CS 201 Homework 01

Bryan Beus

September 9, 2019

Source Code Link: `https://github.com/siddhartha-crypto/cs201/tree/master/hw1`

1 Design

1.1 Diamond

For the `Diamond` program, I divide the diamond into two main parts, and two subparts. The main parts are the top and bottom. The subparts are the left and right sides. For each row of the diamond, I assemble a string of the correct characters first, and then print it to the terminal.

1.2 Dotcross

For the `Dotcross` program, I request the user to input each of the variables for each vector, one at a time. I store all of these in one vector. Then, I perform the calculations and print them to the terminal output.

1.3 Greatest

For the `Greatest` program, I request the user to input integer variables, one at a time. I check each input to make sure it is valid. The valid responses are stored in a vector. Once the user makes a

response that ends the period of inputting, I calculate by iterating through the vector and testing to see if any value is bigger than the current largest value, with the initial value set to the first in the vector array position. I return the final result to the terminal.

1.4 Grid

I initiate an endless while loop that first clears the screen, then draw the current state of the grid (with default values of "."), and then request the user to input the column and row to change. Assuming the user inputs a proper column and row, I update a vector that holds a position for each box variable (default "."), replacing it with the intended "X". If a user enters a negative integer, I end the program.

1.5 Mileskm

I request the user to input the number of miles they would like to convert, and then I return the result times the conversion rate of kilometers/mile.

2 Post Mortem

2.1 Diamond

I have actually done a similar exercise before, and last time I did this type of problem I made things more complicated. Last time, I had a separate function for both left and right side of each line. This time, I was able to combine the rows of `"#"` hashtags into one function per line, and one while statement per top/bottom.

However, as I thought afterwards I'm certain I could simplify this down even further. I could just do a simple add/subtract formula for the left/right sides of each line, without getting into a full function for each.

I am out of time for this week, so this may have to wait for a future experiment.

2.2 Dotcross

The challenge here was to get the `setprecision()` function to work properly. I had to add in a `fixed` function from the standard namespace to get the setprecision to focus on decimal places, as a opposed to total number of digits in the response.

2.3 Greatest

The while loop solution that I use here is fine, so long as I enter whole integers or characters, but if I enter a float value, the while loop executes itself an extra time. I spent a long time trying to solve this problem, and was not able to solve it, nor find help online. I assume the problem is that I do not fully understand how the C++ compiler works. I will ask the TA for assistance.

2.4 Grid

This one seemed simple enough when I began, but became more complicated as time progressed. I did not want to deal with having unique variables for each box, and I did not want to have to manually type out the rows/columns and their structure for each slot.

So, I grouped things down into functions that could be called at each step of drawing the process. This was time consuming to visualize, but was perhaps a cleaner solution in the end, and likely saved time.

The big challenge in terms of coding was the vector of vectors that I used to hold the current state of the grid boxes. I had to use Stackoverflow a lot to discover how to properly place and call these types of vectors.

2.5 Mileskm

This one was relatively straightforward, and I finished it in just a few minutes.

3 Answers to Questions

- A compiler serves to convert code in one language into code in another. Typically, this is from a higher to a lower level language. The stages from source code to executable are build, compile, and link.
- Definitions
  - Header: These files contain C++ source code, but are not intended to be compiled directly. Instead, they are included using an include statement
  - compile-time error: An error that occurs at the time that we attempt to compile the source code
  - linker: The linker generates a single executable from multiple source code and library files
  - statement: An expression of logic using code
- A source code is human readable. An object file is an intermediate file between source and executable files, and is generally less readable.
- We need to internalize the theory. Theory without internalization makes you a critic, and everybody's a critic.
- Generally, a prompt is a response from the computer stating that it is ready for the next user input.
- This is the newline character, or "escape n." This gives us a new line, under normal circumstances (utf 8 etc.).
- Definitions
  - Variable: A variable is a value that can change, depending upon the way the source code utilizes it.
  - Object: A combination of variables, data structures, and functions; an object can also be described as an instance of a class. Objects in object-oriented programming are a high priority in the designer's mindset.

  - literal: A fixed value – something that does not change within the source code. A literal has an opposite nature when compared to a variable.

