Assignment 4: Balanced trees and set operations

xxx

Deadline: xxx

Objectives

You must implement a tree and set API and a multi-tool set manipulation program.

Requirements

You will construct the following programs that demonstrate your tree and set API:

`ident` Prints out the set formed by its input file unchanged.

`union` Computes the set union between two input files.

`inter` Computes the set intersection between two input files.

`substract` Computes the set difference between two input files.

All these programs operate on the set of unique input lines of their respective operands and prints out their output in sorted order. For example, `ident` forms the set of unique input lines from its input and prints them back out in sorted order.

Your implementations must achieve this by loading the input lines as nodes in an AVL tree[1], perform computation on the tree, and print out the results using an in-order traversal of the result tree.

You must submit your work as a tarball[2]. Next to the source code, your archive must contain a text file file named "`AUTHORS`" containing your name and Student ID(s).

Additional program

For a maximum grade you must also provide an additional program `calc` which can compute arbitrary algebraic set expressions:

- `calc` takes an expression as command-line argument;

[1] https://en.wikipedia.org/wiki/AVL_tree

[2] http://lmgtfy.com/?q=how+to+make+a+tarball

- the expression is expressed in reverse polish notation, and can contain the following elements separated by spaces:
  - a word or number, which defines a set containing only that number as single element;
  - the character `.`, which loads the empty set;
  - the character `+`, that computes the union of the last two sets;
  - the character `-`, that computes the set difference of the last two sets;
  - the character `*`, that computes the intersection of the last two sets.
- the resulting set is printed on standard output.

For example:

```
$ ./calc "a b +"
a
b

$ ./calc "a a + b + b c + *"
b
```

Getting started

1. Unpack the provided source code archive; then run `make`.
2. Try out the generated `ident` program and familiarize yourself with its interface.
3. Read the file `tree.h` and understand the interface.
4. Implement the data structure in `tree.c`.
5. Implement the set union and intersection operator, then test the resulting `union` and `intersect` programs.
6. Complete `substract`. Use tree node deletion to achieve this.
7. Complete `calc`.

Grading

Your grade starts from 0, and the following tests determine your grade:

- +0,5pt if you have submitted an archive in the right format with an `AUTHORS` file.
- +0,5pt if your source code builds without errors and you have modified `tree.c` in any way.
- +2pt if your tree API processes insertions and balancing properly.
- +1pt if your `ident` program works properly.
- +1pt if your `inter` program works properly.

- +1pt if your `union` program works properly.
- +1pt if your `substract` program works properly.
- +3pt if your `calc` program works properly.
- -0,5pt if your programs misbehave on zero-sized inputs.
- -0,5pt if your programs misbehave when the last line does not terminate with a newline character.
- -1pt if `valgrind` reports errors while running your program.
- -1pt if `clang -W -Wall` reports warnings when compiling your code.