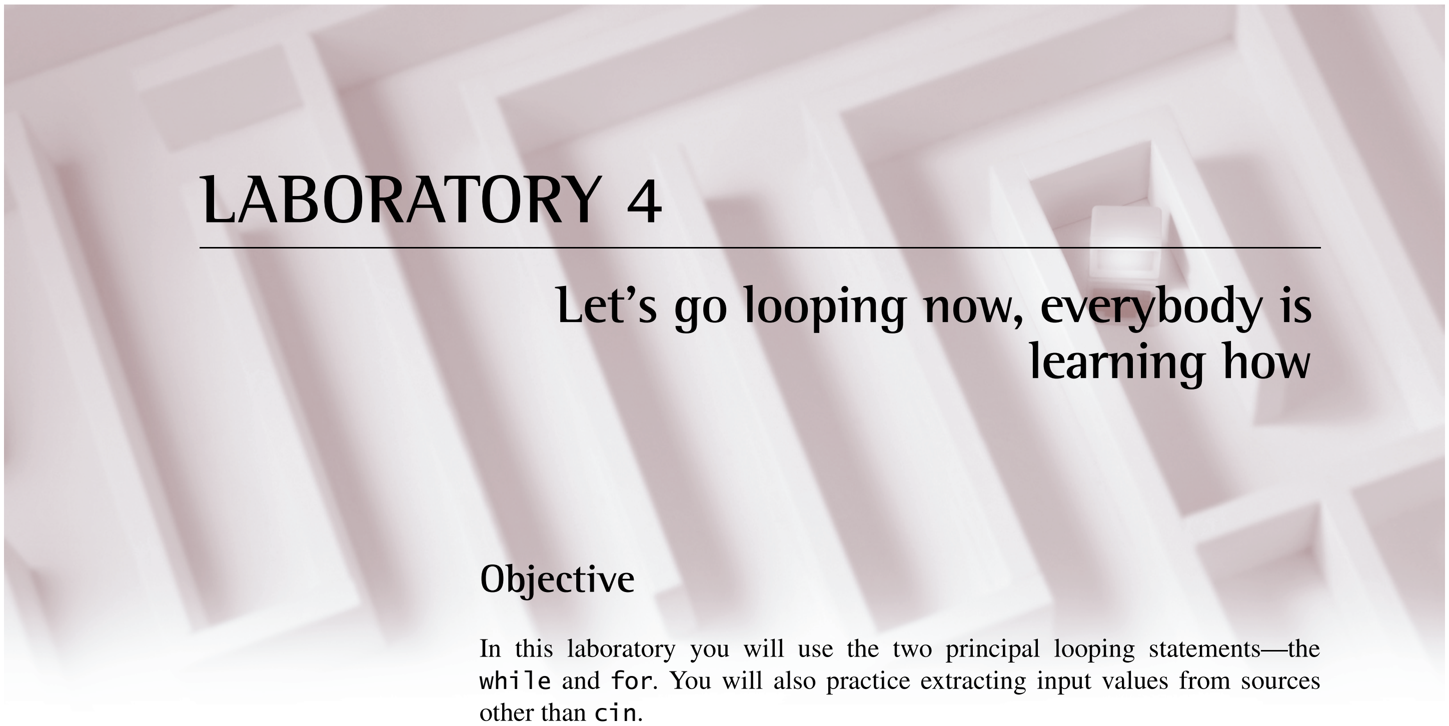


LABORATORY 4

Let's go looping now, everybody is learning how

Objective

In this laboratory you will use the two principal looping statements—the `while` and `for`. You will also practice extracting input values from sources other than `cin`.

Key Concepts

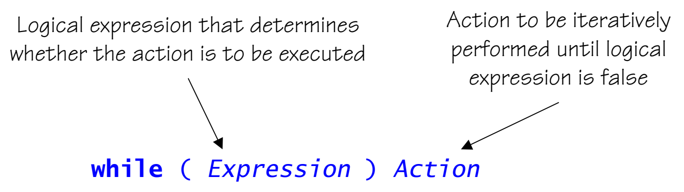
- Looping
- `while` statement
- `for` statement
- Common looping problems
- EzWindows API
- File stream extraction

4.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab04.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

4.2 LOOPING

C++ provides several looping mechanisms. In this lab, you first consider the `while` and then the `for` statement. As a reminder, a `while` statement has the following form:


Logical expression that determines whether the action is to be executed
Action to be iteratively performed until logical expression is false
while (Expression) Action

