Computation I 5EIA1
C Exam: Concordance (v0.7, November 8, 2021)
9 November 2021 13:30–16:30

A concordance is a sorted list of words that occur in a text, together with the indices of where the words occur in the text. For example, the concordance for the text:

```
Dead-Sea scrolls were reverse engineered like this!
```

is

```
Concordance
  Dead-Sea: 0
engineered: 4
      like: 5
   reverse: 3
   scrolls: 1
     this!: 6
      were: 2
```

In this exam you will write a program to create concordances.

function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	% per fn	cumulative %
quit	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5%	5%
add word		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16%	21%
print concordance			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	11%	32%
add index							1	1	1	1	1	1	1	1	1	1	1	1	16%	47%
find word at index									1	1	1							1	11%	58%
original text											1							1	5%	63%
remove word												1	1	1				1	16%	79%
read file															1	1	1	1	21%	100%

Figure 1: Test cases and points per task when correctly implemented. Note that you do not have to implement all functions for subsequent tasks.

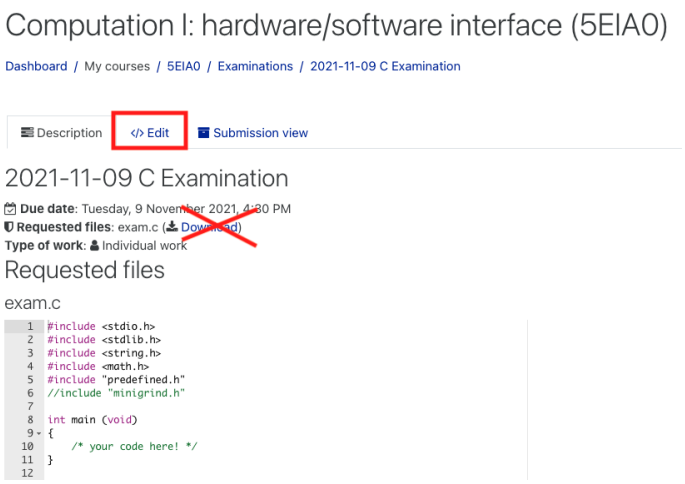
Skipping Tasks by Using Predefined Functions

- In this exam a predefined function is available for a number of tasks. For example `predefined_addIndex`, `predefined_printConcordance`. Therefore, if you get stuck on a task or want to skip it then you can use the predefined function instead of your own function.
- If you use the predefined function for a task anywhere in your code then you will not get points for all of the test cases of that task. For example, if instead of writing your own `addIndex` in Task 5 you use `predefined_addIndex` in later tasks then you will not get the points for test cases 7-8 that are unique for Task 5. You will get points for the other test cases, including those that use `predefined_addIndex`.
- Oncourse will show all the test cases passed, including those using the predefined functions. Points for using predefined functions will be deducted after the exam.
- To use the predefined functions you need to include `#include "predefined.h"` in your program. This should be done automatically when you start editing.
- The predefined functions only work on correct data structures. If an invalid concordance that does not follow the instructions is given to predefined functions then it won't work correctly. An attempt has been made to give error messages when this happens, but they cannot be exhaustive.

