Daniel Leone. CSCI 301-Section 54

StarID: mm8755tt

Due Date: Jul 3, 2023

Instructor: Dr. Jie Meichsner

1. **Design Document**

Introduction

This program implements a virtual ShoppingCart to allow a user to create a list of items for purchase. They are able to add, remove, and modify their order. The user will first add their selections into the cart. A list of their items will be displayed with the price total. They are able to adjust their cart after review of the current total by adding items, removing items, or modifying the quantities of items. The updated list and price total will display once modifications are complete.

Data Structures

Item is a class that contains data members that provide details about an individual item for purchase to include: name of type string, unitPrice of type double, and quantity of type int. String is an array of characters utilized to store name within the item class.

ShoppingCart is a class that inherits from class Bag<Item>. It contains one data member of type double that holds totalPrice.

Bag<ItemType> is a base class that contains an array of ItemType called items. In the context of this program, this array will be of type Item and will hold the contents of a ShoppingCart object. The array size is set by a constant integer of DEFAULT\_BAG\_SIZE. A vector of ItemType is utilized in a function that will copy the contents within array items and return the vector.

Functions

The main program utilizes nonmember functions for menu and display purposes.

* One module allows the user to initialize their ShoppingCart by continuously adding items until the user is done. This consists of one function.
* One module allows the user to add, remove, or modify the quantity of their ShoppingCart. This consists of one main function utilizing three sub functions. The sub functions provide the logic for “add”, “remove”, and “modify quantity.”
* One function displays the ShoppingCart contents and total price.
* One functions are provides a border for display purposes.

Item class provides functions to access and modify data members.

* Three getter functions retrieve values of the data members.
* Three setter functions modify values of the data members.
* Override of “operator ==” allows two Item objects to be compared. The comparison is of name and price.
* Override of “operator >>” and “operator<<” allows istream and ostream to utilize an Item object directly. These are friend functions to Item Class.

ShoppingCart class inherits multiple functions from base class Bag<Item>. It contains one class specific member function, and overrides two functions from class Bag<Item>.

* One getter function to retrieve the value of totalPrice.
* Overrides add function from Bag<Item>. It includes the addition of adjusting total price based on new Item price x quantity. Returns type bool to signal if the object was added or not.
* Overrides remove function from Bag<Item>. It includes the subtraction of adjusting total price based on new Item price x quantity. Returns type bool to signal if the object was removed or not.
* Inherits six additional functions from Bag<Item> that work on array items.

The Main Program

The program beings by initializing an empty ShoppingCart and displaying a welcome message to the user.

The program will provide the user with instruction on how to add items to the ShoppingCart. The add to cart menu will allow the user to continue to add items until he or she decides not to. Note that the cart will only hold the first 6 items despite more being added. When initial adding is complete, the contents and total price will be displayed for the user to review.

The program will provide the user the option to modify his or her cart at this time through the modify menu. The user can choose to add new items, remove current items, or modify the quantities of current items. The contents will be checked to see if the item is in the cart prior to removal or modification. The modification quantity will have to be a valid integer greater than 0 in order to modify. If these conditions are not met, then the user will receive a message informing them.

The program will display the updated contents and total price upon completion of modifying the cart. A “Thanks for shopping…” message will display before the program terminates.

**Structure Chart of the Main Program**

Main Program

menuAddToCart

displayTotal

menuModifyOrder

lineBorder

modifyAdd

modifyRemove

modifyChangeQuantity

**UML**

|  |
| --- |
| Item |
| -name : string  -unitPrice : double  -quantity : int |
| +Item() :  +Item(initName : string, initUnitPrice : double, initQuantity : int) :  +setName(updateName : string) : void  +setUnitPrice(updateUnitPrice : double) : void  +setQuantity(updateQuantity : int) : void  +getName() : string  +getUnitPrice() : double  +getQuantity() : int  +operator==(item2 : Item&) : bool  // FRIEND FUNCTION  +operator>>(ins : istream&, target : Item) : istream&  +operator<<(outs : ostream&, source : Item) : ostream& |

|  |
| --- |
| ShoppingCart |
| -totalPrice : double |
| +ShoppingCart() :  +getTotalPrice() : double  +add(newEntry Item&) : bool  +remove(anEntry Item&) : bool |

1. **Code List**

/\* This program allows a user to add, remove, and modify a virtual ShoppingCart.

