// Yousef Zoumot

// main.cpp

// Coen70HW2.1 \*Chapter 3 Problem 4

//

// Created by Yousef Zoumot on 1/18/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef double value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

sequence();

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const value\_type& entry);

void attach(const value\_type& entry);

void remove\_current();

void addToFront(const value\_type& entry);

void removeFront();

void addToEnd(const value\_type& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

value\_type operator[](size\_type index);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

value\_type current() const;

private:

value\_type data[CAPACITY];

size\_type used;

size\_type current\_index;

};

int main(int argc, const char \* argv[]) {

// insert code here...

sequence s1, 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 s3;

s3= s1+s2;

s3.printValues();

sequence s4;

s4+=s1;

s4+=s2;

s4.printValues();

cout<<s4[0];

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

sequence::sequence ()

{

current\_index = 0;

used = 0;

}

void sequence::start( )

{

current\_index = 0;

}

void sequence::advance( )

{

current\_index++;

}

void sequence::insert(const value\_type& 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++;

}

void sequence::attach(const value\_type& 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++;

}

void sequence::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

void sequence:: addToFront(const value\_type& 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++;

}

void sequence:: removeFront(){

start();

remove\_current();

}

void sequence:: addToEnd(const value\_type& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

void sequence:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

double sequence:: operator[](size\_type index){

value\_type invalid=100000;

if(index<size())

return data[index];

else{

cout<<"This is not a valid index";

return invalid;

};

}

sequence sequence:: 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;

}

void sequence:: operator +=(const sequence& s2){

\*this=\*this+s2;

}

void sequence:: 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

sequence::size\_type sequence::size( ) const

{

return used;

}

bool sequence::is\_item( ) const

{

return current\_index != used;

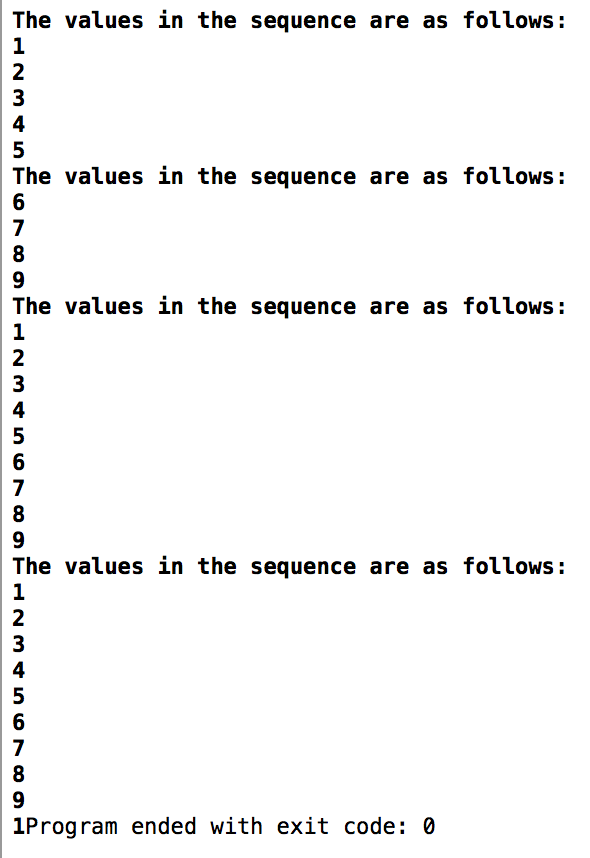
}

sequence::value\_type sequence::current( ) const

{

return data[current\_index];

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.2 \*Chapter 3 Problem 5

//

// Created by Yousef Zoumot on 1/12/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class set

{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

set() {used = 0;}

//MODIFICATION

size\_type erase (const value\_type& target);

bool erase\_one(const value\_type& target);

void insert(const value\_type&entry);

void operator +=(const set& addend);

set operator -(const set& b);

void operator -=(const set& remove);

