Gagnaskipan - Data Structures - Final exam spring 2022
In all the assignments you can look at the ***"__main__" == __name__*** part of the given code as well as the text files ***expected_out.txt*** for clarification of how the program should behave.

<u>This is an open book exam, so all data can be used but it is forbidden to use help from any other people, fellow students or not, or to ask specific questions on forums during the exam.</u>
There are Multiple choice questions and five (**5**) programming problems. Good luck!

(20%) Multiple choice

These questions are in a separate quiz assignment on Canvas.

(20%) Problem 1: Arrays and SLL

You are given the class ArrayList. Implement the following operations:

(10%) build_from_SLL(head)

- Takes a singly-linked list head as a parameter.
- After the operation is done the array list should include the same items as the SLL **<u>in the same order</u>**.

(5%) reverse_from_SLL(head)

- Takes a singly-linked list head as a parameter.
- After the operation is done the array list should include the same items as the SLL **<u>in reverse order</u>**.

(5%) build_reverse_SLL_from_array()

- Returns a singly-linked list node that is the head in a list.
- The returned SLL should include the same items as the array list **<u>in reverse order</u>**.

(20%) Problem 2: Recursion

Implement the following operations. You must use recursion for **full marks**. **Half marks** for non recursive solutions.

(10%) division(numerator, denominator)

- Takes in two integer numbers as input
  - Numerator can't be negative and denominator is always positive
- Returns the numerator divided by denominator
  - You are not allowed to use the built in "/" python operator
- **Raise NotDivisbleException** if numerator is not divisible by denominator

(10%) intersection(head1, head2)

- Takes in two heads of singly linked lists
- Returns a the head of a singly linked list that includes all elements that show up in both head1 and head2
  - Head1 and head2 are sets so no duplicates will show up

(10%) Problem 3: DLL

Implement the following function inside the dll class.

(10%) shift_around(n)

- Shifts items from the back of the list to the front
- Shifts the last N items
  - N can be any positive number
  - N can be larger than the size of the list

(15%) Problem 4: Trees

The assignment is to build and use a tree structure to store a family tree. Each node has a *name* and can have *any number of children*.
You can and probably have to add other classes and/or operations to the class.

The following operations need to be implemented on the class FamilyTree:

(10%) build_tree_from_file(file_name)

- Builds the tree using the names and parents in the given file.
- The first line in the file is a single name that should be in the root of the tree, the matriarch!
- Each line after the first has the name of a node followed by the name of the parent node.
- The lines after the first line are in alphabetical order. The order of children on each node should be the same order as they are in the file. This means alphabetical in these files but you do not need to order them alphabetically, only keep their order the same.
- **First 5%**
  - Make this work for the small file.
  - In this file every name will come **after** its parent name.
  - You can trust that the parent will already be in the tree when the child arrives.
- **Second 5%**
  - Make this work for the big file.
  - In this file a name may come **before** its parent name.

(5%) print_preorder() and print_postorder()

- Prints the *name* from each node in the tree, preorder or postorder, in a single line with a regular space after each name

(15%) Problem 5: General programming

You are given the class StringBuilder which has operations to insert text into a string and to remove text from the string. Your assignment is to implement the operations undo() and redo(). In the following descriptions the word ***operation*** refers to either *insert_into_string* or *remove_from_string*.

(10%) undo(self)

- Makes the string the same as it was before the last ***operation*** was run.
- If *undo()* is called again the string becomes the same as before the ***operation*** before that, and so on.
- If there is no operation to undo, do nothing and don't crash.
- Example:

insert_into_string(0, "string")	string: "string"
insert_into_string(3, "something")	string: "strsomethinging"
remove_from_string(5, 7)	string: "strsoing"
undo()	string: "strsomethinging"
undo()	string: "string"
insert_into_string(4, "ki")	string: "striking"
undo()	string: "string"
undo()	string: ""
undo()	string: "" (don't crash)

(5%) redo(self)

- Makes the string the same as it was before *undo()* was called last.
- If *redo()* is called again the string becomes the same as before the *undo()* before that, and so on.
- If there is no *undo()* to redo, do nothing and don't crash.
- If an ***operation*** is called, the set of previous *undo()* to redo should become empty.
- Example:

insert_into_string(0, "string")	string: "string"
insert_into_string(3, "something")	string: "strsomethinging"
remove_from_string(5, 7)	string: "strsoing"
undo()	string: "strsomethinging"
undo()	string: "string"
redo()	string: "strsomethinging"
redo()	string: "strsoing"
undo()	string: "strsomethinging"
undo()	string: "string"
insert_into_string(4, "ki")	string: "striking"
redo()	string: "striking" (nothing happened)
undo()	string: "string"
undo()	string: ""
undo()	string: "" (don't crash)