// 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{

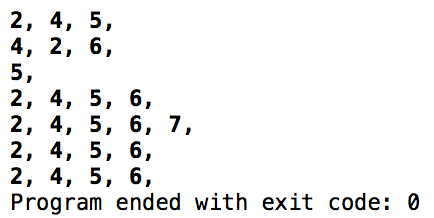
T i;

for(i = 0; i < used; ++i)

if (target == data[i])

return true;

return false;

}

// 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;

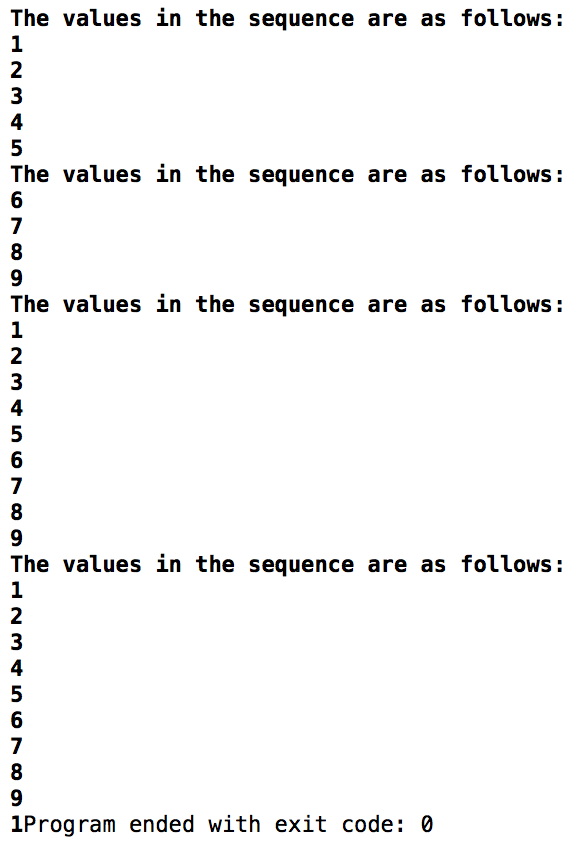
}

template <class T>

T sequence<T>::current( ) const

{

return data[current\_index];

}

// 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);

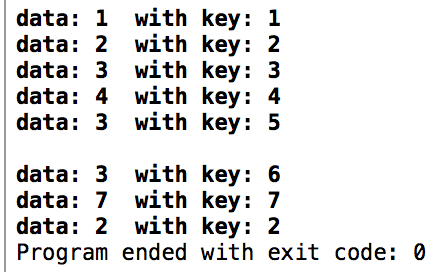
b2.insert(2,2);

b2.insert(3,3);

b2.insert(3,7);

b.prTValues();

b2.prTValues();

 return 0;

}

//

// 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);

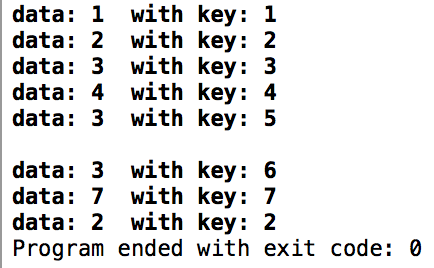
b2.insert(3,6);

b2.insert(7,7);

b2.insert(2,2);

b2.insert(3,3);

b2.insert(3,7);

 b.prTValues();

b2.prTValues();

return 0;

}

// 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);

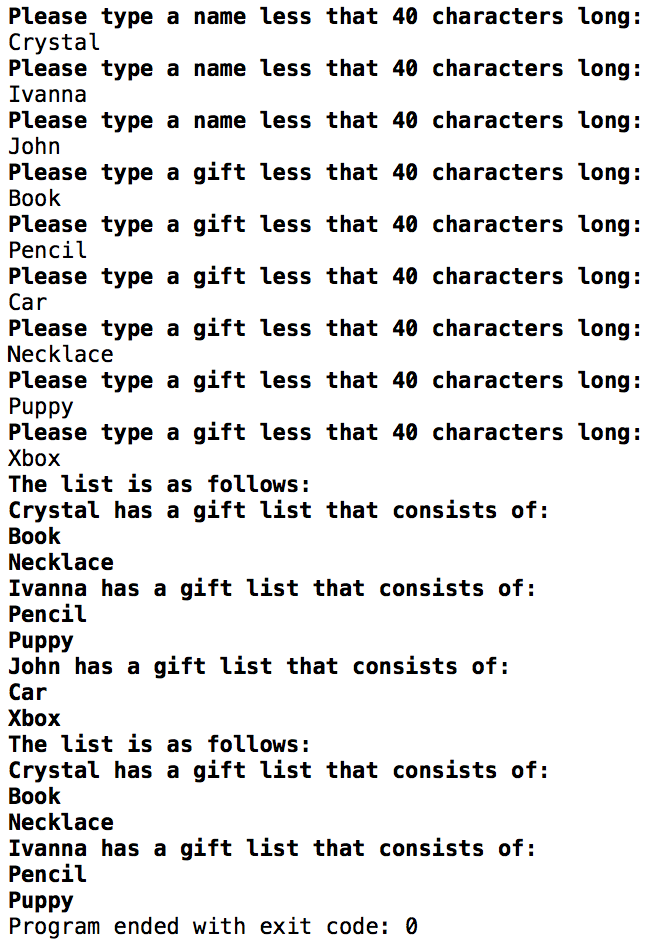
p2.addGift(g2);

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;

}