Project 7: C++ Exercises

Owen O'Reilly | 12/15/2024 | AMS-595

Summary

This assignment was a demonstration of basic C++ coding, involving *if* statements, *for* and *while* loops, variable declaration and manipulation, and methods.

Section 0: Print Vector

A *void* method was created that would accept a vector input and use the console to print out each of its elements via a simple *for* loop. This was used in other exercises requiring the printing of an array throughout the assignment.

Section 1: Switch Statement

The statement provided was translated from Matlab code into C++. It prompts the user for a number and prints different statements based on that input.

```
Enter a number: 1
Positive One
```

Section 2: Fibonacci Sequence

Two *int* variables and a *while* loop were used in a "leapfrog" manner to generate numbers from the Fibonacci sequence up to a value of 4,000,000.

```
Fibonacci sequence up to 4,000,000:
2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1
597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025,
121393, 196418, 317811, 514229, 832040, 1346269, 2178309,
3524578,
```

Section 3: isPrime()

A method was created that accepts an *int* argument and uses a *for* loop to exhaustively check every potential factor of a number. The method returns *true* if the number passed in is prime and *false* if not. Three test cases were used to evaluate the method's accuracy.

```
isprime(2) = 1
isprime(10) = 0
isprime(17) = 1
```

Section 4: Factorize()

A method was created to generate every factor of an input *int*. This was done by exhaustively iterating through every potential factor of the number and checking if its modulus returned 0 (including 1 and the number itself). Three test cases were used to evaluate the method's accuracy.

```
Factors of 2:[1, 2]
Factors of 72:[1, 2, 3, 4, 6, 8, 9, 12, 18, 24, 72]
Factors of 196:[1, 2, 4, 7, 14, 28, 49, 196]
```

Section 5: primeFactorize()

A method was created to generate every prime factor of an input *int*. This was done in the same manner as Factorize(), with the addition of checking both the modulus and primeness of the potential factor. Three test cases were used to evaluate the method's accuracy.

```
Prime factors of 2:[1]
Prime factors of 72:[1, 2, 3]
Prime factors of 196:[1, 2, 7]
```

Section 6: pascalTriangle()

A method was created to generate a row of Pascal's triangle, specified by an *int* parameter input. This was accomplished by creating two arrays of length = row within the method, then iteratively summing one array's adjacent entries and storing the resulting values in the array not currently being manipulated. By conducting these operations twice in a single loop, two new rows could be generated at every cycle, and the most recently operated array returned when the array had been fully populated. The method was tested by printing rows 1-10 of the triangle.

```
Pascal's Triangle: First 10 Rows
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
```

Discussion

These exercises were an excellent demonstration of the power that C++ provides the user, as well as the attention to detail that is required to effectively program in the language. A particular challenge was determining a way to dynamically construct arrays or pre-allocate space, as there is much less native support for arrays in C++ compared to the Python programming that we have been doing thus far.