2023-2학기 고급자료구조

Visual Text Editor (vite)

프로젝트 설계서

20203128 김재영

목차

1. 개요

vi와 같은 개발자 용 텍스트 에디터를 설계한다. 텍스트 에디터는 C언어로 구현하며 리눅스, 윈도우OS, 맥OS 환경에서 컴파일 되고 실행 가능하여야 한다. 또한 컴파일은 makefile을 통해서 수행되어야 한다.

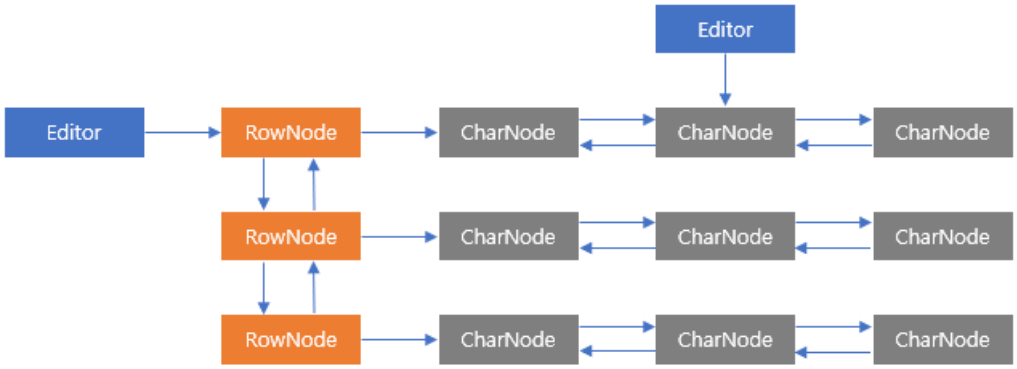
2. 분석

텍스트 에디터는 다음과 같은 기능을 제공한다.

- **텍스트 입력 및 편집**: 사용자는 키보드를 통해 바로 문자를 입력할 수 있고, 뉴라인 문자, 백 스페이스, 라인 나누기 등의 작업을 할 수 있다.
- **이동**: 화살표 키, 홈 키, 엔드 키, 페이지 업, 페이지 다운을 사용하여 빠르게 커서를 이동할 수 있다.
- **파일 생성 및 열기**: 새로운 파일을 생성하거나 기존 파일을 열 수 있다. 새로운 파일을 생성할 경우, 명령어 행에서 'vite' 형식으로 실행한다. 기존 파일을 열 경우, 명령어 행에서 'vite [파일 이름]' 형식으로 실행한다.
- **상태 바**: 에디터의 현재 상태를 보여준다. 이곳에는 '파일 이름', '라인 수', '현재 커서 위치'를 표시하며 맨 아래에서 2번째 라인에 반전 색으로 표시한다.
- **메시지 바**: 맨 아래 라인에 '저장', '탐색', '나가기' 방법을 표시한다. 필요에 따라 다른 정보를 출력한다.
- **저장**: 수정된 파일은 Ctrl-s를 사용하여 저장할 수 있다. 새로운 파일을 저장할 때는 파일 이름을 입력해야 한다.
- **탐색**: Ctrl-f를 사용하여 문자를 찾을 수 있다. 찾고자 하는 문자를 입력하면 처음 찾아진 문자가 하이라이트 되며, 화살표 키를 사용하여 다음 또는 이전으로 이동할 수 있다. Enter를 사용하면 하이라이트 된 문자로 커서가 이동하고 수정 가능하다. Esc키를 사용하면 즉시 탐색을 종료하고 커서는 탐색 이전으로 이동한다.
- **나가기**: Ctrl-q를 사용하여 에디터를 종료할 수 있다. 저장하지 않은 상태로 나가기를 원할 경우 Ctrl-q를 2번 눌러야 한다.

3. 설계

3.1 자료구조



텍스트 저장을 위해 이중 연결 리스트를 활용하며 세 개의 구조체로 구성된다.

텍스트 편집기 특성상 문자의 삽입 삭제가 빈번히 일어나기 때문에 배열이 아닌 연결 리스트를 사용하였고, 커서의 이동을 더욱 쉽게 제어하기 위해 이중으로 연결된 리스트를 사용하였다.

Editor 구조체는 현재 화면의 커서가 가리키는 노드의 행과 열의 주소를 가지고 있고 커서가 이동하면 Editor 또한 같이 이동한다. 화면의 커서의 위치는 문자 노드 사이에 있다고 가정하며 커서의 위치가 한 행의 맨 앞에 있을 경우 Editor의 노드 열 주소는 NULL값을 가진다.

RowNode 구조체는 한 행의 첫 문자의 주소를 가지고 있고 노드들끼리는 이중 연결 리스트로 연결된다.

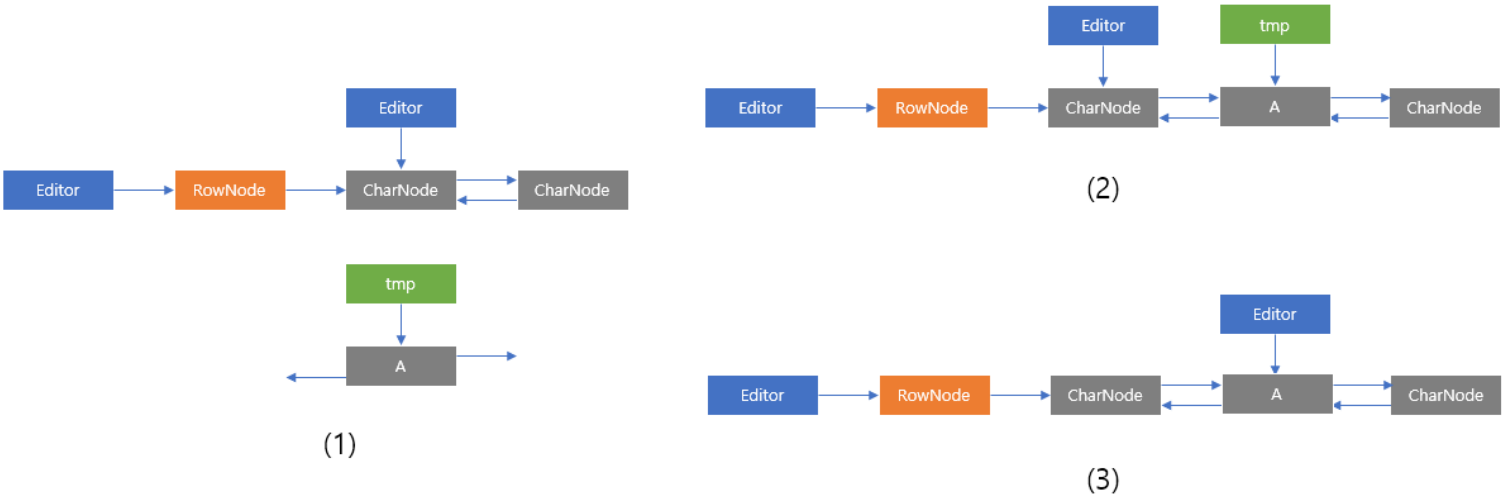
CharNode 구조체는 입력된 문자를 저장하고 노드들끼리는 이중 연결 리스트로 연결된다.

