Part C – Encapsulation

Class with a Resource

Workshop 6 (v1.1)

In this workshop, you will work with a class that holds dynamic resources as well as static ones. In addition to having regular member functions, this class will also be copied. Copying their data members will involve the use of copy constructors and assignment operators. The class used in this workshop will be the Basket class for which we will produce different fruit baskets.

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities

- To implement a copy constructor and copy assignment operator
- To manage a class with dynamic resources
- To use existing objects to create new ones or to assign values to existing objects based on other existing ones.
- To implement custom input/output operators.
- To describe to your instructor what you have learned in completing this workshop

Submission Policy

This workshop has only in lab section and reflection. The DIY section is replaced by the milestones of the project that will be posted shortly.

Workshop 6 is to be submitted during the workshop period from the lab. To get the mark for this workshop you must attend the lab.

Start the workshop few days earlier and come to the workshop sessions with questions and attendance.

Remember: you must be present at the lab in order to get credit for the workshop.
If you attend the lab period and cannot complete the workshop during that period, ask your instructor for permission to complete after the period. You must be present at the lab in order to get credit for the *workshop*.

All your work (all the files you create or modify) must contain your name, Seneca email and student number.
You are responsible to back up your work regularly.
Ask your professor if there are any additional requirements for your specific section.

Citation and Sources

When submitting Workshops, Project and assignment deliverables, a file called **sources.txt** must be present. This file will be submitted with your work automatically.
You are to write either of the following statements in the file "**sources.txt**":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.
Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.
You need to mention the workshop name or assignment name and also mention the file name and the parts in which you received the code for help.
Finally add your name and student number as signature.

By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

Late Submission Penalties:

Late submission: 1 to 6 days -50% after that submission rejected.

- If the content of sources.txt or reflect.txt is missing, the mark for the whole workshop will be **0.**

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:
Run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS06/lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS06/reflect -due<ENTER>

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:
g++ -Wall -std=c++11 -o ws (followed by your .cpp files)
After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws")
valgrind ws <ENTER>

Utils Module

For those of you who would like to reuse their previously developed functions and classes in later workshops, an empty module is added to the workshop called `Utils` (`Utils.cpp` and `Utils.h`). These files are empty and will be compiled with your workshop.
If you don't have any interest in reusing your previous functions, leave these files as they are, and they will not have any effect in the workshop.
If you have anything that you would like to add to the project, place their code in the `".cpp"` file and the definitions and prototypes in the `".h"` file. This header file will be included in the tester program.
In any case when submitting your work, make sure that these two files (empty or not) are submitted along with your workshop.

BASKET MODULE

The **Basket** module is a class that represent fruit baskets.

To accomplish the above follow these guidelines:

Design and code a class named **`Basket.`**

Follow the usual rules for creating a module: (i.e. Compilation safeguards for the header file, the sdds namespace and coding styles your professor asked you to follow).

PRIVATE MEMBERS:

A Basket will have the following data members:

- `m_fruitName`
  This is a **character pointer** representing the name of the fruit basket. This pointer will be used to allocate memory in order to store strings of **dynamic** length.
- `m_qty`
  This is an **integer value that** represents the quantity of fruits for each of the Basket which can not be less than one.
- `m_price`
  This is a **double value** that represents the price of the Basket which can not be less than zero.

PUBLIC MEMBERS:

Constructors/Destructors

Basket objects will be making use of **constructors** to handle their creation. There are three **constructors** that are required:

1. Default constructor – This constructor should set a Basket object to **a safe empty state**. The notion of what the safe empty state will be left up to your design but choose reasonable values.
2. 3 Argument Constructor – This constructor will take in 3 parameters:
   a. A **constant character pointer** that represents the name of the Fruit Basket

b. An **integer value** that represents the quantity of fruits of each of the Basket
c. A **double value** that represents the price of the Basket

You have to validate all the values before setting the values to the data members. **If the provided constant character pointer** for the fruit name is either **nullptr** or an **empty string, quantity and price is less** than zero set the Basket to **a safe empty state**.

