COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #6 – Practice for final, no submission

Name: **Date:**

1. Please complete the following implementation:

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

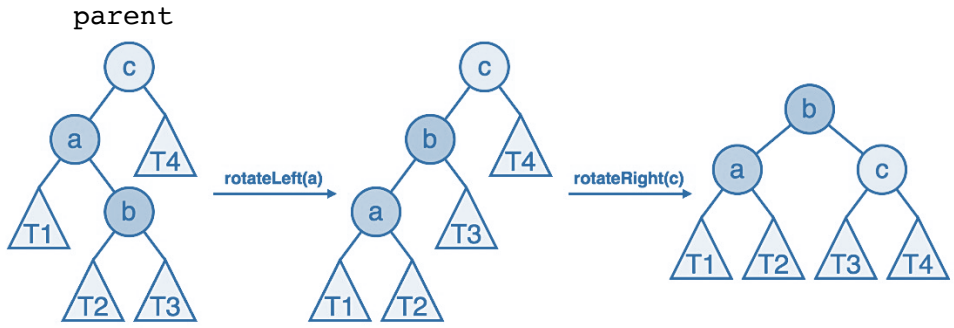
- Using the previous implementation, complete the following function for the bag class given in Appendix 1:

```
1. template < class Item >
2. void bag <Item>::operator = (const bag<Item>& source)
3. // Header file used: bintree.h
```

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

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

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.)




```
1.  template < class Item >
2.  binary_tree_node <Item>* left_right_rotation (binary_tree_node <Item>*& parent)
3.  {
4.      binary_tree_node <Item>* temp;
```

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

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)
```

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:


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

7. What are the outputs of the following programs?

```
1.  #include < iostream >
2.  using namespace std;
3.
4.  class Base1 {
5.      public:
6.          ~Base1()  {
7.          cout << " Base1's destructor" << endl;  }
8.  };
9.  class Base2 {
10.     public:
11.         ~Base2()  {
12.         cout << " Base2's destructor" << endl;  }
13. };
14. class Derived: public Base1, public Base2 {
15.     public:
16.         ~Derived()  {
17.         cout << " Derived's destructor" << endl; }
18. };
19.
20. int main() {
21.     Derived d;
22.     return 0;
23. }
```

```
1.  #include < iostream >
2.  using namespace std;
3.
4.  class Base {
5.      private:
6.          int i, j;
7.      public:
8.          Base (int _i = 0, int _j = 0): i(_i), j(_j) {}
9.  };
10.
11. class Derived: public Base {
12.     public:
13.         void show() { cout << " i = " << i << "  j = " << j;   }
14. };
15.
16. int main(void) {
17.     Derived d;
18.     d.show();
19.     return 0;
20. }
```

```cpp
#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;
}
```

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

class Base {};

class Derived: public Base {};

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

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);
};
```