@file location: /home/STCLOUDSTATE/mm8755tt/PROJECTS/PROJECT\_2/project2.cpp

@Author: Daniel Leone. CSCI 301-Section 54

@StarID: mm8755tt

@Due Date: Jul 3, 2023

@Instructor: Dr. Jie Meichsner

The user will add items to their shopping cart. The program will take the user input

in the form of "name price quantity" to create an item to add to the user's shopping

cart. It will continue to prompt the user to add items until the user decides not to

add anymore. The user can add up to a maximum of 6 items, after which items can be

input but will not be added to the shopping cart. The list of items in the shopping

cart, as well as the total price, will be displayed for review.

After review, the user will be given the option to modify their cart. If he or she

chooses to modify, the program will prompt the user to either add items, remove items,

or change quantities of existing items. The program will inform the user if he or

she tries to remove or modify an item that is not in the cart. Once all modifications

are complete, the updated list of items in the shopping cart, as well as the updated

total price, will be displayed with a "Thanks for shopping..." message.

\*/

#include "ShoppingCart.h"

#include <iostream>

#include <iomanip>

using namespace std;

// Prototypes

void lineBorder(); // set number of lines for display top or bottom border

void displayTotal(ShoppingCart& cart); // displays shopping cart contents and total price

void menuAddToCart(ShoppingCart& cart); // initial menu for user to add items to shopping cart

void menuModifyOrder(ShoppingCart& cart); // modify menu after user reviews initial order

int main()

{

ShoppingCart sCart;

cout << "\nWelcome to XXX SHOPPING CENTER." << endl;

// Allows user to do initial adding to his or her shopping cart

menuAddToCart(sCart);

// Display initial order

cout << "Here is your order: " << endl;

lineBorder();

displayTotal(sCart);

lineBorder();

cout << endl << endl;

// After review, allows user to modify shopping cart contents if desired

menuModifyOrder(sCart);

// Display updated order

cout << "Here is your updated order: " << endl;

lineBorder();

cout << "You have ordered the following items: " << endl;

displayTotal(sCart);

cout << "Thanks for shopping in XXX SHOPPING CENTER." << endl;

lineBorder();

cout << endl << endl;

return 0;

};

// initial menu for user to add items to shopping cart

void menuAddToCart(ShoppingCart& cart){

Item tmpItem;

cout << "Enter the item you selected as the following order: " << endl;

cout << "name unitPrice quantity" << endl;

cout << "(Name can not contain any space. Otherwise errors happen!)" << endl;

// loop utilized for the adding phase

bool adding = true;

while(adding)

{

// Adds item to shopping cart

cout << "--> "; cin >> tmpItem;

cart.add(tmpItem);

// loop used for user selection after adding an item

bool exit = false;

while(!exit){

char selection;

cout << "Want to continue y/n-->"; cin >> selection;

if(selection=='y'){

exit = true; // breaks out of selection loop

}else if(selection=='n'){

adding = false; // breaks out of main loop

exit = true; // breaks out of selection loop

}else{

cout << "Invalid selection." << endl;

continue;

}

cout << endl;

}

}

}

// allows the user to add an item to cart during order modification

void modifyAdd(ShoppingCart& cart){

cout << "Enter the item to add as the following order: " << endl;

cout << "name unitPrice quantity" << endl;

Item tmpItem;

cout << "--> "; cin >> tmpItem;

cart.add(tmpItem);

cout << "The item has been added." << endl;

}

// allows the user to remove an item, if item is in cart, during order modification

void modifyRemove(ShoppingCart& cart){

cout << "Enter the item to remove as the following order: " << endl;

cout << "name unitPrice quantity" << endl;

Item tmpItem;

cout << "--> "; cin >> tmpItem;

if(cart.remove(tmpItem)){

cout << "The item has been removed." << endl;

}else{

cout << "No such item in your shopping cart!" << endl;

}

}

// allows the user to modify an item quantity, if item is in cart, during order modification

void modifyChangeQuantity(ShoppingCart& cart){

Item tmpItem;

cout << "Enter the item to change as the following order: " << endl;

cout << "name unitPrice quantity" << endl;

cout << "--> "; cin >> tmpItem;

if(cart.contains(tmpItem)){ // check to see if the item is in the cart

bool entering = true;

while(entering){

int newQuantity;

cout << "Enter a new quantity --> "; cin >> newQuantity;

