Assignment 2

COMP9021, Session 1, 2013

1 Aims

The main purpose of the assignment is to let you practice the following programming techniques:

- read data from standard input and store them in an array;
- perform operations on arrays;
- execute tests and repetitions;
- make use of the internal representation of data.

2 General presentation

A sudoku grid consists of 9 lines and 9 columns, making up 81 cells, that are grouped in nine 3x3 boxes. In a sudoku puzzle, some but not all of the cells already contain digits between 1 and 9. Here is an example of a sudoku puzzle.

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

Solving a sudoku puzzle means completing the grid so that each digit from 1 to 9 occurs once and only once in every row, once and only one in every column, and once and only once in every box. For instance, the previous puzzle has the following solution.

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

Solving a sudoku puzzle is a very common assignment; it is not difficult and moderately interesting as a "solution" (the completed grid) tells nothing about *how* the solution was reached. More interesting solvers are *logical* in the sense that they (possibly partially only) solve the puzzle in steps and at every step, explain how they made some progress; they do so by using some of the well known techniques that most people who solve sudoku puzzles apply. Two remarks are in order.

- Methods that only discover digits in empty cells are fairly limited; most methods need to keep track of the list of possible digits that can go into a given cell, and making progress might mean reducing that list. To apply techniques of the second kind, it is necessary to first *mark* the grid.
- Often, it is not possible to completely solve a puzzle using exclusively the chosen methods; at some point no progress can be made and then a *random guess* has to be made to either put a digit into a given empty cell, or to remove a digit from the list of possible digits that can go into a given cell. It might subsequently be necessary to *backtrack* and make alternative guesses if the earlier guesses turn out to be inconsistent with a solution.

For this assignment, you will have to implement two such techniques, based on the notions of *forced digits* and *preemptive sets* described in the paper *A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles* by J. F. Crook, Notices of the AMS, 56(4), pp. 460–468. Before anything else, you should study this paper.

The forced digits technique is applied first, followed by the preemptive set technique. When no progress can be made, the forced digits techniques could be applied again, but that might not yield anything; an alternative would be to try and fill some empty cell with one of the possible digits for that cell and apply the preemptive set technique applied again, knowing that that guess might prove wrong and that other possible digits might have to be used instead. In this assignment, we will stop at the point where the preemptive set technique can no longer be applied; hence we can expect that our implementation will only partially solve most puzzles. But the technique is very powerful and as explained in the article, subsumes many of the well known techniques.

You will design and implement a program that will read a sudoku grid whose representation is stored in a file and whose contents is redirected to standard input, and

- either output whether the representation is correct and has no digit that occurs twice on the same row, on the same column or in the same box,
- or output some Latex code, to be redirected to a file, from which a pictorial representation of the grid can be produced,
- or output some Latex code, to be redirected to a file, from which a pictorial representation of the grid to which the forced digits technique has been applied can be produced,
- or output some Latex code, to be redirected to a file, from which a pictorial representation of the grid to which the forced digits technique has been applied and that has been marked can be produced,
- or output some Latex code, to be redirected to a file, from which a pictorial representation of the grid to which the forced digits technique has been applied, that has been marked and to which the preemptive set technique has been applied can be produced.

3 Examples

3.1 First example

Given a file named `sudoku1.txt` whose contents is

```
0 0 1 9 0 0 0 0 8
6 0 0 0 8 5 0 3 0
0 0 7 0 6 0 1 0 0
0 3 4 0 9 0 0 0 0
0 0 0 5 0 4 0 0 0
0 0 0 0 1 0 4 2 0
0 0 5 0 7 0 9 0 0
0 1 0 8 6 0 0 0 7
7 0 0 0 0 9 2 0 0
```

your program run as `a.out <sudoku1.txt` should output

```
There is clearly no solution.
```

3.2 Second example

Given a file named `sudoku2.txt` whose contents is

```
0 0 1 9 0 0 0 0 8
6 0 0 0 8 5 0 3 0
0 0 7 0 6 0 1 0 0
0 3 4 0 9 0 0 0 0
0 0 0 5 0 1 0 0 0
0 0 0 0 1 0 4 2 0
0 0 5 0 7 0 9 0 0
0 1 0 8 4 0 0 0 7
7 0 0 0 0 9 2 0 0
```

your program run as `a.out <sudoku2.txt` should output

```
There is clearly no solution.
```

