Name: Abraham Swaray

CSC 326

September 12, 2017

Main Bag.cpp

// Created by Frank M. Carrano and Tim Henry.

// Copyright (c) 2013 \_\_Pearson Education\_\_. All rights reserved.

#include <iostream> // For cout and cin

#include <string> // For string objects

#include "Bag.h" // For ADT bag

using namespace std;

int removeItem(Bag <string> &mybag, string item);

int main()

{

string food[] = { "Egg", "Soda", "Bread", "Orange", "Rice", "Yam" };

Bag<string> storeBag, fragileBag;

storeBag.add(food[0]);

storeBag.add(food[1]);

storeBag.add(food[2]);

storeBag.add(food[3]);

storeBag.add(food[4]);

storeBag.add(food[5]);

storeBag.add(food[6]);

// Get friend�s guess and check it.

while (storeBag.contains("Egg"))

{

storeBag.remove("Egg");

fragileBag.add("Egg");

cout << endl;

}

while (storeBag.contains("Bread"))

{

storeBag.remove("Bread");

fragileBag.add("Bread");

cout << " The removed items from the bag are: \n ";

fragileBag.displayBagContent();

cout << endl;

cout << " Store Bag now contains the follow item: \n";

storeBag.displayBagContent();

cout << endl;

}

cout << "No more fragile item in this bag!" << endl;

cout << endl;

int n = removeItem(storeBag, "Soda");

cout << n << " Item removed " << endl;

system("pause");

return 0;

};

int removeItem(Bag <string> &mybag, string item) {

int count = 0;

while (mybag.contains(item)) {

mybag.remove(item);

cout << "Item has been removed from the bag" << endl;

count++;

}

return count;

}

Bag.h

// Created by Frank M. Carrano and Tim Henry.

// Copyright (c) 2013 \_\_Pearson Education\_\_. All rights reserved.

/\*\* ADT bag: Array-based implementation.

@file Bag.h \*/

//#include <cstddef>

#ifndef \_BAG

#define \_BAG

#include <iostream>

using namespace std;

template <class ItemType>

class Bag

{

private:

static const int DEFAULT\_BAG\_SIZE = 6;

ItemType items[DEFAULT\_BAG\_SIZE]; // array of bag items

int itemCount; // current count of bag items

int maxItems; // max capacity of the bag

// Returns either the index of the element in the array items that

// contains the given target or -1, if the array does not contain

// the target.

int getIndexOf(const ItemType& target) const;

public:

Bag();

int getCurrentSize() const;

bool isEmpty() const;

bool add(const ItemType& newEntry);

bool remove(const ItemType& anEntry);

void clear();

bool contains(const ItemType& anEntry) const;

int getFrequencyOf(const ItemType& anEntry) const;

void displayBagContent();

int findRepetition(string myarray[], int increment);

}; // end Bag

template<class ItemType>

Bag<ItemType>::Bag() : itemCount(0), maxItems(DEFAULT\_BAG\_SIZE)

{

} // end default constructor

template<class ItemType>

int Bag<ItemType>::getCurrentSize() const

{

return itemCount;

} // end getCurrentSize

template<class ItemType>

bool Bag<ItemType>::isEmpty() const

{

return itemCount == 0;

} // end isEmpty

template<class ItemType>

bool Bag<ItemType>::add(const ItemType& newEntry)

{

bool hasRoomToAdd = (itemCount < maxItems);

if (hasRoomToAdd)

{

items[itemCount] = newEntry;

itemCount++;

} // end if

return hasRoomToAdd;

} // end add

template<class ItemType>

bool Bag<ItemType>::remove(const ItemType& anEntry)

{

int locatedIndex = getIndexOf(anEntry);

bool canRemoveItem = !isEmpty() && (locatedIndex > -1);

if (canRemoveItem)

{

itemCount--;

items[locatedIndex] = items[itemCount]; //this is how is removing item from the bag

} // end if

return canRemoveItem;

} // end remove

template<class ItemType>

void Bag<ItemType>::clear()

{

itemCount = 0;

} // end clear

template<class ItemType>

int Bag<ItemType>::getFrequencyOf(const ItemType& anEntry) const

{

int frequency = 0;

int searchIndex = 0;

while (searchIndex < itemCount)

{

if (items[searchIndex] == anEntry)

{

frequency++;

} // end if

searchIndex++;

} // end while

return frequency;

} // end getFrequencyOf

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

return getIndexOf(anEntry) > -1;

} // end contains

/\* ALTERNATE 1

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

return getFrequencyOf(anEntry) > 0;

} // end contains

\*/

/\* ALTERNATE 2

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

bool found = false;

for (int i = 0; !found && (i < itemCount); i++)

{

if (anEntry == items[i])

{

found = true;

} // end if

} // end for

return found;

} // end contains

\*/

// private

template<class ItemType>

int Bag<ItemType>::getIndexOf(const ItemType& target) const

{

bool found = false;

int result = -1;

int searchIndex = 0;

// if the bag is empty, itemCount is zero, so loop is skipped

while (!found && (searchIndex < itemCount))

{

if (items[searchIndex] == target)

{

found = true;

result = searchIndex;

}

else

{

searchIndex++;

} // end if

} // end while

return result;

} // end getIndexOf

template<class ItemType>

void Bag<ItemType>::displayBagContent() {

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

cout << items[i] << endl;

}

}

//print bag content

/\*

template <class ItemType>

int Bag<ItemType>::findRepetition(string myarray[], int increment) {

increment;

for (int i = 0; i < sizeof(myarray)/sizeof(myarray[0]); i++) {

if(myarray[i] == myarray[i]) {

increment++;

}

else {

cout << "Total Repetition found: " << increment << endl;

}

}

cout << "Total Repetition found: " << increment << endl;

return increment;

}

\*/

#endif

OUTPUT

The removed items from the bag are:

Egg

Bread

Store Bag now contains the follow item:

Yam

Soda

Rice

Orange

No more fragile item in this bag!

Item has been removed from the bag

1 Item removed

Press any key to continue . . .