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 P3 and P4 the programs take input from the keyboard.
In both of them there is an ***extra python script, a tester script***. In both cases 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.*
In both of them the input handling 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.

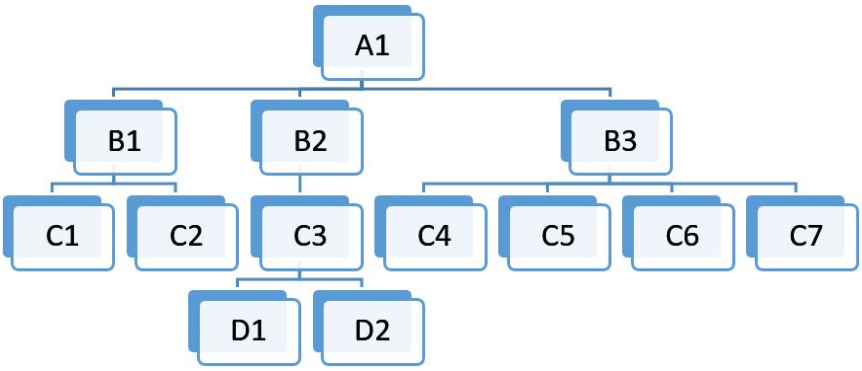
1. **20%**
   Implement the class **Collector** using *arrays*.
   *Normal limitations on the use of a python list apply. In short you can only use the [ ] operator with a single integer (a statement that returns an integer such as [i+n] is OK, as it sends only a single integer to the [ ] operator) and initialize with [None]*n.*
   The class Collector must include the following operations:
   - ***add(value)***: adds the value to the collection
   - ***count(value)***: returns an integer stating how many instances of this value have been added to the collection.

   There must be no limitations on how many items can be added to the set but a newly initialized set must also not allocate an excessive amount of memory.

2. **20%**
    - **10% :** Implement the operation ***divisible*** that takes two integer values, ***a*** *and* ***b***, and returns an integer. The return value is -1 if ***a*** is not divisible (*ísl. deilanlegt*) by ***b***, otherwise the number of times ***a*** is divisible by ***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 ***check_sublist*** that takes two singly-linked lists (head1 and head2) and returns **True** if the second list can be found, in its entirety and in the same order, somewhere in the first list, otherwise **False**.
      Full marks for a fully *recursive solution*.

3. **15%**
   Finish implementing the ***locally stored car database*** program. The program first reads in the file *cars.txt*.
   Each line in the file starts with the name of a car manufacturer. The rest of the line is a list of types of cars from that manufacturer. *The program already reads the list and splits each line into the manufacturer and list of types*. You need to finish this part so that you have the information handy for the rest of the program.
   After the file is read the user can enter commands that are all single words.
   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.

    - If the user enters the name of a manufacturer the program outputs each of its car types in a separate line, but with the manufacturer name in front of it.
        - Example:
          Input: dodge
          Output: dodge aspen
          dodge avenger
          dodge ram
    - If the user inputs a type of car the program outputs that type with the manufacturer name in front.
        - Example:
          Input: ram
          Output: dodge ram
    - If the user inputs ***all*** the program outputs all the types in the database, each in its own line with the manufacturer name in front. The cars should be output in the order in which they are in the file, which is also alphabetical order (*so either is correct, but no other order is*).
    - If the user inputs ***exit*** the program quits (returns).



- A1
  - B1
    - C1
    - C2
  - B2
    - C3
      - D1
      - D2
  - B3
    - C4
    - C5
    - C6
    - C7

4. **20%**

   Here you can see two images representing the same tree. Imagine a tree representing a directory structure in a computer.

   Implement a program that allows the user to traverse such a tree, at any given time being in a current directory. The user can then give commands that affect that current directory or the current location itself.

   The following commands can be given:

   - ***mkdir <name>***: Makes a new node in the tree, under the current node. The new node should be added at the front (leftmost) of the list of children (its siblings).
   - ***cd <name>***: moves the current location (current node) to its child with the name <name>. If the current node has no child node with that name a message is displayed (already written in the base code).
     If <name> is '..' the current location is instead moved to its parent.
     If the current location is the **root** the command 'cd ..' will exit the program.
   - ***ls***: prints the names of all the children (only the children, not the entire subtree) of the current node, each in its own line.

   It is not necessary to use the code given, but all the messages are correctly written there, so use those. It is not necessary to use recursion but a recursive solution may make some things simpler, for example going "up" in the tree ('cd ..').

   For ***5*** of the ***20*** points implement the solution without using any built in python structures (even a list with array limitations). This could be done by implementing the children/siblings in an encapsulating linked list or implementing the sibling list as links within the TreeNode itself. Basically full marks for implementing the entire solution with nodes.