데이터 구조

단어 찾기 프로그램 프로그래밍

컴퓨터 공학

2015211373 권정인

목차

1. 개요 및 작업 환경
2. 전체 코드 및 코드의 요약
3. 컴파일 및 결과
4. 평가 및 개선 방안
5. 개요 및 작업 환경

저는 Windows 10 내에 있는 Ubuntu 16.04 기반 Linux 서브 시스템 내에서, 코드 작성은 VI(VIM)로, 코드 빌드는 GCC로 실행 하였습니다.

코드 작성 시작 전, 우선 eClass에서 ranwords 파일을 Windows의 Internet Explorer 앱으로 제 PC에 다운로드 한 후, Windows에 내장 된 파일 탐색기 앱으로 C:\ 디렉토리로 파일을 옮겼습니다. 그 후, Bash 환경에서 C:\에 있는 ranwords 파일을 Linux 파일시스템의 제 홈 디렉토리로 복사하였습니다. 그에 대한 명령어는 다음과 같습니다 : *cp /mnt/c/ranwords ~/ranwords*

참고로, Linux 서브 시스템 내에서, 실제 C 드라이브의 파티션은 /mnt/c/에 마운트 되며,

Linux 서브 시스템의 홈 디렉토리는 Windows 파일 탐색기에서 *C:\Users\[Windows사용자이름]\AppData\Local\lxss\home\[UNIX 계정명]* 으로 열립니다.

번외로, Linux 서브 시스템의 /home을 제외한 /(루트) 디렉토리는 Windows 파일 탐색기에서 *C:\Users\[Windows사용자 이름]\AppData\Local\lxss\rootfs*

만일, 파일 작성 코드 작성 및 실행을 제 개인 PC의 Linux 서브 시스템이 아닌 학과 유닉스 서버에서 작성하고 실행한다면, ranwords 파일을 홈 디렉토리로 복사한 후, scp 명령어를 이용해, 학과 서버로 복사해야 합니다. 그에 대한 명령어는 아래와 같습니다 : *cp -P 11800 ranwords[학과서버 ID]@dalma.dongguk.ac.kr:~/ranwords*

1. 전체 코드 및 코드의 요약

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\*\* words\_list(int\* count, char\*\* word) //단어 리스트 만들기 시작

{

char buffer[500]; //버퍼 크기 : 500

int i;

FILE\* fileio = fopen("ranwords", "rt"); //파일 ranwords를 텍스트 읽기 모드로 엽니다.

while (fgets(buffer, sizeof(buffer), fileio) != NULL)

{

(\*count)++;

if (\*count % 1000 == 0)

printf(".");

} //단어를 2000개씩 로드완료때마다 ‘.’을 표시합니다.

printf(“%d Words read.”, \*count); //단어 로드가 끝나면 총 단어 수를 표시합니다.

rewind(fileio); //파일 읽기 위치를 처음으로 이동합니다.

word = (char\*\*)malloc((\*count) \* sizeof(char\*));

for (i = 0; i<(\*count); i++)

word[i] = (char\*)malloc(256 \* sizeof(char)); //각 단어를 동적 할당합니다

for (i = 0; i<(\*count); i++)

fscanf(fileio, "%s", word[i]); //fscanf로 ranwords파일을 스트링 타입으로 word[i]에 읽기

fclose(fileio); //ranwords 파일 닫기

return word; //words\_list가 호출되면 동적 할당 된word를 반환합니다

}

int search(char\* buffer, char\*\* word, int count, int\* line)

{

int i, check;

i = 0; \*line = 0; //i와 line을 0으로 초기화합니다

do

{

check = strcmp(word[i], buffer); //word와 buffer 값이 같으면 check는 0이 됩니다

(\*line)++;

i++;

}

while ((check != 0) && (i<count));

//check가 0이 아니고 count가 i보다 클때까지 line과 i를 1씩 증가합니다

즉, check이 0이 아니며(입력값과 단어가 같지 않음), i보다 count가 큰 경우 계속해서 단어의 줄을 늘려가며 단어를 입력값과 비교합니다

return check; //check 반환

}

void mem\_release(char\*\* word, int count)

{

int i;

for (i = 0; i<count; i++)

free(word[i]); //count(총 단어 수)가 i보다 클때까지 word[i]를 메모리에서 반환합니다

free(word); //word를 메모리에서 반환

}

int main(void)

{

FILE\* filecheck = fopen("ranwords", "rt"); //파일이 존재하는지에 대한 여부를 확인하기 위해 단어 찾기와는 상관없는 filecheck 만들기

char buffer[256];

int count = 0, check, line;

char\*\* word = NULL;

system("clear"); //가독성 향상을 위해 화면 비우기

printf("Form of English words\n");

printf("==============================\n");

printf("Reading words from file..");

if(filecheck==NULL)

