1 Order Up! [ / 48 ]

Alyssa P. Hacker is consulting for a restaurant to build a customer and order tracking system. Below is a small sample of the data structure she’s designed. She explains that it uses 5 different STL data structures, but it does NOT use STL vector, or the C-style array, or any custom classes.

The design allows us to study the frequency of meals (combinations of items on the menu) ordered by customers, and who has ordered the meal most recently. For example, 3 different customers have ordered the meal “salad & tea”. Herta has ordered that meal twice and Jacob has ordered it most recently.

To record a customer order in the structure above, we typically use the following 3 argument function:

```
placeOrder( data , "Jacob" , meal with burger, fries, and coke );
```

Where data is the complete structure diagrammed above. This call will update the fifth row of the structure to record that Jacob has now ordered that meal twice, and furthermore, the sequence will be changed so that Jacob is last (swapping places with Victoria) because he has now ordered that meal most recently.

1.1 Fast Food typedefs [ / 6 ]

After looking ahead through the rest of this problem, let’s define a few helpful typedefs. You will be graded on the convenience and efficiency of the overall structure. Also remember that Alyssa said she used 5 different STL types, but NOT vector.
1.2 Organizing the Meals

On the previous page we can see the restaurant’s desired organization for the meals. We should start with the simple (1 item) meals and progress to more complicated meals. Meals of the same complexity are organized alphabetically. Write the necessary helper function so this happens automatically.

1.3 Complexity Analysis

Let’s assume that the restaurant has $i$ items on the menu and $c$ unique customers. Customers have created $m$ different meals (each with at most $k$ menu items). Any one meal has been ordered by at most $d$ different customers, and at most $t$ times by one customer. What is the Big ‘O’ Notation for the following functions?

placeOrder  Note: We don’t ask you to implement this.

namelessOrder  You will implement this on the next page (page 4).

orderMyFavorite  You will implement this in a couple of pages (page 5).
1.4 Convenience for Our Most Loyal Customers: `namelessOrder`

This structure allows us to streamline ordering for the most frequent customers. They don’t even need to leave their name when they place an order. For example:

```plaintext
namelessOrder( data, meal with nuggets and fanta );
```

prints “Welcome back, Ben!” and places his order (because Ben has ordered that meal more often than anyone else!) However, attempts to use this function can fail and result in error messages:

```plaintext
namelessOrder( data, meal with salad, fries, and coke );
```

will print to STDERR: “ERROR: What name should we put on this order?” (because two different people, Herta and Victoria, are tied for most times ordering that meal). And

```plaintext
namelessOrder( data, meal with happy meal and coke );
```

will print to STDERR: “ERROR: No one’s ordered this meal before!”

*sample solution: ~ 30 line(s) of code*
1.5 We Know What You Want! \texttt{orderMyFavorite}\[ \texttt{\quad / 14} \]

Alternately, customers may re-order their personal favorite (most frequently ordered) meal:

\begin{verbatim}
orderMyFavorite( data, "Herta" );
\end{verbatim}

which will respond with the success message: “Placing your order for salad & tea.” However,

\begin{verbatim}
orderMyFavorite( data, "Victoria" );
\end{verbatim}

prints the error message “ERROR: You have multiple, equally-favorite meals.” (because Victoria

\begin{verbatim}
has ordered four different meals, one time each). And
\end{verbatim}

\begin{verbatim}
orderMyFavorite( data, "Louis" );
\end{verbatim}

prints “ERROR: We don’t have any prior orders for you.”

Write a function named `each_level_all_pairs` that takes in a pointer to the root `Node` of a binary tree and prints all pairs of values from each “level” of the tree. For example, the function will print the following 10 pairs for the tree on the right, which has 4 levels:

(2,3) (4,5) (4,6) (5,6) (7,8) (7,9) (7,10) (8,9) (8,10) (9,10)

There are no pairs from the top level (because there’s only 1 value). There’s 1 pair from the second level. We have 3 values and 3 pairs from the third level, and 4 values and 6 pairs from the lowest level.