//CONSTANT MEMBER FUNCTIONS

size\_type size() const { return used;}

size\_type count(const value\_type& target) const;

void printValues();

bool contains(const value\_type& target);

set operator +(const set& b2);

private:

value\_type data[CAPACITY]; //the array to store items

size\_type used; // much of the array is used

};

//NONMEMBER FUNCTIONS for the set class

set operator +(const set& b1, const set& b2);

const set:: size\_type set::CAPACITY;

bool set:: contains(const value\_type& target){

for(size\_type i=0; i<size(); i++){

if(data[i]== target)

return true;

}

return false;

}

set::size\_type set::erase(const value\_type& target){

size\_type index = 0;

size\_type many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bool set::erase\_one(const value\_type& target){

size\_type index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

void set::insert(const value\_type& entry){

assert(size() < CAPACITY);

if(contains(entry)){

return;

}

data[used] = entry;

++used;

return;

}

set set:: operator -(const set& b){

set temp = \*this;

for(set::value\_type i=0; i< b.size(); i++)

temp.erase\_one(b.data[i]);

return temp;

}

void set:: operator -=(const set& remove){

for(set::value\_type i=0; i< remove.size(); i++)

erase\_one(remove.data[i]);

}

set::size\_type set::count(const value\_type& target) const {

size\_type answer;

size\_type i;

answer = 0;

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

if (target == data[i])

++answer;

return answer;

}

void set::operator +=(const set& addend){

assert(size() + addend.size() <= CAPACITY);

for(int i=0; i<addend.size(); i++){

if(!contains(addend.data[i])){

data[used]=addend.data[i];

used++;

}

}

}

set set:: operator +(const set& b2){

set answer;

assert(size() + b2.size() <= CAPACITY);

answer=\*this;

answer+=b2;

return answer;

}

void set :: printValues(){//a function that prints all the values in order to clean up the main function

size\_type index=0;

cout<<"\n";

while(size() > index){

cout<<data[index]<<"\n";

index++;

}

}

int main(int argc, const char \* argv[]) {

// insert code here...

set b, b2;

b.insert(1);

b.insert(2);

b.insert(3);

b.insert(4);

b.insert(3);

b2.insert(4);

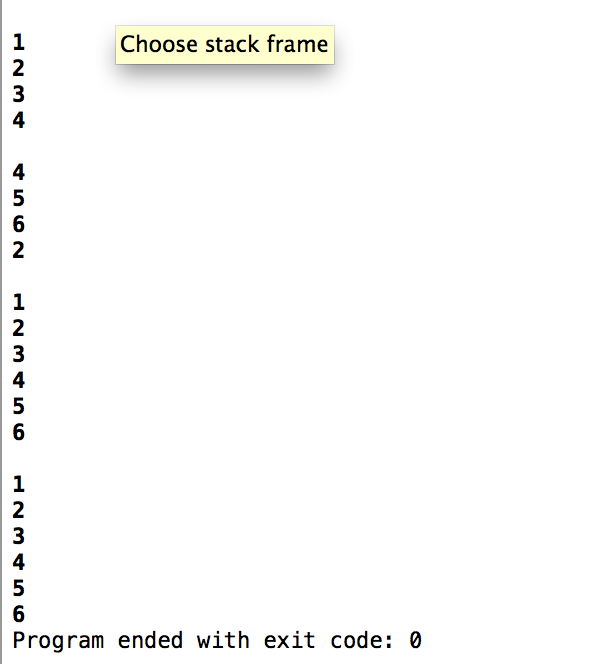
b2.insert(5);

b2.insert(6);

b2.insert(2);

b.printValues();

b2.printValues();



set c;

c=b+b2;

c.printValues();

b+=b2;

b.printValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW2.3 \*Chapter 3 Problem 8

//

// Created by Yousef Zoumot on 1/12/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class bag

