Simple VI Editor

1. 정의 및 기본 개념 및 기능

1.1 파일입출력: VI Editor를 제작하기 위해서는, 우선적으로 파일을 불러오거나 파일이존재하지 않을 때 생성하는 것이 가능 해야 한다. [[1]](#endnote-1)즉 c언어에서 파일 입출력의 문법에 대한 지식이 반드시 필요하다. 주어진 조건에 따르면, 파일은 반드시 편집이 가능하게 불러와져야 하지만, 기존 vi에서는 RO(read only)모드로 열 는 것이 가능하다. 이는 쉘에 argv[2]를 더 입력 받아서 이러한 기능을 추가로 구현할 수도 있다는 것을 나타낸다.

1.2 메모리 할당: 주어진 조건을 충족하기 위해서는 메모리 할당을 하는 법에대한 지식이 필요하다. 그러기위해서는 malloc함수를 이용해야하며, malloc함수를 사용하기 위한 라이브러리가 무엇인지 알고있어야한다.

1.3 파일 읽고 메모리에 내용복사: 일단 파일을 읽기위해서 필요한 함수에대한 지식이 필요하다. 또한 파일의 끝을 판단할 수 있는 방법에 대한 학습이 필요하다.[[2]](#endnote-2)

1.4 draw함수 : 이 함수는 기본적으로 주어진 함수이다. Copy를 받은 영역(lines)을 참조하여 24\*80 만큼의 lines의 character를 출력한다. 이 조건은 반드시 24\*80이라는 편집 박스를 지켜야 하므로, 이를 넘어가는 파일을 대상으로 편집하는 것은 불가능하다는 것을 알 수 있다. 또한 탭과 같은 문자는 하나의 character이지만 세칸을 출력하기 때문에, 제작 과정 중에 탭에 대한 처리가 필요하다.

1.5 실행 메커니즘 : 일단 에디터의 기본적인 실행메커니즘 중에 가장 중요한 것은 종료조건이다. 종료조건은 ex\_command\_mode에서 ‘q!’ or ‘wq’ or ‘q’(조건부 종료)라는 커맨드를 입력하는 것이다. 일단 이를 위해서는 commandmode에서 excommandmode로의 전환이 가능해야 하며, 또한 excommandmode editormode commandmode 각각 전환이 자유로워 하는 것 이다. 그렇기 때문에 mod라는 개념이 필요하다. 그렇기 때문에 excommandmode commandmode editormode를 각각 함수화하여, 리턴값에 따라서 함수(모드)를 전환할수있다. 또한 리턴값에 따라서 종료와 저장과 같은 기능을 실행할수있다.

1.6 Command mode 기능: 커맨드모드는 편집보다는 커서의 이동과 search와 같은 기능에 특화되어야한다.

일단 커맨드모드에서는 커서를 오른쪽,왼쪽,아래,위로 움직일 수 있어야하며 이를 위해서는 커맨드에 따라 적절히 커서의 좌표를 변경할수있어야한다.

커맨드모드에서 커서의 이동을 위해 구현해야 할 사항은 다음과 같다.

‘h’ command : 커서가 왼쪽으로 이동한다.

‘j’ command : 커서가 아래로 이동한다.

‘k’ command : 커서가 위로 이동한다.

‘l’ command : 커서가 오른쪽으로 이동한다.

또한 이에 대한 제약사항이 존재한다. 24\*80이라는 박스에서만 커서가 움직일수있다는것이며, 입력되지 않은 부분으로의 커서의 이동은 불가능하다는 것이다.

그래서 VIM의 경우는 ‘h’ ‘l’의 경우 이동하고자하는 위치가 만일 입력되지 않은 부분이거나, 범위를 초과하는 경우는 이동할수없도록 설계되어있다. 또한 ‘j’와 ‘k’는 만일 이동하고자 하는 위치가 입력되지 않은 부분 일 경우 그 라인의 마지막 입력부분까지 이동한다.

커맨드모드는 커서의 이동 외에도 다음과같은 기능이 존재한다.

‘w’ command : 다음단어의 첫 글자로 이동한다.

‘b’ command : 이전단어의 첫 글자로 이동한다

‘0’ command : 라인의 0번째 글자로 이동한다.

