과 제 2 차 제 안 서

과제명: C언어를 사용한 Visual Text Editor 구현

과제 개발: 20213137 최영재

1. 개발목표

텍스트 에디터는 자주 사용되는 도구로, 코드 작성, 문서 편집 등에 필수적이다.

본 과제의 목표는 <고급 자료구조> 시간에 배운 자료구조와 C언어를 활용해 vi 편집기와 유사한 기능을 가진 텍스트 에디터를 구현하여, 텍스트 에디터의 자료구조 활용 방법 및 터미널 기반 인터페이스 개발에 대한 이해를 심화하는 것에 있다.

2. 프로그램 설명

모든 프로그램은 C언어로 작성되어야 하며, 리눅스, 윈도우 OS, 맥 OS 환경에서 컴파일되고 실행되어야 한다. 또한 Makefile을 통해 컴파일 되어야 한다.

2.1 컴파일

명령어 행에서 다음 명령어를 입력한다.

```
make
```

각 운영체제에서 프로젝트를 쉽게 컴파일할 수 있도록 Makefile을 사용하여 컴파일러 옵션과 필요한 라이브러리들을 운영체제에 맞춰 설정하여, 자동으로 빌드할 수 있도록 한다.

2.2 실행

컴파일 후에 명령어 행에서 다음 명령어를 입력한다.

```
./viva
```

프로그램을 컴파일한 후, 리눅스와 맥OS에서는 실행 파일에 대해 실행 권한을 부여한 뒤,

./viva 명령어로 실행할 수 있다.

이를 위해 Makefile에 실행 권한을 자동으로 부여하는 규칙을 추가할 것이다.

3. 구현 내용 및 사용법

사용자는 새 파일 또는 기존에 있는 파일을 열어 내용을 편집할 수 있으며, 수정된 내용을 저장할 수 있다. 또한 텍스트 편집기의 기능을 재현하기 위해 연결 리스트를 사용하였다.

3.1 새로운 파일 생성

새로운 파일을 생성하려고 하는 경우, 명령어 행에서 다음과 같이 실행한다.

```
./viva
```

3.2 기존에 있는 파일 열기

기존에 있는 파일을 열 경우, 명령어 행에서 다음과 같이 실행한다.

```
./viva viva.c
```

Makefile은 ARGS 변수를 통해 실행 시 필요한 인자를 전달할 뿐이므로, 프로그램 내부에서는 기존에 설명한 대로 명령줄 인자 처리가 필요하다. 프로그램은 main 함수에서 전달된 파일 이름을 처리해야 한다.

따라서 프로그램 실행 시 인자를 전달하고, 프로그램 내부에서 이를 처리하도록 구현할 것이다.

3.3 구현 내용

3.3.1 UI

예제에 있는 텍스트 에디터는 터미널과 같은 형태를 하고 있으며, 이와 유사하게 구현하려면 외부 라이브러리가 필요하다.

따라서 이번 과제에서는 여러가지 운영체제에서 사용 가능한 ncurses 라이브러리를 사용하여 전체 화면을 관리하고 구현할 수 있도록 한다.

윈도우 개발환경에서는 pdcurses를 사용하여 개발할 수 있다.

3.3.2 자료구조 정의

텍스트 편집기의 기능을 구현하기 위해 연결 리스트를 사용하였다.

노드 구조체 정의 - 텍스트의 한 문자(character) 의미

행, 열 필드 - 텍스트 수정, 탐색, 렌더링, 줄 나누기, 스크롤링 등 기능 구현 시 필요한 위치 정보 제공

```
typedef struct Node {
    char character;             // 저장된 문자
    struct Node *prev;          // 이전 노드에 대한 포인터
    struct Node *next;          // 다음 노드에 대한 포인터
    int row;    // 노드의 행 위치
    int col;    // 노드의 열 위치
} Node;
```

텍스트 버퍼 구조체 정의 - 전체 텍스트 의미, 저장 관련 구조체 정의 - modified

```
typedef struct TextBuffer {
    Node *head;             // 리스트의 첫 번째 노드
    Node *tail;             // 리스트의 마지막 노드
    Node *cursor;           // 현재 커서 위치
    int modified;           // 파일이 수정되었는지를 나타내는 플래그
    char filename[256];   // 파일 이름 저장(256 바이트: 대부분의 운영체제에서 파일 이름 길이 제한)
    // 추가 필드...
} TextBuffer;
```

