Dept. of Computer Science & Software Eng., Concordia University
COMP 6481 --- Winter 2021

Programming and Problem Solving

Assignment 3 --- Due Wednesday, April 14, 2021

Part I

Please read carefully: You must submit the answers to all the questions below. However, this part will not be marked. Nonetheless, failing to submit this part fully will result in you missing 50% of the total mark of the assignment.

Question 1

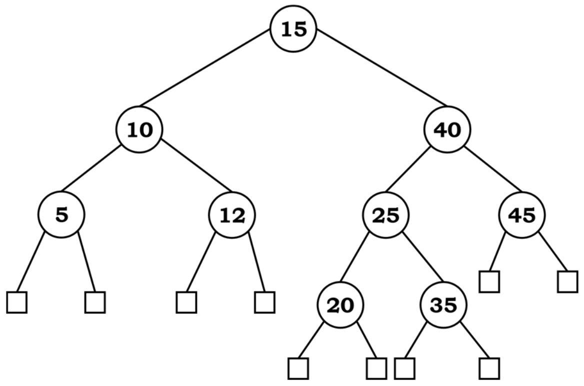
You are given a hash table with size 7 and a hashing function h(k) = k%7. Output the state of the table after inserting the below mentioned values in that specific order when collisions are handled using:

a. Separate chaining
b. Linear probing
c. Double hashing using a second hash function h'(k) = 5 – (k % 5)

The values to be inserted are 19, 26, 13, 48, 17.

Question 2

Consider the below mentioned AVL tree.



a) Insert key 25 and re-balance the tree if required. Show the progress by drawing each state of the tree and the final state after the insertion.

b) Remove node with value 40 from the original tree and re-balance the tree if required. Show the progress by drawing each state of the tree and the final state after the deletion.

Question 3

Develop a **well-documented pseudo code** to create a method for open addressing which is like double hashing but instead of using a second hash function use a pseudorandom number generator seeded by the key.

a. Will this be more efficient approach compared to double hashing?
b. Will you prefer this over double hashing?

Part II

Purpose: The purpose of this assignment is to allow you practice Interfaces, Inner Classes, ArrayLists and Linked Lists, as well as other previous object-oriented and File I/O concepts.

"Education is what remains after one has forgotten what one has learned in school."
-- Albert Einstein

Starting a new phase in education can be an exciting time. The time when you decide step into the world of post-secondary education (college or university). There are a lot of interesting and challenging subjects that you can take, however you should have the right foundation to succed in these courses.

In order to make sure that you have the required skillset to succeed in a course, courses have pre-requites and co-requisites. A prerequisite is one or more courses that you must complete before taking a course at higher grade level. To be accepted into some courses, you will have to prove that you have completed a similar course in the same or a related subject, at a lower grade level. Prerequisites are usually in the same or a related subject, at a lower grade level. A co-requisite on the other hand is a slightly relaxed situation where you may take a specific course if you have finished the co-requisite earlier or even if you register for the co-requisites in the same term. Courses can have both pre-requisites and co-requisites.

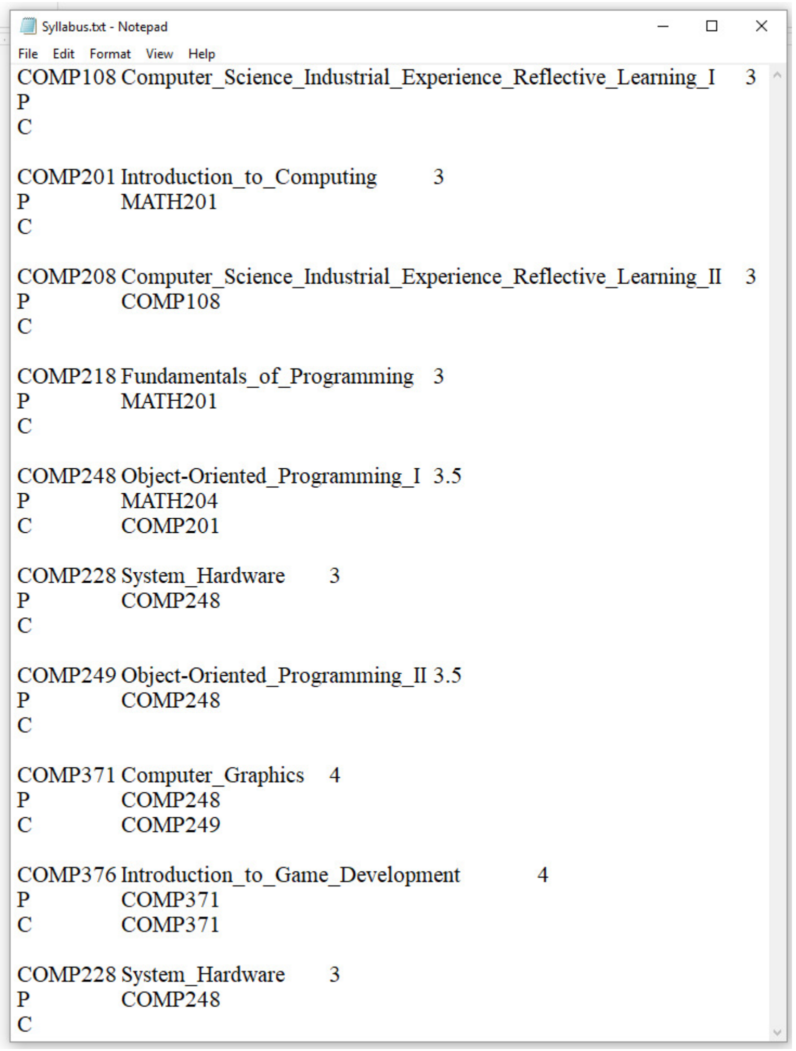
As an example, at Concordia University COMP248 is a pre-requisite for COMP249 and COMP371 is a co-requisite for COMP376. Furthermore, a course can have a same course as both pre-requisite as well as co-requisite (e.g. COMP371 is both pre-requisite and co-requisite for COMP376)

