

Text Editor (final report)

20203043 권수현

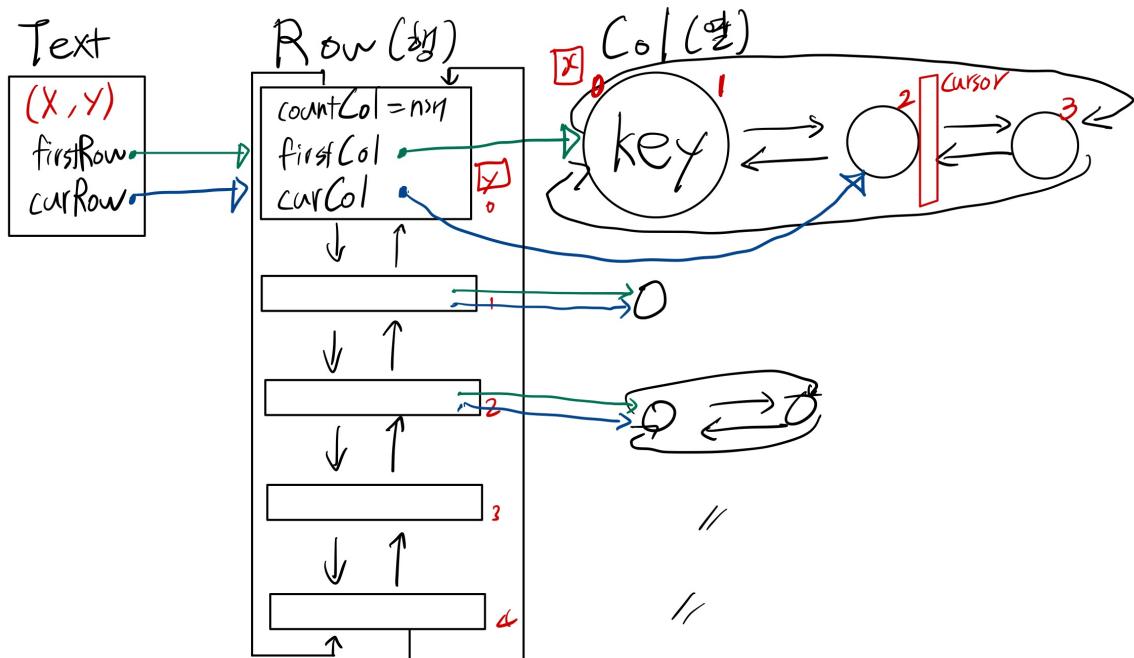
목차

1. 설계 방식
 - 1-1 문자를 받는 구조
 - 1-2 화면 출력 방식
 2. ADT 구현
 - 2-1 구조체
 - 2-2 기능 함수들 (textedit.c/h 파일)
 3. OS에 따른 컨디셔널 컴파일링
 - 3-1 조건부 전처리문에 OS 구분
 - 3-2 OS 별로 같은 기능을 구분하여 만들기
 - 3-3 makefile / 3가지 OS에서 컴파일 실험
- + 오류 또는 다른 방식 설명 : ※ 빨간글자
-