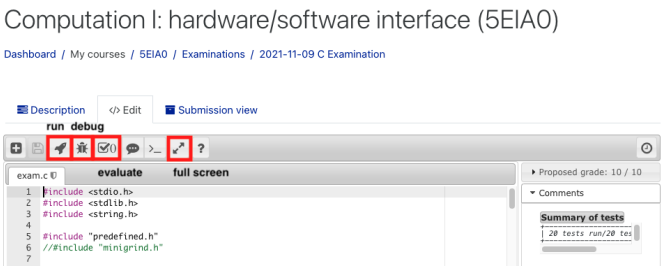
Important

- Your program is automatically submitted at the end of the assignment.
- Your grade is based on the number of cases passed. So try to complete the cases one by one.
- You can press "evaluate" as often as you like during the assignment to evaluate your solution. We advise you to do this regularly during the assignment.
- You must use follow the instructions of the assignment. For example, you may not use an array if a linked-list implementation is asked for.
- You are only allowed to use the Kernighan & Ritchie paper book in the Dutch or English language as a reference during the assignment. No other electronic or printed material are allowed.
- The grade is based on your **last submission**. So make sure you submit a working version that completes as many cases as possible. It's also wise not to make last-minute changes.
- Your program must work for all inputs, not just the test cases. For example, when a `#define MAX 10` must be defined as part of the assignment, then your program should work for all values of `MAX`. We will change the test cases after the assignment (but the number of test cases per task, i.e., the grade of correct programs will not change).
- *Only the standard C libraries described in Appendices B1-B5, B11 of the Kernighan & Ritchie book may be used, unless otherwise indicated.*

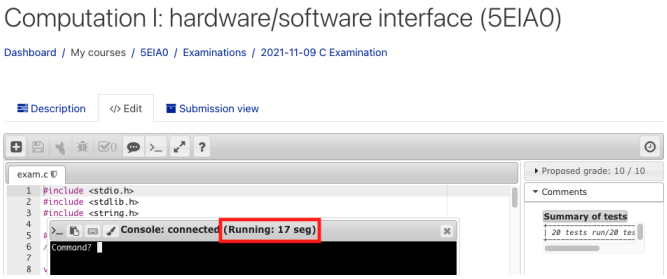
Task 1. Open the correct C programming assignment on oncourse.tue.nl. You will see a screen similar to:



Select the "Edit" tab (*do not select Download*) and you will then see the following screen and initial program:



You can write your own C program in the text editor that is now shown in your browser. Once you press "Save" you can "Run" and "Evaluate" your program. Using the "Run" command you will see a terminal where you can provide input to your program. You can use this to test your program.



Note that "`Running: 17 Seg`" shows the running time; when it does not increase your program has stopped. Close the window by clicking on the small cross in the top-right corner. You can also run the gdb debugger with the "Debug" button, which is useful when your program crashes. Using the "Evaluate" command all test cases are evaluated and at the end the results are displayed (grade and errors if present).
Note that you can move the run/debug window such that you can see your program too. You can scroll (but not edit) your program, which is useful when debugging.

Task 2. Write a C program that asks the user to select the command that needs to be performed. The commands that need to be supported are listed in the following table:

command	operation
q	quit program
w	add word
p	print concordance
i	add index
f	find word at index
o	print original text
W	remove word
r	read file

In this task you only need to implement the quit command. In later tasks you will implement the remainder. Print the error message shown below if an invalid command is given:

```
Command? A
Unknown command 'A'.
Command? q
Bye!
```

Your program must produce the *exact* output, including all spaces, punctuation, capitalisation, quotes, etc.

Hint: You can use the `" %c"` format string for `scanf`.

Your program should now pass test case 1.

Task 3. The `predefined.h` file that you should include in your program with `#include "predefined.h"` contains the following declarations:

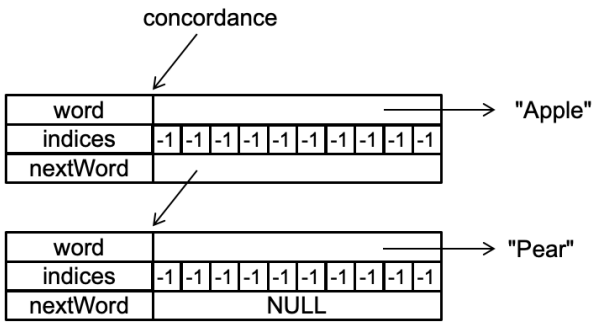
```
#define MAXINDEX 10
typedef struct _node_t {
  char *word;
  int indices[MAXINDEX];
  struct _node_t *nextWord;
} node_t;
extern node_t *predefined_addWord(node_t *concordance, char word[]);
extern void predefined_printConcordance(node_t *concordance);
extern void predefined_addIndex(node_t *concordance, char word[], int index);
extern char *predefined_findWordAtIndex(node_t *concordance, int index);
```

No points will be deducted for including the predefined.h file, only for using the predefined functions. You do *not* need to define the data type `struct _node_t` since it is already defined in the predefined.h file. (You will get a compilation error if you define it yourself too.)

