

Name: _____

Lab Section: _____

NET2006: Object-Oriented Programming

Lab 2: Debugging and Tracing Code having Functions

Overview

This is an *in-class, individual* lab assignment. There are several goals. You will:

- Correct syntax errors.
- Trace program execution to correct logic errors.
- Trace program execution to understand how variables are sent as arguments.
- Use the *call Stack* debugging window to view function execution.

I've placed copies of this lab's source code on my website.

Table of Debugger Function Keys

You've seen this table before, but it's time to highlight two items and ensure that you understand the difference between the two. In addition, I have added a third item (which is also highlighted).

Summary of Useful Debugger Function Keys

| | |
|-------------|--|
| <Ctrl-J> | Match braces. |
| <F8> | Jump to next compiler error message and find matching code. |
| <Ctrl-F10> | Execute to the current cursor position |
| <F9> | Turn Breakpoint On or Off |
| <F11> | Step to the next line of executable code. If it is a function, then step into the function and show detailed steps. |
| <F10> | Step over the next line of executable code. If it is a function, then do all the work of the function but don't show detailed steps. |
| <Shift-F11> | Step out of the function. This is particularly useful if you stepped into a function, but you did not really want to see the details (for example, stepping into the detailed code associated with the use of <i>cout</i>). This allows you to get back to where the initial call was made without having to continue through every intervening line of code. |
| <F7> | Recompile if necessary. |
| <Shift-F5> | Terminate Program Execution |
| <Shift-F9> | Add a Watch value |
| <F5> | Begin Debugging (recompiling if necessary). |
| <Ctrl-F5> | Begin execution without using the debugger. |
| <Alt-Tab> | Switch to another application window (such as your black and white program execution window). |

Details: Program 1: Syntax Error

Here is the first program. It is also available on my website (so that you don't have to type it in).

```
// Lab 2-1 Debugging.cpp
#include <iostream>
using namespace std;

void main()
{
    char cUserChar;
    cout << "Enter a character: ";
    cin >> cUserChar;

    int nUserRepetitions;
    cout << "Enter number of times to repeat it: ";
    cin >> nUserRepetitions;

    RepChar(cUserChar, nUserRepetitions);
}

//-----
// RepChar() -- function definition
void RepChar(char ch, int nTotal) // Function declarator
{
    for(int j=0; j<nTotal; j++) // Function body
        cout << ch;
    cout << endl;
}
```

Dealing with the Syntax Error

As is commonly the case, the compiler doesn't recognize the error until long after the error occurred. So the correction must take place much earlier.

(1 mark) When you successfully compile the corrected program, identify the syntax corrections directly in the program code in this document.

Viewing Program State through the Call Stack Window

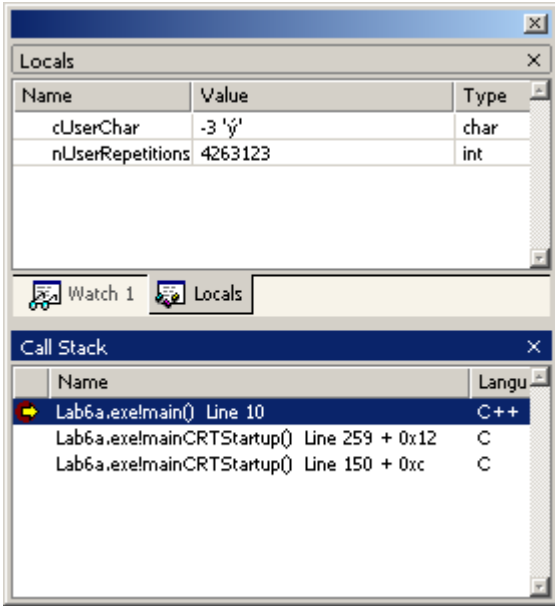
To understand how functions work, you will use the *Call Stack* debugging window.

The program is shown running on the next screen shot. You can see that program execution has suspended on the line of code with the yellow arrow (over the red breakpoint dot).

I have opened the *Locals* window to show values stored in variables. Since I suspended program execution at an early stage, (before variable initialization) the two local variables (*cUserChar* and *nUserRepetitions*) both contain garbage values.

