EEE2020-03 **Spring 2021**

Assignment 6: Splay Tree

Due: Sunday, June 13, 2021, 11:59PM

1 Introduction

- The objective of this assignment is to implement a splay tree ADT.
- In this assignment, you will have to complete the missing parts of `class tree_t` and `class splay_t` based on the lecture slides of `5_trees.pdf`.
- In Ubuntu or Mac terminal, use `wget` to download a copy of skeleton code compressed into `splay.tar`.

```
$ wget https://icsl.yonsei.ac.kr/wp-content/uploads/splay.tar
```

- Decompress the `tar` file, and go to the `splay/` directory. Typing `ls` should show the following list of files.

```
$ tar xf splay.tar
$ cd splay/
$ ls
data.h  input.cc  main.cc   node.h    splay.h  tree.cc
input   input.h   Makefile  splay.cc  tar.sh   tree.h
```

- From the listed files, you have to work on four files, `tree.h`, `tree.cc`, `splay.h`, and `splay.cc`.
- You are allowed to change other files for your own testing and validation, but they will revert to the original state when your assignment is graded.
  - `tree.*` define a base class of binary tree, `class tree_t`.
  - `splay.*` define a derived class of splay tree, `class splay_t`.
  - `node.h` has the definition of tree node, `class node_t`.
  - `input.*` files define `class input_t` that handles file operations. It loads a file named `input` containing 100 unsigned integer numbers. The numbers are used in `main()` to test the operations of splay tree.
  - `main.cc` has the `main()` function.
- To compile the code, type `make` in the terminal. `Makefile` that comes along with the skeleton code has automated all the compiling scripts for you.

```
$ make
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -Wall -Werror -g -o splay.o -c splay.cc
g++ -Wall -Werror -g -o input.o -c input.cc
g++ -Wall -Werror -g -o tree.o -c tree.cc
g++ -o splay main.o splay.o input.o tree.o
```

- Executing the skeleton code prints the following output.

```
$ ./splay
height() = 0
size() = 0

height() = 0
size() = 0
```

```
height() = 0
size() = 0
```

2 Implementation

- `node.h` is identical to previous assignment of binary search tree. A node contains two pointers to `left` and `right` children nodes, and a `data` variable.
- The `node.h` file is complete, and you do not have to touch anything in this file.
- `tree.h` and `tree.cc` defining the base class, `tree_t`, are nearly the same as those in the previous assignment.
- The only difference is that the base class does not implement the `contains()` function, since this function incurs splaying unlike the binary search tree or AVL tree.
- Note that the `contains()` function in `tree_t` is defined as a pure virtual function such as `virtual bool contains(const data_t &m_data) = 0;`
- Except the `insert()`, `remove()`, and `contains()` functions, the splay tree reuses all other functions of the base class. Assume that the splay tree performs splaying operations only for these three functions.
- All other base-class functions do not incur splaying operations, i.e., `size()`, `height()`, `clear()`, `find_min()`, `find_max()`, and `print()`.
- You can simply copy the base-class functions from the previous assignment. If you did not complete the previous homework, you will have to work on the base class functions first.
- The following shows the definition of splay tree in `splay.h`.
- The class definition in the skeleton code includes only public functions. You will probably have to define splay and rotate functions in the `private` section of `splay_t` and write their implementations in the `splay.cc` file.

```
/* splay.h */

#ifndef __SPLAY_H__
#define __SPLAY_H__

#include "tree.h"

// Splay tree based on the binary tree base class
class splay_t : public tree_t {
public:
    splay_t(void);
    ~splay_t(void);

    // Return true if the tree contains the data.
    bool contains(const data_t &m_data);
    // Add a new element to the tree.
    void insert(const data_t &m_data);
    // Remove the element in the tree if it has one.
    void remove(const data_t &m_data);

/************************
 * EEE2020: Assignment 6 *
 ************************/
```

```
/*********************
 * End of Assignment *
 *********************/
};

#endif
```

- The next shows the contents of `splay.cc`. The skeleton code only has the class constructor and destructor of `splay_t`. All others are for the assignment.

```
/* splay.cc */

#include <cstdlib>
#include <new>
#include "splay.h"

/************************
 * EEE2020: Assignment 6 *
 ************************/
// Return true if the tree contains the data.
bool splay_t::contains(const data_t &m_data) {
    /* Assignment */
    return false;       // Correct this.
}

// Add a new element to the tree.
void splay_t::insert(const data_t &m_data) {
    /* Assignment */
}

// Remove the element in the tree if it has one.
void splay_t::remove(const data_t &m_data) {
    /* Assignment */
}

/*********************
 * End of Assignment *
 *********************/




// Constructor
splay_t::splay_t(void) { /* Nothing to do */ }

// Destructor
splay_t::~splay_t(void) { /* Nothing to do */ }
```