{

printf("File ranwords not found. Halting..\n");

return 1;

} //filecheck가 NULL이면, 오류 메시지를 표시하고 1을 리턴합니다

word = words\_list(&count, word);

printf("\n");

printf("Find English words(Sequential Search)\n");

printf("==============================\n");

do

{

printf("Which word do you want to find?\n");

gets(buffer); //입력값을 buffer에 탑재합니다

if (buffer[0] != '\0') //입력값 buffer에 값이 없지 않을경우(값이 있을경우)

{

check = search(buffer, word, count, &line); //상단의 search를 활용하여 입력값 buffer가 파일에 있을경우 0을 반환합니다

if (check != 0) //상단의 파일을 찾았던 check의 값이 0이 아닐경우(입력값이 파일에 없을 경우)

{

printf("\n");

printf("We couldn't find word '%s', %d tried.\n", buffer, line);

} //파일을 찾을 수 없으며, line만큼 시도하였다고 표시하여 줍니다.

else //check의 값이 0일경우(입력값이 파일에 있을 경우)

{

printf("\n");

printf("We found word '%s' in line %d\n", buffer, line);

} //파일을 찾았다는 표시와 함께 몇 번 시도하였는지(즉, 입력값이 파일의 몇 번째에 있는지) 보여줍니다

}

}

while (buffer[0] != '\0'); //입력값 buffer에 입력 값이 없을때 까지 위의 while문을 반복하도록 하세요.

printf("==============================\n");

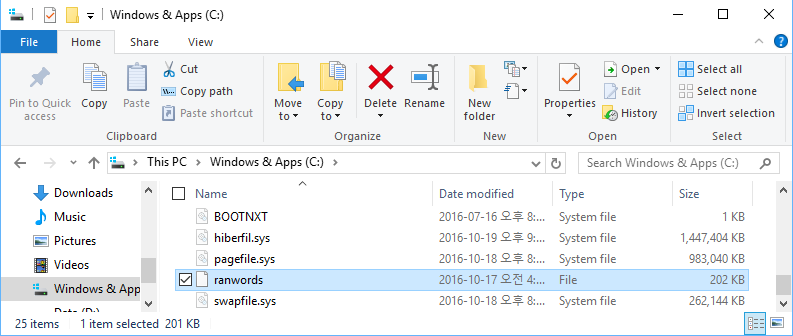
printf("Program halted. Good bye.\n");

mem\_release; //상단의 mem\_release를 실행하여 효율적으로 메모리를 반환합니다.