In this assignment, you will design and implement a tool which will determine if a student can enrol in a specific course based on courses he/she has taken so far. You are given few files, namely, Syllabus.txt containing information about various courses, and some Request.txt files (i.e. Request1.txt, Request2.txt, etc.). Each of these files contains one student enrollement request information (courses completed so far and courses he/she wishes to enrol in). Four of these files are provided with the assignment; however, any other similar files can be given and you should expect that your program will be tested with other Request.txt files. You will parse these files to extract course information and will produce an outcome for each of the course student wants to enrol in. The outcome for each course could be one of the below mentioned options where X represents the courseID, P represents coursID for pre-requisite and C represents courseID for co-requisite.

a) Student can enrol in X course as he/she has completed the pre-requisite P.
b) Student can enrol in X course as he/she is enroling for the co-requisite C.
c) Student can't enrol in X course as he/she doesn't have sufficient background needed.

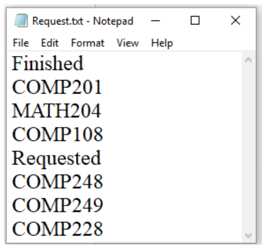
You can assume that every course can have minimum of zero and maximum of one pre-requisite as well as co-requisite. You can also assume that the courses will be listed from basic to advanced to help parse them easily. Request.txt contains a word "Finished" on first line, followed by courseID of those courses. Another word "Requested" after the above information followed by the respective courseIDs. Syllabus.txt contains courseID along with course name (one word seperated by _) and credits assigned to that course.

Next two lines will have P and C indicating pre-requisite and co-requisite respectively fo this course. This set of information is repeated for all the available courses. A sample Syllabus.txt file is depicted in Figure 1 and a sample Request.txt is depicted in Figure 2. A detailed description of all the details that you have to implement for this assignment is avaialble after the two figures.



```
COMP108 Computer_Science_Industrial_Experience_Reflective_Learning_I    3
P
C

COMP201 Introduction_to_Computing       3
P       MATH201
C

COMP208 Computer_Science_Industrial_Experience_Reflective_Learning_II   3
P       COMP108
C

COMP218 Fundamentals_of_Programming     3
P       MATH201
C

COMP248 Object-Oriented_Programming_I   3.5
P       MATH204
C       COMP201

COMP228 System_Hardware 3
P       COMP248
C

COMP249 Object-Oriented_Programming_II  3.5
P       COMP248
C

COMP371 Computer_Graphics       4
P       COMP248
C       COMP249

COMP376 Introduction_to_Game_Development        4
P       COMP371
C       COMP371

COMP228 System_Hardware 3
P       COMP248
C
```

Figure 1. Illustration of Syllabus.txt where P represents pre-requisite and C represents co-requisite



```
Finished
COMP201
MATH204
COMP108
Requested
COMP248
COMP249
COMP228
```

Figure 2. Illustration of Request.txt

I) Create an interface named **DirectlyRelatable** which has a boolean method called isDirectlyRelated (Course C) where C is a object of type Course described in the next part.

