```
//  Yousef Zoumot
//  main.cpp
//  Coen70HW4.1 *Chapter 6 Problem 2a
//
//  Created by Yousef Zoumot on 2/14/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <cassert>
#include <vector>
using namespace std;


template < class T > class set{
public:
    set(T x = 20);
    set(const set& source);
    ~set();
    T erase(const T& target);
    bool erase_one(const T& target);
    void insert(const T& target);
    set operator -(const set& b2);
    set& operator =(const set& source);
    void operator -=(const set& removeIt);
    void operator +=(const set& addend);
    set operator +(const set& b2);
    bool contains(const T& target) const;
    T size() const { return used; }
    T count( const T& target) const;
    void prT();

private:
    T* data;
    T capacity;
    void incSize();
    T used;
};



int main(){
    set<int> a;
    set<int> b;
    set<int> c;
    set<int> d;
    a.insert(2);
    a.insert(2);
    a.insert(4);
    a.insert(5);
```

```
    a.prT();
    b.insert(4);
    b.insert(2);
    b.insert(6);
    b.prT();
    c = a - b;
    c.prT();
    c = a + b;
    c.prT();
    d.insert(7);
    c += d;
    c.prT();
    c -= d;
    c.prT();
    c.erase_one(3);
    c.prT();
}



template<class T>
T set<T>::erase(const T& target){
    T index = 0;
    T many_removed = 0;

    while(index < used){
        if (data[index] == target){
            --used;
            data[index] = data [used];
            ++many_removed;
        }
        else
            ++index;
    }

    return many_removed;

}

template<class T>
set<T>:: set(T x){
    assert(x>0);
    used = 0;
    capacity = x;
    data = new T[x];
}

template<class T>
set<T>:: set(const set& source){
```

```
    data = NULL;
    *this = source;
}
template<class T>
set<T>:: ~set(){
    if (data)
        delete[] data;
}
template<class T>
void set<T>:: incSize(){
    T* temp = new T[2*capacity];
    for(T i = 0; i < capacity; i++){
        temp[i] = data[i];
    }
    delete[] data;
    data = temp;
    capacity *= 2;
}
template<class T>
void set<T>::prT(){
    T i;
    for(i = 0; i < used; i++){
        cout << data[i] << ", ";
    }
    cout << endl;
}
template<class T>
bool set<T>::erase_one(const T& target){
    T index;
    index = 0;
    while((index < used) && (data[index] != target))
        ++index;
    if(index == used)
        return false;
    --used;
    data[index] = data[used];
    return true;
}

template<class T>
void set<T>::insert(const T& entry){
    if(contains(entry))
        return;
    if(size() >= capacity)
        incSize();
    data[used] = entry;
    ++used;
    return;
}
template<class T>
```

```
void set<T>::operator +=(const set& addend){
    T i;
    if(size() + addend.size() >= capacity)
        incSize();
    for(i = 0; i < addend.used; i++){
        if(!contains(addend.data[i])){
            data[used] = addend.data[i];
            used++;
        }
    }
}

template<class T>
set<T> set<T>:: operator -(const set& b2){
    set answer = *this;
    for(T i = 0; i < b2.used; i++)
        answer.erase_one(b2.data[i]);
    return answer;
}
template<class T>
void set<T>:: operator -=(const set& removeIt){
    T i;
    for(i = 0; i < removeIt.used; i++)
        erase_one(removeIt.data[i]);
}
template<class T>
T set<T>::count(const T& target) const {
    T answer;
    T i;
    answer = 0;
    for(i = 0; i < used; ++i)
        if (target == data[i])
            ++answer;
    return answer;
}
template<class T>
set<T>& set<T>:: operator =(const set& source){
    if(this == &source)
        return *this;
    if (data)
        delete[] data;
    if(source.used == 0){
        used = 0;
        capacity = 20;
        data = new T[capacity];
        return *this;
    }
    data = new T[source.capacity];
    for(T i = 0; i < source.capacity; i++){
        data[i] = source.data[i];
```

```
    }
    used = source.used;
    capacity = source.capacity;
    return  *this;
}
template<class T>
