

CST8110: Introduction to Computing

Lab 5: Use the Debugger to Trace Program Execution

Name: _____

Lab Section: _____

Overview

Programming can be challenging at times. There are all sorts of new abstract concepts to learn. I want to show you a tool that gives you a concrete experience of these abstract concepts. The tool is called the *debugger*. This tool allows you to step through program execution and watch program activity move from one instruction to the next. While the program is running, you will be able to:

- View values stored in variables in RAM.
- Dynamically modify values stored in variables.
- Watch the sequence of execution. When program execution moves through a **while** loop or a **for** loop, you can watch the sequence cycle through these loops again and again. When program execution moves through an **if** or an **if-else** construct, you can trace the path that is followed.

This lab exercise is to be completed individually, during the lab period. You can ask questions of me or others in the class.

Details

Building the Program

Here's the program to test (also available on my website).

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

int main()
{
    const int LARGEST_INT = 0x7fffffff;
    // Repairing an endless loop at runtime
    // loop should display 0 to 9 (iterate 10 times), but never moves beyond 0
    cout << "\nRepairing a badly written counting loop\n";
    int nIncrementValue = 0; // silly value, but let's see the behaviour
    int nCount = 0;
    while (nCount < 10) {
        cout << nCount << endl;
        nCount = nCount + nIncrementValue;
    }

    // Displays the powers using the int data type
    // It starts out fine, but then tumbles into an endless loop, displaying 0's
    cout << "\nRepairing a badly written Powers of 2 loop\n";
    int nPowersOf2=1;
    cout << "Bytes for 'nPowersOf2': " << sizeof(nPowersOf2) <<
        ". Maximum value for an int: " << LARGEST_INT << endl;

    while (nPowersOf2 < LARGEST_INT) {
        nPowersOf2 = nPowersOf2*2;
        cout << nPowersOf2 << endl;
    }

    // Displays the powers using the unsigned char data type
    // But it displays odd-looking characters rather than numbers
    const char LARGEST_CHAR = 0x7f;
    cout << "\nRepairing a badly written Powers of 2 loop\n";
    unsigned char cPowersOf2=1;
    cout << "Bytes for 'cPowersOf2': " << sizeof(cPowersOf2) <<
        ". Maximum value for a char: " << LARGEST_CHAR << endl;

    while (cPowersOf2 < LARGEST_CHAR) {
        cPowersOf2 = cPowersOf2*2;
        cout << cPowersOf2 << endl;
    }
}
```

Copy the preceding code into a newly created project in Visual Studio. (Note: If you use any other compiler, the instructions presented here may not work.) The code is also available on my website.

Compile the program (using the function key <F7>). It should compile with no syntax errors (though there are a couple of logic errors that you will explore in this lab exercise). If you do encounter syntax errors, you can use the <F8> function key to move to each of the error messages.

Run the program under the debugger using the <F5> function key.

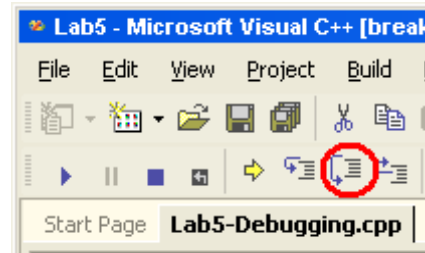
Once program execution begins, you'll notice that the program displays the value 0 endlessly. **The program is trapped in an infinite loop.** You can suspend program execution, therefore interrupting this infinite loop and then inspect the state of the interrupted program. Since I know where the problem is, I'll show you where to inspect the program code.

Look to the first loop in the program, and insert a breakpoint. The breakpoint is turned on using the <F9> function key. Here, I show the section of code where you'll insert the breakpoint. A red dot will appear on the left side when you successfully insert it (overlaid with a yellow arrow to indicate that program execution has suspended at this line of code).

