

# CSCI-1200 Data Structures

## Test 3 — Practice Problems

*Note: This packet contains practice problems from two previous exams. Your exam will contain approximately half as many problems.*

### 1 Tracking Halloween Costume Rental Data [ / 23 ]

In Homework 7 we maintained information about what costume each customer was *currently* renting. In this problem, we would like to track the *history* of costumes each customer has rented. Consider the input sequence of rental receipts:

```
Carol rents pirate costume
Carol rents queen costume
Ann rents squirrel costume
Carol rents pirate costume
Brian rents rockstar costume
Ann rents pirate costume
Brian rents pirate costume
Carol rents rockstar costume
Carol rents pirate costume
```

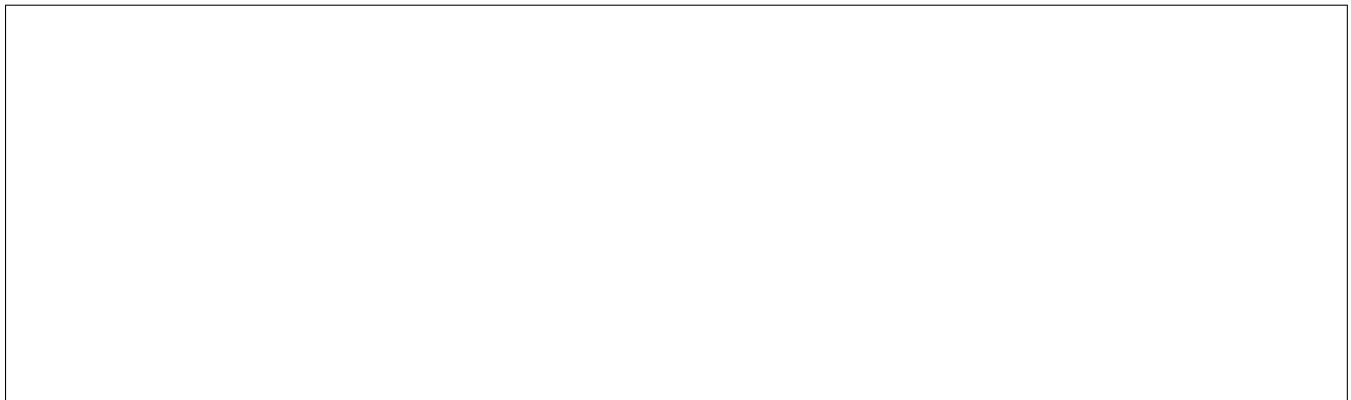
Your task is to read each line of rental information in this format and create a line of output for each rental stating the total number of times the customer has been to the shop and the number of times they rented that costume. The expected output for the above input is:

```
Carol has been to the shop 1 time(s) and rented a pirate costume 1 time(s).
Carol has been to the shop 2 time(s) and rented a queen costume 1 time(s).
Ann has been to the shop 1 time(s) and rented a squirrel costume 1 time(s).
Carol has been to the shop 3 time(s) and rented a pirate costume 2 time(s).
Brian has been to the shop 1 time(s) and rented a rockstar costume 1 time(s).
Ann has been to the shop 2 time(s) and rented a pirate costume 1 time(s).
Brian has been to the shop 2 time(s) and rented a pirate costume 1 time(s).
Carol has been to the shop 4 time(s) and rented a rockstar costume 1 time(s).
Carol has been to the shop 5 time(s) and rented a pirate costume 3 time(s).
```

Your code should be efficient for a store with many different costumes, many customers, and those customers are highly indecisive about which costume to wear.

#### 1.1 Diagram of the costume rental history structure [ / 9 ]

What is the best data structure in which to store this information? Draw a picture of the rental history above stored in your selected structure using the drawing conventions from lecture. *Hint: Use STL map!*



## 1.2 Implementation [ / 14 ]

Now, finish the code below to read and process each line of input and produce each line of output. Your program will read from `std::cin` and output to `std::cout`.

```
int main() {  
    // a data structure to hold the rental history  
  
    typedef  history;  
  
    history data;  
    // read each line  
    std::string name, token, costume, token2;  
    while (std::cin >> name >> token >> costume >> token2) {  
        assert (token == "rents");  
        assert (token2 == "costume");  