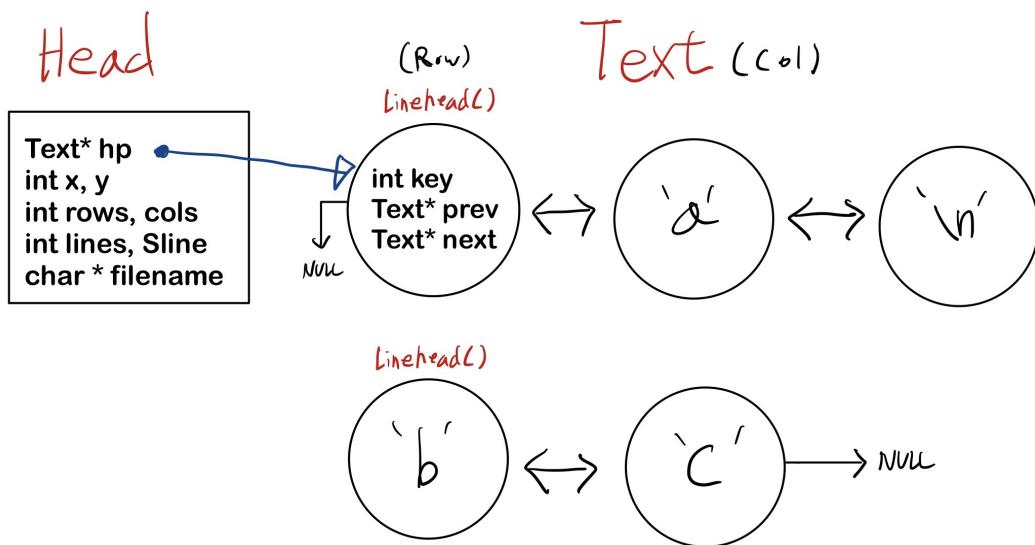
1. 설계 방식

문자를 받는 구조

- 초기 : text, row, col 구조체를 CircleDLL로 연결한 방식



- 수정 : 기존 방식에서 row 구조체를 삭제, CircleDLL에서 DLL 방식으로 변경



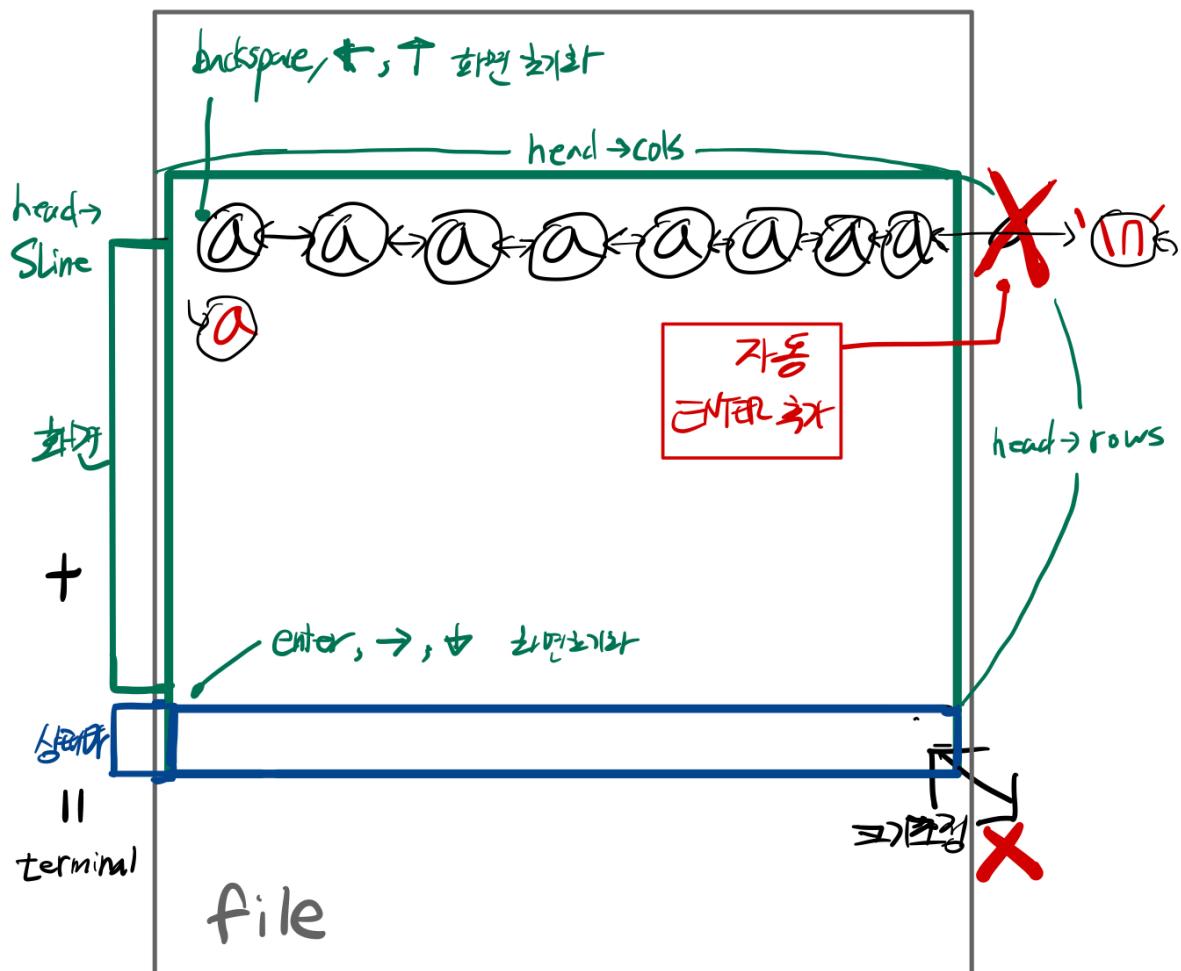
수정 후 해결된 문제점 : CircleDLL에서 DLL로 바뀌어서 조정할 포인터가 줄고 구조체 종류가 하나 줄어들어서 코드가 간단해졌다.

화면 출력 방식

※ 다른방식 : 설계 초기부터 터미널 크기가 고정되었다고 가정하고 구현하였다.

↳ 따라서 터미널 크기 조정 시 오류가 발생하고 터미널 범위를 넘어서면 자동으로 enter가 추가되고 다음 줄 X=0 부터 입력이 시작된다.

이유 : 설계서에서 화면 크기 조정과 글자 입력의 화면의 x축 끝을 넘어갈 때 글자 표현 방식의 언급이 없었다. 첫 5분 제한 발표수업에서야 위 두 가지 기능이 언급 되어서 알게 되었다. ncurses를 사용하여 수정하면 가장 좋지만 프로그램 전체를 수정하기에 리스크가 커서 그대로 진행하였다.