3.3.3 ncurses 초기화 및 설정

```
#include <ncurses.h>

int main() {
    initscr();                  // ncurses 모드 시작
    noecho();                   // 키보드 입력을 화면에 출력하지 않음
    cbreak();                   // 버퍼링 없이 입력을 즉시 처리
    keypad(stdscr, TRUE);    // 특수 키(화살표 키 등)를 처리하도록 설정
    curs_set(1);                // 커서 보이게 설정

    // 프로그램 내용...

    endwin();                   // ncurses 모드 종료
    return 0;
}
```

3.3.4 상태 바

상태 바는 아래에서 2번째 라인에 위치해야 하며, 반전 색으로 보여야 하고, '파일 이름', '라인 수', '현재 커서가 있는 위치'의 정보를 포함한다.

ncurses의 라이브러리를 활용하여 update_status_bar 함수 구현, 함수에는 상태 바 내용 생성, 상태 바에 내용 출력 등의 내용이 포함되어 있다.

파일 이름: 사용자가 열어둔 파일의 이름은 filename 인자로 전달받는다.

라인 수 계산: 파일 전체 라인 수는 텍스트 버퍼에서 관리하는 데이터에서 얻어내고, 각 줄은 배열이나 리스트로 관리된다.

커서 위치 계산: 커서의 현재 위치는 ncurses 라이브러리의 getyx() 함수를 통해 가져온다.

3.3.5 메시지 바

메시지 바는 가장 아래 라인에 위치해야 하며, 초기에는 '저장', '탐색', '나가기' 방법을 표시한다. 프로그램을 사용할 때, 다른 정보를 출력할 수 있다.

update_message_bar 함수에는 운영체제에 따른 단축키 메시지 설정, 메시지 바에 내용 출력(터미널 너비에 맞춰 출력) 등의 내용을 포함한다.

- Windows/Linux: Ctrl + S, Ctrl + F, Ctrl + Q
- macOS: Command + S, Command + F, Command + Q

C언어 컴파일 시 매크로 활용하여 현재 운영체제 감지:

- Windows: _WIN32 매크로
- macOS: __APPLE__ 매크로
- Linux: __linux__ 매크로

각 '저장', '탐색', '나가기'와 같은 명령 처리는 ASCII 값으로 인식하여 처리된다

.

3.3.6 텍스트 수정

앞서 정의했던, 텍스트 구조체들을 바탕으로 텍스트 수정 기능을 구현한다.

1. 노드 생성 함수

입력: 저장할 문자(character)

동적 할당을 사용하여, 노드 크기만큼 메모리를 할당하고, 입력된 문자를 노드의 character 필드에 저장.

prev와 next 포인터는 초기화하여 현재 연결 상태 없음 표시

출력: 초기화된 노드에 대한 포인터 반환

2. 텍스트 버퍼 초기화 함수

입력: 없음

동적 할당을 사용하여, 텍스트 버퍼 크기만큼 메모리를 할당,

head, tail, cursor 포인터를 NULL로 초기화하여 텍스트 비어 있음 표시

출력: 초기화된 텍스트 버퍼에 대한 포인터 반환

3. 문자 입력 함수

입력: 텍스트 버퍼 포인터, 입력할 문자

커서 위치에 새 문자를 삽입하는데, 입력된 문자가 영어인지 확인하여 아니면 함수 종료.

새로운 노드 생성, 리스트가 비어 있는 경우는 새로운 노드를 head, tail, cursor로 설정, 리스트가 비어 있지 않은 경우는 새로운 노드를 현재 커서 위치에 삽입. prev, next 포인터 적절히 업데이트하여 리스트 연결 유지

출력: 없음

4. 백 스페이스 함수

입력: 텍스트 버퍼 포인터

현재 커서 위치의 문자를 삭제하는데, 커서가 NULL인 경우, 아무 작업도 수행하지 않고 종료. 삭제할 노드를 커서 위치로 설정, 삭제할 노드의 이전 노드와 다음 노드 연결.

삭제할 노드가 head일 경우, head를 다음 노드로 변경, tail일 경우는 tail을 이전 노드로 업데이트. 삭제한 노드의 메모리 해제.

출력: 없음

5. 엔터 함수

입력: 텍스트 버퍼 포인터

현재 커서 위치에 줄 바꿈 추가, ₩n을 가진 새 노드 생성.

