```
//
//  main.cpp
//  Coen70HW3.6 *Chapter 5 #17
//
//  Created by Yousef Zoumot on 2/3/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <algorithm>
#include <iostream>
#include <cassert>

using namespace std;

//*********************************************************************//
class Node{
private:
    int _data;
    int _key;
    Node* _next;
    Node* _prev;
public:
    Node(const int& = int(), Node* = NULL);
    int& data(){return _data;}
    Node*& next(){return _next;}
};
//*********************************************************************//

Node* location(Node* front_ptr, size_t position);

//*********************************************************************//

class bag{
public:
    //*********************************************************************//
    bag(){front = NULL; back= NULL; used = 0;}
    bag(const bag& source);
    ~bag(){deleteList(front);}
    //*********************************************************************//
    bool erase_one(const int& target);
    bool contains(const int& target);
    void insert(const int& data);
    int size() const { return used; }
    int count( const int& target);
    void printValues();
    //*********************************************************************//
    bag& operator =(const bag& source);
    void operator -=(const bag& removeIt);
    void operator +=(const bag& addend);
    //*********************************************************************//
    void list_insert(Node*& previous_ptr, const int& data);
    Node* list_search(Node* front_ptr, const int& target);
    void insertAtFront(Node*& front_ptr, Node*& back_ptr, const int& data);
    void list_copy(Node* source_ptr, Node*& front_ptr, Node*& back_ptr);
    void remove(Node*& front_ptr);
    void removeNode(Node*& previous_ptr);
    void deleteList(Node*& front_ptr);
    int grab() const;
private:
    Node* front;
    Node* back;
```

```
    int used;
    void incSize();
};

bag operator +(const bag& b1, const bag& b2);
bag operator -(const bag& source1, const bag& source2);



int main(){
    bag x;
    bag y;
    bag z;
    x.insert(1);
    x.insert(2);
    x.insert(3);
    x.insert(4);
    x.insert(5);
    y.insert(5);
    y.insert(6);
    y.insert(7);
    x.printValues();
    y.printValues();
    x += y;
    z = y;
    x.printValues();
    z.printValues();
    x -= y;
    x.printValues();
    x.erase_one(3);
    x.erase_one(4);
    x.printValues();
}
//*********************************************************************//

    //*********************************************************************//
    Node:: Node(const int& data, Node* next){
        _data = data;
        _next = next;
    }
    //*********************************************************************//
    bag& bag:: operator=(const bag& source){
        if(this == &source)
            return *this;
        deleteList(front);
        used = 0;
        if(source.used == 0){
            used = 0;
            front = NULL;
            back = NULL;
            return *this;
        }
        Node* temp = source.front;
        insert(temp->data());
        temp = temp->next();
        while(temp != source.front){
            insert(temp->data());
            temp = temp->next();
        }
        used = source.used;
        return *this;
```

```
    }
    //*********************************************************************//
    bag:: bag(const bag& source){
        Node* back_ptr;
        list_copy(source.front, front, back_ptr);
    }
    //*********************************************************************//
    void bag::remove(Node*& front_ptr){
        Node* temp = front;
        front_ptr = front_ptr->next();
        delete temp;
        used--;
        return;
    }
    //*********************************************************************//
    void bag::list_insert(Node*& previous_ptr, const int& data){
        Node* insert_ptr = new Node;
        insert_ptr->data() = data;
        insert_ptr->next() = previous_ptr->next();
        previous_ptr->next() = insert_ptr;

        previous_ptr = insert_ptr;

    }
    //*********************************************************************//
    void bag::insertAtFront(Node*& front_ptr, Node*& back_ptr, const int&
data){
        front_ptr = new Node(data, front);
        back_ptr = front;

    }
    //*********************************************************************//
    void bag:: deleteList(Node*& front_ptr){
        while(used != 0)
            remove(front_ptr);
    }
    //*********************************************************************//
    void bag:: removeNode(Node*& previous_ptr){
        Node *temp;
        temp = previous_ptr->next();
        previous_ptr->next() = temp->next();
        delete temp;
    }
