HW9 실습: Binary Search Tree
(과제 아닙니다, 제출하지 않습니다.)

질문
pemds81718@gmail.com

교재를 참고하여, Insert, Get, Delete 를 구현하기.

1. 테스트 케이스

// 첫줄에 명령의 개수, 그 다음 줄부터는 (명령, Key, Element)

<input.in>

```
11
Insert 75 May
Insert 83 Jun
Insert 61 Sep
Insert 82 Oct
Insert 46 Dec
Insert 77 Aug
Insert 92 Nov
Insert 57 Feb
Get 92
Delete 75
Get 75
```

2. BSTNode.h

```
#ifndef __BSTNODE_H__
#define __BSTNODE_H__
#include <memory>

template <typename T>
class BSTNode {
 private:
  T item;
  std::shared_ptr<BSTNode<T>> leftChildPtr;
  std::shared_ptr<BSTNode<T>> rightChildPtr;

 public:
  BSTNode();
  BSTNode(const T& anItem);
  BSTNode(const T& anItem, std::shared_ptr<BSTNode<T>> leftPtr,
          std::shared_ptr<BSTNode<T>> rightPtr);

  void setItem(const T& nodeItem);
  const T& getItem() const;
  T& getItem();
  T* getItemPtr();

  void setLeftChildPtr(std::shared_ptr<BSTNode<T>> leftPtr);
  void setRightChildPtr(std::shared_ptr<BSTNode<T>> rightPtr);

  std::shared_ptr<BSTNode<T>> getLeftChildPtr() const;
  std::shared_ptr<BSTNode<T>> getRightChildPtr() const;
};  // end BSTNode

#include "BSTNode.cpp"
#endif  // __BSTNODE_H__
```

3. BinarySearchTree.h

```h
#ifndef __BINARY_SEARCH_TREE_H__
#define __BINARY_SEARCH_TREE_H__
#include <memory>
#include <ostream>
#include <string>
#include <utility>

#include "BSTNode.h"

template <typename K, typename E>
class BinarySearchTree {
  using ValueType = std::pair<K, E>;
  using TreePtr = std::shared_ptr<BSTNode<ValueType>>;
 private:
  TreePtr rootPtr;

  ValueType* recursiveGet(TreePtr node, const K& key);
  TreePtr remove(TreePtr node, const K& key);
  TreePtr getSuccessor(TreePtr node);
  TreePtr insert(TreePtr node, const ValueType& entry);
  TreePtr selectByRank(TreePtr node, int& rank);
  void printSubtree(std::ostream& os, TreePtr node, const std::string& prefix,
                    bool isLeft) const;

 public:
  BinarySearchTree();

  bool isEmpty() const;
  ValueType* get(const K& key);
  void remove(const K& key);
  void insert(const ValueType& entry);
  // rank 번째로 작은 원소 찾기!
  ValueType* rankGet(int rank);
  void printTree(std::ostream& os) const;
};  // end BinarySearchTree

template <typename K, typename E>
std::ostream& operator<<(std::ostream& os, BinarySearchTree<K, E>& tree) {
  tree.printTree(os);
  return os;
}
#include "BinarySearchTree.cpp"
#endif  // __BINARY_SEARCH_TREE_H__
```

4. hw9.cpp

```cpp
#include <iostream>
#include <string>
#include <utility>

#include "BinarySearchTree.h"

int main() {
  BinarySearchTree<int, std::string> bst;
  int numCommands = 0;
  if (!(std::cin >> numCommands)) {
    return 0;
  }

  for (int i = 0; i < numCommands; ++i) {
    std::string op;
    std::cin >> op;

    if (op == "Insert") {
      int key;
      std::string value;
      std::cin >> key >> value;
      bst.insert(std::make_pair(key, value));
    } else if (op == "Get") {
      int key;
      std::cin >> key;
      auto result = bst.get(key);
      std::cout << "== Find " << key << " ==" << std::endl;
      if (result) {
        std::cout << "(" << result->first << ", " << result->second << ")"
                  << std::endl;
      } else {
        std::cout << "Not Exist!!" << std::endl;
      }
    } else if (op == "Delete") {
      int key;
      std::cin >> key;
      std::cout << "== Before Delete ==" << std::endl;
      std::cout << bst;
      bst.remove(key);
      std::cout << "== After Delete ==" << std::endl;
      std::cout << bst;
    }
  }
```

```
  std::cout << "== Result ==" << std::endl;
  std::cout << bst;
  return 0;
}
```

5. BinarySearchTree.cpp(출력부분)

```cpp
template <typename K, typename E>
void BinarySearchTree<K, E>::printTree(std::ostream& os) const {
  if (!rootPtr) {
    os << "(empty)" << std::endl;
    return;
  }

  const auto& rootItem = rootPtr->getItem();
  os << rootItem.first << ": " << rootItem.second << std::endl;
  printSubtree(os, rootPtr->getLeftChildPtr(), "", true);
  printSubtree(os, rootPtr->getRightChildPtr(), "", false);
}

template <typename K, typename E>
void BinarySearchTree<K, E>::printSubtree(std::ostream& os, TreePtr node,
                                          const std::string& prefix,
                                          bool isLeft) const {
  if (!node) {
    return;
  }

  os << prefix << (isLeft ? "├── " : "└── ") << node->getItem().first << ": "
     << node->getItem().second << std::endl;

  const auto childPrefix = prefix + (isLeft ? "│   " : "    ");
  printSubtree(os, node->getLeftChildPtr(), childPrefix, true);
  printSubtree(os, node->getRightChildPtr(), childPrefix, false);
}
```

6. 결과

실행: Visual Studio 또는 Make 를 통해 컴파일 후...

./실행파일 < input.in

```
== Find 92 ==
(92, Nov)
== Before Delete ==
75: May
├── 61: Sep
│   ├── 46: Dec
│   │   └── 57: Feb
└── 83: Jun
    ├── 82: Oct
    │   ├── 77: Aug
    │   └── 92: Nov
== After Delete ==
77: Aug
├── 61: Sep
│   ├── 46: Dec
│   │   └── 57: Feb
└── 83: Jun
    ├── 82: Oct
    └── 92: Nov
== Find 75 ==
Not Exist!!
== Result ==
77: Aug
├── 61: Sep
│   ├── 46: Dec
│   │   └── 57: Feb
└── 83: Jun
    ├── 82: Oct
    └── 92: Nov
```