- The next code box shows the `main()` function. It is arranged in a similar way as the previous assignment.
- Your are allowed to change `main()` to further test and validate your splay tree implementation between the lines of `* TEST YOUR CODE HERE FOR VALIDATION *` and `* END OF TESTING *`. However, this file will revert to the original state when your assignment is graded.

```
/* main.cc */

#include <iostream>
```

```
#include "input.h"
#include "splay.h"

using namespace std;

// Print the information of splay tree.
void print(splay_t &splay) {
    cout << "height() = "       << splay.height()   << endl;
    cout << "size() = "         << splay.size()     << endl;
    if(splay.size()) {
        cout << "find_min() = " << splay.find_min() << endl;
        cout << "find_max() = " << splay.find_max() << endl;
    }
#ifdef DEBUG
    cout << "nodes: "           << endl; splay.print();
#endif
    cout << endl;
}

int main(int argc, char **argv) {
    // Define a splay tree.
    splay_t splay;

    /************************************
     * TEST YOUR CODE HERE FOR VALIDATION *
     ************************************/
    splay.insert(1);
    splay.insert(3);
    splay.insert(3);
    splay.insert(5);
    splay.insert(7);
    splay.insert(9);
    print(splay);

    splay.contains(1);
    splay.contains(9);
    splay.contains(3);
    splay.contains(1);
    splay.contains(9);
    print(splay);

    splay.remove(7);
    print(splay);
    /******************
     * END OF TESTING *
     ******************/







    /* ************************************************
     * WARNING: DO NOT MODIFY THE CODE BELOW THIS LINE *
     ************************************************/

    // Proceed?
    if(argc != 2) { return 0; }
```

```
    // Empty the splay tree.
    cout << "-----------------" << endl
         << "Test #1: clear()" << endl
         << "-----------------" << endl;
    splay.clear();
    print(splay);


    // Add elements read from the input file to the splay tree.
    cout << "------------------" << endl
         << "Test #2: insert()" << endl
         << "------------------" << endl;
    input_t input(argv[1]);
    for(size_t i = 0; i < input.size(); i++) { splay.insert(input[i]); }
    print(splay);


    // Check if certain elements are contained in the splay tree.
    cout << "--------------------" << endl
         << "Test #3: contains()" << endl
         << "--------------------" << endl;
    data_t d = splay.find_min();
    cout << "contains(" << (d = splay.find_min()) << ") = "
         << (splay.contains(d) ? "yes" : "no")    << endl;
    cout << "contains(" << (d = splay.find_max()) << ") = "
         << (splay.contains(d) ? "yes" : "no")    << endl;
    cout << "contains(" << (d = 125)              << ") = "
         << (splay.contains(d) ? "yes" : "no")    << endl;
    cout << "contains(" << (d = 375)              << ") = "
         << (splay.contains(d) ? "yes" : "no")    << endl;
    cout << "contains(" << (d = 250)              << ") = "
         << (splay.contains(d) ? "yes" : "no")    << endl;
    cout << endl;
    print(splay);


    // Remove elements in the splay tree.
    cout << "------------------" << endl
         << "Test #4: remove()" << endl
         << "------------------" << endl;
    splay.remove(splay.find_min());
    splay.remove(splay.find_max());
    splay.remove(125);
    splay.remove(375);
    splay.remove(250);
    print(splay);


    // Repeat all functions once again.
    cout << "-----------------------------------" << endl
         << "Test #5: clear(), insert(), remove()" << endl
         << "-----------------------------------" << endl;
    splay.clear();
    for(size_t i = input.size(); i > 0; i--) { splay.insert(input[i-1]); }
    splay.remove(104);
    splay.remove(175);
    splay.remove(456);
    splay.remove(25);
    splay.remove(275);
```

```
    splay.remove(53);
    splay.remove(400);
    print(splay);


    return 0;
}
```

- Your assignment is graded based on what you get from the testing part of `main()`. Results above the line of `* WARNING: DO NOT MODIFY THE CODE BELOW THIS LINE *` will be not considered for grading.
- **Test #1:** The first part of testing clears `splay` by calling `splay.clear()`.
- **Test #2:** The test code reads an input data file (i.e., `input` in the `splay/` directory) that contains 100 unsigned integer numbers between 0 and 500. File handling is already implemented, and you may disregard how it works. Numbers in the file are inserted into the splay tree via repeated calls of `splay.insert()`.
- **Test #3:** It calls a set of `contains()` functions to check if `splay` contains certain values. The `contains()` functions incur splaying operations and reshape the tree.
- **Test #4:** This part of code tests the `remove()` function.
- **Test #5:** Lastly, `clear()`, `insert()`, and `remove()` functions are all tested once again. This time, 100 numbers in the `input` file are inserted in the reverse order, and thus a different tree shape is created.
- If both `class tree_t` and `class splay_t` are correctly implemented, executing the code will produce the following output.

```
$ ./splay
height() = 4
size() = 5
find_min() = 1
find_max() = 9

height() = 3
size() = 5
find_min() = 1
find_max() = 9

height() = 2
size() = 4
find_min() = 1
find_max() = 9
```