: 현재 라인의 첫번째 인덱스로 이동한다

‘o’ command : 커서 아래줄 부터 라인을 개행하고, 커서를 아래라인에 위치하게한다.

: 커서 밑의 라인들을 밑으로 내리고, 커서 바로 다음 라인은 공백+널문자로 채운다. 그리고 커서의 y좌표를 밑으로 내려준다.

‘^’ command : 라인의 첫 단어의 첫글자로 이동한다.

: 현재 라인 첫번째 인덱스부터 단어가 나올때까지 루프를 돌리고, 단어가 나오면 커서의 위치를 단어의 위치에 멈춘다

‘$’ command : 라인의 끝 단어의 끝글자로 이동한다

: 현재라인 마지막 인덱스부터 단어가 나올때까지 루프를 돌리고, 단어가 나오면 커서의 위치를 단어의 위치에 멈춘다.

“ : “ command : ex\_command mode로 전환한다.

: command\_mode Func에서 excommandmode에 해당하는 리턴값을 리턴한다.

‘/pattern’ command : 해당패턴을 찾고 커서를 위치시킨다.

: lines를 (\*x , \*y ) 🡪 (WIDTH, HEIGHT) 루프를 돌리면서, ‘search’의 첫 번째 원소와 같은값이 있는지 체크하고, 만일 있다면, pattern의 두번째 인덱스부터 pattern의 마지막 인덱스까지 루프를 돌리면서 그 단어가 일치하는지 체크한다.

‘dd’ command : 커서가 위치한 라인을 지우고, 밑에 있는 라인들을 위로 올린다.

1.7 Ex Command mode 기능 : 파일의 저장과 종료를 담당한다.

Command 는 다음과 같다.

‘w’: 파일을 저장한다. : lines의 내용을 카피해서 파일에 넣는다.

‘wq’; 파일을 저장하고, 종료한다 : lines의 내용을 카피해서 파일에 넣은후, 프로그램을 종료한다.

‘q!’:프로그램을 반드시 종료한다.

‘q’: 만일 파일이 변경된 내용이 없다면, 종료한다. 변경된 내용이 있다면 종료하지않는다.

1.8 Editor mode 기능 : 주로 insert, Enter, Backspace를 담당한다.

**1.9 추가구현 :**

1) VIMRC 파일 로드 : “vimrc”라는 파일을 로드 하여 파일에 특정 커맨드(“set nu”)가 있다면 해당 커맨드의 기능을 실행시킨다. 예를 들어 “set nu”는 에디터 박스의 왼쪽 편에 각 라인의 넘버를 보여준다.

2)ex command 모드의 커맨드에 “set nu”와 같은 커맨드를 입력하여 커맨드 적용: 에디터 박스의 왼쪽 편에 각 라인의 넘버를 보여준다.

3) /pattern : 커맨드모드에서 / 커맨드를 받았을 때 찾고자 하는 패턴을 입력받아서 해당패턴이 위치한곳에 커서를 위치시킨다.

4) :숫자 : 해당 라인의 첫번째 인덱스로 이동한다. (이때 라인의 0번째 인덱스는 숫자 1이다)

2. 기능 및 메커니즘의 구현

**0. 초기화**

프로그램이 시작하면,

FILE\*포인터 변수 file1을 선언하여 argv[1]에 해당하는 파일을 ‘r’로 오픈(파일 존재) 하거나 ‘w’로 오픈(파일 부재)한다. 또한 vimrc파일을 ‘r’또는 ‘w’로 오픈한다.

그리고 파일을 char\*\*형 메모리 공간에 복사한다.

<Tab>의 경우에도 마찬가지로 ‘\t’하나만을 메모리에 넣고, draw에서 세칸을 출력 하는식으로 처리한다.

void draw(char \*\*lines, int x, int y,char \*\*vimrc) 추가한 내용.

**1.”set nu”** : draw에서 char \*\*형 vimrc 변수를 할당된 공간만큼 읽어서, 만일 “set nu”라는

문장이 발견될 경우 shownu라는 int형 변수의 값을 1로 변경한다. 그리고 shownu의 값이 1일 때는 에디터의 왼쪽 벽을 출력하는 대신 ‘ ~’ 또는 라인의 넘버를 출력한다.