{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

bag() {used = 0;}

//MODIFICATION

size\_type erase (const value\_type& target);

bool erase\_one(const value\_type& target);

void insert(const value\_type& entry, int key);

void operator +=(const bag& addend);

bag operator -(const bag& b);

void operator -=(const bag& remove);

//CONSTANT MEMBER FUNCTIONS

size\_type size() const { return used;}

size\_type count(const value\_type& target) const;

void printValues();

private:

value\_type data[CAPACITY];//the array to store items

int keys[CAPACITY];

size\_type used; //How much of the array is used

};

//NONMEMBER FUNCTIONS for the bag class

bag operator +(const bag& b1, const bag& b2);

const bag:: size\_type bag::CAPACITY;

bag::size\_type bag::erase(const value\_type& target){

size\_type index = 0;

size\_type many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bool bag::erase\_one(const value\_type& key1){

size\_type index;

index = 0;

while((index < used) && (keys[index] != key1))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

keys[index]=keys[used];

return true;

}

void bag::insert(const value\_type& entry, int key){

assert(size() < CAPACITY);

for(int i=0; i<size(); i++){

if(keys[i]==key)

return;

}

data[used] = entry;

keys[used]=key;

++used;

return;

}

void bag::operator +=(const bag& addend){

assert(size() + addend.size() <= CAPACITY);

bool tmp=false;

for(int i=0; i<addend.size(); i++){

for(int j=0; j<size(); j++){

if(addend.keys[i]==keys[j])

tmp=true;

}

if(tmp==false){

data[used]=addend.data[i];

keys[used]=addend.keys[i];

used++;

}

tmp=false;

}

//copy(addend.data, addend.data + addend.used, data + used);

//copy(addend.keys, addend.keys + addend.used, keys + used);

//used += addend.used;

}

bag bag:: operator -(const bag& b){

bag temp = \*this;

for(bag::value\_type i=0; i< b.size(); i++)

temp.erase\_one(b.keys[i]);

return temp;

}

void bag:: operator -=(const bag& remove){

for(bag::value\_type i=0; i< remove.size(); i++)

erase\_one(remove.data[i]);

}

bag::size\_type bag::count(const value\_type& target) const {

size\_type answer;

size\_type i;

answer = 0;

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

if (target == data[i])

++answer;

return answer;

}

bag operator +(const bag& b1, const bag& b2){

bag answer;

assert(b1.size() + b2.size() <= bag::CAPACITY);

answer += b1;

answer += b2;

return answer;

}

void bag :: printValues(){//a function that prints all the values in order to clean up the main function

size\_type index=0;

cout<<"\n";

while(size() > index){

cout<<data[index]<<" with index: "<<keys[index]<<"\n";

index++;

}

}

int main(int argc, const char \* argv[]) {

// insert code here...

bag b, b2;

b.insert(1,1);

b.insert(2,2);

b.insert(3,3);

b.insert(4,4);

b.insert(3,5);

b2.insert(3,6);

b2.insert(7,7);

b2.insert(2,2);

b2.insert(3,3);

b.printValues();

b2.printValues();

bag c;

c=b-b2;

c.printValues();

b-=b2;

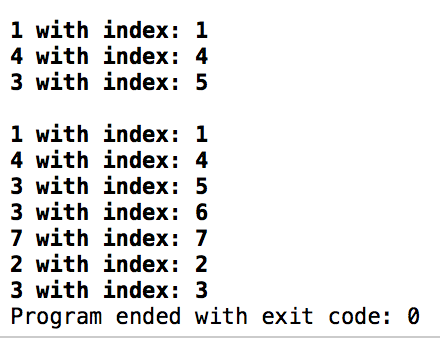
b.printValues();

b+=b2;

b.printValues();

return 0;

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.4 \*Chapter 4 Problem 1 parts A, D, F, G b/c professor said we only need to choose 4 to do

//

// Created by Yousef Zoumot on 1/18/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

class String{

char myString[200];

size\_t length;

public:

