CS 201 Homework 3

Solomon Himelbloom

September 30, 2020

- Repository Link: `https://github.com/techsolomon/cs201`
- Git Commits: `https://github.com/techsolomon/cs201/commits`
- This homework took approximately 6 hours to complete.

1 Design

In this fourth assignment, I was challenged to organize my three programs with multiple files with an appropriate naming convention and linking capability in mind. This also means that each sub-file used multiple commits, functions were broken down into smaller parts, and various improvements were made after testing to achieve an optimal output.

I again utilized a monorepo to store four directories under the hw3 folder – the main program (boxprint), two additional programs (quadratic and kelvin), and a practice folder (for extra credit, such as stl and collatz). Every .cpp file starts with the author's name, the computer science class code, date file was created or last edited, and finally a short description. Each directory also has a Makefile. Finally, header files were created to allow for linking multiple files and defining each function.

2 Post Mortem

During this homework assignment, I found that the hardest part to be transferring from one main function to breaking my code down into multiple files. For the style and design side of things, I think that my program directories and functions were appropriately named but could include more comments for future reference.

Some changes that I hope to include for the next homework assignment include drawing out where each code block needs to be located with pseudocode and filling in my function documentation accordingly. In conclusion, I also hope to work on time management and better budget out how long each file takes to write for a smoother submission process.

3 Answers to Questions

1. Four major types of errors include linker errors, syntax errors, semantic errors, and run-time errors. They are defined as follows – Linker errors appear when you fail to build out your programs with the appropriate files or attempt to build your project where an executable cannot be generated. Syntax errors most commonly come from writing a while or for loop incorrectly without the correct "instructions" for the compiler. I tend to make semantic errors the most when forgetting when to use one equal sign (=) or two (==) and giving code that is not meaningful to the compiler. Finally, run-time errors appear when running your program, such as attempting division by zero.

2. We can safely ignore cosmetic errors when printing code, basic memory management issues, misplaced header files, and missing semi-colons or curly braces omnipresent in student programs.

3. We should guarantee that every project includes the following before completion: frequent code commits in a version

control system (VCS), programs build and run without errors or warnings, functions/files/variables/etc. are given appropriate names, and comments are included in a header with the file name, author name, date created, and a short description. Finally, comments should also define each new function or ambiguous block of code for future reference.

4. There approaches we can take to eliminate errors in programs and produce acceptable software include building and running often, document new functions and files with code comments, and pseudocode and adding unit tests to probe for errors in our attempt to achieve an optimal output.

5. Computer programmers tend to hate debugging as it takes time away from writing addition programs and most bugs are tricky to find and fix with code that you would have initially expected to run properly.

6. Four common steps in debugging a program include: identifying any bugs or errors written in code, isolating certain functions and files to limit the scope of your debugging session, fixing the code through small edits or larger overhauls, and finally reviewing your code that you correctly resolved the issue to end your debugging session. Some helpful steps for debugging a program include reading any error messages flashing in the terminal and listening to your IDE if it suggests any structural or code edits.

7. Commenting is tremendously helpful when debugging your own code or reviewing errors in another code base as they describe the author's intent of writing certain functions or other programming building blocks. These comments, therefore, become an invaluable source of information as we can track expected output and the intention for blocks of code as it differs from the compiler and debugger-thrown errors.

8. Testing is the process that includes building and running through various edge-cases in your program for quality assurance (QA) purposes, whereas debugging is the repeated

process of root cause analysis, fixing a bug through code revisions, and verifying that the fix resolves the issue and runs as expected.