리스트가 비어 있을 경우, 새 노드를 head, tail, cursor로 설정, 리스트가 비어 있지 않은 경우, 새로운 줄 노드를 현재 커서의 위치에 삽입. prev, next 포인터 업데이트하여 리스트의 연결 유지.

출력: 없음

나머지: 메인 루프 및 ncurses 통합

3.3.6 이동

다음과 같은 키를 사용하여 빠르게 이동이 가능하다.

- 화살표 키
- 홈 키
- 엔드 키
- 페이지 업
- 페이지 다운

앞서, 정의했던 노드 구조체의 행, 열 필드를 사용하여 이동 기능을 구현할 수 있다.

1. 커서 위치 업데이트 함수

입력: 텍스트 버퍼 포인터

현재 커서의 행과 열 위치를 업데이트한다.

현재 노드를 순회하면서 커서가 있는 노드의 위치를 찾아 cursor_row와 cursor_col을 업데이트한다.

출력: 없음

2. 커서 이동 함수

입력: 텍스트 버퍼 포인터, 이동 방향

커서를 이동한다. 화살표 키의 경우, prev 또는 next 포인터를 사용하여 커서 이동.

홈 키와 엔드 키는 각각 리스트의 시작과 끝으로 커서 이동.

페이지 업 및 페이지 다운 기능은 여러 노드를 건너뛰는 방식으로 구현.

출력: 없음

나머지: 메인 루프에서 이동 처리

3.3.7 저장

파일이 수정되었을 경우 Ctrl + S 등을 사용하여 저장할 수 있다. 새로운 파일을 저장하려고 할 경우에는 파일 이름을 입력해야 한다.

앞서 정의했던 텍스트 버퍼의 modified 필드를 사용하여 저장 기능을 구현할 수 있다.

1. 파일 저장 함수

입력: 텍스트 버퍼 포인터

텍스트 버퍼의 내용을 파일로 저장한다.

파일 이름이 비어 있으면 사용자에게 새 파일 이름을 입력 받는다.

fopen으로 파일을 열고, fputc를 사용해 텍스트 버퍼의 내용을 파일에 기록한다.

파일 저장이 완료되면 수정 플래그를 초기화한다.

출력: 파일이 저장되었다는 메시지 표시

2. Ctrl + S 등 처리 로직 추가(main문)

메인 루프에 Ctrl + S 등의 입력을 처리하는 코드 추가.

수정된 상태인 경우에만 저장 수행, 수정되지 않은 경우, "변경 없음" 메시지 표시.

3. 수정 상태 관리(수정 기능의 문자 입력 함수 수정)

사용자가 텍스트 수정할 때마다 buffer->modified 플래그 1로 설정.

4. 엔터 입력 시 파일 이름 입력 처리 로직 추가

입력: 텍스트 버퍼 포인터

사용자에게 파일 이름을 입력 받아 저장.

출력: 입력 받은 파일 이름 버퍼에 저장.

3.3.8 탐색

탐색은 Ctrl + F 등을 사용하며, 찾고자 하는 문자를 입력하면 처음 찾아진 문자가 하이라이트 되고, 화살표 키를 사용하며 다음이나 이전으로 이동한다.

Enter를 누르면 검색을 종료하거나 탐색 결과에서 수정이 가능하다. Esc를 누르면 포기하고 커서는 탐색 이전으로 돌아간다.

탐색 기능을 위해 다음과 같은 구조체와 변수를 정의한다.

```
typedef struct SearchResult {
    Node *node;        // 찾은 문자가 있는 노드
    struct SearchResult *next;  // 다음 결과 노드
} SearchResult;

typedef struct SearchState {
    SearchResult *head;  // 탐색 결과의 첫 번째 노드
    SearchResult *current; // 현재 선택된 탐색 결과
} SearchState;
```

1. Ctrl + F 키 입력 처리 함수(Search)

사용자가 Ctrl + F 등의 키를 입력했을 때, 검색 모드로 전환되기 위한 방법

입력: 텍스트 버퍼 포인터

검색어 입력: 사용자로부터 검색할 문자열 입력 받기.

검색 실행: 텍스트 버퍼를 순회하며, 입력된 문자열이 포함된 노드를 찾아 SearchResult 리스트에 저장.

하이라이트: 찾은 결과가 있을 경우, 첫 번째 결과를 하이라이트한다.

결과 탐색: 사용자가 방향키와 Enter 키를 사용하여 결과를 탐색할 수 있도록 한다.