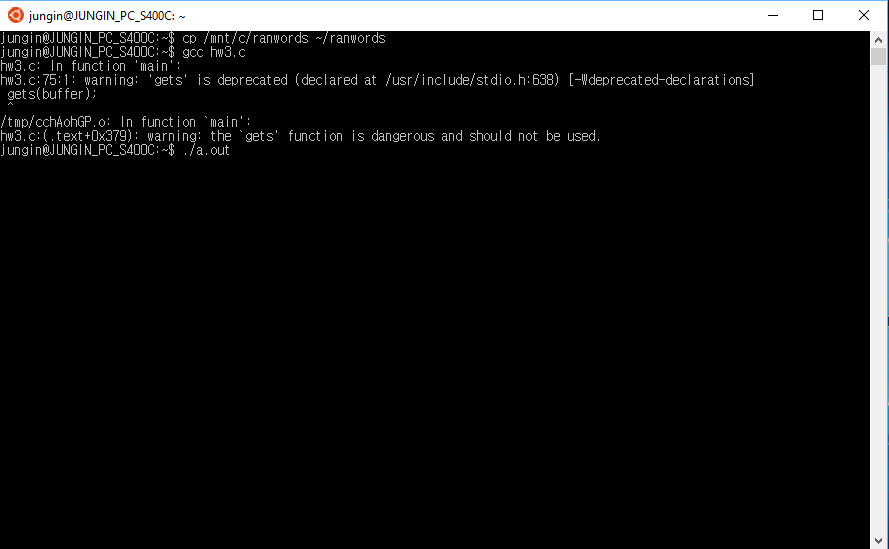
return 0; //0을 리턴하여 정상적으로 프로그램을 종료

}

1. 컴파일 및 결과



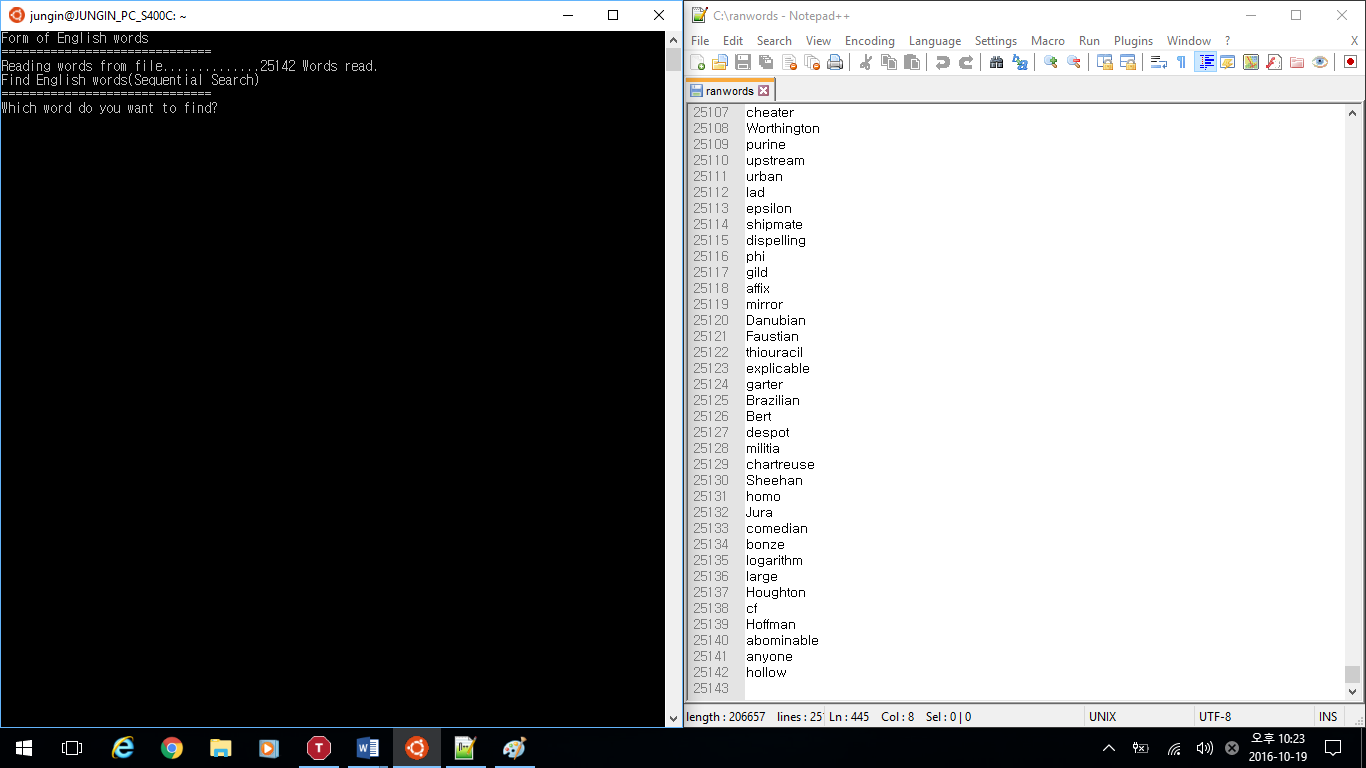
Windows 내의 파일 탐색기를 이용해 접근성이 용이한 C:\로 ranwords 파일을 이동합니다.