```

*sample solution: 12 line(s) of code*

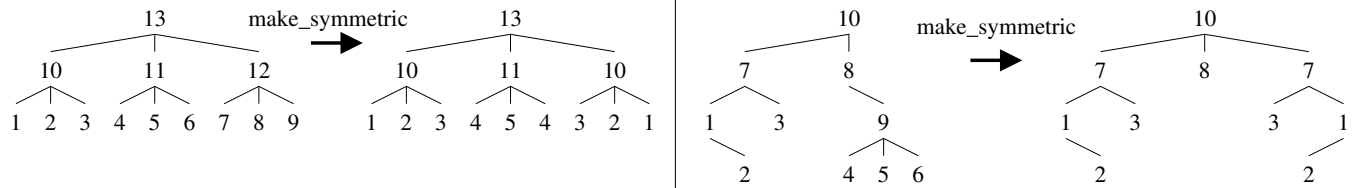
```
    }  
}
```

What is the order notation of your solution? Assume the input has  $n$  lines of rental information, the shop has  $c$  different costumes, there are  $p$  people/customers, each person visits the shop at most  $v$  times, and each person rents a specific costume at most  $k$  times.

## 2 Mirror Image Trinary Trees [ / 20 ]

In this problem you will work with trinary trees of integers, where each `TriNode` has up to 3 children. Your task is to force the structure and data to have mirror symmetry. That is, if you hold up the left half of the structure next to a mirror, the right half of the tree should be reconstructed to look like the mirror reflection. You should change the right half of the tree to look like the left side.

Here are some examples:



Write a function named `make_symmetric` that takes in a single argument, a pointer to the root of a trinary tree. Your function should edit the tree as described and illustrated above. Make sure your code does not have any memory leaks or other memory errors. You'll probably want to write and use one or more helper functions in your solution. Note: You have 1.5 pages to write your answer for this problem.

```
class TriNode {
public:
    TriNode(int v) : val(v), left(NULL),
        middle(NULL), right(NULL) {}
    int val;
    TriNode *left;
    TriNode *middle;
    TriNode *right;
};
```

*sample solution: 21 total line(s) of code*

Mirror Image Trinary Trees (continued)

*sample solution: 21 total line(s) of code*

### 3 Converting Trees to Lists [ / 25 ]

In this problem you will write a recursive function named `binarytree_to_linkedlist` that takes 3 arguments of type pointer to `DualNode<T>` (defined on the right). The first function argument holds the root of a binary tree. Your function should restructure the collection of `DualNodes` to form a simple doubly-linked list of the same data in a *pre-order traversal* of the original tree.