4 Program 1

4.1 Sample Output/Screenshot

Listing 1: Sample Program Output

```
Input text to be included in the box: Dog_Cat_Moose
Enter an integer (number of layers): 4

BOX INPUT TEXT: Dog_Cat_Moose
# OF LAYERS: 4
STRING LENGTH: 13

************************
************************
************************
************************
****                ****
**** Dog_Cat_Moose ****
****                ****
************************
************************
************************
************************

Press ENTER to quit %
```

4.2 Git Commit Messages

Date	Message
2020-09-30	add: right/left spaces and exit
2020-09-30	refactor: link boxprint with boxer source
2020-09-30	add: 4th example printed test case
2020-09-30	add: if/else, adapt characterBox from l03
2020-09-30	add: user input (str text and int layers)
2020-09-30	add: boxer.cpp and boxer.hpp template
2020-09-23	add: hw3 (initial files)

4.3 Source Code

4.4 boxprint.cpp

```
// boxprint.cpp
// Solomon Himelbloom
// 23 September 2020
// Text in a box example for CS 201.

#include "boxer.hpp"
#include <iostream>
#include <string>
using std::string;
using std::cout;
using std::endl;
using std::cin;

int main(int argc, char * argv[]) {
    std::string box_text = "";
    int box_layers = 0;

    cout << "Input text to be included in the box: ";
    cin >> box_text;

    cout << "Enter an integer (number of layers): ";
    cin >> box_layers;

    if (box_layers <= 0) {
        cout << "You typed in zero, a negative number, or a letter." << endl;
        cout << "Please try again with a positive integer." << endl;
    }

    else {
        boxPrinter(box_text, box_layers);
    }

    cout << "" << endl;
    cout << "Press ENTER to quit ";
    while (cin.get() != '\n') ;

    return 0;
}
```

4.5 boxer.cpp

```
// boxer.cpp
// Solomon Himelbloom
// 30 September 2020
// Text in a box [source file] example for CS 201.

#include "boxer.hpp"
#include <iostream>
#include <stdio.h>
#include <string>
using std::cin;
using std::cout;
using std::endl;

```

```
14 void boxPrinter(std::string name, int boarder) {
15     cout << "" << endl;
16     cout << "BOX INPUT TEXT: " << name << endl;
17     cout << "# OF LAYERS: " << boarder << endl;
18     cout << "STRING LENGTH: " << name.size() << endl;
19     cout << "" << endl;
20 
21     int i = 0;
22 
23     while (i < boarder) {
24         int j = 1;
25         while (j < name.size() + 2 + (boarder * 2)) {
26             cout << "*";
27             j++;
28         }
29 
30         cout << "*";
31         cout << "\n";
32         i++;
33     }
34 
35     int left1 = 0;
36     while (left1 < boarder) {
37         cout << "*";
38         left1++;
39     }
40 
41     int center1 = 0;
42     while (center1 < name.size() + 2) {
43         cout << " ";
44         center1++;
45     }
46 
47     int right1 = 0;
48     while (right1 < boarder) {
49         cout << "*";
50         right1++;
51     }
52 
53     cout << " " << endl;
54 
55     int left_spaces = 0;
56     while (left_spaces < boarder) {
57         cout << "*";
58         left_spaces++;
59     }
60 
61     cout << " ";
62     cout << name;
63     cout << " ";
64 
65     int right_spaces = 0;
66     while (right_spaces < boarder) {
67         cout << "*";
68         right_spaces++;
69     }
70 
71     cout << " " << endl;
72 
73     int left2 = 0;
74     while (left2 < boarder) {
75         cout << "*";
76         left2++;
77     }
```

```
78
79      int center2 = 0;
80      while (center2 < name.size() + 2) {
81          cout << " ";
82          center2++;
83      }
84
85      int right2 = 0;
86      while (right2 < boarder) {
87          cout << "*";
88          right2++;
89      }
90
91      cout << " " << endl;
92
93      int k = 0;
94
95      while (k < boarder) {
96          int l = 1;
97          while (l < name.size() + 2 + (boarder * 2)) {
98              cout << "*";
99              l++;
100         }
101
102         cout << "*";
103         cout << "\n";
104         k++;
105     }
106 }
```

