CS 201 Homework 04

Bryan Beus

November 4, 2019

Source Code Link: `https://github.com/siddhartha-crypto/cs201/tree/master/hw5`

1 Design

1.1 Truncstruct

The questions posed in the original homework assignment are relatively straightforward. I intend to create a simple GUI that displays the functionality of the truncstruct lab.

1.2 Bulls and Cows

This is also a straightforward application. The bulls and cows program is already working, and now needs a GUI.

2 Post Mortem

2.1 Truncstruct

Becoming familiar with the new documentation style of the FLTK website was difficult. Because I have so many habits and expectations for documentation in the job that I currently do, I found the lack of certain features and the addition of others made the documentation difficult to navigate and understand.

To solve this problem, I searched through a few other students' Github repositories to get an initial idea of how the FLTK GUI functions functioned. Once I had a general understanding, I was then able to apply those experiences to my own programs.

2.2 Bulls and Cows

With the pattern established in the Truncstruct program, the application of FLTK functionality to Bulls and Cows was easy. I moved the logic in `main()` into a separate `obj->callback()` method to perform all calculations on user input and the correct pattern.

Two challenges that I faced were to get the correct pattern, which is created and stored as a string, into the FLTK callback, and a confusion on the way `int` values are increased using `++` operators.

With the former challenge, the solution I found was to set the value to a non-displayed `Fl_Output` object, and to call that object from within the `obj->callback` method.

With the latter challenge, once I realized that `null` integer variables do not increment properly on the `++` operator, I set the default values to `0` and the problem resolved.

3 Answers to Questions

  - Sequence Data: An arbitrarily sized collection of items, usually of the same type, and can have a designed order
  - Associative Data: An arbitrarily sized collection of items, usually of the same type, where the lookup method uses a key. Typically, the key leads to an associated value.
  - Record Data: A collection of data of a fixed size, and where each item has a fixed type. The types may be different, and each item is a field of the record.

- A tuple holds the same types of data as a pair, including `char`, `int`, etc., but a tuple can hold more than two values.

- To access an element in a tuple, we use the `get<INDEX>` method.
- A tuple can hold many different types of data. For example, a tuple could be of type: `tuple<bool, int, char, int>`. A vector typically must be of one type, such as: `vector<int>`. A vector, on the other hand, can handle iteration more easily, using the `::iterator` type. Tuples do not handle iteration nearly as easily.
- A first-class type can be used without restriction. (Whereas a second-class type cannot be assigned to a variable or returned as a function, and a third-class type cannot be passed as function parameters.)
- To create integer constants using enum, we might try something similar to the following: `enum Flavor { spicy = 10, medium = 5, spicy = 0 }`, and then we can call `Flavor` as a kind of Class object. For example, `Flavor a = spicy; cout << a` would return `10` to the console.
- `struct` holds Record type data.
- To access a member of a `struct` object, we use the `.` period operator.
  - Member: Declared inside a `class` or `struct`
  - Local: Declared inside a function
  - Global: Declared in neither of the above
- A key-value pair is a type of associative data where the key is stored in one location, and it is paired with a value that is stored in another location. These types of data are useful for holding data values that are not appropriate for direct sorting methods, among other use cases
- A library is a collection of code that is generalized in such a way as to be useful across a wide array of other programs and functions.

