Lab Exercises wk04 – Debugging Code with NetBeans

Required Reading

None

Instructions – PLEASE READ *(notice **bold** and underlined phrases)*

Lab Exercise has three parts:

A. Lab Demo – Watch Demo, reproduce it, show your TA
B. Exercises – No lab exercises. Please work on completing Assignment #2
C. Submission – Nothing to submit this week

1. **You are encouraged to work on these lab exercises in teams of two, and each group must reproduce the demo and show the TA that they have completed Part A. Team members should alternate turns at the keyboard typing in the code during Part A.**
2. **Submission deadline: Friday Jan 29th at 10:30am**
3. **The exercises are presented in sequence so that you gradually advance with the material.**
4. **Before you leave the CSIL labs, make sure that a TA looks at your work in order to receive your attendance and lab active participation marks.**
5. **Lab04 Intended learning outcomes**

By the completion of the demo, students should be able to use NetBeans to:

- Single step through programs and methods
- Set breakpoints and view program state and internal variables
- Set Command Line parameters through the NetBeans IDE

A. Lab Demo – Presented by TAs, and repeated by Students

Students must observe demo presented by TA, then reproduce it.

Student must have the TA check off demo program completion by end of tutorial for full marks. Marks will be awarded for attendance but only partial marks for incomplete work at the discretion of the TA.

Even though students may work in teams, <u>each team member</u> must create a running project based on the demo for full marks.

Debugging Code with NetBeans

In this Lab04 demo, you will learn how to debug code using NetBeans. NetBeans is a powerful IDE that allows you to monitor what your program and methods are doing, how they affect variables and state, and use it to locate bugs. This is a far more powerful and efficient debugging technique than adding `cout` statements throughout your code.

In this Demo, we will be using a C++ program that formats text strings as either left justified, right justified, or centered. You will need the following file which is available on the CMPT 135 Course website called:

- **`Lab04Main.cpp`**

To get started,

1. **Create Project `Lab04Justify`**

Create a project called `Lab04Justify` and import the source file for Lab04Main.cpp from the Class website.

- Download Lab04Main.cpp and place it in your Downloads directory, or some other temporary location.
- Start NetBeans. Make a new project called **Lab04Justify**.
- Click on 'Source Files' under the Lab04Justify project to reveal main.cpp. Right-click on it, and select 'Rename', Rename it to Lab04Main.cpp.
- Right click on '**Lab04Justify**' project in the Package Explorer view and select 'Properties' from the menu. Select "C++ Properties" and ensure that the "C++ Standard" is set to **`C++11.`** If it is not already set that way, select it from the menu on the right hand side of that option.
- From the Ubuntu desktop, open the Filing Cabinet and go to your Downloads directory. Double click on file Lab04Main.cpp. By default, it will be opened by GEdit. Click on the window, then press <ctrl-a> to select the entire file, and <ctrl-c> to copy the entire file to your clipboard.
- Click on your NetBeans window, and double-click on Lab04Main.cpp to open it in NetBean's main edit window. Click on the window to select it.

Press <ctrl-a><ctrl-v> to over-write the existing file with the contents of Lab04Main.cpp from your Downloads directory.

- You should now have the contents from your downloaded Lab04Main.cpp in your Lab04Justify NetBeans project.
- **Verify this by clicking on the Green Arrow to "Run Project"**. Your program should compile and run. The output should indicate that it is using default parameters with Width 20 and Justification set to Left. However, there is a bug in the program and should then report "**`Error: No text specified. Exiting...`**".
- The program is ***supposed*** to accept a set of command line parameters and justify the text according to the displayed Usage statement. If no text is specified, it is supposed to Left-Justify the default text about the quick brown fox.
- Instead of using the defaults, it is reporting an error. Your job is to debug this code and fix it!

2. **Open a Debug Perspective**
   - Up to now, we have been using the NetBeans **Run** configuration. In this configuration, we can run and test our program but we cannot set breakpoints or debug our program very easily.
   - The Debug configuration is oriented towards debugging.
   - Enter this configuration by either hitting the "Debug Project"  button beside the green "Run" button, or go to the top level menu bar, and select "Debug Project" from the Debug menu, or simply hit <ctrl-F5>.
   - When you enter this mode, nothing much may appear different but it is using a different configuration. When you enter this mode, it will likely recompile your code and run it.
   - The Output window at the bottom will still contain the output from the program that just ran but now we will be able to step through the running program and display variables at different points in the execution. This will allow us to find the bug that is causing our program to report an error.