II) The **Course** class, which must implement the **DirectlyRelatable** interface, has the following attributes: a courseID (String type), a courseName (String type), a credit (double type), a preReqID (String type), coReqID (String type). It is assumed that course name is always recorded as a single word (_ is used to combine multiple words). It is also assumed that no two courses can have the same courseID.
You are required to write the implementation of the **Course** class. Besides the usual mutator and accessor methods (i.e. getCourseID(), setCoursename()) the class must also have the following:

a) Parameterized constructor that accepts five values and initializes courseID, courseName, credit, preReqID, coReqID to these passed values;
b) Copy constructor, which takes in two parameters, a Course object and a String value. The newly created object will be assigned all the attributes of the passed object, with the exception of the courseID. courseID is assigned the value passed as the second parameter to the constructor. It is always assumed that this value will correspond to the unique courseID rule;
c) clone() method. This method will prompt the user to enter a new courseID, then creates and returns a clone of the calling object with the exception of the courseID, which is assigned the value entered by the user;
d) Additionally, the class should have a toString() and an equals() methods. Two courses are equal if they have the same attributes, with the exception of the courseID, which could be different.
e) This class needs to implement the interface from part I. The method isDirectlyRelated that takes in another Course object C and should return true if C is pre-requisite or co-requisite of the current course object, or vise versa (hence the courses are directly related); otherwise it returns false.

III) The **CourseList** class has the following:

(a) An inner class called **CourseNode**. This class has the following:
  i. Two private attributes: an object of Course and a pointer to a CourseNode object;
  ii. A default constructor, which assigns both attributes to null;
  iii. A parameterized constructor that accepts two parameters, a Course object and a CourseNode object, then initializes the attributes accordingly;
  iv. A copy constructor;
  v. A clone() method;
  vi. Other mutator and accessor methods.

(b) A private attribute called head, which should point to the first node in this list object;

(c) A private attribute called size, which always indicates the current size of the list (how many nodes are in the list);

(d) A default constructor, which creates an empty list;

(e) A copy constructor, which accepts a **CourseList** object and creates a copy of it;

(f) A method called addToStart(), which accepts one parameter, an object from Course class and then creates a node with that passed object and inserts this node at the head of the list;

(g) A method called insertAtIndex(), which accepts two parameters, an object from Course class, and an integer representing an index. If the index is not valid (a valid index must have a value between 0 and size-1), the method must throw a **NoSuchElementException** and terminate the program. If the index is valid, then the method creates a node with the passed Course object and inserts this node at the given index. The method must properly handle all special cases;

(h) A method called deleteFromIndex(), which accepts one integer parameter representing an index. Again, if the index is not valid, the method must throw a NoSuchElementException and terminate the program. Otherwise, node pointed by that index is deleted from the list. The method must properly handle all special cases.

(i) A method called deleteFromStart(), which deletes the first node in the list (the one pointed by head). All special cases must be properly handled.

(j) A method called replaceAtIndex(), which accepts two parameters, an object from Course class, and an integer representing an index. If the index is not valid, the method simply returns; otherwise, the object in the node at the passed index is to be replaced by the passed object;

(k) A method called find(), which accepts one parameter of type String representing a courseID. The method then searches the list for a courseNode with with that courseID. If such an object is found, then the method returns a pointer to that courseNode; otherwise, method returns null. The method must keep track of how many iterations were made before the search finally finds the course or concludes that it is not in the list.

(l) A method called contains (), which accepts a parameter of type String representing a courseID. The method returns true if a course with that courseID is in the list; otherwise, the method returns false.

(m) A method called equals (), which accepts one parameter of type Syllabus. The method returns true if the two lists contain similar courses; otherwise, the method returns false. Recall that two Course objects are equal if they have the same values except for the courseID, which can, and is expected to be, different.

A sample CourseList is demonstrated in Figure 3.

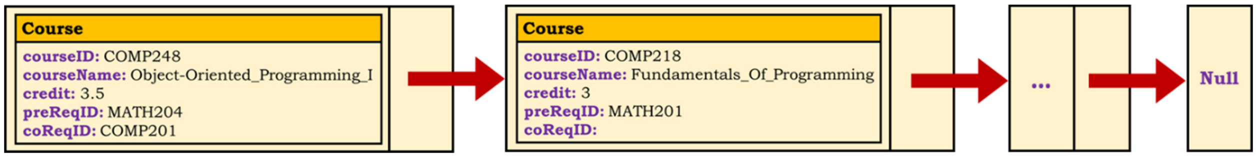


Figure 3. A sample CourseList (LinkedList implementation)

Finally, here are some general rules that you must consider when implementing the above methods:

- Whenever a node is added or deleted, the list size must be adjusted accordingly.
- All special cases must be handled, whether the method description explicitly states that or not.
- All clone() and copy constructors must perform a deep copy; no shallow copies are allowed.
- If any of your methods allows a privacy leak, you must clearly place a comment at the beginning of the method 1) indicating that this method may result in a privacy leak 2) explaining reason behind the privacy leak. Please keep in mind that you are not required to implement these proposals.

IV) Now, you are required to write a public class called **EnrolmentResults**. In the main() method, you must do the following:

(a) Create at least two empty lists from the CourseList class (needed for copy constructor III (e)).

