Yeditepe University
Department of Computer Engineering

CSE 232 Systems Programming
Spring 2020

Term Project

Text Editor

Due to: 15 May 2020

4 Students in a Group

In this project, you will develop a simple **text editor with version control**.

Write your text editor in C and use gcc compiler on Linux.

The editor has two parts. In Part 1 you will implement the editing functions. In Part 2 you will implement the version control.

Part 1

The editor must perform the following operations:

- E (edit): Opens the specified file and reads it into the text buffer
- I (insert): Inserts the line in text buffer
- D (delete): Deletes the line
- P (print): Displays the text on the screen
- S (save): Saves the file
- C (commit): to be explained in Part 2

The editor must keep the text in a text buffer using the following data structure:

```
struct node
{
      int   statno;      // statement number
      char  statement[40];  // max. 40 characters
      int   next;        // points to the textbuffer[] index of the next statement
}
struct node textbuffer[30]; // max. 30 lines
int head;   // points to the first valid statement in textbuffer[]
```

`textbuffer[30]` and `head` are global.

Write the following functions:

`edit(*char filename)`
Opens the specified file (with .txt extension), reads the text and stores it in `textbuffer[]`.

`insert(int statno, *char stat)`
Reads statement number and the statement from keyboard, stores it at the end of `textbuffer[]` and updates the links.

`delete(int statno)`
Reads statement number from keyboard and updates the links.

`print()`
Starting from the head of the `textbuffer[]` follow the links and display the text on the screen.

`save()`
Starting from the head of the `textbuffer[]` follow the links, write the text in the .txt file and close the file.

Reads statement number and the statement from keyboard, stores it at the end of the `textbuffer[]` and updates the links.

Inplement the following algorithm:

```
call edit()
read input from keyboard
while (input is not X) {
      if input is I then call insert()
      if input is D then call delete()
      if input is P then call print()
      if input is S then call save()
      if input is E then call edit()
      read input from keyboard
}
exit
```

Part 2

In Part 2, you will make the necessary changes in the functions that you wrote in Part 1, in order to implement a simple version control system. Your editor must store, only the modifications made in each version, not the complete text. For this purpose, in addition to the .txt file, you will create a .dif file with the same file name. The modifications associated with each version must be stored in the .dif file. Initially .dif file contains 0 (i.e. version 0).

Make the following changes in the functions that you wrote in Part 1:

In `edit()` function, read the text from .txt file into the `textbuffer[]`.
If the user does not give a version number (e.g. "E mytext.txt"), latest version will be edited. In this case, all modifications that have been written in the .dif file will be made on the text in the `textbuffer[]`.
If the user gives a version number (e.g. "E mytext.txt 3" for version 3), editing must start from that version. In this

case, all modifications up to that version must be made in the `textbuffer[]`.

Use the following data structure to store the changes:

```
struct dfs
{
      int   code;       // 1-insertion 2-deletion 0-otherwise
      int   statno;
      char  statement[40];  // only for insertion
}
struct dfs diffs[20]; // max. 20 changes
int version;      // version number
```

`diffs[20]` array and `version` are also global.

In `insert()` function, insert the line in `textbuffer[]` as in Part 1 and write the modification also in `diffs[]` array with code 1.

In `delete()` function, delete the line from `textbuffer[]` as in Part 1 and write the modification also in `diffs[]` array with code 2.

In `save()` function, write the current version number, and then the modifications in `diffs[]` array to the end of the .dif file. Write -1 to mark the end of the modifications. In Part 2, `save()` function will not change .txt file, it will only add the modifications to the end of .dif file.

Also write `commit()` function, that saves all changes permanently. In `commit()` function, starting from the head of the `textbuffer[]` follow the links, and write the text in the .txt file, and close it. Also clear .dif file and write 0 in it (return to version 0). `commit()` function is called when the user enters C. After commit, older version cannot be accessed.

Example:

mytext.txt	mytext.dif
10 aaa	0
20 bbb	
30 ccc	

Editor commands:
E mytext.txt
I
15 ddd
D
20
S

mytext.txt	mytext.dif
10 aaa	1
20 bbb	1 15 ddd

30 ccc	2 20
	-1

Editor commands:
E mytext.txt
D
10
I
25 eee
S

mytext.txt	mytext.dif
10 aaa	1
20 bbb	1 15 ddd
30 ccc	2 20
	-1
	2
	2 10
	1 25 eee
	-1

Editor commands:
E mytext.txt
I
22 fff
C

mytext.txt	mytext.dif
15 ddd	0
22 fff	
25 eee	
30 ccc	