3.2 구조체

```
typedef struct CharNode
{
    char c;                 //입력된 문자를 저장하는 변수
    struct CharNode *prev; //이전 문자 노드를 가리키는 포인터
    struct CharNode *next; //다음 문자 노드를 가리키는 포인터
} CharNode;

typedef struct RowNode
{
```

```
    struct CharNode *str; //문자열의 첫 문자 노드를 가리키는 포인터
    struct RowNode *prev; //이전 행 노드를 가리키는 포인터
    struct RowNode *next; //다음 행 노드를 가리키는 포인터
    int num_char;         //행에 저장된 문자의 개수
} RowNode;

typedef struct Editor
{
    struct RowNode *print_head; //화면의 출력범위를 지정하기 위한 포인터
    struct RowNode *node_row;   //커서가 가리키는 행 노드의 포인터
    struct CharNode *node_col;  //커서가 가리키는 문자 노드의 포인터
    int node_y;                 //데이터 상의 포인터의 y 좌표값
    int node_x;                 //데이터 상의 포인터의 x 좌표값
    int num_lines;              //전체 라인 수를 저장하는 변수
    int cursor_row;             //화면상의 커서의 현재 y 좌표값을 저장하는 변수
    int cursor_col;             //화면상의 커서의 현재 x 좌표값을 저장하는 변수
    int right_move_distance;    //커서가 오른쪽으로 이동할 거리를 저장하는 변수
    int console_height;         //실행된 콘솔창의 높이를 저장하는 변수
    int console_width;          //실행된 콘솔창의 넓이를 저장하는 변수
    int is_saved;               //에디터에 저장상태를 나타내는 변수
    char *filename;             //현재 열려있는 파일의 이름을 저장하는 변수
} Editor;
```

3.3 문자 입력



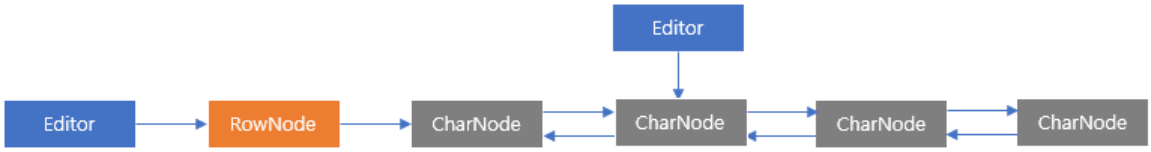
문자가 입력되면 malloc함수를 통해 새 문자 노드를 생성하고 현재 커서가 위치한 노드의 바로 다음에 문자를 삽입하고 연결한다. 삽입된 문자로 node_col 포인터가 이동한다.

행이 비어 있을 경우, 커서가 맨 앞에 위치할 경우, 문자들 사이에 위치할 경우, 맨 뒤에 위치할 경우로 나뉜다.

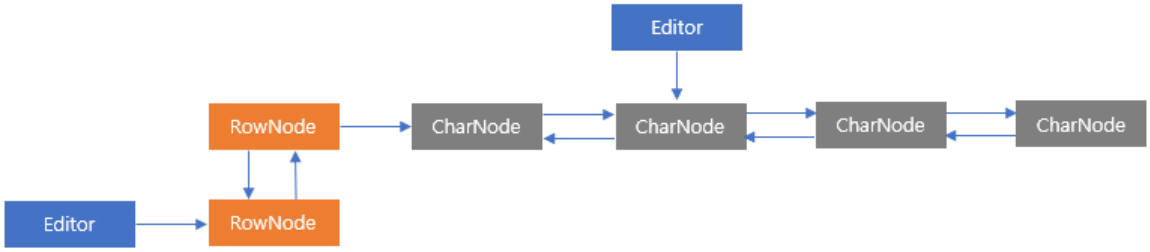
다음은 InsertChar함수에 대해 구현한 코드이다.

```
void InsertChar(Editor *editor, char c)
{
    CharNode *temp = GetNewCharNode(c);//새 문자 노드를 생성

    if(editor -> node_row -> str == NULL)//행이 비어있을 경우
    {
        editor -> node_row -> str = temp;
    }
    else if(editor -> node_col == NULL)//커서가 맨 앞에 있을 경우
    {
        temp -> next = editor -> node_row -> str;
        editor -> node_row -> str -> prev = temp;
        editor -> node_row -> str = temp;
    }
    else if(editor -> node_col -> next == NULL)//커서가 맨 뒤에 있을 경우
    {
        temp -> prev = editor -> node_col;
        editor -> node_col -> next = temp;
    }
    else//커서가 중간에 있을 경우
    {
        temp -> next = editor -> node_col -> next;
        temp -> prev = editor -> node_col;
        editor -> node_col -> next -> prev = temp;
        editor -> node_col -> next = temp;
    }

    editor -> node_col = temp;          //포인터 이동
    editor -> node_x++;                 //포인터 좌표값 변경
    editor -> cursor_col++;             //커서 좌표값 변경
    editor -> node_row -> num_char++;   //문자 수 증가
    editor -> right_move_distance = editor -> cursor_col;
    editor -> is_saved = 0;             //저장상태 변경
}
```

