CS 201, Fall 2021

Homework Assignment 3

Due: 23:59, December 20, 2021

Flower enthusiasts need a software tool to share their flower knowledge, learn more about flowers, and find flower names from their features. You will develop a flower library to achieve this and you are expected to use the linked list data structure for the implementation.

The flower library that you will implement will have the following functionalities. The details are given below.

- Add a flower
- Remove a flower
- Show the list of flowers
- Show the features of a flower
- Add a new feature to flower
- Remove a feature from a flower
- Find all flowers with a feature

Add a flower (addFlower): The flower library will allow the user to add a new flower by indicating a flower name (which should be a string), and a feature list which is a linked list containing strings. In the initialization, flowers have no feature; therefore, the feature list must be empty. Also flower names should be all lower-case even if the given name has upper-case characters. For simplicity, you can expect that all characters are letter or space characters. If a flower with the specified name already exists, the system should show a warning message about the existence of the flower name. (Please see the code given below and its output for the full signature of the method and format of the output.)

Remove a flower (removeFlower): The flower library will allow the user to remove a certain flower from the collection using the flower name. If a flower with the specified name does not exist, you should display a warning message. If it exists, you should remove all the features from the flower and then remove the flower itself. You should also display a message indicating that the flower has been removed. (Please see the code given below and its output for the full signature of the method and format of the output.)

Show the list of flowers (listFlowers): The flower library will allow the user to see the list of all existing flowers. For each flower, you should display the name and features. The order in which you display (and store) the flowers must be lexical (alphabetical) order. (Please see the code given below and its output for the full signature of the method and format of the output.)

Show the features of a flower (listFeatures): The flower library should allow the user to see the features of a flower. If a flower with that name does not exist, you should display a

warning message. Otherwise, you should display the name and its features in lexical order, and a printout that looks like this: magnolia: citronella scented, subtropical, white color

Note that 'c' is before 's' and 's' is before 'w' in the alphabetical order and therefore in the list. (Please see the code given below and its output for the full signature of the method and format of the output.)

Add new feature to flower (addFeature): The library will allow the user to add a new feature to a flower by specifying the flower name. If the feature already exists for that flower, it should display a warning message. Remember that features should be in lexical order. Thus, the new feature should be stored in the correct place according to that order. (Please see the code given below and its output for the full signature of the method and format of the output.)

Remove a feature from flower (removeFeature): The library should allow the user to remove a certain feature from flower by name and display a warning message if that flower name does not exist in the library or if that feature does not exist in that flower, accordingly. (Please see the code given below and its output for the full signature of the method and format of the output.)

Find all flowers with a feature (findFlowers): The library will allow a user to find all flowers that have the given feature and display their names. If the feature does not exist in the system, it should display a warning message. (Please see the code given below and its output for the full signature of the method and format of the output.)

Note: The definitions below are given partially. You must not modify the given parts, but you may define additional data members and member functions in each class if necessary.

Below is the header file for the Flower class (Flower.h)

```
class Flower{
public:
    Flower();
    Flower(string flowerName);
    Flower(const Flower& aFlower);
    ~Flower();

    bool isEmpty() const;
    int getLength() const ;
    bool add(string feature);
    bool remove(string feature);
    string printFlower() const;

private:
    struct FeatureNode{
        string feature;
        FeatureNode* next;
    };
    int size;
    string flowerName;
    FeatureNode *head;    //the features are stored in a sorted singly linear linked list
```

```
    // ...
    //you may define additional member functions and data members, if necessary

};
```

Below is the header file for the FlowerList class (FlowerList.h)

```
#include "Flower.h"
class FlowerList{
public:
    FlowerList();
    FlowerList(const FlowerList& aList);
    ~FlowerList();

    bool isEmpty() const;
    int getLength() const;
    bool retrieve(string flowerName, Flower& flower) const;
    bool add(string flowerName);
    bool remove(string flowerName);

private:
    struct FlowerNode{
        Flower f;
        FlowerNode* next;
    };
    int size;
    FlowerNode* head;     //the flowers are stored in a sorted singly linear linked list
    // ...
    //you may define additional member functions and data members, if necessary
};
```

Below is the header file for the FlowerLibrary class (FlowerLibrary.h)

```
#include "FlowerList.h"
class FlowerLibrary{
public:
    FlowerLibrary();
    ~FlowerLibrary();

    void addFlower(string name);
    void removeFlower(string name);
    void listFlowers() const;
    void listFeatures(string name) const;
    void addFeature(string name,string feature);
    void removeFeature(string name, string feature);
    void findFlowers(string feature) const;
private:
    FlowerList flowers;
    // ...
    //you may define additional member functions and data members, if necessary
};
```