(b) Open the Syllabus.txt file and read its contents line by line. Use these records to initialize one of the CourseList objects you created above. You can use the addToStart() method to insert the read objects into the list. However, the list should not have any duplicate records, so if the input file has duplicate entries, which is the case in the file provided with the assignment for instance, your code must handle this case so that each record is inserted in the list only once.

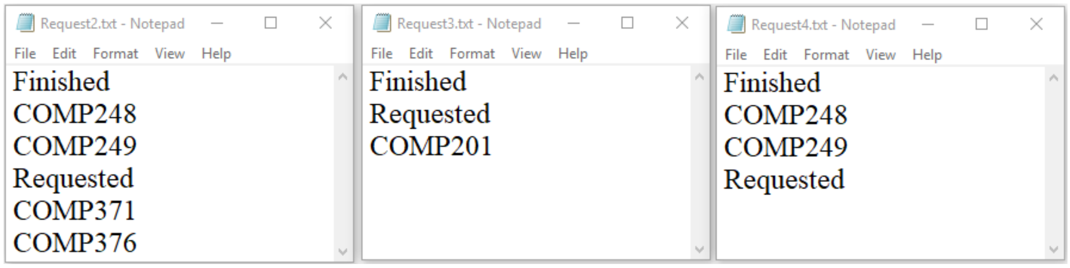
(c) Open a Request.txt file (prompt the user to enter the name of the file that need to be processed, *i.e.* Request3.txt) and create **ArrayList**s from the contents then iterate through each of the courses the student wants to enroll in. Process each of the courses and print the outcome whether student will be able to enroll or not. A sample output for a given file is mentioned below. Again, your program should ask for the file names as your program will be tested against similar input files.

(d) Prompt the user to enter a few courseIDs and search the list that you created from the input file for these values. Make sure to display the number of iterations performed.

(e) Following that, you must create enough objects to test each of the constructors/methods of your classes. The details of this part are left as open to you. You can do whatever you wish as long as your methods are being tested including some of the special cases. You should also make sure to test the isDirectlyRelated() method.

```
Student can enrol in COMP248 course as he/she has completed the pre-requisite MATH204.
Student can't enrol in COMP249 course as he/she doesn't have sufficient background needed.
Student can't enrol in COMP228 course as he/she doesn't have sufficient background needed.
```

Figure 4. A sample outcome of the enrolment request


Request2.txt - Notepad
File Edit Format View Help
Request3.txt - Notepad
File Edit Format View Help
Request4.txt - Notepad
File Edit Format View Help

```
Finished
COMP248
COMP249
Requested
COMP371
COMP376
```

```
Finished
Requested
COMP201
```

```
Finished
COMP248
COMP249
Requested
```

Figure 5. Sample Request.txt files

Outputs for the other sample request.txt files shown in Figure 5 are depicted below.

```
Student can enrol in COMP371 course as he/she has completed the pre-requisite COMP248 and COMP249.
Student can enrol in COMP376 course as he/she is enrolling for co-requisite COMP371.
```

(a)

```
Student can't enrol in COMP201 course as he/she doesn't have sufficient background needed.
```

(b)

```
No enrollment courses found.
```

(c)

Figure 6. Outputs for (a) Request2.txt, (b) Request3.txt, and (c) Request4.txt respectively

Some general information:

a. You should open and close the Syllabus.txt file only once; a better mark will be given for that;
b. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!
c. Again, your program must work for any input files. The files provided with this assignment are only a possible version, and must not be considered as the general case when writing your code.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes.
```
// -----------------------------------------------------
// Assignment (include number)
// Question: (include question/part number, if applicable)
// Written by: (include your name and student id)
// -----------------------------------------------------
```
- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message which includes your name(s).
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy-to-read format.
- End your program with a closing message so that the user knows that the program has terminated.

JavaDoc Documentation:

Documentation for your program must be written in **javaDoc**.
In addition, the following information must appear at the top of each file:

Name(s) and ID(s)	(include full names and IDs)
COMP249	
Assignment #	(include the assignment number)
Due Date	(include the due date for this assignment)

SUBMISSION INSTRUCTIONS

Submission format: All assignment-related submissions must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only – no abbreviations/nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names. **IMPORTANT:** For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You **must** attend the demo and be able to explain their program to the marker. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time slot per group).

Now, please read very carefully:

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**

EVALUATION CRITERIA

IMPORTANT: Part I must fully be submitted. Failure to submit that part will cost 50% of the total marks of the assignment!

Total	**10 pts**
Documentations	**1 pt**
JavaDoc documentations	1 pt
Tasks	**9 pts**
Task#1	1 pts
Task#2	2 pt
Task#3	3 pts
Task#4	3 pts