Declare the variable `concordance` of type pointer to `node_t` in your main function and initialise it to NULL. Write a function `node_t *addWord(node_t *concordance, char word[])` that adds the string `word` to the concordance. The concordance is a linked list of words ordered alphabetically. Initialise all entries in the `indices` array with -1. You must use `malloc` to use the minimum amount of space to store the new string. (Do not use `strdup`.) If the word is already in the concordance then do not insert or change the concordance. Do not give an error message either.

Add the 'w' command to the main function such that the user can add a word to the concordance. A word is any sequence of non-white-space characters and can be read using the `"%s"` format string for `scanf`. An example output is:

```
Command? w
Word? Pear
Command? w
Word? Apple
Command? q
Bye!
```



Hint: To keep track of the space you've malloced and freed you can `#include "minigrind.h"`. Comment out the `#include` when evaluating your program because the debug output will invalidate all tests.

Your program should now pass test cases 1-2, and 3-4 after implementing the print concordance function. If you do not wish to implement the function `addWord` then you can use the `predefined_addWord` function, but you will not get the points for test cases 2-4. If you want to use `predefined_addWord` you can call it without adding any code in your program, since it has been declared in the predefined.h file.

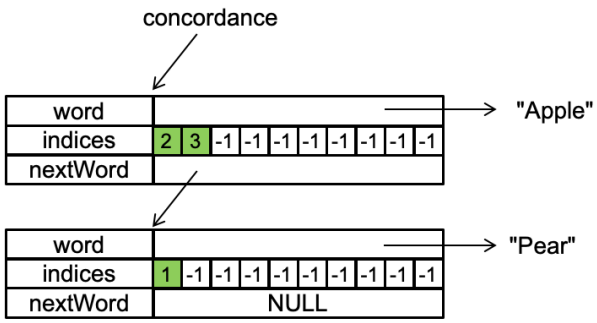
Task 4. Write a function `void printConcordance(node_t *concordance)` that prints all words stored in `concordance`. Add the 'p' command in the main loop. When the concordance contains no words, the function should print a message that the concordance is empty as shown below. Note that the words are right aligned in 10 characters when printed. (Hint: see the field width in K&R Appendix B1.2.) There is no trailing space after the colon :

```
Command? p
The concordance is empty.
Command? w
Word? Coconut
Command? p
Concordance
   Coconut:
Command? w
Word? Kiwi
Command? p
Concordance
   Coconut:
      Kiwi:
Command? w
Word? Banana
Command? p
Concordance
    Banana:
   Coconut:
      Kiwi:
Command? q
Bye!
```

Your program should now pass test cases 1-6. If you do not wish to implement the function `printConcordance` then you can use the `predefined_printConcordance` function, but you will not get the points for test cases 5-6.

Task 5. Write a function `void addIndex(node_t *concordance, char word[], int index)` to add an index to a word in the concordance. Add the 'i' command in the main loop. Use the `indices` array in the `node_t` structure to store the new index. Add the new index in the first empty position (with value -1) in the array. You may assume that the given index is at least zero and is insert only once in the concordance; there is no need to check for this in your code. If all entries in the `indices` array are already taken then exit the function without changing the concordance or printing a message. An example output is:

```
Command? w
Word? Apple
Command? i
Word index? Apple 2
Command? p
Concordance
     Apple: 2
Command? w
Word? Pear
Command? i
Word index? Pear 1
Command? i
Word index? Apple 3
Command? p
Concordance
     Apple: 2 3
      Pear: 1
Command? i
Word index? Lychee 100
Word Lychee not found.
Command? q
Bye!
```



Your program should now pass test cases 1-8. If you do not wish to implement the function `addIndex` then you can use the `predefined_addIndex` function, but you will not get the points for test cases 7-8.

