Assignment 7: Concordances

Course 'Imperative Programming' (IPC031)

Make sure to start working on the assignment **before the lab** session, otherwise you will be too late for the deadline

1 Background

Computers play an important role in the analysis of texts. Concordances are such an example. A concordance is a (usually alphabetically sorted) list of all (or at least the most important) words in a text including their context, to allow careful study of these texts. Before the computer age, concordances were produced manually, which is a time-consuming process, and were conducted only for important texts. In this assignment you design and implement a console program that reads words from a text file and stores them in their order of appearance. Hence, no words are omitted (unless the text file has too many words) and sorting does not take place. After reading the file the user can get information about word frequency, occurrence, and contexts via the console.

2 Learning objectives

After doing this assignment you are able to:

- work with strings;
- work with console I/O that handles user queries and displays information;
- work with text file I/O.

3 Assignment

On Brightspace, "`assignment-07-mandatory-files.zip`" contains a "`main.cpp`" and "`main_test.cpp`", as well as a number of test text files ("`desktop.txt`", "`Vogon poem.txt`", and "`Alice's Adventures In Wonderland.txt`").
The following parts are predefined and must be used in your solution:

- The words are to be stored in the array `content` which can hold at most `MAX_NO_OF_WORDS` strings. The value `MAX_NO_OF_WORDS` is deliberately smaller than the number of words in test text file "`Alice's Adventures In Wonderland.txt`".
- The function to read a single word from an input file stream, `read_word`, must be used in your solution. Note that the purpose of this week's bonus assignment is to create a smarter version of this function, using the same signature (so in the bonus you do not need to change anything else).
- You have a great deal of freedom how to structure your program. However, in order to be able to test your code with "`main_test.cpp`", at least two parts of your solution need to use the functions `enter_command` and `count_command` as described below.

Part 1: Basic user interaction

Design and implement a console program that allows the user to enter the commands:

1. `enter␣`*filename*: the program attempts to open a text file with file name *filename* (*filename* is allowed to contain spaces, *e.g.*, `enter␣`Vogon␣poem.txt). If the open operation is successful, the number of words read is reported to the user. If it fails, then this information is told to the user as well.
   Restrictions:
   - the to be tested part of this command must be embodied in the function `enter_command`;
   - `enter_command` must use the function `read_word` to read the words of the text file.
2. `content`: the program displays only the words that are read from the input text file in their order of appearance.
3. `stop`: the program terminates.

Example:

```
> enter desktop.txt
Successfully read 10 words

> content
Rose
is
a
rose
is
a
rose
is
a
rose.

> stop
>
```

Part 2: Word occurrences

Extend the program created in Part 1 with the following user commands:

4. `count` $word_1 \ldots word_n$: the program counts the number of occurrences of the word sequence $word_1 \ldots word_n$. The program first shows this number to the user, and second shows the total number of words in the text, and third shows the percentage of occurrences of the word sequence with respect to the total number of words in the text.
   Restrictions:
   - the to be tested part of this command must be embodied in the function `count_command`.
5. `where` $word_1 \ldots word_n$: the program first displays all index-positions (the first word having index number 1) of the occurrences of the word sequence $word_1 \ldots word_n$, and second it displays the number of found occurrences.
6. `context` m $word_1 \ldots word_n$: the program first displays all occurrences of the word sequence $word_1 \ldots word_n$, including up to m words immediately preceding and following it, and second it displays the number of found occurrences.

Example:

```
> enter desktop.txt
Successfully read 10 words

> count is a
Found sequence 3 times in 10 words (30%)

> where is a
Found occurrence at index 2
Found occurrence at index 5
Found occurrence at index 8

Found sequence 3 times in 10 words (30%)

> context 3 is a
Rose is a rose is a
is a rose is a rose is a
is a rose is a rose.

Found sequence 3 times in 10 words (30%)
```

4 Products

As product-to-deliver you upload to Brightspace:

- "`main.cpp`" that you have created with solutions for each part of the assignment.
- "`main_test.cpp`" that has been extended with non-trivial unit tests for each new function that you have developed in "`main.cpp`". Note that you should upload this file even if you have not adjusted it.

Deadline

Lab assignment: Friday, October 20, 2023, 23:59h