Problem set 1
TDT4205, Spring 2013

Deadline: 01.02.2013 at 20.00 Contact course staff if you cannot meet the deadline.

Evaluation: Pass/Fail

Delivery: Use It's Learning. Deliver exactly two files:

- *yourusername_ps1.pdf*, with answers to the theory questions
- *yourusername_code_ps1.*{*zip* |*tar.gz* |*tar*} containing only the code you have modified.

General notes: All problem sets are to be done **INDIVIDUALLY**. Code must compile and run on asti.idi.ntnu.no.

Part 1, Theory

Problem 1

Try logging in to asti.idi.ntnu.no (use the guide on It's learning). What version of gcc, flex and bison is installed on the server? (Use the `--version` argument).

Problem 2

What is the difference between an acceptor and a lexical analyzer?

Problem 3

a) Construct a nondeterministic finite automaton for the language `1(0|1)*00(0|1)*`

b) Describe what kind of words the language includes (with examples).

c) Create a transition table for the language, based on your automaton.

Problem 4

From "The C Programming Language" (Kernighan & Ritchie), p. 194:

> *A floating constant consists of an integer part, a decimal point, a fraction part, an e or E, an optionally signed integer exponent, and an optional type suffix, one of f , F, l or L. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing. The type is determined by the suffix; F or f make it float, L or l makes it long double; otherwise it is double.*

Draw a deterministic finite automaton which accepts floating point constants in this format.

Part 2, Code

Problem 1, RPN

This problem is about making a reverse Polish notation (RPN) calculator. For more information about RPN, see `http://en.wikipedia.org/wiki/Reverse_Polish_notation`.

The supplied code archive *ps1.tar.gz* contains a program, `rpn`, which should read an expression in reverse Polish notation from standard input, and write the result to standard output.

The program works by using a stack. When a number is encountered, it is pushed on the stack. When a operator (e.g. '+') is encountered, the two topmost elements of the stack are popped and their result is pushed. When the entire expression has been processed, the result will be the last (and only) element on the stack.

Complete the program by implementing the following functions, in the file *rpn.c*:

- `newRpnCalc()` which should return an initialized RpnCalc struct.
- `push(RpnCalc* r, double n)` which should push `n` on the stack of `r`. The function should expand the size of the stack if it is full.
- `performOp(RpnCalc* r, char op)` which should remove the two topmost elements from the stack, and push the result on the stack. In more detail, denoting the topmost element S and the second topmost T, T `op` S should be pushed.
- `peek(RpnCalc* r)` which should return the topmost element of the stack of `r`. It should return 0 if the stack is empty.

You should only make changes to these functions, not other parts of the provided code. You may assume that the input will always be well formed. The only operators used will be '+', '-', '/' and '*'. The files *10.txt*, *100.txt* and *1000.txt* provide examples of input, and evaluate to 10, 100 and 1000 respectively. You can use the makefile target *test* to test your program with those files (i.e. `make test`).