3.3 Third example

Consider a file named sudoku3.txt whose contents is

```
0 0 1 9 0 0 0 0 8
6 0 0 0 8 5 0 3 0
0 0 7 0 6 0 1 0 0
0 3 4 0 9 0 0 0 0
0 0 0 5 0 4 0 0 0
0 0 0 0 1 0 4 2 0
0 0 5 0 7 0 9 0 0
0 1 0 8 4 0 0 0 7
7 0 0 0 0 9 2 0 0
```

When run as `a.out <sudoku3.txt` your program should output

```
There might be a solution.
```

When run as `a.out bare <sudoku3.txt`, your program should produce some output which if redirected to a file, say `sudoku3_bare.tex`[1], can be given as argument to `pdflatex` to produce a file named `sudoku3_bare.pdf` that views as the grid on page 1.

[1] by actually running `a.out bare <sudoku3.txt >sudoku3_bare.tex`

When run as `a.out forced <sudoku3.txt`, your program should produce some output which if redirected to a file, say `sudoku3_forced.tex`[2], can be given as argument to `pdflatex` to produce a file named `sudoku3_forced.pdf` that views as follows.

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

[2]by actually running `a.out forced <sudoku3.txt >sudoku3_forced.tex`

When run as `a.out marked <sudoku3.txt`, your program should produce some output which if redirected to a file, say `sudoku3_marked.tex`[3], can be given as argument to `pdflatex` to produce a file named `sudoku3_marked.pdf` that views as follows.

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

When run as `a.out worked <sudoku3.txt`, your program should produce some output which if redirected to a file, say `sudoku3_worked.tex`[4], can be given as argument to `pdflatex` to produce a file named `sudoku3_worked.pdf` that views identically to `sudoku3_marked.pdf`.

[3] by actually running `a.out marked <sudoku3.txt >sudoku3_marked.tex`

[4] by actually running `a.out worked <sudoku3.txt >sudoku3_worked.tex`

3.4 Fourth example

Consider a file named `sudoku4.txt` whose contents is

```
039500000
000800070
000010904
100400003
000000000
007000860
006708200
010090005
000001008
```

When run as `a.out <sudoku4.txt` your program should output

```
There might be a solution.
```

When run as `a.out bare <sudoku4.txt`, your program should produce some output which if redirected to a file, say `sudoku4_bare.tex`[5], can be given as argument to `pdflatex` to produce a file named `sudoku4_bare.pdf` that views as follows.

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

[5] by actually running `a.out bare <sudoku4.txt >sudoku4_bare.tex`

When run as `a.out forced <sudoku4.txt`, your program should produce some output which if redirected to a file, say `sudoku4_forced.tex`[6], can be given as argument to `pdflatex` to produce a file named `sudoku4_forced.pdf` that views as follows.

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

[6]by actually running `a.out forced <sudoku4.txt >sudoku4_forced.tex`

When run as `a.out marked <sudoku4.txt`, your program should produce some output which if redirected to a file, say `sudoku4_marked.tex`[7], can be given as argument to `pdflatex` to produce a file named `sudoku4_marked.pdf` that views as follows.

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

[7] by actually running `a.out marked <sudoku4.txt >sudoku4_marked.tex`

When run as `a.out worked <sudoku4.txt`, your program should produce some output which if redirected to a file, say `sudoku4_worked.tex`[8], can be given as argument to `pdflatex` to produce a file named `sudoku4_worked.pdf` that views as follows.