Task 6. Write a function `char *findWordAtIndex(node_t *concordance, int index)` that returns a pointer to the word stored at index in the concordance. Return NULL if there is no word at that index. You can assume that there are no duplicate indices in the concordance. But there may be no word at that index, in which case the message `There is no word at index XX.` (with XX replaced by the index) must be given. Add the 'f' command in the main loop with the behaviour shown below.

```
Command? w
Word? Apple
Command? i
Word index? Apple 2000
Command? i
Word index? Apple 2003
Command? i
Word index? Apple 1984
Command? w
Word? Samsung
Command? i
Word index? Samsung 2015
Command? w
Word? Oppo
Command? i
Word index? Oppo 2020
Command? p
Concordance
     Apple: 2000 2003 1984
      Oppo: 2020
   Samsung: 2015
Command? f
Index? 2003
The word at index 2003 is Apple.
Command? f
Index? 1990
There is no word at index 1990.
Command? q
Bye!
```

Your program should now pass test cases 1-10. If you do not wish to implement the function `findWordAtIndex` then you can use the `predefined_findWordAtIndex` function, but you will not get the points for test cases 9-10.

Task 7. Write a function `void printOriginalText(node_t *concordance, int index)` that prints the original text from the concordance. First, define `maxIndex` in the main function to keep track of the largest index that has been inserted in the concordance. Print words separated by a space all on one line. If there is no word at an index then print a question mark. You can see this in the output below, where there is no word at index 0 and index 2. You can assume that there are no duplicate indices in the concordance. Note that indices are zero or greater. Add the 'o' command in the main loop.

```
Command? o
Command? w
Word? first
Command? w
Word? third
Command? w
Word? fourth
Command? i
Word index? first 1
Command? i
Word index? third 3
Command? i
Word index? fourth 4
Command? p
Concordance
     first: 1
    fourth: 4
     third: 3
Command? o
? first ? third fourth
Command? q
Bye!
```

Your program should now pass test cases 1-11. (There is no predefined function for this task.)

Task 8. Write a function `void removeWord(node_t **concordance, char word[])` that removes the word from the concordance. You must call `free` to return the space of the removed word to the system. Use the command 'W' to remove a word. Make sure to free *all* space that was malloc'd.

```
Command? w
Word? Union
Command? w
Word? Difference
Command? w
Word? Singleton
Command? i
Word index? Difference 1
Command? i
Word index? Difference 6
Command? i
Word index? Singleton 2
Command? i
Word index? Union 9
Command? i
Word index? Union 0
Command? p
Concordance
Difference: 1 6
 Singleton: 2
     Union: 9 0
Command? W
Word? Singleton
Command? p
Concordance
Difference: 1 6
     Union: 9 0
Command? W
Word? Difference
Command? p
Concordance
     Union: 9 0
Command? W
Word? Union
Command? p
The concordance is empty.
Command? q
Bye!
```

Your program should now pass test cases 1-14. (There is no predefined function for this task.)

Task 9. Write a function `void readFile(node_t **concordance, char *filename, int *index)` that inserts all words in the file with name `filename` in the concordance. For example, given the file `test` with contents:

```
Dead-Sea scrolls were reverse engineered like this!
```

then it prints the number of words that were read from this file. If you read multiple files then the indices keep increasing (i.e. it is as if you read one long file). This is correct output:

```
Command? r
File name? test2
Inserted 7 words.
Command? p
Concordance
  Dead-Sea: 0
engineered: 4
      like: 5
   reverse: 3
   scrolls: 1
     this!: 6
      were: 2
Command? r
File name? test2
Inserted 7 words.
Command? p
Concordance
  Dead-Sea: 0 7
engineered: 4 11
      like: 5 12
   reverse: 3 10
   scrolls: 1 8
     this!: 6 13
      were: 2 9
Command? q
Bye!
```

Your program should now pass test cases 1-18. (There is no predefined function for this task.)

Submission: Your *last* submission will be automatically graded.

