Programm- & Systemverifikation

Assignment 1

Georg Weissenbacher

184.741

Part 1 of Assignment 1

- Your task: Implement a **balanced (AVL) binary search tree**.

Part 1 of Assignment 1

- Your task: Implement a **balanced (AVL) binary search tree**.
- The Internet is an endless source for solutions for programming assignments
  - It's on the interwebs, it must be true!

Part 1 of Assignment 1

- Your task: Implement a **balanced (AVL) binary search tree**.
- The Internet is an endless source for solutions for programming assignments
  - It's on the interwebs, it must be correct!

Part 1 of Assignment 1

- Your task: Implement a **balanced (AVL) binary search tree**.
- The Internet is an endless source for solutions for programming assignments
  - It's on the interwebs, it must be correct!
- To save you some effort, we've already downloaded a solution from

`http://www.refcode.net/2013/02/balanced-avl-binary-search-trees.html`

Implementation details...

```
/* recursive tree structure                    */
typedef struct _tree
{
  struct _tree * left;
  struct _tree * right;
  int element;
  int height;
} Tree;
```

The implementation provides the following functions:

- `insert(e, t)`: Insert element `e` into the tree `t`
  - Returns a pointer to a modified tree
  - Duplicate elements are ignored
- `delete(e, t)`: Remove element `e` from the tree `t`
  - Returns a pointer to the modified tree
  - Non-existent elements are ignored
- `find(e, t)`: Find element `e` in the tree `t`
  - Returns a pointer to the respective sub-tree (`NULL` on failure)

A test case

The web-page also provides a test-case to demonstrate that the implementation works:

- Insert 20, 5, 15, 9, 13, 2, 6, 12, 14, 15, 16, 17, 18, 19
- Delete 14, 13, 5, 10, 15, 16, 19, 18, 20
- What's left: 2, 6, 9, 12, 17

A test case

The web-page also provides a test-case to demonstrate that the implementation works:

- Insert 20, 5, 15, 9, 13, 2, 6, 12, 14, 15, 16, 17, 18, 19
- Delete 14, 13, 5, 10, 15, 16, 19, 18, 20
- What's left: 2, 6, 9, 12, 17

We wrote a test harness for you.

- That's how nice we are.
- You can find the source code on TISS.

The test case above succeeds.

Testing the implementation

- Devise a test scenario with
  - at most 5 insertion operations and
  - no deletions

  such that `find (e, t)` fails even though the element `e` was inserted into `t`.

Testing the implementation

- Devise a test scenario with
  - at most 5 insertion operations and
  - no deletions

  such that `find (e, t)` fails even though the element `e` was inserted into `t`.
  Use the format

  $$\{\texttt{'i'}, e_1\}, \{\texttt{'i'}, e_2\}, \ldots, \{\texttt{'f'}, e_i\}$$

  to specify your scenario, where $\{\texttt{'i'}, e_1\}$ denotes the insertion of element e_1 and `'f'` invokes a find operation (c.f. the source file).

Testing the implementation

- Devise a test scenario with
  - at most 5 insertion operations and
  - no deletions

  such that `find (e, t)` fails even though the element `e` was inserted into `t`.
  Use the format

  $$\{\texttt{'i'}, e_1\}, \{\texttt{'i'}, e_2\}, \ldots, \{\texttt{'f'}, e_i\}$$

  to specify your scenario, where $\{\texttt{'i'}, e_1\}$ denotes the insertion of element e_1 and `'f'` invokes a find operation (c.f. the source file).
- Explain what happens when you call `free_tree(t)` after executing your test scenario.

Adding assertions to `avl.c`

For any node, a balanced tree maintains the following *invariants:*

- The height of the left and right sub-tree differs by at most 1;
- The elements in the left sub-tree are smaller than the elements in the right sub-tree.

Use assertions to add *pre-* and *post-conditions* to the following functions, such that a bug resulting in the violation of these invariants is caught by an assertion:

- `insert`
- `delete`
- `single_rotation_with_left`, `single_rotation_with_right`
- `double_rotation_with_left`, `double_rotation_with_right`

Part 2 of Assignment 1

- Add an *inductive invariant* to the code
- Use it to show that the assertion after the loop holds
- Add comments to the code explaining
  - why your assertion is an inductive invariant
  - why it shows that the assertion after the loop holds

```
int x = i;
int y = j;
while (x != 0)
{
  x--;
  y--;
  assert (?);
}
assert ((i != j) || (y == 0));
```

Submitting your solution

- Your solution must be submitted via TUWEL by April 9, 4pm
  - Late submissions will not be accepted
  - We'll make part 2 available soon
- Answer all questions and submit your solution as a PDF
- Make sure the file contains your student ID and your name
- Do not submit the source code file
  - it is provided for your convenience only