- <curses.h> 사용 X, 특정 경우 화면 초기화를 통해 구현

- 처음에 딱 한번 터미널 크기(rows/cols)를 받아온다.
 - \hookrightarrow head \rightarrow rows(행) = rows - 3 (하단바 제외) / head \rightarrow cols(열) = cols
- 대부분의 경우를 직접 화면을 초기화하여 구현
 - 화면 깜빡임과 속도 느려짐을 고려해서 최대한 초기화를 적게 구현

2. ADT 구현

구조체

- Text , Head 구조체

```
typedef struct Text {
    int key;
    struct Text* prev;
    struct Text* next;
}Text;

typedef struct Head {
    Text* hp; //text구조체의 head를 가리킴
    int x, y; //현재 커서 x,y
    int rows, cols; //화면 최대 행, 열
    int lines; //입력된 라인 수
    int Sline; //출력될 라인의 시작
    char* filename; //파일이름
}Head;
```

Text 구조체 : 글자를 저장하고 DLL로 연결 가능

Head 구조체 : 연결된 Text 구조체의 Head를 가리키고 전체적인 Text Editor를 관리

- Row 구조체 대체 \rightarrow searchLineHead 함수

```
Text* searchLineHead(Head* head)
{
    Text* tmp = head->hp;
    int countline = -1;

    while (tmp != NULL)
    {
        if (tmp->key == ENTER) countline += 1;
        if ((head->y + head->Sline) - 1 == countline)
        {
```

```

        return tmp; //head는 '\n'
    }
    tmp = tmp->next; //마지막 \n이 linehead 값
}

return head->hp; //countline == -1
}

```

Head 구조체가 가리키는 첫 텍스트를 제외하고는 2번째 line이상부터는 해당 줄의 head(첫 글자)를 가리키기가 힘들다.

따라서 ENTER ('\n')를 기준으로 linehead를 찾아준다.

기능 함수들 (textedit.c/h 파일)

- editText 함수 : **전체 기능 담당** (복잡한 기능은 따로 함수로 만듦)

1. _getch()로 키보드의 아스키 코드 값을 받는다.
2. key값에 맞게 함수를 실행하여 **텍스트 수정** 또는 **이동**을 수행한다.

이 때 Text DLL가 재조정 되며 Head 구조체 안에 x,y,lines 등의 여러 정보들도 업데이트 된다.

3. 꼭 필요한 경우에만 화면과 하단바를 초기화 한다.

- Add(), Delete() : **텍스트 수정**

자료구조 시간에 배웠던 DLL의 중간 삽입/삭제 코드를 참고하여 구현하였다.

1. 현재 커서 ($head \rightarrow x$, $head \rightarrow y$) 위치와 해당 줄의 linehead를 고려한다.
 - ↳ 현재 y축과 페이지 시작 라인($head \rightarrow Sline$)을 고려해서 searchLineHead() 함수로 해당 라인을 찾고 현재 x축을 찾는다.
2. 해당 좌표에 삽입 삭제에 맞게 Text DLL을 재조정한다.

- **이동**

1. cursor() : 화살표 키

UP, DOWN, LEFT, RIGHT 기능을 수행한다.

터미널 창의 크기는 제한 되어있기에 이를 모두 고려하였다.

기능은 메모장과 동일하다.

2. HOME, END, PAGE UP/DOWN

HOME, END는 메모장과 동일하다.

PAGE UP/DOWN은 해당 줄(head → Sline)을 기준으로 터미널의 전체 행(head → rows) 만큼 up, down 한다.

↳ 예외 경우

page up : 더 이상 상위 페이지가 없을 시 (0,0)으로 이동. / page down: 더 이상 하위 페이지가 없을 시 전체 글의 마지막 줄로 이동.

- **initWindow()** : 화면 초기화

1. system("cls")로 화면을 비운다.

2. 스크롤을 위해서 tmp를 시작 줄(head → Sline)에 첫 글자로 이동 시킨다.

3-1. tmp가 NULL을 가리킬 때 까지 문자(영어, 빙칸) 또는 ENTER('\n')를 출력한다.

3-2. ENTER('\n')는 cnt2로 Enter의 수를 count하여 cnt2 == 마지막 줄(head → rows + 2) 일 때까지 출력하게 하여 화면을 터미널의 Row만큼만 출력 가능하게 한다.

+ 큰 파일을 불러 오는 경우는 따로 화면 초기화를 한다.