4.6 boxer.hpp

```
1  // boxer.hpp
2  // Solomon Himelbloom
3  // 30 September 2020
4  // Text in a box [header file] example for CS 201.
5
6  #include <iostream>
7  #include <string>
8
9  #ifndef FILE_BOXER_HPP_INCLUDED
10 #define FILE_BOXER_HPP_INCLUDED
11
12 void boxPrinter(std::string name, int boarder);
13
14 #endif
```

5 Program 2

5.1 Sample Output/Screenshot

Listing 2: Sample Program Output

```
0
-273.15
```

5.2 Git Commit Messages

Date	Message
2020-09-30	refactor: fix kelvin conversion; add: changelog comments
2020-09-30	add: broken kelvin original example
2020-09-23	add: hw3 (initial files)

5.3 Source Code

```
// kelvin.cpp
// Solomon Himelbloom
// 23 September 2020
// Broken Kelvin example for CS 201.

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

// // ORIGINAL: Converts Celsius to Kelvin
// double ctok(double c) {
//     int k = c + 273.15;
//     return int;
// }

// int main() {
//     double c = 0;
//     cin >> d;
//     double k = ctok("c");
//     cout << k << '/n';
// }

// FINAL: Converts Celsius to Kelvin

// changed from double to void
double ctok(double c) {
    // changed operation from addition to subtraction
    int k = c;
    c -= 273.15;
    // needs to return ctok instead of int
    return c;
}

int main() {
    double c = 0;
    // d is not defined; changed to the double c
    cin >> c;
    // new mention of k: needs to be initialized
    double k;
    // c is passed into the ctok function w/o quotes
    k = ctok(c);
    // escape character was facing the wrong direction
    cout << k << '\n';
}
```

6 Program 3

6.1 Sample Output/Screenshot

Listing 3: Sample Program Output

```
a = 1
```

```
b = 4
c = 3
x1 = -1
x2 = -3

a = -5
b = 1
c = -8
Fail: The resulting discriminant (b^2-4ac) is negative
   .
The quadratic equation has no real roots.

a = 0
b = 0
c = 1
Error: The resulting discriminant (b^2-4ac) is zero.
Please try again.
```

6.2 Git Commit Messages

Date	Message
2020-09-30	add: support for non-real roots
2020-09-30	refactor: sample output edge case; add: x1, x2
2020-09-30	add: discriminant, testing; include: math
2020-09-30	add: quadraticFormula function notation
2020-09-23	add: hw3 (initial files)

6.3 Source Code

```cpp
// quadratic.cpp
// Solomon Himelbloom
// 23 September 2020
// Quadratic formula for CS 201.

#include <iostream>
#include <math.h>
using std::cout;
using std::endl;
using std::cin;

// QUADRATIC FORMULA: x = -b +/- sqrt(b^2-4ac)/(2a)
void quadraticFormula(int a, int b, int c) {
    float discriminant, x1, x2;
    discriminant = b*b - 4*a*c;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;

    if (discriminant == 0) {
        cout << "Error: The resulting discriminant (b^2-4ac) is zero." << endl;
        cout << "Please try again." << endl;
        cout << " " << endl;
    }

    else if (discriminant < 0) {
        cout << "Fail: The resulting discriminant (b^2-4ac) is negative." << endl;
        cout << "The quadratic equation has no real roots." << endl;
        cout << " " << endl;
    }

    else {
        x1 = (-b + sqrt(discriminant)) / (2*a);
        x2 = (-b - sqrt(discriminant)) / (2*a);

        cout << "x1 = " << x1 << endl;
        cout << "x2 = " << x2 << endl;
        cout << " " << endl;
    }

}

int main() {
    // Pass: Discriminant is positive.
    quadraticFormula(1, 4, 3);

    // Fail: Discriminant is negative.
    quadraticFormula(-5, 1, -8);

    // Check: Discriminant is zero.
    quadraticFormula(0, 0, 1);
}
```