VITE 프로그램 설계서

20203082\_이승진

1. **목적**

윈도우, 리눅스, 맥 환경에서 컴파일되고 실행이 가능하며 읽어오기, 수정, 저장이 가능한 텍스트 에디터 개발.

1. **기능**

상태 바

▪ 아래에서 2번째 라인에 반전색으로 표시, 파일 이름, 라인 수, 현재 커서 위치 표시

메시지 바

▪ 가장 아래 라인에 표시, 저장, 탐색, 나가기 등 추가 정보 표시

텍스트 수정

▪ 키보드를 이용하여 바로 수정 가능. 영어만 수정하는 것으로 가정

이동

▪ 화살표 키

* ←, → 입력 시 문자 단위 이동
* ↑, ↓ 라인 단위 이동

▪ 홈 키

* Home키 입력 시 현재 라인의 가장 앞으로 이동

▪ 엔드 키

* End키 입력 시 현재 라인의 가장 끝으로 이동

▪ 페이지 업

* Page Up 입력 시 이전 페이지로 이동.

▪ 페이지 다운

* Page Down 입력 시 다음 페이지로 이동.

저장

– Ctrl-S ( Ctrl\_A in Linux/Mac) 입력 시 저장 가능. 새로운 파일 저장 시 파일 이름 입력

탐색

– Ctrl-F 입력 시 찾고자 하는 문자 하이라이트. 화살표키를 이용하여 이동. Enter 입력 시 검색 종료 및 탐색 결과에서 수정, Esc 입력 시 탐색 포기 및 탐색 이전 커서로 이동.

프로그램 종료

– Ctrl-Q ( Ctrl\_E in Linux/Mac)입력 시 프로그램 종료. 만일 저장하지 않았을 시 메시지바에 경고 문자 출력, 한 번 더 Ctrl-Q 입력 시 저장 없이 종료

1. **구현 라이브러리**

|  |  |  |
| --- | --- | --- |
|  | **호환 OS** | **내용** |
| **wordlist.h** | **Windows/ Linux/ Mac** | **텍스트가 저장될 구조체 및 구조체 컨트롤 함수 선언** |
| **ui.h** | **for Windows** | **콘솔/터미널 인터페이스 컨트롤 함수 선언** |
| **for Linux** |
| **for Mac** |
| **io.h** | **for Windows** | **입력된 경로 파일 읽기/쓰기 함수,**  **키보드 입력 처리 함수 선언** |
| **for Linux** |
| **for Mac** |

**청크 (Chunk)**

이전 버전: 더블링크드 리스트 자료구조로 랩핑 방식으로 편집 영역을 보여주도록 계획

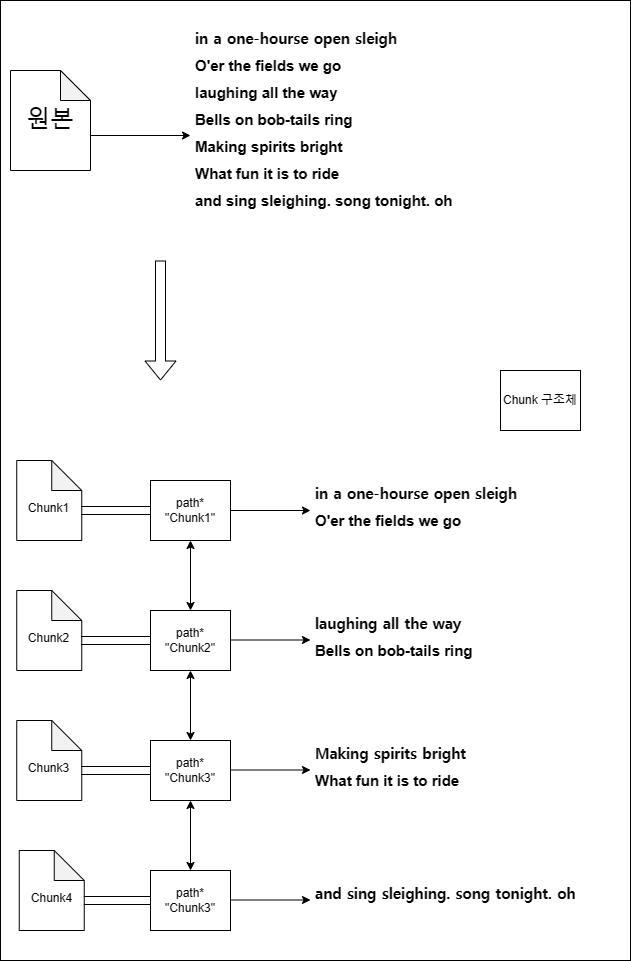
문제 발생: 에서 200메가바이트 이상의 파일을 읽을 시 컴퓨터가 멈추는 현상 발견