Below is an example of the main.cpp file that we will use to test your program.
Note: It is crucial that you do not change the include statements in the main code when you create your own. We will use a different main.cpp file to test your program, but it will import the same exact libraries as this one. So the code you submit needs to work with that configuration.

```
#include <iostream>
using namespace std;
#include "FlowerLibrary.h"


int main(){
        FlowerLibrary L;

        L.listFlowers();

        // Testing add flower
        cout << "Testing add flower" << endl;
        cout << endl;

        L.addFlower("Magnolia");
        L.addFlower("SCARLET PIMPERNEL");
        L.addFlower("verbascum");
        cout << endl;

        L.addFlower("Magnolia");
        L.addFlower("magnolia");
        cout << endl;

        L.listFlowers();
        cout << endl;

        // Testing remove flower
        cout << endl;
        cout << "Testing remove flower" << endl;
        cout << endl;

        L.removeFlower("Magnolia");
        L.removeFlower("lantana");
        cout << endl;

        L.removeFlower("LANTANA");
        L.removeFlower("VERBASCUM");
        L.removeFlower("oleander");
        cout << endl;

        L.addFlower("verbascum");
        cout << endl;

        L.listFlowers();
        cout << endl;

        // Testing add feature
        cout << endl;
        cout << "Testing add feature" << endl;
        cout << endl;
```

```
        L.addFlower("calla lilly");
        cout << endl;
        L.listFeatures("magnolia");
        cout << endl;

        L.addFeature("verbascum", "yellow");
        L.addFeature("verbascum", "biennial");
        L.addFeature("verbascum", "Perennial");
        L.addFeature("calla lilly", "ornamental");
        cout << endl;

        L.addFeature("calla lilly", "ornamental");
        L.addFeature("Magnolia", "ornamental");
        cout << endl;

        L.addFlower("magnolia");
        L.addFeature("Magnolia", "ornamental");
        L.addFeature("Magnolia", "citronella scented");

        L.listFeatures("magnolia");
        cout << endl;

        L.listFeatures("calla lilly");
        cout << endl;

        // Testing remove feature
        cout << endl;
        cout << "Testing remove feature" << endl;
        cout << endl;

        L.removeFeature("magnolia", "citronella scented");
        L.removeFeature("magnolia", "yellow");

        L.listFeatures("magnolia");

        // Testing finding flowers
        cout << endl;
        cout << "Testing finding flowers" << endl;
        cout << endl;
        L.findFlowers("ornamental");
        L.findFlowers("yellow");
        cout << endl;
        L.findFlowers("white");

        return 0;
}
```

Sample output of the program looks like:

```
Library is empty.
Testing add flower

magnolia has been added into the library.
scarlet pimpernel has been added into the library.
verbascum has been added into the library.

magnolia cannot be added into the library because it already exists.
magnolia cannot be added into the library because it already exists.

magnolia: No feature
scarlet pimpernel: No feature
verbascum: No feature


Testing remove flower

magnolia has been removed from the library.
lantana cannot be removed because it's not in the library.

lantana cannot be removed because it's not in the library.
verbascum has been removed from the library.
oleander cannot be removed because it's not in the library.

verbascum has been added into the library.

scarlet pimpernel: No feature
verbascum: No feature


Testing add feature

calla lilly has been added into the library.

magnolia isn't found in library

yellow is added into verbascum
biennial is added into verbascum
perennial is added into verbascum
ornamental is added into calla lilly

ornamental already exists in calla lilly
magnolia isn't found in library

magnolia has been added into the library.
ornamental is added into magnolia
citronella scented is added into magnolia
magnolia:citronella scented, ornamental

calla lilly:ornamental
```

```
Testing remove feature

citronella scented is removed from magnolia
yellow doesn't exist in magnolia
magnolia:ornamental

Testing finding flowers
ornamental flowers:calla lilly, magnolia
yellow flowers: verbascum

white flowers: there is no such flower
```

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header files. You MUST use linked-list in your implementation. You will get no points if you use fixed-sized arrays, dynamic arrays or any other data structures such as vectors/arrays from the standard library. However, if necessary, you may define additional data members and member functions.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.
3. Your code must not have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.
4. Otherwise stated in the description, you may assume that the inputs for the functions (e.g., the name of the flower) are always valid so that you do not need to make any input checks.
5. Make sure that each file that you submit (each and every file in the archive) contains your name and student number at the top as comments.

NOTES ABOUT SUBMISSION:

This assignment is due by 23:59 on December 20, 2021. **This homework will be graded by your TA Mahmud Sami Aydın (sami.aydin[at]bilkent.edu.tr). Please direct all your homework-related questions to him.**

1. In this assignment, you must have separate interface and implementation files (i.e., separate .h and .cpp files) for your classes. The file names should be "Flower.h", "Flower.cpp", "FlowerList.h", "FlowerList.cpp", "FlowerLibrary.h" and "FlowerLibrary.cpp". You should also submit other .h and .cpp files if you implement additional classes. We will test your implementation by writing our own main function.Thus, you should not submit any file that contains the main function.

   Although you are not going to submit it, we recommend you to write your own driver file to test each of your functions. However, you SHOULD NOT submit this test code (we will use our own test code).

2. The code (main function) given above is just an example. We will test your implementation also using different main functions, which may contain different function calls. Thus, do not test your implementation only by using this example code. Write your own main functions to make extra tests (however, do not submit these test codes).

3. You should put your "Flower.h", "FlowerList.h", "FlowerLibrary.h" and "FlowerLibrary.cpp" (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder (in this zip file, there should not be any file

containing the main function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number.

The submissions that do not obey these rules will not be graded.

4. **Then, before 23:59 on December 20th, you need to upload this zipped file containing only your header and source codes (but not any file containing the main function) to Moodle.**

   No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

5. You are free to write your programs in any environment (you may use Linux, Mac OS X or Windows). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. If we could not get your program to work properly on the dijkstra machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on "dijkstra.ug.bcc.bilkent.edu.tr" before submitting your assignment.