**2. <TAB>** : lines[][]에 <TAB>이 char형으로 하나 저장이 되지만, 공백을 세개를 출력하기 때문에 추가작업을 하지않는다면 에디터 박스가 변형될 수 있다. 이러한 문제를 해결하기 위해 각 라인에 <TAB>이라는 값이 몇 개가 있는지 확인 해야 한다. 그렇기 때문에 tabnuber[]에 탭의 숫자를 저장하고, draw의 루프를 돌 때 j는 WIDTH-tabnumber[i]만큼만 돌게 하여 에디터가 변형되는 문제를 해결할 수 있다.

**1. mod의 자유로운 변환.**

이를 위해 int All\_Func(char \*\*lines, int \*x,int \*y, int \*status,char \*filename,char vimrcset)라는 함수를 메인함수의 루프에 작성한다.

이때 All\_Func()는 함수가 끝날 때 마다 \*status를 return하고, \*status를 파라미터로 받아서 커맨드모드를 실행할것인지 익스커맨드모드를 실행할것인지, 에디터모드를 실행할것인지, 종료할것인지 결정하고 실행하는 프로그램의 큰 메커니즘을 구성한다.

이러한 메커니즘을 위해서는 각 모드에 대해 함수를 나눠주는 것이 필요하며, 코드작성

의 편의상 command\_mode()함수를 “commod.h”라는 헤더파일에 작성하였고,

Ex\_command\_mode()함수를 “excommod.h”라는 헤더파일에 작성하고, Editor\_mode()함수를 “editmod.h”라는 헤더파일에 작성해주었다.

또한 리턴 값이 해당하는 기능을 쉽게 알아보기 쉽게 하기 위해,:

defined COMA ---> Command\_mode

defined EDIT ---> Editor\_mode

defined EXCO ---> Ex\_Command\_mode

defined exit ---> if command is 'q' in Ex\_command\_mode, status is exit

defined EXIT ---> if able exit, status is EXIT, Program is end.

defined FAIL ---> if unable exit, status is FAIL

defined SAVE ---> if command is 'w', status is SAVE

defined SAEX ---> if command is 'wq', status is SAEX

#define 을 이용하여 리턴값의 기능을 알아보기 쉽도록 설정하였다.

**2. “commod.h” 헤더파일에 대한 설명과 커맨드모드의 기능 구현.**

이 헤더파일은 Command\_mode일때의 커맨드를 처리하기위한 함수와 이를 위해 필요한 함수들이 정의 되어있다.

**2-1. int Command\_mode(char\*\* lines, int \*x, int \*y, int \*filename,char \*\*vimrcset, int\* status)**

\*기능 구현 방식\*:

1) command ‘dd’ : 일단 라인넘버를 구하고, lines[\*y]의 원소를 모두 공백으로 만들고,

\*y~linenumber-1 라인의 원소를 각각 위의 라인으로 옮긴다.(라인을 위로 올리는 작업)

그리고 마지막 라인을 모두 널문자로 채우면서, y좌표를 1만큼 위로 올린다.

2) command ‘o’ : 일단 라인넘버를 구하고, 아까와 반대의 방식으로 라인을 아래로 내리는 작업을 실행한다. 그리고 현재 라인의 다음 라인을 첫번째 인덱스를 제외하여 널문자로 채워넣으면서, y좌표를 아래로 내리고 x좌표를 0으로 설정한다 그리고 EDIT을 리턴한다.

3) command ‘i’ : 에디터모드로의 변환을 위해 EDIT이라는 값을 리턴한다.

4) command ‘:’ : 익스커맨드모드로의 변환을 위해 EXCO라는 값을 리턴한다.

5) command ‘/’ : lines를 (\*x , \*y ) 🡪 (WIDTH, HEIGHT) 루프를 돌리면서, ‘search’의 첫 번째 원소와 같은값이 있는지 체크하고, 만일 있다면, pattern의 두번째 인덱스부터 pattern의 마지막 인덱스(strlen을 이용하여 구할수있다,)까지 루프를 돌리면서 그 단어가 일치하는지 체크한다.

Else. command ‘h’, ‘j’, ‘k’,’l’,’w’,’b’,’$’,’0’,’^’ : 해당 커맨드를 Cusor()함수에 전달한다.

