CSE344 Homework1 Report

Yeşim Yalçın-200104004094

1-) Problem Definition and My Solution Approach

The problem was to develop a simple text editor which has the abilities to replace a specific string with another specific string. The specified string could be anywhere in the text meaning it did not have to be an exact word. Moreover there could be additional details like matching strings only at the line starts, only at the line ends etc.

My approach was to firstly eliminate some possible exceptions that could occur because of invalid command line inputs. The user needs to enter 2 command line arguments for the program to function.
The first argument must include the specified strings and the additional details related to them. They also needed to be inputed in a specific way. (Ex: “/” placements) I have firstly checked if this input was entered in a valid way. If it did not, I have printed out the problem onto stderr and then exited the program.
The second argument must be a txt file. I have checked if the argument string had .txt format. If it did not, I have added .txt at the end of the string. However, I did not check if it had any other formats. In that case, the program will give an error during file opening process.

After I knew that I had 2 valid arguments, I have started to break the first argument into parts. I collected all details related to 2 specified strings. I have created a custom struct “stringPairs” to store the data. This struct stores the strings, if they contain any repeatable characters, where they contain. If they are case insensitive, if they need to match at the line start etc I collected all. I thought if I collected all the data at the start, the string matching can be done a lot more easily on the file.

After I had all the details stored in an organized way, I have opened the input file also a temporary file. If any of the opening fails, the problem is printed onto stderr and it exits. I also have locked the input file in case the program is run multiple times at the same time. This lock controls if there’s any locks on the file and if there is it exits the program letting the process that actually locked the file run first.

My aim was to read from the input file, read word by word, do the string matching and replacements on that word, write the word onto the temporary file. If there is more than one string pairs stored to process, the first one has priority over the second one. When all the words are processed, I truncated the input file then copied everything inside the temporary file into the input file. Then unlinked the temporary file. At the end closed all the opened files and remove the locks.

I have done all the operations according to my aim considering other possible read, write system call errors.

I have not used any libraries other than “string.h” during all my string matching & replacing process. “string.h” library was allowed. All the other libraries I used were either necessary for the system calls or they were needed for simple things like using bool type or using printf.

2-) System Requirements

As I have mentioned before, the program needs 2 properly inputed command line arguments.
The first one:
-Must start with “/” and end with “/” or “/i”.
-Must have a “/” to separate the strings in a pair.
-Must have exactly three “/” for each string pairs.
-String pairs must be separated with “/i;/” or “/;/”.
-The strings cannot include “^”, “$”, “[“, “]”, “/”, “*”, outside of their purposes.
-A string cannot have both “^” and “$”.
-If there is an empty string, it will not give an error but nothing will happen.
-“^”, “*”, “$” cannot be in an empty string.
-“[]” must include at least two characters inside.
-A string cannot start with “*”.

The second one:
-Should be a txt file.
-Does not need to include .txt at the end, the program automaticaly assumes it is a txt even if it cannot find .txt at the end of the string.
-Can reside in the same working directory, if the file name is enough.
-If not in the same working director inputs like: “~/Documents/input.txt” is also fine.
All these rules are checked in the program.

Extra requirements that are not checked in the program:
- The first argument should be inputed inside single quotes in a UNIX environment as it was shown in the homework PDF.
-“[]” cannot include more than 94 characters inside. Because 94 is the number of characters there are than can be inputed with a keyboard according to the ASCII table.
-A string that is wanted to be replaced with, or that is wanted to be replaced as cannot be longer than 1000 characters.

You can run the program with ./output command after compiling it using the makefile.

3-) The Parts I Have Completed

I have completed all the requirements of the homework and tested in various ways.