// if newQuantity is greater than 0, removes old item with old quantity and adds tmpItem with updated quantity

if(newQuantity > 0){

cart.remove(tmpItem);

tmpItem.setQuantity(newQuantity);

cart.add(tmpItem);

cout << "The quantity has been modified." << endl;

entering = false; // breaks entering loop

}else{

cout << newQuantity << " is not a valid input." << endl;

}

}

}else{

cout << "No such item in your shopping cart!" << endl;

}

}

// menu to modify cart after review

void menuModifyOrder(ShoppingCart& cart){

// functions used: modifyAdd(), modifyRemove(), modifyChangeQuantity()

bool modifyOrder = true;

while(modifyOrder){

char selection;

cout << "Want to modify your order? y/n-->"; cin >> selection;

if(selection=='y'){

int choice;

cout << "What do you want? Enter 1: add 2: remove 3: change quantity\n--> "; cin >> choice;

if(choice==1){

modifyAdd(cart);

}

else if(choice==2){

modifyRemove(cart);

}

else if(choice==3){

modifyChangeQuantity(cart);

}

else{

cout << "Invalid selection." << endl;

}

}else if(selection=='n'){

modifyOrder = false; // breaks main loop

}else{

cout << "Invalid selection." << endl << endl;

continue;

}

cout << endl;

}

}

// creates a line for console display

void lineBorder(){

for(int i = 0; i<50; i++)

{ cout << "-"; }

}

// display shopping cart contents and total price

void displayTotal(ShoppingCart& cart){

// Library facilities used: iomanip

// setw() keeps columns aligned with item display

cout << endl << setw(20) << left << "Name"<< setw(20) << left <<"Unit\_Price"<< setw(20) << left << "Quantity" << endl;

// creates a vector<Item> set by copying shopping cart contents via Bag<Item>::toVector()

// iterates through contents and displays to console

vector<Item> v = cart.toVector();

for(int i = 0; i < cart.getCurrentSize(); i++){

cout << v[i] << endl;

}

// display total price

cout << "The total charge is $" << cart.getTotalPrice() << endl;

}

------------------------------------------------------------------------------------------------------------------------------------------

#ifndef SHOPPINGCART\_H

#define SHOPPINGCART\_H

#include "Item.h"

#include "Bag.h"

class ShoppingCart : public Bag<Item>

{

private:

/\*\* total price of Items in the ShoppingCart \*/

double totalPrice;

public:

/\*\* CONSTRUCTORS\*/

/\*\* Default Constructor

/\* @pre None

/\* @post initializes Bag<Item> default values and sets

totalPrice to 0.0 by default\*/

ShoppingCart();

/\*\* CONSTANT MEMBER FUNCTIONS\*/

/\*\* returns the total price of Items in the ShoppingCart

/\* @pre None

/\* @post The value returned is the totalPrice of Items in the ShoppingCart.\*/

double getTotalPrice() const { return totalPrice;}

/\*\* MODIFICATION MEMBER FUNCTIONS\*/

/\*\* Adds a new entry to this ShoppingCart and adjusts totalPrice.

/\* @param newEntry The object to be added as a new entry.

/\* @pre None

/\* @post If successful, newEntry is stored in the ShoppingCart and

the count of items in the bag has increased by 1. The totalPrice

is increased by the (price x quantity) of new entry.

/\* @return True if addition was successful, or false if not. \*/

bool add(const Item& newEntry);

/\*\* Removes an entry from this ShoppingCart, if possible, and adjusts totalPrice.

/\* @param anEntry The object to be removed.

/\* @pre Desired item for removal is in ShoppingCart.

/\* @post If successful, anEntry is removed from the ShoppingCart and

the count of items in the bag has decreased by 1. The totalPrice

is decreased by the (price x quantity) of an entry.

