COMP 2140 Assignment 2: Sets and Circular Linked Lists

Helen Cameron and Robert Guderian

Due: Friday, October 26, 2018 at 4:30 p.m.

Hand-in Instructions

Go to COMP2140 in UM Learn; then, under "Assessments" in the navigation bar at the top, click on "Assignments". You will find an assignment folder called "Assignment 2". (If you cannot see the assignment folder, follow the directions under "Honesty Declaration" above.) Click the link and follow the instructions. Please note the following:

- Submit ONE .java file file only. The .java file must contain all the source code that you wrote. The .java file must be named `A2<your last name><your first name>.java` (e.g., `A2CameronHelen.java`).
- Please do not submit anything else.
- We only accept homework submissions via UM Learn. Please DO NOT try to email your homework to the instructor or TA or markers — it will not be accepted.
- We reserve the right to refuse to grade the homework or to deduct marks if you do not follow instructions.
- **Assignments become late immediately after the posted due date and time.** Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.

How to Get Help

Your course instructor is helpful: For our office hours and email addresses, see the course website on UM Learn (on the right side of the front page).

For email, please remember to put "[COMP2140]" in the subject and use a meaningful subject, and to send from your UofM email account.

Course discussion groups on UM Learn: A discussion group for this assignment is available in the COMP 2140 course site on UM Learn (click on "Discussions" under "Communications"). Post questions and comments related to Assignment 2 there, and we will respond. Please do not post solutions, not even snippets of solutions there, or anywhere else. We strongly suggest that you read the assignment discussion group for helpful information.

Computer science help centre: The staff in the help centre can help you (but not give you assignment solutions!). See their website at http://www.cs.umanitoba.ca/undergraduate/computer-science-help-centre.php for location and hours. You can also email them at helpctr@cs.umanitoba.ca.

Programming Standards

When writing code for this course, follow the programming standards, available on this course's website on UM Learn. Failure to do so will result in the loss of marks.

Question

Remember: You will need to read this assignment many times to understand all the details of the program you need to write.

Goal: The purpose of this assignment is to write a set class (implemented as a circular linked list) and an application that tests it.

Code you can use: You are permitted to use and modify the code your instructor gave you in class for linked lists You are NOT permitted to use code from any other source, nor should you show or give your code to any other student. Any other code you need for this assignment, you must write for yourself. We encourage you to write ALL the code for this assignment yourself, based on your understanding of circular linked lists — you will learn the material much better if you write the code yourself.

A **set** is simply a collection of items in which there are no duplicate items. Thus, $A = \{1, 2, 3\}$ is a set but $B = \{1, 1, 3, 5\}$ is not because 1 appears twice in B. There is no particular order to the items in a set, but we humans like to see the items in sorted order, if possible.

Write a class that implements a set of integers using an **ordered** circular linked list **with a dummy node.**

Your set class should provide the following operations:

Constructor: Create an empty set.

Insert: Add a new item (an integer) to the set. Please remember that

- No duplicates are allowed.
- The circular linked list is an ordered circular linked list with a dummy node.

Just link in a new node in the correct spot.

Delete: Remove an item (an integer) from the set, if it appears in the set. (Just unlink the node containing the item.)

Union: This operations takes two sets, `setA` and `setB`, and creates a third set (written `setA U setB`) that contains all the items from both `setA` and `setB` — without duplicates, of course.

For example, if `setA` $= \{1, 2, 3, 4\}$ and `setB` $= \{2, 3, 5\}$, then the union of `setA` and `setB` is

$$\texttt{setA U setB} = \{1, 2, 3, 4, 5\}.$$

This operation should *not* change either of the two original sets, `setA` and `setB`. It should create a new, independent set (something like a "deep" copy of a list, rather than a "shallow" copy).

Because you are required to use ordered circular linked lists, you should use an algorithm very similar to the merge step of merge sort:

- Use two pointers, one pointer into each of the two original lists. These pointers should start at the first item in their lists.
- If the two pointers are pointing at two items that are not equal, copy the smaller one into the union and move the pointer to the item after the smaller one.
- If the two pointers are pointing at two equal items, put *one* copy of the item into the union and move *both* pointers to the next items in their sets.

Your implementation should be $O(n)$, if the sum of the two sets' sizes is n. (Note: You should not use `insert()` to insert items into the new set because that changes the algorithm to $O(n^2)$. Instead, you need to remember the last node you added to the new set and insert the next new node after it.)

Difference: This operation also takes two sets, `setA` and `setB`, and creates another set (written `setA \ setB`) that contains all the items from `setA` that are *not* in `setB`. Note that the order of the operands matters: `setA \ setB` is not the same as `setB \ setA`.

For example, if `setA` $= \{1, 2, 3, 4\}$ and `setB` $= \{2, 3, 5\}$, then `setA \ setB` $= \{1, 4\}$ and `setB \ setA` $= \{5\}$.

