Dept. of Computer Science & Software Eng., Concordia University
COMP 6481 --- Fall 2023

Programming and Problem Solving

Assignment 2 --- Due Thursday, November 23, 2022

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

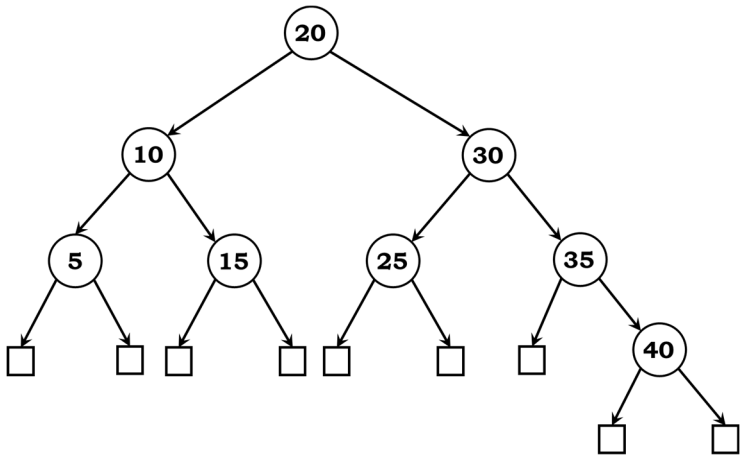
You are given a hash table with size 8 and a hashing function h(k) = k%8. Output the state of the table after inserting the following values in that specific order when collisions are handled using:

a. Separate chaining
b. Linear probing
c. Double hashing using a second hash function h'(k) = 7 – (k % 7)

The values to be inserted are 14, 29, 12, 44, 21.

Question 2

Consider an AVL tree with the following elements: 10, 20, 5, 25, 30, 15, 35, 40.



a) Insert element 12 into the AVL tree and illustrate the tree's balance after insertion.

b) Remove element 20 from the AVL tree and illustrate the tree's balance after removal.

Question 3

Design a well-documented pseudo code for implementing open addressing using a unique method that combines double hashing with pseudorandom number generation. In this approach, the second hash function is not a fixed function but relies on a pseudorandom number generator seeded by the key. Your task is to develop the pseudocode for this method and address the following questions:

a. How does this unique approach with pseudorandom number generation compare to traditional double hashing in terms of collision resolution and efficiency?

b. What are the potential advantages and disadvantages of using pseudorandom numbers for probing in open addressing?

Part II

Purpose: The purpose of this assignment is to allow you practice Array Lists and Linked Lists, as well as other previous object-oriented concepts.

"I fear not the man who has practiced 10,000 kicks once, but I fear the man who has practiced one kick 10,000 times."

--- Bruce Lee

Halloween, celebrated on the night of October 31st, is an enchanting and festive holiday known for its unique blend of spooky and whimsical traditions. Rooted in Celtic and European folklore, Halloween has evolved into a cultural phenomenon embraced by people worldwide. It's a time when children and adults don costumes, transforming into witches, ghosts, superheroes, and various imaginative characters, and venture out for trick-or-treating, parties, and haunted house adventures. Pumpkins are carved into eerie Jack-o'-lanterns, while haunted decorations adorn houses. The holiday is marked by the sweet indulgence of candy, with a variety of confections, from chocolate bars to candy corn, being distributed to eager trick-or-treaters. Halloween's captivating fusion of ancient folklore, creative expression, and communal festivities make it a cherished and intriguing celebration for people of all ages.

Picture an enchanting Halloween evening where children gather for an extraordinary Candy Collection Competition. In this whimsical event, kids don their most imaginative costumes and embark on a candy-collecting adventure like no other. Each child receives a unique, magic-infused bag that can store candies in fantastical ways. The competition takes them through a series of themed candy zones, from the "Witches' Delight" to the "Ghostly Gobstoppers Garden," where they collect candies with exceptional flavors and properties. The most creative and artistically presented candy collection wins a coveted prize—a golden candy goblet filled with enchanted treats. This imaginative Competition ignites the spirit of Halloween, sparking creativity, joy, and sweet memories for all the young participants. These candies have monetary and utilty value (frenzy of the candy), both of which are used to compute the collection value. The top three prizes are decided upon the monetary value of the collection and upon the utility value in case two children are tied on the monetary value. A detailed list of the ten candies is mentioned below.

