University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics

Program 2: Recursion

Purpose

The programming assignment will provide an exercise in using recursion. There are two problems which need to be solved. The first asks for a recursive function to calculate the Catalan number. This problem will also require the student to utilize command line arguments for their program. The second exercise is a path-finding problem in two-dimensional space. The second problem will require the student to have good overall factoring and class design. This will be a part of the final grade.

This assignment will also require the student to use Linux to build and run their applications as we will test these programs solely on the Linux Lab systems.

Problem 1: The good Mr. Catalan.

The Catalan numbers form a sequence of natural numbers that occur in many counting problems. They are named after the mathematician Eugene Charles Catalan (http://en.wikipedia.org/wiki/Eug%C3%A8ne_Charles_Catalan) and are defined by the following recursive formula:

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i \, C_{n-i} \quad \text{for } n \geq 0;$$

Write a program called Catalan which takes one argument on the command line and calls a recursive function which computes the nth Catalan number. The program then prints out the result to std::cout.

Make sure that your program gracefully handles negative numbers and those which you deem too large to solve. Also, your solution must be recursive mirroring above formula.

For instance,

```
catalan 4
 14
Or
catalan 10
 16796
```

Problem 2: "The Greedy Robot" or "Lost in the Supermarket"

A robot is positioned on an integral point in a two-dimensional coordinate grid (x_r, y_r). There is a treasure that has been placed at a point in the same grid at (x_t, y_t). All x's and y's will be integral values. The robot can move up (North), down (South), left (West), or right (East). Commands can be given to the robot to move one position in one of the four direction. That is, "E" moves a robot one slot East (to the right) so if the robot was on position (3, 4), it would now be on (4, 4). The command N would move the robot one position north so a robot at position (4, 4) would be at (4, 5).

Because the robot cannot move diagonally, the shortest distance between a robot at (x_r, y_r) and a treasure at (x_t, y_t) is

$$| x_r – x_t | + | y_r - y_t | = \text{ShortestPossibleDistance}$$

Write a recursive program which determines all the unique shortest possible paths from the robot to the treasure with the following stipulation: *The robot may never move in the same direction more than* ***max_distance*** *times in a row.*

The input to the program will be the max_distance value followed by the position of the robot (x_r, y_r), followed by the position of the treasure (x_t, y_t). Assume that all five are integers and do not worry about error conditions in inputs. Take these parameters as arguments to the program on the command line.

Invoking greed_robot:

% greedy_robot **max_distance robot_x robot_y treasure_x treasure_y**

max_distance: maximum number of moves in one direction before a turn is required
robot_x: starting X coordinate of robot
robot_y: starting Y coordinate of robot
treasure_x: X coordinate of treasure
treasure_y: Y coordinate of treasure

For instance, an input of 2 1 3 -2 4 corresponds to the robot starting at position (1, 3) and needing to get to the treasure at position (-2, 4) with the constraints that one can only move 2 steps in one direction before having to shift to a new position.

The output of the program should be the listing of all the unique shortest possible paths followed by the number of unique paths. The paths must follow the stipulation whereby the robot cannot move in the same direction more than max_distance times in a row. A path should be output as a string of characters with each character corresponding to a direction the Robot should move. For instance, NNEEENENE corresponds to having the robot move North, North, East, East, East, North, East, North, East. This would be one answer to the input: 3 1 2 6 6, which corresponds to (1, 2) -> (6,6) with a max_distance of 3.

Notice that not all combinations of robots / treasures will have a solution. As there may not be a ShortestPossibleDistance given the stipulation. For instance, 2 3 3 3 7 has shortest possible distance of 4 but no way to get there in that distance without going North more than 2 times in a row.

For the input 2 1 2 3 5 which corresponds to (1,2) -> (3,5) the output should be:

```
% greedy_robot 2 1 2 3 5
NNENE
NNEEN
NENNE
NENEN
NEENN
ENNEN
ENENN
Number of paths:  7
```

Testing greedy_robot

A shell script is distributed with this project that tests a few different combinates of inputs for the greedy_robot executable. This script should be expanded to comprehensively the greedy_robot program. For these tests you may want to think of moving the robot in all directions, different distances, as well as straight lines. The test cases may also want to explore the limits of your program.

You will turn in the enhanced testing scripts as well as the results of your program running these tests.

NOTE 1: For both programs' inputs are given as arguments to the program.
NOTE 2: This program must be solved recursively, creating paths as the robot moves towards the treasure

Turn In:

- In a .zip file
  - All .cpp / .h files required to build catalan
  - All .cpp / .h files required to build greedy_robot

- **robot_tests** script showing the test cases you created for your robot
- **robot_tests_results.txt** showing the results of your test cases running