Part C – Encapsulation

CONSTRUCTORS

Workshop 4 (0.75 – at_home drafted)

In this workshop, you are to initialize the data within an object of class type upon its creation as well as manage its state when the object goes out of scope. This classes used will represent a Flower and a Bouquet.

LEARNING OUTCOMES

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

- to define the special member function that initializes the data at creation time
- to define a default constructor that sets an object to a safe empty state
- to overload a constructor to receive information from a client
- to define a special member function that manages cleanup of dynamic resources at destruction time

SUBMISSION POLICY

The workshop is divided into 3 sections;
in-lab - 30% of the total mark
To be completed before the end of the lab period and submitted from the lab.
at-home - 35% of the total mark
To be completed within 2 days after the day of your lab.
DIY (Do It yourself) – 35% of the total mark
To be completed within 3 days after the at-home due date.
The *in-lab* section is to be completed after the workshop is published, and before the end of the lab session. The *in-lab* is to be submitted during the workshop period form the lab.
If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the

in-lab portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.
If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is 2 days after your scheduled *in-lab* workshop (23:59) (even if that day is a holiday).
The DIY (Do It Yourself) section of the workshop is a task that utilizes the concepts you have done in the in-lab + at-home section. This section is completely open ended with no detailed instructions other than the required outcome. You must complete the DIY section up to 3 days after the at-home section.
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 the DIY part of the workshop, 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 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:

-*In-lab* portion submitted late, with *at-home* portion:
0 for *in-lab*. Maximum of **DIY+at-home**/10 for the workshop.

-at-home or DIY submitted late:

1 to 2 days, -20%, 3 to 7 days -50% after that submission rejected.

-If any of *the at-home* or in-lab portions is missing, the mark for the whole workshop will be **0/10**

-If DIY portion is missing you will lose the mark for the DIY portion of the workshop.

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/WS04/in_lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS04/at_home -due<ENTER>
~profname.proflastname/submit 244/NXX/WS04/DIY -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>
```

IN-LAB (30%)

FLOWER MODULE

The **Flower** class is used as a basic representation of a real flower.

To accomplish the above follow these guidelines:

Design and code a module named `Flower`.

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

In your `Flower` header file, predefine the following constants as integers (in **sdds** namespace).

`NAME_MAX_LEN` with a value of 25.
`COL_MAX_LEN` with a value of 15.

These values are going to be used as the maximum sizes for some of the data members in the Flower class.

Create a class called **Flower**.

PRIVATE MEMBERS:

This class holds the details of a flower using three member variables:

- `f_name`
  This is a **char pointer** that is used to store the name of the flower.
- `f_colour`
  This is a **char pointer** that is used to store the colour of the flower.
- `f_price`
  This is a **double** value that is the price of the flower.

PUBLIC MEMBERS:

Constructors/Destructors

The Flower object will be making uses of **constructors** to handle its object creation. There are two **constructors** that are required:

1. Default constructor – This constructor should set a Flower 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 char pointer** for the Flower's name
   b. A **constant char pointer** for the Flower's colour
   c. A **double value** for the price of the Flower

   These three parameters will be used to initialize a new Flower object. However, they do need to be validated beforehand. **If any of the**

parameters are invalid, instead of setting the Flower with the values as is set the Flower to **a safe empty state**. The definition for validity is as follows:

- The Flower's name **can't be null** or an **empty string**
- The Flower's colour **can't be null** or an **empty string**
- The Flower's price **has to be greater than 0**

In the case that all the parameters are valid, then allocate **new dynamic memory** for the data members that require it and copy the values from the parameters to them.

As the Flower object makes use of dynamic memory, **deallocation** of that memory will be required. When a **Flower** object goes out of scope, **any memory that was newly allocated should be cleaned up**.

Additionally, a message should be printed. If the Flower going out of scope is empty, then it should print followed by a newline:

```
"An unknown flower has departed..." <newline>
```

If the Flower is not empty then it should print (where the [ ] values are substituted with the Flower's colour and name respectively:

```
"[Flower colour] [Flower name] has departed..." <newline>
```

Other Members

```
const char* name() const;
```
This is a query function that returns the name of the Flower.