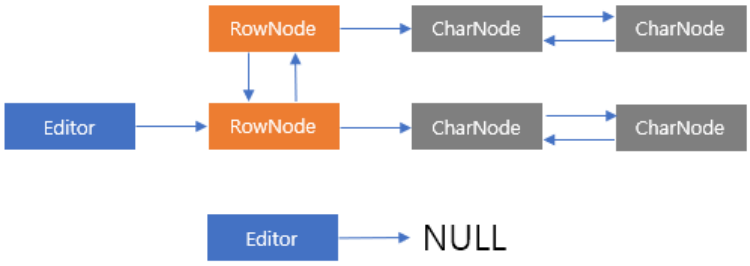
3.4 Enter 입력



(1)



(2)



(3)

Enter 입력 시 malloc함수를 통해 행 노드를 생성하여 현재 행 다음에 삽입한 후 삽입된 행으로 node_row 포인터가 이동하고 현재 커서 오른쪽에 있는 문자들을 다음 행으로 옮겨준다.

이때 커서의 위치는 문자열의 맨 앞에 위치하므로 node_col의 값은 NULL이 된다.

커서가 맨 앞에 위치할 경우, 문자들 사이에 위치할 경우, 맨 뒤에 위치할 경우로 나뉜다.

다음은 InsertRow함수와 SplitLine함수를 구현한 코드이다.

```
void InsertRow(Editor *editor)
{
    RowNode *temp = GetNewRowNode();//새 행 노드를 생성

    if(editor -> node_row -> next == NULL)//마지막 행인 경우
    {
        temp -> prev = editor -> node_row;
        editor -> node_row -> next = temp;
```

```
    }
    else
    {
        temp -> next = editor -> node_row -> next;
        temp -> prev = editor -> node_row;
        editor -> node_row -> next -> prev = temp;
        editor -> node_row -> next = temp;
    }

    editor -> node_row = editor -> node_row -> next;//포인터 이동
    editor -> node_y++;                             //포인터 좌표값 변경
    editor -> cursor_row++;                         //커서 좌표값 변경
    editor -> num_lines++;                          //전체 라인 수 증가
    editor -> is_saved = 0;                         //저장상태 변경
}

void SplitLine(Editor *editor)
{
    int n = editor -> cursor_col;

    if(editor -> node_col == NULL)//커서가 맨 앞에 있을 경우
    {
        editor -> node_row -> str = editor -> node_row -> prev -> str;
        editor -> node_row -> prev -> str = NULL;
    }
    else if(editor -> node_col -> next == NULL)//커서가 맨 뒤에 있을 경우
    {
        editor -> node_col = NULL;
        editor -> node_x = 0;
        editor -> cursor_col = 1;
    }
    else//커서가 중간에 있을 경우
    {
        editor -> node_row -> str = editor -> node_col -> next;
        editor -> node_col -> next -> prev = NULL;
        editor -> node_col -> next = NULL;
        editor -> node_col = NULL;
        editor -> node_x = 0;
        editor -> cursor_col = 1;
    }

    editor -> right_move_distance = editor -> cursor_col;
    //num_char 업데이트
    editor -> node_row -> num_char = editor -> node_row -> prev -> num_char -
    (n - 1);
    editor -> node_row -> prev -> num_char = n - 1;
}
```

3.5 BackSpace 입력(커서의 위치가 맨 앞이 아닐 경우)

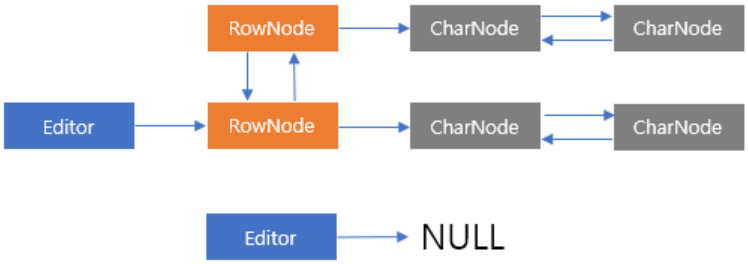
현재 커서의 위치가 문자열의 맨 앞이 아닐 경우 BackSpace 입력 시 단순히 현재 커서가 위치한 노드를 free함수를 통해 삭제한다. 바로 전 노드로 node_col 포인터가 이동한다.

첫번째 문자를 삭제할 경우, 커서가 문자들 사이에 위치할 경우, 맨 뒤에 위치할 경우로 나뉜다.

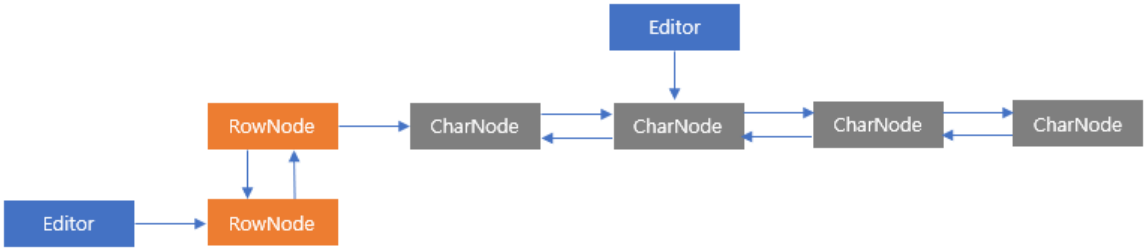
다음은 DeleteChar함수에 대해 구현한 코드이다.

```
void DeleteChar(Editor *editor)
{
    CharNode *temp = editor -> node_col;//커서가 가리키고 있는 문자 노드의 주소

    if(editor -> node_col == NULL)//커서가 맨 앞에 있을 경우
        return;
    else if(editor -> node_col == editor -> node_row -> str)//첫번째 문자를
삭제할 경우
    {
        if(editor -> node_col -> next == NULL)
            editor -> node_row -> str = NULL;
        else
        {
            editor -> node_row -> str = temp -> next;
            temp -> next -> prev = NULL;
        }
        editor -> node_col = NULL;//포인터 이동
    }
    else if(editor -> node_col -> next == NULL)//커서가 맨 뒤에 있을 경우
    {
        editor -> node_col = editor -> node_col -> prev;//포인터 이동
        temp -> prev -> next = NULL;
    }
    else//커서가 중간에 있을 경우
    {
        editor -> node_col = editor -> node_col -> prev;//포인터 이동
        temp -> next -> prev = temp -> prev;
        temp -> prev -> next = temp -> next;
    }

    free(temp);
    editor -> node_x--;              //포인터 좌표값 변경
    editor -> cursor_col--;          //커서 좌표값 변경
    editor -> node_row -> num_char--; //문자 수 감소
    editor -> right_move_distance = editor -> cursor_col;
    editor -> is_saved = 0;          //저장상태 변경
}
```