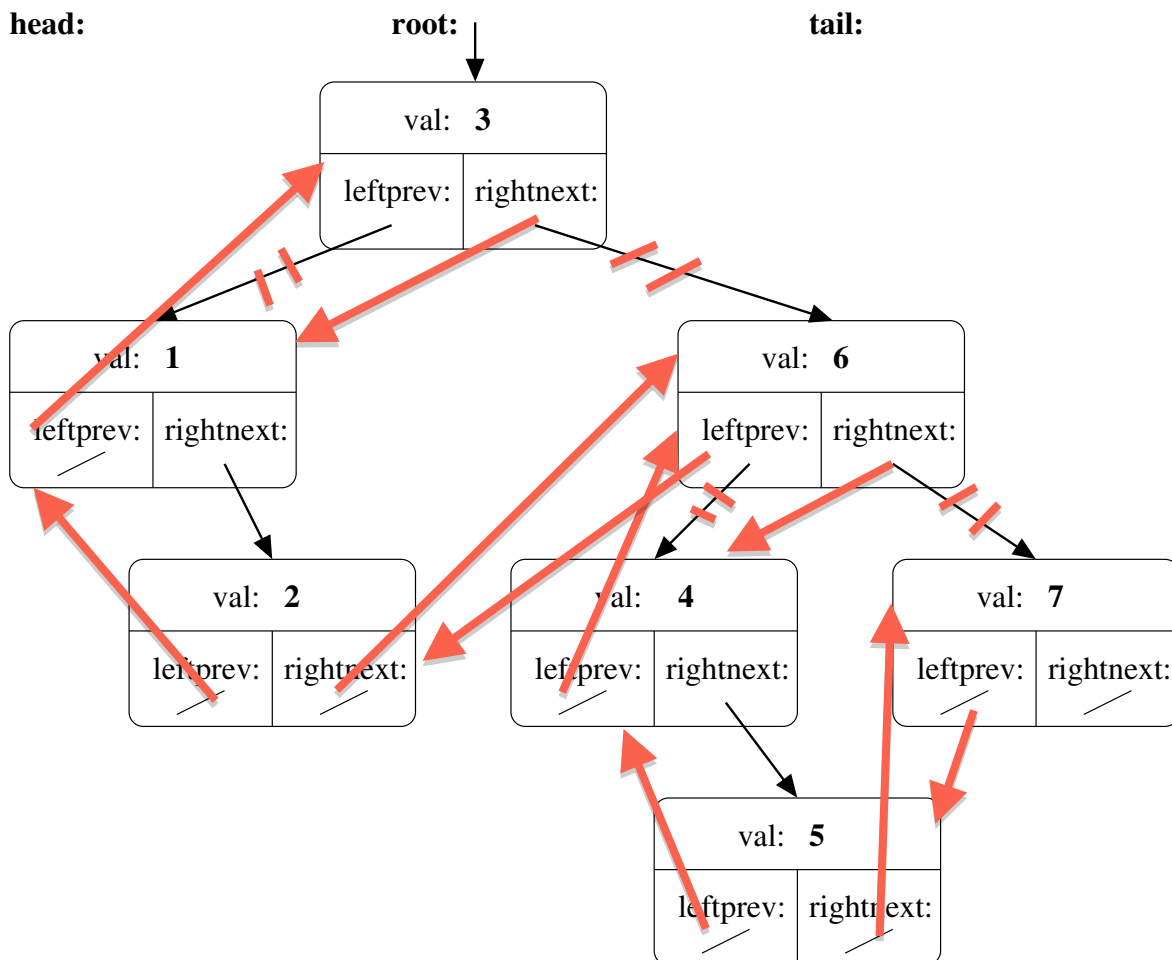
The second and third function arguments will store the head and tail, respectively, of the resulting linked list. The same two `DualNode` pointer variables within each `DualNode` are re-purposed once the data is re-structured.

*Important Note:* No new `DualNodes` are created by this function.

```
template <class T>
class DualNode {
public:
    DualNode<T>(const T& v) {
        val = v;
        leftprev = NULL;
        rightright = NULL;
    }
    T val;
    DualNode<T>* leftprev;
    DualNode<T>* rightright;
};
```

#### 3.1 Diagram of `binarytree_to_linkedlist` [ / 8 ]

First, edit the diagram below to show which pointers must change to convert the structure to a doubly-linked list that is a pre-order traversal of the original tree. Please be neat!



*Hint: Only two pointers in this structure are NULL after this function!*

### 3.2 Implementation of `binarytree_to_linkedlist` [ / 17 ]

Now, implement the `binarytree_to_linkedlist` function.

*sample solution: 25 line(s) of code*

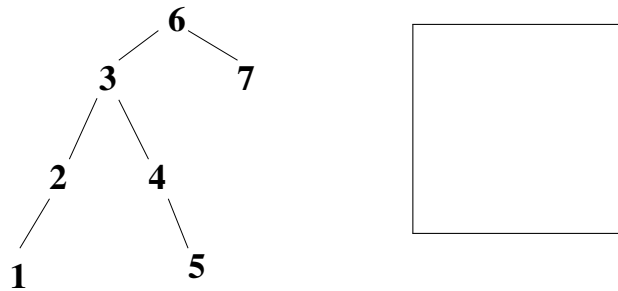
What is the order notation of your function for an input tree with  $n$  nodes and overall height  $h$ ?

## 4 Tree Traversal Bingo [ / 16]

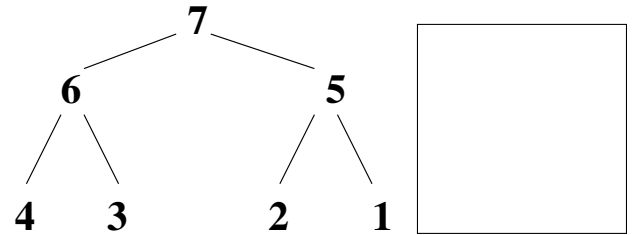
For each of the diagrams below, write a letter corresponding to one of the following statements that accurately describes the diagram. Each letter should be used exactly once.

