Running the code:

In order to run the code, the program must be in same directory as the users.

In order to compile, the following code must be written in terminal:

```
$ gcc problem2.C -o file -lm
```

In order to run and use the file handling functions, the user has to use the following commands (example):

```
./file fcreate firstfile.txt
```

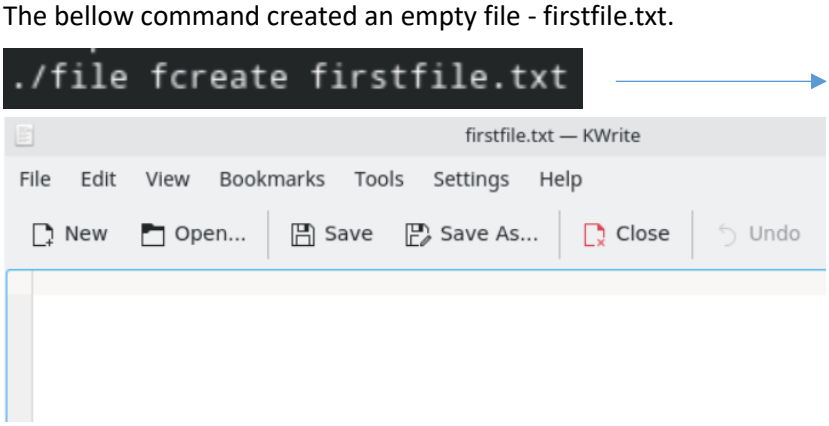
The above command is used to create (using fcreate) textfile firstfile.txt. Every command must start with **./file.**

File Operations:

fcreate <filename.txt> - used to create a file with name *filename.* This is one of the simplest built command.

The bellow command created an empty file - firstfile.txt.

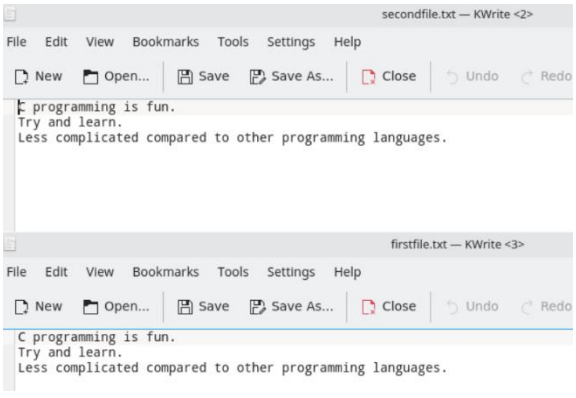
```
./file fcreate firstfile.txt
```



fcopy <fromfile.txt> <tofile.txt> - copy the contents of the file *fromfile.txt* to *tofile.txt.*

```
./file fcopy firstfile.txt secondfile.txt
```

So the command to the left copies the contents of the firstfile.txt to secondfile.txt. It is not necessary to write separate command to create secondfile.txt.



The second file is created automatically when we wrote:

fp = fopen (secondfile, “w”);

If there is no such file, mode “w” allows to automatically create and copy the contents of the firstfile.txt.

fdelete <filename.txt> - delete a file with name *filename.*

```
./file fdelete secondfile.txt
```

The command is made possible with ***remove()*** function. If it is successful then it gives a message. Otherwise it throws an error.

fdisplay <filename.txt> - display the contents of an existing file with a name *filename.txt.*

```
[u2110522@emu-04 CS132]$ ./file fdisplay firstfile.txt
C programming is fun.
Try and learn.
Less complicated compared to other programming languages.
```

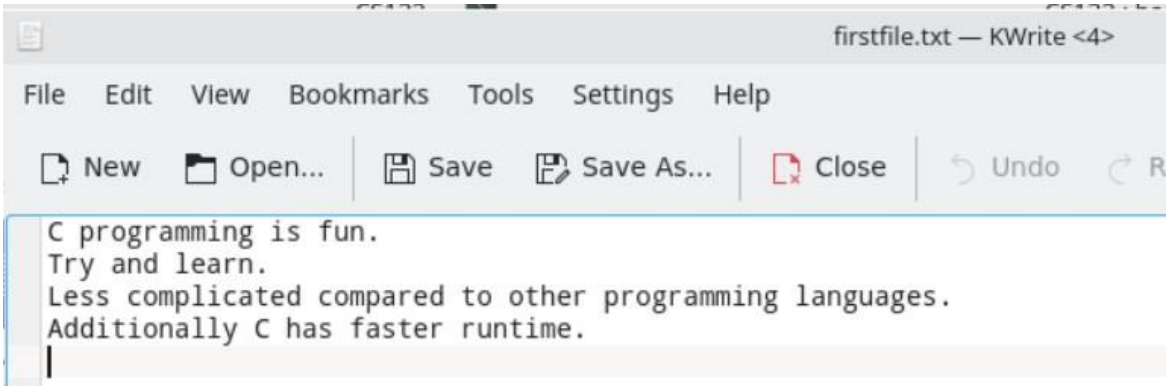
.

