Prof. Mo Chen
Simon Fraser University

Assignment 8

CMPT 125
Due Mar. 27

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use search engines to find solutions for the assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment

Submission Instructions:

- You may type or write your answer as long as it is readable.
- Submit two files on CourSys
  1. `report.pdf`, which contains a write-up of your solutions to the assignment
  2. `DetBST.cpp`, which contains the code you wrote in Problem 2.

Problem 1

Answer any two out of three parts of this question (2 marks). For code implementations, just the **typed** snippet is accepted.

Alternatively, you may answer all three questions correctly for 3 marks (i.e. 1 bonus mark); however, if you answer all three questions, you will receive −1 mark for each wrong answer. (This penalty does not apply if you only answer two out of three parts).

a) Consider the following code. Fill the blank with proper class header to match the sample cases.

```
#include<iostream>
using namespace std;
// Write the class header here
------------------------------
class A {
public:
    T x;
    U y;
    A() { cout << "called" << endl; }
    A(T x, U y){ cout << x <<         << y << endl; };
};

int main()  {
    int num1 = 0;
    double num2 = 0;
    char c;
    cin >> num1;
    cin >> num2;
    cin >> c;
    A<char> a;
    A<char, int> (c, num1);
    A<int, double> (num1, num2);
    return 0;
}
```

Example 1

Input: (from `num1`, `num2`, `c`)

```
2
3.4
```

```
n
```

Output:

```
called
n 2
2 3.4
```

Example 2

Input: (from `num1`, `num2`, `c`)

```
5
5.4
i
```

Output:

```
called
i 5
5 5.4
```

Example 3

Input: (from `num1`, `num2`, `c`)

```
3
3.4
p
```

Output:

```
called
p 3
3 3.4
```

b) Consider a binary tree T. Let $|T|$ be the number of nodes in T, x be a node in T, and L_x, R_x are the left/right subtree of x, respectively. We say that x has the "approximately balanced property", $ABP(x)$, if $|R_x| \leq 2|L_x|$ and $|L_x| \leq 2|R_x|$. What is the maximum height of a binary tree T on n nodes where $ABP(root)$ holds? ($root$ is the root note of T.)

c) Complete the following snippet that for a given binary tree and a sum, returns true if there exist a path from root to leaf such that the accumulation of all the values along the path equals the given sum, and return false if no such path can be found.

```
int IsSum(struct node* node, int sum) {
    if (node == NULL) {
        return(sum == 0);
    } else {
        int subSum = sum - node->data;
        // Write the code here
        ------------------------------
    }
}
```

Problem 2

Write two versions of C++ code (named `DetBST.cpp`) that given nodes of a binary tree, detects whether it is a binary search tree or not. (3 marks)

Version 1: Suppose you have helper functions `minValue()` and `maxValue()` that return the minimum or maximum value (you may assume values are of type `int`) from a non-empty tree. Complete the function named `DetBST1()` that returns `true` if a tree is a binary search tree and `false` otherwise by checking every node in tree.

Version 2: Version 1 runs slowly since it traverses over some parts of the tree many times. A better solution looks at each node only once. The trick is to write a utility helper function `BSTUtil()` that traverses down the tree keeping track of the narrowing min and max allowed values as it goes, looking at each node only once. The initial minimum and maximum values should be `INT_MIN` and `INT_MAX` (and narrow from there).

```
int minValue(struct node* node) {
    struct node* current = node;
    while (current->left != NULL) {
        current = current->left;
    }
    return(current->data);
}

int maxValue(struct node* node) {
    struct node* current = node;
    while (current->right != NULL) {
        current = current->right;
    }
    return(current->data);
}

```

```
int DetBST1(struct node* node) {
// write the block of code for Version one here
}

int DetBST2(struct node* node) {
    return(BSTUtil(node, INT_MIN, INT_MAX));
}

int BSTUtil(struct node* node, int min, int max) {
// write the block of code for Version 2 here
}
```