Like union, this operation should *not* change either of the two original sets, `setA` and `setB`. It should create a new, independent set.

Like union, you can use an algorithm similar to the merge step of merge sort. For the difference algorithm, when `setA` is the first operand and `setB` is the second operand (i.e., when we're creating `setA \ setB`):

- If the `setA` pointer is pointing at an item smaller than the item the `setB` pointer is pointing at, copy the `setA` item into the difference set and move the `setA` pointer to the next item in `setA`.
- If the `setB` pointer is pointing at an item smaller than the `setA` pointer, simply move the `setB` pointer to the next item in `setB`. (Remember that no item in `setB` is in `setA \ setB`.)
- If the pointers are pointing at equal items, then move both pointers to the next items in their sets. (Again, no item in `setB` is in `setA \ setB`.)

Again, your implementation should be $O(n)$, if the sum of the two sets' sizes is n, so don't use `insert()`.

Print: This operation simply prints out the items in the given set, with an appropriate header. The items should surrounded by braces { }, separated by commas, and be printed no more than 10 items per line.

Finally, write a testing method (that is called by your `main` method) that creates an array of sets and performs a sequence of operations on the sets. The testing method should read in the number of sets (i.e., the size of the array) and then the sequence of operations from a file:

- The first line of the file will contain the number of sets.
- After the first line, each line contains one set operation to be performed.
- The first thing on an operation line is a character telling you which operation to perform. The rest of the line contains the operands needed for the operation.
- If the operation is an insert, then the line has the form

  ```
  I setIndex intItem
  ```

  where "`I`" is the character telling you that the operation is insert, `setIndex` is the set you should insert into (i.e., an index into the array of sets that the testing method creates), and `intItem` is the integer item to be added to the set.

  For example,

  ```
  I 0 13
  ```

  means "insert `13` into `sets[0]`" (if "`sets`" is the name of the array of sets that the testing method creates).

- If the operation is a deletion, then the line has the form

  ```
  D setIndex intItem
  ```

where "D" is the character telling you that the operation is delete, setIndex is the set you should delete from (i.e., an index into the array of sets that the testing method creates), and intItem is the integer item to be deleted from the set.

For example,

```
D 1 45
```

means "delete 45 from sets[1]" (if "sets" is the name of the array of sets that the testing method creates).

- If the operation is a set union, then the line has the form

```
U set1 set2 set3
```

where "U" is the character telling you that the operation is union, set1 and set2 are the sets you should union (i.e., they are each an index into the array of sets that the testing method creates), and set3 is the set to which the union should be assigned (another index into the array of sets).

For example,

```
U 0 3 2
```

means "union sets[0] and sets[3], and assign the resulting set to sets[2]".

- If the operation is a set difference, then the line has the form

```
\ set1 set2 set3
```

where "\" is the character telling you that the operation is difference, set1 and set2 are the sets you should take the difference of (i.e., they are each an index into the array of sets that the testing method creates), and set3 is the set to which the difference should be assigned (another index into the array of sets).

For example,

```
\ 2 1 3
```

means "construct sets[2] \ sets[1], and assign the resulting set to sets[3]".

- If the operation is print, then the line has the form

```
P set1
```

where "P" is the character telling you that the operation is print, and set1 is the set you should print (i.e., it is an index into the array of sets that the testing method creates).

Note that backslash ("\") is an "escape character" in Java, so that you must type two backslashes in a String or char to get one backslash — that is, type String "\\" or char '\\'.

You may assume that the input file contains no errors.

The file a2testdata.txt (available in the UM Learn where you got this assignment) contains preliminary test data for you to use to get your program running. Here is sample output corresponding to that input file:

```
COMP 2140 Assignment 2 Fall 2018
Sets and Circular Linked Lists

Set 0
{ }
Set 0
{ 33 }
Set 0
{ 4, 33 }
Set 0
{ 4, 19, 33 }
Set 1
{ 4, 10, 16, 33, 49 }
Set 0
{ 4, 19, 33 }
Set 1
{ 4, 10, 16, 33, 49 }
Set 2
{ 4, 10, 16, 19, 33, 49 }
Set 2
{ 4, 10, 16, 19, 49 }
Set 2
{ 4, 10, 16, 19 }
Set 2
{ 4, 10, 19 }
Set 2
{ 10, 19 }
Set 0
{ 4, 19, 33 }
Set 1
{ 4, 10, 16, 33, 49 }
Set 0
{ 4, 19, 33 }
Set 1
{ 4, 10, 16, 33, 49 }
Set 2
{ 19 }
Set 0
{ 4, 19, 33 }
Set 1
{ 4, 10, 16, 33, 49 }
Set 2
{ 10, 16, 49 }

Program completed normally.
```

We will provide another file `a2data.txt` a few days before the assignment is due.