Algorithms and Data Structures

EADS 2025Z (LAB-104)

Task #3 (term 2025Z)

Task #3 deadline is Wednesday, January 14th, 2026, 2 pm.

There are several steps to complete:

- Design `avl_tree` class according to a very limited specification below.
- When Implementing your class write unit tests in parallel (try writing the test before actually implementing the behaviour of your tree).
- You should implement your class template in the `avl_tree.hpp` file, and provide tests of your container (i.e. the tree) in `avl_tree_test.hpp` and `main.cpp` files.
- Implement one external function template: `maxinfo_selector` (neither a method nor a friend in the `avl_tree` class).
- Implement one external function: `count_words` (neither a method nor a friend in the `avl_tree` class).
- Be prepared to modify fragments of your solution or write additional function/template during the lab class on January 14th.

1 PART1 - CLASS DESIGN

Design a class to represent AVL tree. Write unit tests for the designed class, at least one test per method.

```
template <typename Key, typename Info>
class avl_tree
{
public:
	avl_tree();
	avl_tree (const avl_tree& src);
	~avl_tree();
	avl_tree& operator=(const avl_tree& src);

	// insert (key, info) pair

	// remove the element by given key

	// check if given key is present

	// print nicely formatted tree structure as diagram resembling the tree

	Info& operator[](const Key& key); // indexing operator permitting update
	const Info& operator[](const Key& key) const; // indexing operator

	// define and implement, what else can be useful in such a container?
};
```

2 PART 2 – ADDITIONAL FUNCTIONS

Implement one additional function template and one additional function (neither methods nor friends in the avl_tree class):

Function #1
– The function template maxinfo_selector selects a couple of the highest value elements from the avl_tree with respect to Info (Info must have comparison operators defined)

```
template <typename Key, typename Info>
std::vector<std::pair<Key, Info>>
      maxinfo_selector(const avl_tree<Key, Info>& tree, unsigned cnt);
```

`maxinfo_selector` returns *cnt* elements of the *tree* with the highest values of the info member.

Function #2
– The function count_words counts word occurrences in the input stream by collecting them in a specific type of the `avl_tree<string, int>`

```
avl_tree<string, int> count_words(istream& is);
```

`count_words` creates a dictionary, stored in the `avl_tree<string, int>`, mapping words found in the `is` stream to number of occurrences of each word in this stream. You can use Academic_Regulations.txt or any text file published on the Gutenberg Project page - https://www.gutenberg.org/) as a test for your `count_words` function.