4-) Tests

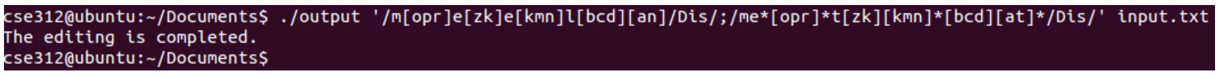
I have provided an example input.txt file that can be run with the following commands:

1-) '/me*sel*a/Dis/;/me*sel*a*/Dis/'
2-) '/me*sel*a/Dis/i;/me*sel*a*/Dis/i'
3-) '/m[opr]e[zk]e[kmn]l[bcd][an]/Dis/;/me*[opr]*t[zk][kmn]*[bcd][at]*/Dis/'
4-) '/m[opr]e[zk]e[kmn]l[bcd][an]/Dis/i;/me*[opr]*t[zk][kmn]*[bcd][at]*/Dis/i'
5-) '/[abc]ms[kt][mn][op]/Dis/;/defne/Dis/'
6-) '/[abc]ms[kt][mn][op]/Dis/i;/defne/Dis/i'
7-) '/t*o*rmn*/Dis/;/[mo]*n*trs*k*/Dis/'
8-) '/t*o*rmn*/Dis/i;/[mo]*n*trs*k*/Dis/i'

The input.txt file is sectioned with names like “Birinci” or “Üçüncü + Dördüncü”. These mean which commands especially effect them. Some of the words have only 1 string match, some have multiple. Some have half matches hidden inside to show the program ignores everything aside from exact matches. Also all lines have 3 words. So if “^” or “$” are wanted to be added, their effects can be observed easily.
I have prepared the commands and test words in a way to show all possible scenarios therefore the words are meaningless.

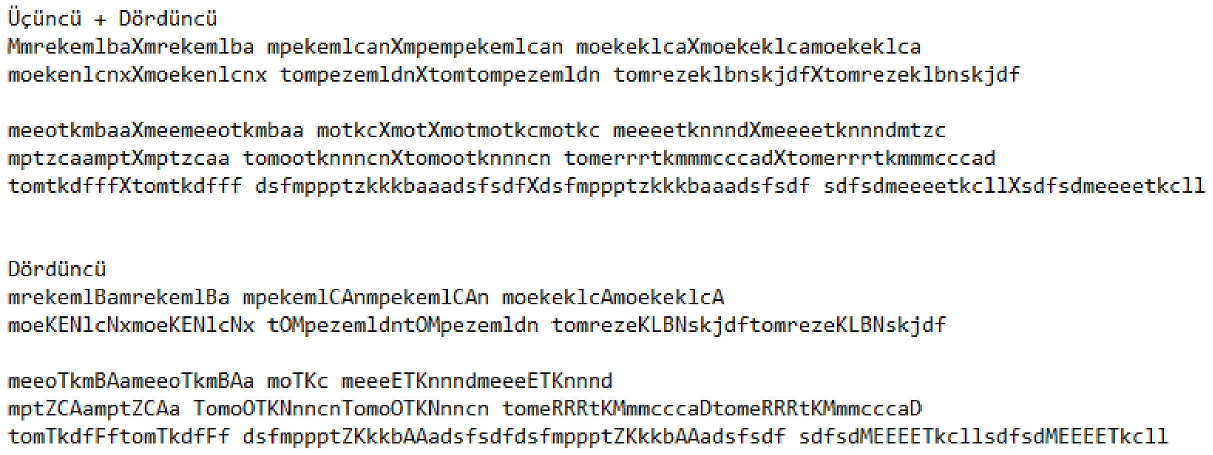
I will give some examples from the tests:

3-) '/m[opr]e[zk]e[kmn]l[bcd][an]/Dis/;/me*[opr]*t[zk][kmn]*[bcd][at]*/Dis/'