```
const char* colour() const;
```
This is a query function that returns the colour of the Flower.

```
double price() const;
```
This is a query function that returns the price of the Flower.

```
bool isEmpty()const;
```
returns true if the `Flower` object is Empty and false otherwise.

```
void setEmpty();
```

Sets the `Flower` object into a **safe empty state**.

```
void setName(const char* prompt);
```

This function sets the name of the Flower using data entered from a user. The prompt parameter contains a string that will prompt the user prior to accepting user data for the name. When setting the name, pay careful attention to the length of data given by the user and if that violates any of our constraints. **Additionally, give care to any needs of allocating or deallocating memory**.

If a user enters data of an invalid length, **they should be prompted of their error and asked to try again until they provide a correct value**. The following prompt should be used in this case:

```
"A flower's name is limited to 25 characters... Try again: "
```

```
void setColour(const char* prompt);
```

Simlar to the setName function but for the colour instead. Again, keep in mind of the **length of the data** and **give care to any needs of allocating or deallocating memory**.

If a user enters data of an invalid length, **they should be prompted of their error and asked to try again until they provide a correct value**. The following prompt should be used in this case:

```
"A flower's colour is limited to 15 characters... Try again: "
```

```
void setPrice(const char* prompt);
```

This function sets the price of the Flower. Keep in mind that **the price cannot be less than or equal to zero**. If a user

If a user enters invalid, **they should be prompted of their error and asked to try again until they provide a correct value**. The following prompt should be used in this case:

```
"A flower's price is a non-negative number... Try again: "
```

```
void setFlower();
```

This function will attempt to set every data member of the Flower by

utilizing the other set functions. After setting the data members of the Flower, it will display those details by using the **display()** function. Refer to the sample output section "**Testing Setting an Empty Flower**" for the formatting and prompt messages used.

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

If the object is empty, it will print followed by a new line:

```
"This is an empty flower..."
```

If the object isn't empty then it will print details of the Flower in the following sequence with substitutions for each [ ] element:

```
"Flower: [Flower colour] [Flower name] Price: [Flower price]"
```

There should also be a **newline** at the end of this display whether it is an empty or non-empty Flower object. **Refer to the sample output for details.**

At the end of the function the **cout object** should be **returned**.

UTILITIES

Similar to WS02, the **utils** module has been included in this workshop. You may use the functions there to assist with the coding of the Flower module if you find them useful. **It is advised to take a look at the read function that deals with C-Style strings**.

UTILITIES (OPTIONAL)

In addition to what was provided in WS02, a new function prototype is provided in utils.h and an empty definition is provided in utils.cpp:

```
/*
A set function that generically works with cstring pointer copying w/ dynamic
memory
A src and dest are given as parameters where the latter is a pointer
reference.
If the src isn't nullptr, normal processing occurs otherwise dest is set to
nullptr.
```

```
Assumes dest is allocated or has been set to a safe empty state / nullptr.
The max_len dicates how much memory is to be allocated to the dest at maximum
for the copying / the max number of characters to copy. It is nullbyte non
inclusive.
*/
void copystr(char*& dest, const char* src, unsigned int max_len);
```

This function once completed can provide a mechanism to copy cstrings with dynamic allocation in mind up to a max length. The implementation of this function is **optional** for the purposes of the **in_lab**. However, it may be useful to attempt it so you can use such a utility for the rest of the coding.

A possible step by step approach to this function could be:

- First deallocate any memory that may have been allocated in **dest** pointer
- Check if the **src** cstring is either **empty** or **nullptr**. If it is then simply set dest to be **nullptr**.
- If **src** is a valid string then allocate memory for **dest** based on the length of **src** (remembering to consider the nullbyte). If the length of **src** is greater than max_len then simply use max_len instead.
- Once memory have been allocated then copy the characters from **src** into **dest**. Remember to account for the nullbyte afterwards.

IN-LAB MAIN MODULE