3. **Executing a Program One Line at a Time**

   A basic ability of any good IDE is the ability to control the execution of a program — to be able to stop the program when and where you want and view its current state. The simplest example of that (from a user perspective) is line-by-line execution. This is called *Single Stepping* through a program,.

   a. **In the Edit window, scroll through Lab04Main.cpp until you find main() around line 221. Click on the line number beside where**

main() starts and a red square should appear on line 221. This indicates that you have just set a breakpoint at the start of main. Do the same thing on the first IF statement after main around line 224. You should have two breakpoints set now.

b. Now when you hit the "Debug Project"  **button, the program will stop at the first breakpoint at the start of `main()`. Because it stopped, there is no output yet produced. The program is stopped at this breakpoint waiting for you.**

c. Look at the line under main() where Parms is declared. Notice that there is a green arrow where its line number used to be. This indicates that this is where the program will resume once you tell it to continue running.

d. **When you are at a break point, you can display the values of any variables that are in scope, or set more breakpoints, or tell the debugger to continue until it hits the next breakpoint, or reaches the end of the program.**

e. **In bottom window, besides the output window, you will notice there are three new tabs: the Variables Tab, the Call Stack Tab, and the Breakpoints Tab. You may also notice that at the very bottom there is a little orange ball bouncing back and forth: this is telling you that you are in Debug Mode and that your program is waiting for you. Go to the Window menu at the top, select Debugging, and you will see a list of all the various debug windows you can use. We will stick to these three for now.**

f. **Click on the Variables window. Notice that the Variables window shows three variables: `parms`, `argc`, `argv`. At this point in the program, these are the only variables in scope.**

g. **This window displays the values of argc and argv which have well defined values at this point. Parms has not yet been initialized by our program and so contains uninitialized values.**

h. **Argc is set to one because only one command line argument was passed in: the name of the program itself. Argv contains a pointer to a pointer to the name (because it is an array of pointers to character arrays - but you do not need to know these details yet!).**

i. **Click on the Call Stack window. Notice that it only contains one line containing main() and its two parameters (argc=1, argv=0x7fffff…). When main() calls ParseCmdParms, it too will appear in this window along with its parameters. This window can be very useful for understanding how a function was called when it had a bug.**

j. **Now click on the Breakpoints window. This contains the set of breakpoints where you have told your program to stop. At this point, we have set two: both in Lab04Main.cpp at lines 221 and 224. Notice that the green arrow is indicated on the first breakpoint. That is where are program is stopped.**

k. **Hit the Continue button**  **(or F5 button) to continue execution until it hits the next breakpoint. Notice that the IF statement on line 224 is now highlighted in green and the green arrow in the Breakpoints menu has moved to the second break point.**

l. **If we hit Continue again, the program would run to the end because there are no more breakpoints. But breakpoints are not the only way to debug. We can also single step through the program execution.**

m. **Let's single step into the ParseCmdParms routine by hitting the StepInto**  **button (or F7 button). This tells the debugger to execute the next step in the program. If that step is a procedure call or function call, it will step into the routine and stop, waiting for the next command. You may notice that before it enters ParseCmdParms, it first shows the declaration of struct Parms which it needs as a parameter for the call to ParseCmdParms. Hit the StepInto button a several times until you reach the switch statement.**

n. **Now look at the Call Stack window once again. It still contains main(), but now ParseCmdParms has been pushed onto the stack along with its parameters and their values.**

o. **Click on the Variables window. Now you can view all the variables that are in scope within ParseCmdParms. Argc is still one, Argc is unchanged, but parms has not been initialized thanks to the three statements before the switch statements.**

p. **Review the switch statement and the value of argc which is used by the switch statement. Can you predict where StepInto will go next? Hit StepInto and see if you are correct.**

q. **The debugger went to the `Case` statement where `argc == 1` and is about to do a `cout` statement.**

r. **We could StepInto each of those cout statements, but that would not be interesting. Instead, click on the line number associated with the return statement to create a new breakpoint there.**

s. **Now hit the Continue button and the debugger will skip past all those cout statements and stop at the return statement, just before ParseCmdParms is about to return to main.**

t. **In the Variable Windows, notice that value of parms. It has a width of 20, it is specifying Left justification, and its inputText is set to something but it is unclear what. Click on the dots icon  on the right, and NetBeans will pop up a new window to show you the rest of the contents of parms. Can you see what inputText is set to now? You should see the string value near the end of this display. No need for you to understand the gory details, just notice that the field inputText is set to "The quick brown fox jumped over the lazy dog".**

u. **Close the window and hit the StepOut  button (or the <ctrl-F7> button). The StepOut button continues running the program until it completes the current function and returns to the caller where it then stops once again, waiting for the next debug commend. In this case, it will continue running until it returns to main() following the call to ParseCmdParms.**

v. **Now look at the Variables window and check what the current value of parms is now. Does it still contain the values that it had within the ParseCmdParms procedure? If not, then why not?**

w. **To exit debugging mode, terminate the current debug session by hitting the Finish Debugger Session  Button (or the <shift-F5> button).**

x. **You can remove individual breakpoints by click on the red square where the breakpoint is, and it will be deleted and the line number redisplayed.**

y. **To delete all breakpoints, go to the BreakPoint window and right-click to get the breakpoint menu. From there, you can select Delete-All to delete all breakpoints, or Disable-All to disable the breakpoints for now (but you can reenable them later).**

z. **Once you have deleted (or disabled) all the breakpoints, you can hit the Continue button to complete the execution of the program to the end.**