*Note: The extra space between pairs of different levels is optional.*

```
template <class T>
class Node {
  public:
    T value;
    Node* left;
    Node* right;
};
```
Write a function named `treevision` that takes in two `Node` pointers and modifies the first tree to match the second tree in shape and values. The function returns a `Trio` of three numbers, indicating how many nodes were edited, how many nodes were added, and how many nodes were removed. For example `treevision(foo,bar)` will return the values 2 3 1 because 2 nodes were edited (`'a'`→`'A'` and `'d'`→`'D'`), 3 nodes were added (`'Q'`, `'F'`, and `'G'`) and 1 node was removed (`'e'`).

```cpp
class Node {
public:
    Node(char v): value(v) { left = right = NULL; }
    char value;
    Node* left;
    Node* right;
};

class Trio {
public:
    Trio(int e, int a, int r): edit(e), add(a), remove(r) {} 
    int edit;
    int add;
    int remove;
};
```

*sample solution: 25 line(s) of code*
Draw a balanced binary tree with in-order traversal: the fox jumped over the lazy dogs.

Draw a binary search tree with leaves: $2^{10}, 10, 10^2$, and $\sqrt{10}$.

Draw a balanced ternary tree with pre-order traversal: 13 12 11 10 9 8 7 6 5 4 3 2 1.

Draw a red-black tree with values ‘a’–‘g’, with root ‘e’ and 3 red nodes: ‘b’, ‘d’, and ‘g’.
Cameron (who helped develop YACS) is thinking about an extension to assist students in maximizing use of TA office hours for their registered classes. He has access to the following tables of data from the registrar using STL maps and vectors (shown here with sample data):

```
<table>
<thead>
<tr>
<th>ta_assignments</th>
<th>office_hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abby</td>
<td>Monday 3pm, AE Thursday 2pm, Folsom</td>
</tr>
<tr>
<td>Eric</td>
<td>Friday 4pm, Sage</td>
</tr>
<tr>
<td>Milo</td>
<td>Tuesday 5pm, J-Rowl</td>
</tr>
<tr>
<td>Ryan</td>
<td>Thursday 4pm, Folsom Wednesday 6pm, Folsom</td>
</tr>
<tr>
<td>Sinclair</td>
<td>Thursday 4pm, Folsom</td>
</tr>
<tr>
<td>Tim</td>
<td>Thursday 4pm, Folsom</td>
</tr>
<tr>
<td>Wendy</td>
<td>Thursday 4pm, Folsom</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
</tr>
<tr>
<td>Louis</td>
</tr>
</tbody>
</table>
```

Cameron proposes we write a function `GetHelp` that takes in these 3 tables of data, and the name of a student, and returns an STL set of strings with the day, time, and location of all relevant office hours. For example, by printing the output of:

```
GetHelp(ta_assignments, students, office_hours, "Ben");
```

We will learn that Ben should attend the following office hours:

```
Thursday 4pm, Folsom
Tuesday 5pm, J-Rowl
Wednesday 6pm, Folsom
```

### 5.1 Just the typedefs, Ma’am

After looking ahead through the rest of this problem, let’s define a few helpful `typedefs`:

```
typedef type_t;
typedef type_s;
typedef type_o;
```
5.2 Let’s get Ben some help in Office Hours!

Write the `GetHelp` function below. Perform some basic error checking. We suggest you use `assert` to verify any assumptions as you work through the implementation.

```cpp
std::set<std::string> GetHelp(const type_t &ta_assignments,
                   const type_s &students,
                   const type_o &office_hours,
                   const std::string &student) {

    sample solution: 18 line(s) of code

}
```
5.3 What Would Milo Do?

Milo notices someone forgot their laptop charger in office hours. In addition to posting a message on the Submitty Discussion Forum, Milo would like to message students in the other courses that share his assigned office hour times and locations. Let’s write another function, `GetShared`, that returns an STL `set` of the names of other courses that share an office hour time and location with the specified TA. For example, if we print the return value of:

```
GetShared(ta_assignments, office_hours, "Milo");
```

We will learn that Milo’s Data Structures office hours overlap with these other courses:

```
bio
csl
```

```cpp
class std::set<
set<std::string> GetShared(const type_t &ta_assignments,
const type_o &office_hours,
const std::string &ta) {
```
5.4 Who’s Afraid of the Big Bad ’O’ Notation? [ / 7 ]

What is the running time for each of the functions you wrote? Let’s say that we have $s$ students, each student takes on average $k$ courses, the university has $t$ TAs, each TA is assigned on average to $h$ office hour slots per week, the university has $c$ total courses, and the output contains $x$ values. Justify your answers.