```
/*************************************************************************
// OOP244 Workshop 4: Constructors, Destructors, Current Object IN LAB
// File FlowerTester.cpp
// Version 1.0
// Date     2019/09/26
// Author   Hong Zhan (Michael) Huang
// Description
// Tests the creation, usage and destruction of the Flower class
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*************************************************************************/
#include <iostream>
#include "Flower.h"
using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
```

```
  for (int i = 0; i < len; i++, cout << ch);
    return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}

int main() {
  cout << "Testing Empty Flowers" << endl;
  line(64, '-') << endl;
  number(1) << endl;
  Flower f;
  f.display() << endl;

  cout << "Testing Empty Strings" << endl;
  line(64, '-') << endl;
  number(2) << endl;
  Flower f2[2]{ {"Rose", "", 7.5}, {"", "Ruby", 7.8 } };
  f2[0].display();
  f2[1].display() << endl;

  cout << "Testing Non Empty Flowers" << endl;
  line(64, '-') << endl;
  number(3) << endl;
  Flower f3[3]{ {"Rose", "Red", 2.25} ,{"Rose", "Blue", 9.99} , {"Sunflower",
"Yellow", 1.25} };
  for (int i = 0; i < 3; i++)
    f3[i].display();

  cout << "\nTesting Setting an Empty Flower" << endl;
  line(64, '-') << endl;
  number(4) << endl;
  Flower f4;
  f4.display();
  f4.setFlower();

  cout << "\nTesting Destructor" << endl;
  line(64, '-') << endl;
  number(5) << endl;

  return 0;
}
```

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Testing Empty Flowers
-----------------------------------------------------------------
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
This is an empty flower...

Testing Empty Strings
-----------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
This is an empty flower...
This is an empty flower...

Testing Non Empty Flowers
-----------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
Flower: Red Rose Price: 2.25
Flower: Blue Rose Price: 9.99
Flower: Yellow Sunflower Price: 1.25

Testing Setting an Empty Flower
-----------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
This is an empty flower...
Beginning the birth of a new flower...
Enter the flower's name... : This is not a real flower name
A flower's name is limited to 25 characters... Try again: <ENTER>
A flower's name is limited to 25 characters... Try again: Iris
Enter the flower's colour... : This is not a real colour
A flower's colour is limited to 15 characters... Try again: <ENTER>
A flower's colour is limited to 15 characters... Try again: Pink
Enter the flower's price... : -123
A flower's price is a non-negative number... Try again: 1.2
The flower's current details...
Flower: Pink Iris Price: 1.2

Testing Destructor
-----------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Pink Iris has departed...
Yellow Sunflower has departed...
Blue Rose has departed...
Red Rose has departed...
An unknown flower has departed...
An unknown flower has departed...
An unknown flower has departed...
```

IN-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 utils and Flower modules and the FlowerTester.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/WS04/in_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.

AT_HOME (35%)

FLOWER MODULE (ENHANCED)

In the Flower Module add the following function overloads (prototypes in the header and definitions in the implementation files respectively).

PUBLIC MEMBERS:

```
void setName(const char* name, int len)
```

This function copies the value provided by the name parameter into the Flower's f_name data member. **Pay attention to any dynamic memory allocation needs. If possible, use previously existing code to assist here.**

```
void setColour(const char* colour, int len)
```

This function copies the value provided by the colour parameter into the Flower's f_colour data member. **Pay attention to any dynamic memory allocation needs. If possible, use previously existing code to assist here.**

```
void setPrice(double price)
```

This function copies the value provided by the price parameter into the Flower's f_price data member. If the price given is less than zero then set the price to be 1 instead.

BOUQUET MODULE

A **Bouquet** is an arrangement of **Flowers**. The Bouquet class will aim to represent this idea. It is highly advised to **reuse code** from the in_lab to accomplish this at_home section (Flower and utils).

To accomplish the above follow these guidelines:

Design and code a module named `Bouquet`.

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

In your `Bouquet` header file, predefine the following constants as integers (in **sdds** namespace).

`MESS_MAX_LEN` with a value of 25.
`MAX_FLOWERS` with a value of 3.

These values are going to be used as the maximum sizes for some of the data members in the **Bouquet** class.

Create a class called **Bouquet**.