**2.2 void Cursor(char \*\*lines, int\* x, int \*y, int value)**

6) command ‘h’ : x좌표를 1만큼 빼서 커서를 왼쪽으로 이동시킨다. 또한 x좌표가 0일때는 왼쪽으로 이동할수없다.

7) command ‘j’ : 만일 lines[y+1][x] 가 널문자가 아니라면, y좌표를 1증가시킨다.(커서를 밑으로 내린다.) , 널문자라면 다음라인의 끝 단어의 끝으로 이동시킨다. 이때 끝 단어의 끝이라는 것을 확인하기 위해 다음라인에서 WIDTH-1부터 시작해서 루프를 시작하여 널문자가 아닌 원소를 탐색한다. 그리고 탐색에 성공한다면 x와 y의 좌표를 탐색된 위치로 지정한다.

8) command ‘k’ : 만일 lines[y-1][x]가 널문자가 아니라면 y좌표를 1 감소시킨다. 널문자라면 널문자가 아닌 원소가 나올 때 까지 탐색하여 탐색된 지점을 커서의 좌표로 지정한다.

9) command ‘l’ : 만일 커서가 널 문자에 위치하여 있지 않다면 x좌표를 1증가시킨다.

10) command ‘0’ : x좌표를 0으로 변경한다

11) command ‘^’ : 현재라인의 첫번째 인덱스부터, 공백이나 탭이 아닌 원소를 찾는 탐색을 시작한다. 탐색에 성공하면 x를 탐색된 지점으로 지정한다.

12) command ‘$’ : 해당라인의 WIDTH-1 인덱스부터 0 인덱스까지, 널 문자 또는 공백 또는 탭 문자가 아닌 문자를 탐색하고, 만일 널,탭,공백문자가 아니라면 탐색을 멈추고 탐색에 성공한 지점을 커서의 좌표로 x좌표를 변경해준다.

**‘w’ 와 ‘b’ command**

13) command ‘w’ : 크게 두 가지 케이스로 나눠서 코드를 작성하였다. 커서의 위치가 공백 또는 탭 문자 또는 널 문자 일 때 flag =0으로 , 커서의 위치가 공백 또는 탭 문자 또는 널문자가 아니라면 flag = 1로 케이스를 설정하였다.

Case1. flag == 0 : 커서의 위치부터 공백,탭,널 문자를 제외한 문자가 나올 때 까지 순차적으로 탐색을 진행하여, 공백,널,탭 문자가 아닌 경우, 즉 탐색에 성공하면 해당 지점을 커서의 지점으로 지정한다.

Mechanism: cursor(start) 🡪 Search 🡪 word(탐색성공)

Case2. flag == 1 : 커서의 위치부터 탐색을 진행하는데, 커서가 flag==0에 해당되는 지점을 지나면 flag = 0으로 flag의 값을 변경하고, flag == 0 인 상태에서 flag == 1에 해당되는 지점을 만나면 탐색을 중지하고, 탐색에 성공한 지점을 커서의 위치로 지정한다.

Mechanism: cursor(flag == 1) 🡪 search 🡪 space or ‘\0’ or ‘\t’, flag == 0 🡪 search 🡪 flag == 1, search success

예외 : search를 하는 중 특수문자가 나오면 search를 중지하고, 해당지점을 커서의 위치로 지정한다.

14) command ‘b’ : ‘w’와 마찬가지로 케이스를 크게 두 가지로 나눠서 코드를 작성하였다.( flag == 0일때, flag == 1일때) , 또한 Case1. flag==1인경우에는 다시 두가지로 케이스로 나눠서 코드를 작성하였다.(커서가 단어의 첫번째 인덱스일 때, 그렇지 않을 때)

Case1. flag == 1:

Sub case1. 커서가 단어의 첫번째 인덱스인 경우.

Mechanism:

cursor(flag == 1,tmp==0) 🡪search 🡪 space or tab or null(flag ==0, tmp += 1, tmp = 1) 🡪search 🡪word(flag==0, tmp = 1)🡪space(flag == 0, tmp += 1, tmp = 2) 🡪 set cursor position.

