CIS 210, Fall 2016
Introduction to Computer Science

Assignment 2, part 1: Counting majors

This assignment is due at 5pm on Friday, October 7. Use Canvas to turn in counts.py electronically.

Purpose

This is primarily an exercise in careful loop design with a 'case breakdown' of what should happen on the first iteration, different possibilities in the loop body, and after the last iteration.

Pair Assignment

You are encouraged to use *pair programming* to complete this assignment. Work together with one classmate.

Before writing code at the computer, you should work together and independently on the design. Each of you should be able to clearly explain how the program or a part of the program will work. When you are convinced that you both understand how the code will work, then and only then are you ready to write the code.

Counting majors

CIS 210 is taken by nearly all CIS majors, but also by many students majoring in other fields, and by students who have not yet declared a major. What is the distribution? We can find out, based on a column from the class list that gives the major code for each student. I saved that column to a file. Part of that file looks like this:

```
EC
CIS
EC
UNDL
UNDL
CIS
BI
HIST
MACS
CIS
```

We'll create a Python program to count the number of times each major code appears. The output from a different term looked like this:

```
BIC 1
CIS 64
EC 5
GEOG 1
GS 1
HIST 1
```

Counting or summarizing items in a sequence like this is a very common task. There is more than one way to do it; we'll look at an approach that starts by putting items in sorted order (so that each group to be summarized appears together in the sequence) and then scanning them in a loop.

A common loop idiom, sometimes called “control break logic”, is to summarize something for groups of contiguous data items. For example, we might want to summarize the list ['a', 'a', 'a', 'x', 'x'] by counting: 3 a, 2 x. A loop to summarize adjacent items with something in common requires considering at least four cases: The first item in the sequence, an item that starts a new sub-sequence of like items, an item that ends one sub-sequence and starts another, and the last item (or last sub-sequence) in the sequence. See below for discussion of how to handle these.

Requirements

Your program will read a file containing a list of major codes, which may be in arbitrary order. Each major code appears on a line by itself, and every line in the file contains a major code (there are no blank lines). Your program will print one line for each distinct major code along with a count of the number of times that major code appears in the file. For example, suppose file part.txt contains these lines:

```
CIS
CIS
PEN
CIS
UNDL
UNDL
CIS
CIS
GEOG
UNDL
CIS
CIS
MATH
CIS
MACS
CIS
```

Then, running the counts.py program would work like this:

```
> $ python3 counts.py part.txt
CIS 9
GEOG 1
MACS 1
MATH 1
PEN 1
UNDL 3
```

We will print the output in alphabetical order by major code.

Notes and hints on the majors counter

I will provide you with starter code that reads the whole file into an array, and then sorts that array so that lines are in alphabetical order. This will also group all the entries for a major code together. So, for example, the part.txt file above is converted into this sorted list:

```
['CIS', 'CIS', 'CIS', 'CIS', 'CIS', 'CIS', 'CIS', 'CIS',
 'CIS', 'GEOG', 'MACS', 'MATH', 'PEN', 'UNDL', 'UNDL', 'UNDL']
```

You have to provide the logic to count the items in each subsequence (e.g., the 9 occurrences of CIS) and print the counts.

You will, of course, loop through the items in the list. It's easiest to think about your logic first by considering what happens within the loop, dividing into two cases:

- The current item is the same as the item before it. We will increase the count for that item.
- The current item is different from the item before it. This must be the end of one group and the beginning of another. We will print the count for the group we are leaving, and start a new count for the group we are starting.

This logic will have to be adjusted a bit so that we handle the beginning (starting the first group) and the end (summarizing the last group) correctly. If the list contains at least one item, we can initialize the current major code to the first major code in the list, and start the count at 0; then the first case for the loop body above will do the right thing (it will 'continue' counting that group of major codes).

There are a couple of ways to handle the last group. Perhaps the simplest is to add a special, bogus major code at the end of the list. This is called a 'sentinel value', and is a common trick to avoid having to write special code for handling the end of a sequence. Of course we need to be very sure that the sentinel value cannot occur as a major code in the file. We might for example add this statement before the loop:

```
    majors.append("This is not a valid major code")
```

With these careful adjustments to the initialization, the two cases described above for the loop body will handle the first and last group correctly, without any special handling inside or after the loop.

```
    if len(majors) == 0:
        print("File is empty")
        return
```

Here is some starter code to get started, and a sample data file.

Grading

35 points possible

- 15: Program runs and consistently produces correct output for any text file containing a sequence of major codes, one per line. You may assume that valid major codes are sequences of alphabetic characters only (no spaces, punctuation, or special characters).
- 10: Follows CIS 210 coding guidelines, including author identification and header .
- 10: Clarity. The program should not only be consistent with the requirements and approach described here, but it should be very easy to read the program and verify its consistency with the spec.