GetHelp

GetShared

5.5 Making some Mappy Improvements [ / 3 ]

Is this the best organization of data to accomplish these tasks? Describe a minor change to one of these tables (still sticking with STL maps and vectors) that will simplify the algorithm(s) you wrote above and/or improve the big ’O’ notation. Write 2-3 sentences describing the change and why it’s an improvement.

5.6 Are Hash Tables always better than Binary Search Trees? [ / 3 ]

If we switched the maps and sets in this problem to be hash tables, what would be one advantage/improvement? What would be one disadvantage/loss? Write 2 concise, technical, and well-written sentences.
In this problem we will inspect the structure of a memory diagram composed of Node objects, as declared on the right. We will decide if the current arrangement, contents, and coloring of nodes is a valid, well-balanced, binary search tree. Note: “red” nodes are visualized white. On the next two pages you will complete the implementation of the recursive functions used in the fragment of code below:

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

```cpp
if (!is_tree(root)) std::cout << "NOT A TREE\n";
else if (!is_binary(root)) std::cout << "NOT A BINARY TREE\n";
else if (!is_bst(root)) std::cout << "NOT A BINARY SEARCH TREE\n";
else if (!is_red_black(root)) std::cout << "NOT A RED BLACK TREE\n";
else std::cout << "you're a well-balanced binary search tree!\n";
```

### 6.1 Diagrams

For each of the 4 diagrams below, write the statement that will print when the diagram is passed to the code fragment above. **Hint: Each of the four "NOT A ..." phrases will be used exactly once.**

Now, draw a diagram that contains the integers 1-9 that will print the statement “you’re a well-balanced binary search tree!”
**6.2 is_tree Implementation**

```cpp
template <class T> bool is_tree(Node<T> *n) {
    std::set<Node<T>*> already_seen;
    return is_tree(n, already_seen);
}
```

**6.3 is_binary Implementation**

```cpp
template <class T> bool is_binary(Node<T> *n) {
    sample solution: 10 line(s) of code
}
```

```cpp
template <class T> bool is_binary(Node<T> *n) {
    sample solution: 7 line(s) of code
}
```
template <class T>
bool is_bst(Node<T> *n, Node<T> *lower=NULL, Node<T> *upper=NULL) {

sample solution: 6 line(s) of code
}

template <class T> bool double_red(Node<T> *n) {

sample solution: 6 line(s) of code
}

template <class T> int black_count(Node<T> *n) {

sample solution: 6 line(s) of code
}

template <class T> bool is_red_black(Node<T> *n) {
    if (double_red(n)) return false;
    if (black_count(n) == -1) return false;
    return true;
}


Write a function named `construct_breadth` that takes in an STL list of STL strings and creates and returns a pointer to a well-balanced binary tree of `Node`s with breadth-first traversal order that matches the input.

```
class Node {
public:
    Node(const std::string &v)
    { value=v; left=right=NULL; }
    std::string value;
    Node* left;
    Node* right;
};
```

Sample solution: 20 line(s) of code

Give an input list of 10 RPI building names that when passed to `construct_breadth` will create a valid binary search tree. Also, draw the produced tree.
Louis B. Reasoner has started a new job in the RPI housing office and is tasked with re-writing their dorm assignment software. An example of the input text file is shown on the right. He has already written code to parse the input file and store the data:

```cpp
    type_b dorms;
    type_c people;
    std::ifstream istr("dorm_data.txt");
    std::string dorm, first, last;
    int room;
    while (istr >> dorm >> room >> first >> last) {
        dorms[dorm][room].push_back(std::make_pair(first, last));
        people[std::make_pair(last, first)] = std::make_pair(dorm, room);
    }
```