1. Fizzleberry Firecrunch: This candy crackles with a utility value of 8 and a cost of $2.00.
2. Goblin Gummy Grins: These gummies have a utility value of 7 and are worth $1.50.
3. Moonlight Swirls: These delights offer a refreshing utility value of 9 and cost $1.75.
4. Witch's Brew Bonbons: With a utility value of 6 and priced at $2.50, these bonbons pack a magical punch.
5. Starlight Sugar Sprinkles: These colorful candies have a utility value of 7 and are priced at $1.25.
6. Dragon Scale Delights: These crunchy candies have a utility value of 8 and cost $2.25.
7. Unicorn Rainbow Drops: With a utility value of 9 and a cost of $1.75, these candies bring a burst of color to your collection.
8. Mystic Marshmallows: These fluffy treats boast a utility value of 7 and are worth $1.50.
9. Enchanted Chocolate Truffles: These rich truffles offer a utility value of 9 and are priced at $2.00.
10. Fairy Feathers: These cotton candy wisps have a utility value of 9 and cost $1.25.

In this assignment, you will design and implement a tool which will determine the top three places of the competition. You are given few samples, containing information about participants, their candy collection and a few requests. Each request contains one/more participant names. Your program will have to get the input through scanner and must work for inputs other than the sample input as well. You will input this information and will produce an outcome for each participant present in request(s). The outcome for each request should be one of the following responses where X signifies the participant name.

a) Participant X wins GOLD with collection having highest monetary value
b) Participant X wins GOLD with collection having highest monetary and utility value
c) Participant X wins SILVER with collection having second highest monetary value
d) Participant X wins SILVER with collection having second highest utility value
e) Participant X wins BRONZE with collection having third highest monetary value
f) Participant X wins BRONZE with collection having third highest utility value
g) Participant X is not in top three owing to collection with low monetary value
h) Participant X is not in top three owing to collection with low utility value

You can assume that the competition has finished and all relevant information is made available. The information given to you may not be in any specific order (information structure for each participant will be same but order in which participants are entered will not be same). A sample of participant information input data is depicted in Figure 1 (ParticipantName and number of each type of candy collected), and a sample of request data is depicted in Figure 2. A detailed description of all the details that you have to implement for this assignment is made avaialble after these two figures.

```
Shadowcaster       7 5 4 5 6 7 2 1 0 9
CrypticWraith      2 3 4 5 6 7 2 1 0 9
NightshadeSorcerer 7 0 4 5 1 7 2 1 0 3
PhantomHex         9 5 1 1 1 7 2 1 0 4
EnchantressEclipse 0 5 4 5 6 7 2 1 7 9
HauntedHarbinger   1 1 1 1 1 1 1 1 0 1
MysticMoonshade    9 0 0 0 9 0 9 0 9 9
DarkSpellweaver    7 5 4 5 6 7 2 1 0 6
SoulboundSpecter   2 2 2 8 8 8 2 1 8 9
EerieEldritch      8 8 8 8 8 8 2 1 2 7
```

Figure 1. Illustration of participant information input

```
EnchantressEclipse
EerieEldritch
SoulboundSpecter
HauntedHarbinger
```

Figure 2. Illustration of Requests

I) Create an interface named **Winable** which has a single parameter boolean method called isInTheTopThree (Participant P) where P is an object of type **Participant** described below.