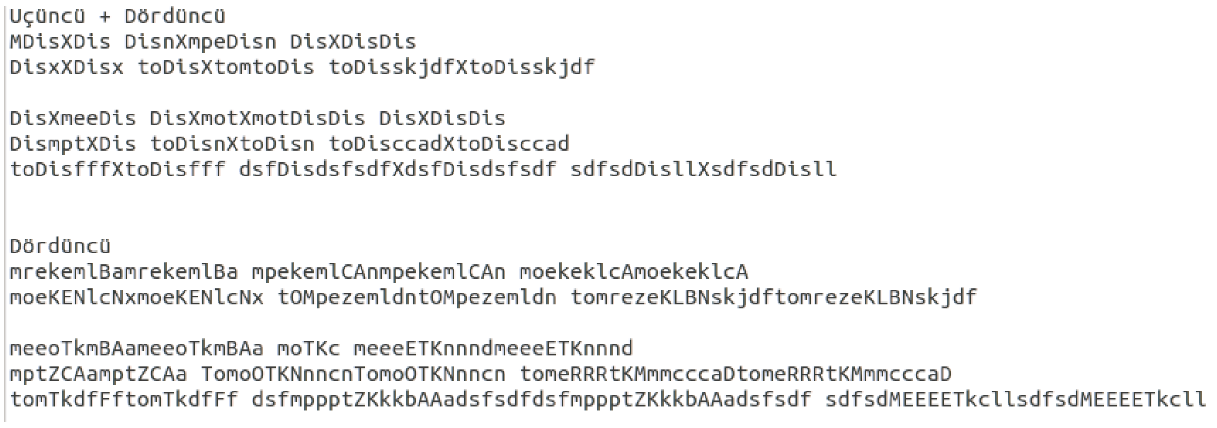
```
cse312@ubuntu:~/Documents$ ./output '/m[opr]e[zk]e[kmn]l[bcd][an]/Dis/;/me*[opr]*t[zk][kmn]*[bcd][at]*/Dis/' input.txt
The editing is completed.
cse312@ubuntu:~/Documents$
```

Before:



```
Üçüncü + Dördüncü
MmrekemlbaXmrekemlba mpekemlcanXmpempekemlcan moekeklcaXmoekeklcamoekeklca
moekenlcnxXmoekenlcnx tompezemldnXtomtompezemldn tomrezeklbnskjdfXtomrezeklbnskjdf

meeotkmbaaXmeemeeotkmbaa motkcXmotXmotmotkcmotkc meeeetknnndXmeeeetknnndmtzc
mptzcaamptXmptzcaa tomootknnncnXtomootknnncn tomerrrtkmmmcccadXtomerrrtkmmmcccad
tomtkdfffXtomtkdfff dsfmppptzkkkbaaadsfsdfXdsfmppptzkkkbaaadsfsdf sdfsdmeeeetkcllXsdfsdmeeeetkcll


Dördüncü
mrekemlBamrekemlBa mpekemlCAnmpekemlCAn moekeklcAmoekeklcA
moeKENlcNxmoeKENlcNx tOMpezemldntOMpezemldn tomrezeKLBNskjdftomrezeKLBNskjdf

meeoTkmBAameeoTkmBAa moTKc meeeETKnnndmeeeETKnnnd
mptZCAamptZCAa TomoOTKNnncnTomoOTKNnncn tomeRRRtKMmmcccaDtomeRRRtKMmmcccaD
tomTkdfFftomTkdfFf dsfmppptZKkkbAAadsfsdfdsfmppptZKkkbAAadsfsdf sdfsdMEEEETkcllsdfsdMEEEETkcll
```

After:



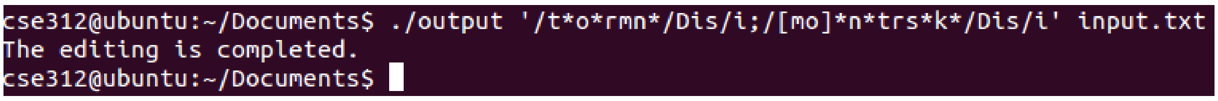
```
Uçüncü + Dördüncü
MDisXDis DisnXmpeDisn DisXDisDis
DisxXDisx toDisXtomtoDis toDisskjdfXtoDisskjdf

DisXmeeDis DisXmotXmotDisDis DisXDisDis
DismptXDis toDisnXtoDisn toDisccadXtoDisccad
toDisfffXtoDisfff dsfDisdsfsdfXdsfDisdsfsdf sdfsdDisllXsdfsdDisll


Dördüncü
mrekemlBamrekemlBa mpekemlCAnmpekemlCAn moekeklcAmoekeklcA
moeKENlcNxmoeKENlcNx tOMpezemldntOMpezemldn tomrezeKLBNskjdftomrezeKLBNskjdf

meeoTkmBAameeoTkmBAa moTKc meeeETKnnndmeeeETKnnnd
mptZCAamptZCAa TomoOTKNnncnTomoOTKNnncn tomeRRRtKMmmcccaDtomeRRRtKMmmcccaD
tomTkdfFftomTkdfFf dsfmppptZKkkbAAadsfsdfdsfmppptZKkkbAAadsfsdf sdfsdMEEEETkcllsdfsdMEEEETkcll
```

“Üçüncü + Dördüncü” section is the same as “Dördüncü” section. Only difference is “Dördüncü” section needs case insensivity. Because the command does not include case insensivity, “Dördüncü” section is left the same.