4. **Fix Lab04Main.cpp so that ParseCmdParms works and show the TA**

a. **Fix ParseCmdsParms so that parms is set correctly when the function returns. Try to solve this on your own, or with your partner. You can ask your TA for help if you get stuck.**

b. **When you have fixed the program, it should display the following in the output window when you run it:**

```
Using default  parameters:
 Width=20 Justification=Left Text="The quick brown fox jumped over the lazy dog"

The quick brown fox
```

```
jumped over the lazy
dog

RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms
```

- **Show your TA this output for partial Participation Marks. You must also get Command Line Processing working for full Participation Marks. Keep going!**

5. **Command Line Arguments**

Many programs accept command line arguments when they are invoked. For example, `Lab04Main.cpp` justifies the same text string every time in exactly the same way. We might instead want to use the command line to pass our own text to be formatted by this program. One might want to specify the width and justification type as left, right, or center.

a. Compile and test Lab04Main.cpp from the command line.

To call `Lab04Main.cpp` from the Command Line, you need to build it from the command line. Using a terminal window, go to your NetBeans folder with your Lab04Main.cpp program (using the `cd` command) and compile it using the following command:

```
g++ -std=c++11 Lab04Main.cpp -o justify
```

Once compiled, you can invoke it from the command line. Here is one example:

```
justify 20 C "This is the text to fill and justify."
```

This would produce the following output:

```
 parameters:
 Width=20 Justification=Center Text="This is the text to fill and justify. "

This is the text to
 fill and justify.
```

b. Testing Command Line Parameters using NetBeans

To set the command line parameters using NetBeans, select the Project Lab04 from the project explorer window, and right-click and select properties.

From the Properties window, select the "Run" category. You will see a menu of Configuration Options. At the top you can select either the "Release" configuration or the "Debug" Configuration. Select "Debug" for running with the debugger (the other option is the configuration used when you hit the RUN button instead of the DEBUG button).

The first Configuration option under "General" is called "Run Command". This is where you set the command line parameters within NetBeans.

The Run Command starts with "${OUTPUT_PATH}". Click on this, and add your command line parameters after this string. Set this configuration

Option as follows: (all on one line)

"${OUTPUT_PATH}" 35 Right Everything should be made as simple as possible, but no simpler. - Albert Einstein

Run Lab04Main.cpp from NetBeans with the Run Command set as above. The program should produce the following output:

```
 parameters:
 Width=35 Justification=Right Text="Everything should be made as simple as possible,
but no simpler. - Albert Einstein "

Everything should be made as simple
     as possible, but no simpler. -
                  Albert Einstein

RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms
```

c. **Update the Block Comment at the top of Lab04Main.cpp with your name, student number, email address, etc. Students need not update or enhance the other comments associated with the functions within this file.**

Check in with the TA

Demonstrate to the TA that you are able to pass command line arguments to Lab04Main and have it right justify your text as shown above.

B. Lab Exercises – Please work on Assignment #1

There are no Lab Exercises this week. Please spend the remaining time working on the Assignment #1 programming problems. Use this time to ask your TA for help. You may discuss your work with your partner, but the code you submit for the assignment must be your own.

Now that you know how to use the NetBeans debugger, you will find that debugging your programs will be much easier. It might even be fun.

As you work on Assignment #1, please realize that you will not only be graded on program correctness, but also on program clarity, efficiency, and simplicity.

Once you have a working program, be sure that you comment it well. When you think it is done, go back and look for ways to make the code clearer, more efficient, and less complex.

As Albert is famous for saying: *Everything should be made as simple as possible, but no simpler*.

C. Lab Exercise Submission – To be completed by Students

Students are responsible for submitting the requested work files by the stated deadline for full marks. Since Lab Exercise solutions may be discussed in class following the submission deadline, late submissions will NOT be accepted. It is the student's responsibility to submit on time. If you do not have access to CSIL or issues with the computers in CSIL, please contact the CSIL Help Desk at `helpdesk@cs.sfu.ca`

Students may work in teams of two for the lab, and submit a single set of files on behalf of the group in CourSys.

1. **You must submit your final version of the following file before the deadline. Students must ensure that all submitted code compiles and is properly commented and formatted for readability:**
   - **`Lab04Main.cpp`**
2. **For students working in the lab with a partner, only one submission is required for the group of two students**
3. **Files are to be submitted into CourSys under Lab04. Use the Manage Groups menu to create a group name unique to you and that includes the week number.**
4. **Attendance and Participation marks are based on everyone in the group. If you are the only person in the group, then your mark is your own. If there are two members in a group, both must attend the tutorial and make progress in order for either student to receive marks. So choose your partner wisely!**