//CONSTRUCTOR FOR THE STRING CLASS

String(const char str[] = "");

String(const char c);//part A

//CONSTANT MEMBER FUNCTIONS FOR THE STRING CLASS

int searchF(const char c);//Part F returns index of first occurrence or returns -1

int searchA(const char c);//Part G returns number of occurrences or returns 0

//MODIFICATION MEMBER FUNCTIONS FOR THE STRING CLASS

void replaceChar(const char c, int index);//part D

//Tester Function

void printValues();

};

String:: String(const char str[]){

int i=0;

if(str[i]=='\0'){

myString[i]='\0';

length++;

return;

}

while(str[i]!= '\0'){

myString[i]=str[i];

i++;

length++;

}

return;

}

String::String(const char c){

if(c=='\0'){

myString[0]=c;

length++;

return;

}

myString[0]=c;

length++;

myString[1]='\0';

length++;

return;

}

void String:: replaceChar(const char c, int index){

myString[index]=c;

return;

}

int String:: searchF(const char c){

int i=0;

while(myString[i]!='\0'){

if(myString[i]==c)

return i;

i++;

}

cout<<"Not Found ";

return -1;

}

int String:: searchA(const char c){

int i=0;

int answer=0;

while(myString[i]!='\0'){

if(myString[i]==c){

answer++;

}

i++;

}

return answer;

}

void String:: printValues(){

int i=0;

while(myString[i]!='\0'){

cout<<myString[i];

i++;

}

cout<<"\n";

}

int main(int argc, const char \* argv[]) {

String s1("Hello my name is Johnny");

s1.printValues();

char c='A';

String s2(c);

s2.printValues();

s2.replaceChar('B', 0);

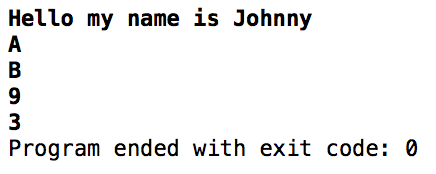
s2.printValues();

c='n';

cout<<s1.searchF(c)<<"\n";

cout<<s1.searchA(c)<<"\n";

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.5 \*Chapter 4 Problem 2A

//

// Created by Yousef Zoumot on 1/18/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <assert.h>

#include<vector>

#include <cstdlib>//Provides size\_t

using namespace std;

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef double value\_type;

typedef std::size\_t size\_type;

//CONSTRUCTOR

sequence(const size\_t cap);

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const value\_type& entry);

void attach(const value\_type& entry);

void remove\_current();

void addToFront(const value\_type& entry);

void removeFront();

void addToEnd(const value\_type& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

value\_type operator[](size\_type index);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

value\_type current() const;

private:

value\_type\* data;

size\_t capacity;

size\_type used;

size\_type current\_index;

void increaseS();

};

int main(int argc, const char \* argv[]) {

// insert code here...

sequence s1(100), s2(100);

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 s3(100);

s3= s1+s2;

s3.printValues();

sequence s4(100);

s4+=s1;

s4+=s2;

s4.printValues();

cout<<s4[0];

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

sequence::sequence (const size\_t cap)

{

current\_index = 0;

used = 0;

capacity=cap;

data= new value\_type[capacity];

}

void sequence::start( )

{

current\_index = 0;

}

void sequence::advance( )

{

current\_index++;

}

void sequence::insert(const value\_type& 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++;

}

void sequence::attach(const value\_type& 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++;

}

void sequence::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

void sequence:: addToFront(const value\_type& 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++;

}

void sequence:: removeFront(){

start();

remove\_current();

}

void sequence:: addToEnd(const value\_type& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

void sequence:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

double sequence:: operator[](size\_type index){

value\_type invalid=100000;

if(index<size())

return data[index];

else{

cout<<"This is not a valid index";

return invalid;

};