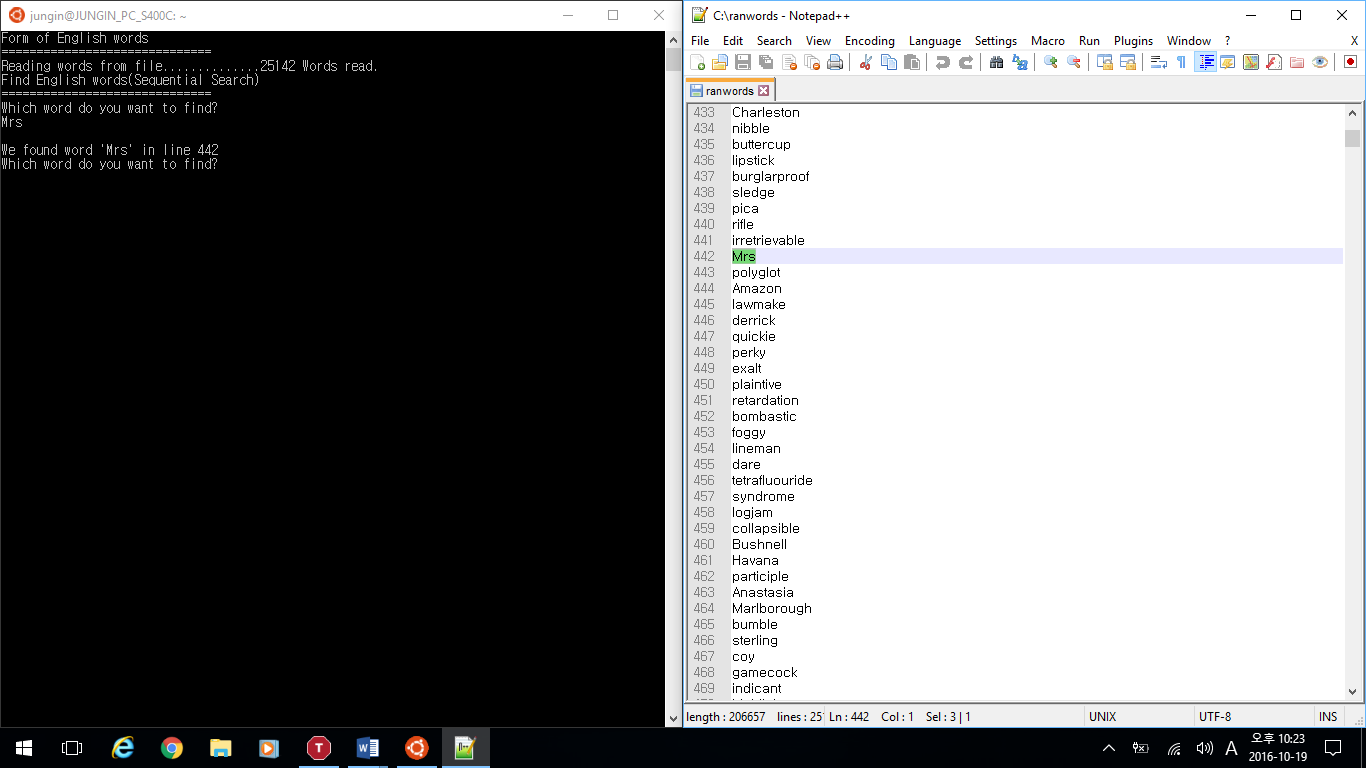
Windows 10 내의 Linux 서브시스템에 접근하여 스크린샷 속 명령을 실행합니다.

위에서부터, C:\에 있는 ranwords 파일을 홈 디렉토리로 복사, GCC를 이용하여 코드 빌드(컴파일), 컴파일하여 생성된 a.out 실행

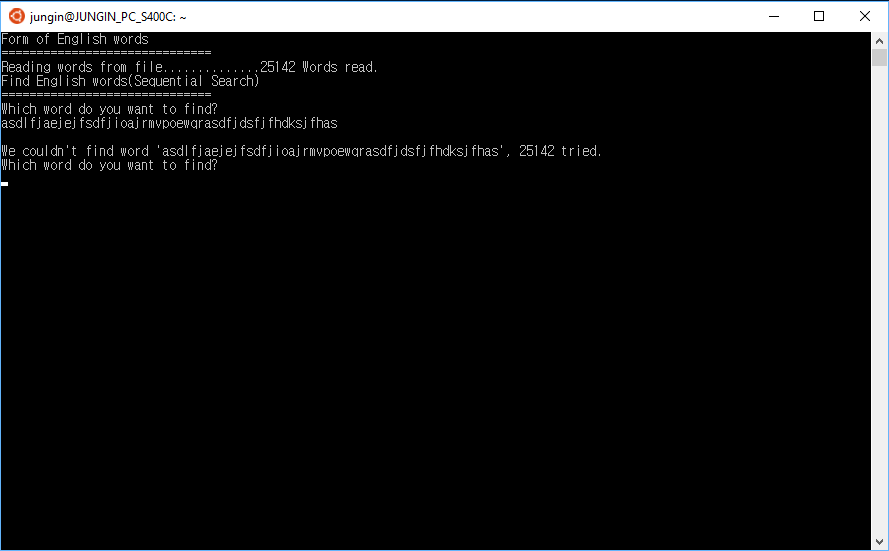
코드를 컴파일 하는 도중 경고 문구가 발생하였지만, 코드에서 사용한 gets의 약한 보안성에 대한 경고일 뿐, 빌드하고 실행하는데는 크게 지장이 없으므로 무시하였습니다.