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

[8] by actually running `a.out worked <sudoku4.txt >sudoku4_worked.tex`

3.5 Fifth example

Consider a file named sudoku5.txt whose contents is

```
0 9 0 7 0 0 8 6 0
0 3 1 0 0 5 0 2 0
8 0 6 0 0 0 0 0 0
0 0 7 0 5 0 0 0 6
0 0 0 3 0 7 0 0 0
5 0 0 0 1 0 7 0 0
0 0 0 0 0 0 1 0 9
0 2 0 6 0 0  3 5 0
0 5 4 0 0 8 0 7 0
```

When run as `a.out <sudoku5.txt` your program should output

```
There might be a solution.
```

When run as `a.out bare <sudoku5.txt`, your program should produce some output which if redirected to a file, say `sudoku5_bare.tex`[9], can be given as argument to `pdflatex` to produce a file named `sudoku5_bare.pdf` that views as follows.

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

[9]by actually running `a.out bare <sudoku5.txt >sudoku5_bare.tex`

When run as `a.out forced <sudoku5.txt`, your program should produce some output which if redirected to a file, say `sudoku5_forced.tex`[10], can be given as argument to `pdflatex` to produce a file named `sudoku5_forced.pdf` that views as follows.

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

[10] by actually running `a.out forced <sudoku5.txt >sudoku5_forced.tex`

When run as `a.out marked <sudoku5.txt`, your program should produce some output which if redirected to a file, say `sudoku5_marked.tex`[11], can be given as argument to `pdflatex` to produce a file named `sudoku5_marked.pdf` that views as follows.

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

[11] by actually running `a.out marked <sudoku5.txt >sudoku5_marked.tex`

When run as `a.out worked <sudoku5.txt`, your program should produce some output which if redirected to a file, say `sudoku5_worked.tex`[12], can be given as argument to `pdflatex` to produce a file named `sudoku5_worked.pdf` that views as follows.

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

[12] by actually running `a.out worked <sudoku5.txt >sudoku5_worked.tex`

4 Detailed description

4.1 Input

The input is expected to consist of 9 lines of 9 digits, with possibly lines consisting of spaces only that will be ignored and with possibly spaces anywhere on the lines with digits.

4.2 Output

If not run as `a.out`, `a.out bare`, `a.out forced`, `a.out marked` or `a.out worked` (followed by `<`*`filename`* where *`filename`* is the name of a file that stores the input) then the program should print out two lines that read

```
I expect no command line argument or "bare", "forced", "marked" or "worked"
         as unique command line argument.
```

and immediately exit. Otherwise, if the input is incorrect, that is, does not satisfy the conditions spelled out in the previous section, then the program should print out a single line that reads

```
Incorrect input.
```

and immediately exit.

4.2.1 When a.out is run with no command-line argument

If the input is correct and the program is run as `a.out` (followed by `<`*`filename`* where *`filename`* is the name of a file that stores the input) then the program should output either

```
There is clearly no solution.
```

or

```
There might be a solution.
```

The first output indicates that some row, column or box contains twice the same digit. The second output indicates that no row, column or box contains twice the same digit.

Pay attention to the expected format, including spaces. Note that your program should output no blank line. For a given test, the output of your program will be compared with the expected output; your program will pass the test if and only if both outputs are absolutely identical, character for character, including spaces. For the provided examples, the expected outputs are available in files that end in

_output.txt. To check that the output of your program on those examples is correct, you can redirect it to a file and compare the contents of that file with the contents of the appropriate _output.txt file using the diff command. If diff silently exits then your program passes the test; otherwise it fails it. For instance, run

```
a.out <sudoku1.txt >sudoku1_my_output.txt
```

and then

```
diff sudoku1_my_output.txt sudoku1_output.txt
```

to check whether your program succeeds on the first provided example.

4.3 When a.out is run with bare as command-line argument

If the input is correct and the program is run as a.out bare (followed by <*filename* where *filename* is the name of a file that stores the input) then the program should output some lines that redirected to a file, say sudoku.tex, can be given as an argument to pdflatex to produce a file named sudoku.pdf that depicts the grid. The provided examples will show you what sudoku.tex should contain.

Pay attention to the expected format, including spaces and blank lines. Lines that start with % are comments; there are 9 such lines. The output of your program redirected to a file will be compared with the expected output saved in a file (of a different name of course) using the diff command. For your program to pass the associated test, diff should silently exit, which requires that the contents of both files be absolutely identical, character for character, including spaces and blank lines. Check your program on the provided examples using the associated .tex files. For instance, run

```
a.out bare <sudoku1.txt >my_sudoku1_bare.tex
```

and then

```
diff my_sudoku1_bare.tex sudoku1_bare.tex
```

to check whether your program succeeds on the first provided example.