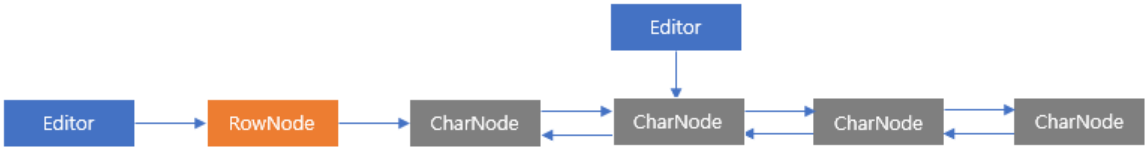
3.6 BackSpace 입력(커서가 위치가 맨 앞에 있을 경우)



(1)



(2)



(3)

커서의 위치가 문자열의 맨 앞에 있을 경우 BackSpace 입력 시 node_col 포인터는 이전 행의 맨 마지막으로 이동 후 현재 행의 문자열을 이전 행의 문자열 다음으로 옮겨주고 행 노드를 삭제하고 node_row 포인터가 이동한다.

현재 행과 이전 행의 문자열이 있을 때와 없을 때로 나뉜다.

다음은 MergeLineg함수와 DeletRow함수를 구현한 코드이다.

```
void MergeLine(Editor *editor)
{
    if(editor -> node_row -> prev == NULL)//현재 행이 첫번째 행인 경우
        return;
    else if(editor -> node_row -> prev -> str == NULL)//이전 행이 비어있는 경우
    {
        editor -> node_col = NULL;
        editor -> node_x = 0;
        editor -> cursor_col = 1;
```

```
        if(editor -> node_row -> str != NULL)//현재 행이 비어있는 경우
            editor -> node_row -> prev -> str = editor -> node_row -> str;
    }
    else
    {
        editor -> node_col = editor -> node_row -> prev -> str;
        editor -> node_x = 1;
        editor -> cursor_col = 2;
        while(editor -> node_col -> next != NULL)
        {
            editor -> node_col = editor -> node_col -> next;
            editor -> node_x++;
            editor -> cursor_col++;
        }

        if(editor -> node_row -> str != NULL)//현재 행이 비어있는 경우
        {
            editor -> node_col -> next = editor -> node_row -> str;
            editor -> node_col -> next -> prev = editor -> node_col;
            editor -> node_row -> str = NULL;
        }
    }

    editor -> right_move_distance = editor -> cursor_col;
    //num_char 업데이트
    editor -> node_row -> prev -> num_char += editor -> node_row -> num_char;
}

void DeleteRow(Editor *editor)
{
    RowNode *temp = editor -> node_row;//커서가 가리키고 있는 행 노드의 주소

    if(editor -> node_row -> prev == NULL)//첫번째 행인 경우
        return;
    else if(editor -> node_row -> next == NULL)//마지막 행인 경우
    {
        editor -> node_row = editor -> node_row -> prev;//포인터 이동
        temp -> prev -> next = NULL;
    }
    else
    {
        editor -> node_row = editor -> node_row -> prev;//포인터 이동
        temp -> next -> prev = temp -> prev;
        temp -> prev -> next = temp -> next;
    }

    free(temp);
```

```
    editor -> node_y--;     //포인터 좌표값 변경
    editor -> cursor_row--; //커서 좌표값 변경
    editor -> num_lines--;  //전체 라인 수 감소
    editor -> is_saved = 0; //저장상태 변경
}
```

3.7 방향키 입력

3.7.1 왼쪽 방향키

왼쪽 방향키를 입력하면 왼쪽으로 포인터와 커서가 한 칸 이동한다.

커서가 맨 앞에 있을 때 왼쪽 방향키를 입력할 경우 포인터와 커서는 전 행의 맨 뒤로 이동한다.

다음은 MovePointerLeft함수에 대한 구현이다.

```
void MovePointerLeft(Editor *editor)
{
    if(editor -> node_col == NULL)//커서가 맨 앞일 경우
    {
        if(editor -> node_row -> prev == NULL)//현재 위치가 첫번째 행일 경우
            return;
        else if(editor -> node_row -> prev -> str == NULL)//전 행이 비었을 경우
        {
            editor -> node_row = editor -> node_row -> prev;
            editor -> node_y--;
            editor -> cursor_row--;
        }
        else
        {
            /*전 행의 맨 끝으로 이동*/

            editor -> node_row = editor -> node_row -> prev;
            editor -> node_y--;
            editor -> cursor_row--;
            editor -> node_col = editor -> node_row -> str;
            editor -> node_x = 1;
            editor -> cursor_col = 2;

            while(editor -> node_col -> next != NULL
            {
                editor -> node_col = editor -> node_col -> next;
                editor -> node_x++;
```

```
                editor -> cursor_col++;
            }
        }
    }
    else
    {
        editor -> node_x--;
        editor -> cursor_col--;
        editor -> node_col = editor -> node_col -> prev;
    }

    editor -> right_move_distance = editor -> cursor_col;
}
```

3.7.2 오른쪽 방향키

오른쪽 방향키를 입력하면 오른쪽으로 포인터와 커서가 한 칸 이동한다.