```
int main()
{
    const int LARGEST_INT = 0x7fffffff;
    // Repairing an endless loop at runtime
    // loop should display 0 to 9 (iterate 10 times), but never moves beyond 0
    cout << "\nRepairing a badly written counting loop\n";
    int nIncrementValue = 0; // silly value, but let's see the behaviour
    int nCount = 0;
    while (nCount < 10) {
        cout << nCount << endl;
        nCount = nCount + nIncrementValue;
    }
}
```

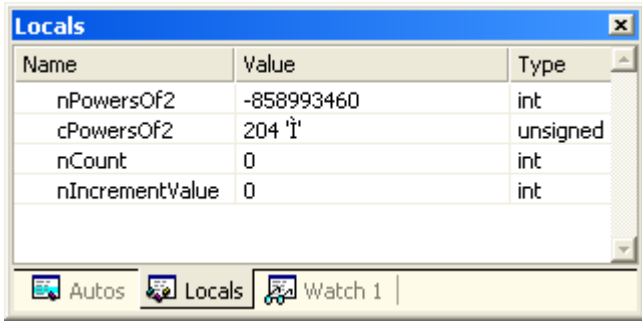
Stepping Through Code Under Debugger

With program execution now in suspension, you can step through the code to watch what's happening. You can use the following button to step a line at a time:



While stepping through the code, you'll want to inspect the values of variables that are *declared* and *defined* in your program. These values are most easily viewed through the *Locals* debug window¹. It should look something like this:

¹There are three related *Debug* windows: *Locals*, *Autos* and *Watch*. All three show values in variables, but present the variables a little differently. The most useful is the *Locals* window.



The **while** loop is to be a counting loop. That is, it is supposed to display the values from 0 to 9, by adjusting the value stored in **nCount**.

Identify the line of code where **nCount** is supposed to be incremented (increased by 1 on each cycle of the loop).

What is the value of **nIncrementValue** (the variable holding the amount to by which **nCount** will be increased)?

What should the value of **nIncrementValue** be?

Make the change to the line of code setting the initial value of **nIncrementValue**. In a moment, you will re-run the program to see how it behaves with the correction. But before you do, make sure that the breakpoint is still set (is the red dot still visible on the target line of code).

- Begin program execution with the function key <F5>.²
- Program execution should hit the breakpoint, and the program will go into suspension. The red dot will now be superimposed with a yellow arrow — indicating the line of code which will be executed next.
- Ideally, you can position the windows on your screen to give a view of both the compiler environment (Visual Studio) and the executing program (the black and white console screen). Ask me or another student, if you need help doing this.
- Ensure that you can see the *Locals* debug window. List the names of the variables and the values that are shown in this window (there should be four)

Variable Name	Variable Value

Why are some of these starting values so strange?

- Execute the current line of code using <F10>. Show the value that has changed?

Name	Value

Dealing with a Different Endless Loop

The next loop is supposed to display the powers of 2. But it doesn't behave as expected.

- Execute the code using <F10> to step a line at a time, and record the output that appears on the black and white console output screen. The loop progresses nicely for a while, but then some unusual behaviour develops (write the output in two columns to make it fit the box here):

- Check out the value of the constant called **LARGEST_INT**. Do this by letting the mouse drift over the label **LARGEST_INT**. Record the value that appears in the popup hint window.

² Because you probably made changes since the last execution, the compiler will announce that the project configuration is out of date and offer to let you build it. To this, you will answer Yes.

Name	Value
LARGEST_INT	

Here's a classic problem with *integer* family variables. As the integer gets larger and larger, more and more of the bits turn from **0** to **1**. Eventually, the *sign-bit* is also flipped to **1**, driving the number negative (even though we were trying to make the number a larger positive number. And since this negative number is clearly less than the value of **LARGEST_INT**, the program continues to loop . . . endlessly.

- To fix it, change the variable **nPowersOf2** to an **unsigned int**. You will need to terminate program execution using <Shift-F5>. Make the edit change, adding the word **unsigned**.
- Following your edit change, restart program execution with <F5>. The program should behave as expected now.

Dealing with Unusual Display Characters

Set a breakpoint on the following line of code in the last loop of the program:

```
unsigned char cPowersOf2=1;
```

A **char** (either **signed** or **unsigned**) is a member of the *integer* family of numbers, but it only allocates 1 byte (8-bits) of storage for a single **char**³. So we would expect it to display the following: **1 2 4 8 16 32 64 128**

What is displayed as the value for **cPowersOf2** in the *Locals* debug window each time through the loop? (Note: There are two things: a character symbol and a corresponding number.)

To correct this, you need to tell **cout** that the value it is receiving should be interpreted as an **int** instead of its real type, that is, **char**. Use the *cast* operation to do this. The **cout** line of code would become:

```
cout << int(cPowersOf2) << endl;
```

Try running the program again and check the output.

Evaluation Process

- I will check your work and assess it during the lab period.
- You will keep the document as a reference for future debugging work.

Total / 9

³Remember an **int** consumes 4 bytes (32-bits) of storage.
Lab 5: Use the Debugger to Trace Program Execution