- **initToolbar()** : 상태 바, 메시지 바

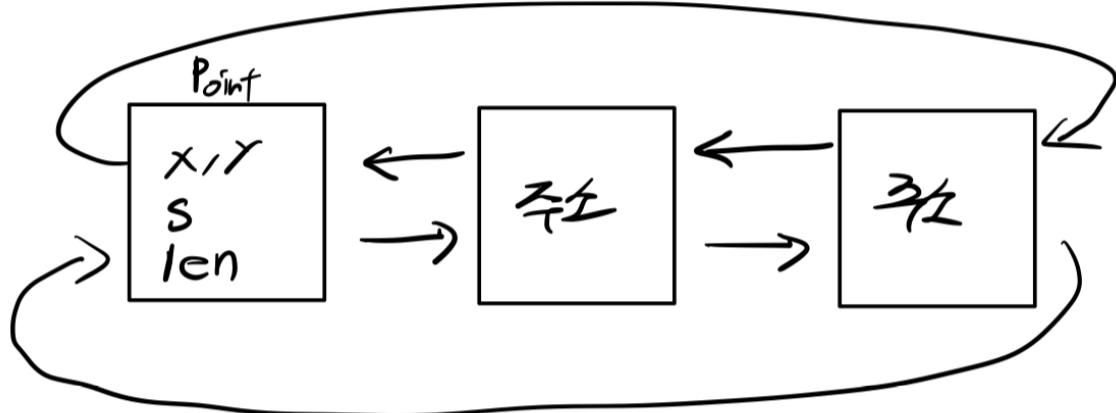
터미널 행의 마지막으로 이동하여 상태 바, 하단 바를 출력한다.

※ 다른방식 : 설계서에 ‘현재 커서 위치 표시해야 한다’를 x/y축으로 이해하고 표시하였다.
따라서 no ft 다음에 표시되는 건 현재 터미널에 표시되는 커서의 x,y 좌표값이다.

- Ctrl + f : 탐색

Point 구조체 : 탐색 단어의 주소를 저장하며 CircleDLL로 연결을 위한 구조체

```
typedef struct Point {
    int x, y;
    int s; //시작라인
    int len; //단어 길이
    struct Point* prev;
    struct Point* next;
}Point;
```



find() : 탐색 기능 함수

(발표 수업 중 교수님께서 말씀하신 주소저장 방법을 적용)

1. 탐색하고자 하는 문자열 입력 '\n'키로 입력완료
2. 탐색 단어가 있는 Text 구조체의 주소(head → x/y, head → Sline)를 모두 찾아 Circle DLL에 연결
3. 방향 키 입력 시 찾아둔 주소를 바탕으로 탐색 글자 길이 만큼 색 반전 및 해당 단어로 커서 이동
4. ENTER키 입력 시 마지막 탐색 위치로 커서 이동하고 탐색 기능 종료, 따라서 탐색한 위치에서 수정 가능
5. ESC키 입력 시 탐색 종료

↳ ※ Esc 두 번 입력 필요 (조건문에 키값을 무조건 받아야 해서 수정이 불가능, 시간 초로 제어를 해보려 했지만 실패)

- save() : Ctrl + S, 저장

```

FILE* file;
//파일 저장
if (head->filename != NULL)
{
    file = fopen(head->filename, "wt");
    while (tmp != NULL)
    {
        fputc(tmp->key, file);
        tmp = tmp->next;
    }
}

```

```
    fclose(file);
}
```

모든 Text 노드들을 fputc를 이용해 파일에 쓴다. (fopen, “wt”)

- Ctrl + q / Ctrl + q X 2 : 나가기

1. Ctrl+q 입력 시 “Enter/Ctrl+q” 문구가 하단바에 출력된다.
2. 이후 Enter 입력 시 : Ctrl+q를 한 번 입력한 걸로 받는다.
 - ↳ 저장된 적이 없는 파일이면 저장을 실행한다.
 - ↳ 저장된 적이 있는 파일이면 저장 없이 나가기를 실행한다.
3. 이후 Ctrl+q 입력 시: Ctrl+q를 두 번 입력한 걸로 받는다.
 - ↳ 나가기를 실행한다.

- 파일 불러오기 : 실행 , 명령행 인자 사용

```
int main(int argc, char* argv[])
{
    if (argc == 2)
    {
        // 기존 파일 열고 텍스트 편집
    }
    else
    {
        createToolbar(head);
        // else 새 파일 경우 초기화면 출력
    }
    // 텍스트 편집 시작
}
```

1. 새 파일 : 기존 그대로
2. 기존 파일 : fgetc(file)로 key값을 text노드에 넣어 연결하였다. 속도를 위해 반복문 없이 ‘p’포인터로 DLL의 가장 마지막을 가리키게 하여 연결하였다. (fopen, “rt”)
 - ↳ 이 때 속도를 위해 fileWindow() 로 initWindow()와 달리 제일 첫 페이지를 출력한다.

3. OS에 따른 컨디셔널 컴파일링

조건부 처리문에 OS 구분

```
#ifdef _WIN32 // Windows 플랫폼
#include <conio.h> // _getch
#include <windows.h>
#define UP 72
#define DOWN 80
#define RIGHT 77
#define LEFT 75
#define BACKSPACE 8
#define HOME 71
#define END 79
#define PAGE_UP 73
#define PAGE_DOWN 81
#define ENTER 13
#define CLEAR_SCREEN "cls"
#else // Linux 및 macOS 플랫폼
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>
#define UP 65
#define DOWN 66
#define RIGHT 67
#define LEFT 68
#define BACKSPACE 127
#define HOME 1
#define END 4
#define PAGE_UP 53
#define PAGE_DOWN 54
#define ENTER 10
#define CLEAR_SCREEN "clear"
#endif
```

#define ESC 27, Ctrl_f 6, Ctrl_s 19, Ctrl_q 17. 이 4가지 키를 제외하고는 모두 달라서 구분 하였다.