커서가 문자열의 맨 뒤에 있을 때 오른쪽 방향키를 입력할 경우 포인터와 커서는 전 행의 맨 앞으로 이동한다.

다음은 MovePointerRight함수에 대한 구현이다.

```
void MovePointerRight(Editor *editor)
{
    if(editor -> node_row -> str != NULL && editor -> node_col ==
NULL)//커서가 맨 앞일 경우
    {
        editor -> node_x++;
        editor -> cursor_col++;
        editor -> node_col = editor -> node_row -> str;
    }
    else if(editor -> node_row -> str == NULL || editor -> node_col -> next ==
NULL)//행이 비었거나 커서가 맨 뒤에 있을 경우
    {
        if(editor -> node_row -> next == NULL)//마지막 행일 경우
            return;
        else
        {
            /*다음 행의 맨 앞으로 이동*/
            editor -> node_row = editor -> node_row -> next;
            editor -> node_y++;
            editor -> cursor_row++;
            editor -> node_col = NULL;
            editor -> node_x = 0;
            editor -> cursor_col = 1;
        }
```

```
    }
    else
    {
        editor -> node_x++;
        editor -> cursor_col++;
        editor -> node_col = editor -> node_col -> next;
    }

    editor -> right_move_distance = editor -> cursor_col;
}
```

3.7.3 위쪽 방향키

위쪽 방향키를 입력하면 전 행의 맨 앞으로 이동 후 Editor.right_move_distance의 크기만큼 오른쪽으로 이동한다. 만약 문자열이 Editor.right_move_distance의 크기보다 작을 경우 포인터와 커서는 문자열의 맨 마지막에 위치할 것이다.

Editor.right_move_distance는 문자, Enter, Backspace, 왼쪽 방향키, 오른쪽 방향키, Home, End를 누를 때마다 업데이트 해준다.

다음은 MovePorintUp함수에 대한 구현이다.

```
void MovePointerUp(Editor *editor)
{
    if(editor -> node_row -> prev == NULL)//전 행이 없을 경우
        return;

    /*전 행의 맨 앞으로 이동*/
    editor -> node_row = editor -> node_row -> prev;
    editor -> node_y--;
    editor -> cursor_row--;

    /*right_move_distance 만큼 오른쪽으로 이동*/
    if(editor -> node_row -> str == NULL || editor -> right_move_distance ==
1)
    {
        editor -> node_col = NULL;
        editor -> node_x = 0;
        editor -> cursor_col = 1;
    }
    else
    {
        editor -> node_col = editor -> node_row -> str;
        editor -> node_x = 1;
        editor -> cursor_col = 2;
```

```
        while(editor -> cursor_col <= editor -> right_move_distance - 1)
        {
            if(editor -> node_col -> next == NULL)
                break;
            editor -> node_col = editor -> node_col -> next;
            editor -> node_x++;
            editor -> cursor_col++;
        }
    }
}
```

3.7.4 아래쪽 방향키

아래쪽 방향키를 입력하면 다음 행의 맨 앞으로 이동 후 Editor.right_move_distance의 크기만큼 오른쪽으로 이동한다. 만약 문자열이 Editor.right_move_distance의 크기보다 작을 경우 포인터와 커서는 문자열의 맨 마지막에 위치할 것이다.

Editor.right_move_distance는 문자, Enter, Backspace, 왼쪽 방향키, 오른쪽 방향키, Home, End를 누를 때마다 업데이트 해준다.

다음은 MovePorintDown함수에 대한 구현이다.

```
void MovePointerDown(Editor *editor)
{
    if(editor -> node_row -> next == NULL)
        return;

    /*다음 행의 맨 앞으로 이동*/
    editor -> node_row = editor -> node_row -> next;
    editor -> node_y++;
    editor -> cursor_row++;

    /*right_move_distance 만큼 오른쪽으로 이동*/
    if(editor -> node_row -> str == NULL || editor -> right_move_distance ==
1)
    {
        editor -> node_col = NULL;
        editor -> node_x = 0;
        editor -> cursor_col = 1;
    }
    else
    {
        editor -> node_col = editor -> node_row -> str;
        editor -> node_x = 1;
        editor -> cursor_col = 2;
```

```
        while(editor -> cursor_col <= editor -> right_move_distance - 1)
        {
            if(editor -> node_col -> next == NULL)
                break;
            editor -> node_col = editor -> node_col -> next;
            editor -> node_x++;
            editor -> cursor_col++;
        }
    }
}
```

3.8 특수 이동키 입력

3.8.1 Home

Home키를 입력하면 커서가 현재 행의 맨 앞으로 이동한다.

다음은 MovePointerHome함수에 대한 구현이다.

```
void MovePointerHome(Editor *editor)
{
    editor -> node_col = NULL;
    editor -> node_x = 0;
    editor -> cursor_col = 1;
    editor -> right_move_distance = editor -> cursor_col;
}
```

3.8.2 End

End키를 입력하면 커서가 현재 행의 맨 뒤로 반복문을 통해 한 칸씩 이동한다.

다음은 MovePointerEnd함수에 대한 구현이다.

```
void MovePointerEnd(Editor *editor)
{
    if(editor -> node_row -> str == NULL)
        return;

    if(editor -> node_col == NULL)
    {
        editor -> node_col = editor -> node_row -> str;
        editor -> node_x = 1;
        editor -> cursor_col = 2;
```

```
    }

    while(editor -> node_col -> next != NULL)
    {
        editor -> node_col = editor -> node_col -> next;
        editor -> node_x++;
        editor -> cursor_col++;
    }

    editor -> right_move_distance = editor -> cursor_col;
}
```

3.8.3 PgUp

PgUp키를 입력하면 우선적으로 MovePointerUp함수를 이용해 현재 화면 상의 첫번째 행으로 이동한다. 만약 현재 위치가 화면 상의 첫번째 행에 있다면 입력 시 일정범위 만큼 Editor.print_head포인터가 이동해 페이지를 이동한다.

