CS214 – Data Structures – 2018

Assignment (3,4) V1.0

[240 Points]

Deadline and Submission:

1. At least one team member should submit the compressed group solution as a zip file containing the programs under acadox –> tasks (name your assignment file "A3_ID1_ID2_ID3_ID4_G#_G#_G#_G#.zip"). e.g. A3_20168383_20163838_20163839_20163840_G1_G1_G1_G1.zip.
2. The deadline for submitting the electronic solution of this assignment is N/A.

About this assignment:

1. This assignment will be solved in teams of 3 except those who got exceptions from the same lab.
2. The weight of the assignment is 240 points
3. All team members should understand all assignment problems.
4. All code must be standard ANSI C++.
5. Assume any missing details and search for information yourself.
6. Any cheating in any part of the assignment is the responsibility of the whole team and whole team will be punished.
7. Your solution should compile, run without run time errors, **leave no memory leaks or dangling pointers**, and handle all cases.

Problems:

Section#1 [120 points]

Problem 1: Linked lists [46 points]

In this problem, you should develop a linked list class similar to that provided in the C++ STL. The public interface of your class should provide basic insertion and deletion functions. In addition, it should provide an iterator class **as an inner class** in order to access the data stored in the list. For example, if we have a list of three elements, and want to access the second element, we should declare an iterator and initialize it to the position of the first element, and move to the second position as shown in the code below:

```
list<int> myList;
myList.push_back(1);
myList.push_back(2);
myList.push_back(3);
list<int>::iterator it = myList.begin();
it++;
cout<< *it;
```

notice the usage of the scope operator in the declaration of the iterator, this is because the iterator class is defined as an inner class inside the list class:

```
template<class type>
class myList {
public:
    class iterator {
        // your code for the iterator class here
    };

    // your code for the list class here
};
```

Your list class should be a template class. [3 points]

The list class should have the following public interface:

- `list()` – default constructor. [2 points]
- `list(type value, int initial_size)` – constructs a list having 'initial_size' elements whose values are 'value'. [3 points]
- `~list()` – a destructor to clear the list and leave no memory leaks. [3 points]
- `int size()` – returns the current number of elements in the list. [2 points]
- `void insert(type value, iterator position)` – adds an element at position specified by the iterator. For example, if the passed iterator currently points to the second element, this element will be shifted on position, and the new value should be added at the second position. [3 points]
- `iterator erase(iterator position)` – erases the element specified by the iterator and return an iterator to the next element, throws exception if position points **after the last element**. [3 points]
- `list<type>& operator = (list<type> another_list)` – overloads the assignment operator to **deep copy** a list into another list and return the current list by reference. [3 points]
- `iterator begin()` – returns an iterator pointing to the first element. [3 points]
- `iterator end()` – returns an iterator pointing **after the last element**. [3 points]

You should develop an iterator class the following public interface:

- `void operator ++ ()` – overloads the operator ++, it should advance the iterator one position towards the end of the list, throws exception if it is currently pointing **after the last element**. [3 points]
- `void operator -- ()` – overloads the operator --, it should move the iterator one position toward the beginning of the list, throws exception if it is currently pointing to the first element of the list. [3 points]
- `type& operator * ()` – overloads the dereference operator to return the value contained in the current node by refence to allow its modification. [3 points]
- `bool operator == (const iterator &)` – overloads the equality comparison operator, should return true if the passed operator points to the same node. [3 points]

All node pointers in the list class of the iterator class should be private and inaccessible from outside of the class. [3 points]

No memory leaks or dangling pointers. [3 points]

It is highly recommended to implement it as a double linked list, however, it is up to you.

As mentioned above, our .end() function should return an iterator pointing to a position **after the last element** as the STL .end() function does. This can be done easily by having a dummy node after the actual tail, this dummy node contains no data, but mark the end of the list. So, physically, our list is never empty, which will ease the implementation of insertion and remove operations, as now we don't have to handle an "empty list" case. However, this dummy node should be disregarded when we return the size.

