Assignment 5

Due: November 13, 2018 (11:59pm)

Assignment

You will write three small recursive problems to solve different problems.

Purpose

Ensure you know how to write recursive functions.

Before starting

Read Chapter 9.1 and 9.3.

How to Turn In

Submit all source code files (`*.cpp`, `*.cxx`, `*.h`) in a collection as a zip on Blackboard.

Files

- `recursion.cpp`: This is the only file you need to work in. You need to write the three functions using recursive calls as described below. Your functions should require no loops.
- `maze.h` and `maze.cpp` : A header and implementation file for the `maze` and `maze_node` class. Their use is described below for problem 3.

The triangle function

See the post condition of the `triangle` function call in the attached files. Create the function which prints the shape described there. Hint: Only one of the arguments changes in the recursive call.

The stepping stone function

You're standing at the edge of a pond with stepping stones leading to the middle of the pond. The stepping stones are in a straight line with the last one being in the center of the pond. You can take a small stride to move one stepping stone, and a large stride to move two (skipping one). You want to count the number of ways to step to the middle of the pond. For example, a pond with three stepping stones can be traversed in three different ways: 1) three small strides, 2) one small stride followed by one large stride, 3) one large followed by one small. A pond with four stepping stones can be traversed in five different ways.

Write the recursive function
`int count_ways_to_step(int number_of_stepping_stones)` that takes a positive `number_of_stepping_stones` value and returns the number of different ways to traverse the pond.

Hint: Wherever you are you have two choices of how to make your next step (except if you are 1 stepping stone from the finish).

The maze solving function

Imagine a maze which at every intersection branches either to the left, to the right, or both at each new intersection, as shown in Figure 1. Starting from the blue node, you want to know the path to the green node, taking either the right or left paths (moving upward from the bottom).

Write a function which will print this path printing 0s for left turns and 1s for right turns. You may print the path in reverse order (this will be easier).

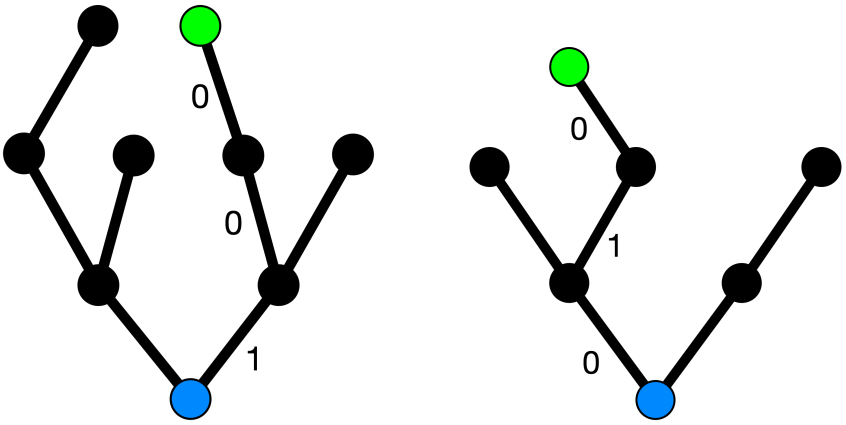
Use the `maze` class provided by `maze.h` to test your maze solving code. The constructor for maze takes an integer for which maze to build. Maze 0 and 1 are provided. The `get_start()` will give you the starting node of the maze. From each `maze_node` you can check if the left() or `right()` function of that `maze_node` leads to another maze node or to the `nullptr`. The `nullptr` signifies a deadend in the maze. The `is_finish()` function of a `maze_node` will return `true` if it is the finish node and `false` otherwise. You do not need to change the maze code in any way, unless you want to add additional mazes to test. Your code will be tested a different maze than just these two during grading. You can assume there will always be a finish node with an available path to it. You can also assume that each node has only one node that leads to it.

Other notes

There is no explicit exam script for this assignment. However, the results are fairly straight forward and should be easy to check on your own.

Optional (for extra 10% points)

Add an additional maze solving function. This version should be able to handle mazes which have multiple finish nodes. Additionally it must handle mazes



(a) The maze which will be built using the constructor `maze(0)`.

(b) The maze which will be built using the constructor `maze(1)`.

Figure 1: Examples of mazes and the corresponding 0s and 1s for turns to get from the start to the finish.

which have branches that join with other branches, but only with other branches that have had the same number of turns taken so far. That is, each turn takes you one level deeper into the maze, and nodes which are n levels deep may have 2 paths that lead to them from the $n-1$ depth. These the left and right paths do not need to make spatial sense (a path which has only taken left turns so far may join with a branch that has only taken right). Write a recursive function which prints all possible paths to all possible finishes. This recursive function may return any type you want, even a custom class. You may also write helper functions if they are useful.