OS 별로 같은 기능을 구분하여 만들기

- 터미널 크기를 받아오는 부분

```
#ifdef _WIN32 // Windows 플랫폼
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    int rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
```

```

        int cols = csbi.srWindow.Right - csbi.srWindow.Left + 1;
#else // Linux 및 macOS 플랫폼
    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
    int rows = w.ws_row;
    int cols = w.ws_col;
    // Ctrl+S 및 Ctrl+Q 기능 비활성화
    struct termios term;
    tcgetattr(STDIN_FILENO, &term);
    term.c_iflag &= ~(IXON | IXOFF);
    tcsetattr(STDIN_FILENO, TCSANOW, &term);
#endif

```

리눅스에서는 <window.h>로 터미널 크기를 받아 올 수 없기 때문에 <sys/ioctl.h>를 사용하였다.

- 키보드 키를 받아오는 부분

```

int custom_getch()
{
#ifdef _WIN32
    return _getch();
#else
    struct termios oldt, newt;
    int ch;

    // 현재 터미널 설정 저장
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;

    // 입력 모드 수정 (버퍼 없이, 문자 모드)
    newt.c_lflag &= ~(ICANON | ECHO);

    // 터미널 설정 적용
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);

    // 키 입력 받기
    ch = getchar();

    // 이전 터미널 설정으로 복구
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);

    return ch;
#endif
}

```

리눅스는 <conio.h>를 사용해서 _getch() 함수를 사용 할 수 없기 때문에 <termios.h>를 사용해서 custom_getch() 함수를 만들었다.

