**Indian Institute of Technology, Guwahati**

**Department of Computer Science and Engineering**

**Data Structure Lab (CS210)**

**Assignment: 11**

**Date: 10rd November, 2016.** **Total Marks: 50 (Lab Assignment)**

**You need to complete the problem 1 and 2 by 3PM and 4:45PM, respectively. After 3:30PM and 5:15PM, problem 1 and problem 2, respectively, will not be evaluated.**

**Lab Assignment:**

1. You need to diagram the sorting process in a particular way: each swap of two keys is written on a separate line, under the previous swap. For example, here's an illustration of selection sort.

```
83 18 72 21 93 44  7 21
-----------------------
 7                 83
-----------------------
      21 72
-----------------------
         21          72
-----------------------
            44 93
-----------------------
               72    93
```

You must write down EVERY swap of two keys. You do not have to note when a key is swapped with itself; for instance, in the selection sort above, the 18 is swapped with itself, but I haven't written it down. In each row, you are not allowed to write down keys that haven't moved.

Implement Selection Sort. Print each iterations of selection sort as described above. **[18]**

**TEST:** Test your programs with following inputs:

- 876 23 365 88 33 23 55 22 22 1 2 34 6 6
- 1 2 3 4 5 6 7 8
- 8 7 6 5 4 3 2 1
- 1 1 1 1 1 1 1 1
- 83 18 72 21 93 44 7 21

2. One popular tree representation spurns separately encapsulated linked lists so that siblings are directly linked. It retains the "item" and "parent" references, but instead of referencing a list of children, each node references just its leftmost child. Each node also references its next sibling to the right. The "nextSibling" references are used to join the children of a node in a singly-linked list, whose head is the node's "firstChild". This tree is called a "SibTree", since siblings are central to the representation.

```
struct SibTreeNode {
  int data;
  SibTreeNode *parent;
  SibTreeNode *firstChild;
  SibTreeNode *nextSibling;
}
```

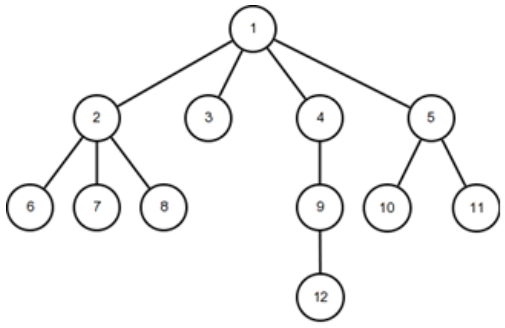
```
struct SibTree {
  SibTreeNode *root;
  int size;
}
```

Your task is to implement insertChild(), and removeLeaf() and printTree() methods of the SibTreeNode.

**Inserting New Children:** insertChild() takes three parameters**: key1, key2 and c**. First, you need to find the node which has data as 'key1'. You may assume that all the nodes in the tree have distinct data. Create a new child with key2 that is the $c^{th}$ child (from the left) of the node with key1. Existing children numbered c or higher are shifted one place to the right to accommodate. If c < 1, act as if c is 1. If node with key has fewer than c children, the new node is the last sibling. If there is no node with 'key1', your code should indicate that. Don't forget that SibTrees have a "size" field that needs to be updated. **[15]**

**Removing a Leaf:** removeLeaf() take a parameter 'key'. This function removes the node with data 'key' from the tree if it is a leaf, and does nothing if it is not a leaf. **[12]**

**Print a Tree:** You need to print a tree. It is acceptable even if your function does not print the children in proper indented positions. For example, printing the following for the given tree in is fine. **[5]**




```
Size: 12
Tree:
1
2 3 4 5
6 7 8 9 10 11
12
```

**TEST:**
Insert the numbers 1 to 12 in an initial empty tree such a way so that the structure of the resultant tree would be like the above tree. (Note that the above tree is not represented as a SibTree) Now, perform following operations in sequence and show the tree after each operations

1. removeLeaf(7)
2. removeLeaf(3)
3. insertChild(2, 99, 2)
4. insertChild(9, 44, 2)
5. insertChild(5, 88, 1)
6. insertChild(7, 22, 1)
7. removeLeaf(4)