SWE2024-41: System Programming Assignment (Spring 2023)

Programming Assignment #1

Due: April 23th, 11:59 PM

1. Introduction

You may get used to UNIX file I/O and data structure with this assignment.

2. Specification

In this assignment, your goal is to read a text file with the file name entered as a command line input and write a code that performs a specific search function. After your program starts, your program waits for user's keyword input and performs searching function mentioned below.

① Searching single word locations

If your program receives input with a single word, search for a word's location in a given file.

- Find the word in the file and print it out on `stdout` in the form of

`"[line number]:[start index of the word]"`

② Searching several words locations

If your program receives input with multiple word (separated with a single space and no wrapper), search for lines containing all of the words.

- Find the line(s) containing all of the words in the file and print it on `stdout` in the form of

`"[line number]"`

③ **Searching several consecutive words locations**

If your program receives a phrase input wrapped in double quotation marks (“”), search the lines containing the phrase. More than 2 words can be given as input

- There are no linebreaks (‘₩n’) inside the phrase.
- Continuous spaces or tabs may exist inside the phrase.
- Find the line containing received phrase in the file and print it on `stdout` in the form of

`"[line number]:[start index of the phrase]"`

- If several duplicates of the phrase exist inside a single line, print all of them out as described below. Example of Input File Line: HA HA HA HA /* first line of input file */

```
$ "HA HA"
1:0 1:3 1:6
```

④ **Searching simple regular expressing keyword locations**

If your program receives input in which asterisk sign(*) is between two rods like `[word1] * [word2]`, print the location of the keyword that contains at least one word between `word1` and `word2`. In this case, input values are always two rods.

- There are no spaces between `[word1]` and *, or * and `[word2]`
- Do not consider the case of `[word2]*[word1]`
- Find the location of keyword explained above and print it on `stdout` in the form of

`"[line number]"`

⑤ **Supplementary explanations**

- The definitions of **line number** and **start index of the word (or phrase)**

  * **line number** – the index of the line which contains the keyword. **You must count empty line. Starts from 1**

  * **start index of the word (or phrase)** – the start index of the word (or keyword or phrase) in

a line.

You must count white spaces and writing symbols. Starts from 0

- If the received keywords appear multiple times in the file, then you have to print them all.
  - For ①, ③**, print all locations** even if they exist in the same line.
  - For ②, ④, print once for a line
- **If more than one output for a single search, you must add a single space after a single keyword location. You must print a new line when a search is done. For example,**

```
$ "he is"
1955:33 2315:42 5885:22
(New Line)
```

- Further explanation about inputs including * or " :
  - The cases where * and " are included altogether will not be tested
  - For inputs that include ", refer to ③
  - For inputs that include *, refer to ④
  - For inputs that include neither * nor ", and include two or more words, refer to ②
  - For inputs that include neither * nor ", and include one word, refer to ①
- Search results should be output sequentially from the top of the file.
- Input text file name is given as command line argument (`argv[1]`)
- It is assumed that the input text file is larger than the system's memory, and the length of the longest string in the input text file is assumed to be smaller than the system's memory. That is, it is not possible to place all the input files into memory at once.
- Characters in input text files and keywords include only characters within the ASCII code range.
- There can be consecutive spaces in input text files.
- Input text files do not begin with a space.
- Keywords do not begin or end with spaces.

- **You may follow and refer to the details about keyword input at "6. Example" section.**
- The code executes infinitely waiting for input and terminates execution when the input keyword is "PA1EXIT".

3. Score Policy

- **It is based on a 100-point scale and must follow the format described above, otherwise no points will be awarded.**
- **Submissions after the deadline will be deducted 10 points per day. (After 10 days, 0 points for submission)**
- A certain amount of discussion is allowed, but the source code must be written by oneself.
- Scoring is carried out according to whether each test case passed or not, and the points assigned when the test case passed are included in the total score, and the points assigned when the test case was not passed are not included in the total score. There is no partial credit for each test case.
- Several input files are used for scoring, including the input file provided as an example, and the capacity of each input file can be up to several tens of GB.
- If your code takes more than 5 minutes to complete each test case, the score for that test case is not provided. All test cases are within 10 seconds of completion, based on the InUiYeJi cluster.
- The memory of the scoring server is assumed to be 1 GB.

4. Restriction

- This assignment is based on implementation in a **Linux environment.**
- Available header files are limited to the following files. If other headers are used, you get 0 points.

(1) At least one **hand-implemented** header file (required)

(2) `unistd.h fcntl.h stdlib.h sys/types.h sys/stat.h errno.h`

- The name of the binary executable file created through `Makefile` is set to **`pa1`** (lower case).

- Words: Strings separated by spaces, case insensitive.

- Examples of words that can be used in word search are as follows.

    `e.g.) god, and, adam, brother's, priests', kirjath-arba, sons', score:1`

- Phrase: It is a string separated by newlines and case insensitive.

- Blank space: Tab, space, or new line.

- Words containing writing symbols are considered different words. For example, a search for **`he`** should include only **`he`** in the search results, not **`her`** or **`he's`**.

5. Submit instructions

pa1 (directory)

- ***.c (C code files)**
- ***.h (Header files)**
- **Makefile**

Submit using the command:

```
$ ~swe2024-41_23s/bin/submit pa1 pa1
```

6. Example (Words highlighted in red are given as inputs)

```
$ ./pa1 500-Days-of-Summer_s.txt

500 days

6 6419

he is

50 1039 1822 1955 2256 2315 3494 3503 4353 4360 4445 4831 5101 5885
6325

"he is"

1955:33 2315:42 5885:22

he*is

2315 4445 5101

she he

1370 1512 1513 3423 3473 3478 3550 4255 4510 4515 5413 5672 6154
6188

loved

106:30 1122:9 1150:24 3961:25 4739:17 5921:28

…
```

```
$ ./pa1 500-Days-of-Summer_s.txt > result.out

he

hey

she

tom

summer is

"summer is"

landscape

together

we*her

no much

immediately

no*much

quarterback

we

PA1EXIT


$ diff –bsq result.out answer.out
Files result.out and answer.out are identical
$ diff result.out answer.out
$
```