COMP2113 Programming Technologies / ENGG1340 Computer Programming II

Assignment 3

Deadline: 7 December (Thursday) 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 3.

- General Instructions
- Problem 1: C++ STL (50 marks)
- Problem 2: Binary Search Tree in C (50 marks)

Total marks: 100 marks

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 2 tasks and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each task in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Coding environment

For Problem 1 on C++ programming, make sure the following compilation command is used to compile your program:

```
g++ -pedantic-errors -std=c++11 1.cpp –o 1
```

For Problem 2 on C programming, make sure the following compilation command is used to compile your program:

```
gcc -pedantic-errors -std=c11 2.c –o 2
```

Submission

Name your programs as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (*.c & *.cpp) and no other files. **Compress this directory as a [uid].zip file where [uid] is your university number** and check carefully that the correct files have been submitted. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Filename	Description
1.cpp	Problem 1
2.c	Problem 2

Late submission

If you submit within 3 days after the deadline, there will be 50% mark deduction. After that, no mark will be given.

Evaluation

Your code will be <mark>auto-graded for technical correctness</mark>. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a <mark>case-by-case basis</mark> during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for <mark>logical redundancy</mark>. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Getting help

You are not alone! If you find yourself stuck on something, post your questions to the course forum, send us emails or come to our support sessions. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don't know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments. However, you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment you should post them in the discussion forums.

Problem 1: C++ STL [40%]

You are required to develop an application to maintain student exam grades. This application allows you to enter, search information of student and also view all exam marks of the students based on different rules. The main purpose of building this application is to store and view the student examination records. A student structure is defined as follows:

```
struct Student {
    int id;
    string name;
    Student() {
        id = 0;
        name = "invalid";
    };
    Student(int inputid, string inputname) {
        id = inputid;
        name = inputname;
    };
};
```

A table class is defined as follow. Note that class is optional in our course. You are not required to work on the class but just implement some member functions inside.

```
class Table {
public:
    void InsertStudent(Student x, int y);
    void SearchbyID(int x);
    void SearchbyGrade(int y);
    void Statistics();
    void PrintAll();
    // You can add more functions if you want.
private:
    /* The data type is decided by you */ records;
};
```

We would like you to implement a program *1.cpp* that supports 5 commands, namely **InsertStudent**, **PrintAll**, **SearchbyID**, **SearchbyGrade** and **Statistics**. All commands are case-sensitive.

The program source code is shown below.

```cpp
#include<iostream>
#include<string>
#include<algorithm>
#include<map>
#include<vector>
// You can add more libraries here (if needed)

using namespace std;

// Define structure, classes and member functions
struct Student {
    // Code given above
};

class Table {
    // Code given above
};

bool operator<(const Student&a, const Student&b) {
    // This function may be required by your implementation.
}

//insert one record in to the map
void Table::InsertStudent(Student x, int y) {
    // To be implemented
}

//return the name and grade of the student with id x
void Table::SearchbyID(int x) {
    // To be implemented
}

//return the id and name of the student with grade y
void Table::SearchbyGrade(int y) {
    // To be implemented
}

void Table::Statistics() {
    // To be implemented
}

//Print all records in the accending order of id
void Table::PrintAll() {
    // To be implemented
}

int main() {
    Table t;
    string command;
    int id;
```

```
        string name;
        int grade;

        while (cin >> command) {
            if (command == "InsertStudent") {
                cin >> id >> name >> grade;
                Student s = {id, name};
                t.InsertStudent(s, grade);
            }
            else if (command == "SearchbyID") {
                cin >> id;
                t.SearchbyID(id);
            }
            else if (command == "SearchbyGrade") {
                cin >> grade;
                t.SearchbyGrade(grade);
            }
            else if (command == "PrintAll") {
                t.PrintAll();
            }
            else if (command == "Statistics") {
                t.Statistics();
            }
            else if (command == "exit") {
                break;
            }
        }
        return 0;
}
```

1. InserStudent *[id] [name] [grade]* - Insert the record with ID, name and grade into the table. For example, we issue the following commands to insert 6 records into the table.

```
InsertStudent 2023001 Chim 98
InsertStudent 2023002 Jolly 96
InsertStudent 2023003 Kevin 89
InsertStudent 2023004 Ruihua 93
InsertStudent 2023005 Kenneth 89
InsertStudent 2023006 Ben 99
```

- You can assume that the name of the students will not contain any space
- If a student with the ID is already in the graph, output "Student exists.".
- The ID of the students may not start from 0 and may not be in ascending order.

- Decide a suitable STL container to implement the Table::records member variable. We need to access to a student and grade given the student ID in other operations. Some container may greatly simplify your program ☺.

2. PrintAll - Before you proceed to the next part, you may implement PrintAll() to output the name of all records in the table object in the ascending order of the student ID and validate the correctness of your program.

```
PrintAll
2023001 Chim 98
2023002 Jolly 96
2023003 Kevin 89
2023004 Ruihua 93
2023005 Kenneth 89
2023006 Ben 99
```

3. SearchbyID *[id]* - Search for a student with the ID in the table.
   For example, we issue the following command to search for the name and grade in the table.

```
SearchbyID 2023003
Kevin
87
```

- If there's no record in the table with student ID equal to the input, output "No such student" on screen

4. SearchbyGrade *[grade]* - Print the student IDs and names with this grade.

```
SearchbyGrade 89
2022003 Kevin
2022005 Kenneth
```

- If there is no student with this grade, output "No such record.".
- If there are more than one records, output them in ascending order of the student ID, line by line.