- (A) is a priority queue
- (B) cannot be colored as a red-black tree
- (C) has breadth-first traversal: 7 6 5 4 3 2 1
- (D) is not a tree
- (E) has post-order traversal: 1 2 3 4 5 6 7
- (F) is a binary search tree

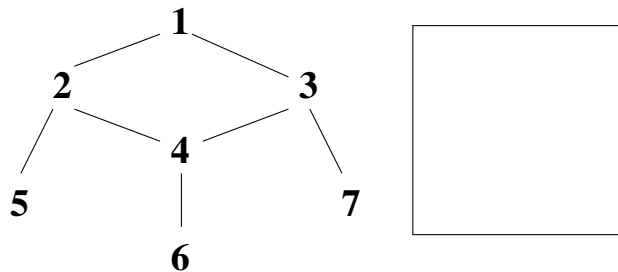
4.1 [ /3]



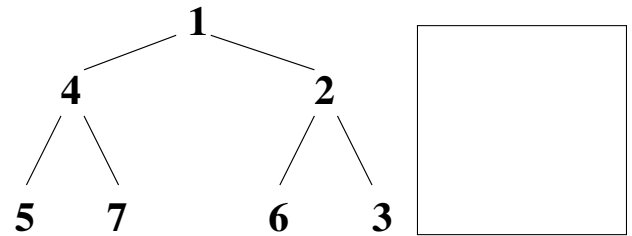
4.2 [ /3]



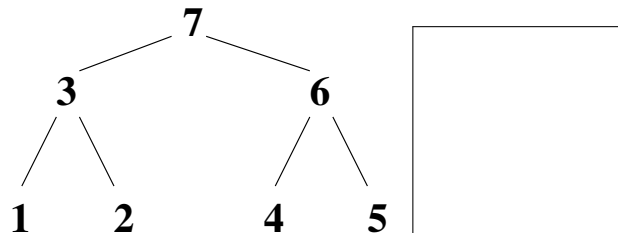
4.3 [ /3]



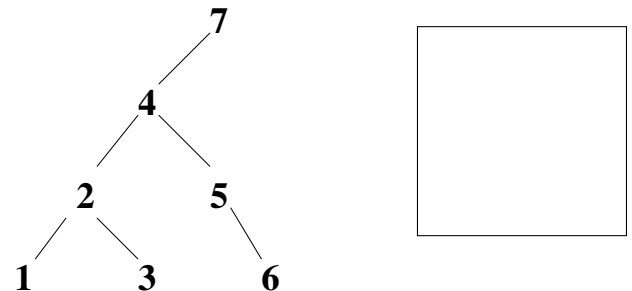
4.4 [ /3]



4.5 [ /3]



4.6 [ /3]



## 5 Sorting with a Set [ / 13]

Write a fragment of C++ code to read in  $n$  integers from `std::cin` and then output those numbers to `std::cout` in reverse sorted order (largest first). Your code should use the STL `set` container class. You may not use arrays, vectors, lists, or maps. You may assume there are no duplicates in the input data.

*sample solution: 12 line(s) of code*

What is the order notation of your solution in terms of  $n$ , the number of integers read from `std::cin`? Write 2 or 3 concise and well written sentences justifying your answer.