Input / output test cases

Long lines have been wrapped at 70 characters for legibility. When your program output is compared to the expected output lines will not be wrapped.

Case 01

Input:

```
A
q
```

Output:

```
Command? Unknown command 'A'.
Command? Bye!
```

Case 02

Input:

```
w
Apple
w
Pear
q
```

Output:

```
Command? Word? Command? Word? Command? Bye!
```

Case 03

Input:

```
w
Apple
p
w
Pear
p
w
Apple
p
w
Orange
p
q
```

Output:

```
Command? Word? Command? Concordance
     Apple:
Command? Word? Command? Concordance
     Apple:
      Pear:
Command? Word? Command? Concordance
     Apple:
      Pear:
Command? Word? Command? Concordance
     Apple:
    Orange:
      Pear:
Command? Bye!
```

Case 04

Input:

```
p
w
dd!
w
aa?
w
bb.
w
eee
w
ccc
p
q
```

Output:

```
Command? The concordance is empty.
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Word? Command? Concordance
       aa?:
       bb.:
       ccc:
       dd!:
       eee:
Command? Bye!
```

Case 05

Input:

```
w
Eline
w
Vere
w
Meren
w
Koele
p
w
Kaas
w
Max
w
Havelaar
p
q
```

Output:

```
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Concordance
     Eline:
     Koele:
     Meren:
      Vere:
Command? Word? Command? Word? Command? Word? Command? Concordance
     Eline:
  Havelaar:
      Kaas:
     Koele:
       Max:
     Meren:
      Vere:
Command? Bye!
```

Case 06

Input:

```
w
dddd
w
bb
w
ccc
w
a
w
ggggggg
w
hhhhhhhh
w
iiiiiiiii
w
ffffff
w
eeeee
p
q
```

Output:

```
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Word? Command? Word? Command? Word? Command? Word? Command? Word?
Command? Concordance
         a:
        bb:
       ccc:
      dddd:
     eeeee:
    ffffff:
   ggggggg:
  hhhhhhhh:
 iiiiiiiii:
Command? Bye!
```

Case 07

Input:

```
w
Apple
i
Apple 2
p
w
Pear
i
Pear 1
i
Apple 3
p
i
Lychee 100
q
```

Output:

```
Command? Word? Command? Word index? Command? Concordance
     Apple: 2
Command? Word? Command? Word index? Command? Word index? Command?
Concordance
     Apple: 2 3
      Pear: 1
Command? Word index? Word Lychee not found.
Command? Bye!
```

Case 08

Input:

```
w
stokkens
w
grokkens
w
scholier
w
schalker
i
stokkens 100
i
grokkens 101
i
scholier 200
i
scholier 201
i
scholier 199
i
stokkens 99
w
houtje
p
i
houtje 10
p
i
houtje 20
p
i
houtje 5
p
w
touwtje
i
touwtje 15
i
houtje 12
i
touwtje 13
p
q
```

Output:

```
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Word index? Command? Word index? Command? Word index? Command? Word
index? Command? Word index? Command? Word index? Command? Word?
Command? Concordance
  grokkens: 101
    houtje:
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20 5
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word? Command? Word index? Command? Word index? Command?
Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20 5 12
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
   touwtje: 15 13
Command? Bye!
```

Case 09

Input:

```
f
10
w
Apple
i
Apple 2000
i
Apple 2003
i
Apple 1984
f
2000
f
2003
f
2020
q
```

Output:

```
Command? Index? There is no word at index 10.
Command? Word? Command? Word index? Command? Word index? Command?
Word index? Command? Index? The word at index 2000 is Apple.
Command? Index? The word at index 2003 is Apple.
Command? Index? There is no word at index 2020.
Command? Bye!
```

Case 10

Input:

```
w
Apple
i
Apple 2000
i
Apple 2003
i
Apple 1984
w
Samsung
i
Samsung
2015
w
Oppo
i
Oppo 2020
p
f
1990
f
1984
f
2000
f
2003
f
2020
q
```