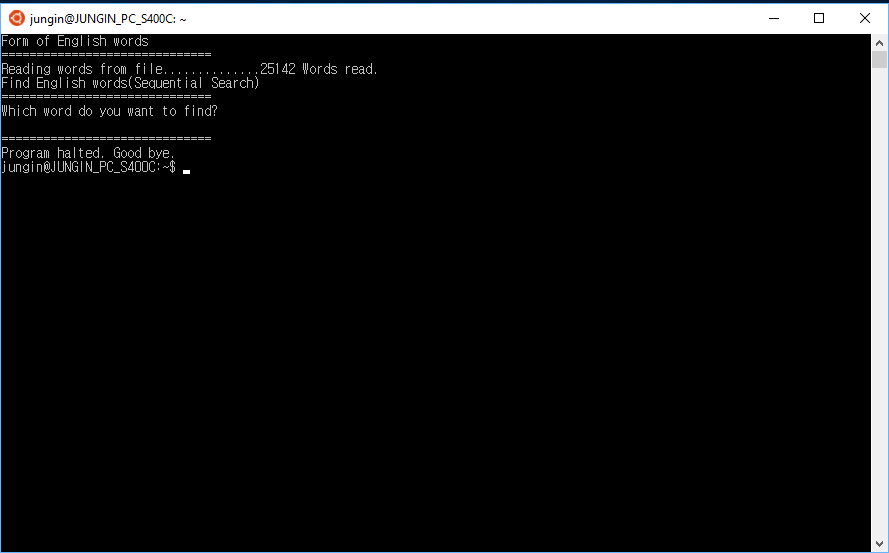
프로그램 실행의 첫 화면입니다. System(“clear”)를 사용하여, 화면이 초기화된 후 실행되는 것을 확인할 수 있습니다. 또한 오른쪽의 텍스트 편집기(notepad++)앱에서 확인할수 있는 바와 같이, 총 단어수가 정상적으로 출력됨을 알 수 있었습니다.



파일 ranwords에 포함되어 있는 ‘Mrs를 검색하여 보았습니다. 왼쪽 bash 창에서는 찾았다는 문구가 정상적으로 표시됨을 알 수 있었습니다. 또한 오른쪽 텍스트 편집기 와 비교해보았을때, 단어를 찾은 시도 수(즉, 단어가 몇번째 줄에 있는지)를 정상적으로 표시해 주고 있음을 확인하였습니다.



두번째로, 단어장에 없는 임의의 영문을 작성해 검색해 보았습니다. 단어를 찾을 수 없다는 메시지와 함께 몇 번 시도를 하였는지(즉, 총 단어 수)를 정상적으로 표시 한 후, 다시 찾을 단어를 입력하라는 메시지가 정상적으로 출력되며, 새로 검색할 단어 입력 준비를 마쳤습니다.



마지막으로, 아무것도 입력하지 않았을때는, 프로그램을 닫는다는 메시지를 표시함과 동시에 프로그램이 정상적으로 종료됨을 확인할 수 있었습니다.

1. 평가 및 개선 방안
2. 처음 코드를 작성할 때, do-while문이 아니라 for문과 if-else 문을 사용하려고 했었습니다. 하지만 인터넷 검색 결과, 이것은 do-while문 안에 새로 if-else 문을 사용하여 코드를 작성해아 한다는것을 알았습니다.
3. 저는 개인적으로, do-while문과 for 문 과 같은 반복문에 약한 것 같습니다. 앞으로 반복문 작성에 관하여 연습이 필요한 것 같습니다.
4. 메모리를 반환하는 free문의 필요성을 몰랐지만, 친구의 조언에 따라 메모리를 반환하는 기능을 구현하였습니다. 하지만, 저는 아직 작은 파일에 대해서는 메모리 반환의 필요성은 없다고 생각합니다.(최근의 컴퓨터 사양 증가에 따라)
5. 아직도 제가 사용하는 시스템에서는 Bash에서 한글 출력이 제대로 되지 않아, 영문으로 작성되었습니다. 학과 서버에 접속하여 작업을 하려하면, 한글 입력이 제대로 되지 않음을 알 수 있었습니다.
6. UNIX(Linux)내의 텍스트 편집기인 VI(VIM)은 저에게는 아직 적응이 필요한 것 같습니다. GUI환경(Windows에서의 텍스트 편집기(notepad++와 같은))에서 코드 작성을 한다면, 조금 더 편하게 작업을 할 수 있을 것 같습니다. 하지만 과제의 가이드라인을 따라, VIM으로 코드 작성을 하였습니다. 또한, 제 작업환경에서(Windows 10 내의 bash), bash 이외의 환경에서 Linux 파일 시스템 파일을 쓰기 작업할 경우, 권한(Permission) 설정이 손실되기 때문에, 임의로 GUI환경에서의 텍스트 편집기로 작업을 할 수 없었습니다.