Line Operations

fappendline <filename.txt> – create a new line of *content* at the end of file with name *filename.txt.*

Ask for the content after getting specified the name of the file.

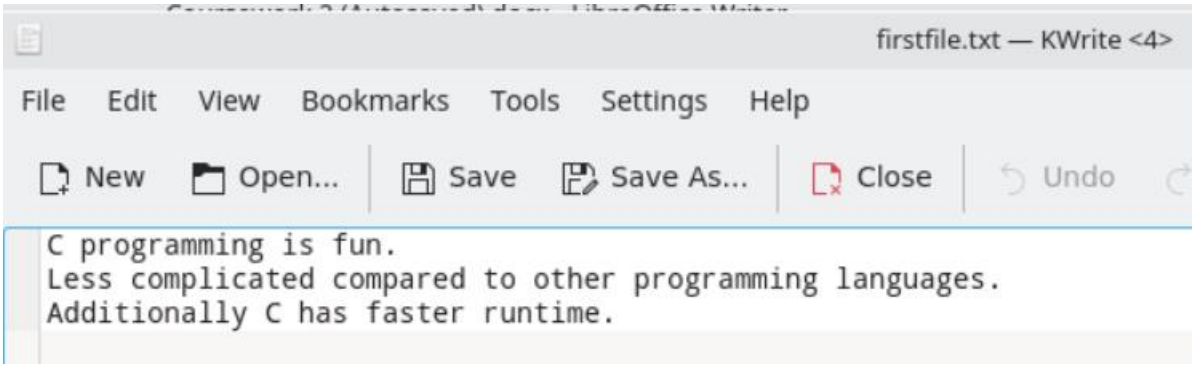
```
[u2110522@emu-04 CS132]$ ./file fappendline firstfile.txt
Please, type your line to be appended:
Additionally C has faster runtime.
```



In order to write we used: fp = fopen(firstfile, “a”); (in ‘append’ mode).

fdeleteline <filename.txt> {line number} – delete a line of content at a particular *line number* in a file with name *filename.txt.*

```
[u2110522@emu-04 CS132]$ ./file fdeleteline firstfile.txt 2
The line 2 has been successfuly deleted.
```

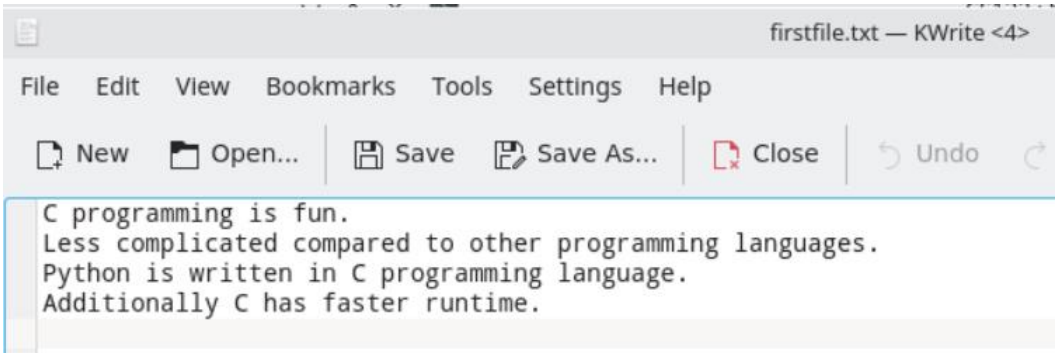


To implement this functionality, I first copied the contents of the firstfile.txt, except for the content in the 2nd line, to the temporarily created textfile. Then the contents of the temporary file is copied back to the firstfile.txt. The temporary file has been deleted.

finsertline <filename.txt> {line number} – create a new line of content at a particular *line number* in a file with name *filename.txt.*

Ask for the content after getting specified the name of the file.

```
[u2110522@emu-04 CS132]$ ./file finsertline firstfile.txt 3
Python is written in C programming language.
```