    //*********************************************************************//
    Node* bag::list_search(Node* front_ptr, const int& target){
        Node* cursor;
        for(cursor = front_ptr; cursor ->next() != NULL; cursor = cursor-
>next())
            if(target == cursor->next()->data())
                return cursor;
        return NULL;
    }
    //*********************************************************************//

    Node* location(Node* front_ptr, size_t position){
        assert(position>0);
        Node* cursor;
        cursor = front_ptr;
        for(size_t i = 1; (i < position) && (cursor != NULL); ++i)
            cursor = cursor->next();
        return cursor;
```

```
}

//*********************************************************************//
void bag::list_copy(Node* source_ptr, Node*& front_ptr, Node*& back_ptr){
    front_ptr = NULL;
    back_ptr = NULL;
    if(source_ptr == NULL)
        return;
    Node* temp = source_ptr;
    insertAtFront(front_ptr, back_ptr, source_ptr->data());
    temp = temp -> next();
    while(temp){
        list_insert(back_ptr, temp->data());
        temp = temp->next();
    }
}
//*********************************************************************//
bool bag::erase_one(const int& target){
    Node* cursor = front;
    Node* prev = back;
    if (cursor == NULL)
        return false;
    if(cursor->data() == target){
        prev->next() = cursor->next();
        free(cursor);
        front = prev->next();
        used--;
        return true;
    }
    else{
        prev = cursor;
        cursor = cursor->next();

        while(cursor != front){
            if(cursor->data() == target){
                if(cursor==back){
                    back=prev;
                }
                if(cursor==front){
                    front=cursor->next();
                }
                prev->next() = cursor->next();
                delete cursor;
                used--;
                return true;
            }
            else{
                prev = cursor;
                cursor = cursor->next();
            }
        }
        return false;
    }

}
//*********************************************************************//
void bag::insert(const int& data){
    if(used == 0){
        insertAtFront(front, back, data);
        front -> next() = front;
    }
```

```
    else{
        list_insert(back, data);
    }
    used++;
    return;

}
//*********************************************************************//
void bag:: operator -=(const bag& removeIt){
    Node* cursor = removeIt.front;
    erase_one(cursor->data());
    cursor = cursor->next();
    while(cursor != removeIt.front){
        erase_one(cursor->data());
        cursor = cursor->next();
    }
}
//*********************************************************************//
void bag::operator +=(const bag& addend){
    Node* temp = addend.front;
    insert(temp -> data());
    temp = temp->next();
    while(temp != addend.front){
        insert(temp->data());
        temp = temp->next();
    }
}
//*********************************************************************//
int bag::count(const int& target) {
    int answer;
    Node* cursor;
    answer = 0;
    cursor = list_search(front, target);
    while(cursor != NULL){
        answer++;
        cursor = cursor->next();
        cursor = list_search(cursor, target);
    }
    return answer;
}
//*********************************************************************//
int bag::grab() const{
    int i;
    Node* cursor;
    assert(size() > 0);
    i = (rand() % size()) + 1;
    cursor = location(front, i);
    return cursor->data();
}
//*********************************************************************//
bag operator -(const bag& source1, const bag& source2){
    bag answer;
    answer = source1;
    answer -= source2;
    return answer;
}

//*********************************************************************//
bag operator +(const bag& b1, const bag& b2){
    bag answer;
    answer += b1;
```

```
    answer += b2;
    return answer;
}
//*********************************************************************//
void bag::printValues(){
    Node* cursor = front;
    while(cursor){
        cout << cursor->data() << ", ";
        cursor = cursor->next();
        if(cursor == front){
            break;
        }
    }
    cout << endl;
    cout << "front: " << front->data() << endl;
    cout << "back: " << back->data() << endl;
    cursor = front;
    cout << endl << endl << endl;
}
//*********************************************************************//\
```

```
1, 2, 3, 4, 5,
front: 1
back: 5



5, 6, 7,
front: 5
back: 7



1, 2, 3, 4, 5, 5, 6, 7,
front: 1
back: 7



5, 6, 7,
front: 5
back: 7



1, 2, 3, 4, 5,
front: 1
back: 5



1, 2, 5,
front: 1
back: 5
```