COSC2320: Data Structures and Algorithms
HW2: Union and Intersection of Doubly Linked Lists Using Recursion

1 Introduction

You will write a C++ program to find the intersection and/or union of two doubly linked lists using recursion. **You are not allowed to use the STL Library.**

A set is a collection of distinct entities regarded as a unit, being either individually specified or (more usually) satisfying specified conditions. In set theory the *union* of two sets A and B, is the is the set of all distinct elements in A and B. The *intersection* of two sets A and B is the set that contains all elements of A that also belong to B (or equivalently, all elements of B that also belong to A), but no other elements. In mathematics, the notion of bag (or multiset) is a generalization of the notion of set in which members are allowed to appear more than once. For this homework we will work with bags.

2 Input

The input is one script file, in the same format as the script file from Homework 1, and two or more data files containing words. The script file will ask you to read 2 or more files into doubly linked lists and operate on them. Two new operations will be introduced:

```
union(L1,L2,L3)
```

will find the *union* of sets L1 and L2 and store the result in L3.

```
intersection(L1,L2,L3)
```

will find the *intersection* of sets L1 and L2 and store the result in L3.

These new operations will require the modification of the script parser provided by the TA's. A new version of the parser is posted.

We will define a word as a sequence of characters. These characters can be any of the 95 printable characters, except the space, which will be used as a separator. Words will be up to 30 characters long and will be separated, as mentioned before, by spaces and carriage returns ('\n'). We will not test you with longer words, and we will not use any non-printable characters other than the carriage return and the EOF character.

Notice that words can be capitalized. This should not interfere with your results: a capitalized word is the same as a non-capitalized word for the purposes of this homework. Uppercase letters can be anywhere in the word, for ex.: "Hungry", "hungry" and "hUngry" should be counted as the same word. By default all uppercase letters should be changed to lowercase. **Words will be separated by one or more spaces and one or more carriage returns. For this homework we will not use other separators.** The input files will contain no punctuation.

Another important detail is that input files might be empty, since the empty set($\emptyset$) is a valid in set theory. You can use the properties of the empty set:

$$A \cup \emptyset = A$$
$$A \cap \emptyset = \emptyset$$

to produce your results.

In order to simplify the problem, the formatting of the original files will be discarded. All uppercase characters must be changed to lower case. When creating the doubly linked lists in memory you should create them in alphabetical order. You should also keep track of how many times each word appears in the input file. Typically this can be done adding a new field to your node, but you are free to devise your own method.

3 Program and output specification

The main program should be called "recursion".

Syntax:

```
recursion script=input.script
```

Note that the file name will not necessarily be the same every time, so your program will have to take that into account. You can use the Command Line Parser that is provided in the TA's homepage.

When asked to output a doubly linked list to a file you must write one word per line followed by a space and the number of occurrences of the word. Please note this change from the previous homework. If the number of occurrences is 0 you must not output the word. Similarly, if the linked list is empty, the file created will be empty. When calculating the intersection of two linked lists you must only keep one copy of each word in the output list.

4 Examples

INPUTS

```
input1.txt
-----------------------------------------------
alpha beta gamma delta Alpha

<EOF>

input2.txt
-----------------------------------------------
cat                  dog bird<EOF>

input3.txt
-----------------------------------------------
alphA

dog
<EOF>
```

input.script

```
read(l1,'input1.txt')
read(l2,'input2.txt')
read(l3,'input3.txt')
union(l1,l2,lu1)
write(lu1,'out1.txt',forward)
union(lu1,l3,lu2)
write(lu2,'out2.txt',reverse)
intersection(l1,l2,li1)
write(li1,'out3.txt',forward)
intersection(l3,lu1,li2)
write(li2,'out4.txt',reverse)
```

RESULTS

out1.txt

```
alpha 2
beta 1
bird 1
cat 1
delta 1
dog 1
gamma 1
<EOF>
```

out2.txt

```
gamma 1
dog 2
delta 1
cat 1
bird 1
beta 1
alpha 3
<EOF>
```

out3.txt

```
<EOF>
```

out4.txt

```
dog 1
alpha 1
<EOF>
```

5 Requirements

- Your program must be able to create and handle multiple doubly linked lists. (Hint: Think about creating a *linked list of linked lists* or an *array of linked lists*)
- When writing to a file you will be asked to write either in alphabetical order or in reverse alphabetical order. Since your linked lists will be created in order, this will involve writing starting from the head or from the tail. You don't need to use recursion to print your files.
- When performing the union and intersection of the linked lists you must use only recursion. Iterators and loops are not allowed. This will be strictly enforced.
- You are allowed to use loops and iterators when reading and writing from a file, in order to recycle the code you used in the previous homework set. However you must create a recursive search method and a recursive list traversing method. (Hint: Look at the example in the book where you are shown how to print a list backwards)
- You must create a log file that records the length of each linked list after each operation. This log file should also be used to output any warnings and errors you might encounter.
- The program should not halt when encountering errors in the script. It should just send a message to the log file and continue with the next line. The only error that is unrecoverable is a missing script file.
- Do not use the STL library.
- Your program should write error messages to the screen. Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.
- Test cases. Your program will be tested with 10 test scripts, going from easy to difficult. You can assume 80% of test cases will be clean, valid input files. If your program fails an easy test script 10-20 points will be deducted. A medium difficulty test case is 10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.
- A program not submitted by the deadline is zero points. A non-working program is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 80 or higher. In general, correctness is more important than speed.