makefile / 3가지 OS에서 컴파일 실험

“vite.c, textedit.c, textedit.h” 파일을 vite 프로그램으로 만드는 makefile

```
CC = gcc
CFLAGS = -Wall -Wextra

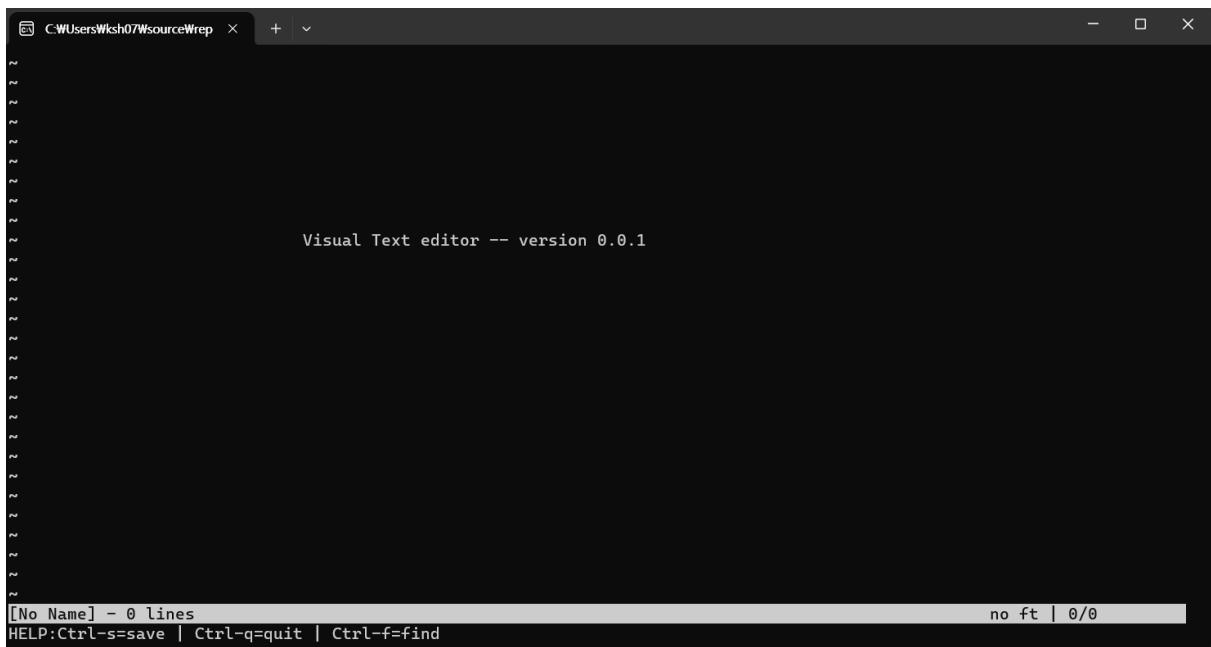
vite: vite.o textedit.o
    $(CC) $(CFLAGS) -o vite vite.o textedit.o

vite.o: vite.c textedit.h
    $(CC) $(CFLAGS) -c vite.c

textedit.o: textedit.c textedit.h
    $(CC) $(CFLAGS) -c textedit.c

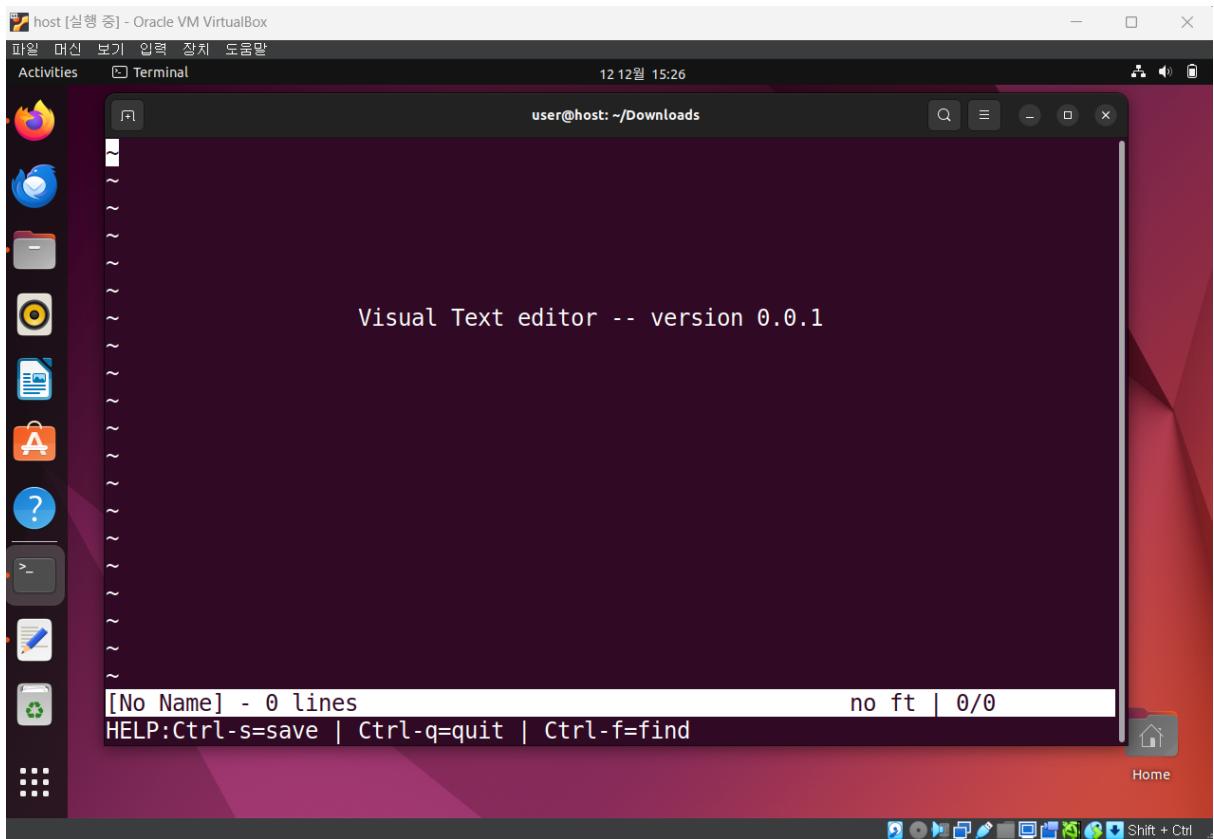
clean:
    rm -f *.o vite
```

- 윈도우 컴파일



└ 원활히 컴파일 된다.