- Types of literals: In C++, we have strings, floats, doubles, longs, integers, chars, and many more.
- Five names to avoid, so as to avoid confusion
  - test - While this word is legal, it should not be used, because bash scripts and the terminal use this for various functions.
  - `_var` - Underscores at the start of words are used in libraries, and therefore, although legal, should be avoided
  - isNotValid - Avoid negation.
  - T - One letter variables should be avoided, since they can create code that is difficult to read.
- Double may contain a decimal place, whereas int is only full integers. Information can be lost.

4 Sample Output

Listing 1: "Diamond"

```
siddhartha@zuko:diamond$ ./diamond
Enter the number of lines to print in the diamond: 8
       #
      ###
     #####
    #######
   #########
  ###########
 #############
###############
 #############
  ###########
```

```
   #########
    #######
     #####
      ###
       #
```

Listing 2: Dotcross

```
siddhartha@zuko:dotcross$ ./dotcross
Provide three floating point numbers for vector A and
   three more for vector B:
Ax: 1.01
Ay: 2.01
Az: 0.5523
Bx: 2.01
By: -1.05
Bz: 1.001
A dot B = 0.47245
A cross B = (2.59193, 0.09911, -5.10060)
```

Listing 3: Greatest

```
siddhartha@zuko:greatest$ ./greatest
Provide a sequence of positive integers, ending with
   zero or a negative number.
Enter a positive integer (0 or negative to end): 1
Enter a positive integer (0 or negative to end): 5
Enter a positive integer (0 or negative to end): 253
Enter a positive integer (0 or negative to end): 022
Enter a positive integer (0 or negative to end): 0

The greatest number entered: 253
```

Listing 4: Grid

```
Note: LaTeX struggles with the Unicode characters used
    in the program to draw the grid walls. Please run
   this in your terminal to see the grid structure.
```

```
     1     2     3     4     5

 1   .     .     .     .     .

 2   .     .     X     .     .

 3   .     .     .     .     X

 4   .     .     .     .     .

 5   .     .     .     .     .


Enter the number of the column you would like to fill:
    1
Enter the number of the row you would like to fill: -1
```

Listing 5: Mileskm

```
siddhartha@zuko:mileskm$ ./mileskm
Enter the number of miles to convert to kilometers:
   2.51
The number of kilometers in 2.51 miles is: 4.03859
```

5 My Programs

5.1 Diamond

```
/**
 * diamond.cpp
 * Bryan Beus
 * 7 Sept 2019
 * Diamond assignment for CS 201
 */

#include <iostream>
#include <string>

using namespace std;

int main() {

    // Declare the variable to hold the number of lines

    int num_lines;

    // Request info about the number of lines

    cout << "Enter the number of lines to print in the diamond: ";

    // Capture user input

    cin >> num_lines;

    // Check to make sure that the input is within our valid
    ↪  parameters

    if (num_lines == 0 || num_lines < 0 || cin.fail()) {

        cout << "The value you entered is not a valid number" <<
        ↪  endl;

        return 0;
    }

    // Set the doubled value of num_lines

    int two_times = num_lines * 2;

    // Set our current vertical position on the diamond grid

    int pos_ver = 0;

    // Declare an empty string to use for additive iteration

    string current_line;

```

```
48      // Begin while loop to print first half of diamond
49
50      while (pos_ver < num_lines) {
51
52          // Set or reset current_line variable value to empty
53
54          current_line = "";
55
56          // Use iterative adding to the current_line string to
            ↪  build the current line's # output
57
58          for (int i = 1; i < two_times; i++) {
59
60              // Set the equations to decide whether each character
                ↪  in the string should be a " " or a "#"
61
62              if (
63                     (num_lines - pos_ver <= i && i <= num_lines) ||
64                      (num_lines <= i && i <= num_lines + pos_ver)
65                 )
66              {
67
68                  // Add a "#" if the equation produce a true result
69
70                  current_line = current_line + "#";
71              } else {
72
73                  // Add a space, if not
74
75                  current_line = current_line + " ";
76              }
77
78          }
79
80          // Print the resulting current_line variable to the
            ↪  terminal
81
82          cout << current_line << endl;
83
84          // Increase our vertical position in preparation to move
            ↪  to the next line
85
86          pos_ver += 1;
87      }
88
89      // Because we're creating a diamond, we want to skip the
        ↪  first row, so that we do not get a fully mirrored image
        ↪  in the vertical direction
90
91      pos_ver += 1;
92
93      // Begin while loop for bottom half of diamond
94
95      while (pos_ver < two_times) {
96
```

```
97          // Reset current_line value to empty
98
99          current_line = "";
100
101         // Use iterative adding to create the string to print to
            ↪  the console
102
103         for (int i = 1; i < two_times; i++) {
104
105            // Equations to determine whether to add a "#" or a " "
106
107             if (
108                     (pos_ver < i + num_lines && i <= num_lines) ||
109                      (num_lines <= i && i < two_times - pos_ver +
                         ↪  num_lines)
110                )
111             {
112
113                 // Add a "#"
114
115                 current_line = current_line + "#";
116             } else {
117
118                 // Or add a " "
119
120                 current_line = current_line + " ";
121             }
122         }
123
124         // Print the resulting string to the terminal
125
126         cout << current_line << endl;
127
128         // Increase our vertical position
129         pos_ver += 1;
130
131     }
132
133     // End
134
135     return 0;
136 }
```