}

sequence sequence:: operator +(const sequence& s2){

sequence temp(100);

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;

}

void sequence:: operator +=(const sequence& s2){

\*this=\*this+s2;

}

void sequence:: printValues(){

cout<<"The values in the sequence are as follows: "<<"\n";

size\_type i;

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

cout<<data[i]<<" \n";

}

void sequence:: increaseS(){

value\_type\* tmp= new value\_type[2\*capacity];

for(int i=0; i<size(); i++){

tmp[i]=data[i];

}

delete[] data;

data=tmp;

capacity\*=2;

}

// CONSTANT MEMBER FUNCTIONS

sequence::size\_type sequence::size( ) const

{

return used;

}

bool sequence::is\_item( ) const

{

return current\_index != used;

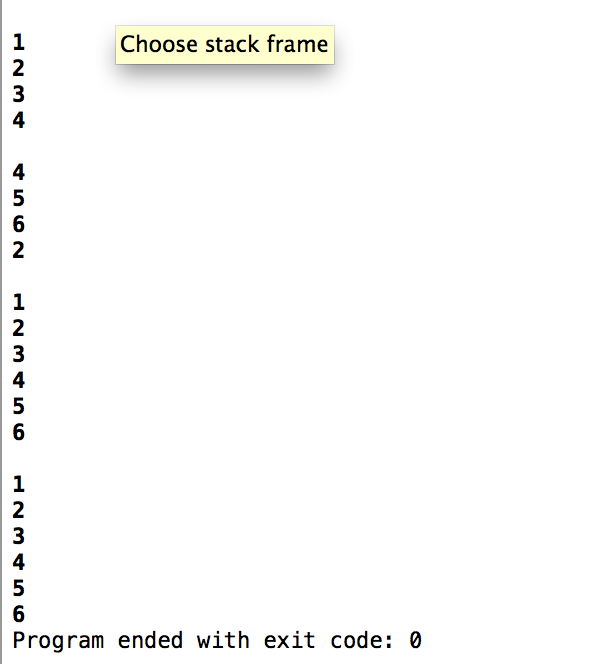
}

sequence::value\_type sequence::current( ) const

{

return data[current\_index];

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.6 \*Chapter 4 Problem 2b

//

// Created by Yousef Zoumot on 1/23/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <algorithm>

#include <iostream>

#include <cassert>

using namespace std;

class set{

int\* data;

int capacity;

void incSize();

int used;

public:

set(int x = 20);

set(const set& source);

~set();

int erase(const int& target);

bool erase\_one(const int& target);

void insert(const int& 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 int& target) const;

int size() const { return used; }

int count( const int& target) const;

void printValues();

};

int main(){

set a;

set b;

set c;

set d;

a.insert(1);

a.insert(2);

a.insert(2);

a.insert(3);

a.printValues();

b.insert(3);

b.insert(2);

b.insert(5);

b.printValues();

c = a - b;

c.printValues();

c = a + b;

c.printValues();

d.insert(3);

c += d;

c.printValues();

c -= d;

c.printValues();

c.erase\_one(1);

c.printValues();

}

int set::erase(const int& target){

int index = 0;

int many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

set:: set (int x){

assert(x>0);

used = 0;

capacity = x;

data = new int[x];

}

set:: ~set(){

if (data)

delete[] data;

}

set:: set(const set& source){

data = NULL;

\*this = source;

}

void set:: incSize(){

int\* temp = new int[2\*capacity];

for(int i = 0; i < capacity; i++){

temp[i] = data[i];

}

delete[] data;

data = temp;

capacity \*= 2;

}

void set::printValues(){

int i;

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

cout << data[i] << " ";

}

cout << endl;

}

bool set::erase\_one(const int& target){

int index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

void set::operator +=(const set& addend){

int 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++;

}

}

}

set set:: operator -(const set& b2){

set answer = \*this;

for(int i = 0; i < b2.used; i++)

answer.erase\_one(b2.data[i]);

return answer;