/\* @return True if removal was successful, or false if not. \*/

bool remove(const Item& anEntry);

};

#include "ShoppingCart.cpp"

#endif

------------------------------------------------------------------------------------------------------------------------------------------

/\*\* FILE: ShoppingCart.cpp \*/

/\*\* CLASS IMPLEMENTED: ShoppingCart inherited from Box<Item> (See ShoppingCart.h for documentation.) \*/

#include "ShoppingCart.h"

ShoppingCart::ShoppingCart()

{

Bag<Item>(); // Default constructor

totalPrice = 0.0;

}

bool ShoppingCart::add(const Item& newEntry)

{

/\*\* Utilize superclass Bag<Item>::add() function

Check if Item can be added to ShoppingCart

If return true, add an item to ShoppingCart \*/

if(Bag<Item>::add(newEntry))

{

// If can add, add to current totalPrice by (newEntry's price x newEntry's quantity)

totalPrice += (newEntry.getUnitPrice() \* newEntry.getQuantity());

}

}

bool ShoppingCart::remove(const Item& anEntry)

{

/\*\* Utilize superclass Bag<Item>::remove() function

Check if Item is contained in ShoppingCart

If return true, remove Item from ShoppingCart \*/

if(Bag<Item>::remove(anEntry))

{

// If can remove, subtract from current totalPrice by (anEntry's price x anEntry's quantity)

// Assumes user input correct quantity \*/

totalPrice -= (anEntry.getUnitPrice() \* anEntry.getQuantity());

}

}

------------------------------------------------------------------------------------------------------------------------------------------

#ifndef ITEM\_H

#define ITEM\_H

#include <iostream>

using namespace std;

class Item

{

private:

/\*\* name of this Item \*/

string name;

/\*\* price of this Item \*/

double unitPrice;

/\*\* quantity of this Item \*/

int quantity;

public:

/\*\* CONSTRUCTORS\*/

/\*\* Default Constructor

/\* @pre None

/\* @post sets the Item name = " ", price = 0.0, quantity = 0 by default\*/

Item();

/\*\* Constructor with parameters

/\* @param initName the name of the Item

/\* @param initPrice the price of the Item

/\* @param initQuantity the quantity of the Item

/\* @pre None

/\* @post sets the Item to(initName, initPrice, initQuantity)\*/

Item(string initName, double initUnitPrice, int initQuantity);

/\*\* MODIFICATION MEMBER FUNCTIONS\*/

/\*\* set the Item's name to a new value

/\* @param updateName the value of the Item's name

/\* @pre None

/\* @post Item's name changed to(updateName)\*/

void setName(string updateName);

/\*\* set the Item's price to a new value

/\* @param updatePrice the value of the Item's price

/\* @pre None

/\* @post Item's price changed to(updatePrice)\*/

void setUnitPrice(double updateUnitPrice);

/\*\* set the Item's quantity to a new value

/\* @param updateQuantity the value of the Item's quantity

/\* @pre None

/\* @post Item's quantity changed to(updateQuantity)\*/

void setQuantity(int updateQuantity);

/\*\* CONSTANT MEMBER FUNCTIONS\*/

/\*\* returns the name of the item

/\* @pre None

/\* @post The value returned is the name of the Item.\*/

string getName() const {return name;};

/\*\* returns the price of the item

/\* @pre None

/\* @post The value returned is the price of the Item.\*/

double getUnitPrice() const {return unitPrice;};

/\*\* returns the quantity of the item

/\* @pre None

/\* @post The value returned is the quantity of the Item.\*/

int getQuantity() const {return quantity;};

/\*\* compares two Items by name and price and returns bool value

/\* @param item1 is the lhs Item for comparison

/\* @param item2 is the rhs Item for comparison

/\* @pre None

/\* @post item1 and item2 are compared by both name and price.

If name and price match, function returns true; Else, function returns false. \*/

bool operator==(const Item& item2) const;

/\*\* FRIEND NONMEMBER FUNCTIONS\*/

/\*\* writes the string name, double price and int quantity of an Item to the istream

/\* Friend Class to Item

/\* @param ins is the istream

/\* @param target is a Item

/\* @pre None

/\* @post the name, price, and quantity of target are modified by user input

and written to ins. The return value is the istream ins.\*/

friend istream& operator>>(istream& ins, Item& target);

/\*\* writes the name, price, and quantity to the ostream

/\* @param outs is the ostream,

/\* @param source is a fraction

/\* @pre None

/\* @post the name, the price, and the quantity of source have been

written to outs. The return value is the ostream outs.\*/

friend ostream& operator<<(ostream& outs, Item& source);

};

#include "Item.cpp"

#endif

------------------------------------------------------------------------------------------------------------------------------------------

/\*\* FILE: Item.cpp \*/

/\*\* CLASS IMPLEMENTED: Item (See Item.h for documentation.) \*/

#include <iostream>

#include <iomanip>

#include "Item.h"

using namespace std;

Item::Item()

{

name = " "; // Default Constructor

unitPrice = 0.00;

quantity = 0;

