CS 202 Iditarod - Anchorage

Bryan Beus

April 7, 2020

Source Code Link: `https://github.com/siddhartha-crypto/cs202/tree/master/iditarod/anchorage`

1 Design

1.1 Fibonacci

The design provided in the homework assignment is sufficient. Each function is already provided. I intend to keep this simple by keeping everything in one `main.cpp` file.

1.2 Factorial

This is similar to the Fibonacci assignment, and should require nothing further than adaptation of the existing code.

2 Post Mortem

2.1 Fibonacci

This was a simple exercise in basic logic and execution. Writing this code did not take long, and for a few small troubleshooting moments, the assignment passed fairly quickly.

2.2 Factorial

As with the Fibonacci section, this section did not take long and required little more than trial and error with some adapted code.

3 Commit History

3.1 Fibonnaci

2020-04-07 Initiate Challenge 1
2020-04-07 Write a recursive fibonacci function that returns the nth number
2020-04-07 Write a non-recursive function that returns the nth number of the Fib sequence.

3.2 Factorial

2020-04-07 Write a function that returns the factorial of nth integer using recursion
2020-04-07 Write a function that returns the factorial of the nth integer without recursion

4 Sample Output

4.1 Fibonnaci

```
0th number: 0
1th number: 1
2th number: 1
3th number: 2
4th number: 3
5th number: 5
6th number: 8
7th number: 13
8th number: 21
9th number: 34
10th number: 55
Recursive Fibonacci nth number: 55
0th number: 0
1th number: 1
2th number: 1
3th number: 2
4th number: 3
5th number: 5
6th number: 8
7th number: 13
8th number: 21
9th number: 34
Non-recursive Fibonacci nth number: 55
```

4.2 Factorial

```
0th number: 1
1th number: 1
2th number: 2
3th number: 6
4th number: 24
5th number: 120
6th number: 720
```

```
7th number: 5040
8th number: 40320
9th number: 362880
10th number: 3628800
Recursive Factorial nth number: 3628800
0th number: 1
1th number: 1
2th number: 2
3th number: 6
4th number: 24
5th number: 120
6th number: 720
7th number: 5040
8th number: 40320
9th number: 362880
10th number: 3628800
Non-recursive Factorial nth number: 3628800
```

5 My Programs

5.1 Main.cpp (Includes Both)

```
/*
 * main.cpp
 * CS202
 * April 7, 2020
 * Bryan Beus
 * Anchorage station for Iditarod Challenge
 */

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

using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;

// Print the current number and value in a sequence
```

```cpp
void printCurrentSetup(const int& current_num, const int& current_value) {
    cout << current_num << "th number: " << current_value << endl;
}

// Calculate and return the factorial value of the provided integer
int factorial_non_recursive(const int& final_num) {
    int current_num = 0;
    int current_value = 1;
    vector<int> values;

    // First term does not require a calculation
    while ((current_num < final_num) && current_num < 1) {
        printCurrentSetup(current_num, current_value);
        current_num++;
        values.push_back(current_value);
    }

    // Calculate the factorial value until the final value is reached
    while (current_num <= final_num) {
        current_value = current_num * values.at(current_num - 1);
        values.push_back(current_value);
        printCurrentSetup(current_num, current_value);
        current_num++;
    }

    return current_value;
}

// Calculate and return the factorial value using recursion
int factorial_recursive(const int& final_num, int& current_num, int& current_value, vector<int>& values) {

    // The first term has a different calculation pattern
    if (current_num == 0 && current_num < final_num) {
        printCurrentSetup(current_num, current_value);
        values.push_back(current_value);
        current_num++;
        printCurrentSetup(current_num, current_value);
        values.push_back(current_value);
        factorial_recursive(final_num, current_num, current_value, values);
    } else {
        while (current_num < final_num) {
            current_num++;
            current_value = current_num * values.at(current_num - 1);
            printCurrentSetup(current_num, current_value);
            values.push_back(current_value);
```

```
66              factorial_recursive(final_num, current_num,
                ↪  current_value, values);
67          }
68      }
69
70      return current_value;
71  }
72
73  // Calculate and return the Fibonacci value of the provided
    ↪  integer
74  int fibonacci_non_recursive(const int& final_num) {
75      int current_num = 0;
76      int current_value = 0;
77      vector<int> values;
78
79      // First term has unique calculation
80      if (current_num < final_num) {
81          printCurrentSetup(current_num, current_value);
82          values.push_back(current_value);
83          current_num++;
84          current_value++;
85          values.push_back(current_value);
86      }
87
88      // Second term has unique calculation
89      if (current_num < final_num) {
90          printCurrentSetup(current_num, current_value);
91          current_num++;
92          values.push_back(current_value);
93      }
94
95
96      while (current_num < final_num) {
97          printCurrentSetup(current_num, current_value);
98          current_num += 1;
99          current_value = values.at(current_num - 1) +
                ↪  values.at(current_num - 2);
100         values.push_back(current_value);
101     }
102
103     return current_value;
104 }
105
106 // Print the Fibonacci value of the provided integer using
    ↪  recursion
107 int fibonacci_recursive(const int& final_num, int& current_num,
    ↪  int& current_value, vector<int>& values) {
108
109     // First term has unique calculation
110     if (current_num == 0 && current_num < final_num) {
111         printCurrentSetup(current_num, current_value);
112         values.push_back(current_value);
113         current_num++;
```

```
114         current_value++;
115         printCurrentSetup(current_num, current_value);
116         values.push_back(current_value);
117         fibonacci_recursive(final_num, current_num,
            ↪  current_value, values);
118     } else {
119         while (current_num < final_num) {
120             current_num += 1;
121             current_value = values.at(current_num - 1) +
                ↪  values.at(current_num - 2);
122             printCurrentSetup(current_num, current_value);
123             values.push_back(current_value);
124             fibonacci_recursive(final_num, current_num,
                ↪  current_value, values);
125         }
126     }
127
128     return current_value;
129 }
130
131 int main() {
132
133     // Execute recursive fibonacci number finder
134     int number;
135     int final_num = 10;
136     int current_num = 0;
137     int current_value = 0;
138     vector<int> values_fibonacci;
139     number = fibonacci_recursive(final_num, current_num,
        ↪  current_value, values_fibonacci);
140     cout << "Recursive Fibonacci nth number: " << number << endl;
141
142     // Execute recursive fibonacci number finder
143     number = fibonacci_non_recursive(final_num);
144     cout << "Non-recursive Fibonacci nth number: " << number <<
        ↪  endl;
145
146     // Execute recursive factorial number finder
147     current_num = 0;
148     current_value = 1;
149     vector<int> values_factorial;
150     number = factorial_recursive(final_num, current_num,
        ↪  current_value, values_factorial);
151     cout << "Recursive Factorial nth number: " << number << endl;
152
153     // Execute recursive factorial number finder
154     number = factorial_non_recursive(final_num);
155     cout << "Non-recursive Factorial nth number: " << number <<
        ↪  endl;
156
157     return 0;
158 }
```