}

int set::count(const int& target) const {

int answer;

int i;

answer = 0;

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

if (target == data[i])

++answer;

return answer;

}

void set:: operator -=(const set& removeIt){

int i;

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

erase\_one(removeIt.data[i]);

}

void set::insert(const int& entry){

if(contains(entry))

return;

if(size() >= capacity)

incSize();

data[used] = entry;

++used;

return;

}

set& set:: operator =(const set& source){

if(this == &source)

return \*this;

if (data)

delete[] data;

if(source.used == 0){

used = 0;

capacity = 20;

data = new int[capacity];

return \*this;

}

data = new int[source.capacity];

for(int i = 0; i < source.capacity; i++){

data[i] = source.data[i];

}

used = source.used;

capacity = source.capacity;

return \*this;

}

bool set:: contains(const int& target) const{

int i;

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

if (target == data[i])

return true;

return false;

}

set set::operator +(const set& b2){

set answer = \*this;

if(answer.size() + b2.size() >= capacity)

incSize();

for(int i = 0; i < b2.used; i++){

if(!answer.contains(b2.data[i])){

answer.data[used] = b2.data[i];

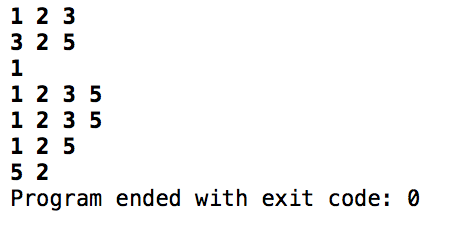
answer.used++;

}

}

return answer;

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.7 \*Chapter 4 Problem 2e

//

// Created by Yousef Zoumot on 1/12/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <algorithm>

#include <iostream>

#include <cassert>

using namespace std;

class bag{

public:

bag(int x = 20);

bag(const bag& source);

~bag();

int erase(const int& target);

bool erase\_one(const int& target);

bool contains(const int& target);

void insert(const int& entry, int key);

int size() const { return used; }

int count( const int& target) const;

void printValues();

bag operator +(const bag& b2);

bag& operator =(const bag& source);

bag operator -(const bag& b2);

void operator -=(const bag& removeIt);

void operator +=(const bag& addend);

private:

int\*\* data;

int capacity;

int used;

void incSize();

};

int main(){

bag a;

bag b;

bag c;

bag d;

a.insert(1, 1);

a.insert(2, 2);

a.insert(3, 3);

a.insert(4, 4);

b.insert(5, 5);

b.insert(6, 6);

b.insert(7, 7);

a.printValues();

b.printValues();

c = a + b;

c.printValues();

c = a - b;

c.printValues();

d.insert(1,6);

d.insert(2, 2);

d.insert(3, 5);

d.printValues();

c -= d;

c.printValues();

d.erase\_one(6);

d.erase\_one(5);

d.printValues();

}

int bag::erase(const int& target){

int index = 0;

int many\_removed = 0;

while(index < used){

if (data[index][0] == target){

--used;

data[index][0] = data[used][0];

data[index][1] = data[used][1];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bag:: bag (int x){

assert(x>0);

used = 0;

capacity = x;

data = new int\*[x];

for(int i = 0; i < x; i++){

data[i] = new int[2];

}

}

bag:: bag(const bag& source){

data = NULL;

\*this = source;

}

bag:: ~bag(){

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

}

void bag::printValues(){

int i;

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

cout <<data[i][0]<<" " ;

}

cout << endl;

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

cout << data[i][1] << " ";

}

cout << endl << endl << endl;

}

bool bag::erase\_one(const int& target){

int index;

index = 0;

while((index < used) && (data[index][1] != target))

++index;

if(index == used)

return false;

--used;

data[index][1] = data[used][1];

data[index][0] = data[used][0];

return true;

