# Advanced Programming - HW3

**Homework 3 - Spring 2022 Semester**
**Deadline: Sunday Farvardin 14st - 11:59 pm**

## Outline

In this homework we are going to implement a *Binary Search Tree (BST)*. A *binary tree* is a tree graph in which nodes can only have upto 2 children. A *Binary Search Tree* is a binary tree in which the right child of each node have a greater value than the left one.

We are going to implement 2 classes one of them is called `Node` which represents each node in the graph, and the other one is `BST` which is responsible to conncet nodes in a  way to construct a binary search tree. Since the `Node` class is a direct property of the `BST` class; It should be defined inside the BST class.

**note.** You are only allowed  to alter `bst.cpp/h` and only the debug section of `main.cpp`.

# Node Class

Use the code fraction bellow to implement this class. **note.** you may need to add some keywords to these functions if necessary. other than these keywords you are not allowed to change the functions or add new member functions to this class **unless otherwise specified in the following**.

```
class Node
{
public:
    Node(int value, Node* left, Node* right);
    Node();
    Node(const Node& node);

    int value;
    Node* left;
    Node* right;
};
```

---

Each node has a integer value and can point to its children using left and right pointers. If one or more children does not exist assign `nullptr` to them.

- **std::cout**
  Implement the `<<` operator so that you would be able to use `std::cout` to print a node in one line and in a beautiful way, the output should contain the node's address, the node's value, node's left child address, and node's right child address.
- **inequality with an int**
  implement proper functions so that every possible inequality between a node and an int can be made i.e.

```
node > 4    node >= 5    node < 6    node <= 5    node == 3
5 < node    6 <= node    2 > node    2 >= node    3 == node
```

# BST Class

Use the code fraction bellow to implement this class. **note.** you may need to add some keywords to these functions if necessary. other than these keywords you are not allowed to change the functions or add new member functions to this class **unless otherwise specified in the following**.

```
class BST
{
public:
    Node*& get_root();
    void bfs(std::function<void(Node*& node)> func);
    size_t length();
    bool add_node(int value);
    Node** find_node(int value);
    Node** find_parrent(int value);
    Node** find_successor(int value);
    bool delete_node(int value);

private:
    Node* root;
};
```

---

This class is responsible to construct a BST graph by connceting the nodes in a proper manner. `root` is the only member variable of this class which points to the first node of the binary search tree.

- **get_root**
  returnts the private member variable `root`.
- **bfs**
  *Breadth-first search (BFS)* is an algorithm for searching a tree by iterating through each and every node; Implement this algorithm!
  This function has a input of `std::function`, this type behaves like a pointer to function and can wrapp any callable objects in c++ like ordinary or lambda functions. you are going to use lambda functions in this homework so feel free to search and learn about them.
  The user should be able to use this argument and apply any functions on the nodes of the tree i.e. call the object `func` on every node found by BFS algorithm.
- **length**
  returnts the number of nodes exist in bst.
- **add_node**
  adds a new node to BST in the proper place. **note.** the user should not be able to add a value that already exists in the tree.

- **find_node**
  finds the node in tree with the specified value and returns a pointer to the pointer of the node.
  **note.** do not use bfs algorithm to find the nod, instead use the property of binary search trees: *that is why we use them in the first place*.
- **find_parrent**
  finds the parrent node of a node with specified value and returns a pointer to the pointer of that parrent.
  **note.** do not use bfs algorithm to find the node.
- **find_successor**
  finds a successor of a node with specified value.
- **delete_node**
  deletes a node with the specified value. if there is no such node with that value return false.
- **std::cout**
  Implement the `<<` operator so that you would be able to use `std::cout` to print a BST in a beautiful way. to do so print each node of the BST. You'll receive extra points if you manage to generate a output exactly as bellow (stars are part of the output).

```
********************************************************************************
0x188dee70      => value:25       left:0x188dedd0      right:0x188dedf0
0x188dedd0      => value:10       left:0x188dee30      right:0x188ded90
0x188dedf0      => value:50       left:0               right:0x188dee50
0x188dee30      => value:7        left:0               right:0
0x188ded90      => value:15       left:0               right:0
0x188dee50      => value:53       left:0               right:0
binary search tree size: 6
********************************************************************************
```

- **operator++**
  Implement both `++` operators so that they would add 1 to each and every nodes in the tree.
  do not forget the difference between `bst++` and `++bst`.
- **destructor**
  As you now, when dealing with dynamic objects in a class creating destructors are a neccessaty. Fortunately for you I will burden the implementation of your destructor myself, so implement the bellow code as your class destructor.

```
BST::~BST()
{
    std::vector<Node*> nodes;
    bfs([&nodes](BST::Node*& node){nodes.push_back(node);});
    for(auto& node: nodes)
        delete node;
}
```

- **Constructors**
  Implement 3 constructors for the class: **I)** defualt constructor. **II)** copy constructor. **III)** move constructor.
- **operator=**
  Implement both versions of `operator=` (copy and move versions).

# Challenge

- If you reached this section congratulations, there is only one part left and its an easy one. Make arrangements so you can add any number of nodes using the constructor. meaning the bellow code should work.

```cpp
BST bst1{5, 1, 10, 2, 8, 50, 4, 60};
BST bst2{3, 2, 100, 20, 8, 50, 4, 60, 44, 23};
```

# Finally

As mentioned before, do not alter other files already populated except otherwise indicated. In case you want to test your code you may use the `debug` section of the `main.cpp`.

```cpp
if (true) // make false to run unit tests
{
    // debug section
}
else
{
    ::testing::InitGoogleTest(&argc, argv);
    std::cout << "RUNNING TESTS ..." << std::endl;
    int ret{RUN_ALL_TESTS()};
    if (!ret)
        std::cout << "<<<SUCCESS>>>" << std::endl;
    else
      std::cout << "FAILED" << std::endl;
}
return 0;
```

**GOOD LUCK**