COEN 79: Object-Oriented Programming
Homework #6

Tamir Enkhjargal

March 14, 2020

Question #1

Please complete the following implementation:

```
template <class Item>
binary_tree_node <Item>* tree_copy(const binary_tree_node <Item>* root_ptr) {
    binary_tree_node<Item>> *l_ptr;
    binary_tree_node<Item>> *r_ptr;
    if(root_ptr == NULL)
        return NULL;
    else {
        l_ptr = tree_copy(root_ptr->left());
        r_ptr = tree_copy(root_ptr->right());
    }
}
```

Using the previous implementation, complete the following function for the bag class given in Appendix 1:

```
template <class Item>
void bag<Item>::operator = (const bag<Item>& source) {
    if(this == &source)
        return;
    tree_clear(root_ptr);
    root_ptr = tree_copy(source.root_ptr);
}
```

Question #2

For the bag class defined in Appendix 1, please complete the following function:

```
template <class Item>
void bag<Item>::insert(const Item& entry) {
    binary_tree_node<Item> *cursor = root_ptr;
    bool done = false;
    if(root_ptr == NULL) {
        root_ptr = new binary_tree_node<Item>(entry);
        return;
    }
    do {
        if(cursor->data() >= entry) {
            if(cursor->left() != NULL)
                cursor = cursor->left();
            else {
                cursor->left() = new binary_tree_node<Item>(entry);
                done = true;
            }
        }
```

```
        else {
            if(cursor->right() != NULL)
                cursor = cursor->right();
            else {
                cursor->right() = new binary_tree_node<Item>(entry);
                done = true;
            }
        }
    }
    while(!done);
}
```

Question #3

Suppose `MINIMUM` is 1000 for a B-tree. The tree has a root and one level of 1000 nodes (children) below that. How many *entries* are in the tree? (Please provide a range and explain your answer).

If there are 1000 nodes, then there are 1000 - 1 = 999 entries. Each of these nodes can also have 1000 entries. Therefore, the minimum is: $1000*1000+999 = 1000999$ and maximum of $1000 * 2 * 1000 + 999 = 2000999$ entries.

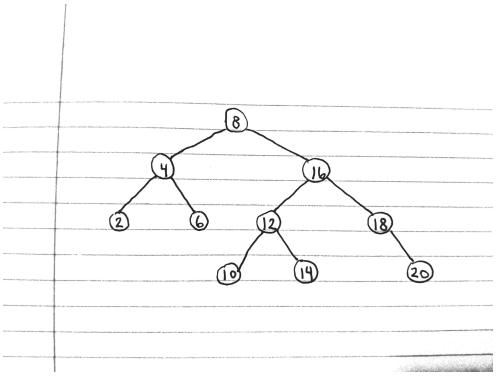
Question #4

Write a function to perform *left-right* rotation on the following AVL tree. The figure shows the steps. (Note: Please implement the function in two steps: (1) left rotation, (2) right rotation.)

```
template <class Item>
binary_tree_node <Item>* left_right_rotation(binary_tree_node <Item>*& parent) {
    binary_tree_node <Item>* temp1;
    temp1 = parent->right();
    parent->set_right(temp1->left());
    temp1->set_left(parent);
    binary_tree_node <Item>* temp2;
    temp2 = temp1->left();
    temp1->set_left(temp2->right());
    temp2->set_right(temp1);
    return temp2;
}
```

Question #5

Add the following numbers to an AVL tree. Please draw the final tree. `2, 4, 6, 8, 10, 12, 20, 18, 16, 14`



Question #6

The following functions are available:

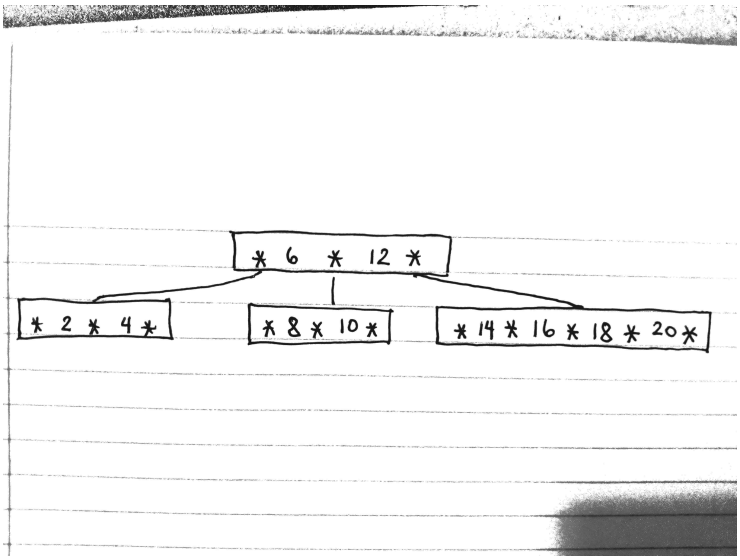
```
int height();
int diff();
binary_tree_node left_rotation();
binary_tree_node right_rotation();
binary_tree_node left_right_rotation();
binary_tree_node right_left_rotation();
```

Complete the following function, which balances a tree rooted at `temp`.

```
template <class Item>
binary_tree_node<Item>* balance(binary_tree_node <Item>*& temp) {
    if(diff(temp) >= 2) {
        if(diff(temp->left()) == -1)
            return balance(left_right_rotation(temp));
        else
            return balance(left_rotation(temp));
    }
    else if(diff(temp) <= -2) {
        if(diff(temp->right()) == 1)
            return balance(right_left_rotation(temp));
        else
            return balance(right_rotation(temp));
    }
}
```

Question #7

Please add the following numbers to a B-Tree with `MIN` = 2. Draw the final tree. `2, 4, 6, 8, 10, 12, 20, 18, 16, 14`



Question #8

Complete the following function. Present a *recursive* implementation.

```
template <class Item>
void bst_remove_max(bianry_tree_node <Item>*& root_ptr, Item& removed) {
    binary_tree_node<Item> *oldroot_ptr;
    assert(root_ptr != NULL);
    if(root_ptr->right() != NULL)
        bst_remove_max(root_ptr->right(), removed);
    else {
        removed = root_ptr->data();
        oldroot_ptr = root_ptr;
        root_ptr = root_ptr->left();
        delete oldroot_ptr;
    }
}
```

Question #9

Please implement the following function (*recursively).*

```
template <class Item>
void flip(binary_tree_node <Item>* root_ptr) {
    if(root_ptr == NULL)
        return;
    else {
        binary_tree_node <Item>* temp;

        flip(root_ptr->left());
        flip(root_ptr->right());

        temp = root_ptr->left();
        root_ptr->left = root_ptr->right;
        root_ptr->right = temp;
    }
}
```

Question #10

What are the outputs of the following programs?

```
Derived's destructor
Base2's destructor
Base1's destructor

The program will not compile, because Derived only has public
inheritance, therefore it can't access Base's private variables:
i and j. Derived only has access to Base's constructor;

Inside Q

The program will not compile on line 10,
because you can't point a Derived object to a Base object.
```

Appendix 1:

Bag class with binary search tree.

```
1.  template < class Item >
2.  class bag {
3.
4.  public:
5.      // TYPEDEFS
6.      typedef std::size_t size_type;
7.      typedef Item value_type;
8.
9.      // CONSTRUCTORS and DESTRUCTOR
10.     bag() {  root_ptr = NULL;  }
11.     bag(const bag& source);
12.     ~bag();
13.
14.     // MODIFICATION functions
15.     size_type erase(const Item& target);
16.     bool erase_one(const Item& target);
17.     void insert(const Item& entry);
18.     void operator += (const bag& addend);
19.     void operator = (const bag& source);
20.
21.     // CONSTANT functions
22.     size_type size() const;
23.     size_type count(const Item& target) const;
24.     void debug() const {  print(root_ptr, 0);  }
25.
26. private:
27.     binary_tree_node<Item>* root_ptr; // Root pointer of binary search tree
28.     void insert_all (binary_tree_node<Item>* addroot_ptr);
29. };
```