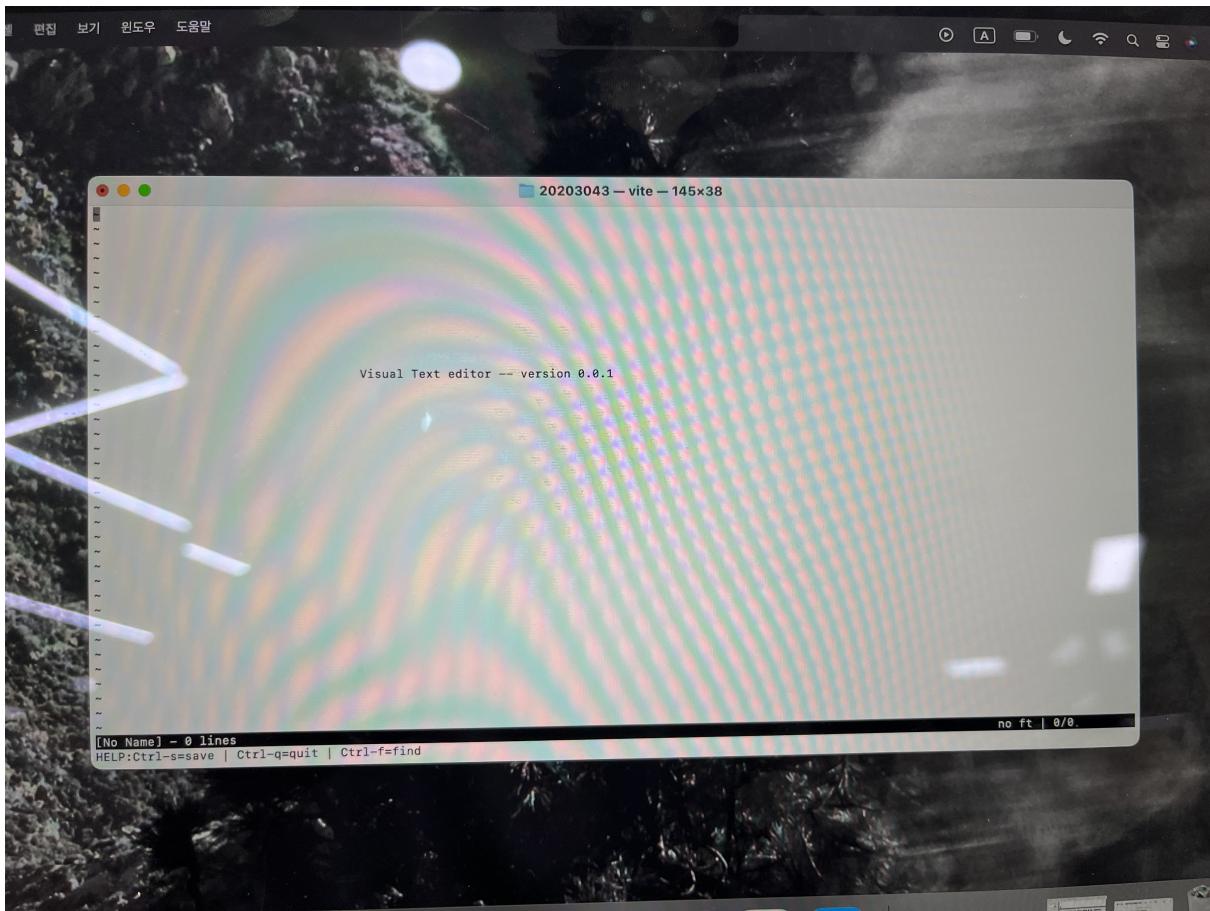
- 리눅스 (oracle) 컴파일



└ 원활히 컴파일 된다.

※ 오류

1. **end**키만 동작이 오류가 난다. (키 값을 제어해도 동일하여 포기하였다.)
 2. 터미널 창이 전체화면인 상태에서 프로그램을 시작하면 터미널의 행렬을 1씩 작게 받아온다. (다른 크기의 터미널은 X, 이유를 알 수 없음.)
- Mac OS 컴파일



└ 원활히 컴파일 된다.

※오류

ctrl+s와 ctrl+q 키는 터미널 제어를 통해 해결. 하지만 **page up/down, home/end** 키는 ncurses와 같은 헤더 파일을 사용하지 않아 기능을 끄지 못하였다.

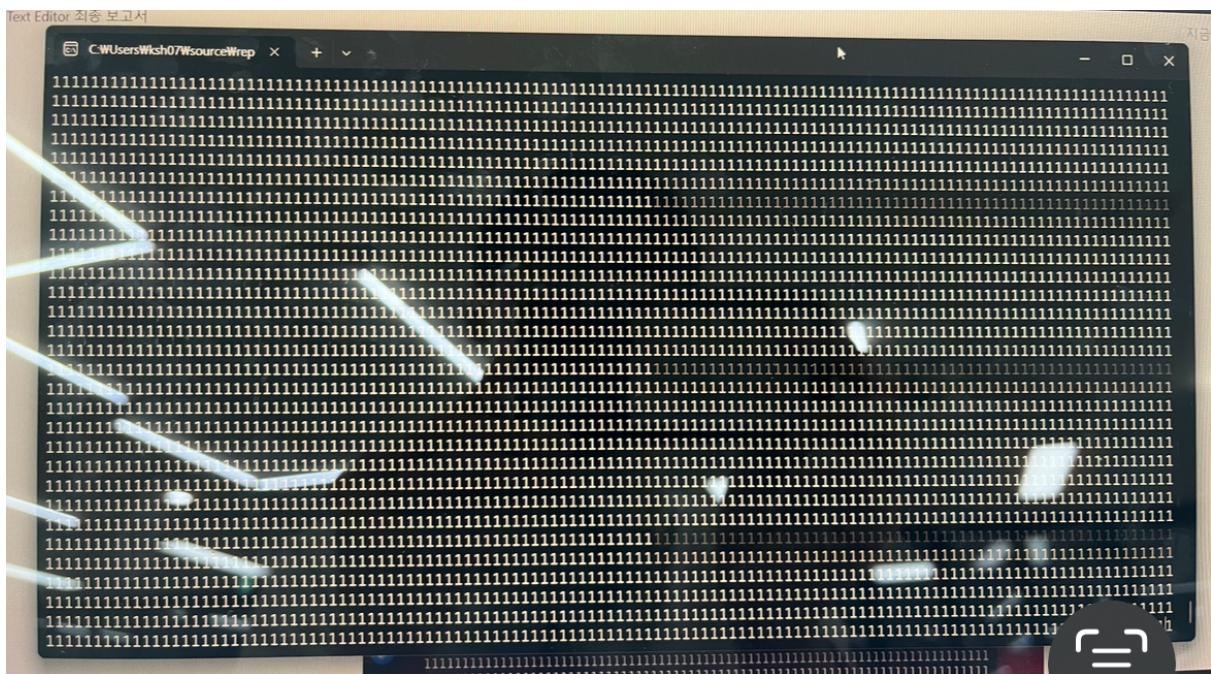
※ 공통적 오류 : 5MB이상의 큰 txt 파일을 불러왔을 때 불러와서 text노드에 저장은 되지만 글자 수정 기능이 제대로 동작하지 않는다.

