Computer Science 352 Summer 2023
Programming Assignment 4
Due 6/30/2023 by 7pm

In PA three, you were supposed to write a simplified version of the **cut** command. This PA will require you to write a simplified version of the **grep** unix command. You should ensure that you are following all of the rules from the style guide, which can be found on the class website. For the project, you should end up with the following directory structure:

```
pa4
└── sgrep
    ├── sgrep.c
    └── makefile
```

The **`makefile`** should have two rules: one for **`sgrep`** and one for **`clean`**. The **`sgrep`** rule should compile **`sgrep.c`** with the required gcc flags and produce an executable named **`sgrep`** in the current working directory. The **`clean`** rule should delete the **`sgrep`** executable from the current working directory.

SGrep (Simple Grep)

You should write a C program named **`sgrep.c`**. This program will be a simplified version of the **grep** command that can be found on many UNIX systems, including lectura. You should have already used this command by now, perhaps for PA1. As with PA 3, I recommend spending a bit of time playing around with the command and reading the man page (**`$ man grep`**) for the regular command before starting this program. The **`grep`** command can be used for searching for strings and patterns from standard input. Your program should function similarly, but it does not have to support as many arguments and features compared to the regular command.

This program should expect between 0 and 3 flag arguments, and one additional argument which specifies what to search for. The program should support these three flag arguments in any position on the command line:

`-o` Only print the exact part(s) of each line that matches the search, not the entire line. The default behavior of **`sgrep`** will be to print the whole line if a match is found.

`-i` Use a case-insensitive search. The default behavior will be to search case-sensitive.

`-e` Search using a pattern specification. The default behavior will be to search for the exact search string provided.

The other command-line argument will be some sequence of ascii-characters that represent what to search for. The search term can either function in a literal search (search of the exact string the user provides) or, with the **`-e`** flag, the user can give a pattern (one search term that can match multiple strings). More information about these patterns will be shown later in the specification. The program should support the arguments being specified in any order.

The search will act on whatever is “fed” to the program via standard input, and should continue to search until EOF is reached. The search results should be printed to standard output. A few examples of valid ways to run this program (after being compiled to an executable file named **`sgrep`**):

`$ cat todo.txt | ./sgrep orange`
> Print out every line containing the string “**orange”** from **todo.txt**

`$ cat words.txt | ./sgrep -i -o ee`
> Print out every occurrence of the string “**ee**” “**Ee**” “**eE**” or “**EE**” from **`words.txt`**

`$ cat stuff.txt | ./sgrep -e b~z`
> Print out every line containing a subsequence that starts with “**b**” and ends with “**z**” from **`stuff.txt`**

`$ cat words.txt | ./sgrep -e a..z`
> Print out every line containing a subsequence that starts with “**a**”, ends with “**z**”, and contains two characters in-between from **`words.txt`**

The program will get its input from standard in. The input may have multiple lines, so the program should continue reading input until it receives a **EOF** line indicating that the program may stop processing input lines and may exit / return. This program may assume that the max line width is 200 characters wide, and it should not assume a maximum number of input lines.

The -e Option

If you use the -e option, this allows the user to specify a search pattern rather than just a search word. The search pattern supports regular ascii characters, but it treats two particular characters in a special way: **periods** (**`.`**) and **tilde** (**`~`**).

A **period** is a placeholder for any character. Thus, for example, the search term **`"c.t"`** would match words such as **`"cat"`**, **`"cut"`**, and **`"cot"`**.

A **tilde** matches one or more characters in that particular position. Thus, the search pattern **`"c~t"`** would match words such as **`"cat"`**, **`"cut"`**, and **`"cot"`** but also should match strings such as **`"crest"`** and **`"crop or not"`**. You can assume that a tilde will always have a non-tilde character immediately before it and after it within the search pattern. In order to help you understand valid search term formatting, shown below are several valid and invalid search pattern strings:

Valid	Invalid
`c~zen~o..abc`	`~abc.~`
`c……z~n`	`wonder.~.`
`..lend~save~to`	`~crazi..ness~`
`.ee.`	`~.~.`

If the program does not receive at least one command-line argument, it should print **`"Requires more command-line arguments.\n"`** to standard error and then exit with exit code 1.