II) The **Participant** class, which must implement **Winable** interface, has the following attributes: a participantID (String), a participantName (String), a candyCollection (int[]). It is assumed that participant name is always recorded as a single word (_ is used to combine multiple words). It is also assumed that no two participants can have same participantID. You are required to write implementation of the **Participant** class. Besides the usual mutator and accessor methods (*i.e.* getParticipantID(), setParticipantName(), *etc.*) class must also have the following:

a) Parameterized constructor that accepts three values and initializes participantID, participantName, candyCollection, and points to these passed values;

b) Copy constructor, that takes in two parameters, a Participant object and a String value. The newly created object will be assigned all the attributes of the passed object, with the exception of the participantID. participantID is assigned value passed as the second parameter to the constructor. It is always assumed that this value will correspond to the unique participantID rule;

c) clone() method, that will prompt a user to enter a new participantID, then creates and returns a clone of the calling object with the exception of the participantID, which is assigned the value entered by the user;

d) Additionally, class will have a toString() and an equals() method. Two participants are equal if they have the same attributes, with the exception of the participantID, which could be different.

e) This class needs to implement interface from part I. The method isInTheTopThree that takes in another Participant object P and should return true if P is from the same group as the current participant object, or vise versa; otherwise it returns false. There will be two groups; one with top three and the other with the rest.

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

(a) An inner class called **ParticipantNode**. This class has the following:

i. Two private attributes: a participant object and a pointer to a ParticipantNode object.

ii. A default constructor, which assigns both attributes to null.

iii. A parameterized constructor that accepts two parameters, a Participant object and a ParticipantNode 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 points to the first node in this ParticipantList.

(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 **ParticipantList** object and creates a copy of it.

(f) A method called addToStart(), which accepts only one parameter, an object from Participant class, 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 the Participant class, and an integer representing an index. If the index is not valid (a valid index must have a value between zero and size-1), then the method must throw a **NoSuchElementException** and terminates the program. If index is valid, then the method creates a node with passed Participant 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 int parameter representing an index. If index is not valid, 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 Participant class, and an integer representing an index. If index is not valid, the method simply returns; otherwise, object in list at passed index must be replaced with the object passed.

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

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

(m) A method called equals (), which accepts one parameter of type ParticipantList. Method returns true if the two lists contain similar participants; otherwise, the method returns false. Recall that two Participant objects are equal if they have the same values except for the participantID, which is expected to be, different.

Finally, here are some general rules that you must consider while implementing 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 **CompetitionResults**. In main() method, you must do the following:

(a) Create at least two empty lists of ParticipantList class (for copy constructor III (e)).

(b) Input participant information as depicted in Figure 1 to initialize one of the ParticipantList objects you created above. You can use the addToStart() method to insert the ParticipantNode objects into the list. However, the list should not have any duplicate records, so if the input has duplicate entries, your code must handle this case so that each record is inserted in the list only once.

(c) Input request information as depicted in figure 2 and create **ArrayList** from the contents. Iterate over each participant. Process each of the participants and print the outcome whether the participant will be in top three or not. A sample output for a given input is mentioned below. Again, your program must ask for the input

information through console as your program will be tested against similar input information.

(d) Prompt the user to enter a few participantIDs and search the list that you created for these values. Make sure to display 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 if your methods are being tested including some of the special cases. You must also test the isInTheTopThree() method.

```
Participant EnchantressEclipse wins BRONZE with collection having third highest utility value
Participant EerieEldritch wins GOLD with collection having highest monetary value
Participant SoulboundSpecter wins SILVER with collection having second highest monetary value
Participant HauntedHarbinger is not in top three owing to collection with low monetary value
```

Figure 3. A sample outcome of the enrolment request

Some general information:

a. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!

b. How about a Candy class. Focus on the optimal design.

c. Again, your program must work for any input data. The sample input provided with this assignment is 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, provide 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 must 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 and ID	(include full name and ID)
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 contain your name(s) and student ID(s). Use "official" name only – no abbreviations/nick names; capitalize the "last" name. Inappropriate submissions will be heavily penalized.

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 demo schedule will be determined and announced by the markers, and students must reserve a time slot for the demo.

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 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 pts
Task#3	3 pts
Task#4	3 pts