5.2 Dotcross

```
1  /**
2   * dotcross.cpp
3   * Bryan Beus
4   * 8 Sept 2019
5   * Dotcross assignment for CS 201
6   */
```

```cpp

#include <iostream>
#include <string>
#include <vector>
#include <iomanip>

using namespace std;

int main() {

    // Declare a vector to hold user inputs

    vector<float> dotcross;

    // Declare a float variable to hold user inputs

    float input;

    // Declare a vector to hold the variables names

    vector<string> xyz;
    xyz.push_back("Ax");
    xyz.push_back("Ay");
    xyz.push_back("Az");
    xyz.push_back("Bx");
    xyz.push_back("By");
    xyz.push_back("Bz");

    // Request user input

    cout << "Provide three floating point numbers for vector A
    ↪  and three more for vector B: " << endl;

    // Initiate a while loop that continues until the user has
    ↪  provided all six inputs for vectors A and B

    while (dotcross.size() < 6) {

        // Request user input for the current vector variable

        cout << xyz.at(dotcross.size()) << ": ";
        cin >> input;

        // Test whether user input is valid, and if not, repeat
        ↪  the while loop
        // Otherwise, add the user input to the vector variables

        if (cin.fail()) {
            cin.clear();
            cin.ignore(1000,'\n');
            cout << "The input you provided is not valid. Please
            ↪  try again." << endl;
        } else {
            dotcross.push_back(input);
```

```
        }
    }

    // Declare a dot product variable and perform the dot-product calculation

    float dotprod = dotcross.at(0) * dotcross.at(3) + dotcross.at(1) * dotcross.at(4) + dotcross.at(2) * dotcross.at(5);

    // Delcare a vector of the cross-product vector and perform each calculation
    vector<float> crossprod;
    crossprod.push_back(dotcross.at(1) * dotcross.at(5) - dotcross.at(2) * dotcross.at(4));
    crossprod.push_back(dotcross.at(2) * dotcross.at(3) - dotcross.at(0) * dotcross.at(5));
    crossprod.push_back(dotcross.at(0) * dotcross.at(4) - dotcross.at(1) * dotcross.at(3));

    // Report the results

    cout << fixed << setprecision(5) <<  "A dot B = " << dotprod << endl;
    cout << "A cross B = (" << crossprod.at(0) << ", " << crossprod.at(1) << ", " << crossprod.at(2) << ")" << endl;

    return 0;
}
```