다음은 MovePointerPgUp함수에 대한 구현이다.

```
void MovePointerPgUp(Editor *editor)
{
    if(editor -> cursor_row == 1)
    {
        /*일정 범위만큼 print_head 포인터 이동*/
        int i;
        for(i = 0; i < (editor -> console_height - 4); i++)
        {
            MovePointerUp(editor);
            if(editor -> cursor_row < 1 && editor -> print_head -> prev !=
NULL)
            {
                editor -> print_head = editor -> print_head -> prev;
                editor -> cursor_row++;
            }
        }
        ClearText(editor, 1);
        PrintText(editor, editor -> print_head, 1);
    }
    else
    {
        /*화면상의 첫번째 행으로 이동*/
        while(editor -> cursor_row != 1)
            MovePointerUp(editor);
    }
}
```

3.8.4 PgDn

PgDn키를 입력하면 우선적으로 MovePointerDown함수를 이용해 현재 화면 상의 마지막 행으로 이동한다. 만약 현재 위치가 화면 상의 마지막 행에 있다면 입력 시 일정범위 만큼 Editor.print_head포인터가 이동해 페이지를 이동한다.

다음은 MovePointerPgDn함수에 대한 구현이다.

```
void MovePointerPgDn(Editor *editor)
{
    if(editor -> cursor_row == editor -> console_height - 2)
    {
        /*일정 범위만큼 print_head 포인터 이동*/
        int i;
        for(i = 0; i < (editor -> console_height - 4); i++)
        {
            MovePointerDown(editor);
            if(editor -> cursor_row > editor -> console_height - 2)
            {
                editor -> print_head = editor -> print_head -> next;
                editor -> cursor_row--;
            }
        }

        ClearText(editor, 1);
        PrintText(editor, editor -> print_head, 1);
    }
    else
    {
        /*화면상의 마지막 행으로 이동*/
        while(editor -> cursor_row != editor -> console_height - 2)
        {
            if(editor -> node_row -> next == NULL)
                break;
            MovePointerDown(editor);
        }
    }
}
```

3.9 실행 및 기존 파일 열기

프로그램 실행 시 메인 함수에서 명령행 인자 argc, argv를 받아온다.

추가 명령행 인자가 없을 경우 파일은 생성하지 않고 프로그램만 실행하고 나중에 저장을 할 때 파일을 생성한다.

argc의 값이 2보다 클 경우 fputs 함수를 통해 오류를 출력하고 exit함수를 통해 프로그램을 종료한다.

argc의 값이 2일 경우 먼저 fopen함수를 통해 "r"모드(읽기모드)로 파일을 연다. 만약 파일이 존재하지 않을 경우 오류를 출력하고 프로그램을 종료한다. 정상적으로 파일이 열렸다면 파일의 문자를 하나씩 받아와서 CharNode에 저장한다.

다음은 Open_File함수에 대한 구현이다.

```
void Open_File(Editor *editor, int argc, char *argv)
{
    if (argc > 2)
    {
        fputs("(HELP)\n\nRun:                vite\nOpen existing file: vite
<filename>\n", stderr);
        exit(1);
    }
    else if(argc == 1)
        return;

    FILE *fp = fopen(argv, "r");

    if(fp == NULL)//파일이 존재하지 않는다면
    {
        fputs("File does not exist.\n", stderr);
        exit(1);
    }

    editor -> filename = argv;//현재 열린 파일의 이름을 저장

    char fpc;
    while((fpc = fgetc(fp)) != EOF)//파일의 문자들을 리스트에 저장
    {
        if(fpc == 10)
        {
            InsertRow(editor);
            SplitLine(editor);
            if(editor -> cursor_row > editor -> console_height - 2)
            {
                editor -> print_head = editor -> print_head -> next;
                editor -> cursor_row--;
            }
        }
        else
        {
            if(editor -> cursor_col >= editor -> console_width - 1)
            {
```

```
                InsertRow(editor);
                SplitLine(editor);
                if(editor -> cursor_row > editor -> console_height - 2)
                {
                    editor -> print_head = editor -> print_head -> next;
                    editor -> cursor_row--;
                }
            }
            InsertChar(editor, fpc);
        }
    }

    fclose(fp);
}
```

3.10 Ctrl-s 입력(저장하기)

Ctrl-s를 입력했을 때 현재 저장할 파일이 정해지지 않았으면 우선적으로 화면의 맨 밑에 줄에서 파일의 이름을 입력 받고 파일을 생성한 후 저장한다.

입력은 새로운 에디터 구조체를 선언한 후 오리지널 에디터가 입력 받는 형식과 똑같이 입력 받는다. 다만, 새로운 에디터 구조체는 한 행까지만 입력을 받을 수 있다.

파일 이름에는 '/'문자가 들어가지 않게 애초에 입력을 받지 않게 제한을 했다. Esc키 입력을 통해서 저장하기를 취소하고 바로 전 상태로 커서가 이동할 수 있다. Enter키 입력을 통해서 입력 받기를 종료하고 입력 받은 문자열을 배열로 복사하여 Editor.filename에 저장한다. 만약 아무 문자도 입력하지 않고 Enter키를 입력하면 Esc키와 마찬가지로 저장하기를 취소하고 바로 전 상태로 돌아간다.

저장할 파일의 이름이 정해졌으면 fopen함수의 "w"모드를 통해 파일을 생성하거나 쓰기 모드로 파일을 연다. 그 후 에디터에 저장된 문자들을 순차적으로 하나씩 파일에 저장한다.

다음은 SaveFile함수의 파일 내용 저장 부분에 대한 구현이다.

```
    /*파일 내용 저장*/
    FILE *fp = fopen(editor -> filename, "w");

    RowNode *temp1 = editor -> print_head;
    CharNode *temp2;

    while (temp1 -> prev != NULL)
        temp1 = temp1 -> prev;

    while(temp1 != NULL)
```

```
    {
        temp2 = temp1 -> str;
        while(temp2 != NULL)
        {
            fputc(temp2 -> c, fp);
            temp2 = temp2 -> next;
        }
        fputc('\n', fp);
        temp1 = temp1 -> next;
    }

    editor -> is_saved = 1;

    fclose(fp);

    PrintStatusBar(editor);
    PrintMessageBar(editor, "File has saved");
    MoveCursor(editor);
}
```

3.11 Ctrl-q입력(나가기)

Ctrl-q를 입력했을 때 Editor.is_saved변수의 값이 1인 상태(저장된 상태)라면 프로그램을 바로 종료한다.