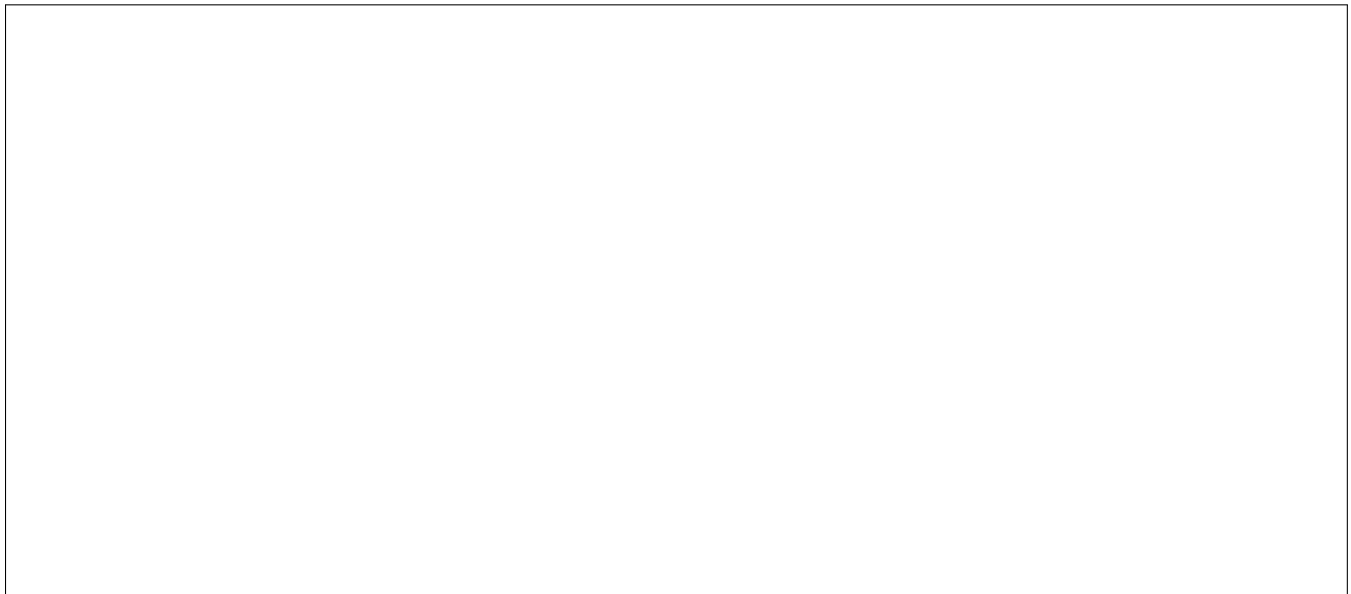
## 6 Majority Maps [ /23]

Consider the following organization of voting data:

```
std::map<std::string, std::string> voter_a;
voter_a["color"] = "blue";
voter_a["senator"] = "schumer";
voter_a["fast car"] = "porsche";
std::map<std::string, std::string> voter_b;
voter_b["color"] = "red";
voter_b["fast car"] = "ferrari";
voter_b["GM"] = "chuck carletta";
voter_b["senator"] = "gillibrand";
std::map<std::string, std::string> voter_c;
voter_c["GM"] = "the moose";
voter_c["fast car"] = "jaguar";
voter_c["color"] = "blue";
std::map<std::string, std::string> voter_d;
voter_d["color"] = "green";
voter_d["fast car"] = "ferrari";
voter_d["GM"] = "chuck carletta";
voter_d["senator"] = "gillibrand";
std::map<std::string, std::string> voter_e;
voter_e["senator"] = "gillibrand";
voter_e["fast car"] = "delorean";
voter_e["color"] = "blue";
std::vector<std::map<std::string, std::string> > voters;
voters.push_back(voter_a); voters.push_back(voter_b); voters.push_back(voter_c);
voters.push_back(voter_d); voters.push_back(voter_e);
```

### 6.1 Diagram of the voters structure [ /8]

Draw a picture of the data stored in the `voters` variable using the conventions from lecture:



### 6.2 Implementing majority [ /15]

Now, write the function `majority` to determine the 'winner' of each category. The winner must win a *majority* of the votes; that is, more than half of the voters must choose that entity. For example:

```
std::cout << "fast car = " << majority(voters, "fast car") << std::endl;
std::cout << "color = " << majority(voters, "color") << std::endl;
```

```
std::cout << "senator   = " << majority(voters, "senator") << std::endl;
std::cout << "GM       = " << majority(voters, "GM") << std::endl;
```

Results in this output:

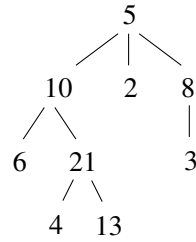
```
fast car = (no consensus)
color    = blue
senator  = gillibrand
GM       = (no consensus)
```

Make sure to use `const` and reference as appropriate in your implementation.

What is the order notation of your solution? Assume there are  $v$  voters,  $c$  categories,  $k$  different entities in each category receiving votes, and the entity receiving the most votes receives  $m$  votes.