* 텍스트를 배열에 저장하는 형태로 전환, 랩핑 방식 포기
* 더 큰 파일을 읽기위해 전체 파일 나눠 편짐 영역만 메모리에 올릴 수 있도록 계획 \*청크파일
* line 구조체

**typedef** **struct** **line** {  
 //윗라인과 아래라인  
 **struct** **line**\* **up**;  
 **struct** **line**\* **down**;  
  
 //문자열 저장 배열, 배열 크기, 텍스트 길이  
 **char**\* text;  
 **int** arrSize;  
 **int** textSize;  
}line;

* chunk 구조체

**typedef** **struct** **chunk** {  
 //이전 청크와 다음 청크  
 **struct** **chunk**\* **prev**;  
 **struct** **chunk**\* **next**;  
  
 //청크 고유 path  
 **char**\* path;  
  
 //라인 리스트와 리스트 크기  
 line\* line\_list;  
 **int** listSize;  
   
 //청크 활성화 상태  
 **int** state;  
}chunk;



* 원본 파일을 일정량의 라인을 가진 청크 파일로 분리하여 각각의 청크 파일의 경로를 가진 Chunk구조체 리스트를 생성
* cursor 구조체

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**typedef** struct cursor {  
 //커서 위치값  
 **int** x;  
 **int** y;  
  
 //현재 참조 청크  
 chunk\* curChunk;  
  
 //현재 참조 라인  
 **int** line;  
}cursor;

* 현재 참조하고 있는 청크와 인접 청크만 활성화시켜 메모리에 올린다.
* 변경사항은 실시간으로 청크 파일에 반영 저장하지 않고 종료 시 청크 파일만 삭제하면 원본파일을 유지할 수 있다.
* 편집영역 표시: cd 구조체

**typedef** **struct** **coordinate** {  
 **int** x;  
 **int** y;  
}cd;

cd startCd = { 0,0 }, endCd = { srCol - 1, srRow - 3 };

* 단순하게 행과 열에 대한 정보를 담을 cd 구조체이다. startCd와 endCd두개를 선언하여 편집영역 좌상단과 우하단을 지정하여 그 내부에 들어오는 텍스트 들을 화면에 보여준다
* 커서 이동에 따른 편집영역 이동: checkCursor

**void** checkCursor(cd\* startCd, cursor\* curCur, cd\* endCd) {  
 **if** (curCur->**x** < startCd->**x**) {  
  
 startCd->**x** = curCur->**x**;  
 endCd->**x** = startCd->**x** + 99;  
 }  
 **else** **if** (curCur->**x** > endCd->**x** - 2) {  
 endCd->**x** = curCur->**x** + 2;  
 startCd->**x** = endCd->**x** - 99;  
 }  
  
 **if** (curCur->**y** < startCd->**y**) {  
 startCd->**y** = curCur->**y**;  
 endCd->**y** = startCd->**y** + 22;  
 }  
 **else** **if** (curCur->**y** > endCd->**y**) {  
 endCd->**y** = curCur->**y**;  
 startCd->**y** = endCd->y - 22;  
 }  
  
}

* 현재 커서의 x,y 값이 startCd와 endCd 내부영역을 벗어날 시 startCd와 endCd의 값을 조절해 현재 커서가 startCd와 endCd의 사에이 위치할 수 있도록 조졸한다.
* 고유 파일명 생성: getUniquePath