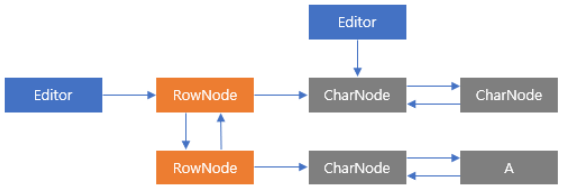
Editor.is_saved변수의 값이 0인 상태(저장되지 않은 상태)라면 경고 메시지를 출력하고 키를 한 번 입력을 받는다. Ctrl-q를 입력하면 저장하지 않은 채로 프로그램이 종료되고, Ctrl-s를 입력하면 SaveFile함수가 실행이 된다. 그 외의 문자가 입력될 경우 에디터는 전의 상태로 돌아간다.

다음은 ExitEditor함수에 대한 구현이다.

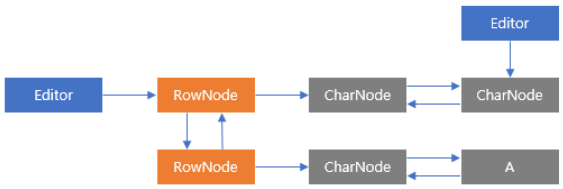
```
void ExitEditor(Editor *editor)
{
    if(editor -> is_saved == 0)//저장되지 않은 상태
    {
        PrintMessageBar(editor, "!!WARNING!! File has not been saved. Ctrl-q
to quit without saving.");

        int c = ReadKey();

        if(c == CTRL_S)
        {
            SaveFile(editor);
            return;
```

```
        }
        else if(c == CTRL_Q)
        {
            system(CLEAR);
            exit(0);
        }
        else
        {
            /*에디터가 전의 상태로 돌아간다.*/
            PrintMessageBar(editor, "Help: Ctrl-s = save | Ctrl-q = quit |
Ctrl-f = find");
            MoveCursor(editor);
            return;
        }
    }
    else
    {
        system(CLEAR);
        exit(0);
    }
}
```

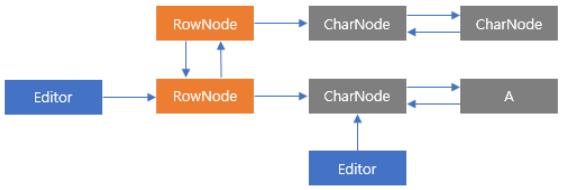
3.12 Ctrl-f입력(탐색)



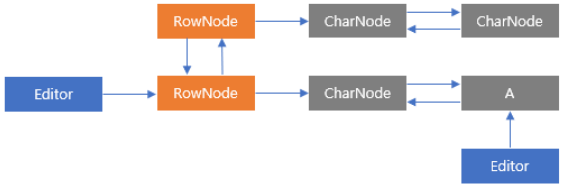
(1)



(2)



(3)



(4)

Ctrl-f를 입력하면 Ctrl-s와 마찬가지로 맨 마지막 줄에서 탐색할 문자열을 입력 받는다.

문자열은 한 행만 입력 받을 수 있으며 Esc를 입력하면 문자 입력을 종료하고 에디터는 전의 상태로 돌아가며 Enter를 입력하면 탐색을 시작한다. Enter입력 시 아무 문자도 입력하지 않으면 에디터는 전의 상태로 돌아간다.

탐색을 시작하면 node_col, node_row포인터가 첫 행 첫 문자 노드로 이동하여 MovePointerRight함수를 이용하여 텍스트 전체를 검사한다. 만약 이동하다가 탐색할 문자열의 맨 앞의 문자와 같은 문자가 나올 경우 IsEquals함수를 통해 두 문자열이 서로 같은 지를 검사하고 같은 경우 현재의 에디터 상태를 에디터 상태 배열 editor_state[]에 저장한다.

IsEquals함수는 두 문자열의 문자를 하나씩 서로 맞는 지 검사하는 방식이다.

editor_state[]배열의 크기는 초기값 100이며 탐색한 결과의 수가 100의 배수가 될 때마다 realloc함수를 통해 크기를 늘려준다.

탐색이 끝나고 탐색 결과가 없을 경우 메시지를 출력하고 에디터는 전의 상태로 돌아간다. 결과값이 있을 경우 editor_state[0]의 상태일 때로 이동해 문자열을 반전색으로 출력한다.

그리고 다시 키 입력 별로 처리를 한다. 방향키를 입력할 경우 에디터의 다음 상태나 전의 상태로 이동해 문자열을 찾을 수 있으며 현재 에디터 상태에서 Enter를 입력할 경우 문자열을 수정할 수 있다. Esc를 입력할 경우 에디터는 탐색 전의 에디터 상태로 돌아간다.

다음은 Search함수의 문자열 탐색 부분과 IsEquals함수에 대한 구현이다.

```
int count = 0;
Editor *editor_state = (Editor *)malloc(sizeof(Editor) * 100);//에디터의
현재상태를 저장하기 위한 배열
while(1)
{
    MovePointerRight(editor);
    if(editor -> cursor_row > (editor -> console_height / 2))
    {
        editor -> print_head = editor -> print_head -> next;
        editor -> cursor_row--;
    }

    if(editor -> node_col != NULL)
    {
        if(editor -> node_col -> c == temp_editor.node_row -> str -> c)
        {
            if(IsEquals(editor -> node_col, temp_editor.node_row -> str))
            {
                editor_state[count] = *editor;
                count++;

                if(count % 100 == 0)
                    editor_state = (Editor *)realloc(editor_state,
sizeof(Editor) * (count + 100));
            }
        }
    }

    if(editor -> node_row -> next == NULL)
    {
        if(editor -> node_col == NULL)
        {
            if(editor -> node_row -> str == NULL)
                break;
        }
        else
        {
            if(editor -> node_col -> next == NULL)
                break;
        }
    }
}
```

```
int IsEquals(CharNode *node_col, CharNode *str)
{
    while(str != NULL)
    {
        if(node_col == NULL || (str -> c != node_col -> c))
            return 0;

        str = str -> next;
        node_col = node_col -> next;
    }

    return 1;
}
```

3.13 커서 이동

커서 이동을 위해 안시(ANSI) 이스케이프 시퀀스를 이용한다.