set<T> set<T>::operator +(const set& b2){
    set answer = *this;
    if(answer.size() + b2.size() >= capacity)
        incSize();
    for(T i = 0; i < b2.used; i++){
        if(!answer.contains(b2.data[i])){
            answer.data[used] = b2.data[i];
            answer.used++;
        }
    }
    return answer;


}
template<class T>
bool set<T>:: contains(const T& target) const{
    T i;
    for(i = 0; i < used; ++i)
        if (target == data[i])
            return true;
    return false;

}
```

```
2, 4, 5,
4, 2, 6,
5,
2, 4, 5, 6,
2, 4, 5, 6, 7,
2, 4, 5, 6,
2, 4, 5, 6,
Program ended with exit code: 0
```

```
// Yousef Zoumot
//  main.cpp
//  Coen70HW4.2 Chapter 6 Problem 2b
//
//  Created by Yousef Zoumot on 2/14/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <assert.h>
#include <cstdlib>//Provides size_t

using namespace std;

template <class T>
class sequence{
public:
    //TYPEDEFS and MEMBER CONSTANTS
    typedef std::size_t size_type;
    static const size_type CAPACITY=30;
    //CONSTRUCTOR
    sequence();
    //MODIFICATION MEMBER FUNCTIONS
    void start();
    void advance();
    void insert(const T& entry);
    void attach(const T& entry);
    void remove_current();

    void addToFront(const T& entry);
    void removeFront();
    void addToEnd(const T& entry);
    void lastToCurrent();

    sequence operator +(const sequence& s2);
    void operator +=(const sequence& s2);

    T operator[](size_type index);

    void printValues();

    //CONSTANT MEMBER FUNCTIONS
    size_type size() const;
    bool is_item() const;
    T current() const;
private:
    T data[CAPACITY];
    size_type used;
    size_type current_index;
};
```

```
int main(int argc, const char * argv[]) {
    // insert code here...
    sequence<int> s1;
    sequence<int> s2;
    s1.addToEnd(1);
    s1.addToEnd(2);
    s1.addToEnd(3);
    s1.addToEnd(4);
    s1.addToEnd(5);
    s2.addToEnd(6);
    s2.addToEnd(7);
    s2.addToEnd(8);
    s2.addToEnd(9);
    s1.printValues();
    s2.printValues();
    sequence<int> s3;
    s3= s1+s2;
    s3.printValues();
    sequence<int> s4;
    s4+=s1;
    s4+=s2;
    s4.printValues();
    cout<<s4[0];

    return 0;
}

// MODIFICATION MEMBER FUNCTIONS
template <class T>
sequence<T>::sequence ()
{
    current_index = 0;
    used = 0;
}
template <class T>
void sequence<T>::start( )
{
    current_index = 0;
}
template <class T>
void sequence<T>::advance( )
{
    current_index++;
}
template <class T>
void sequence<T>::insert(const T& entry)
{
    if(current_index==used){
        data[current_index]=entry;
```

```
        used++;
        return;
    }
    size_type i;
    for (i = used; i > current_index; i--)
        data[i]= data[i-1];

    data[current_index] = entry;
    used++;
}
template <class T>
void sequence<T>::attach(const T& entry)
{
    if(!is_item()){
        data[current_index]=entry;
        used++;
        return;
    }
    size_type i;
    for (i = used; i > current_index+1; i--)
        data[i] = data[i+1];

    data[current_index+1] = entry;
    current_index++;
    used++;
}
template <class T>
void sequence<T>::remove_current( )
{
    size_type i;
    for (i= current_index; i < used-1; i++)
        data[i] = data[i+1];
    used--;
}
template <class T>
void sequence<T>:: addToFront(const T& entry){

    if(current_index==used){
        data[current_index]=entry;
        used++;
        return;
    }
    size_type i;
    for (i = used; i > 0; i--)
        data[i]= data[i-1];

    data[0] = entry;
    start();
    used++;
}
```

```
template <class T>
void sequence<T>:: removeFront(){
    start();
    remove_current();
}
template <class T>
void sequence<T>:: addToEnd(const T& entry){
    current_index=used;
    data[current_index]=entry;
    used++;
}
template <class T>
void sequence<T>:: lastToCurrent(){
    data[current_index]=data[used-1];
    used--;
}
template <class T>
T sequence<T>:: operator[](size_type index){
    T invalid=100000;
    if(index<size())