Consider the following program that is meant to display the sum of the integers in the range 1 … `n`, where `n` is a user-supplied positive value. The program has two mistakes.

```
#include <iostream>
using namespace std;
int main() {
   cout << "Enter a positive integer: ";
   int n;
   cin >> n;
   int i = 1;
   int Sum = 0;
   while (i <= n) {
      Sum = Sum + n;
      ++i;
   }
   cout << "The sum from 1 to " << n << " is "
    << Sum << endl;
   return 0;
}
```

- One mistake is a matter of good programming practice—the user-supplied value is not verified to be positive. Determine where a modification should be made so that if a negative value is supplied, the inverse of that value is used instead. The modification should notify the user that a change was made to the input. The inverse can be computed in the following manner:

  ```
  n = -n;
  ```

- The second mistake in the program takes place in updating `Sum`. Identify the mistake and determine what the corrected expression should be.
- Start the compiler as described in previous labs. Open the file called `sum.cpp`. It should contain the preceding program along with some additional comments.
- Determine manually the sum of the integers from `1` to `n` for the following values of `n`: `3`, `5`, `10`. Record your answer in the following table.
- Correct the two mistakes. Then build and run the program to observe the output using the preceding values for `n`. (You will need to create a default

n	manual	program
3		
5		
10		
-5		
0		

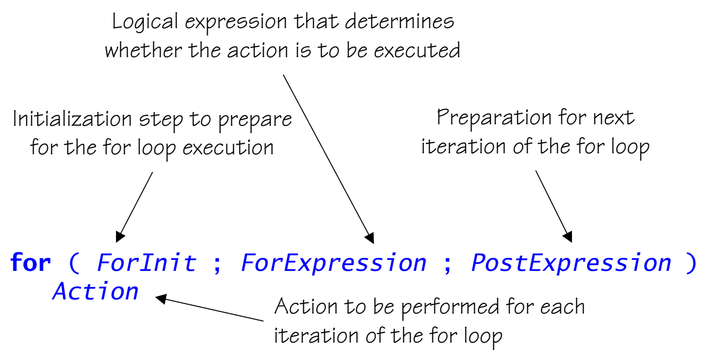
workspace.) Also try -5 and 0 as inputs. Record the program answers in the preceding table.

- Did you get the same answers for your manual calculations as you did from the computer program? If not, try to figure out why. If you cannot determine the reason, seek help.
- Let's modify the program to calculate a different sum. Rather than calculating the sum of integers from 1 to n, you will instead calculate the sum of integers from m to n, where m and n are user-supplied values and $m \leq n$.
- The program requires several changes.
  - — Two input values m and n must be prompted and extracted.
  - — n does not have to be positive.
  - — The value of m must be verified to be less than n. If m > n, then swap the values of m and n and tell the user of your correction.
  - — Object i, instead of taking on the values 1 … n, must take on the values m … n.
- Modify sum.cpp so that it correctly implements the preceding changes.
- Manually determine the sum of values in the intervals -5 … -1; -3 … 3; and 5 … 10. Record your answers in the following table.

m … n	manual	program
-5 … -1		
-3 … 3		
5 … 10		
8 … 6		

- Build and run the modified program to observe the output using the specified intervals. Also try the interval 8 … 6. Record your program answers in the preceding table.

- Does your program perform correctly? If not, try to figure out why. Show your results to your laboratory instructor. ✔
- Modify `sum.cpp` to use a `for` loop instead of a `while` loop, where a `for` statement has the following form.


Logical expression that determines whether the action is to be executed
Initialization step to prepare for the for loop execution
Preparation for next iteration of the for loop
Action to be performed for each iteration of the for loop

```
for ( ForInit ; ForExpression ; PostExpression )
   Action
```

- For our problem the initialization step component of the `for` statement should define and initialize object `i`. The loop test expression does not need to change. The post expression needs to increment `i` once for each updating of `Sum`. The updating of `Sum` should be the sole action of the `for` loop body.
- Build and run the modified program to observe the output using the same intervals as previously. Record your program answers in the following table.

m … n	program
-5 … -1	
-3 … 3	
5 … 10	
8 … 6	

- Does your program perform correctly? If not, try to figure out why. Show your results to your laboratory instructor. ✔
- Save and close `sum.cpp` workspace.

Consider the following program.

```
#include <iostream>
using namespace std;
int main() {
    int Counter1 = 0;
    int Counter2 = 0;
    for (int i = 1; i <= 10; ++i) {
        ++Counter1;
    }
    for (int j = 1; j <= 15; ++j) {
        ++Counter2;
    }
    cout << "Counter1: " << Counter1 << endl;
    cout << "Counter2: " << Counter2 << endl;
    return 0;
}
```

- Determine the output of the program and record your answers in the following table.

count1.cpp	manual program
counter1	
counter2	

- Open the file called `count1.cpp`. It should contain the preceding program along with several additional comments.
- Build and run the program. (You will need to create a default workspace). Record the program output in the preceding table. Did you get the same answers for your manual calculations as you did from the computer program? If not, try to figure out why. If you cannot determine the reason, seek help.
- Close the current workspace.