Write a main function to test all the above.

Problem 2: Stacks [22 Points]

In this problem, you should develop a stack class similar to that provided in the C++ STL. You can use arrays or your linked list class developed in the previous problem as an underlying data structure, however, you cannot use any of the C++ STL classes in this problem.

Your stack class should be template. [3 Points]

The stack class should have the following public interface:

- `stack()` – default constructor. [2 Points}
- `stack(type value, int intial_size)` – constructs a stack having 'initial_size' elements whose values are 'value'. [3 points]
- `~stack()` – a destructor to clear the stack and leave no memory leaks. [3 Points]
- `type& top()` – returns the top element by reference. [3 Points]
- `void pop()` – removes the top element. [3 Points]
- `void push(type value)` – adds an element to the top of the stack. [3 Points]
- `int size()` – returns the number of elements in the stack. [2 Points]

Write a main function to test all the above.

Problem 3: Queues [22 Points]

In this problem, you should develop a queue class similar to that provided in the C++ STL. You can use arrays or your linked list class developed in the previous problem as an underlying data structure, however, you cannot use any of the C++ STL classes in this problem.

Your queue class should be template. [3 Points]

The queue class should have the following public interface:

- `queue()` – default constructor. [2 Points}
- `queue(type value, int intial_size)` – constructs a queue having 'initial_size' elements whose values are 'value'. [3 points]
- `~queue()` – a destructor to clear the queue and leave no memory leaks. [3 Points]
- `type& front()` – returns the first element by reference. [3 Points]
- `void pop()` – removes the first element. [3 Points]
- `void push(type value)` – adds an element to the back of the queue. [3 Points]
- `int size()` – returns the number of elements in the queue. [2 Points]

Write a main function to test all the above.

Problem 4: Using STL list [10 Points]

In this problem, you **must** use C++ STL linked list. Given two sorted linked lists, merge them in the first one without using an extra list, and without keeping any duplicates.

Write a main function to test.

Problem 5: Using STL stack [10 Points]

In this problem, you **must** use C++ STL stack. Given an input string of brackets '(' and ')', square brackets '[' and ']', curly brackets '{' and '}', and multiple line comment token '/*' and '*/', check if this string is valid or not.

A string is considered valid if and only if:

- For each opened bracket, there should be a closing bracket of the same type.
- Each closing bracket should close the lastly opened bracket.

Multiple line comment tokens consist of two characters, and any text between them should be ignored. However, they are treated the same way as brackets regarding the rules described above.

- `([{}])(){}[]{[]}` – valid.
- `({)}` – invalid.
- `({/*)}]]]]]]}*/})` – valid.
- `({/*[][[]]]]]})` – invalid, the comment is not closed.
- `[{/*******/}]` – valid.

Write a main function to test.

Problem 6: Using STL queue [10 Points]

In this problem, you **must** use C++ STL queue. Reimplement part of the stack data structure using **only one queue** as an underlying data structure. Reimplement the class for 'int' data type only and with the following functions only:

- `int top()` – returns the top element. [3 Points]
- `void pop()` – removes the top element. [3 Points]
- `void push(int value)` – adds an element to the top of the stack. [4 Points]

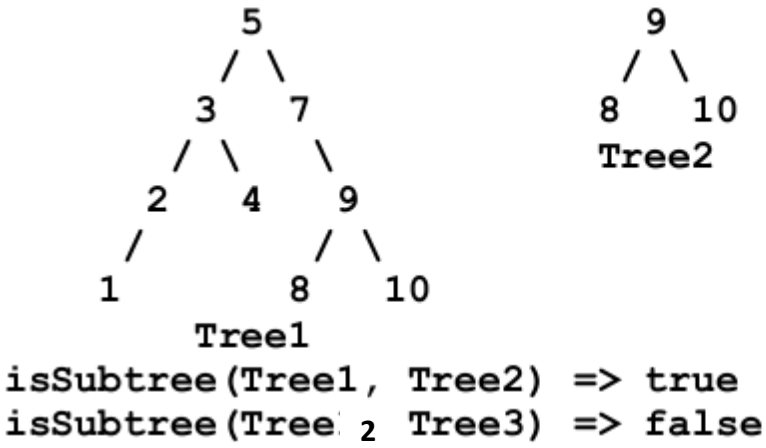
Write a main function to test.