        return data[index];
    else{
        cout<<"This is not a valid index";
        return invalid;
    };
}
template <class T>
sequence<T> sequence<T>:: operator +(const sequence& s2){
    sequence temp;
    size_type i=0;
    size_type f=0;
    while(temp.size() < size()){
        temp.data[i]=data[i];
        i++;
        temp.used++;
    }
    while (temp.size() < (size()+s2.size())) {
        temp.data[i]=s2.data[f];
        f++;
        i++;
        temp.used++;
    }
    return temp;
}
template <class T>
void sequence<T>:: operator +=(const sequence& s2){
    *this=*this+s2;

}
template <class T>
```

```
void sequence<T>:: printValues(){
    cout<<"The values in the sequence are as follows: "<<"\n";
    size_type i;
    for(i=0; i<size(); i++)
        cout<<data[i]<<" \n";
}

// CONSTANT MEMBER FUNCTIONS
template <class T>
size_t sequence<T>::size( ) const
{
    return used;
}
template <class T>
bool sequence<T>::is_item( ) const
{
    return current_index != used;
}
template <class T>
T sequence<T>::current( ) const
{
    return data[current_index];
}
```

```
The values in the sequence are as follows:
1
2
3
4
5
The values in the sequence are as follows:
6
7
8
9
The values in the sequence are as follows:
1
2
3
4
5
6
7
8
9
The values in the sequence are as follows:
1
2
3
4
5
6
7
8
9
1Program ended with exit code: 0
```

```
//  Yousef Zoumot
//  main.cpp
//  Coen70HW4.3 *Chapter 6 Problem 2e
//
//  Created by Yousef Zoumot on 2/14/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <cassert>
#include <cstdlib> //provide size_t

using namespace std;

template<class T>
class Keyed_Bag
{
public:
    //CONSTRUCTOR
    Keyed_Bag();
    //MODIFICATION
    bool erase_one(const T& target);
    void insert(const T& entry, T key);
    void operator +=(const Keyed_Bag& addend);
    Keyed_Bag operator -(const Keyed_Bag& b);
    void operator -=(const Keyed_Bag& remove);
    //CONSTANT MEMBER FUNCTIONS
    T size() const { return used;}
    T count(const T& target) const;
    void prTValues();
private:
    struct Node{
        Node* _prev;
        Node* _next;
        T _data;
        T _key;
        Node(T data, T key, Node* prev = NULL, Node* next =
NULL){
            this->_data = data;
            this->_key=key;
            this->_prev = prev;
            this->_next = next;
        }
        T& data(){return _data;};
        Node*& next(){return _next;};
        Node*& prev(){return _prev;};

    };
    Node* head;
    T used;            //How much of the array is used
```

```
};

template<class T>
Keyed_Bag<T>::Keyed_Bag(){
    head=NULL;
    used=0;
}

//NONMEMBER FUNCTIONS for the Keyed_Bag class
//Keyed_Bag operator +(const Keyed_Bag& b1, const Keyed_Bag& b2);



template<class T>
bool Keyed_Bag<T>::erase_one(const T& key1){
    Node* tmp=head;
    while(tmp->_next!=NULL && tmp->_key != key1)
        tmp=tmp->_next;
    if(tmp->_next==NULL)
        return false;
    --used;
    if(tmp->_prev!=NULL)
        tmp->_prev->_next=tmp->_next;
    if(tmp->_prev==NULL)
        head=tmp->_next;
    delete tmp;
    return true;
}

template<class T>
void Keyed_Bag<T>::insert(const T& entry, T key){
    Node* tmp=new Node(entry, key);
    Node* dummy=head;
    Node* mummy=head;
    if(head==NULL){
        head=tmp;
        return;
    }
    while(mummy!=NULL){

        if(mummy->_key==tmp->_key)
            return;
        mummy=mummy->_next;
    }
    while(dummy->_next!=NULL){
        dummy=dummy->_next;
    }
    dummy->_next=tmp;
    tmp->_prev=dummy;
    tmp->_next=NULL;
```

```
    ++used;
    return;
}

template<class T>
void Keyed_Bag<T>:: prTValues(){//a function that prTs all the
values in order to clean up the main function
    Node* tmp=head;
    cout<<"\n";
    while(tmp->_next!=NULL){
        cout<<"data: "<<tmp->_data<<"  with key: "<<tmp-
>_key<<"\n";
        tmp=tmp->_next;
    }
}



int main(int argc, const char * argv[]) {
    // insert code here...