**char**\* getUniquePath() {  
 **char** buffer[80] = "Chunk";  
 **int** idx = 5;  
 **static** **int** chunkNum = 1;  
 **int** num = chunkNum;  
  
 chunkNum++;  
  
 while (num != 0) {  
 buffer[idx++] = num % 10 + '0';  
  
 num /= 10;  
 }  
 buffer[idx] = '\0';  
  
 **char**\* path = (**char**\*)malloc(strlen(buffer) + 1);  
 strcpy(path, buffer);  
  
 return path;  
}

* 고유 번호 생성을 위해 static 정적 변수를 사용하여 매 호출마다 chunkNum을 상승시켜 동일하지 않은 값 유지
* 청크 병합과 분할: divChunk, mergeChunk

**void** divChunk(cursor\* curCur) {  
 chunk\* curChunk = curCur->**curChunk**;  
  
 chunk\* newChunk = getChunk();  
 line\* next\_list = curChunk->**line\_list**;  
  
 //divide 기준 라인  
 int sepLine = curChunk->**listSize** / 2 + 1;  
  
 **for** (int i = 1; i < sepLine; i++) {  
 next\_list = next\_list->**down**;  
 }  
  
 //새 청크 활성화  
 newChunk->**state** = 1;  
  
 //현재 커서가 참조해야할 청크 판단  
 int side = curCur->**line** + 1 < sepLine ? 0 : 1;  
  
  
 //청크 연결  
 newChunk->**prev** = curChunk;  
 newChunk->**next** = curChunk->**next**;  
 curChunk->**next** = newChunk;  
 **if** (newChunk->**next** != NULL) newChunk->**next**->**prev** = newChunk;  
  
 //라인리스트 청크에 할당  
 newChunk->**line\_list** = next\_list;  
  
 //청크간 라인리스트 연결 삭제  
 next\_list->**up**->**down** = NULL;  
 next\_list->**up** = NULL;  
  
 newChunk->**listSize** = curChunk->**listSize** - sepLine + 1;  
 curChunk->**listSize** = sepLine - 1;  
  
 **if** (side == 0) {  
 **if** (newChunk->**next** != NULL) chunkOff(newChunk->**next**);  
 }  
 **else** {  
 **if** (curChunk->**prev** != NULL) chunkOff(curChunk->**prev**);  
  
 curCur->**curChunk** = newChunk;  
 curCur->**line** = curCur->**line** - sepLine + 1;  
 }  
}  
  
void mergeChunk(cursor\* curCur) {  
 chunk\* curChunk = curCur->**curChunk**;  
 line\* curLine = curChunk->**line\_list**;  
  
 //다음 청크가 존재하는 경우  
 **if** (curChunk->**next** != NULL) {  
 chunk\* targetChunk = curChunk->**next**;  
  
 line\* connect1 = curChunk->**line\_list**;  
 line\* connect2 = targetChunk->**line\_list**;  
  
 **while** (connect1->**down** != NULL) {  
 connect1 = connect1->**down**;  
 }  
  
 //두 청크간 line\_list 연결  
 connect1->**down** = connect2;  
 connect2->**up** = connect1;  
  
 //청크 재연결  
 curChunk->**next** = targetChunk->**next**;  
 **if** (targetChunk->**next** != NULL) {  
 targetChunk->**next**->**prev** = curChunk;  
 }  
  
 curChunk->**listSize** += targetChunk->**listSize**;  
  
 //대상 청크 지우기  
 targetChunk->**prev** = NULL;  
 targetChunk->**next** = NULL;  
 targetChunk->**line\_list** = NULL;  
  
 remove(targetChunk->**path**);  
  
 free(targetChunk->**path**);  
 free(targetChunk);  
  
 //다음 청크 활성화  
 **if** (curChunk->**next** != NULL) chunkOn(curChunk->**next**);  
 }  
  
 //현 청크 사이즈 오버시 divChunk 실행  
 **if** (curChunk->**listSize** > 46) divChunk(curCur);  
}

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

* 현재 편집영역이 비활성화 되어있는 청크영역을 침범할 수 있는 가능성이 있기에 각 청크는 최소한의 line수를 유지하고 있어야한다. -> mergeChunk
* 한 청크가 너무 많은 라인을 가지고 있을 경우 청크를 선언하여 메모리를 관리히는 의미가 퇴색된다.