Output:

```
Command? Word? Command? Word index? Command? Word index? Command?
Word index? Command? Word? Command? Word index? Command? Word?
Command? Word index? Command? Concordance
     Apple: 2000 2003 1984
      Oppo: 2020
   Samsung: 2015
Command? Index? There is no word at index 1990.
Command? Index? The word at index 1984 is Apple.
Command? Index? The word at index 2000 is Apple.
Command? Index? The word at index 2003 is Apple.
Command? Index? The word at index 2020 is Oppo.
Command? Bye!
```

Case 11

Input:

```
w
Dead-Sea
w
engineered
w
this!
w
scrolls
w
like
w
were
w
reverse
p
i
this!  6
o
i
reverse 3
o
i
Dead-Sea 0
o
i
like 5
o
i
were 2
o
i
scrolls 1
o
i
engineered 4
p
o
q
```

Output:

```
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Word? Command? Word? Command? Word? Command? Concordance
  Dead-Sea:
engineered:
      like:
   reverse:
   scrolls:
     this!:
      were:
Command? Word index? Command? ? ? ? ? ? ? this!
Command? Word index? Command? ? ? ? reverse ? ? this!
Command? Word index? Command? Dead-Sea ? ? reverse ? ? this!
Command? Word index? Command? Dead-Sea ? ? reverse ? like this!
Command? Word index? Command? Dead-Sea ? were reverse ? like this!
Command? Word index? Command? Dead-Sea scrolls were reverse ? like
this!
Command? Word index? Command? Concordance
  Dead-Sea: 0
engineered: 4
      like: 5
   reverse: 3
   scrolls: 1
     this!: 6
      were: 2
Command? Dead-Sea scrolls were reverse engineered like this!
Command? Bye!
```

Case 12

Input:

```
w
Apple
i
Apple 3
p
w
Pear
i
Pear 1
i
Apple 3
w
Lychee
i
Lychee 90
i
Lychee 100
i
Apple 22
w
Empty
p
W
Apple
p
W
Pear
p
W
Empty
p
W
Lychee
p
q
```

Output:

```
Command? Word? Command? Word index? Command? Concordance
     Apple: 3
Command? Word? Command? Word index? Command? Word index? Command?
Word? Command? Word index? Command? Word index? Command? Word index?
Command? Word? Command? Concordance
     Apple: 3 3 22
     Empty:
    Lychee: 90 100
      Pear: 1
Command? Word? Command? Concordance
     Empty:
    Lychee: 90 100
      Pear: 1
Command? Word? Command? Concordance
     Empty:
    Lychee: 90 100
Command? Word? Command? Concordance
    Lychee: 90 100
Command? Word? Command? The concordance is empty.
Command? Bye!
```

Case 13

Input:

```
w
stokkens
w
grokkens
w
scholier
w
schalker
i
stokkens 100
i
grokkens 101
i
scholier 200
i
scholier 201
i
scholier 199
i
stokkens 99
w
houtje
p
i
houtje 10
p
i
houtje 20
p
i
houtje 5
p
w
touwtje
i
touwtje 15
i
houtje 12
i
touwtje 13
p
W
grokkens
W
touwtje
p
W
scholier
p
W
schalker
W
stokkens
p
W
houtje
p
q
```

Output:

```
Command? Word? Command? Word? Command? Word? Command? Word? Command?
Word index? Command? Word index? Command? Word index? Command? Word
index? Command? Word index? Command? Word index? Command? Word?
Command? Concordance
  grokkens: 101
    houtje:
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20 5
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word? Command? Word index? Command? Word index? Command?
Word index? Command? Concordance
  grokkens: 101
    houtje: 10 20 5 12
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
   touwtje: 15 13
Command? Word? Command? Word? Command? Concordance
    houtje: 10 20 5 12
  schalker:
  scholier: 200 201 199
  stokkens: 100 99
Command? Word? Command? Concordance
    houtje: 10 20 5 12
  schalker:
  stokkens: 100 99
Command? Word? Command? Word? Command? Concordance
    houtje: 10 20 5 12
Command? Word? Command? The concordance is empty.
Command? Bye!
```