4.4 When a.out is run with forced as command-line argument

If the input is correct and the program is run as a.out forced (followed by <*filename* where *filename* is the name of a file that stores the input) then the program should output some lines that redirected to a file, say sudoku_forced.tex, can be given as an argument to pdflatex to produce a file named sudoku_forced.pdf that depicts the grid where all forced digits have been added. A forced digit is a digit that must fill an empty cell in a box because that box does not contain that digit yet and all other

empty cells in that box are on a row or on a column that contains that digit. As forced digits are being discovered and fill empty cells, more forced digits might be discovered that could not be discovered in the first round. So the program must make sure that no forced digit can be added to the grid that will be produced when running `a.out forced`. The provided examples will show you what `sudoku_forced.tex` should contain.

The comments on the output of the program given in Section 4.3 apply.

4.5 When a.out is run with marked as command-line argument

If the input is correct and the program is run as `a.out marked` (followed by `<`*filename* where *filename* is the name of a file that stores the input) then the program should output some lines that redirected to a file, say `sudoku_marked.tex`, can be given as an argument to `pdflatex` to produce a file named `sudoku_marked.pdf` that depicts the grid where all forced digits have been added and all possible digits have been added to the corners of the empty cells. The possible digits for an empty cell are the the digits that do not occur on the same row, on the same column or in the same box. The provided examples will show you what `sudoku_marked.tex` should contain.

The comments on the output of the program given in Section 4.3 apply.

4.6 When a.out is run with worked as command-line argument

If the input is correct and the program is run as `a.out worked` (followed by `<`*filename* where *filename* is the name of a file that stores the input) then the program should output some lines that redirected to a file, say `sudoku_worked.tex`, can be given as an argument to `pdflatex` to produce a file named `sudoku_worked.pdf` that depicts the grid where all forced digits have been added, all possible digits have been added to the corners of the empty cells, and the preemptive set technique has been applied until it cannot allow one to eliminate any possible digit from any cell (which might be because the puzzle has been solved). The provided examples will show you what `sudoku_worked.tex` should contain.

The comments on the output of the program given in Section 4.3 apply.

5 Assessment and submission

5.1 Assessment

Up to eight marks will reward correctness of solutions by automatically testing your program on some tests, all different to the provided examples. Read carefully the part on program output to maximise your chances of not failing some tests for stupid reasons.

Up to one mark will reward good formatting of the source code and reasonable complexity of the underlying logic as measured by the level of indentation of statements. For that purpose, the `mycstyle` script will be used, together with your customised `style_sheet.txt`, that you have to submit, and in which **Maximum level of indentation has to be set to 7** for this assignment. If the script identifies

problems different to excessive indentation levels then you will score 0 out of 1. If the script identifies excessive indentation levels (which means that at least one line exhibits at least 6 indentation levels), then you will score 0.5 out of 1. If the script identifies no problem then you will score 1 out of 1. If your program attempts too little and contains too little code then the `mycstyle` script won't be used and you will score 0 out of 1.

Up to one mark will reward good comments, good choice of names for identifiers and functions, readability of code, simplicity of statements, compactness of functions. This will be determined manually.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

5.2 Submission

Your program will be stored in a file named `sudoku.c`, which has to be developed from the provided template. You can add whatever you want but do not remove anything from the template, and do not modify the `main()` function.

Go through `assignment_checklist.pdf`, and make sure that you can tick all boxes (or at the very least, are aware that you should tick them all). Then upload your files (the source code of your program and your customised `style_sheet.txt`) using WebCMS. Assignments can be submitted more than once: the last version is marked. Your assignment is due by May 12, 11:59pm.

5.3 Reminder on plagiarism policy

You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of algorithms, not C code. But you must implement the solution on your own. Submissions are routinely scanned for similarities that occur when students copy and modify other people's work, or work very closely together on a single implementation. Severe penalties apply to a submission that is not the original work of the person submitting it.