다음은 5MB.txt 파일로 테스트한 결과이다.

- 윈도우

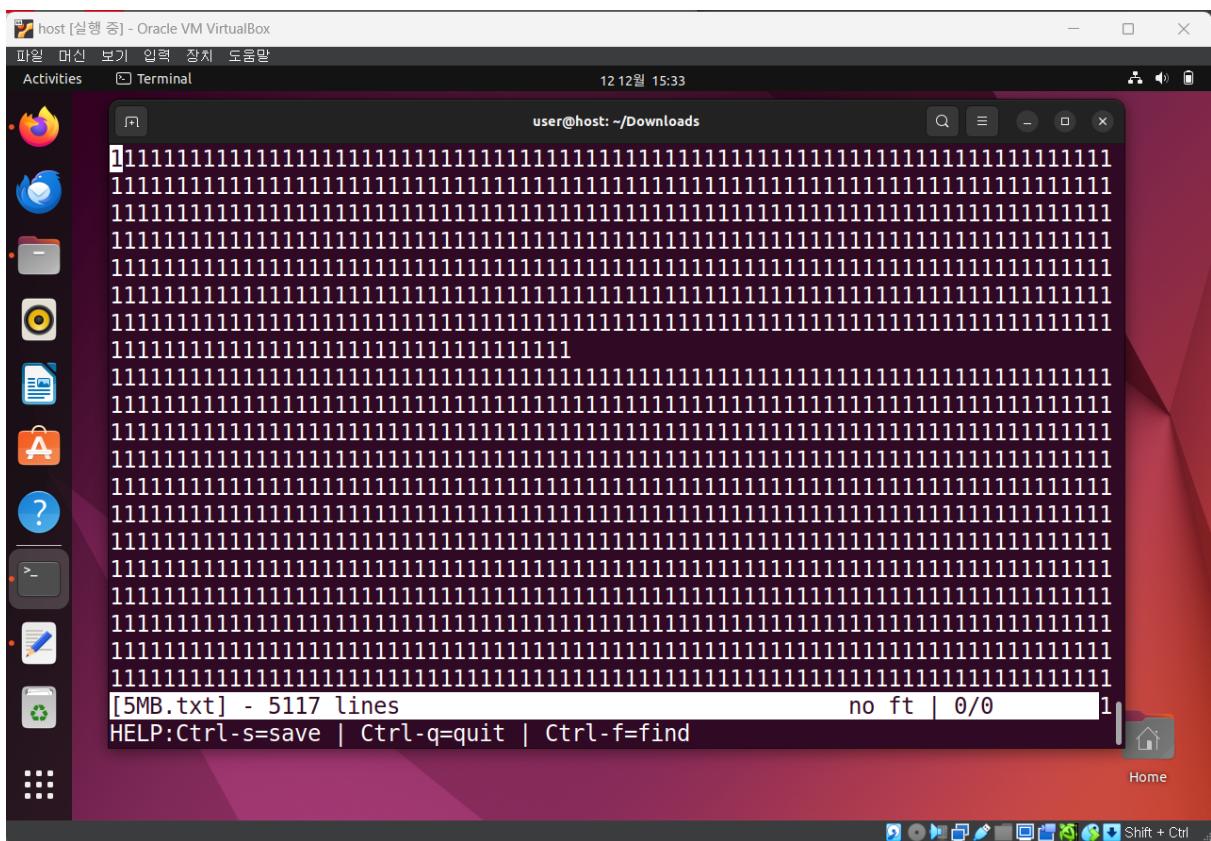
시간오버

이유 : 윈도우에서만 큰 파일을 불러올 때 printf와 같은 구문이 없는데도 터미널에 모든 글자가 나열되다가 전부 스크롤 되며 나열되고 나서야 내가 작성한 프로그램이 시작한다. (이유를 모르겠다.)



- 리눅스

1초 안에 원활히 동작한다.



- 맥

1초 안에 원활히 동작한다.

