CSC3200 - FALL 2025 - ASSIGNMENT 5

DEADLINE: DEC 1ST 23:59

1. Size-Tracking AVL Tree with Iterator(100%)

Please review the notes on AVL trees from lecture 19. For this problem, your objective is to implement a templated AVL tree that, in addition to heights also keeps track of the sizes of subtrees.

```
namespace student_std {
template <typename Key, typename Comp = std::less<Key>>
class avl_tree {
    class avl_node {
    public:
        using size_type = std::size_t;
        Key const& value() const;                          //O(1)
        avl_node const* parent() const;                    //O(1)
        avl_node const* left() const;                      //O(1)
        avl_node const* right() const;                     //O(1)
        // Must return the number of nodes in the subtree
        // that has this node as root.
        size_type size() const { return m_size; }          //O(1)
        std::ptrdiff_t height() const { return m_height; } //O(1)
    private:
        size_type m_size;   // Must be maintained.
        std::ptrdiff_t m_height; // Must be maintained.
        // ...
    };
    class iterator {
    public:
        using value_type = avl_node;
        using reference = value_type const&;
        using pointer = value_type const*;
        using difference_type = std::ptrdiff_t;
        using iterator_category = std::bidirectional_iterator_tag;
        iterator(pointer node = nullptr);
        iterator& operator++();         //O(log n)
        iterator operator++(int);       //O(log n)
        iterator& operator--();         //O(log n)
        iterator operator--(int);       //O(log n)
        reference operator*() const;    //O(1)
        pointer operator->() const;     //O(1)
    private:
        // ...
```

```
    };
public:
    using key_type = Key;
    using node_type = avl_node;
    using size_type = std::size_t;
    using comparison = Comp;
    using const_iterator = iterator;
    using iterator = const_iterator;
    avl_tree();
    iterator insert(Key const&);     //O(log n)
    iterator erase(Key const&);      //O(log n)
    iterator find(Key const&) const; //O(log n)
    bool contains(Key const&) const; //O(log n)
    size_type size() const;          //O(1)
    std::ptrdiff_t height() const;   //O(1)
    iterator begin() const;          //O(log n), returns node with minimal key
    iterator end() const;            //returns iterator(nullptr)
    // returns iterator to root node or end() if empty.
    iterator root() const;           //O(1)
private:
// ...
};
}
```

You may assume that `Key` has an `bool operator<(Key const&)`, is default-constructible and trivially copyable, that `Comp` is trivially default-constructible and has a call operator `bool operator()(Key const&, Key const&) const`. Your datastructure must provide all methods shown above. You may add more private method and fields as implementation details. It is not required to be copyable and must not leak memory. The iterator traversal must be in-order. Your datastructure must suffice the usual height constraints for AVL-trees. It is guaranteed that no duplicate keys will be inserted.

Remark. You may review the example code from lecture 19 as help. Consider carefully how subtree size should be updated during rotations.

Remark. You may use your code from HW4 to get started with the in-order iterator.

Remark. You can include STL headers for the iterator tag, size_t, for std::less, etc. and for other facilities that you may find helpful.

Submission Format

One file: student_avl_tree.h.