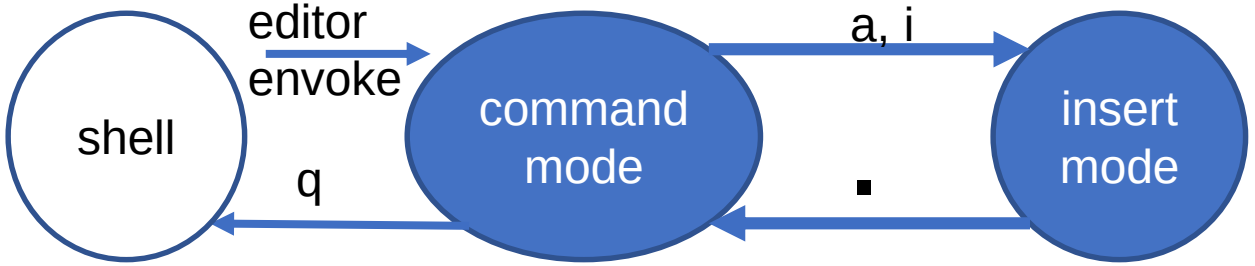
HW #1 Line Editor

Overview of the Mission

- Implement the line editor in C Programming Language
- by **using the dynamic memory allocation** for storing file contents of lines of file under edit
- Use **char ** buffer** global variable for this memory management

- Line editor has two modes



- Line editor has line by line text editing function
- It has a buffer to store the file contents
- Use array of **function pointer** to implement the user interface

Line Editor Function

1. Starting

   $ edit <file_name> [options]

   - At the beginning of program, the internal buffer is initialized to read file contents. If the file does not exist, create it as empty file. And, current line pointer is set to line #1.
   - options: -p <char> : change the command mode prompt as <char>

2. edit modes (2 modes):
   - command mode: [address] command [arguments]
   - insert mode : enterting contents by typing

3. mode change
   - changing from command mode to insert mode: enter a (append after current line) or i(insert before current line) command
   - changing from insert mode to command mode: entering "." at the beginning of a line

4. prompt

   '*' (default prompt) indicates the editor is in command mode

   $editor file.txt –p # : change the prompt as '#'

Commands (1)

- **q** : exit from edit program
- **s** : show file status (total_line_number and default_file_name)
- **h** : help command to display the commands available to users
- **<line#> i** :
  - It enter 'insert' mode (valid *<line#>* ranges ***1 ~ <last_line>+1)***
  - and insert new line between <line#> and <line#>+1
  - In the 'insert' mode, you can enter unlimited new contents and the *insert mode is closed (program mode is changed to command mode) by entering '.' at the beginning of new line*
  - If the *<line#>* is given as ***'<last_line#>+1'*** it works as concatenating the new input contents to current file contents (It works as if new contents is appended to the end of current existing file)

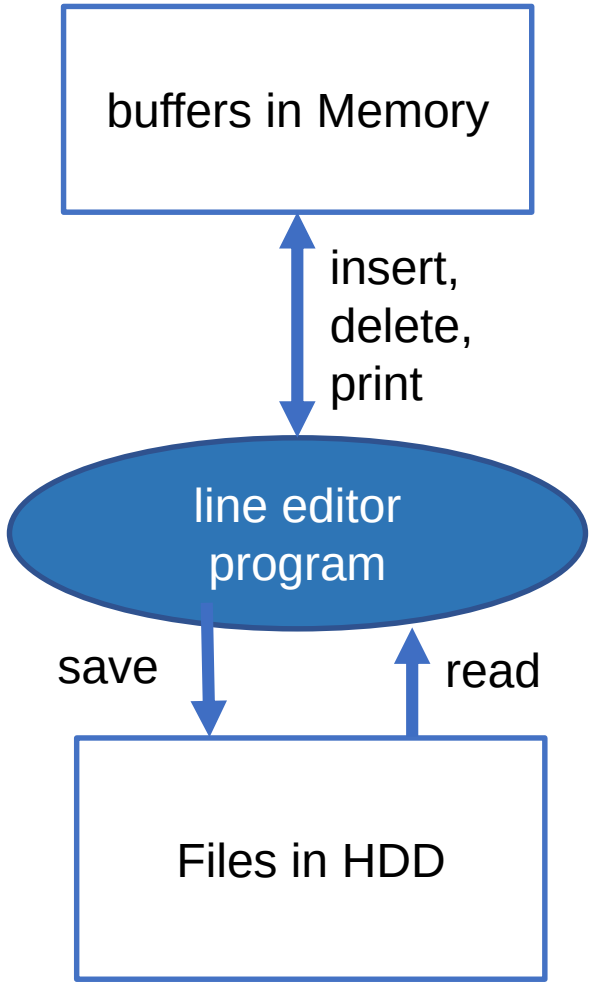
Commands (2)

- [*<address>*] **w** *[<file_name>]* : write (save) contents to a file
  - <default_filename> is set as <file_name>
  - if <file_name> is omitted, <default filename> is applied
  - if <address> is omitted, the address range is 1,last_line

- <address> p : print contents of buffer to screen
- <address> d: delete contents from buffer
- <address> format: <begin_line#>,<end_line#>
  - <begin_line#> can be line number, 1 ~ last_line
  - <end_line#> can be line number, 1 ~ last_line

Example ($: shell prompt, * : editor prompt)

```
$ edit file1.txt
* 1 i
This is a sample file.
The . ends the input and returns to command.
.
* s
total_line: 2, default_file: file1.txt
* 1,2 p
This is a sample file.
The . ends the input and returns to command.
* w
* q
$
```



Code Skeleton

```
int print (char **buf) ;
int del (char **buf);
int save (char **buf);
int change (char **buf);
int insert (char **buf);
int quit (char **buf);
FILE * process_cmd_args (int, char **);
char ** read_init_buffer (FILE *);
CMD get_cmd(char);
typedef enum { PRINT=0, DEL,
    SAVE, CHANGE, INSERT, QUIT } CMD;
```

```
typedef enum cmd {PRNT, DEL, SAV, STAT, INS, QUIT} CMD;
main(int argc, char *argv[])
{
  int (*exec_cmd[])(char **)  = {print, del, save, status, insert, quit};
  char prompt = '*';

  // open file and change prompt
  FILE * fp = process_cmd_args(argc, argv, & prompt);
  char ** buffer = read_init_buffer(fp);
  int ret = 0;
  while(ret >= 0) {// read command and execute
      CMD cmd = get_cmd (prompt);
      ret = (exec_cmd[cmd])(buffer);
      if(ret > 0)
         printf("%s₩n", msg[ret]);
   }
}
```

Bonus: Implement the additional options

1. If you try quiting edit without saving the buffer after modifying it, warning message generated and reconfirm user before exiting the program.
2. -p <new_prompt> command line option
   : change the editor command mode prompt
3. r <file_name> command of editor
   : read <file_name> and insert it after the current line