설명: 커서의 위치(flag == 1,tmp == 0 상태)부터 왼쪽에서 위로 탐색을 시작하고 , 만일 tmp 가 0이고 flag 가 1인 상태에서 flag ==0에 해당 되는 지점을 지난다면 tmp의 값을 1증가시키고 flag를 0으로 변경한후, 탐색을 계속 진행한다. 탐색을 하면서 flag가 1인 지점을 지난다면 flag를 1로 변경한후 탐색을 계속 진행한다. 그리고 탐색을 진행하면서

마찬가지로 flag가 1값 일때 flag==0에 해당하는 지점을 만난다면 flag 를 변경하고 tmp를 1증가시킨다. 이때 가 tmp가 2값을 갖고있다면 탐 색을 중단하고 커서가 탐색에 성공한 지점의 오른쪽지점을 가리키게 하 면서 ‘b’ 커맨드를 최종적으로 종료한다.

:예외. 특수문자인 지점을 지날 때는 그지점을 커서의 위치로 지정하고 커맨드 종료

Sub case2. 커서가 단어의 첫번째 인덱스가 아닐때.

Mechnism.

Cursor 🡪 search 🡪 space or tab search success, set cursor position

설명: 커서의 위치부터 왼쪽에서부터 탐색을 진행하고, 만일 스페이스 또 는 탭을 만난다면, 그 지점의 오른쪽 지점을 커서의 좌표로 지정하고 커 맨드를 마친다.

:예외. 특수문자인 지점을 지날 때는 그 지점을 커서의 위치로 지정하고 커맨드 종료..

Case2. flag == 0:

Mechanism.

Cursor(flag == 0) 🡪 search 🡪 word(flag ==1) 🡪 space(flag ==0) 🡪set cursor position

설명: 커서의 위치(flag ==0)부터 탐색을 진행하고, flag가 1인 지점을 지 지날 때는 flag를 1로 변경하고 flag가 0인 지점을 지날때는 flag 를 0으 로 변경한다. 만일 공백이나 탭을 만났을때 이전 flag가 1인 상태였다면 탐색된 지점의 오른쪽 지점을 좌표로 지정하고 커맨드를 종료한다.

: 예외 . 특수문자인 지점을 지날때는 탐색을 중지하고 그 지점으로 커서 의 위치를 지정해준다.

#w또는 b 명령어 관련 참고사항. 만일 탐색하는 라인에 탐색대상이 없을 때는 다음라인 또는 이전라인에서 탐색을 진행한다. 그리고 이런 과정의 구현은, 커서의 좌표를 다음라인으로 변경하고 다시 Cursor()함수를 불러오는 식으로 구현하였다.

**3. “excommod.h” 헤더파일에 대한 설명과 커맨드모드의 기능 구현**

이 헤더파일은 Ex\_command\_mode일 때 커맨드를 처리하기위한 함수와 이에 필요한 함수들로 구성되어있다. 주로 저장과 종료와 관련된 기능을 수행한다.

**3-1. int Ex\_command\_mode(char\*\*lines,int\*x,int\*y,char\*filename,char\*\*vimrcset,int \*status)**

커맨드를 gets함수로 받고 커맨드에 해당하는 기능을 실행한다.

#설명이 필요한 변수\*

1. char \* 형 command : 명령어를 입력 받았을 때 저장하는 공간이다.
2. Int 형 vimrclen : vimrcset 의 세로 길이를 저장한다. :

#기능 :

1. ‘w’ command : SAVE 값을 리턴 한다.
2. ‘q’ command : exit 값을 리턴한다.
3. ‘wq’ command : SAEX 깂을 리턴한다
4. ‘q!’ command : EXIT 값을 리턴한다.
5. 참고 : 각 Mod함수들에서 어떤 값을 리턴하면 status에는 해당 리턴 값이 저장되고 메인함수의 All\_Func함수는 해당 리턴 값을 메인 함수의 status에 리턴 한다. 그리고나서 메인 함수의 루프를 돌면서 All\_Func를 실행하는데 이때 status를 파라미터로 받게 되고 ,All\_func는 status에 저장된 리턴 값에 따라 모드를 실행하는 구조이다.

이때 SAVE와 exit, SAEX는 커맨드 모드를 실행하게 하게 되는데, command모드에서는 SAVE, exit, SAEX과 같은 status를 “excommod.h”에 저장되어있는 System\_check()라는 함수에 파라미터로 전달하면서 System\_check에서는 리턴값에 따라서 저장을 하는 기능을 수행 하거나, 종료를 할 것인지 종료를 할 수 없는지 또는 커맨드모드에 그대로 있을 것인지를 리턴 해주는 메커니즘이다.