-> divChunk

예시) 화면 크기를 가로 100, 세로 25으로 고정시켜 놓았기 때문에 편집영역의 세로 값 23을 최소로 46읠 최대로 설정해 놓았다.

**if** (curChunk->listSize > 46) {  
 divChunk(curCur);  
}

if (**curChunk->listSize** < 23) mergeChunk(**curCur**);

* 문자열 탐색: search

**void** search(char\* target, chunk\* chunk\_list, sr\* search\_list, int\* count) {  
 fprintf(stdout, " \r SEARCHING...");  
  
 int targetLen = strlen(target);  
  
 chunk\* curChunk = chunk\_list;  
 sr\* curSr = search\_list;  
   
 //커서  
 int x = targetLen, y = 0, ln = 0;  
  
 **while** (curChunk != NULL) {  
 chunkOn(curChunk);  
  
 line\* curLine = curChunk->**line\_list**;  
 //나쁜 문자 탐색 방식 이용  
 int badChar = targetLen, find = 1;  
  
 **while** (curLine != NULL) {  
  
 **for** (int i = 0; i <= curLine->**textSize** - targetLen;) {  
  
 //target과 비교  
 **for** (int j = targetLen - 1; j >= 0; j--) {  
 badChar--;  
   
 **if** (curLine->**text**[i + j] != target[j]) {  
 find = 0;  
 break;  
 }  
 }  
  
 **if** (find) {  
 curSr->**srCursor** = getCursor(curChunk);  
 curSr->**srCursor**->**line** = ln;  
 curSr->**srCursor**->**x** = x;  
 curSr->**srCursor**->**y** = y;  
  
 curSr->**next** = getSr();  
 curSr->**next**->**prev** = curSr;  
 curSr = curSr->**next**;  
  
 (\*count)++;  
 }  
  
 i += badChar == 0 ? 1 : badChar;  
 x += badChar == 0 ? 1 : badChar;  
 badChar = targetLen;  
 find = 1;  
 }  
   
 curLine = curLine->**down**;  
  
 x = targetLen;  
 y++;  
 ln++;  
 }  
   
  
 chunkOff(curChunk);  
 curChunk = curChunk->next;  
  
 ln = 0;  
 }  
}

1. 알고리즘

- 보이어-무어 나쁜 문자 탐색방식을 이용하여 선형탐색 보다 조금 더 효율적인 방식으로 탐색을 진행한다.

1. 수행방식

- 탐색을 진행할 청크만 활성화 시켜 탐색 후 검색결과를 searchInfo에 저장한다. 나머지 청크에서도 이를 반복한다.

1. 탐색 결과 저장

- 검색 결과가 위치한 chunk와 라인의 위치를 반환하여 탐색 결과를 searchInfo에 저장하여 반환

**io.h**

* 빠르게 파일 읽어오기: readf

**void** **readf**(**char**\* path, chunk\* curChunk, **int**\* lines) {  
 FILE\* fp = fopen(path, "rt");  
  
 chunk\* tmpChunk = curChunk;  
 **char** ch = fgetc(fp);  
 **int** c\_lines = 1;  
  
 **int** now = 1;  
 fseek(fp, 0, SEEK\_END);  
 **int** end = ftell(fp);  
 fseek(fp, 0, SEEK\_SET);  
  
 FILE\* c\_fp = fopen(tmpChunk->path, "wt");  
  
 //최대 1kb, 1줄씩 읽어온다  
 **char** buffer[1024] = { 0 };  
  
 **while** (fgets(buffer, 1024, fp) != NULL) {  
 now = ftell(fp);  
  
 **if** (buffer[strlen(buffer) - 1] == '\n') {  
 (\*lines)++;  
  
 c\_lines++;  
 }  
  
 fputs(buffer, c\_fp);