}

void bag:: incSize(){

int\*\* temp = new int\* [2\*capacity];

for(int i = 0; i < 2\*capacity; i++){

temp[i][0] = data[i][0];

temp[i][1] = data[i][1];

}

for(int i = 0; i < capacity; i++){

temp[i][0] = data[i][0];

temp[i][1] = data[i][1];

}

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

data = temp;

capacity \*= 2;

}

void bag::insert(const int& entry, int key1){

if(size() == capacity)

incSize();

data[used][0] = entry;

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

if(data[i][1] == key1){

cout << "That key is already used. Enter another";

cin >> key1;

i = 0;

}

}

data[used][1] = key1;

++used;

return;

}

void bag::operator +=(const bag& addend){

\*this = \*this + addend;

}

bag bag:: operator -(const bag& source){

/\*bag answer;

answer.data = NULL;

answer = \*this;\*/

bag answer;

if(answer.data){

for(int i = 0; i < answer.capacity; i++){

delete[] answer.data[i];

}

delete[] answer.data;

}

answer.data = new int\*[capacity];

for(int i = 0; i < capacity; i++)

answer.data[i] = new int[2];

for(int i = 0; i < source.capacity; i++){

answer.data[i][0] = data[i][0];

answer.data[i][1] = data[i][1];

}

answer.capacity = capacity;

answer.used = used;

for(int i = 0; i < source.used; i++){

answer.erase\_one(source.data[i][1]);

}

return answer;

}

void bag:: operator -=(const bag& removeIt){

\*this = \*this - removeIt;

}

int bag::count(const int& target) const {

int answer;

int i;

answer = 0;

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

if (target == data[i][0])

++answer;

return answer;

}

bag& bag:: operator =(const bag& source){

if(this == &source)

return \*this;

if(data){

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

}

if(source.used == 0){

used = 0;

capacity = 20;

data = new int\*[20];

for(int i = 0; i < 20; i++){

data[i] = new int[2];

}

return \*this;

}

data = new int\*[source.capacity];

for(int i = 0; i < source.capacity; i++)

data[i] = new int[2];

for(int i = 0; i < source.used; i++){

data[i][0] = source.data[i][0];

data[i][1] = source.data[i][1];

}

capacity = source.capacity;

used =source.used;

return \*this;

}

bag bag::operator +(const bag& source){

bag answer;

answer = \*this;

if(source.used + used >= capacity)

answer.incSize();

for(int i = 0; i < source.used; i++){

if(!answer.contains(source.data[i][1])){

answer.insert(source.data[i][0],source.data[i][1]);

}

}

return answer;

}

bool bag:: contains(const int &target){

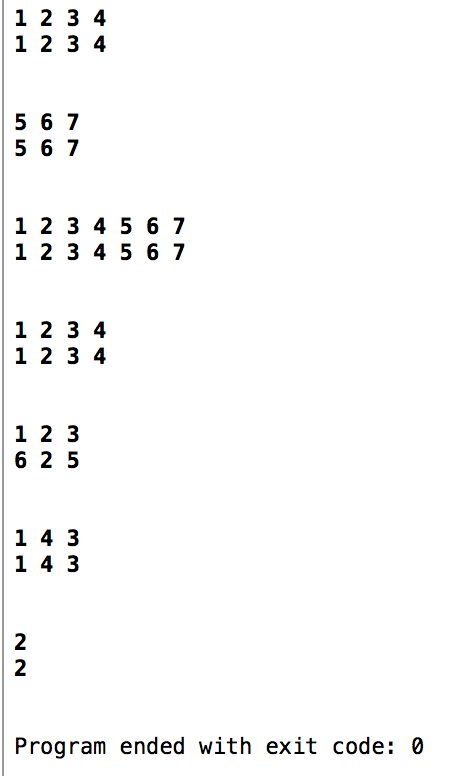
for(int i = 0; i < used; i++){

if(data[i][1] == target)

return true;

}

return false;

}