Data structures - final exam

By handing in this exam the student confirms that she/he has not used any material other than the exam folder itself and her/his otherwise empty programming environment.

You can always make helper classes, helper functions, helper variables or helper parameters with default values, etc. Anything that does not change the way the base functionality is called.

Use the given tests and ***expected_out.txt*** files to better understand the expected functionality.

In P5 the programs take input from the keyboard.
In that case there is an ***extra python script, a tester script***. The tester script opens a file, redirects the input to that file and runs the student program. This basically means the lines in the file are treated as user input. You can use this to make quicker tests but you can also run the student program directly and give input through the keyboard. *The **expected_out.txt** files give the output expected from the unchanged input files but students can either change the input files or make new ones for their specific tests.*
The input handling in this case has also already been programmed. Students are allowed to erase this and/or erase comments or other lines from the code. However, bear in mind that input text and output messages **should be *exactly* as they are in the current code**.

25% Multiple choice.

- The questions are in a separate assignment/quiz on Canvas.

75% Programming problems.

1. **15%**
   You are given the class **ArraySet** that uses *arrays*.
   *Normal limitations on the use of a python list apply. In short you can only use the [ ] operator with a single integer (a statement that returns an integer such as [i+n] is OK, as it sends only a single integer to the [ ] operator) and initialize with [None]*n.*
   You must add to the class ***ArraySet*** the following operations:
   - ***are_duplicates()***: returns **True** if the same value appears more than once in the collection, otherwise **False**.
   - ***remove_duplicates()***: removes all instances but one of each item value in the collection.

   A __*str*__ operation is not part of this assignment but implementing some such output functionality can help students track what their program is doing.

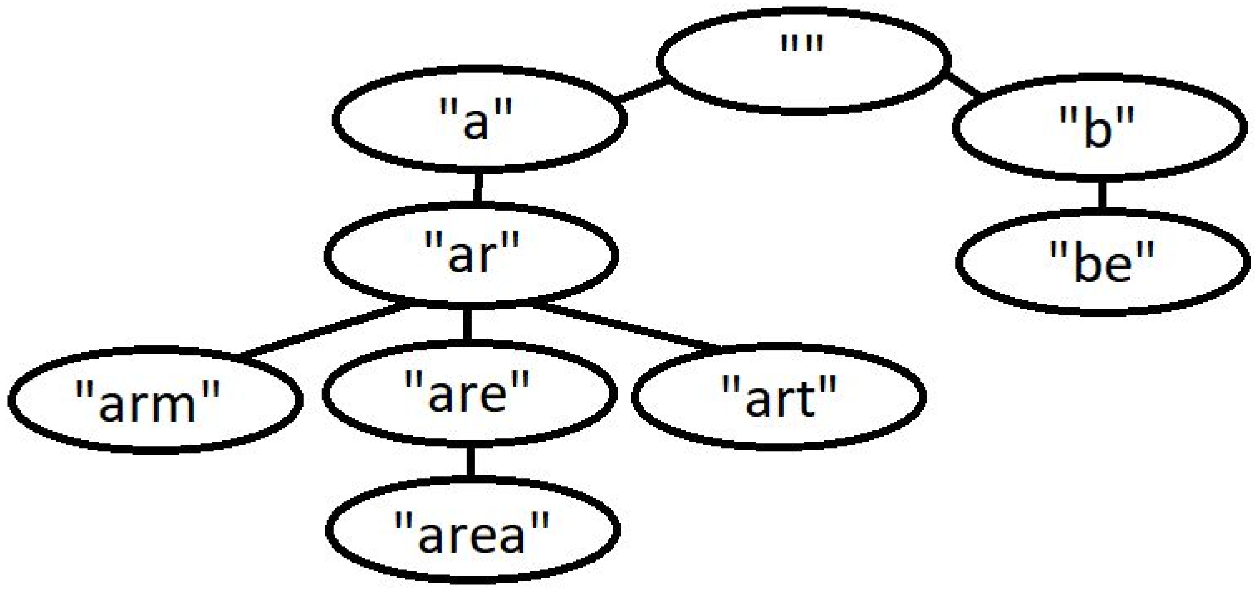
2. **20%**
   - **10% :** Implement the operation ***modulus*** that takes two integer values, ***a** and **b***, and returns an integer. The return value is the number left over from an integer division of ***a*** and ***b***, basically **a % b**.
     You can <u>**not**</u> use the operators ***, /** or **%** or any similar built in or 3rd party functionality. Basically only use **+, -, <, ==, >** and regular flow statements.
     This *must* be implemented with ***recursion***.
   - **10% :** Implement the operation ***get_at*** that takes a singly-linked list (***head***) and an integer value (***position***) and returns the value of the item at that position in the list. The position of the first item (***head***) is **0**.
     If the position is not a valid position in the list raise ***PositionOutOfBounds()***.
     Full marks for a fully *recursive solution*.

3. **15%**
   You are given the class **DLL** that uses *nodes* in a ***doubly-linked list***.
   You must add to the class ***DLL*** the following operations:
   - ***are_duplicates()***: returns **True** if the same value appears more than once in the collection, otherwise **False**.
   - ***remove_duplicates()***: removes all instances but one of each item value in the collection.

   A __*str*__ operation is not part of this assignment but implementing some such output functionality can help students track what their program is doing. Make sure all connections in the *doubly-linked list* are maintained correctly.

4. **15%**
   Finish implementing the class ***AlphabetTree***.
   The structure of this tree has the following rules:
   The data of a node is a string of length equal to the level of the node, so the root has no data (or empty string), the level below the root has length 1 strings, the children of those nodes strings of length 2 etc.
   The data of each node is the same string as its parent, but with one letter added to the end.
   No two nodes in the tree have the same data.
   So the path from the root to a node with the data "track" is:
   "" - "t" - "tr" - "tra" - "trac" - "track"
   The word trace has the path:
   "" - "t" - "tr" - "tra" - "trac" - "trace"
   Note that all the nodes except the last (level 5) one are *the same nodes* (shared by both paths).
   Implement the following operations:
   - ***add_word(word)***: Adds a new node to the tree with the string word as it's data, if it's not already there. In addition it adds all the nodes needed on the path from the root to that node.
   - ***print_all***: Prints the entire contents of the tree, preorder.
   - ***print_leaves***: Prints all the leaves in the tree, from left to right.

   .
   Here is an example of this type of tree:



If the word "***bear***" was now added it would use the nodes "***b***" and "***be***" (they are already there) and add the child "***bea***" to "***be***" and then add the child "***bear***" to "***bea***".
If the word "***by***" was added the node "***by***" would be added as the second child on the node "***b***".

5. **10%**
   Finish implementing the ***item assembly***.
   The program collects items input by the user and counts how many items are added with the same value.
   - If the user inputs ***add*** whatever string follows is added to the assembly.
     - Example: ***add milk***
       The string ***milk*** is added
   - If the user inputs ***remove*** whatever string follows is removed from the assembly.
     - Example: ***remove milk***
       The string ***milk*** is removed (all instances)
   - If the user inputs ***print*** the program outputs all the items in the assembly. Each value is in a separate line followed by a colon (:), a space and the number of such items in the assembly.
   - If the user inputs ***exit*** the program quits (returns).

   All these commands have been programmed into the program's main loop. You only have to implement the program's responses.
   Full marks for optimal time complexity and clear and simple code.