- To enable the `print()` function of `tree_t`, you have to clean up the build files and recompile the code by typing `make "OPT=-DDEBUG"`. Note that are double `D`'s in the optional flag.

```
$ make clean
rm -f input.o main.o splay.o tree.o splay

$ make "OPT=-DDEBUG"
g++ -Wall -Werror -g -o input.o -c input.cc -DDEBUG
g++ -Wall -Werror -g -o main.o -c main.cc -DDEBUG
g++ -Wall -Werror -g -o splay.o -c splay.cc -DDEBUG
g++ -Wall -Werror -g -o tree.o -c tree.cc -DDEBUG
g++ -o splay input.o main.o splay.o tree.o

$ ./splay
height() = 4
```

```
size() = 5
find_min() = 1
find_max() = 9
nodes:

...9

      ...7

            ...5

                  ...3
                        ...1

height() = 3
size() = 5
find_min() = 1
find_max() = 9
nodes:

...9

            ...7
                  ...5
      ...3
            ...1

height() = 2
size() = 4
find_min() = 1
find_max() = 9
nodes:
      ...9
...5

      ...3
            ...1
```

- To test the program with the `input` data file, add `input` to the run command as follows.

```
$ make clean
rm -f input.o main.o splay.o tree.o splay

$ make
g++ -Wall -Werror -g -o input.o -c input.cc
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -Wall -Werror -g -o splay.o -c splay.cc
g++ -Wall -Werror -g -o tree.o -c tree.cc
g++ -o splay input.o main.o splay.o tree.o

$ ./splay input
height() = 4
size() = 5
find_min() = 1
find_max() = 9

height() = 3
size() = 5
find_min() = 1
```

```
find_max() = 9

height() = 2
size() = 4
find_min() = 1
find_max() = 9

----------------
Test #1: clear()
----------------
height() = -1
size() = 0

-----------------
Test #2: insert()
-----------------
height() = 14
size() = 100
find_min() = 9
find_max() = 497

-------------------
Test #3: contains()
-------------------
contains(9) = yes
contains(497) = yes
contains(125) = no
contains(375) = no
contains(250) = yes

height() = 12
size() = 100
find_min() = 9
find_max() = 497

-----------------
Test #4: remove()
-----------------
height() = 14
size() = 97
find_min() = 24
find_max() = 493

-----------------------------------
Test #5: clear(), insert(), remove()
-----------------------------------
height() = 11
size() = 95
find_min() = 9
find_max() = 497
```

- Even if your code compiles and runs to completion with correct results, you need to double-check the code with `valgrind` to confirm no memory leaks.

```
$ valgrind ./splay input

...

==2020== All heap blocks were freed -- no leaks are possible
```

```
...
```

- In Mac OS, `valgrind` is not well supported. You may use `leaks` instead to test memory leaks.

```
$ leaks --atExit -- ./splay input

...

Process 2020: 0 leaks for 0 total leaked bytes.

...
```

3 Submission

- When the assignment is done, execute the `tar.sh` script in the `splay/` directory.
- It will compress the `splay/` directory into a `tar` file named after your student ID such as `2020140000.tar`.

```
$ ./tar.sh
rm -f input.o main.o splay.o tree.o splay
a splay
a splay/main.cc
a splay/input.h
a splay/splay.cc
a splay/splay.h
a splay/input
a splay/data.h
a splay/Makefile
a splay/tree.h
a splay/node.h
a splay/input.cc
a splay/tree.cc
a splay/tar.sh

$ ls
2020140000.tar  input     input.h  Makefile  splay.cc  tar.sh   tree.h
data.h          input.cc  main.cc  node.h    splay.h   tree.cc
```

- Upload the tar file (e.g., `2020140000.tar`) on LearnUs. Do not rename the `tar` file or C++ files included in it.

4 Grading Rules

- The following is the general guideline for grading. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and the grader may add a few extra rules for fair evaluation of students' efforts.

  -3 points: A submitted `tar` file is renamed and includes redundant tags such as `hw6`, student name, etc.

  -5 points: A program code does not have sufficient amount of comments. Comments in the skeleton code do not count. You must make an effort to clearly explain what each part of your code intends to do.

  -5 points each: There are total five sets of testing in `main()`, such as `Test #1: clear()`, $\cdots$ For each incorrect block, 5 points will be deducted.

  -15 points: The code compiles and runs, but it has memory leaks or crashes at the end with errors.

  -25 points: The code does not compile or fails to run (e.g., no or completely wrong results), but it shows substantial efforts to complete the assignment.

-30 points: No or late submission. The following cases will also be regarded as no submissions.

* Little to no efforts in the code (e.g., submitting nearly the same version of code as the skeleton code) will be regarded as no submission. Even if the code is incomplete, you must make substantial amount of efforts to earn partial credits.
* Fake codes will be regarded as cheating attempts and thus not graded. Examples of fake codes are i) hard-coding a program to print expected outputs to deceive the grader as if the program is correctly running, ii) copying and pasting random stuff found on the Internet to make the code look as if some efforts are made. More serious penalties may be considered if students abuse the grading rules.

Final grade = F: The submitted code is copied from someone else. All students involved in the incident will be penalized and given F for the final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, feel free to discuss your concerns with the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee2020`
- Begging partial credits for no valid reasons will be treated as a cheating attempt, and such a student will lose all scores of the assignment.