PRIVATE MEMBERS:

This class holds the details of a flower using three member variables:

- `b_price`
  This is a **double** that is used to store the price of the Bouquet. The price will be the sum of the prices of the individual Flowers that make up the Bouquet.
- `b_message`
  This is a **char pointer** that is used to store a message (Eg. Congratulations) associated with the Bouquet.

- `b_flowers`
  This is a **Flower pointer** that is used to store Flowers in the Bouquet. It will require **dynamic memory allocation**.
- `b_flowerCount`
  This is an **integer** that is used to store the number of Flowers in the Bouquet. This is used much like an index to an array (of Flowers).

PUBLIC MEMBERS:

Constructors/Destructors

The Bouquet object similar to the Flower module will be using constructors. There are two **constructors** that are required:

1. Default constructor – This constructor should set a Bouquet 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 Flower pointer** for the Flowers in the Bouquet
   b. An **integer** for the number of Flowers in the Bouquet which will have a default value of **MAX_FLOWERS** (the constant defined in the header).
   c. A **const char pointer** for the message of the Bouquet

   These three parameters will be used to initialize a new Bouquet object. Similar to the Flower module previously, some validation will need to occur. **If the Flowers pointer provided is nullptr**, set the Bouquet to **a safe empty state**.

   In the case that the **Flowers pointer is valid**, then do the following:
   - Check the integer value provided as the number of flowers and if that number is less and MAX_FLOWERS set the **b_flowersCount** data member to it. If it is greater than MAX_FLOWERS, set b_flowersCount to that **constant**.
   - Allocate **new dynamic memory** for the data member **b_flowers** based on the value of b_flowersCount and copy the values from the constant Flower pointer parameter (name, colour, price). **Consider the use of Flower's public members to accomplish this.**

- Tally the price of all the Flowers into a sum and store that sum in b_price for the price of the Bouquet.
- Lastly, copy the **const char pointer** parameter with the **message** to the data member b_message.

As the Bouquet object makes use of dynamic memory, **deallocation** of that memory will be required. When a **Bouquet** object goes out of scope, **any memory that was newly allocated should be cleaned up**.

Additionally, a message should be printed. If the Bouquet going out of scope is empty, then it should print followed by a new line:

```
"An unknown bouquet has departed..." <newline>
```

If the Bouquet is not empty, then it should print:

```
"A bouquet has departed with the following flowers..." <newline>
```

Other Members

```
const char* message() const;
```
This is a query function that returns the message of the Bouquet.

```
double price() const;
```
This is a query function that returns the price of the Bouquet (This is the sum of the prices of the Flowers in the Bouquet).

```
bool isEmpty()const;
```
returns true if the `Bouquet` object is Empty and false otherwise.

```
void setEmpty();
```
Sets the `Bouquet` object into a **safe empty state**.

```
void setMessage(const char* prompt);
```
This function sets the message of the Bouquet using data entered from an user. The prompt parameter contains a string that will prompt the user prior to accepting user data for the message. When setting the name, pay careful attention to the length of data given by the user and if that violates any of our constraints. **Additionally, give**

care to any needs of allocating or deallocating memory.

If a user enters data of an invalid length, **they should be prompted of their error and asked to try again until they provide a correct value**. The following prompt should be used in this case:

```
"A bouquets's message (non-empty) is limited to 30 characters...
Try again: "
```

Consider the use of similar techniques as in the in_lab for similar set functions.

`void discardBouquet();`

This function will discard a Bouquet by deallocating any memory it had allocated for its data members and then setting the object to a **safe empty state**. It will then print a message followed by a new line:

```
"Discarding the current bouquet..." <newline>
```

`void arrangeBouquet();`

This function will attempt to arrange a Bouquet through the use of user input to define the Flowers in the Bouquet. The sequence of actions this function does is as follows:

- Print out a message of: `"Arranging a new bouquet..." <newline>`
- Check if the current object is empty. If it isn't empty, prompt the user if they wish to discard the current Bouquet. The prompt should be:

  ```
  "This bouquet has an arrangement currently, discard it? (Y/N): "
  ```