Case 14

Input:

```
w
houtje
W
houtje
p
w
boutje
i
boutje 10
p
i
boutje 20
p
W
boutje
i
boutje 5
w
boutje
i
boutje 22
p
W
boutje
w
snauwtje
i
snauwtje 33
i
snauwtje 44
p
W
snauwtje
p
q
```

Output:

```
Command? Word? Command? Word? Command? The concordance is empty.
Command? Word? Command? Word index? Command? Concordance
    boutje: 10
Command? Word index? Command? Concordance
    boutje: 10 20
Command? Word? Command? Word index? Word boutje not found.
Command? Word? Command? Word index? Command? Concordance
    boutje: 22
Command? Word? Command? Word? Command? Word index? Command? Word
index? Command? Concordance
  snauwtje: 33 44
Command? Word? Command? The concordance is empty.
Command? Bye!
```

Case 15

Input:

```
p
r
test2
p
q
```

Output:

```
Command? The concordance is empty.
Command? File name? Inserted 7 words.
Command? Concordance
  Dead-Sea: 0
engineered: 4
      like: 5
   reverse: 3
   scrolls: 1
     this!: 6
      were: 2
Command? Bye!
```

Case 16

Input:

```
p
r
test1
p
q
```

Output:

```
Command? The concordance is empty.
Command? File name? Inserted 15 words.
Command? Concordance
       are: 3
      file: 8
     hello: 0
      here: 14
       how: 2 5
        is: 6 9 13
        it: 10 12
     there: 1 11
      this: 7
       you: 4
Command? Bye!
```

Case 17

Input:

```
p
r
test1
p
r
test2
p
r
no-file
q
```

Output:

```
Command? The concordance is empty.
Command? File name? Inserted 15 words.
Command? Concordance
       are: 3
      file: 8
     hello: 0
      here: 14
       how: 2 5
        is: 6 9 13
        it: 10 12
     there: 1 11
      this: 7
       you: 4
Command? File name? Inserted 7 words.
Command? Concordance
  Dead-Sea: 15
       are: 3
engineered: 19
      file: 8
     hello: 0
      here: 14
       how: 2 5
        is: 6 9 13
        it: 10 12
      like: 20
   reverse: 18
   scrolls: 16
     there: 1 11
      this: 7
     this!: 21
      were: 17
       you: 4
Command? File name? Cannot open file no-file.
Command? Bye!
```

Case 18

Input:

```
r
test2
o
i
like 10
i
like 9
p
o
f
5
f
10
f
9
f
0
o
W
like
f
5
f
3
o
W
Dead-Sea
W
this!
p
W
reverse
W
scrolls
W
were
o
W
engineered
p
q
```

Output:

```
Command? File name? Inserted 7 words.
Command? Dead-Sea scrolls were reverse engineered like this! ?
Command? Word index? Command? Word index? Command? Concordance
  Dead-Sea: 0
engineered: 4
      like: 5 10 9
   reverse: 3
   scrolls: 1
     this!: 6
      were: 2
Command? Dead-Sea scrolls were reverse engineered like this! ? ? like
like
Command? Index? The word at index 5 is like.
Command? Index? The word at index 10 is like.
Command? Index? The word at index 9 is like.
Command? Index? The word at index 0 is Dead-Sea.
Command? Dead-Sea scrolls were reverse engineered like this! ? ? like
like
Command? Word? Command? Index? There is no word at index 5.
Command? Index? The word at index 3 is reverse.
Command? Dead-Sea scrolls were reverse engineered ? this! ? ? ? ?
Command? Word? Command? Word? Command? Concordance
engineered: 4
   reverse: 3
   scrolls: 1
      were: 2
Command? Word? Command? Word? Command? Word? Command? ? ? ? ?
engineered ? ? ? ? ? ?
Command? Word? Command? The concordance is empty.
Command? Bye!
```