## 7 Counting Levels in General Trees [ /16]

Write a function `count_at_level`, that takes in 2 arguments: a pointer to a `Node` and an integer `level` and returns the number of elements at that *level* of the tree. For the example to the right, we have 1 node at level 0 (the root, 5), 3 nodes at level 1, 3 nodes at level 2, and 2 nodes at level 3.



```
template <class T>
class Node {
public:
    T value;
    std::vector<Node*> children;
};
```

When your function is asked to count the elements on the 2nd level of the tree above, in what order does it visit the nodes of the tree? List every node that is visited in calculating the answer, not just the nodes on the 2nd level.

What is the common name for the traversal order of your function: “breadth first”, “bottom up”, “top down”, “depth first”, “in order”, “reverse order”, “random order”, or “other”?

## 8 Perfectly Balanced Binary Search Trees [ /16]

Write a templated function `construct_balanced` that takes in one argument, a sorted STL vector of template type `T`, and returns a pointer to the root `Node` of a perfectly balanced binary search tree containing these values. You may assume the vector contains  $n$  values, where  $n = 2^h - 1$ , and  $h$  is the height of the final tree.

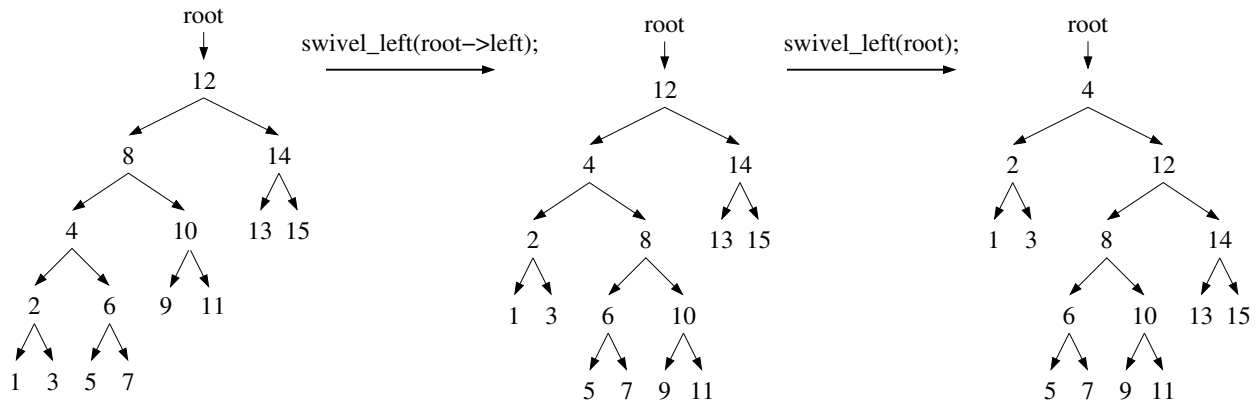
Your solution should be recursive. You may define and use a helper function as part of your solution. Make sure to correctly initialize all links between the Nodes (parent, left, and right).

```
template <class T> class Node {
public:
    Node(const T& v) :
        value(v), parent(NULL),
        left(NULL), right(NULL) {}
    T value;
    Node *parent;
    Node *left;
    Node *right;
};
```

