F28PL1 Programming Languages
Laboratory 9

Write a SML program to count how often each unique word appears in a text.

A word is a sequence of upper and lower case letters.

Words and counts are held in a list of tuples, in alphabetical word order.

A)

i) write a function to remove all the non-letters off the front of a list of characters:

```
e.g. strip [#″.″,#″ ‶,#″t#,#″h″,#″e″] ==> [#″t″,#″h″,#″e″]
```

- if the list is empty, return the empty list
- if the list starts with a letter, return the list
- otherwise remove all the non-letters from the start of the tail of the list

ii) write a function to take the next word off a list of characters. The function returns a tuple of the word as a list of characters and the rest of the list.

```
e.g. next [#″t″,#″h″,#″e″,#″ ‶,#″c″,#″a″,#″t″,#″ ‶…] ==>
([#″t″,#″h″,#″e″],[#″ ‶,#″c″,#″a″,#″t″,#″ ‶….]) :
char list * char list
```

- if the list is empty, return a tuple of two empty lists
- if the list starts with a non-letter, return a tuple of the empty list and the list
- otherwise, take the rest of the next word from the tail of the list returning a tuple of the rest of the next word and the rest of the tail of the list, put the head of the original list, which is the head of the word, onto the front of the rest of the word, and return the whole word list and the rest of the tail of the list
- Hint: use a let with a tuple pattern to match the tuple for the rest of word and rest of tail of list

iii) write a function to turn a list of characters into a list of individual words as strings

```
e.g. words [#″t″,#″h″,#″e″,#″ ‶,…] ==>
[‶the″,″cat″,″sat″,″on″,″the″,″mat″] : string  list
```

- if the list is empty, return an empty list
- otherwise, get the first word from the stripped list as a list of characters and the rest of the list, get the list of words from the rest of list and put the imploded first word on the front of the word list
- Hint: use a local definition with a tuple pattern to match the first word and rest of list

B)

iv) write a function to increment the count for a word in a list of words and counts.

```
e.g. incCount ‶sat″ [(‶cat″,1),(‶the″,1)] ==> [(‶cat″,1),
(‶sat″,1),(‶the″,1)]  : (string * int) list
e.g. incCount ‶the″ [(‶cat″,1),(‶mat″,1),(‶on″,1),(‶sat″,1),
(‶the″,1)] ==> [(‶cat″,1),(‶mat″,1),(‶on″,1),(“sat”,1),(“the”,2)] :
 (string * int) list
```

- to increment the count for a word in an empty list, create a list with a tuple for the word and a count of 1
- to increment the count for a word in a list, if the word is the word in the first tuple, return the list with the first tuple’s count incremented
- otherwise if the word comes before the word in the first tuple in the list, add a new tuple for the word and a count of 1 on the front of the list
- otherwise, put the head of the list on the front of the result of incrementing the count for the word in the tail of the list

v) write a function which given a list of words and an empty list, constructs the frequency count list

```
e.g. counts [“the”,”cat”,”sat”,”on”,”the”,”mat”] [] ==>
[(“cat”,1),(mat”,1),(”on”,1),(“sat”,1),(“the”,2)];
```

vi) write a function which given a file name, opens the file, inputs the whole file as a string, explodes the string, makes the word count list, closes the file and returns the word count list

vii) write a function which given a word count list displays it on standard output as a table of words and counts