Evan Nguyen

Joseph Guzman

CECS 275

Spring 2022

Lab 6

Outputs are shown first, then code screenshots.

Output:

```
input
Welcome to your Spring 2022 enrollment center. Please enter your
ID: 025625195
Your first name: Evan
Your last name: Nguyen
--------------------------------------------------------------------------------
Based on your major: COMPUTER ENGINEERING, here are the courses availble to you:
COURSE ID# 1: Spring 2022 CECS 275: MW 5:30 PM - 7:30 PM
COURSE ID# 2: Spring 2022 CECS 346: TuThu 5:30 PM - 7:30 PM
COURSE ID# 3: Spring 2022 MUS 458: ONLINE ASYNCHRONOUS
COURSE ID# 4: Spring 2022 ENGL 317: TuThu 9:30 AM - 10:30 AM
COURSE ID# 5: Spring 2022 REC 340: ONLINE ASYNCHRONOUS
COURSE ID# 6: Spring 2022 THEA 122: ONLINE ASYNCHRONOUS
COURSE ID# 7: Spring 2022 CECS 460: Friday 10:30 AM - 4:30 PM
COURSE ID# 8: Spring 2022 CECS 174: MW 8:00 AM - 9:30 AM
--------------------------------------------------------------------------------
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): 1
Enrolled!
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): 2
Enrolled!
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): 5
Enrolled!
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): 6
Enrolled!
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): 7
Enrolled!
What courses would you like to enroll in? Enter the number(Press F to finish enrolling): F
--------------------------------------------------------------------------------
You have enrolled in these courses:
ID         : 25625195
First Name : Evan
Last Name  : Nguyen
----------Schedule for Spring 2022----------
-> CECS 275: MW 5:30 PM - 7:30 PM
-> CECS 346: TuThu 5:30 PM - 7:30 PM
-> REC 340: ONLINE ASYNCHRONOUS
-> THEA 122: ONLINE ASYNCHRONOUS
-> CECS 460: Friday 10:30 AM - 4:30 PM
--------------------------------------------------------------------------------
Would you like to drop any of these courses (Press F if not)?
Please enter the COURSE # shown previously to drop the selected course.
-> CECS 275: MW 5:30 PM - 7:30 PM
-> CECS 346: TuThu 5:30 PM - 7:30 PM
-> REC 340: ONLINE ASYNCHRONOUS
-> THEA 122: ONLINE ASYNCHRONOUS
-> CECS 460: Friday 10:30 AM - 4:30 PM
5
Deleted!
Would you like to drop any of these courses (Press F if not)?
Please enter the COURSE # shown previously to drop the selected course.
-> CECS 275: MW 5:30 PM - 7:30 PM
-> CECS 346: TuThu 5:30 PM - 7:30 PM
-> THEA 122: ONLINE ASYNCHRONOUS
-> CECS 460: Friday 10:30 AM - 4:30 PM
7
Deleted!
```

```
Would you like to drop any of these courses (Press F if not)?
Please enter the COURSE # shown previously to drop the selected course.
-> CECS 275: MW 5:30 PM - 7:30 PM
-> CECS 346: TuThu 5:30 PM - 7:30 PM
-> THEA 122: ONLINE ASYNCHRONOUS
F
--------------------------------------------------------------------------------
Remaining courses sorted with course number :
ID         : 25625195
First Name : Evan
Last Name  : Nguyen
----------Schedule for Spring 2022----------
-> THEA 122: ONLINE ASYNCHRONOUS
-> CECS 275: MW 5:30 PM - 7:30 PM
-> CECS 346: TuThu 5:30 PM - 7:30 PM


...Program finished with exit code 0
Press ENTER to exit console.
```

