CSCI 103: Introduction to Programming
Lab 3

January 26, 2024

Lab Overview

- Goals
  - Practice compiling multi-file applications at the terminal (command line)
  - Learn about debuggers and use the Codio debugger to find errors in a sample code
- Process
  - Guided demo of using the debugger (Part 1 in Lab 3 Codio Assignment)
  - Practice compiling a multi-file application and using your own "header files" (Part 2 in Lab 3 Codio Assignment)
  - Practice debugging a program with errors (Part 3) - Must get checked off by a Lab staff to get credit (after giving your best effort)
  - Debugging challenge: Decrypt the secret messages (Part 4)

Compiling programs

$ g++ -g -Wall test.cpp file2.cpp -o test

- `-g -Wall` : are option flags
  - `-g` : provide debugging feature to your program
  - `-Wall` : show all warnings
- `test.cpp file2.cpp` : source files you want to compile
  - **You must list ALL source files (however many there are)**
- `-o test` : compiles and links files into an executable named `test` (-o is also a flag)
  - You can name the executable whatever you want!

Prototypes Reminder

- You have learned that we need function prototypes before you call a function that is defined elsewhere.

```
#include <iostream>
using namespace std;
  // no prototype for sum()
bool done = false;
int main()
{
  int array[3] = {4,5,6};
  // how does compiler know if
  // this is a valid func. call
  int val = sum(array, 3);
  cout << val << endl;
  return 0;
}
```

```
extern bool done;
int sum(int a[], int n)
{
  int s = 0;
  for(int i = 0; i < n; i++) {
    s += a[i];
  }
  done = true;
  return s;
}
```

Header Files (1)

- Rather than re-typing (or copy/pasting) prototypes into any source files that want to use those functions, we can put the prototypes in a separate "header" file (aka .h file) and then #include that header file

```
// prototype only
int sum(int a[], int n);
```

split-sum.h

```
extern bool done;
int sum(int a[], int n) {
  // implementation
}
```

split-sum.cpp

note these are two different files! .h vs .cpp

Including Header Files (2)

- We then #include the header file containing the prototypes into any application wishing to use them
- Remember we still have to supply all the .cpp source files on the g++ command to compile the application (you don't compile header files!)

```
#include <iostream>
#include "split-sum.h"
using namespace std;
bool done = false;
int main()
{
  int array[3] = {4,5,6};
  int val = sum(array, 3);
  cout << val << endl;
  return 0;
}
```

split-main.cpp

```
// prototype only
int sum(int a[], int n);
```

split-sum.h

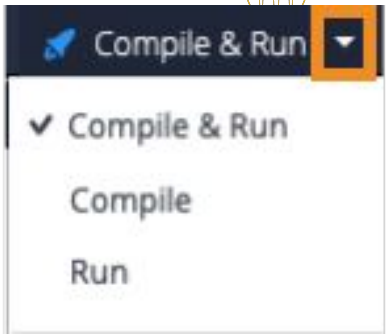
```
extern bool done;
int sum(int a[], int n) {
  // implementation
}
```

split-sum.cpp

```
#include <iostream>
#include "split-sum.h"
using namespace std;
int main() {
  int vals[5];
  // ...
  cout << sum(vals, 3) << endl;
```

other-app.cpp

Running programs



$ ./test

- Remember: test was the name of the executable, but you can name the executable anything when you compile
- This loads and executes the program
- Some assignments in Codio will have the automated compile and run button enabled (looks like a Rocketship)
  - Use the drop-down arrow to the right of the “Compile & Run” option to change the button to just Compile or just Run.
- Most others will not have this enabled: you should know how to compile and run directly via the terminal
  - You will not have this button when you go to classes like CS104!

Techniques for debugging programs without using debugger tool

- **Print Statements**
  - Put print statements (cout) in the loops and conditional statements. This will help you to understand the flow of the program and localise your error.
- **Commenting code**
  - If you are unable to find where exactly the code is breaking, maybe try commenting out chunks of code to see if you can get a smaller, simpler program to work, and then add back in the other code little by little to find where the error occurred.

These are more time consuming and require more effort in localising the error. Thus, we use debuggers.

Debuggers

Debuggers

- **Allows you to:**
  - Set a breakpoint (the code will run and then stop when it reaches the certain line of code)
  - Step through your code line by line so that you can see where the flow of the program goes
  - Print variable values when you have stopped at a certain line of code.

Debuggers

- In codio, Use the “Debug Current File” on the far right of the top menu bar to launch the debugger targeting the file your cursor is in.
- When you run the program , It highlights the exact line where the code breaks.

- Thus, whenever the code does not run as you expect or want, one of your first steps should be : **Run Debugger**

Breakpoints

- Allows you to specify lines of code where you want the flow to stop and analyse the values of the variables/function calls/etc.
- In codio, you can set it up by clicking on the margin of the code and see a red dot appear at the line of the code.

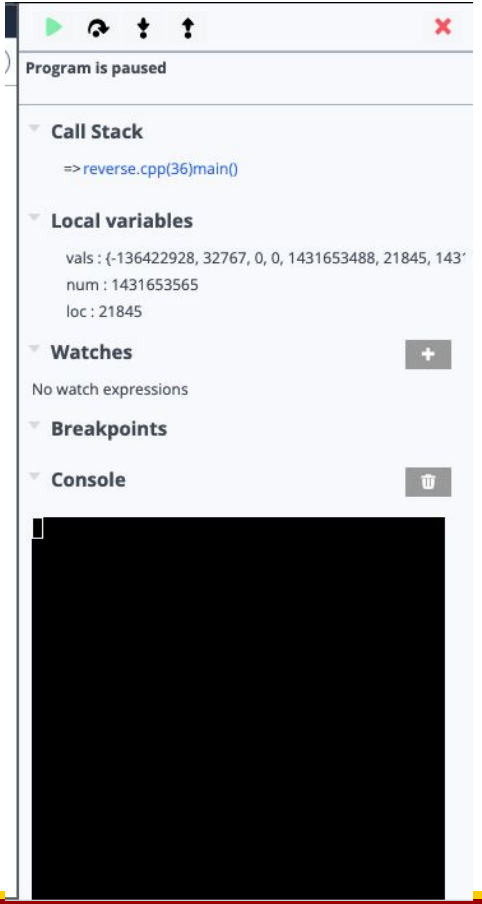
```
35
36  int main() {
37      int vals[10] = {9, 7, 14, 20, 2, 8,
38      cout << "Enter a number: ";
39      int num;
40      cin >> num;
41
```

Call Stack

- Allows you to see the functions being called and what the system stack looks like.
- For example here, the “main()” function calls “reverseAndFind()”
- We can view the value of local variables in each function by clicking that function in the stack area.

Runs the program (until breakpoint). *Note: it will pause temporarily when at a cin statement and wait for you to type the input.*

Step Over : Run the current line and pause at the next line of code.



Finishes the current function's code (or hit the breakpoint), and then pause at the next line of the previous function on the stack

Step into : If current line is function, it steps into the function pausing at the first line of the function body.
If current line is not a function, it executes the line (like step over)

Your Turn

- Carefully read through the Codio guides to complete the tasks they indicate
  - 1 compilation task
  - 2 debugging tasks
  - Type in the requested information at the end of each exercise.
  - **When done with Part 2 and 3 exercises (or with 20 minutes remaining),** raise your hand and **go through your answers with a CP/TA to get checked off.**
    - **Make sure you have 100% on your lab!**
  - Then try Part 4 on your own (Debugging challenge)
- Review the lecture slides regarding compilation and debugging.
- Let CPs & TAs know if you have any questions

Happy Coding!