    Keyed_Bag<int> b, b2;
    b.insert(1,1);
    b.insert(2,2);
    b.insert(3,3);
    b.insert(4,4);
    b.insert(3,5);
    b.insert(7,4);
    b.insert(8,5);
    b.insert(9,6);

    b2.insert(3,6);
    b2.insert(7,7);
    b2.insert(2,2);
    b2.insert(3,3);
    b2.insert(3,7);
    b.prTValues();
    b2.prTValues();

    return 0;
}
```

```
data: 1  with key: 1
data: 2  with key: 2
data: 3  with key: 3
data: 4  with key: 4
data: 3  with key: 5

data: 3  with key: 6
data: 7  with key: 7
data: 2  with key: 2
Program ended with exit code: 0
```

```
//
//  main.cpp
//  Coen70HW4.4
//
//  Created by Yousef Zoumot on 2/14/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//



#include <iostream>
#include <cassert>
#include <cstdlib> //provide size_t
#include <utility>

using namespace std;

template<class T, class K>
class Keyed_Bag
{
public:
    //CONSTRUCTOR
    Keyed_Bag();
    //MODIFICATION
    bool erase_one(const T& target);
    void insert(const T& entry, T key);
    void operator +=(const Keyed_Bag& addend);
    Keyed_Bag operator -(const Keyed_Bag& b);
    void operator -=(const Keyed_Bag& remove);
    //CONSTANT MEMBER FUNCTIONS
    T size() const { return used;}
    T count(const T& target) const;
    void prTValues();
private:
    struct Node{
        Node* _prev;
        Node* _next;
        T _data;
        K _key;
        Node(T data, K key, Node* prev = NULL, Node* next =
NULL){
            this->_data = data;
            this->_key=key;
            this->_prev = prev;
            this->_next = next;
        }
        T& first(){return _data;};
        K& second(){return _key;};
        Node*& next(){return _next;};
        Node*& prev(){return _prev;};
```

```
    };
    Node* head;
    T used;             //How much of the array is used
};

template<class T, class K>
Keyed_Bag<T,K>::Keyed_Bag(){
    head=NULL;
    used=0;
}

//NONMEMBER FUNCTIONS for the Keyed_Bag class
//Keyed_Bag operator +(const Keyed_Bag& b1, const Keyed_Bag& b2);



template<class T, class K>
bool Keyed_Bag<T,K>::erase_one(const T& key1){
    Node* tmp=head;
    while(tmp->_next!=NULL && tmp->_key != key1)
        tmp=tmp->_next;
    if(tmp->_next==NULL)
        return false;
    --used;
    if(tmp->_prev!=NULL)
        tmp->_prev->_next=tmp->_next;
    if(tmp->_prev==NULL)
        head=tmp->_next;
    delete tmp;
    return true;
}

template<class T, class K>
void Keyed_Bag<T,K>::insert(const T& entry, T key){
    Node* tmp=new Node(entry, key);
    Node* dummy=head;
    Node* mummy=head;
    if(head==NULL){
        head=tmp;
        return;
    }
    while(mummy!=NULL){

        if(mummy->_key==tmp->_key)
            return;
        mummy=mummy->_next;
    }
    while(dummy->_next!=NULL){
        dummy=dummy->_next;
```

```
    }
    dummy->_next=tmp;
    tmp->_prev=dummy;
    tmp->_next=NULL;
    ++used;
    return;
}

template<class T, class K>
void Keyed_Bag<T,K>:: prTValues(){//a function that prTs all the
values in order to clean up the main function
    Node* tmp=head;
    cout<<"\n";
    while(tmp->_next!=NULL){
        cout<<"data: "<<tmp->_data<<"  with key: "<<tmp-
>_key<<"\n";
        tmp=tmp->_next;
    }
}



int main(int argc, const char * argv[]) {
    // insert code here...