Now consider the following program:

```
#include <iostream>
using namespace std;
int main() {
    int Counter1 = 0;
    int Counter2 = 0;
    for (int i = 1; i <= 10; ++i) {
        ++Counter1;
        for (int j = 1; j <= 15; ++j) {
            ++Counter2;
        }
    }
    cout << "Counter1: " << Counter1 << endl;
    cout << "Counter2: " << Counter2 << endl;
    return 0;
}
```

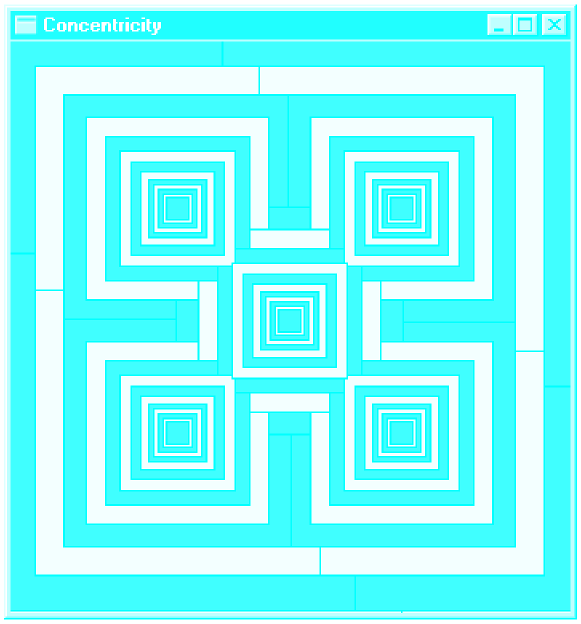
- Determine the output of the program and record your answers in the following table.

count2.cpp	manual program
counter1	
counter2	

- Open the file called `count2.cpp`. It should contain the preceding program along with several additional comments.
- Build and run the program. (You will need to create a default workspace). Record the program output in the preceding table. Did you get the same answers for your manual calculations as you did from the computer program? If not, try to figure out why. If you cannot determine the reason, ask for help.
- Why do `count1.cpp` and `count2.cpp` produce different results?
- ✎ These simple programs demonstrate the power of nested looping—with nested loops, actions can take place a significant number of times. Show your results to a laboratory instructor. ✔
- Close the current workspace.

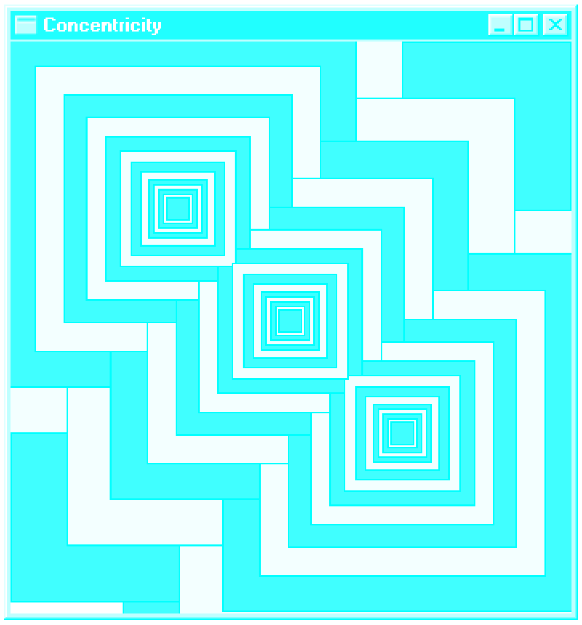
4.3 CONCENTRIC RECTANGLE DISPLAY

The next exercise shows how loops can accomplish a significant amount of work using a small number of statements. You will be drawing a geometric picture that should resemble the one in the following figure. The picture has five elements and each element is a series of concentric rectangles that alternate in color.



- Open the project `five.dsw` that uses the EzWindows API.
- Open the file `five.cpp` of project `five.dsw`. The principal component of that file is a function `ApiMain()`. As written, the function displays only

three of the concentric elements. If you run the program with this ApiMain(), the output would resemble the following display.