If the -e option is enabled and the search term is not correctly formatted, the program should print **`"Invalid search term.\n"`** to standard error and then exit with exit code 2. In order to be correctly formatted, a search term should contain only upper and lower case letters, digits, periods, and tildes. Also, a tilde must have a letter immediately before and after it.

Examples

Lets walk through a few examples of how the program should behave, which will hopefully give you a better idea of how it should be implemented. Assume that you have written the program correctly, and that you created a makefile to compile it to an output executable file named **sgrep**. Run this to get your executable:

```
$ make
```

Let's also assume that you have a file in this same directory named **`words.txt`** and it has the following contents:

```
please excuse this fooh soups
specification
WONDERFUL day for fishing
zoos zoologists zoomies
one fish two FISH, red fish blue FISH
```

Lets try testing out our **sgrep** program. First, let's do a simple search for all the lines containing the word "fish":

```
$ cat words.txt | ./sgrep fish
WONDERFUL day for fishing
one fish two FISH, red fish blue FISH
$
```

Great! Two lines, as should be expected. Now, let's change the search to use the -o option to only include the exact parts of the lines that matches:

```
$ cat words.txt | ./sgrep -o fish
fish
fish
fish
$
```

Notice that there are three results. The first comes from the one appearance of the word "fishing" on line 2, and the other two come from the two appearances of the word "fish" on line 4. The two upper-case FISH were not included due to case-sensitivity. Lets add the case-insensitive option:

```
$ cat words.txt | ./sgrep -o fish -i
fish
fish
FISH
fish
FISH
$
```

Notice also that in all cases, the results should be printed out in the order that they appear in from standard input. Now, we decide we want to search for all four-letter words that begin with an f and end with an h. We can use this command which includes the -e argument:

```
$ cat words.txt | ./sgrep -i f..h -o -e
fooh
fish
fish
FISH
fish
FISH
$
```

Another example using the -e argument:

```
$ cat words.txt | ./sgrep -e -o s~i~ca...n
specification
$
```

Also, strep should not report overlapping matches. For example, this only prints out two lines to standard out:

```
$ printf "zaazbbzcccz\n" | ./sgrep -o -e "z~z"
zaaz
zcccz
$
```

But this prints three:

```
$ printf "zaazzbbzzcccz\n" | ./sgrep -o -e "z~z"
zaaz
zbbz
zcccz
```

Test Cases and Restrictions

For this PA, there will be a number of test-cases that test the various flags and combinations of the flags. Due to this, it will be possible to pass some of the test cases, even if you do not finish supporting all three of the options correctly. If you want full points you should support all of the features described in the spec. However, say you finish the general search code, case-insensitivity, and the -o option, but run out of time to support -e. In this case, you should still submit what you have and try to pass at least a few of the test cases.

For this assignment, you may only use a limited selection of functions from the C standard library. You may only use these and no others:

`printf  fprintf  strcpy  strlen  strcmp  strncmp  fgets  scanf`

Submitting Your PA

After you have completed this PA, double check that the file structure and file / directory names match what is shown in the project overview on the first page, ensure that your code compiles and runs correctly on lectura, and that the code follows the rules from the style guide. You should not include any files in the submission other than the ones shown on the first page of this spec. If you are on a mac, you should avoid zipping the file on the mac, because it might include one or more hidden / extra files. Instead, zip on lectura.

Once you are ready to submit, zip up the **`pa4`** directory by running:

```
$ zip -r pa4.zip ./pa4
```

Then, turn this file to the PA 4 dropbox on gradescope. There will be some test cases that will not be visible until after the grades get published. You should ensure you pass the visible ones, and also test your code thoroughly.