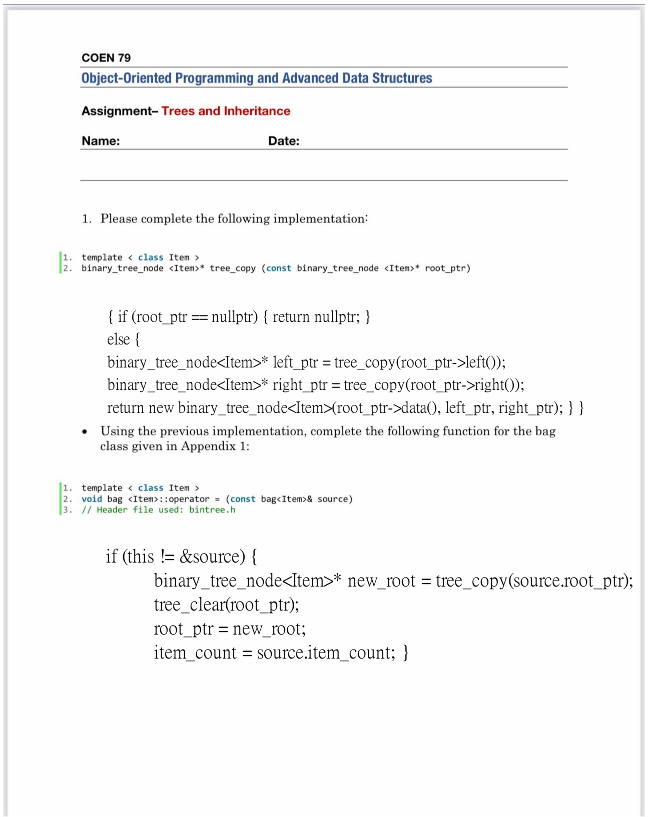
Assignment 7

Sunday, June 4, 2023 6:13 PM



COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment– **Trees and Inheritance**

Name: **Date:**

1. Please complete the following implementation:

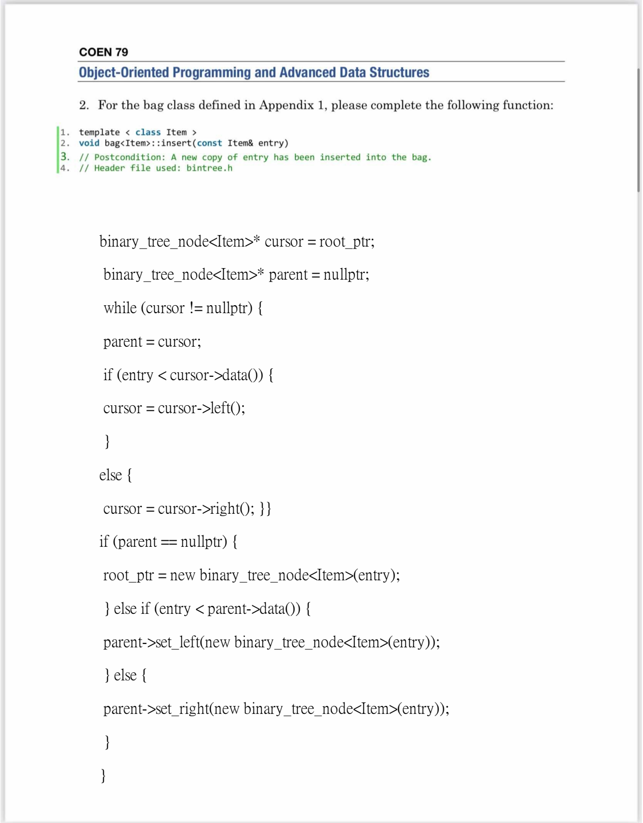
```
template < class Item >
binary_tree_node <Item>* tree_copy (const binary_tree_node <Item>* root_ptr)
```

```
{ if (root_ptr == nullptr) { return nullptr; }
else {
binary_tree_node<Item>* left_ptr = tree_copy(root_ptr->left());
binary_tree_node<Item>* right_ptr = tree_copy(root_ptr->right());
return new binary_tree_node<Item>(root_ptr->data(), left_ptr, right_ptr); } }
```

- 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)
// Header file used: bintree.h
```

