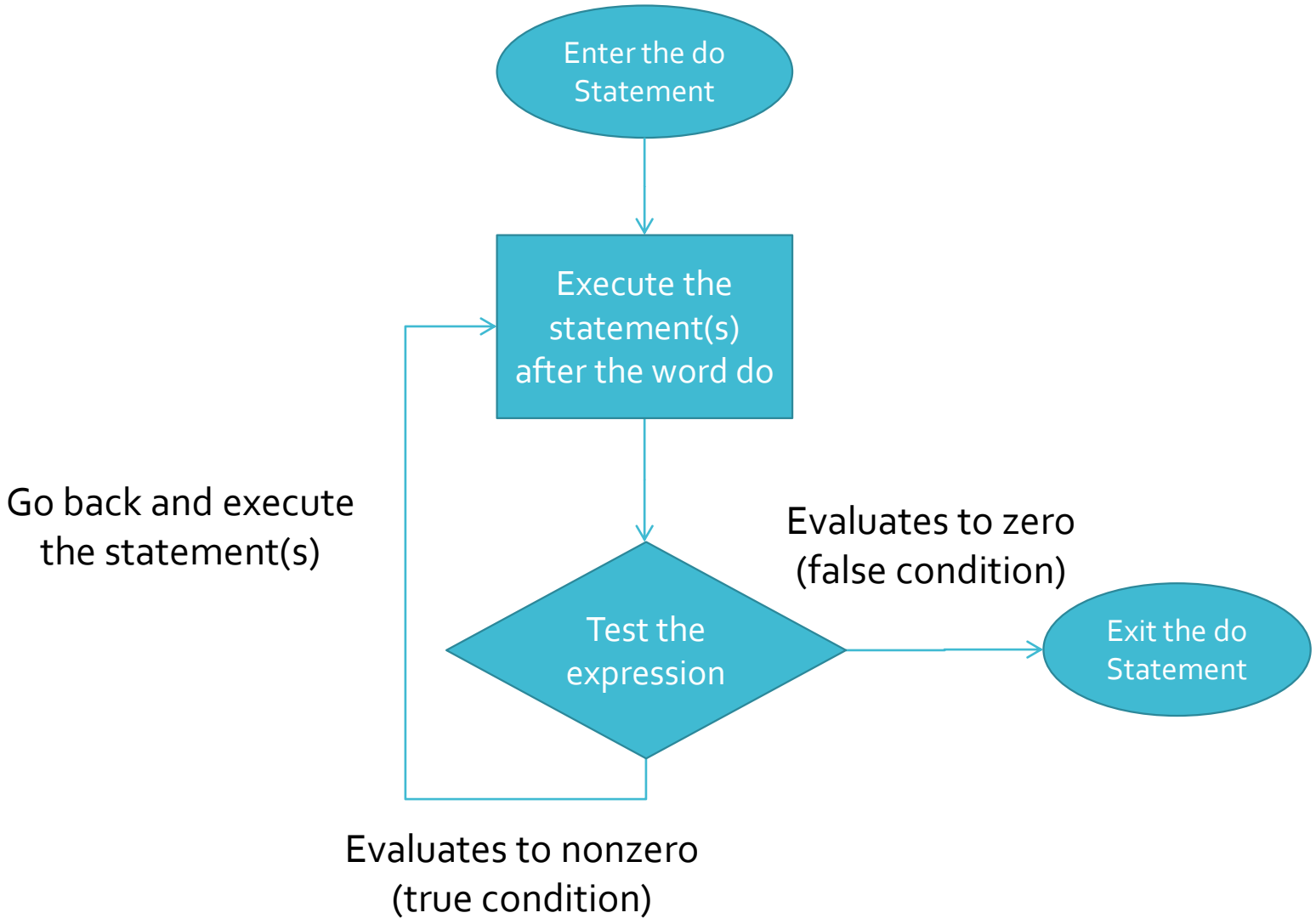


The do Statement

- 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 ;
```

Flowchart of the do Statement



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 ID number and re-enter";
        }
        else
                break; // Break if a valid id number was entered
} while(1); // This expression is always true
```