- The initial section of function ApiMain() first defines several constants. This section also defines and opens the window that contains the graphical displays.

```
const float Size = 10;
const float Cx = Size / 2.0;
const float Cy = Size / 2.0;
const float offset = Size / 4.0;
const float n = 15;
const float Scaling = 0.8;
SimpleWindow W("Concentricity", Size, Size);
W.Open();
```

- Object Size represents the width and height of the window that contains the concentric rectangles. Objects Cx and Cy represent the center coordinates of the window. You use constants Size, Cx, and Cy to construct and open object W. The concentric rectangles are displayed in W. Object Offset represents one-quarter of the length of a side of window W. You can use Offset to help determine the centers of the concentric rectangles that abut the sides of the window. Constant n represents the number of concentric rectangles drawn per series. Object Scaling represents the factor by which the concentric rectangles shrink.

- Function `ApiMain()` next defines an object `Side`. Object `Side` represents the current size of a side of the rectangles being drawn. The first rectangles that are displayed have sides whose lengths are equal to `Size`.

  ```
  float Side = Size;
  ```

- A `for` loop occurs next in the function. The loop iterates `n` (15) times. In each iteration, three rectangles are drawn—one rectangle for each of the three concentric series being displayed. (Your job is to modify the code so that it also displays the other two series.)

- The first task of the `for` loop is to determine the color of the rectangles to be drawn for the current iteration. If the value of `i` is even, the rectangle is yellow; otherwise, the rectangle is blue.

  ```
  color C;
  if (i % 2 == 0)
     C = Yellow;
  else
     C = Blue;
  ```

- Three rectangles are then constructed: `R1`, `R2`, and `R3`. In each iteration, `R1` represents the current rectangle from the concentric series that abuts the upper-left corner of window `W`. Objects `Cx`, `Cy`, and `Offset` determine the center of this rectangle. By subtracting `Offset` from `Cx` and from `Cy`, you are indicating a rectangle closer to the origin with respect to both the x-axis and y-axis.

  ```
  RectangleShape R1(
   W, Cx - Offset, Cy - Offset, C, Side, Side);
  ```

- Rectangle `R2` represents the current rectangle from the series that abuts the lower-right of window `W`. A positive `Offset` added to the center coordinate gives `R2` the correct location.

  ```
  RectangleShape R2(
   W, Cx + Offset, Cy + Offset, C, Side, Side);
  ```

- `R3` is constructed to be at the center of window `W`. Because it is at the center, object `Offset` is not needed.

  ```
  RectangleShape R3(W, Cx, Cy, C, Side, Side);
  ```

- Once the three rectangles are constructed, they are drawn. Note that the order in which the rectangles are drawn can affect the display.

  ```
  R1.Draw();
  R2.Draw();
  R3.Draw();
  ```

- To prepare for the next iteration, object `Side` is updated. In the next iteration, you draw rectangles whose size is 80 per cent (the value of `Scaling`) of the current rectangles.

  ```
  Side = Side * Scaling;
  ```

- Now modify the `for` loop body to construct and draw two more rectangles per iteration. These rectangles should take care of the corners of the window currently being ignored.
- ✎ Demonstrate your completed work to a laboratory instructor. Why are the rectangles from one iteration left on the screen for the next iteration? ✔
- Before proceeding to the next section, close the current workspace.

4.4 USING INPUT FILE STREAMS

We often need to examine a data file using some program or command. There are also instances where we want to read data both from a file and the keyboard within the same program. Similarly, we may want to have display output both to the monitor and a file. In this exercise, you learn to use input file streams to extract data from a file. The other types of processing will be considered elsewhere.

- Open the file `wc.cpp`. The program in this file counts the number of words in the text that it extracts. As written, the program extracts its values from standard input.
- Examine the program to get a sense of how it accomplishes its task. In particular, observe that the program uses the library function `isspace()`. This function, when a given a character value as its parameter, returns true if the character is white space and returns false otherwise. By default, a character is considered white space if it is a blank, a tab, a form feed, a new-line character, a backspace, or a carriage return character.
- Compile and run the program. (You will need to create a default workspace). Note that the program does not issue a prompt to the user to start supplying text. Type the following text as your input. When you are finished typing the text, you need to signal to the program that you are done. On most PCs, typing a line consisting solely of a `CTL-z` followed by `Enter` produces the signal; other systems often use `CTL-d` followed by a carriage return.

  ```
  C++ is the greatest.
  This course is fantastic.
  This-line-is-a-single-word.
  ```

- Make sure that the answer provided by the program agrees with your expectation.

You will now modify the program to extract values from a file rather than from standard input. The class `ifstream` is the input stream type for representing a file. The type is defined in the standard library file `fstream`. In this example, you define and use a stream object named `fin` that represents the input file stream that you need to process. The only thing special about the name `fin` is that it is reminiscent of `cin`; for example, you could have also used the name `FileIn`.