```
if (this != &source) {
        binary_tree_node<Item>* new_root = tree_copy(source.root_ptr);
        tree_clear(root_ptr);
        root_ptr = new_root;
        item_count = source.item_count; }
```



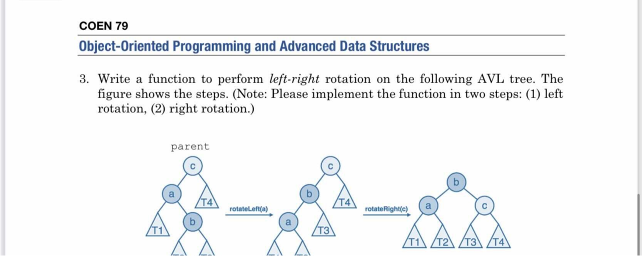
COEN 79

Object-Oriented Programming and Advanced Data Structures

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

```
template < class Item >
void bag<Item>::insert(const Item& entry)
// Postcondition: A new copy of entry has been inserted into the bag.
// Header file used: bintree.h
```

```
binary_tree_node<Item>* cursor = root_ptr;
 binary_tree_node<Item>* parent = nullptr;
 while (cursor != nullptr) {
 parent = cursor;
 if (entry < cursor->data()) {
 cursor = cursor->left();
 }
else {
 cursor = cursor->right(); }}
if (parent == nullptr) {
 root_ptr = new binary_tree_node<Item>(entry);
 } else if (entry < parent->data()) {
 parent->set_left(new binary_tree_node<Item>(entry));
 } else {
 parent->set_right(new binary_tree_node<Item>(entry));
 }
}
```

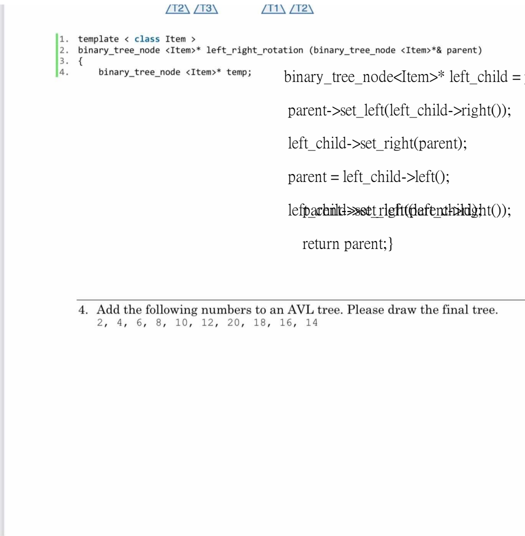


COEN 79

Object-Oriented Programming and Advanced Data Structures

3. 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.)

parent
c
a
T4
T1
b
rotateLeft(a)
c
b
T4
a
T3
rotateRight(c)
b
a
c
T1
T2
T3
T4



/T2\ /T3\ /T1\ /T2\

```
template < class Item >
binary_tree_node <Item>* left_right_rotation (binary_tree_node <Item>*& parent)
{
    binary_tree_node <Item>* temp;
```

binary_tree_node<Item>* left_child = parent->left();

parent->set_left(left_child->right());

left_child->set_right(parent);

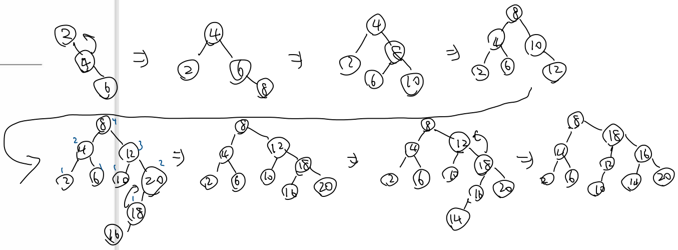
parent = left_child->left();

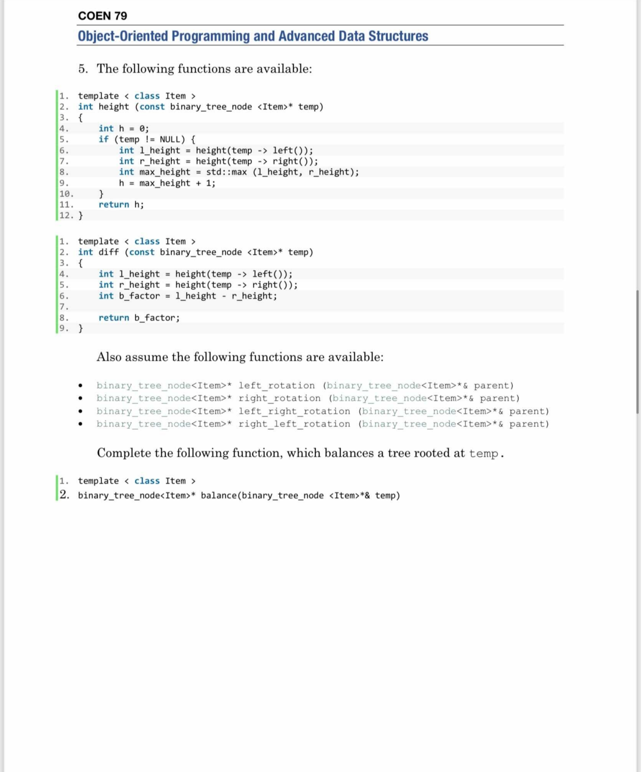
left_child->set_left(parent->right());

parent->set_right(left_child);

return parent;}

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





COEN 79

Object-Oriented Programming and Advanced Data Structures

5. The following functions are available:

```
template < class Item >
int height (const binary_tree_node <Item>* temp)
{
    int h = 0;
    if (temp != NULL) {
        int l_height = height(temp -> left());
        int r_height = height(temp -> right());
        int max_height = std::max (l_height, r_height);
        h = max_height + 1;
    }
    return h;
}
```

```
template < class Item >
int diff (const binary_tree_node <Item>* temp)
{
    int l_height = height(temp -> left());
    int r_height = height(temp -> right());
    int b_factor = l_height - r_height;

    return b_factor;
}
```

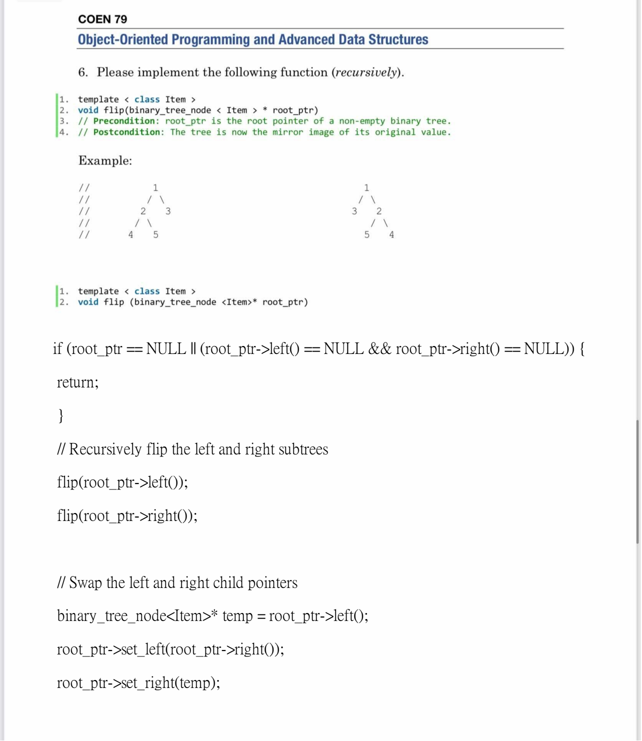
Also assume the following functions are available:

- `binary_tree_node<Item>* left_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* right_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* left_right_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* right_left_rotation (binary_tree_node<Item>*& parent)`

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

```
template < class Item >
binary_tree_node<Item>* balance(binary_tree_node <Item>*& temp)
```

```
int balance_factor = diff(temp);
 if (balance_factor > 1) {
     if (diff(temp->left()) > 0) {
            temp = right_rotation(temp);
     } else {
           temp = left_right_rotation(temp);
     }
 } else if (balance_factor < -1) {
       if (diff(temp->right()) < 0) {
            temp = left_rotation(temp);
 } else {
 temp = right_left_rotation(temp);
 } }return temp;
}
```



COEN 79

Object-Oriented Programming and Advanced Data Structures

6. Please implement the following function (*recursively*).