I have also opened the *Call Stack* window. Notice that there are three items shown, the top-most of which indicates that program execution has suspended in *main()*.

I can step through program execution using <F10> until I get to the function call to *RepChar()*. At that point, I will use <F11>. Using <F11> allows me to step into the function called **RepChar()** and watch the detailed execution proceed.



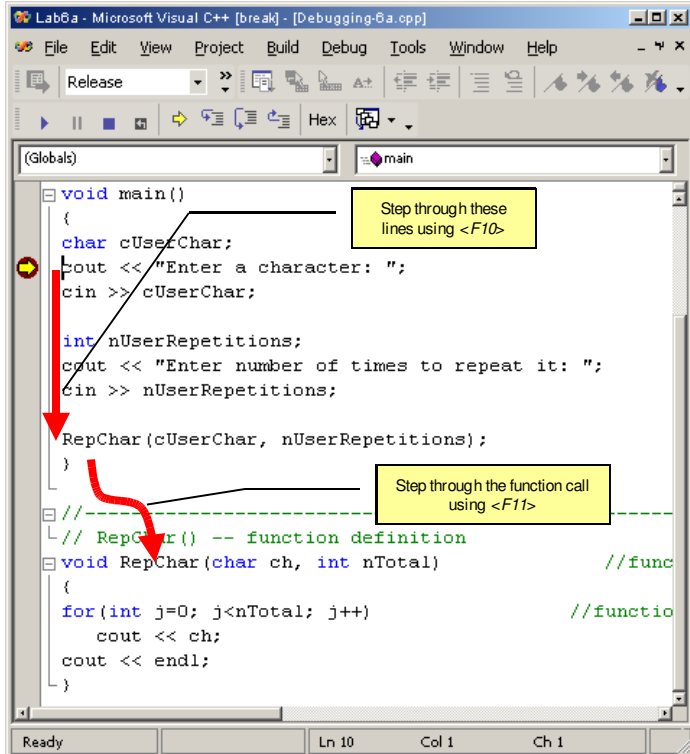
(1 mark) What shows up in the *Call Stack* window?

Continue stepping through the function **RepChar()** using *<F10>* again¹ until you exit the function.

(1 mark) Where does program execution go after leaving the **RepChar()** function?

(1 mark) What is now shown in the *Locals* window?

Let's assume that you've hit the breakpoint, stepped through to the function call (to **RepChar()**) with *<F10>* and then used *<F11>* to step into the **Repchar()** function.



(1 mark) What shows up in the *Locals* window?

¹ If you make the mistake of stepping with *<F11>* while stepping at a **cout** statement, program execution will step into all the detailed Microsoft code associated with **cout**. You don't want to be there. If that happens, you can step out again using *<Shift-F11>*.

Details: Program 2: Syntax and Logic

Here is the second program.

```
// Lab 2-2-Debugging -- demonstrates return values, converts pounds to kg
#include <iostream>
using namespace std;

void ConvertFahrenheitToCelcius(float fTemp); // Prototype / declaration

void main()
{
float fTemperature;

cout << "Enter a temperature in Fahrenheit: "
cin >> fTemperature;
cout << "You entered: " << fTemperature << endl
ConvertFahrenheitToCelcius(fTemperature);
cout << "The value is now: " fTemperature << endl;
}

// Converts temperature from Fahrenheit to Celcius
void ConvertFahrenheitToCelcius(float fTemp)
{
fTemp = (fTemp-32.0f) \ 9.0f * 5.0f;
}
```

Syntax Errors

There are three syntax errors. If you can recognize them without compiling, great! You're developing the skill of *pattern recognition*. If you can't see them right away, that's fine. Use the compiler to find them.

(3 marks) When you successfully compile the corrected program, identify the syntax corrections directly in the program code in this document.

Logic Errors

After correcting the syntax, the program will compile and will execute. However, there is one *logic* error. How does the behaviour of the buggy code differ from the working executable that you acquired from my website.

(1 mark) What behaviour tells you that the program has a logic error?

Using the debugger, you can trace program execution to identify where the error is occurring and then correct it.

