CPSC 2150 – Algorithms and Data Structures II
Lab3: Recursion as Problem Solving Tool
Total - 50 Marks

Learning Outcomes

- Apply recursion as a problem-solving technique
- Implement and program recursive functions with C++

Resources

- Chapter 3.5 of the text book

Description

Part 1- Eric Goes Home: Eric lives in a world composed of streets and avenues laid out in a regular rectangular grid that looks like this:

Suppose that he lives on the intersection of 1st Street and 2nd Avenue as shown in the diagram. He has a meeting with Gary in a cafe located in intersection of Broadway and Main Street as shown in the diagram. Even if Eric wants to avoid going out of the way, there are still several equally short paths from his home to the cafe. For example, in this diagram there are three possible routes, as follows:

- Move left, left, down.
- Move left, down, left.
- Move down, left, left.

A. [20 marks] write a recursive function named **numPathsFromHome** that given the starting position, returns the number of paths Eric could take to go to the cafe, subject to the condition that Eric doesn't want to take any unnecessary routes, and can therefore only move west or south (left or down in the diagram). Watch out for the edge case where the street or avenue numbers are zero!

Use the following function prototype:

```
int numPathsFromHome(int street, int avenue);
```

B. [5 marks] Calculate the time complexity of **numPathsFromHome**. (answers.pdf)

Part 2 - Subsequences: If S and T are strings, we say that S is a subsequence of T if all the letters of S appear in T in the same relative order that they appear in S. For example, ***pin*** is a subsequence of the ***programming***, and the string ***singe*** is a subsequence of ***springtime***. However, ***steal*** is not a subsequence of ***least***, since the letters are in the wrong order, and ***p*** is not a subsequence of ***team*** because there is no ***p*** in ***team***.

Note that the empty string ("") is a subsequence of every string.

A. [20 marks] Write a function named **hasSubsequence** that given two c-strings returns a Boolean to represent whether the second string is a subsequence of the first.

Your solution must be recursive and must use the following function prototype:

```
bool hasSubsequence(const char[] T, const char[] S);
```

Note: This problem has a beautiful recursive decomposition. Think about the following questions:

- What strings are subsequences of an empty string?
- What happens if the first character of the parameter **S** matches the first character of the parameter **T**? What happens if it doesn't?

You can assume that the cstrings are case-sensitive, so the word ***AGREE*** is not a subsequence of the word ***agreeable***.

B. **[5 bonus marks]** Calculate the time complexity of **hasSubsequence. (answers.pdf)**

Part 3 - [5 marks] Write a main function that calls and tests numPathsFromHome and hasSubsequence.

Submit to D2L

*Make a **zip file** named* **StudentNumber-*lab3.zip*** *including **answers.pdf** and **lab3.cpp*** by the end of the lab time**.** For example, if your student number is 10023449, the submitted file must be named as **10023449-lab3.zip**.