```
template < class Item >
void flip(binary_tree_node < Item > * root_ptr)
// Precondition: root_ptr is the root pointer of a non-empty binary tree.
// Postcondition: The tree is now the mirror image of its original value.
```

Example:

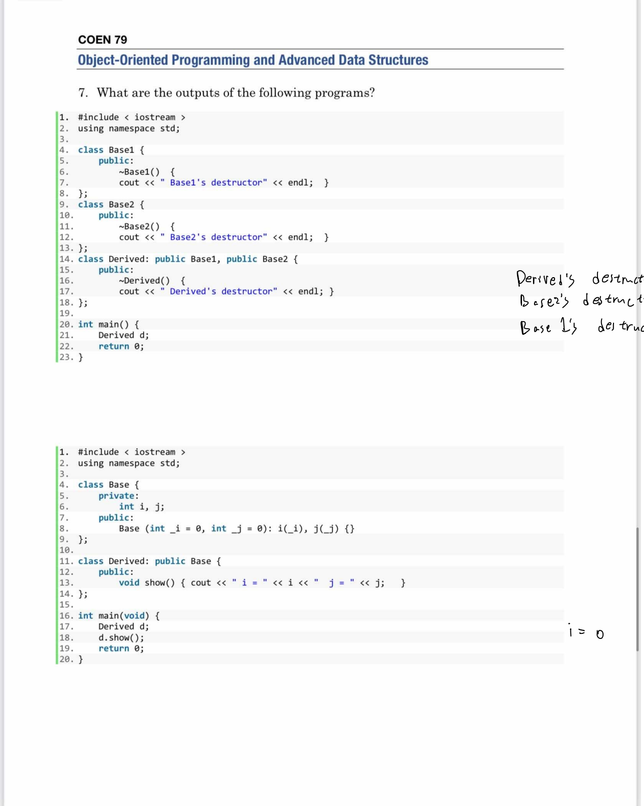
```
//          1                              1
//         / \                            / \
//        2   3                          3   2
//       / \                                / \
//      4   5                              5   4
```

```
template < class Item >
void flip (binary_tree_node <Item>* root_ptr)
```

```
if (root_ptr == NULL || (root_ptr->left() == NULL && root_ptr->right() == NULL)) {
 return;
 }
 // Recursively flip the left and right subtrees
 flip(root_ptr->left());
 flip(root_ptr->right());


 // Swap the left and right child pointers
 binary_tree_node<Item>* temp = root_ptr->left();
 root_ptr->set_left(root_ptr->right());
 root_ptr->set_right(temp);
```

```
}
```



COEN 79

Object-Oriented Programming and Advanced Data Structures

7. What are the outputs of the following programs?

```
#include < iostream >
using namespace std;

class Base1 {
    public:
        ~Base1()  {
        cout << " Base1's destructor" << endl;  }
};
class Base2 {
    public:
        ~Base2()  {
        cout << " Base2's destructor" << endl;  }
};
class Derived: public Base1, public Base2 {
    public:
        ~Derived()  {
        cout << " Derived's destructor" << endl; }
};

int main() {
    Derived d;
    return 0;
}
```

Derived's destructor
Base2's destructor
Base 1's destructor

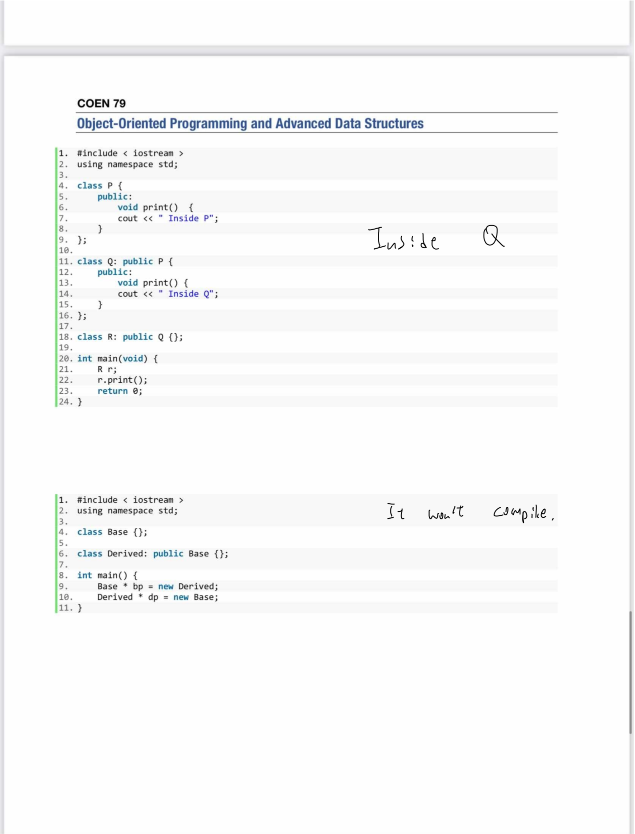
```
#include < iostream >
using namespace std;