(1 mark) Make a code change directly on the source code to correct the logic error? (Hint: You'll have to change the function so that it returns the newly calculated value. The returned value will then be assigned back into **fTemperature**.)

Details: Program 3: Logic Error

Here is the third program.

```
// Lab 2-3-Debugging.cpp
// demonstrates passing structure as argument
#include <iostream>
using namespace std;
//*****
struct Distance { // English distance
int nFeet;
float flnches;
};
//*****
void DisplayEnglishDistance(Distance dToDisplay); // Function declaration
void EditEnglishDistance(Distance dToEdit); // Function declaration
//*****
//int main()
//*****
int main()
{
Distance dLength1, dLength2; // Declare and define two Distance objects

// Get length dLength1 from user
cout << "Enter Feet: "; cin >> dLength1.nFeet;
cout << "Enter Inches: "; cin >> dLength1.flnches;

// Get length dLength2 from user
cout << "\nEnter Feet: "; cin >> dLength2.nFeet;
cout << "Enter Inches: "; cin >> dLength2.flnches;

cout << "\ndLength1 = ";
DisplayEnglishDistance(dLength1); // Display contents of first length
cout << "\ndLength2 = ";
DisplayEnglishDistance(dLength2); // Display contents of second length
EditEnglishDistance(dLength2); // Send second length object to editing
cout << "\ndLength2 = ";
DisplayEnglishDistance(dLength2); // Display length 2
return 0;
}
//*****
// DisplayEnglishDistance(Distance dToDisplay)
// display structure of type Distance in nFeet and flnches
//*****
void DisplayEnglishDistance(Distance dToDisplay) // Definition of function
{
cout << dToDisplay.nFeet << "\-" << dToDisplay.flnches << "\n\n";
}
//*****
// void EditEnglishDistance(Distance dToEdit)
//*****
void EditEnglishDistance(Distance dToEdit) // Definition of function
{
cout << "\nEditing existing object. dToEdit = ";
DisplayEnglishDistance(dToEdit);
cout << "Enter Feet: "; cin >> dToEdit.nFeet;
cout << "Enter Inches: "; cin >> dToEdit.flnches;
cout << "\nAfter input, dToEdit = ";
DisplayEnglishDistance(dToEdit);
}
```

Logic Errors

The program will compile and will execute as it stands. However, there is one *logic* error. How does the behaviour of the buggy code differ from the working executable that you acquired from my website.

(1 mark) What behaviour tells you that the program has a logic error?

(1 mark) What is the address of **dLength2**?

What is the address of **dToEdit**?

Using the debugger, you can trace program execution to identify where the error is occurring and then correct it.

(1 mark) Make a code change directly on the source code to correct the logic error? (Hint: You'll have to change the function so that it returns the newly edited **Distance** object **dToEdit**. The returned value will then be assigned back into **dLength2**.)

Using Reference Arguments to Fix the Logic Errors

There is an alternative way to fix the problem — by using reference arguments. Here's the original function declaration:

```
void EditEnglishDistance(Distance dToEdit); // function declaration
```

You probably changed it to return a **Distance** object. Now change it to the following. You are asking to send arguments (in this case **dLength2**) to the function *by reference* rather than *by value*. This means that **dLength2** will be accessible through the *alias* **rdToEdit**.

```
void EditEnglishDistance(Distance& rdToEdit); // function declaration
```

The function definition must be changed to match.

With this change, you have eliminated the return of **dToEdit** and the assignment to **dLength2**. Thus the function becomes a **void** function again. Now execute code. See if it works.

Execute the code again, but this time with a suitable breakpoint. to discover the addresses of your objects. You should be able to prove that **dLength2** and **dToEdit** are now one and the same object (where **dToEdit** is an *alias* for **dLength2**).

(1 mark) What are the addresses of the following:

dLength2 before the function call:

dToEdit in the function:

dLength2 after the function call:

(3 marks) Explain why the two different approaches give you two different patterns of addressing.

Submission Process

- Submit this by the beginning of the next lab period.
- Some of you may be able to finish this in the current lab period (for a bonus mark).