즉 정확히는 ex\_command모드함수에서는 저장이나 종료와 같은 커맨드를 받았다는것을 전달해주는 역할을 하고, 저장이나 종료기능을 수행하지는 않는다.

.

# Excommand 추가구현 :

1. ‘:숫자’ command : 해당숫자의 라인의 첫번째 인덱스로 이동하고 COMA를 리턴한다.

Case1. 만일 커맨드의 첫번째 원소가 숫자에 해당하는 아스키값을 갖고있으면서, 커맨드의 길이가 1이라면, x값을 0으로 변경하고, y값을 첫번째 원소에서 ‘0’-1를 뺀 값으로 변경한다.

Case2. 만일 커맨드의 첫번째 원소가 숫자에 해당하는 아스키값을 갖고있으면서 커맨드의 길이가 2이고, 두번째 원소또한 숫자에 해당하는 원소를 갖고있다면 , x값을 0으로 변경하고, y값을 (첫번째원소값-‘0’)X10 – (두번째원소값-‘0’)-1로 변경한다.

1. ‘set nu’ command : 에디터에 라인의 넘버를 보여준다.

$1. 실행조건. 커맨드를 strcmp함수를 이용하여 커맨드 여섯 자리와 “set nu”문장이 일치하는지 비교하고 strlen으로 커맨드가 여섯 자리인지 확인하여 기능을 실행한다.

$2.구현 방식: excommandmode함수에 파라미터로 받은 vimrcset의 마지막 라인의 다음라인, 즉 vimrcset의 vimrclen+1라인에 strcpy 함수를 이용하여 vimrcset에 “set nu” 라는 스트링을 추가(복사)해준다. 그리고 COMA 값을 리턴하여 커맨드모드로 변환 한다.

$3. 참고 : 각 모드가 실행 될 때 마다 draw 함수를 실행하는데 , vimrcset은 draw에 파라미터로 전달된다. 그리고 draw에서는 vimrcset에 “set nu”라는 문장이 있는지 체크하고, 에디터에 숫자를 출력할지 여부를 결정한다.

$4. strcpy함수는 <string.h>헤더파일에 내장된 함수이다.

draw 함수는 “a.h”헤더파일에 작성되어있다.

1. “set nonu” command : 에디터에 라인의 넘버를 보여주는 것을 멈춘다.

$1. 실행조건 : 커맨드를 strcmp함수를 이용하여 커맨드 여덟 자리와 “set nonu”문장이 일치하는지 비교하고 strlen으로 커맨드가 여덟 자리인지 확인하여 기능을 실행한다.

$2. 구현 : vimrcset의 각 라인을 탐색하면서 , strcmp를 활용하여 vimrcset에 “set nu”라는 문장이 존재하는 라인을 탐색하고, stycpy를 활용하여 그 라인을 공백으로 채워 넣는다. 그리고 COMA라는 값을 리턴하다.

$3. strcmp함수는 <string.h> 헤더파일에 내장된 함수이다.

**3-1. int System\_check(char\*\*lines, char\*filename, int status, char \*\*vimrcset )**

1. SAVE( ‘w’) :

1)기능: 편집내용을 파일에 저장하고 커맨드모드로 전환한다.

2)구현: afterFile이라는 FILE\* 형 변수를 이용하여 filename(기존 텍스트 파일)이라는 파일을 “w”모드로 열고, lines에 있는 내용을 char형 문자 하나 하나씩 fputc함수를 이용하여 파일에 순차적으로 저장(복사)한다. 그리고 COMA를 리턴 한다.

2. SAEX(‘wq’):

1) 기능: 편집내용을 파일에 저장하고, 에디터를 종료 시킨다.

2) 구현: SAVE와 동일한 방식으로 파일에 편집내용을 저장하고, EXIT을 리턴한다.

3.exit:(‘q’)

1) 기능: 저장되지 않은 내용이 없다면 에디터를 종료하고, 저장되지않은 내용이 있 다면 FAIL을 리턴 하여, 다음 커맨드모드에서 경고 메시지가 출력되게 한다.