class Base {
    private:
        int i, j;
    public:
        Base (int _i = 0, int _j = 0): i(_i), j(_j) {}
};

class Derived: public Base {
    public:
        void show() { cout << " i = " << i << "  j = " << j;   }
};

int main(void) {
    Derived d;
    d.show();
    return 0;
}
```

i = 0 j = 0



COEN 79

Object-Oriented Programming and Advanced Data Structures

```
#include < iostream >
using namespace std;

class P {
    public:
        void print()  {
        cout << " Inside P";
    }
};

class Q: public P {
    public:
        void print() {
        cout << " Inside Q";
    }
};

class R: public Q {};

int main(void) {
    R r;
    r.print();
    return 0;
}
```

Inside Q

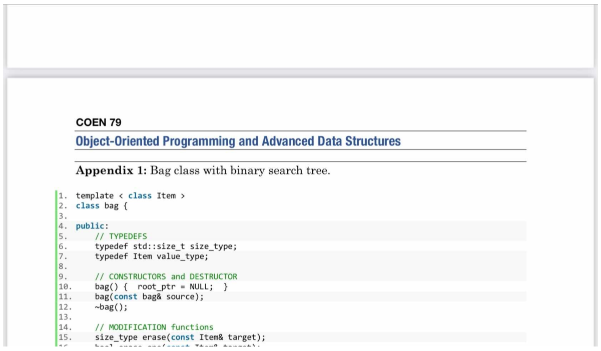
```
#include < iostream >
using namespace std;

class Base {};

class Derived: public Base {};

int main() {
    Base * bp = new Derived;
    Derived * dp = new Base;
}
```

It won't compile.



COEN 79

Object-Oriented Programming and Advanced Data Structures

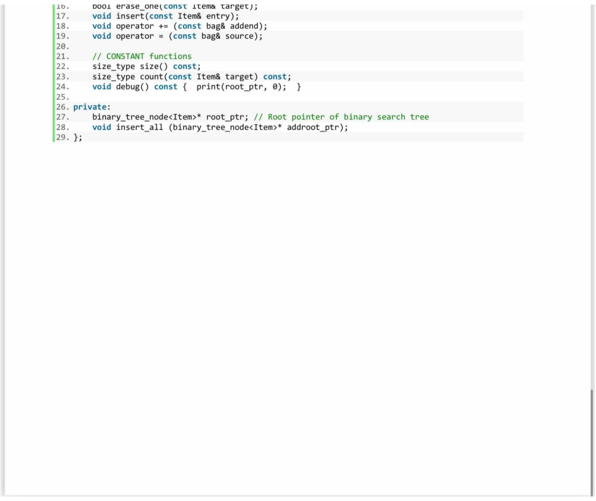
Appendix 1: Bag class with binary search tree.

```
template < class Item >
class bag {

public:
    // TYPEDEFS
    typedef std::size_t size_type;
    typedef Item value_type;

    // CONSTRUCTORS and DESTRUCTOR
    bag() {  root_ptr = NULL;  }
    bag(const bag& source);
    ~bag();

    // MODIFICATION functions
    size_type erase(const Item& target);
```



```
    bool erase_one(const Item& target);
    void insert(const Item& entry);
    void operator += (const bag& addend);
    void operator = (const bag& source);

    // CONSTANT functions
    size_type size() const;
    size_type count(const Item& target) const;
    void debug() const {  print(root_ptr, 0);  }

private:
    binary_tree_node<Item>* root_ptr; // Root pointer of binary search tree
    void insert_all (binary_tree_node<Item>* addroot_ptr);
};
```