//라인 수가 46개가 될 시 다음 청크 생성  
 **if** (c\_lines > 46) {  
 **if** (c\_fp != NULL) {  
 fclose(c\_fp);  
 c\_fp = NULL;  
 }  
  
 tmpChunk->next = getChunk();  
 tmpChunk->next->prev = tmpChunk;  
  
 tmpChunk = tmpChunk->next;  
  
 c\_fp = fopen(tmpChunk->path, "wt");  
  
 c\_lines = 1;  
 }  
  
 fprintf(stdout, "\r Now Loading.. %d / %d", now, end);  
 }  
  
 fclose(fp);  
 fclose(c\_fp);  
}

* 최대 1키바이트로 1줄씩 텍스트를 읽어온다. 한 청크당 라인 수가 46개에 도달 시 다음 청크를 생성하여 이어붙이도록 한다.
* 빠르게 데이터 저장: writef

void writef(char\* path, chunk\* chunk\_list, int lines) {  
 chunk\* tmpChunk = chunk\_list;

//최대 1kb씩 저장  
 char **buffer[1024]** = { 0 };  
  
 int now = 0;  
  
 FILE\* fp = fopen(path, "wt");  
  
 gotoXY(99, 24);  
  
 while (tmpChunk != NULL) {  
 FILE\* c\_fp = fopen(tmpChunk->path, "rt");  
  
 int readChars = fread(**buffer,** sizeof(char), 1024, c\_fp);  
  
 while (readChars > 0) {  
  
 fwrite(**buffer,** 1, readChars, fp);  
  
 readChars = fread(**buffer,** sizeof(char), 1024, c\_fp);  
 }  
  
 fclose(c\_fp);  
  
 tmpChunk = tmpChunk->next;  
  
 fprintf(stdout, "\rSaving... ", now, lines);  
 }  
  
 fclose(fp);

* 원본파일 삭제 후 재생성하여 한 청크내에서 최대 1kb씩 읽어와 재생성한 파일에 이어 붙이는 방식으로 구현

**ui.h**

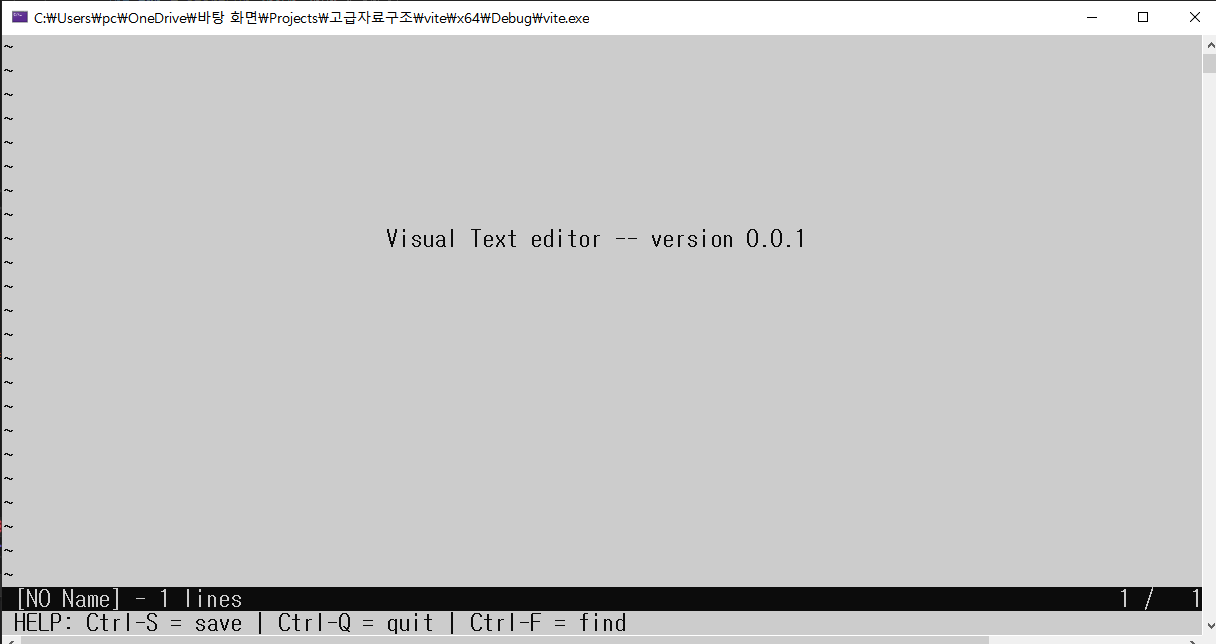
* 화면 깜빡임 현상 축소를 위한 백버퍼 구현

char buffer[23][101];  
text\_print(startCd, curCur, endCd, buffer);  
  
system("cls");  
  
for (int i = 0; i < 23; i++) {  
 fprintf(stdout, "%s\n", buffer[i]);  
 continue;  
}