Main.cpp

```cpp
/*
 * Answer to Lab 6
 * CECS 275 - Spring 2022
 * @author Evan Nguyen
 * @author Joseph Guzman
 * @version 1.0.0
 *
 */

#include <iostream>
#include "Student.h"
#include "LinkedList.h"
#include "Course.h"
#include <random>
#include <ctime>
#include <string>
#include <vector>

using namespace std;

int main()
{
    int yourID;
    string yourFirst;
    string yourLast;
    cout << "Welcome to your Spring 2022 enrollment center. Please enter your \nID: ";
    cin >> yourID;
    cout << "Your first name: ";
    cin >> yourFirst;
    cout << "Your last name: ";
    cin >> yourLast;

    // create the student object
    Student myStudent = Student(yourID, yourFirst, yourLast);

    // provide a vector of unsorted courses
    vector<Course> catalog = { Course(275, "CECS", "MW 5:30 PM - 7:30 PM", "Spring 2022"),
                               Course(346, "CECS", "TuThu 5:30 PM - 7:30 PM", "Spring 2022"),
                               Course(458, "MUS", "ONLINE ASYNCHRONOUS", "Spring 2022"),
                               Course(317, "ENGL", "TuThu 9:30 AM - 10:30 AM", "Spring 2022"),
                               Course(340, "REC", "ONLINE ASYNCHRONOUS", "Spring 2022"),
                               Course(122, "THEA", "ONLINE ASYNCHRONOUS", "Spring 2022"),
                               Course(460, "CECS", "Friday 10:30 AM - 4:30 PM", "Spring 2022"),
                               Course(174, "CECS", "MW 8:00 AM - 9:30 AM", "Spring 2022")};
    cout << "--------------------------------------------------------------------------------\n";
    cout << "Based on your major: COMPUTER ENGINEERING, here are the courses availble to you:\n";
```

```
    for (int i = 0; i < catalog.size(); i++) {
        cout <<  "COURSE ID# " << i + 1 << ": ";
        cout << catalog[i].getSemester() << " ";
        cout << catalog[i].getName() << " ";
        cout << catalog[i].getNumber() << ": ";
        cout << catalog[i].getTime() << "\n";
    }

    bool isDone = false;
    string selection;

    cout << "-------------------------------------------------------------------------------\n";
    // add course loop
    while (!isDone) {
        cout << "What courses would you like to enroll in? Enter the number(Press F to finish enrolling): ";
        cin >> selection;
        if (selection == "F") {
            isDone = true;
        } else {
            myStudent.addCourse(catalog[stoi(selection) - 1]);
            cout << "Enrolled!\n";
        }

    }
    cout << "-------------------------------------------------------------------------------\n";
    cout << "You have enrolled in these courses:\n";
    myStudent.toString();

    cout << "-------------------------------------------------------------------------------\n";
    bool remove = false;

    while(!remove){
        cout << "Would you like to drop any of these courses (Press F if not)? " << endl;
        cout << "Please enter the COURSE # shown previously to drop the selected course.\n";
        myStudent.showCourses();
        cin >> selection;
        if (selection == "F") {
            remove = true;
        } else {
            myStudent.dropCourse(catalog[stoi(selection) - 1]);
            cout << "Deleted!\n";
        }
    }

    cout << "-------------------------------------------------------------------------------\n";
    cout << "Remaining courses sorted with course number :" << endl;
    myStudent.sortCourse();
    myStudent.toString();

    return 0;
}
```

Student.h

C: > Users > nguye > Documents > OneDrive > CLASS > CECS 275 > CECS-275-LABS > Lab6 > Lab6 > Student.h > S

```
#pragma once
#include "LinkedList.h"
#include <string>

using namespace std;
class Student{
private:
    int studentID;
    string firstName;
    string lastName;
    LinkedList<Course> courseList;

public:
    // Constructors
    Student();
    Student(int ID, std::string first, std::string last, LinkedList<Course> list);
    Student(int ID, std::string first, std::string last);

    // Get functions
    string getFullName() const;
    int getStudentId() const;
    LinkedList<Course> getSchedule() const;

    // Course functions *access the linked list
    void dropCourse(Course c);
    void addCourse(Course c);
    void sortCourse();
    void showCourses() const;

    void toString() const;
};

```

Course.h

C: > Users > nguye > Documents > OneDrive > CLASS > CECS 275 > CE

```
#pragma once
#ifndef COURSE_H
#define COURSE_H

using namespace std;
#include <string>
#include <iostream>

class Course {
    private:
        int courseNumber;
        string courseName;
        string courseTime;
        string courseSemester;
    public:
        Course();
        Course(int, string, string, string);
        void toString();

        // Getters and setters
        void setNumber(int number);
        void setName(string name);
        void setTime(string time);
        void setSemester(string semester);

        int getNumber() const;
        string getName() const;
        string getTime() const;
        string getSemester() const;

        // Operator overloaders
        bool operator==(Course &c);
        bool operator!=(Course &c);
        bool operator<(Course &c);
        bool operator>=(Course &c);
};

#endif
```