Section#2 (50 points):

Create template binary search tree class with this name BSTFCI and create node class with name BSTNode (10 points)

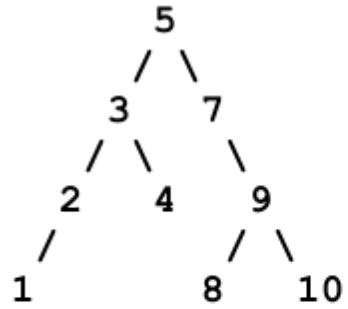
1. **Add Checking Tree Balance (10 points)**
   A Balanced Binary Tree is a tree where the heights of the two child sub-trees of any node differ by at most one AND the left subtree is balanaced AND the right subtree is balanced. Add method called „isBalance" to BSTFCI this method will check in the BST is balance or not.

2. **Tree Comparison (10 points)**
   Write a function that decides if a BSTFCI T2 is a sub-tree of another BSTFCI T1. Prototype: bool isSubTree(BSTFCI* t1, BSTFCI* t2); Note: You may need to write another function that takes 2 BSTNodes and compares their sub-trees.



3. **Print Range (10 points)**

Add a recursive function named printRange in the class BSTFCI that stores integers and given a low key value and a high key value, it prints in sorted order all records whose key values fall between the two given keys. Function printRange should visit as few nodes in the BST as possible. **You should NOT** traverse ALL the tree inorder and print the ones in the range. This will not be considered a correct solution. You should do smart traversal to only traverse the related parts.




```
printRange(3, 6)    => [3,4,5]
printRange(8, 15)   => [8,9,10]
printRange(6,6)     => []
```

4. **Tree Application (10 points)**

● Write an index builder application that takes text consisting of lines and prints a list of the words of the text and the lines they appear on are printed next to them.

● The application works by building a binary search tree using BSTFCI and each node contains a word and a vector of that contains the list of lines where this word exists. For each new word, the program finds it and adds the line number to the vector. If word is not found, it is added to the tree. Then traverse the tree in-order to print the nodes.

● You need to remove punctuation marks like . and , from the text before processing it.

● For example, the text below produces the given index, Test Your code on the given text and 1 more examples.

```
I am for truth,
no matter who tells it.
I am for justice,
no matter who it is for or against.
Malcolm X
```

against	4	matter	4
am	1, 3	no	2, 4
for	1, 3, 4	or	4
I	1, 3	tells	2
is	4	truth	1
it	2, 4	who	2,4
justice	3	X	5
Malcolm	5		

Section#3 (35 points)

1. Expression Tree Evaluation (10 points)

● Write an algorithm that accepts an arithmetic expression written in prefix notation and builds an expression tree for it. And then traverse to evaluate the expression. The evaluation should start after the whole expression has been entered.

● Implement your algorithm as a function.

● Write five test cases for your application and run them.

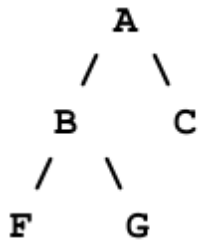
```
expression: + 3 * 4 / 8 2
tree:
          +
         / \
        3   *
           / \
          4   /
             / \
            8   2
Evaluation: 19
```

2-Tree Traversal (15 points)