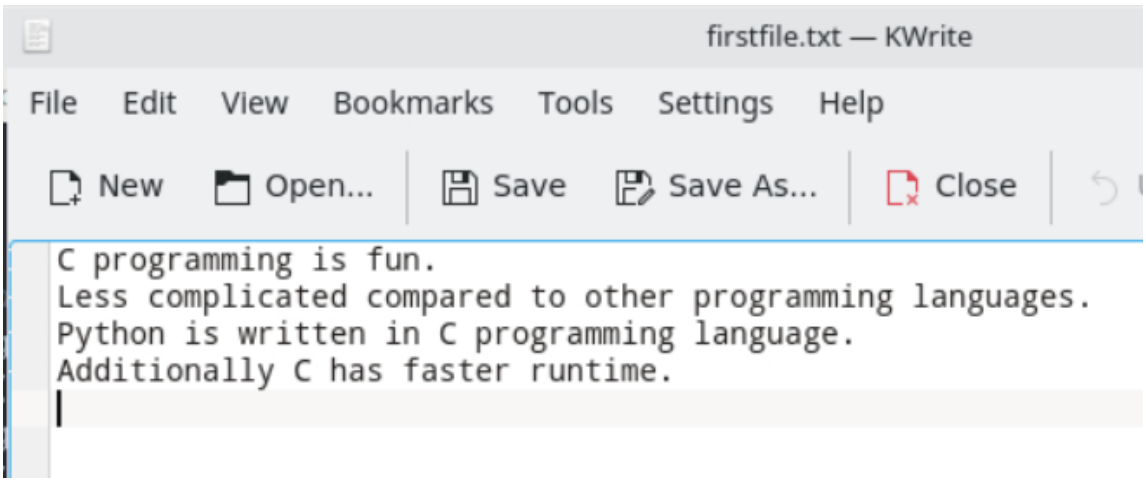


In order to implement this functionality, I implemented a technique similar to the previous ones. I have created temporary file and copied the contents of the original file to the temporary one including the new line input by the user. Then I copied back the contents in temporary file to the original firstfile.txt file.

fdisplayline <filename.txt> {line number} – display the contents of a file at a particular *line number* in a file with name *filename.txt.*

```
[u2110522@emu-04 CS132]$ ./file fdisplayline firstfile.txt 4
Additionally C has faster runtime.
```

As it can be seen above, the 4th line of the text file firstfile.txt was called and the result was the same as the 4th line in the picture below.



In order to implement this command, I just traced the lines that algorithm points. Once I reached the specified line, I used printf() method to display it.

nolines <filename.txt> – show the number of lines in a file with name *filename.txt*.

```
[u2110522@emu-04 CS132]$ ./file nolines firstfile.txt
There are 4 lines.
```

```
if (pointer == '\n') {
  pointer = fgetc(fp);
  // if the next line does not contain ' ' and EO
  if (pointer != ' ' && pointer != EOF) lines ++;
}
```

The number of lines of text is found by counting the number of '\n' lines. In some cases this might not be the case.

```
if (pointer == '\n') {
  pointer = fgetc(fp);
  // if the next line does not contain ' ' and EOF,
  if (pointer != ' ' && pointer != EOF) lines ++;
}
```

Therefore I included one more condition into my code:

changelog - Display the sequence of operations performed on all files created by your program, including the number of lines following each operation.

```
[u2110522@emu-04 CS132]$ ./file changelog
./file fdisplay
./file fclear
./file fdisplay
./file fdisplay testtext.txt
./file fcopy firstfile.txt
./file fdisplay firstfile.txt
./file fappendline firstfile.txt
./file fdeleteline firstfile.txt
./file finsertline firstfile.txt
./file fdisplayline firstfile.txt
./file nolines firstfile.txt
[u2110522@emu-04 CS132]$
```

Before doing any operation, I decided to create text file in which all the change log operations on any text files are stored.

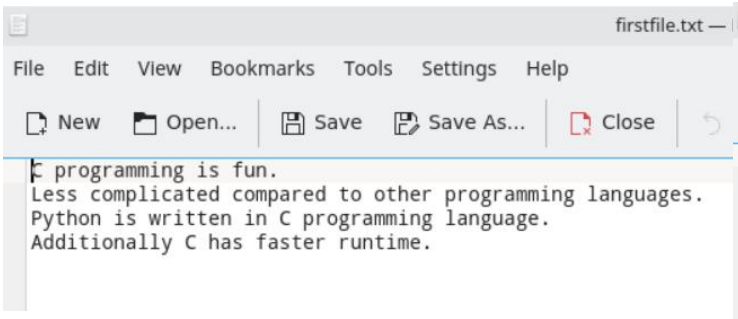
After doing the operations on text files, these operations are put into change log file.

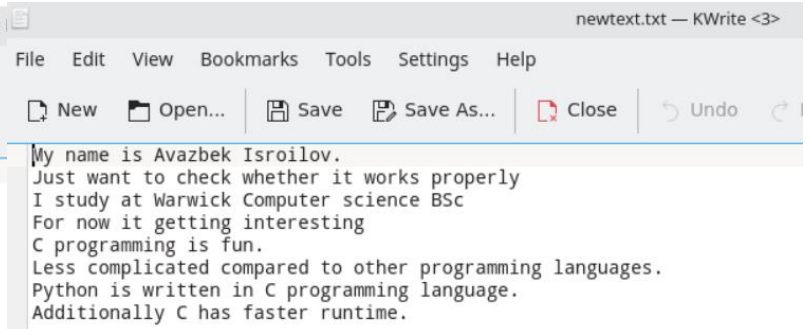
And when requested they are displayed in console.

fconcat <fromfile.txt> <tofile.txt> - concatenates the contents of the fromfile.txt to tofile.txt.

From my research concatenating the contents of files (such as datasets and etc.) has a lot applications in collecting data used for training ML models.

```
./file fconcat firstfile.txt newtext.txt
```





fclear <textfile.txt> - clears the contents of the file.

When I was testing my code in this coursework, I had to clean many of the files to continue with my testing. Even though it is still possible to implement fclear using other basic operator first fdelete, followed by fcreate, fclear is much more convenient.

```
./file fclear firstfile.txt
```