예) 편집 영역과 같은 크기의 배열을 만들어 해당 배열에 미리 텍스트를 넣은 뒤 한 번에 프린트하는 방식으로 화면 깜빡인 현상 축소

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ‘~’ | ‘\0’ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘\0’ |
| ‘~’ | ‘\0’ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘\0’ |
| ‘~’ | ‘\0’ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘\0’ |
| ‘~’ | ‘\0’ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘\0’ |
| ‘~’ | ‘\0’ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘ ‘ | ‘\0’ |

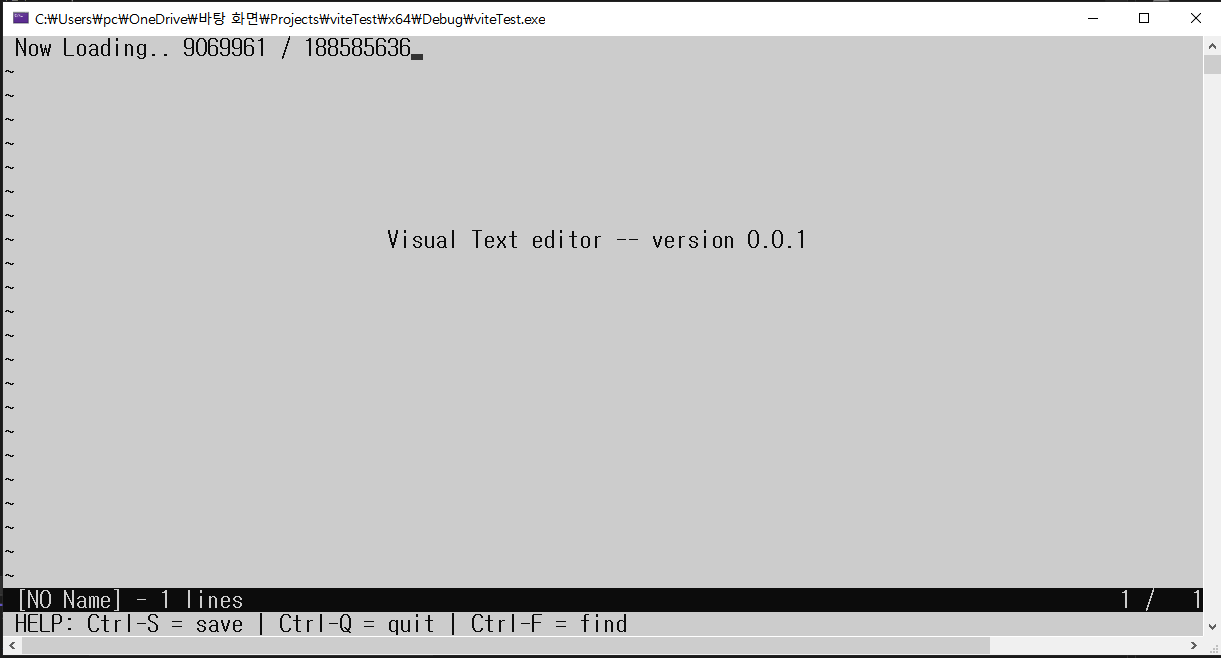
* 화면 출력 예시



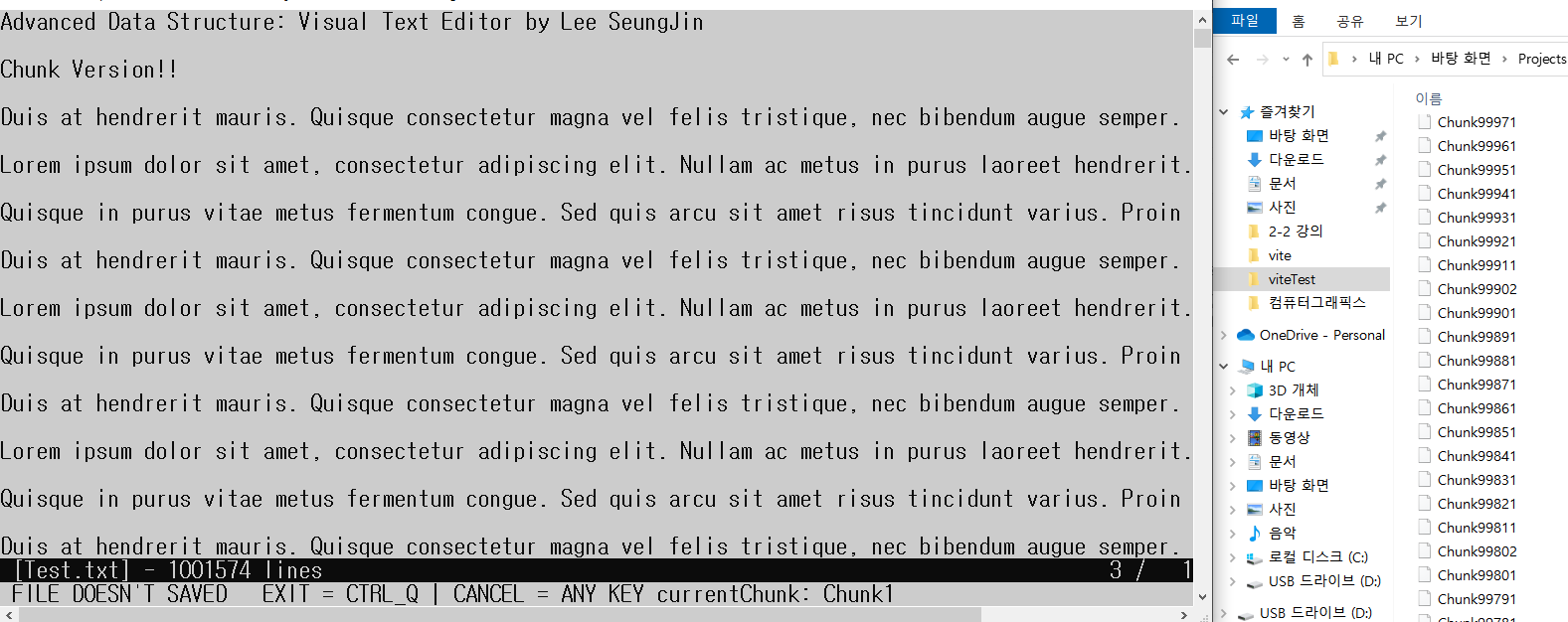
텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 약 190MB 파일 로딩 화면



* 로딩까지 대략 5분가량 소요됨



**개선사항 혹은 전달 사항**

1. 라인 수로 청크의 양을 결정하고 있기에 한 라인만 길게 텍스트가 쓰여 있다면 메모리 관리 효과가 없다.
2. 원본파일과 같은 크기의 청크 파일들을 생성해야 하기 때문에 원본 파일 크기만큼의 하드웨어 공간이 요구된다.
3. 문자 삽입이나 삭제가 배열을 한 칸씩 밀고 당기는 방식으로 구현되어 있어 한 라인이 너무 길 경우 시간 소비가 과할 수 있다.
4. 기술적 한계로 윈도우에서는 종료키 Ctrl\_Q, 저장키 Ctrl\_S이지만 리눅스와 맥에서는 종료키 Ctrl\_E, 저장키 Ctrl\_A로 전환되어 있다.
5. 윈도우와 맥에서 ui가 정상작동하는 것을 확인하였으나 리눅스는 기능은 제대로 작동하나 ui에 약간의 오류가 있는 것을 확인하였다. \* 과제 제출 후에도 수정예정.

**프로그램 순서도**

도표, 텍스트, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명