출력: 찾은 결과를 하이라이트하고, 결과가 없을 경우 해당 메시지를 출력한다.

2. 하이라이트 함수

입력: 노드 포인터

ncurses의 attron 함수를 이용하여 하이라이트 효과 활성화

mvaddch를 사용해 해당 위치에 있는 문자 출력

attroff를 호출하여 하이라이트 효과 해제.

출력: 하이라이트된 문자를 화면에 표시.

3. 언하이라이트 함수

입력: 노드 포인터

하이라이트된 문자를 원래 색상으로 되돌리는 기능.

mvaddch를 호출하여 하이라이트가 해제된 문자를 원래대로 출력.

출력: 원래 문자 화면에 표시

4. 탐색 문자 수정 함수

입력: SearchState 포인터

현재 선택된 탐색 결과의 문자를 수정하는 기능.

현재 선택된 탐색 결과의 노드를 가져와 사용자로부터 수정할 문자를 입력 받고, 선택된 노드의 문자를 새 문자로 변경한다.

출력: 수정된 문자 적용

5. 이전 결과 반환 함수

입력: SearchState 포인터

현재 선택된 탐색 결과에서 이전 결과를 찾는 기능.

탐색 결과 리스트를 순회하면서 현재 선택된 결과의 이전 결과를 찾는다.

출력: 이전 결과 반환, 찾지 못하면 NULL 반환.

3.3.9 나가기

Ctrl + Q를 눌러 나간다. 만약 저장하지 않은 상태로 나가려면 Ctrl + Q를 2번 이상 누른다.

나가기 기능을 구현하기 위해서는, 사용자가 텍스트를 수정한 후 저장하지 않고 나가려 할 때, 경고를 표시하고, 두 번 이상의 Ctrl + Q 입력으로 나갈 수 있도록 해야 한다.

1. 나가기 관리 함수

입력: 텍스트 버퍼 포인터

사용자가 Ctrl + Q를 눌렀을 때, 나가기 전에 저장되지 않은 내용이 있는지 확인하고, 두 번 이상 Ctrl + Q를 입력해야만 종료할 수 있도록 한다.
quit_attempts 변수를 사용하여 Ctrl + Q가 몇 번 눌렸는지 기록.

파일이 수정된 상태(buffer->modified == true)라면, 첫 번째 시도 시 사용자에게 경고 메세지를 표시하고, 두 번째 시도에서 강제 종료.

수정된 내용이 없다면 즉시 프로그램 종료.

출력: 경고 메시지를 출력 또는 프로그램 종료

2. 에디터 종료 함수

입력: 없음

에디터 종료 함수, 모든 리소스 정리 후 프로그램 안전 종료.

ncurses 라이브러리의 endwin() 호출 후 ncurses 라이브러리 종료.

프로그램에서 할당된 동적 메모리 해제.

exit(0) 호출하여 프로그램 종료.

출력: 없음

3. 텍스트 수정 표시 함수

입력: 텍스트 버퍼 포인터

사용자가 텍스트를 수정하면 텍스트 버퍼가 수정되었음을 표시한다.

텍스트 버퍼가 수정될 때마다 buffer->modified 플래그를 true로 설정한다.

출력: 수정 여부 기록

4. quit_attemps 초기화 함수

입력: 없음

사용자가 나가기 동작을 취소하거나 파일을 저장했을 때, 다시 나가기 시도를 할 경우, quit_attemps 변수를 리셋한다.

출력: 없음

5. 나가기 절차 처리 함수

입력: 텍스트 버퍼 포인터, 사용자가 입력한 키 값

사용자가 나가기 단축키 Ctrl + Q를 입력했을 때 호출되는 함수로, 나가기 절차를 처리한다.

출력: 적절한 나가기 동작 처리

6. 저장 묻기 함수

입력: 텍스트 버퍼 포인터

사용자가 나가기 전에 파일이 수정된 경우, 저장할지를 묻는 함수.

파일이 수정된 경우, 사용자가 저장할지를 묻는 메시지 표시.

사용자가 'y' 또는 'Y'를 입력하면 파일을 저장하고, 'n' 또는 'N'을 입력하면 저장하지 않고 종료.

다른 키 입력 시 종료를 취소하고 나가기 시도 리셋.

출력: 사용자 선택에 따른 파일 저장 또는 나가기 절차 리셋