LinkedList.h

C: > Users > nguye > Documents > OneDrive > CLASS > CECS 275 > CECS-275-LABS > Lab

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include<iostream>
#include "Course.h"

using namespace std;


//*****************************************
//* ListNode class creates a type used to  *
//* store a node of the linked list.       *
//*****************************************

template <class T>
class ListNode {
    public:
        T value;            // node value
        ListNode<T>* next;  // pointer to the next node

        // Constructor
        ListNode(T nodeValue) {
            value = nodeValue;
            next = nullptr;
        }

};

// ********************
// * LinkedList class *
// ********************

template <class T>
class LinkedList {
    private:
        ListNode<T>* head; // List head pointer
        ListNode<T>* sorted; // List head pointer
    public:
        // Constructor
        LinkedList() {
            head = nullptr;
        }

        // Linked list operations
        ListNode<T>* getHead() const;
        void appendNode(T);
        void deleteNode(T);
        void displayList() const;
        void sortedInsert(ListNode<T>* newnode);
        void insertionsort();
};
```

```cpp
// ***************************************************
// * appendNode appends a node containing the value  *
// * passed into newValue, to the end of the list.   *
// * for lab 6, this is the add function             *
// ***************************************************

template <class T>
void LinkedList<T>::appendNode(T newValue) {
    ListNode<T>* newNode;
    ListNode<T>* nodePtr;
    // allocate a new node and store newValue there
    newNode = new ListNode<T>(newValue);

    // if there are no nodes in the list
    // make newNode the first node
    if (!head) {

        head = newNode;

    } else {

        // initialize nodePtr to head of list
        nodePtr = head;

        // find the last node in the list
        while (nodePtr->next) {

            nodePtr = nodePtr->next;
        }

        nodePtr->next = newNode;
    }
}

// ****************************************************************
// * deleteNode searches for a node with searchValue as its value. *
// * The node, if found, is deleted from the list and from memory  *
// * For lab 6, this is the drop function                          *
// ****************************************************************

template <class T>
void LinkedList<T>::deleteNode(T searchValue) {

    ListNode<T>* nodePtr;       // To traverse the list
    ListNode<T>* previousNode;  // To point to the previous node

    // If the list is empty, do nothing
    if (!head) {

        return;
```

```cpp
// ****************************************************
// * displayList shows the value stored in each node *
// * of the linked list pointed to by head.          *
// ****************************************************

template <class T>
void LinkedList<T>::displayList() const {

    ListNode<T>* nodePtr; // To move through the list

    // Position noedPtr at the head of the list
    nodePtr = head;

    // While nodePtr points to a node, traverse
    // the list
    while (nodePtr) {

        // Display the value in this node
        std::cout << nodePtr->value << std::endl;

        // Move to the next node
        nodePtr = nodePtr->next;
    }
}

template <class T>
ListNode<T>* LinkedList<T>::getHead() const{
    return head;
}


template <class T>
void LinkedList<T>::sortedInsert(ListNode<T>* newnode)
{
    /* Special case for the head end */
    if (sorted == NULL || sorted->value >= newnode->value) {
        newnode->next = sorted;
        sorted = newnode;
    }
    else {
        ListNode<T>* current = sorted;
        /* Locate the node before the point of insertion
         */
        while (current->next != NULL
               && current->next->value < newnode->value) {
            current = current->next;
        }
        newnode->next = current->next;
        current->next = newnode;
    }
}
```

```cpp
// function to sort a singly linked list
// using insertion sort

template <class T>
void LinkedList<T>::insertionsort()
{

    ListNode<T>* current = head;

    // Traverse the given linked list and insert every
    // node to sorted
    while (current != NULL) {

        // Store next for next iteration
        ListNode<T>* next = current->next;

        // insert current in sorted linked list
        sortedInsert(current);

        // Update current
        current = next;
    }
    // Update head to point to sorted linked list
    head = sorted;
}
#endif
```