Sentinel

- SENTINEL: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, e.g., -999 for a test score
- Used to terminate input when user may not know how many values will be entered

Sentinel Example

```
// This program calculates the total number of points a soccer team has
// earned over a series of games.  The user enters a series of point
// values, then -1 when finished.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
        int game = 1,     // game counter
            points = 0,    // to hold a number of points
            total = 0;       // accumulator

        cout << "Enter the number of points your team has earned"
             << endl << "so far in the season, then enter -1 when "
              << "you're finished." << endl << endl;
        cout << "Enter your points for game # " << game << ": ";
        cin >> points;
```

Sentinel Example (2)

```
		while (points != -1)
		{
			total += points;
			game++;
			cout << "Enter your points for game # " << game <<
": ";
			cin >> points;
		}

		cout << endl << "The total points are " << total << endl;

		return 0;
}
```

Sentinel Example Sample Run

Enter the number of points your team has earned
so far in the season, then enter -1 when you're finished.

Enter your points for game #1: 7 [Enter]

Enter your points for game #2: 9 [Enter]

Enter your points for game #3: 4 [Enter]

Enter your points for game #4: 8 [Enter]

Enter your points for game #5: 6 [Enter]

Enter your points for game #6: -1 [Enter]

The total points are 34

Deciding Which Loop to Use

- The while loop is a conditional pretest loop
  - Iterates as long as a certain condition exits
  - Validating input
  - Reading lists of data terminated by a sentinel
- The do-while loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The for loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - Open the file
  - Use the file (read from, write to, or both)
  - Close the file

Files: What Is Needed

- Use fstream header file for file access
- File stream types:
  - ifstream for input from a file
  - ofstream for output to a file
  - fstream for input from or output to a file
- Define file stream objects:

```
ifstream infile;
ofstream outfile;
```

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the open member function:

```
infile.open("inventory.dat");

outfile.open("report.txt");
```

- Filename may include drive, path information
- Output file will be created if necessary
  - Existing file will be erased first
- Input file must exist for open to work

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
        cout << "File open failure!";
}
```

- Can also use the fail member function

Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
infile >> qtyInStock >> qtyOnOrder;
```

Using Loops to Process Files

- The stream extraction operator >> returns true when a value was successfully read, false otherwise
- Can be tested in a while loop to continue execution as long as values are read from the file:

```
while (inputFile >> number) ...
```

Closing Files

- Use the close member function:

```
infile.close();

outfile.close();
```

- Don't wait for operating system to close files at program end
  - May be limit on number of open files
  - May be buffered output data waiting to send to file

Letting the User Specify a Filename

- The open member function requires that you pass the name of the file as a null-terminated string, which is also known as a C-string
- String literals are stored in memory as null-terminated C-strings, but string objects are not

Letting the User Specify a Filename (2)

- string objects have a member function named c_str
- It returns the contents of the object formatted as a null-terminated C-string
- Here is the general format of how you call the c_str function

```
stringObject.c_str()
```

Letting the User Specify a Filename Example

```
// This program lets the user enter a filename.
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
        ifstream inputFile;
        string filename;
        int number;

        // Get the filename from the user
        cout << "Enter the filename : ";
        cin >> filename;

        // Open the file
        inputFile.open(filename.c_str());
```

Letting the User Specify a Filename Example (2)

```
        // If the file opened successfully, process it
        if (inputFile)
        {
                // Read the numbers from the file and display them
                while (inputFile >> number)
                {
                        cout << number << endl;
                }

                // Close the file
                inputFile.close();
        }
        else
        {
                // Display an error message
                cout << "Error opening the file." << endl;
        }

        return 0;
}
```

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 (2)

- 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 += num;
```

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

Common Programming Errors (3)

- 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 (2)

- 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 (3)

- 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 do one of the following
    - Alter the tested expression's value
    - Force a break from the loop

Summary (4)

- Can use files instead of keyboard, monitor screen for program input, output
- Steps:
  - Open the file
  - Use the file (read from, write to, or both)
  - Close the file