BİL 102 – Computer Programming HW 08

Last Submission Date: May 6, 2013 – 13:59

1. **(20Pts)**Provide iterative implementations to all recursive functions operating on sets in the case study of chapter 10.5 in your text book.
2. **(20Pts)**Provide recursive implementations to “strchr” and “strrchr” functions of string library using the names “strChr” and “strRChr”. **You are not allowed to change the prototypes or use helper functions**.
3. **(30 Pts)**Permutation has the following iterative and recursive definitions:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \; \forall \, integers \; n, k > 0$$
$$\binom{n}{0} = 1, \qquad n \geq 0,$$
$$\binom{0}{k} = 0, \qquad k > 0$$

   Implement the following recursive and iterative functions function to calculate permutation:

   int getPermutationIter(int n, int k)
   int getPermutationRec(int n, int k)

4. **(30 Pts)**In this part you will solve 4x4 Sudoku puzzles. You will write a recursive function to solve any solvable 4x4 Sudoku. Unknown elements will have 0 values in the Sudoku. You will get the puzzle from a text file named **SudokuIn.txt** and write the solution to **SudokuOut.txt**. Implement the following functions:

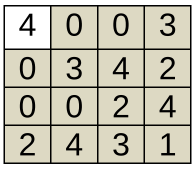
   int readSudoku(const char fName[], int sudoku[][4]): gets Sudoku from the file with given name return 0 on success, -1 if file not found, -2 if the file has a bad format.

   int solveRec(int sudoku[][4], value): a **recursive** function to solve Sudoku. Finds the first unspecified element, assigns the parameter value to it, and calls itself with all possible values for the next unknown unless it is in the base case. Note that when returning from an unsuccessful call, recorded values should be deleted (should be marked as unknown)

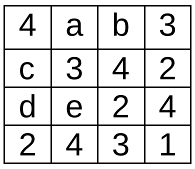
   /*a wrapper function for solveRec*/
   int solve(int sudoku[][4])

   void printSudoku(FILE* stream, int sudoku[][4])

“solveRec()” will consider all possibilities ($4^{unknown}$ possibilities exists) until it finds a solution. Consider the following Sudoku. Solution of the puzzle is the solution of the shaded part.

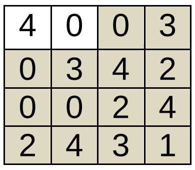


4	0	0	3
0	3	4	2
0	0	2	4
2	4	3	1



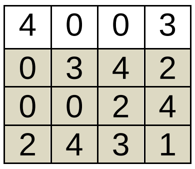
4	a	b	3
c	3	4	2
d	e	2	4
2	4	3	1

If the shaded part in the following figure is solved this solution can be used to solve the shaded part above:



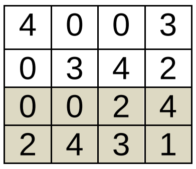
4	0	0	3
0	3	4	2
0	0	2	4
2	4	3	1

Similarly, the solution of the shaded part below can be used for the solution of the shaded part above:

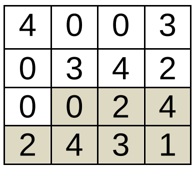


4	0	0	3
0	3	4	2
0	0	2	4
2	4	3	1

The same rule applies for the following figures as well:



4	0	0	3
0	3	4	2
0	0	2	4
2	4	3	1



4	0	0	3
0	3	4	2
0	0	2	4
2	4	3	1

Therefore, the solution can be built recursively. In a function, recursive calls will be performed for each possible values of an unknown. Study the following table which demostrates the function calls for the Sudoku above.

sR: solveRecursively()

1st Level calls	2nd Level Calls	3rd Level Calls	4th Level Calls	5th Level Calls	Unknowns				
					a	b	c	d	e
					0	0	0	0	0
sR(s,1)					1	0	0	0	0
	sR(s,1)				1	1	0	0	0
	sR(s,2)				1	2	0	0	0
	sR(s,3)				1	3	0	0	0
	sR(s,4)				1	4	0	0	0
sR(s,2)					2	0	0	0	0
	sR(s,1)				2	1	0	0	0

		sR(s,1)			2	1	1	0	0
			sR(s,1)		2	1	1	1	0
			sR(s,2)		2	1	1	2	0
			sR(s,3)		2	1	1	3	0
				sR(s,1)	2	1	1	3	1

5. **BONUS (30Pts)**: Can we apply the algorithm we used in part4 for a 9x9 Sudoku? In principle it should be OK but what about the complexity? Does it end in a reasonable time, can it work with reasonable resources? In part 4, in worst case we try all possibilities. So, if there are 40 unknowns, there exist 9^{40} possibilities to test, which is clearly impossible to handle. Study the algorithm for solveSudoku() in hw6. Improve this algorithm using a recursive function such that puzzles unsolvable by using this algorithm should be solvable if they really have really a solution in a reasonable time using reasonable resources (without segmentation or insufficient memory errors). Also provide some basic I/O functions for a 9x9 Sudoku. Read the Sudoku from a text file Sudoku9x9.txt

General:
1. Obey honor code principles.
2. Obey coding convention.
3. Do not forget to put the required **tags** in the main function.
4. Your submission should include the following file only:
   HW08_<student_name>_<studentSirname>_<student number>.c
5. Deliver the printout of your code **until the last submission date**.
6. Do not use non-English characters in any part of your homework (in body, **file name**, etc.).