Copy Constructor – This constructor will take in 1 parameter: A **constant-Basket reference** that will be treated as a source object from which we will create a new Basket.

We are copying all the parameters of the parameter Basket and using their values to create a new Basket with the same values. As a Basket is composed of resources where some of them are static and others are dynamic, attention will need to be paid when copying values.

For the quantity and the price data members the copying can be done via **shallow copying.** Whereas the name needs to apply **deep copying** (recall the need for independence of resources).

Destructor- Free the memory in the destructor that you dynamically allocated to avoid memory leak.

Other Members

```
void setName(const char*);
```
This member function is used to set the name of the Basket based on the parameter. Recall that the name is a character pointer thus pay attention to the need for **allocation** and **deallocation** of **memory** when setting the name. Recall the need for a **nullbyte** at the end of the name **dynamic** character array.

```
void setQty(int);
```
This member function is used to set the quantity of fruits of the Basket based on the parameter.

```
void setPrice(double);
```
This member function is used to set the price of the Basket based on the parameter.

```
bool isempty() const;
```

This function will return true if the object is in safe empty state.

```
bool addPrice(double);
```

This function receives a price and add that price with the current price. If the price is not positive value, it does nothing.

```
ostream& display(ostream& os) const;
```

This member function will display the current details of the Basket. This display function takes in a **parameter** of **ostream** type and when printing out the below output, *instead of directing the text to cout*, **direct it to the os parameter instead**. Consider **referencing week 6 handout examples of custom I/O** if difficulties arise.

If the Basket is in an **empty** state the following will be printed:

```
"This Basket has not yet been filled" <newline>
```

If the Basket isn't empty then it will print:

```
"Basket Details" <newline>
"Name: [name]" <newline>
"Quantity: [qrt]" <newline>
"Price: [price]" <newline>
```

The price should have two decimal point precision.
Refer to the sample output for details.

Lastly at the end of the function **return the os object**.

Operator Overload Members

Copy Assignment Operator

Implement a copy assignment operator for the Basket class that has the following signature:

```
Basket& operator=(const Basket&);
```

This operator receives an unmodifiable reference to a **Basket** object and copies the content of that object into the current object and returns a reference to the current object, as updated.

Consider how you may best add this overload but keep redundancy to a minimum.

```
bool operator==(const Basket&)
```

This function compares two Baskets against one another. It returns a **true** value if and only if: The name, quantity and price are the same between the two. Any other case will return **false**.

```
bool operator!=(const Basket&);
```

After increasing the price of Basket this function again compare two baskets against one another to check if they are still identical or not. It returns true if they are not identical.

Operator Overload Helpers

```
ostream& operator<<(ostream& os, const Basket& bsk);
```

This operator overload allows for the use of language such as:

```
cout << b1;
```

Where b1 is an object of Basket type. This overload will perform the following:

1. It makes a call to the Basket class's **display** function through the **bsk** parameter.
2. It returns the **os** parameter.

Design your class to avoid code duplication (**Reuse your code!)** and make sure that your implementation does not create memory leaks.

