EECS 268: Spring 2009
Laboratory 5: Advanced Recursion – Recursive Backtracking

Due: 11:59:59 pm the day before your lab meets the week of February 23

Lecture Topics: Advanced Recursion
Lab Topics: string stream class, vector class

1. Introduction

The KU Sudoku Team has commissioned you to create a Sudoku solver to test their latest batch of puzzles. For a particular puzzle, they want your program to print *both* the board as read from the file *and* either a single valid solution or a message indicating that no solution is available. The primary goal of this lab is to utilize modern programming language tools in algorithmic problem solving.

You are required to implement your Sudoku solver using a recursive backtracking algorithm.

The rules of Sudoku are simple: you are given a 9x9 grid (with nine rows, nine columns, and nine 3x3 boxes). Each row, column, and 3x3 box must contain the numbers 1 through 9 with no repeats.

As stated above, you must use a recursive backtracking strategy similar to the *8 Queens* problem given in your textbook. If you try every value at every location to find one that works, your algorithm will be computationally inefficient. Rather than test every possible value at every location, you may easily eliminate the value options you try at a location by skipping values at are already used within the location's row, column, or 3x3 box.

2. Class Structure

The Sudoku solver will be implemented as a class with a public interface to load a puzzle from file, call the solve procedure, and print a puzzle to file.

To store the values representing the current state of the board the stl vector template class will be used. This can be done as a single vector or a set of nested vectors to create a two dimensional structure.

Iteration, when possible, will be done using a vector iterator. One location where this should be used is in the print method.

3. Input & Output

Command line I/O

From the command line, the user will specify the input file containing one Sudoku puzzle and one output file to write the results to.

$ ./lab5 <input_file> <output_file>

If the program is solvable, the program will write *both* the board as read from the file *and* a single valid solution to file. Your output must be readable such that the user may easily see the 9x9 grid and a value for every location. If no solution exists, the program will write *both* the board as read from the file *and* a message indicating that no solution is available.

Input File Format

The format of the puzzle will be 81 non-blank characters that you will read one by one. The first 9 characters will be the top row of the Sudoku puzzle; the final 9 characters will be the bottom row of the puzzle. Each non-blank character will either be a digit ($1 \leq d \leq 9$) or an underscore (_) signifying a blank spot in the puzzle. (There

may or may not be blank characters, new lines, and other white space between the non-blank characters.)

```
_ 1 _ 4 _ _ _ _ 2
_ _ 4 _ 7 2 _ _ _
_ 8 3 _ _ 5 _ 7 _
_ 3 8 2 _ 7 _ _ _
_ 5 2 1 _ 4 8 3 _
_ _ _ 5 _ 3 7 2 _
_ 2 _ 3 _ _ 9 4 _
_ _ _ 7 2 _ 6 _ _
9 _ _ _ _ 6 _ 1 _
```

4. Grading Criteria

To improve your understanding of backtracking recursion, please fully design and implement the Sudoku solver on your own. Discussion with your neighbor is acceptable. However, you must submit your own original work, and not the work of others OR work from previous semesters. The TAs have been asked to monitor this assignment closely.

50 **Recursive backtracking algorithm**
⇒ Must use a valid backtracking algorithm with recursion to get more than 10% in this section.
10 **Proper use of the vector class**
10 **Correct handling of input and output**
15 **Code style & output formatting**
15 **Documentation**