Here if we examine some of the words:
1-) **MmrekemlbaXmrekemlba**: “**mrekemlba**” matches with **m[opr]e[zk]e[kmn]l[bcd][an]** therefore both of them are turned to “Dis”. However non matching parts: M at the start and X in the middle is left the same. Result: **MDisXDis**

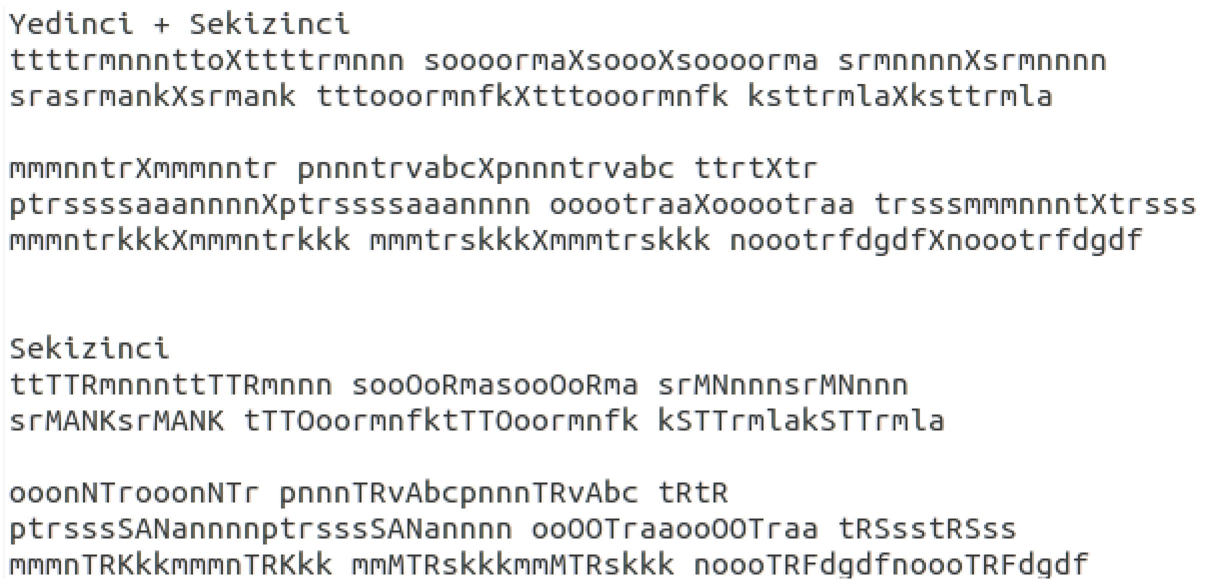
2-) **meeotkmbaaXmeemeeotkmbaa**: “**meeotkmbaa**” matches with **me*[opr]*t[zk][kmn]*[bcd][at]*** therefore both of them are turned into “Dis”. However, non matching parts: Xmee is left the same. Result: **DisXmeeDis**

8-) '/t*o*rmn*/Dis/i;/[mo]*n*trs*k*/Dis/i'