5. Statistics - Print the maximum, minimum and median of grades

```
Statistics
Maximum 99
Median 94.5
Minimum 89
```

- The median is the middle of a sorted list of grades.
- If they are even amount of records, you must calculate the average of the middle pair.

Problem 2: Binary Search Tree in C [50%]

A Binary Search Tree has the following characteristics:
- **Binary tree** – Each node in the tree has at most two child nodes (can be one child node or two child nodes).
- **Search tree** – Keys on the left sub-tree must be smaller than the keys on the right sub-tree.
- When searching for a particular key, we do not need to scan through all the items. For example, searching for key 14 in the tree above, we need to do 2 comparisons only.
    - o Compare with the root, since 14 is larger than 8, we go to the right path.
    - o Compare with node 10, since 14 is larger than 10, we go to the right path.
    - o Found key 14.

In this problem, you are provided a template program 2.c. Complete the program so that it can handle the commands below. All commands are case-sensitive.
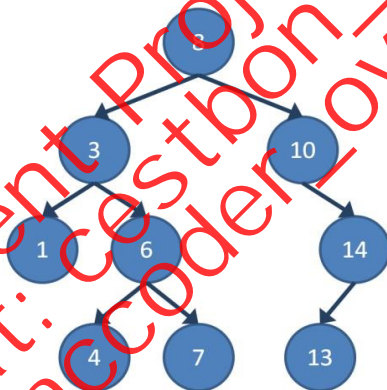
- Insert X
    - o Insert the element X into the tree
- Print
    - o Print all elements in the tree in ascending order
- FindMin
   - o Look for the minimum element in the tree

- FindMax
  - Look for the maximum element in the tree
- Find X
  - Look for the element X in the tree
- End
  - Quit the program

You can assume that X is always an integer. Also you can assume that all elements in the tree are unique (that is no two identical elements).

In the template program, a `treeNode` structure is given. It contains three member variables:
- `int key` – the key of `treeNode`
- `struct treeNode *left` – the pointer to the left subtree
- `struct treeNode *right` – the pointer to the right subtree

struct treeNode
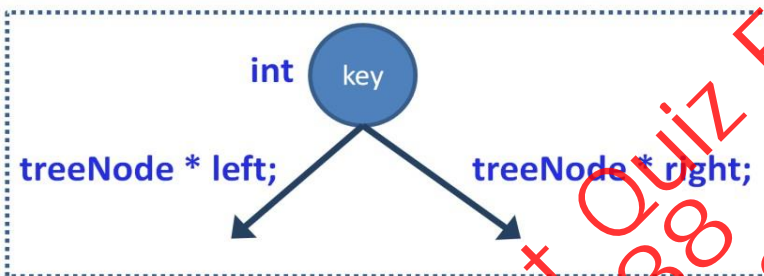
Besides, the `void Print(treeNode *currentNode)` function has been implemented for you. This function can traverse the tree rooted at `currentNode` from left to right and print the nodes in ascending order of the keys.

You should implement the following functions:
- `treeNode * Insert(treeNode * currentNode, int key);`
  - Insert a new node with key `key` into the tree rooted at `currentNode`. The function should return a pointer pointing the root node.
  - Hint: You should consider at least 2 cases: the tree is empty or the tree is non-empty. Also bear in mind that the key in the left child node must be smaller than the key in the right child node.
- `treeNode * FindMin(treeNode *currentNode);`
  - Look for the minimum key in the tree rooted at `currentNode`. The function should return a pointer pointing the minimum node.
- `treeNode * FindMax(treeNode *currentNode);`
  - Look for the maximum key in the tree rooted at `currentNode`. The function should return a pointer pointing to the maximum node.
- `treeNode * Find(treeNode * currentNode, int key);`
  - Look for the key `key` in the tree rooted at `currentNode`. The function should return a pointer pointing to the target node if found and return NULL if not found.

Hint: All these 4 functions can be implemented in *recursive* manner.

Sample Run

Assume we have input2_x.txt in the current directory (available on Moodle). If we run the program like this: (assuming that we are working on Linux)

```
$ gcc -pedantic-errors -std=c11 2.c -o 2
$ ./2 < input2_x.txt
```

The program will print as follows.

Input file: input2_1.txt
Insert 2
Insert 6
Insert 4
Insert 8
Insert 10
Print
End

Output:
2 4 6 8 10

Input file: input2_2.txt
Insert 21
Insert 13
Insert 40
Insert 6963
Insert 2023
Print
FindMin
FindMax
End

Output:
13 21 40 2023 6963
Minimum element is 13
Maximum element is 6963

Input file: `input2_3.txt`
```
Insert 8
Insert 10
Insert 6
Insert 7
Find 7
Find 6
Find 10
Find 5
Find 11
Insert 5
Find 5
Insert 0
Insert 9999
Print
End
```

Output:
```
Element 7 found
Element 6 found
Element 10 found
Element 5 not found
Element 11 not found
Element 5 found
0 5 6 7 8 10 9999
```

Note

- Your program MUST be a C program (NOT a C++ program). Your program will be checked after the submission deadline. Your score will be 0 if we find that your program is not a C program.
- You MUST implement binary search tree (BST) in your program. Your program will be checked after the submission deadline. Your score will be 0 if we find that you use traditional arrays to store keys in your implementation.