5.3 Greatest

```
/**
 * greatest.cpp
 * Bryan Beus
 * 7 Sept 2019
 * Greatest assignment for CS 201
 */

#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Declare a vector to hold all user inputs

vector<int> collection;

// Declare a variable to hold each user input
```

```
19
20 int input;
21
22 // Declare a bool vector to indicate whether or not the user
   ↪  inputs allow for a while loop to end
23
24 bool cont = true;
25
26 // Declare the final variable that holds the greatest integer
27
28 int final_val;
29
30 // A function to request user input
31
32 void request_int() {
33     cout << "Enter a positive integer (0 or negative to end): ";
34 }
35
36 // A function to calculate the final result
37
38 int calculate_final() {
39
40     // Declare the current largest integer, and set it equal to
       ↪  the first user input in the vector
41
42     int current = collection.at(0);
43
44     // Iterate through the user inputs
45     // If any user input is larger than the current largest user
       ↪  input, set the new largest integer as the current largest
46
47     for (int i = 0; i < collection.size(); i++) {
48         if (collection.at(i) > current) {
49             current = collection.at(i);
50         }
51     }
52
53     // Return the final largest integer
54
55     return current;
56 }
57
58 int main() {
59
60     // Request user input
61
62     cout << "Provide a sequence of positive integers, ending with
       ↪  zero or a negative number." << endl;
63
64     // Initiate a while loop that continues until the user has
       ↪  finished inputting integers
65
66     while (cont) {
67
```

```
68          request_int();
69          cin >> input;
70
71          // Test that the user input is valid
72
73          if (cin.fail()) {
74              cont = false;
75              cin.clear();
76              cin.ignore(1000, '\n');
77              cout << endl;
78              cout << "The value you entered is not an integer.
                ↪  Please restart the program." << endl;
79              return 0;
80
81              // If the user input is less than or equal to zero,
                ↪  assume that the user is finished and calculate
                ↪  the final result
82
83
84              // If the input is not valid, end the program
85
86          } else if (input > 0 && !cin.fail()) {
87              collection.push_back(input);
88          } else if (input <= 0 && collection.size() > 0) {
89              cont = false;
90              final_val = calculate_final();
91          } else if (input <= 0 && collection.size() == 0) {
92
93              // If the user input is less than or equal to zero,
                ↪  but there are not integers in the vector, request
                ↪  the user to continue inputting integers
94
95                  cout << "Please enter an integer greater than 0
                    ↪  before ending the program." << endl;
96
97                  // Catchall error
98
99          } else {
100             cout << "We encountered an unexpected error. Please
                ↪  review the source code." << endl;
101             cont = false;
102             return 0;
103         }
104     }
105
106     // Add an extra line for formatting
107
108     cout << endl;
109
110     // Return the final result
111
112     cout << "The greatest number entered: " << final_val << endl;
113
114     return 0;
```

```
}
```