The first thing you must do is associate `fin` with the actual file. Suppose the input file was contained on the `c:` drive of our system in a file named `example.txt` in the directory `\text` of the `c:` drive. To define and initialize `fin` to be associated with that file, you do the following:

```
ifstream fin("c:\\text\\example.txt");
```

The two backslashes are necessary because a backslash in C++ indicates the start of a special character sequence (e.g., `\n` represents the new-line character). The preceding code statement defines an object `fin` of type `ifstream` that will do its extracting from the file `example.txt`.

- Using the above definition as a model, use the file `input1.txt` as the input file from which `wc.cpp` does its extractions. The file is in the same directory as `wc.cpp`. Add your definition of `fin` before the `while` loop.
- At the beginning of the file `wc.cpp`, add the following preprocessor line so that the definitions in the `fstream` library are part of your program.

  ```
  #include <fstream>
  ```

- These changes allow the stream `fin` to be an input source. To extract values from it, you need to change the stream being processed in the `while` loop test expression from `cin` to `fin` (i.e., the program performs a `fin.get(CurrentCharacter)`. Make this change to the program.
- Open the file `input1.txt` and examine its contents. Record your expectation of the program's output in the following table.

file	manual program
`input1.txt`	
`input2.txt`	

- Compile and run the program in `wc.cpp`; make sure you agree with the program output.
- Modify the definition of `fin` so that it is now associated with the file `input2.txt`, which is located in the same directory as `input1.txt`.
- Open the file `input2.txt` and examine its contents. Record your expectation of the program's output in the preceding table.
- Recompile and run the program in `wc.cpp` and make sure you again agree with the program output. Show your results to your laboratory instructor. ✔
- Close all existing files.

4.5 FLAWED LOOPING

Improper initialization statements and termination conditions are often major causes of program misbehavior. The next examples illustrate some common mistakes.

- Open the file `fraction.cpp`. Examine the program to get a sense of what occurs during execution.

```
#include <iostream>
using namespace std;
int main() {
   int Total;
   int n = 2;
   int Fraction = 1/n;
   int LoopCounter = 0;
   for (Total = 0; Total != 1; Total += Fraction) {
      ++LoopCounter;
   }
   cout << "The total is " << Total << "\n";
   return 0;
}
```

- Run the program and observe its output or in this case lack of output. (You will need to create a default workspace). The program has an infinite loop. To terminate the program, you must enter an escape sequence. On PCs this sequence is normally CTL-ALT-DEL. A help window should appear, and you can then terminate the errant process. Sometimes as a result of the infinite loop, you are forced to restart the machine without saving your program. In this case you will lose your unsaved modifications. Therefore, as a precaution, you should always save a program before running it.
- Using your debugger, observe the values of objects `LoopCounter`, `Fraction`, and `Total`. Step through the program to determine the problem.
- ✎ Correct the problem with the object types and again step through the program to observe the behavior of the objects. The program should now work correctly. Show your results to a laboratory instructor. ✔
- Modify `n` so that it is **`float`** and has the value `10.0`. Again step through the program and observe the output. If necessary, terminate the program.

✎ The first time you ran the program, a mismatch occurred in the types of fractional expression and the object `Fraction` itself. That error is most likely not the problem with this run. Therefore, a different problem is occurring in the loop. The problem now is that the fraction `1/10.0` is not represented perfectly by the machine. As a result, the desired sum is not achieved. Is this problem something you can fix or overcome? Show your results to a laboratory instructor. ✔

- Close the existing files and then open the file `upper.cpp`.

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

```
int main() {
   int NbrUpperCase;
   char CurrentCharacter;
   while (cin >> CurrentCharacter) {
      if (isupper(CurrentCharacter)) {
         ++NbrUpperCase;
      }
   }
   cout << "Upper case chars: " << NbrUpperCase
    < endl;
   return 0;
}
```

- Examine the program to get a sense of what occurs during execution. Modify the program so that it reads from the file `input3.txt`. Make and run the program.
- ✎ The program does not produce the correct output, does it? Step through the program and trace the behavior of object `NbrUpperCase`. Fix the program and again step through it to verify your correction. Show your results to a laboratory instructor. ✔

These examples have demonstrated representation error, numerical inaccuracy, and lack of initialization. Such problems can lead to infinite loops and incorrect output. A useful programming technique is to develop a *loop invariant* for each loop. The invariant is a Boolean expression or predicate that describes what you expect to be true each time through the loop. By developing code that ensures the correctness of the invariant, you can increase your chances of producing correct code.

4.6 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.