In-Lab Main Module

```
#include<iostream>
#include<cstring>
#include"Basket.h"
#include"Basket.h" //intentional

using namespace std;
using namespace sdds;

int main(){

	cout << "-----------------------------------------------" << endl;
	cout << "**** Basket default constructor ****" << endl;
	Basket b;
	cout << b;

	//checking input validity  3 argument constructor
	cout << "-----------------------------------------------" << endl;
	cout << "**** Checking Input Validity (invalid) ****" << endl;

	Basket b1[] = { { "cherry", -2, 2.50 },
	{ "orange", 3, -5.50 },
	{ nullptr, 4, 6.50 }
	};

	for (int i = 0; i < 3; i++){

		cout << b1[i];

	}
	cout << "-----------------------------------------------" << endl;

	cout << "**** Checking Input Validity (valid) ****" << endl;
	Basket b2("apple", 12, 20.50);
	Basket b3("banana", 14, 10.50);
	Basket b4("pears", 10, 12.50);
	cout << b2;
	cout << "-----------------------------------------------" << endl;
	cout << b3;
	cout << "-----------------------------------------------" << endl;
	cout << b4;
	cout << "-----------------------------------------------" << endl;

	//copy constructor
	cout << "**** Validating copy constructor ****" << endl;
	Basket b5 = b3;
	cout << "A new basket of banana is created" << endl;
	cout << b5;
	cout << "-----------------------------------------------" << endl;
	cout << b3;
	cout << "-----------------------------------------------" << endl;

	//copy assignment operator
	cout << "**** Checking copy assignment operator ****" << endl;
	b2 = b4;
	cout << b2;
	cout << "-----------------------------------------------" << endl;
```

```
	cout << b4;
	cout << "-------------------------------------------" << endl;

	//checking identical
	cout << "**** Basket comparing original and copy ****" << endl;
	if (b2 == b4) cout << "Both baskets are carrying the same fruits" << endl;
	else
		cout << "They are not same type of Baskets" << endl;
	cout << "-------------------------------------------" << endl;

	//Increasing the price of original
	cout << "Basket increasing the price of original" << endl;
	b4.addPrice(1.0);
	cout << b4;
	cout << "-------------------------------------------------------------------"
	     << endl;

	//After increasing price checking the identical again
	cout << "**** Basket comparing after increasing the price and quantity ****"
	     << endl;

	if (b2 != b4) cout << "Baskets are not same" << endl;

	return 0;

}
```

EXECUTION EXAMPLE

```
-------------------------------------------
**** Basket default constructor ****
The Basket has not yet been filled
-------------------------------------------
**** Checking Input Validity (invalid) ****
The Basket has not yet been filled
The Basket has not yet been filled
The Basket has not yet been filled
-------------------------------------------
**** Checking Input Validity (valid) ****
Basket Details
Name: apple
Quantity: 12
Price: 20.50
-------------------------------------------
Basket Details
Name: banana
Quantity: 14
Price: 10.50
-------------------------------------------
Basket Details
Name: pears
Quantity: 10
Price: 12.50
-------------------------------------------
**** Validating copy constructor ****
A new basket of banana is created
Basket Details
Name: banana
```

```
Quantity: 14
Price: 10.50
-----------------------------------------
Basket Details
Name: banana
Quantity: 14
Price: 10.50
-----------------------------------------
**** Checking copy assignment operator ****
Basket Details
Name: pears
Quantity: 10
Price: 12.50
-----------------------------------------
Basket Details
Name: pears
Quantity: 10
Price: 12.50
-----------------------------------------
**** Basket comparing original and copy ****
Both baskets are carrying the same fruits
-----------------------------------------
Basket increasing the price of original
Basket Details
Name: pears
Quantity: 10
Price: 13.50
-------------------------------------------------------------
**** Basket comparing after increasing the price and quantity ****
Baskets are not same
Press any key to continue . . .
```

LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload sources.txt, Basket module and the BasketTester.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS06/lab<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

REFLECTION

Study your final solutions for each deliverable of the Workshop, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a file named `reflect.txt` that contains your **detailed description of the topics that you have learned** in completing this workshop and mention any issues that caused you difficulty and how you solved them. Add any other comments you wish to make.

The reflection should also contain any thoughts and learning from the final project milestones when possible.

Reflection Submission

You can submit your reflection 4 to 6 days after your lab session. Upload reflect.txt to your matrix account. Then, run the following command from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS06/reflect <ENTER>

and follow the instructions generated by the command.

To see the when you can submit your reflection and when the submission closes, (like any other submission) add **-due** to the end of your reflection submission command:

```
~profname.proflastname/submit 244/NXX/WS06/reflect -due<ENTER>
```