5.4 Grid

```
/**
 * grid.cpp
 * Bryan Beus
 * 8 Sept 2019
 * Grid assignment for CS 201
 */

#include <iostream>
#include <vector>
#include <string>

using namespace std;

// Define the number of Rows (R) and Columns (C)

#define R 5
#define C 5

// Define the Width of the blank spaces in the cells
// This must be an odd number

#define Width 5

// Set a default function to print a series of blank spaces of length <Width>

void print_full_width() {
    for (int j = 0; j < Width; j++) {
        cout << " ";
    }
}

// Set a default function to print a series of blank spaces of half of length <Width>

void print_half_width() {
    for (int j = 0; j < (Width - 1) / 2; j++) {
        cout << " ";
    }
}

// Set a default function to print a series of double bars of length <Width>

void print_full_bar() {
    for (int i = 0; i < Width; i++) {
```

```
        cout << "";
    }
}

// Print the top of the grid

void print_top_line() {

    // Vertically clear at least one line in the terminal, then
    ↪  print the <Width> blank spaces

    cout << endl;

    print_full_width();

    // Print each column number, followed by <Width> blank spaces

    for (int i = 1; i <= C; i++) {

        cout << i;

        print_full_width();
    }

    cout << endl;

    // Print the top row of the grid

    cout << "   ";

    for (int i = 0; i < C - 1; i++) {

        print_full_bar();

        cout << "";

    }

    print_full_bar();

    cout << "" << endl;
}

// A function to fill a whole row that has no variables or grid
↪  corners

void print_fill_row() {

    cout << "   ";

    for (int i = 0; i < C; i++) {

        print_full_width();

        cout << "";
    }
```

```
98
99      cout << endl;
100 }
101
102 // A function to fill a row that has variables, including row
    ↪  numbers and variables inside the grid boxes
103 // Row requires both the current row to print and a vector that
    ↪  has the current state of grid boxes (X's or .'s)
104
105 void print_var_row(int row, vector< vector<string> > collection)
    ↪  {
106
107     // Begin the row
108
109     cout << " " << row << "";
110
111     // Iterate through each grid box, printing the variable that
        ↪  is in the vector
112
113     for (int i = 0; i < C; i++) {
114
115         print_half_width();
116
117         cout << collection[row - 1][i];
118
119         print_half_width();
120
121         cout << "";
122     }
123
124     cout << endl;
125 }
126
127 // A function to print a row that divides the grid boxes
128
129 void print_bar_row() {
130     cout << "  ";
131
132     for (int i = 0; i < C - 1; i++) {
133
134         print_full_bar();
135
136         cout << "";
137     }
138
139     print_full_bar();
140     cout << "";
141
142     cout << endl;
143 }
144
145 // A function to print the bottom line of the grid
146
147 void print_bottom_line() {
148
```

```
149     cout << "   ";
150 
151     for (int i = 0; i < C - 1; i++) {
152 
153         print_full_bar();
154 
155         cout << "";
156     }
157 
158     print_full_bar();
159 
160     cout << "" << endl;
161 }
162 
163 int main() {
164 
165     // Declare the col and row integer variables
166     // These are used for each user input
167 
168     int col;
169     int row;
170 
171     // Declare a vector of vectors of strings
172     // This holds the current state of the grid box values
173 
174     vector< vector<string> > collection;
175 
176     // Expand the number of string-vectors within the main vector
        ↪  to account for the number of rows on the grid
177 
178     collection.resize(R);
179 
180     // Set a default column value of "." for each of the <C>
        ↪  number of grid boxes
181 
182     for (int i = 0; i < R; i++) {
183         for (int j = 0; j < C; j++) {
184             collection[i].push_back(".");
185         }
186     }
187 
188     // Initiate an endless while loop to draw and redraw the
        ↪  current grid to the screen
189     // and to listen for user input/updates
190 
191     while (true) {
192 
193         // Clear the terminal
194 
195         cout << "\033[2J\033[1;1H";
196 
197         // Draw the current state of the grid
198 
199         print_top_line();
200 
```

```
201         for (int i = 0; i < R; i++) {
202             print_fill_row();
203             print_var_row(i + 1, collection);
204             print_fill_row();
205             if (i != R - 1) {
206                 print_bar_row();
207             }
208         }
209 
210         print_bottom_line();
211         cout << endl;
212 
213         // Ensure the col and row variables are reset to 0 for
            ↪  this iteration of the endless while loop
214 
215         col = 0;
216         row = 0;
217 
218         // Declare variables to query whether or not the user has
            ↪  entered a valid row/column number
219 
220         bool col_viable = false;
221         bool row_viable = false;
222 
223         // Initiate a while loop to listen for user input until a
            ↪  valid column number is entered
224 
225         while (!col_viable) {
226 
227             // Request user input
228 
229             cout << "Enter the number of the column you would like
                ↪  to fill: ";
230             cin >> col;
231 
232             // Test whether user input is valid, and if not repeat
                ↪  the request for user input
233             // Otherwise, update the col_viable variable to allow
                ↪  while loop to end
234 
235             if (cin.fail() || col == 0 || C < col) {
236                 cin.clear();
237                 cin.ignore(1000, '\n');
238                 cout << "The value you provided is not valid.
                    ↪  Please try again." << endl;
239             } else if (col < 0) {
240                 return 0;
241             } else {
242                 col_viable = true;
243             }
244         }
245 
```

```
246         // Initiate a while loop to listen for user input until a
            ↪  valid row number is entered
247
248         while (!row_viable) {
249
250             // Request user input
251
252             cout << "Enter the number of the row you would like to
                ↪  fill: ";
253             cin >> row;
254
255             // Test whether user input is valid, and if not repeat
                ↪  the request for user input
256             // Otherwise, update the row_viable variable to allow
                ↪  while loop to end
257
258             if (cin.fail() || row == 0 || R < row) {
259                 cin.clear();
260                 cin.ignore(1000, '\n');
261                 cout << "The value you provided is not valid.
                    ↪  Please try again." << endl;
262             } else if (row < 0) {
263                 return 0;
264             } else {
265                 row_viable = true;
266             }
267         }
268
269         // At the provided user row and column, update the
            ↪  vector-vector value to 'X'
270
271         collection[row - 1][col - 1] = 'X';
272
273     }
274
275     return 0;
276 }
```

5.5 Mileskm

```
1  /**
2   * mileskm.cpp
3   * Bryan Beus
4   * 7 Sept 2019
5   * Mileskm assignment for CS 201
6   */
7
8  #include <iostream>
9  #include <string>
10
```

```cpp
using namespace std;

int main() {

    // Set a default conversion value for kilometers

    float km_per_m = 1.609;

    // Declare a variable for user input

    float input;

    // Request user input

    cout << "Enter the number of miles to convert to kilometers: ";

    cin >> input;

    // Test that the user input is valid, and if not, end the program

    if (cin.fail()) {
        cout << "The value you entered is not a valid number. Please enter a numerical value." << endl;
        return 0;
    }

    // Return the final result and calculations

    cout << "The number of kilometers in " << input << " miles is: " << input * km_per_m << endl;

    return 0;
}
```