Problem set 1, Topic

TDT4205, Spring 2014

Deadline: dd.mm.yyyy at hh.mm 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 your modified versions of the files:
  - *ps1.c*

Cooperation: This assignment should be done individually.

General notes: Code must compile and run on `login.stud.ntnu.no`. You should only make changes to the files indicated. Do not add additional files or third party code/libraries.

Additional information and details about the problems can be found in the recitation slides for this problem set.

Part 1, Theory

Problem 1

Try logging in to `login.stud.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 interpreter and compiler?

Problem 3

a) Construct a nondeterministic finite automaton for the language `10*10(0|1)*0`

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, Arrays and sorting

Implement the following functions in *ps1.c*.

a) `create_random_array(int size, int n)` The function should return a dynamically allocated array of size `size`, and fill it with random values between 0 and `n`.

b) `print_array(int* array, int size)` The function should print the elements of the array `array`, which is of length `size`.

c) `sort(int* array, int size)` The function should sort the numbers in the array `array`, which is of length `size` in increasing order, in-place. Any suitable sorting algorithm can be used. Do not use `qsort()` from `stdlib.h`.

Problem 2, Search trees

Implement the following functions in *ps1.c*.

a) `create_blank_node()` The function should return a dynamically allocated `Node`. All fields should be set to `NULL` or 0.

b) `insert_node(Node** root, Node* node)` Should insert the node `node` into the tree rooted at the node pointed to by `root`. If the root is `NULL`, `node` should be the root.

c) `create_tree(int* array, int size)` The function should create a tree of all the numbers in the array `array`, which is of length `size`. This should be done by repeatedly calling `insert_node()`, with a node for each number in the array.

d) `print_tree(Node* root, int offset)` The function should print all the values of the nodes of the tree rooted at the node `root`. Each node should be printed on a separate line. A node should be printed before its children, and all its left child should be printed before its right child. Each node should have one more space printed in front of it than its parent. There should be an `offset` number of spaces before the root. An example of a tree, and how it should be printed, is shown in figure 1.

e) `search(Node* root, int n)` The function should search for the number `n` in the tree rooted at `root`. If the value is present in the tree, the function should return 1, otherwise it should return 0.

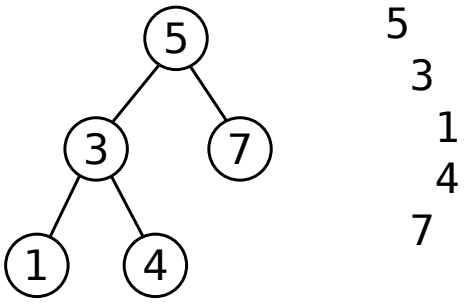

Figure 1: Example tree with output from `print_tree()`

Problem 3, Integration

Implement the following functions in *ps1.c*.

a) `integrate(double (*function)(double), double start, double end, double stepsize)` The function should compute the definite integral of the function `function`, on the interval from `start` to `end`, using the rectangle method, with a rectangle width of `stepsize`.