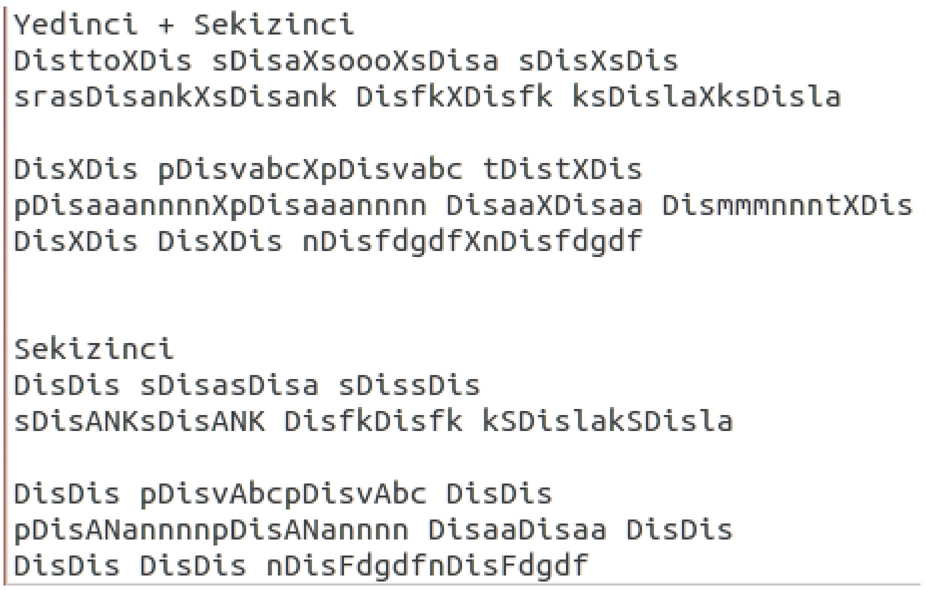
```
cse312@ubuntu:~/Documents$ ./output '/t*o*rmn*/Dis/i;/[mo]*n*trs*k*/Dis/i' input.txt
The editing is completed.
cse312@ubuntu:~/Documents$ 
```

Before:



```
Yedinci + Sekizinci
ttttrmnnnttoXttttrmnnn soooormaXsoooXsoooorma srmnnnnXsrmnnnn
srasrmankXsrmank tttooormnfkXtttooormnfk ksttrmlaXksttrmla

mmmnntrXmmmnntr pnnntrvabcXpnnntrvabc ttrtXtr
ptrssssaaannnnXptrssssaaannnn oooотraaXoooотraa trsssmmmnnntXtrsss
mmmntrkkkXmmmntrkkk mmmtrskkkXmmmtrskkk noootrfdgdfXnoootrfdgdf


Sekizinci
ttTTRmnnnttTTRmnnn sooOoRmasooOoRma srMNnnnsrMNnnn
srMANKsrMANK tTTOoormnfktTTOoormnfk kSTTrmlakSTTrmla

ooonNTrooonNTr pnnnTRvAbcpnnnTRvAbc tRtR
ptrsssSANannnnptrsssSANannnn ooOOTraaooOOTraa tRSsstRSss
mmmnTRKkkmmmnTRKkk mmMTRskkkmmMTRskkk noooTRFdgdfnoooTRFdgdf
```

After:



```
Yedinci + Sekizinci
DisttoXDis sDisaXsoooXsDisa sDisXsDis
srasDisankXsDisank DisfkXDisfk ksDislaXksDisla

DisXDis pDisvabcXpDisvabc tDistXDis
pDisaaannnnXpDisaaannnn DisaaXDisaa DismmmnnntXDis
DisXDis DisXDis nDisfdgdfXnDisfdgdf


Sekizinci
DisDis sDisasDisa sDissDis
sDisANKsDisANK DisfkDisfk kSDislakSDisla

DisDis pDisvAbcpDisvAbc DisDis
pDisANannnnpDisANannnn DisaaDisaa DisDis
DisDis DisDis nDisFdgdfnDisFdgdf
```

Because the command include case insensivity, this time both “Sekizinci” and “Yedinci + Sekizinci” sections are changed.

Here if we examine some of the words:
1-) **soooormaXsoooXsoooorma**: “**oooorm**” matches with “**t*o*rmn***” therefore both of them are turned to “Dis”. However, non matching parts are left the same. Result: **sDisaXsooXsDisa**

2-) **ooonNTrooonNTr**: “**ooonNTr**” matches with “**[mo]*n*tr*k***” (with case insensivity) therefore both of them are turned to “Dis”. Result: **DisDis**

I have also added one sample valgrind report with the use of one of the commands I have mentioned.