2) 구현: beforeFile이라는 FILE\*형 변수를 이용하여, filename 파일을 “r”모드로 오픈하 고, beforeStr이라 char\*\*형 변수를 선언하고, 33\*88만큼을 malloc으로 할당한 다. 그리고 beforeFile변수를 통해 파일이 종료될 때 까지 파일을 읽으면서,

파일에 저장되 있는 문자(공백 탭포함)들을 beforStr에 저장(복사)헤준다.

그 다음 beforStr의 원소들과 lines의 원소들이 일치하는지 다른지를 체크하 여 만일 다른 원소가 존재한다면 FAIL을 리턴 하여 주고, 만일 다른 원소가 없다면 EXIT을 리턴 하여 준다.

1. **“editmod.h” 헤더파일에 대한 설명과 에디터모드의 기능 구현**

**4-1. int Editor\_mode(char\*\*lines,int\*x,int\*y,char\*filename,char\*\*vimrcset)**

에디터 모드의 전반적인 메커니즘을 구성한다. 키를 입력 받고, 키에 따라서 모드 전환할 것인지, 편집할것인지 결정한다..

만일 커맨드가 ESC가 아니라면 “editmod.h”헤더파일에 작성된 Edit\_ch()함수에 커맨드를 전달하면서 실행시킨다. 커맨드의 기능의 실행은 Edit\_ch()함수에서 이루어진다.

**4-2. void Edit\_ch(char \*\*lines, int \*x, int \*y, char c)**

키(char c)의 적용을 담당한다..

1. BACKSPACE : lines[y][i]의 원소를 lines[y][i-1]에 저장한다. 이 작업을 x(커서의 x좌표)부터 WIDTH-1까지 반복한다.
2. ENTER : shift의 개념(배열의 원소를 해당 방향으로 옮기는 것)을 이용해서 lines의 y+1라인부터 아래로 옮겨주고, y라인에서 x보다 큰 인덱스에 해당하는 원소들을 y+1 라인의 0번째 인덱스부터 저장(복사)해 준다. 그리고 y라인의 x보다 큰 인덱스에 해당하는 원소들을 널문자로 변경한다. 마지막으로 y를 1증가 시키고 x를 0 으로 변경한다.
3. INSERT&TAB&SPACE: shift의 개념을 이용하여, lines의 y라인의 x부터 WIDTH-1까지 Shift\_Right(오른쪽으로 당긴다)하고 lines[y][x]에는 키(c)를 넣어주고 x를 1 증가시킨다. 이때 탭도 마찬가지로 shift\_right을 하고 lines[y][x]에 ‘\t’을 넣어준다. 참고로 탭은 draw에서 처리되게 작성하였기때문에 에디터 모드에서는 별도의 예외처리가 없다. 스페이스도 마찬가지 방식으로 구현되었다.

**5 . “a.h” 헤더파일**

**5.1 void draw(char \*\*lines, int x, int y, char\*\* vimrc)**

0) 파라미터 vimrc가 추가됨: 메인 함수에서 “.vimrc”파일의 내용을 vimset이라는 포인터영역에 저장되어지고, vimset은 draw의 파라미터로 전달되어진다.

1) “set nu” 구현 : Draw함수에서는 vimrc를 “string.h”헤더파일의 strncmp파일을 이용하여 “set nu”라는 문장이 있는지 체크한다. 만일 있다면 shownu라는 변수에 1값을 넣고 없다면 shownu라는 변수에 0값을 넣는다. 그리고 shownu의 값이 1이라면 왼쪽벽을 출력하는 대신 숫자 또는 “~”를 출력한다.

2)TAB의 구현: 일단 draw에서 lines의 원소들을 출력한때, 만일 원소가 탭이라면 세 칸을 출력하도록 한다. 또한 한 라인당 출력하는 원소 수를 tabnumer[i]\*2번 덜 출력하게 하여 에디터의 박스 형태를 보존한다.

3) #define 사용

#define FAIL 21

#define EXIT 11

#define EDIT 12

#define EXCO 13

#define COMA 14

#define exit 22

#define SAVE 15

#define SAEX 16

#define SEAR 17

#define SEARR 18

1. [↑](#endnote-ref-1)
2. [↑](#endnote-ref-2)