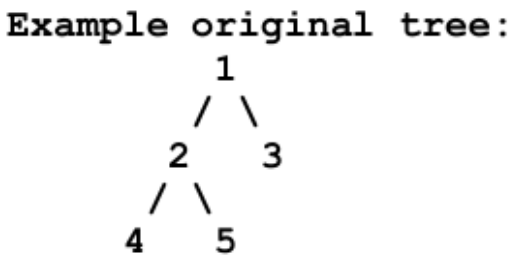
● Assume a binary tree of non-repeated characters. Write a function printPostOrder that takes two strings representing the preorder traversal and the in-order traversal of the tree. Then the function prints the post-order traversal. The function prototype is: void printPostOrder (string preorder, string inorder);

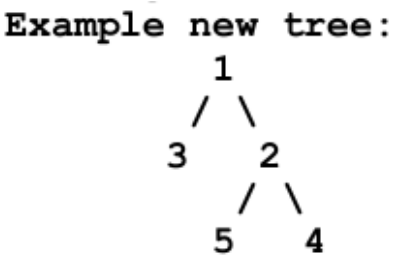
● Write a program that implements five test cases to test your function.

● A sample function call and the corresponding output is shown below:
printPostOrder ("ABFGC", "FBGAC")
=> FGBCA



3- Tree Flipping (10 points) Assume a binary tree. Write a flip method that takes the node which the mirror image of the tree will start from if no parameter send to the function the default value will be the root node.
void flip(Node* node = root)





Section#4: (35 point)

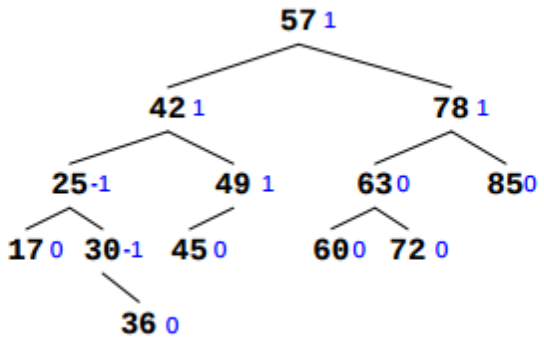
Problem#1 Implement BST AVL tree (23 points)

1.Insert Element into the tree (6 points)
2.Delete Element from tree (14 points)
3.Display Balanced AVL Tree: (5 points)
- InOrder traversal
- PreOrder traversal
- PostOrder traversal

Problem#2 (2 points)

Explain on the BST AVL what will happen when you add element with value 61 Note: for solve this problem you should view what happen in two steps and show the changes in balance factors
Step1: just add the node to the tree (0.5 point)
Step2: make the tree balance(1.5 points)

Balance factors shown next to each node in blue



Problem#3 (8 points)

Apply the following operations on the BST Note: The questions are independent on each other for solving any question you should do two steps

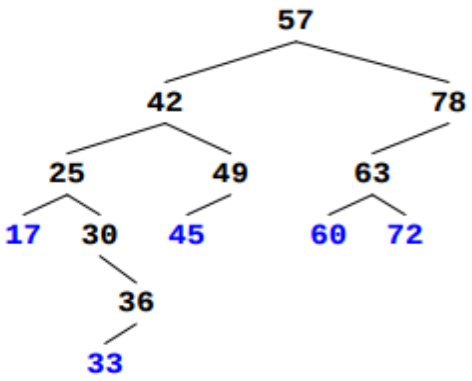
Step1: redraw the tree after applying the operation (50%)

Step2: explanation for your step (50%)

1) Delete the node with value 60 (2 points)

2) Delete the node with value 78 (3 points)

3) Delete the node with value 42 (3 points)



Rules:

- Cheating will be punished by giving -2 * assignment mark.
- Cheating is submitting code or report taken from any source that you did not fully write yourself (from the net, from a book, from a colleague, etc.)
- Giving your code to others is also considered cheating. Both the giver and the taker will get -10.
- People are encouraged to help others fix their codes but cannot give them their own code.
- Do not say we solved it together and we understand it. You can write the algorithm on paper together but each group should implement it alone.
- If you do not follow the delivery style (time and files names), your assignment will be rejected.