  Accept the user's input of a **single char** to be only either Y or N. Any other input is invalid, and they should be prompted to retry:

  ```
  "The decision is either Y or N... Try again: "
  ```

  If the decision ends up being 'Y' then discard the current Bouquet through the use of the **discardBouquet()** member function. **If the decision ends up being 'N' then do nothing for the remainder of the function** except to print out:
  `"No new arrangement performed..." <newline>`

- In the case that the decision previously was 'Y' to discard the current Bouquet or if the Bouquet was empty to being with, proceed with the arrangement. This entails first asking the user the number of Flowers they wish to have in this Bouquet with the following prompt:

  "Enter the number of flowers in this bouquet (Valid: 1-3)... :"

  Note the range of valid values. Take in the user's input and validate it. Prompt them to retry if an invalid value was given:

  "The valid range is 1-3... Try again: "

  Following a valid value, proceed to set the data member b_flowerCount to this value. Use this value to then **allocate dynamic memory** for the Flowers (b_flowers data member) in the Bouquet. After allocating set the value of each Flower using the relevant **set functions** from the Flower module. **Be sure to clear the input stream buffer after setting each Flower instance**.

- Lastly set the b_message for the Bouquet using the relevant member function and prompting the use with:

  "Enter a message to send with the bouquet...: "

- Finish off the arrangement by printing out:

  "A bouquet has been arranged..."

  Consider the use of **public members from Flower and any utils functions** to complete this function. Refer to the sample output section "**Testing Arranging an Empty Bouquet**" for the formatting and prompt messages used.

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

If the object is empty, it will print followed by a new line:

"This is an empty bouquet..." <newline>

If the object isn't empty then it will print details of the Bouquet in the following sequence with substitutions for each [ ] element. The price should be printed with **two decimal place precision**.

```
"This bouquet worth [price] has the following flowers..." <newline>
```

Following the above it will **print the details of each Flower** in the Bouquet. Consider the use of public member functions from Flower here.

There should also be a **newline** at the end of this display whether it is an empty or non-empty Flower object. **Refer to the sample output for details.**

At the end of the function the **cout object** should be **returned**.

AT-HOME MAIN MODULE

```
/*****************************************************************************
// OOP244 Workshop 4: Constructors, Destructors, Current Object AT HOME
// File BouquetTester.cpp
// Version 1.0
// Date       2019/09/29
// Author     Hong Zhan (Michael) Huang
// Description
// Tests the creation, usage and destruction of the Bouquet class
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*****************************************************************************/
#include <iostream>
#include "Flower.h"
#include "Bouquet.h"
using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
  for (int i = 0; i < len; i++, cout << ch);
  return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}

int main() {
```

```
  // A sample set of flowers to test with.
  Flower f1[3]{ {"Rose", "Red", 2.25} ,{"Rose", "Blue", 3.33} , {"Sunflower", "Yellow",
1.25} };

  cout << "Testing An Empty Bouquet" << endl;
  line(64, '-') << endl;
  number(1) << endl;
  Bouquet b;
  b.display() << endl;

  cout << "Testing A Non-Empty Bouquet 2-Args" << endl;
  line(64, '-') << endl;
  number(2) << endl;
  Bouquet b2 { f1, 3 };
  b2.display() << endl;

  cout << "Testing A Non-Empty Bouquet 3-args" << endl;
  line(64, '-') << endl;
  number(3) << endl;
  Bouquet b3{ f1, 3, "Happy Happy Day" };
  b3.display() << endl;

  cout << "\nTesting Arranging an Empty Bouquet" << endl;
  line(64, '-') << endl;
  number(4) << endl;
  Bouquet b4;
  b4.arrangeBouquet();
  b4.display() << endl;

  cout << "\nTesting Arranging an Non-Empty Bouquet (No Discard)" << endl;
  line(64, '-') << endl;
  number(5) << endl;
  b4.arrangeBouquet();
  b4.display() << endl;

  cout << "\nTesting Arranging an Non-Empty Bouquet (Discard)" << endl;
  line(64, '-') << endl;
  number(6) << endl;
  b4.arrangeBouquet();
  b4.display() << endl;

  cout << "\nTesting Discarding a Bouquet" << endl;
  line(64, '-') << endl;
  number(7) << endl;
  b4.discardBouquet();

       cout << "\nTesting Destructors" << endl;
  line(64, '-') << endl;
  number(8) << endl;

       return 0;
}
```

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Testing An Empty Bouquet
----------------------------------------------------------------
```

```
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
This is an empty bouquet...