}

Item::Item(string initName, double initUnitPrice, int initQuantity)

{

name = initName; // Constructor sets name to string value

unitPrice = initUnitPrice; // Constructor sets price to double value

quantity = initQuantity; // Constructor sets quantity to int value

}

void Item::setName(const string updateName)

{

name = updateName;

}

void Item::setUnitPrice(const double updateUnitPrice)

{

unitPrice = updateUnitPrice;

}

void Item::setQuantity(const int updateQuantity)

{

quantity = updateQuantity;

}

bool Item::operator==(const Item& item2) const

{

// if lhs == rhs, return true

return

(name == item2.getName())

&&

(unitPrice == item2.getUnitPrice());

}

//NON-MEMBER FRIEND FUNCTIONS

istream& operator>>(istream& ins, Item& target)

{

// Library facilities used: iostream

// Friend of: Item class

cin >> target.name >> target.unitPrice >> target.quantity;

return ins;

}

ostream& operator<<(ostream& outs, Item& source)

{

// Library facilities used: iostream and iomanip

// Friend of: Item class

outs << setw(20) << left << source.name << "$" << setw(20) << left << source.unitPrice << setw(20) << left << source.quantity;

return outs;

}

------------------------------------------------------------------------------------------------------------------------------------------

1. **User Document**

The program **project2** generates a virtual shopping cart for the user to add, remove, and modify items. Items are identified by their name and price, but not quantity. The user will begin by adding items to his or her cart. The contents and total price will be displayed. The user will be provided the option to modify his or her order. The updated contents and total price will be displayed when the user is done modifying his or her order.

The program’s name is **project2.cpp**. It is located at the following directory on **centOS**:

**/home/STCLOUDSTATE/mm8755tt/PROJECTS/PROJECT\_2**

To compile, simply enter:

**g++ project2.cpp**

To run the program, enter **a.out**, then follow the prompts to add items to the shopping cart. The items will be added in the format: *name unitPrice quantity*. The program assumes that the user will only try to add a maximum of six items. If the user goes over this limit, the items after item six will not be added.

The beginning of the program will look like this with example user input.

**Welcome to XXX SHOPPING CENTER.**

**Enter the item you selected as the following order:**

**name unitPrice quantity**

**(Name can not contain any space. Otherwise errors happen!)**

**-->T-shirt 19.99 2**

After the first selection, the user will be asked whether he or she would like to add another item simply with **y** or **n** as such. If **y**, he or she will provide another item. If **n**, the user will receive a display of his or her cart along with the total price:

**Want to continue y/n-->y**

**-->Sweater 39.99 2**

**Want to continue y/n-->n**

**Here is your order:**

**------------------------------------------------------------------------**

**Name Unit\_Price Quantity**

**T-shirt $19.99 2**

**Sweater $39.99 2**

**The total charge is $119.96**

**------------------------------------------------------------------------**

The user will be provided the option to modify his or her cart. When prompted, a **y** or **n** will allow the user to choose. If the user chooses **y**, the options to add, remove, or modify quantity will be presented with the selection being numerical values **1** for add, **2** for remove, and **3** for change quantity.

**Want to modify your order? y/n-->y**

**What do you want? Enter 1: add 2: remove 3: change quantity\n-->1**

If the user chooses to add an item to the cart, they will be prompted in a similar way as the start of the program.

**Enter the item you selected as the following order:**

**name unitPrice quantity**

**-->iphone\_case 25.50 3**

**The item has been added.**

The user will be prompted again asking whether he or she wants to modify the order.

**Want to modify your order? y/n-->y**

**What do you want? Enter 1: add 2: remove 3: change quantity\n-->2**

If the user chooses to remove an item, they will receive the following prompt. Note, that if the item does not exist or the user does not input the name and price properly, the user will be prompted with a message stating that the item is not in the cart. The user will once again be prompted to choose whether he or she wants to modify the cart.

**Enter the item to remove as the following order:**

**name unitPrice quantity**

**-->Sweater 29.99 2**

**No such item in your shopping cart!**

**Want to modify your order? y/n-->y**

**What do you want? Enter 1: add 2: remove 3: change quantity\n-->2**

**Enter the item to remove as the following order:**

**name unitPrice quantity**

**-->Sweater 39.99 2**

**The item has been remove.**

The user will be prompted again asking whether he or she wants to modify the order.