## 9 Swiveling for Balancing and Unbalancing [ /28]

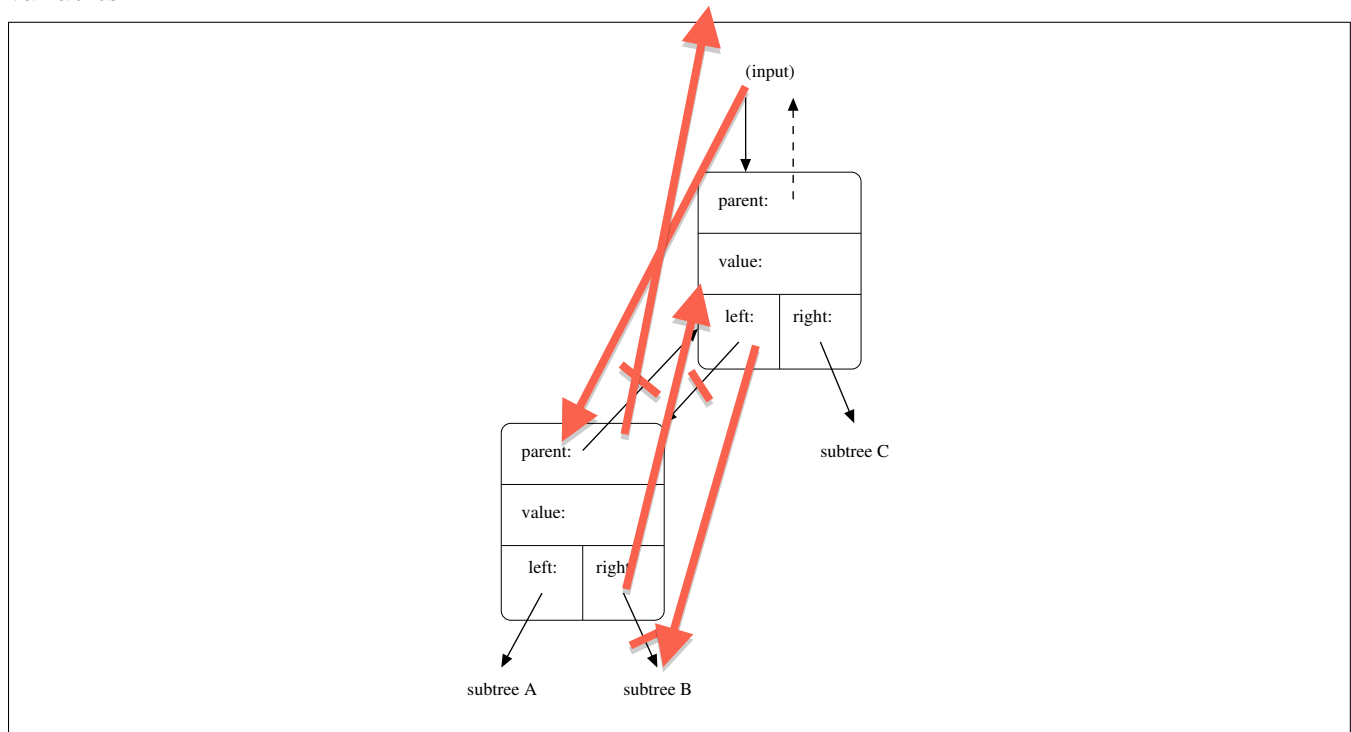
In this problem we will write a helper function `swivel_left` that can be used to adjust the balance/imbalance of a binary search tree structure. This templated function takes in a single argument: a pointer to the root `Node<T>`, *by reference*, and does a local rearrangement of the nodes in the tree, so that the left node of the input `Node` is bumped up a level and the input node is bumped down a level. The binary search tree property of the entire tree is maintained. For example, here are two calls to the `swivel_left` function:

```
template <class T> class Node {
public:
    Node(const T& v) :
        value(v), parent(NULL),
        left(NULL), right(NULL) {}
    T value;
    Node *parent;
    Node *left;
    Node *right;
};
```



### 9.1 Diagram of `swivel_left` [ /8]

First, edit the diagram below to show all of the changes that must happen to the local structure when `swivel_left` is called. Note: We will not edit the values stored in any `Node`, we will only edit the pointer variables.



## 9.2 Implementation of `swivel_left` [ /12]

Now, implement the `swivel_left` function. Make sure your implementation handles the corner cases where some of the pointers are NULL.

## 9.3 Converting a binary search tree into a sorted linked list [ /8]

Finally, write a recursive function `flatten` that takes in the root of a binary search tree and uses `swivel_left` to restructure the binary search tree to form a completely unbalanced tree that is essentially a sorted linked list with the root pointing at the smallest element.

## 10 Data Structure Comparison Short Answer [ /13]

Write 3-4 concise and well-written sentences comparing the data structures below and presenting your justification for each answer.

**Hash table vs. map** [ /6] We've just learned that hash tables are amazing and magically fast for find, insert, and erase operations. Why would someone choose a map over a hash table? What key feature of iteration for a map is not true for a hash table?

**set vs. vector** [ /7] STL **sets** ensure there are no duplicate entries in a collection of entities; for example, in HW7 a set could be used to ensure that one person did not checkout multiple copies of the same book from the library. However, in this application an STL **vector** is preferable to an STL **set**. What information cannot be represented in a **set**? What memory/performance advantages does a **vector** have over a **set**?