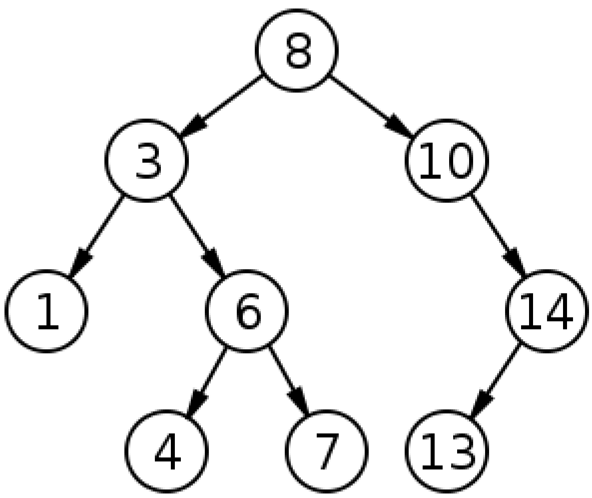
Advanced Programming Exam

The exam consists of a written project followed by an oral discussion. The written project is due to February 19, 2020, at 8:55 AM. Orals take place on 19th and 20th of February.

- Work in groups make of two/three people.
- Use a separate git repository.
- You have to upload all and only the **source files** you wrote, with a **Makefile** and a **readme.md** where you describe how to compile, run your code, and a short report on what you have done, understood, and, eventually, benchmarked.
- Your code must have no memory leaks.
- No warnings must appear if compiled with the flags `-Wall -Wextra`.
- Don't change the name of the functions!
- Once finished, send me either the link to the repository or a `tar.gz` with the files (`.git` folder included).

Binary search tree

The project consists of the implementation of a **template** binary search tree (BST). A BST is a hierarchical (ordered) data structure where each **node** can have at most two children, namely, **left** and **right** child. Each node stores a **pair** of a **key** and the associated **value**. The binary tree is ordered according to the keys. If we assume that we sort the keys in ascending order (i.e., we use the less than `<` operator), then given a node `N`, all the nodes having keys **smaller** than the key of the node `N` are on the **left**. All the nodes with a key **greater** than the key of the node `N` are on the **right**.



Practically speaking, given the binary tree in the picture, if you need to insert a new node with `key=5`, you start from the root node `8`, you go left since `5<8`, you reach node `3`, then you go right, you land in `6`, you go left reaching node `4`. Node `4` has no right child, so the new node `5` becomes the right child of node `4`. If a key is already present in the tree, the associated value **is not** changed.

From the implementation point of view, a node has two pointers `left` and `right` pointing to the left and right child, respectively. The pointers point to `nullptr` if they have no children.

It is useful to add a pointer (head, root, whatever you like) pointing to the top node, called **root node**.

Tree traversal

The tree must be traversed in order, i.e., if I "print" the tree in the picture, I expect to see on the screen the sequence

```
1 3 4 6 7 8 10 13 14
```

node `1` is the first node, and node `14` is the last one.

Assignments

You have to solve the following tasks using modern C++14 (C++17 is welcome as well).

- Implement a **template** binary search tree class, named `bst`.
- The class has three templates:
  - the key type
  - the value type associated with the key
  - the type of the comparison operator, which is used to compare two keys. `std::less<key_type>` should be used as default choice.
- Implement proper iterators for your tree (i.e., `iterator` and `const_iterator`). Forward iterator is sufficient.
- Mark `noexcept` the right functions.
- Remember the KISS principle.
- Implement at least the following public member functions.

Insert

```
std::pair<iterator, bool> insert(const pair_type& x);
std::pair<iterator, bool> insert(pair_type&& x);
```

They are used to insert a new node. The function returns a pair of an iterator (pointing to the node) and a bool. The bool is true if a new node has been allocated, false otherwise (i.e., the key was already present in the tree). `pair_type` can be for example `std::pair<const key_type, value_type>`.

Emplace

```
template< class... Types >
std::pair<iterator,bool> emplace(Types&&... args);
```

Inserts a new element into the container constructed in-place with the given args if there is no element with the key in the container.

Clear

```
void clear();
```

Clear the content of the tree.

Begin

```
iterator begin();
const_iterator begin() const;
const_iterator cbegin() const;
```

Return an iterator to the left-most node (which, likely, is not the root node).

End

```
iterator end();
const_iterator end() const;
const_iterator cend() const;
```

Return an iterator to one-past the last element.

Find

```
iterator find(const key_type& x);
const_iterator find(const key_type& x) const;
```

Find a given key. If the key is present, returns an iterator to the proper node, `end()` otherwise.

Balance

```
void balance();
```

Balance the tree. A simple and naive implementation is fine. The aim of this exam is not producing a super-performant code, but learning c++. No extra points for complicated algorithms.

Subscripting operator

```
value_type& operator[](const key_type& x);
value_type& operator[](key_type&& x);
```

Returns a reference to the value that is mapped to a key equivalent to `x`, performing an insertion if such key does not already exist.

Put-to operator

```
friend
std::ostream& operator<<(std::ostream& os, const bst& x);
```

Implement the friend function **inside** the class, such that you do not have to specify the templates for `bst`.

Copy and move

The copy semantics perform a deep-copy. Move semantics is as usual.

Erase

```
void erase(const key_type& x);
```

Removes the element (if one exists) with the key equivalent to key.

Hints

- **Big hint** use `std::pair<const key_type,value_type>`, which is defined in the header `utility`
- start coding and using the iterators ASAP.