**Want to modify your order? y/n-->y**

**What do you want? Enter 1: add 2: remove 3: change quantity\n-->3**

If the user chooses to modify the quantity of an item, they will receive the following prompt. If the user tries to modify an item by a value of less than zero, they will be prompted to input a new value until a valid value is made.

**Enter the item to change as the following order:**

**name unitPrice quantity**

**-->T-shirt 19.99 2**

**Enter a new quantity --> -3**

**-3 is not a valid input.**

**Enter a new quantity --> 3**

**The quantity has been modified.**

If the user is done and chooses **n**,the updated contents and total price will be displayed and the program will terminate.

**Want to modify your order? y/n-->n**

**Here is your updated order:**

**------------------------------------------------------------------------**

**You have ordered the following items:**

**Name Unit\_Price Quantity**

**iphone\_case $25.5 3**

**T-shirt $19.99 3**

**The total charge is $136.47**

**Thanks for shopping in XXX SHOPPING CENTER.**

**------------------------------------------------------------------------**

The cart should reflect the changes. The display now informs the user that the items have been ordered. It also ends with a “Thank for shopping...” message.

1. **Test Data Plan**

|  |  |  |
| --- | --- | --- |
| **valid input values** |  |  |
| add() | input: item (name unitPrice quantity) | iphone\_case 25.50 3 |
|  | expected output: item added | “The item has been added.”  \*iphone\_case 25.50 3 |
|  |  |  |
| remove() | input: item (name unitPrice quantity) | Sweater 39.99 2 |
|  | expected output: item removed | “The item has been removed.”  \*ITEM REMOVED |
|  |  |  |
| changeQuantity() | input: item (name unitPrice quantity) | T-shirt 19.99 2 |
|  | expected output: prompt | “Enter a new quantity --> “ |
|  | input: integer | 3 |
|  | expected output: item quantity modified to new integer value | “The quantity has been modified.”  \*T-shirt 19.99 3 |
|  |  |  |
| **boundary values** |  |  |
| add() | input: item (name unitPrice quantity) 1st – 6th item | iphone\_case 25.50 3 |
|  | expected output: added value | iphone\_case 25.50 3 |
|  |  |  |
| remove() | input: item (name unitPrice quantity) contained in cart | Sweater 39.99 2 |
|  | expected output: value | ITEM REMOVED |
|  |  |  |
| changeQuantity() | input: item (name unitPrice quantity) contained in cart | T-shirt 19.99 2 |
|  | expected output: prompt | “Enter a new quantity --> “ |
|  | input: integer greater than 0 | 3 |
|  | expected output: value | T-shirt 19.99 3 |
|  |  |  |
| **invalid input values** |  |  |
| add() | input: item (name unitPrice quantity) 7th+ item | T-shirt 19.99 2 |
|  | expected output: item not added | “The item has been added.”  Item not added |
|  |  |  |
| remove() | input: item (name unitPrice quantity) contained in cart | Sweater 29.99 2 |
|  | expected output: prompt | “No such item in your shopping cart!” |
|  |  |  |
| changeQuantity() | input: item (name unitPrice quantity) contained in cart | T-shirt 19.99 2 |
|  | expected output: prompt | “No such item in your shopping cart!” |
|  |  |  |
|  | input: integer less than or equal to 0 | -3 |
|  | expected output: prompt | “-3 is not a valid input.” |

1. **Summary**

In this project we implemented a virtual shopping cart that uses various prompts to allow a user to add, remove, and modify items through user input. It displays the contents of the cart and displays the total price.

The implementation focused on the concept of inheritance. ShoppingCart inherited from base class Bag<Item>. Many aspects of inheritance were required to properly execute the program. One example is utilizing base class functions through the subclass. Another example is overriding virtual functions to fit the more specific subclass. Having an understanding of the base class is very important to fully utilize the base class and maximize efficiency.

A secondary concept that was worked on was working with a template class. We did not create a template class, but we were required to work with one. Creating the Item class and having it function with Bag<ItemType> functions was not difficult, but it was important that criteria were met for them to work properly.

Including the implementation file of a class at the end of the header file was new for me. It allowed the main program (or any program that includes the header file) to include implementations by simply “including” the header file. This worked well for compiling and cohesiveness.

The current implementation can be improved. It makes some assumptions of the user. An improvement would be implementing better controls for user input, to keep the user bounded by the parameters of the program.