4 Sample Output

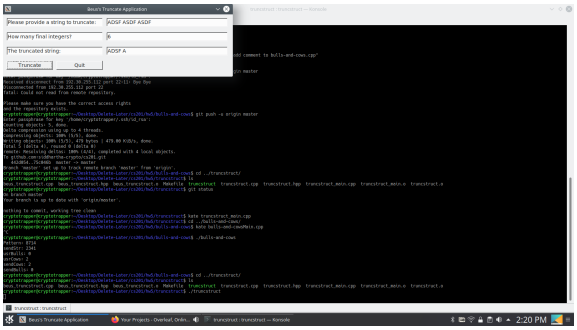

Figure 1: Truncstruct Output

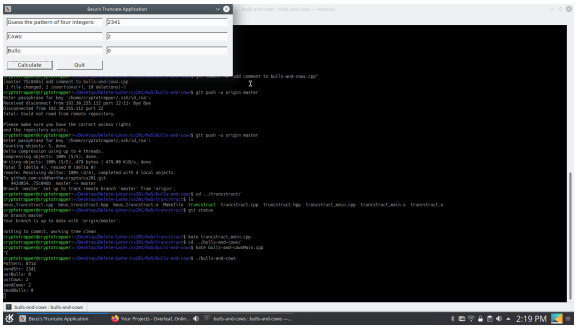

Figure 2: Bulls and Cows Output

5 My Programs

5.1 Truncstruct

```
1 /*
2 * trunstruct\_main.cpp
3 * Bryan Beus
4 * CS 201
5 * October 23, 2019
6 * Main file for trunstruct lab22
7 */
```

```
8
9  #include <iostream>
10 #include <string>
11 #include <vector>
12 #include <sstream>
13
14 #include "truncstruct.hpp"
15 #include "beus_truncstruct.hpp"
16
17 #include <FL/Fl.H>
18 #include <FL/Fl_Text_Display.H>
19 #include <FL/Fl_Window.H>
20 #include <FL/Fl_Output.H>
21 #include <FL/Fl_Input.H>
22 #include <FL/Fl_Widget.H>
23 #include <FL/Fl_Button.H>
24
25 using std::string;
26 using std::istringstream;
27
28 int main(int argc, char **argv) {
29
30     // Declare Fl_Output pointer objects for displaying
       ↪  instructions and information
31     Fl_Output *stringInputDisplay = nullptr;
32     Fl_Output *countPrompt = nullptr;
33     Fl_Output *resultDisplay = nullptr;
34
35     // Delcare Fl objects for representing display
36     Fl_Input *usrStr = nullptr;
37     Fl_Input *usrCount = nullptr;
38     Fl_Output *res = nullptr;
39     Fl_Button *truncateUsrStr = nullptr;
40     Fl_Button *quit = nullptr;
41
42     // Declare FLTK Window
43   Fl_Window *window = new Fl_Window(640,170, "Beus's Truncate
      ↪  Application");
44     window->begin();
45
46     // Declare initial window outputs and inputs for child 0 and 1
47     stringInputDisplay = new Fl_Output(10, 10, 270, 25, 0);
48     stringInputDisplay->value("Please provide a string to
       ↪  truncate:");
49     usrStr = new Fl_Input(290, 10, 340, 25);
50
51     // Declare initial values for output and input of child 2 and
       ↪  3
52     countPrompt = new Fl_Output(10, 50, 270, 25, 0);
53     countPrompt->value("How many final integers?");
54     usrCount = new Fl_Input(290, 50, 340, 25);
55
56     // Delcare initial values of childs 4 and 5
```

```
    resultDisplay = new Fl_Output(10, 90, 270, 25, 0);
    resultDisplay->value("The truncated string:");
    res = new Fl_Output(290, 90, 340, 25, 0);

    // Declare buttons for user actions
    truncateUsrStr = new Fl_Button(10, 130, 130, 25, "Truncate");
    truncateUsrStr->callback(truncArbitrary);

    quit = new Fl_Button(150, 130, 130, 25, "Quit");
    quit->callback(quitProgram);

    // End window
    window->end();
    window->show(argc, argv);

    // Launch window
    return Fl::run();

}
```

5.2 Bulls and Cows

```
/**
 * bulls-and-cowsMain.cpp
 * CS 201
 * Bryan Beus
 * October 15, 2019
 * The main file for bulls-and-cows
 */

#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <sstream>
#include <map>
#include <fstream>
#include <stdlib.h>

#include "bulls-and-cows.hpp"

#include <FL/Fl.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Output.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Button.H>

using std::cout;
```

```
using std::cerr;
using std::cin;
using std::endl;
using std::vector;
using std::string;
using std::noskipws;
using std::getline;
using std::find;
using std::istringstream;
using std::stringstream;
using std::ifstream;
using std::rand;

int main(int argc, char **argv) {

    // Initiate a string to hold the correct pattern
    string pattern = "";

    // Call the setPattern function to set the pattern
    setPattern(pattern);

    // Declare Fl_Output pointer objects for displaying instructions and information
    Fl_Output *currentGuessDisplay = nullptr;
    Fl_Output *resultCowsDisplay = nullptr;
    Fl_Output *resultBullsDisplay = nullptr;

    // Delcare Fl objects for representing display
    Fl_Input *usrGuess = nullptr;
    Fl_Output *resCows = nullptr;
    Fl_Output *resBulls = nullptr;
    Fl_Output *correctPattern = nullptr;


    // Declare buttons
    Fl_Button *calculateRes = nullptr;
    Fl_Button *quit = nullptr;

    // Declare FLTK Window
  Fl_Window *window = new Fl_Window(640,170, "Beus's Truncate Application");
    window->begin();

    // Declare text and output for user guesses, childs 0 and 1
    currentGuessDisplay = new Fl_Output(10, 10, 270, 25, 0);
    currentGuessDisplay->value("Guess the pattern of four integers:");
    usrGuess = new Fl_Input(290, 10, 340, 25);

    // Declare initial values of childs 2 and 3
    resultCowsDisplay = new Fl_Output(10, 50, 270, 25, 0);
    resultCowsDisplay->value("Cows:");
    resCows = new Fl_Output(290, 50, 340, 25, 0);
```

```
79
80      // Declare initial values of childs 4 and 5
81      resultBullsDisplay = new Fl_Output(10, 90, 270, 25, 0);
82      resultBullsDisplay->value("Bulls:");
83      resBulls = new Fl_Output(290, 90, 340, 25, 0);
84
85      // Declare correct pattern as child 6
86      correctPattern = new Fl_Output(10,10,0,0,0);
87      correctPattern->value(pattern.c_str());
88
89      // Declare buttons for user actions
90      calculateRes = new Fl_Button(10, 130, 130, 25, "Calculate");
91      calculateRes->callback(calculateBullsAndCows);
92
93      quit = new Fl_Button(150, 130, 130, 25, "Quit");
94      quit->callback(quitProgram);
95
96      // End window
97      window->end();
98      window->show(argc, argv);
99
100     // Launch window
101     return Fl::run();
102 }
```