다음은 안시 이스케이프 시퀀스를 이용한 커서 이동 함수 MoveCursor함수에 대한 구현이다.

```
void MoveCursor(Editor *editor)
{
    printf("\033[%d;%dH", editor -> cursor_row, editor -> cursor_col);
}
```

3.14 상태 바

상태 바 구현을 위해 snprintf함수를 이용한다. 먼저 상태 바에 출력될 문자의 수를 계산하여 remainingSpace를 계산하여 snprintf함수를 통해 미리 출력할 문자열을 statusbar변수에 넣어 주고, 반전색으로 출력한다.

다음은 PrintStausBar함수에 대한 구현이다.

```
void PrintStatusBar(Editor *editor)
{
    char *statusbar = (char *)malloc(editor->console_width + 1);
    int remainingSpace;

    if(editor -> filename == NULL)
    {
        remainingSpace = editor->console_width - (strlen("[No Name] -  linesno
ft | /") + CountDigits(editor->num_lines) + CountDigits(editor->node_y) +
CountDigits(editor->node_x));
```

```
        snprintf(statusbar, editor->console_width + 1, "[No Name] - %d
lines%*sno ft | %d/%d", editor->num_lines, remainingSpace, "", editor->node_y,
editor->node_x);
    }
    else
    {
        remainingSpace = editor->console_width - (strlen(editor->filename) +
strlen("[] -  linesno ft | /") + CountDigits(editor->num_lines) +
CountDigits(editor->node_y) + CountDigits(editor->node_x));
        snprintf(statusbar, editor->console_width + 1, "[%s] - %d lines%*sno
ft | %d/%d", editor->filename, editor->num_lines, remainingSpace, "", editor-
>node_y, editor->node_x);
    }

    printf("\033[%d;1H", editor->console_height - 1);//마지막에서 두번째 줄로
이동
    printf("\033[7m");//반전색
    printf("%s", statusbar);
    printf("\033[0m");//스타일 초기화
    free(statusbar);
}
```

3.15 메시지 바

메시지 바 출력을 위해 매개변수로 문자열을 받고 커서가 맨 마지막 줄로 이동하여 현제 메시지를 지우고 문자열을 출력한다.

다음은 PrintMessageBar함수에 대한 구현이다.

```
void PrintMessageBar(Editor *editor, char *string)
{
    printf("\033[%d;1H", editor -> console_height);
    printf("\033[K");//현재 커서위치부터 한 행 지우기
    printf("%s", string);
}
```

3.16 터미널 설정

키 입력을 위해 윈도우에서는 getch함수를 리눅스 및 맥 운영체제에서는 getchar함수를 이용한다. getchar함수를 getch함수처럼 엔터를 입력하지 않고 키 입력을 받자마자 프로그램을 동작하기 위해 termios.h를 이용하여 논캐노니컬 모드로 터미널 설정을 변경한다.

다음은 터미널 설정을 변경하는 코드이다.

```
#ifndef _WIN32
struct termios newt;
tcgetattr(STDIN_FILENO, &newt);

newt.c_lflag &= ~(ICANON | ECHO | IEXTEN | ISIG);
newt.c_iflag &= ~(IXON | IXOFF);

tcsetattr(STDIN_FILENO, TCSANOW, &newt);
#endif
```

3.17 컨디셔널 컴파일

운영체제별로 프로그램이 동작하기 위해 운영체제 매크로 변수와 전처리문을 이용한다.

```
#ifdef _WIN32//윈도우 64 비트, 32 비트 운영체제 매크로
    /*코드*/
#elif __APPLE__//맥 os 매크로
    /*코드*/
#else// 그 외 운영체제(ex.리눅스)
    /*코드*/
#endif
```

4. 프로그램 실행 과정

프로그램 실행 -> 에디터 구조체 선언 -> 구조체 초기화 -> 초기 화면 생성 -> 키 입력 별 프로세스 순으로 동작한다.

```
#ifndef _WIN32
struct termios newt;
tcgetattr(STDIN_FILENO, &newt);

newt.c_lflag &= ~(ICANON | ECHO | IEXTEN | ISIG);
newt.c_iflag &= ~(IXON | IXOFF);

tcsetattr(STDIN_FILENO, TCSANOW, &newt);
#endif
```

5. 문제점

현재 키 입력 방식으로는 리눅스, 맥 운영체제에서 Esc를 입력 받을 경우 바로 함수값을 리턴하지 못한다.

해결방법을 아직 찾지 못해 임시로 Esc키를 입력하는 부분을 Ctrl-q로 변경하였다.

문자열의 길이가 화면의 가로 크기보다 커지는 경우 텍스트를 제대로 출력할 수 없다.

해결방법으로 추후에 화면이 횡스크롤이 가능하게 구현하거나 한 행에 무한히 긴 텍스트를 입력해도 텍스트 전체를 출력할 수 있게 구현할 예정이다. 현재는 임시로 행의 맨 끝에서 문자 입력으로 인해 현재 행의 문자열의 크기가 일정 크기를 넘을 경우 자동으로 Enter처리를 하게 구현하였으며, 커서가 맨 뒤에 있지 않은 경우 문자 입력이 안 되게 제한을 했다. 또한, BackSpace로 인해 라인이 합쳐지는 경우 합쳐진 문자열의 크기가 일정 크기를 넘으면 행을 추가하여 문자열 일부분을 추가한 행으로 넘겨주었다