He wrote a helper function to email students with their room assignment and roommate information:

```cpp
void Info( /*?*/ dorms, /*?*/ people, /*?*/ first, /*?*/ last) {
    std::string dorm;
    int room;
    LookupRoomAssignment(people, first, last, dorm, room);
    if (dorm == "") {
        std::cout << first << " " << last << " does not have a room assignment." << std::endl;
    } else {
        std::cout << first << " " << last << " is assigned to " << dorm << " " << room << "." << std::endl;
        PrintRoommates(dorms, first, last, dorm, room);
    }
}
```

And here is sample usage of his code:

```cpp
Info(dorms, people, "Fred", "Harrison");
Info(dorms, people, "Sally", "Harrison");
Info(dorms, people, "Sally", "Morris");
Info(dorms, people, "Chris", "Thompson");
std::cout << "Barton has " << NumRoomsWithOccupancy(dorms, "Barton", 3) << " triple(s)." << std::endl;
std::cout << "Crockett has " << NumRoomsWithOccupancy(dorms, "Crockett", 1) << " singles(s)." << std::endl;
```

Resulting in this output:

Fred Harrison is assigned to Nugent 316.
Fred's roommate(s) are: George
Sally Harrison does not have a room assignment.
Sally Morris is assigned to Barton 201.
Sally's roommate(s) are: Alice Jessica
Chris Thompson is assigned to Crockett 106.
Chris has no roommates.
Barton has 1 triple(s).
Crockett has 2 singles(s).

You will write a few functions to help Louis finish the implementation and then analyze the running time.
8.1 The typedefs

First, let’s fill in these `typedef` declarations. Note: `type_a` is optional, but it may help simplify your code.

```
typedef type_a;
typedef type_b;
typedef type_c;
```

8.2 Data Structure Sketch

Sketch the contents of the `dorms` variable for the sample input text file. Follow the conventions from lecture for this diagram.
8.3 Implementation of NumRoomsWithOccupancy [ / 8 ]

sample solution: 12 line(s) of code

8.4 Implementation of LookupRoomAssignment [ / 9 ]

sample solution: 10 line(s) of code
8.5 Implementation of PrintRoommates

Sample solution: 17 line(s) of code

8.6 Order Notation

If the housing system contains $s$ students and $d$ dorms, with $r$ rooms per dorm, and an average/max of $k$ students per room, what is the running time for each of the functions above?

NumRoomsWithOccupancy

LookupRoomAssignment

PrintRoommates
Alyssa P. Hacker has augmented our binary search tree with an additional member variable, the *height* of each node. She argues that this small change will allow *erase* to help minimize the overall tree height. She has defined a leaf node to have height = 1. Fill in the missing pieces to finish the implementation of *erase*.

```cpp
class Node {
public:
    int value;
    int height;
    Node* left;
    Node* right;
};

bool erase(Node* &n, int v) {
    bool answer = true;
    if (n == NULL) { return false; }
    if (v < n->value) { answer = erase(n->left, v); }
    else if (v > n->value) { answer = erase(n->right, v); }
    else {
        if (n->right == NULL && n->left == NULL) { delete n; n = NULL; }
        else if (n->right == NULL) { Node* tmp = n->left; delete n; n = tmp; }
        else if (n->left == NULL) { Node* tmp = n->right; delete n; n = tmp; }
        else { // handle the case of 2 children
            sample solution: 11 line(s) of code
        }
    }
    if (n != NULL) { // update the height
        sample solution: 5 line(s) of code
    }
    return answer;
}
```

20
10 Exactly Balanced Binary Search Trees Insertion Order

Draw an exactly balanced binary search tree that contains the characters: b, e, r, n, a, d, and i.

In order for a binary tree to be exactly balanced, what must be true about \( n \), the number of elements?

Using the simple binary search tree insertion algorithm from lecture, give two different orderings for insertion of these elements that will result in an exactly balanced binary search tree.

Suppose we have an STL set named data, whose size allows construction of an exactly balanced tree (as you specified earlier). Let’s write a recursive function named `print_balanced_order` that will print to `std::cout` an ordering of these elements for insertion to make an exactly balanced binary search tree.

```cpp
print_balanced_order(data.begin(), data.end());
```

*sample solution: 16 line(s) of code*