Testing A Non-Empty Bouquet 2-Args
------------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
This bouquet worth 6.83 has the following flowers...
Flower: Red Rose Price: 2.25
Flower: Blue Rose Price: 3.33
Flower: Yellow Sunflower Price: 1.25
with a message of: Congratulations

Testing A Non-Empty Bouquet 3-args
------------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
This bouquet worth 6.83 has the following flowers...
Flower: Red Rose Price: 2.25
Flower: Blue Rose Price: 3.33
Flower: Yellow Sunflower Price: 1.25
with a message of: Happy Happy Day


Testing Arranging an Empty Bouquet
------------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
Arranging a new bouquet...
Enter the number of flowers in this bouquet (Valid: 1-3)... : 0
The valid range is 1-3... Try again: 2
Beginning the birth of a new flower...
Enter the flower's name... : Iris
Enter the flower's colour... : Pink
Enter the flower's price... : 1.2
The flower's current details...
Flower: Pink Iris Price: 1.20
Beginning the birth of a new flower...
Enter the flower's name... : Dandelion
Enter the flower's colour... : Yellow
Enter the flower's price... : 1.3
The flower's current details...
Flower: Yellow Dandelion Price: 1.30
A bouquet has been arranged...
Enter a message to send with the bouquet...: <enter>
A bouquets's message (non-empty) is limited to 30 characters... Try again: Congrats
This bouquet worth 2.50 has the following flowers...
Flower: Pink Iris Price: 1.20
Flower: Yellow Dandelion Price: 1.30
with a message of: Congrats


Testing Arranging an Non-Empty Bouquet (No Discard)
------------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Arranging a new bouquet...
This bouquet has an arrangement currently, discard it? (Y/N): N
No new arrangement performed...
This bouquet worth 2.50 has the following flowers...
Flower: Pink Iris Price: 1.20
Flower: Yellow Dandelion Price: 1.30
with a message of: Congrats
```

```
Testing Arranging an Non-Empty Bouquet (Discard)
-----------------------------------------------------------------
6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6
Arranging a new bouquet...
This bouquet has an arrangement currently, discard it? (Y/N): Y
Discarding the current bouquet...
Yellow Dandelion has departed...
Pink Iris has departed...
Enter the number of flowers in this bouquet (Valid: 1-3)... : 1
Beginning the birth of a new flower...
Enter the flower's name... : Lily
Enter the flower's colour... : White
Enter the flower's price... : 1.4
The flower's current details...
Flower: White Lily Price: 1.40
A bouquet has been arranged...
Enter a message to send with the bouquet...: Welcome
This bouquet worth 1.40 has the following flowers...
Flower: White Lily Price: 1.40
with a message of: Welcome


Testing Discarding a Bouquet
-----------------------------------------------------------------
7 - 7 - 7 - 7 - 7 - 7 - 7 - 7 - 7 - 7
Discarding the current bouquet...
White Lily has departed...

Testing Destructors
-----------------------------------------------------------------
8 - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8
An unknown bouquet has departed...
A bouquet has departed with the following flowers...
Yellow Sunflower has departed...
Blue Rose has departed...
Red Rose has departed...
A bouquet has departed with the following flowers...
Yellow Sunflower has departed...
Blue Rose has departed...
Red Rose has departed...
An unknown bouquet has departed...
Yellow Sunflower has departed...
Blue Rose has departed...
Red Rose has departed...
```

AT-HOME SUBMISSION

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

If not on matrix already, upload `Bouquet, Flower and utils modules` and the `BouquetTester.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/WS04/at_home<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.

DIY (35%)

...

DIY SUBMISSION

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

If not on matrix already, upload utils and Subject modules and the subjectTester.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/WS04/DIY<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.