    Keyed_Bag<int, double> b, b2;
    b.insert(1,1);
    b.insert(2,2);
    b.insert(3,3);
    b.insert(4,4);
    b.insert(3,5);
    b.insert(7,4);
    b.insert(8,5);
    b.insert(9,6);

    b2.insert(3,6);
    b2.insert(7,7);
    b2.insert(2,2);
    b2.insert(3,3);
    b2.insert(3,7);
    b.prTValues();
    b2.prTValues();

    return 0;
}
```

```
data: 1  with key: 1
data: 2  with key: 2
data: 3  with key: 3
data: 4  with key: 4
data: 3  with key: 5

data: 3  with key: 6
data: 7  with key: 7
data: 2  with key: 2
Program ended with exit code: 0
```

```
//  Yousef Zoumot
//  main.cpp
//  Coen70HW4.5 Chapter 6 Problem 8
//
//  Created by Yousef Zoumot on 2/14/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//



#include <iostream>
#include <cassert>
#include <cstdlib> //provide size_t
#include <utility>

using namespace std;

class Gift{
    char _gift[40];
public:
    void typeGift();
    void printGift();
};

class Person{
private:
    char name[40];
public:
    Person(){used_g = 0;};
    Gift gifts[100];
    void addGift(Gift& g);
    void typeName();
    void printName();
    int used_g;
};

class Gift_List{
    Person people[100];
    int used_p;
public:
    Gift_List(){used_p=0;};
    void addPerson(Person& p);
    void removeLast();
    void printList();
};

void Gift_List:: printList(){
    cout<<"The list is as follows: "<<"\n";
    for(int i=0; i<used_p; i++){
        people[i].printName();
```

```
        cout<< " has a gift list that consists of: "<<"\n";
        for(int k=0; k<people[i].used_g; k++ ){
            people[i].gifts[k].printGift();
            cout<<"\n";
        }
    }

}

void Gift_List:: removeLast(){
    used_p--;
}

void Gift_List:: addPerson(Person& p){
    people[used_p]=p;
    used_p++;
}
void Person:: addGift(Gift& g){
    gifts[used_g]=g;
    used_g++;
}

void Gift:: typeGift(){
    cout<<"Please type a gift less that 40 characters long: "<<
"\n";
    cin>>_gift;
}

void Gift:: printGift(){
    cout<<_gift;
}

void Person:: printName(){
    cout<<name;

}

void Person:: typeName(){
    cout<<"Please type a name less that 40 characters long: "<<
"\n";
    cin>>name;
}

int main(int argc, const char * argv[]) {
    // insert code here...
    Person p1, p2, p3;
    p1.typeName();
    p2.typeName();
    p3.typeName();
    Gift g1, g2, g3, g4, g5, g6;
```

```
    g1.typeGift();
    g2.typeGift();
    g3.typeGift();
    g4.typeGift();
    g5.typeGift();
    g6.typeGift();
    p1.addGift(g1);
    p1.addGift(g4);
    p2.addGift(g2);
    p2.addGift(g5);
    p3.addGift(g3);
    p3.addGift(g6);

    Gift_List gl;
    gl.addPerson(p1);
    gl.addPerson(p2);
    gl.addPerson(p3);
    gl.printList();
    gl.removeLast();
    gl.printList();
    return 0;
}
```

```
Please type a name less that 40 characters long:
Crystal
Please type a name less that 40 characters long:
Ivanna
Please type a name less that 40 characters long:
John
Please type a gift less that 40 characters long:
Book
Please type a gift less that 40 characters long:
Pencil
Please type a gift less that 40 characters long:
Car
Please type a gift less that 40 characters long:
Necklace
Please type a gift less that 40 characters long:
Puppy
Please type a gift less that 40 characters long:
Xbox
The list is as follows:
Crystal has a gift list that consists of:
Book
Necklace
Ivanna has a gift list that consists of:
Pencil
Puppy
John has a gift list that consists of:
Car
Xbox
The list is as follows:
Crystal has a gift list that consists of:
Book
Necklace
Ivanna has a gift list that consists of:
Pencil
Puppy
Program ended with exit code: 0
```