

**파일 속에서 단어 검색한 후 랭킹 도출.**

**이민지**

**중앙대학교 소프트웨어 전공**

**Searching the word in text files and Ranking them.**

**Lee Min Ji**

**department of software, Chungang UNIV.**

**● 요 약 ●**

본 논문에서는 여러 개의 파일에서 단어를 검색하는 프로그램에 대해 설명하고 있다.

주요어 : c언어, 파일, 단어, 검색, 랭킹

# 목 차

<b>제 1장. 서론</b>	-----	(3)
1.1 연구 배경 및 목적		
1.2 연구 내용 및 방법		
1) 프로그램 설계		
2) 프로그램 구현		
3) 프로그램 적용 및 분석		
1.3 논문의 구성		
1.4 기대효과		
<b>제 2장. 프로그램 설계</b>	-----	(4)
2.1 랭킹의 기준		
2.2 동작 알고리즘		
2.3 검색 가능한 단어의 기준		
<b>제 3장. 프로그램 구현</b>	-----	(6)
3.1 단어 찾기		
3.2 폴더 내의 파일 열고 닫기	-----	(10)
3.3 랭킹 매기기	-----	(11)
<b>제 4장. 프로그램 적용 및 분석</b>		
4.1 단어 찾기	-----	(16)
4.2 폴더 내의 파일 열고 닫기	-----	(17)
4.3 랭킹 매기기	-----	(18)
4.4 프로그램의 한계점	-----	(19)
<b>제 5장. 최종 코드와 프로그램 실행 방법</b>		
5.1 최종 코드	-----	(20)
5.2 프로그램 실행 방법		
5.2.1 github와 폴더의 위치	-----	(29)
5.2.2 작업 경로 설정	-----	(30)
<b>제 6장. 결론</b>	-----	(32)

# 제 1장. 서론

## 1.1 연구 배경 및 목적

정보화의 홍수 시대를 살고 있는 요즘, 필요한 정보만을 선별하는 과정은 선택이 아닌 필수가 되었다. 그리고 정보의 선별에서 얼마나, 어떻게 더 효율적으로 정확하게 선별 할 수 있느냐에 대한 중요성이 커지고 있다.

이에 따라 좀 더 효율적이고 정확한 정보 선별 프로그램의 제작에 관심을 가지게 되었고, 프로그램 제작의 가장 기초 단계인 여러 개의 파일에서 단어 찾기 프로그램을 만들게 되었다.

## 1.2 연구 내용 및 방법

### 1) 프로그램 설계

처음에는 한꺼번에 전체 프로그램을 설계하려고 하니 잘 되지 않았다. 그래서 부분을 나누어 설계하는 것으로 결정했다. 일단, 랭킹의 기준을 설계했다. 그리고 한 파일 내에서 단어 유무 파악 방법을 설계했다. 그 다음 폴더내의 전체 파일을 열고 닫는 방법을 설계하고, 전체 파일에서 단어의 유무를 파악하여 랭킹의 기준에 따라 출력하는 방법을 설계했다.

### 2) 프로그램 구현

설계 한 내용을 바탕으로 하여 프로그램을 구현하였다.

### 3) 프로그램 적용 및 분석

구현된 프로그램을 여러 개의 시험 데이터에 적용시켜 그 값이 맞게 출력 되는지 확인해 보았다.

## 1.3 논문의 구성

본 논문의 구성은 다음과 같다. 2장은 프로그램 설계로 랭킹의 기준, 동작 알고리즘 3장은 프로그램 구현으로 단어 찾기, 폴더 내의 파일 열고 닫기, 랭킹 매기기 4장은 프로그램 적용 및 분석으로 단어 찾기, 폴더 내의 파일 열고 닫기, 랭킹 매기기, 프로그램의 한계점 5장은 최종코드와 프로그램 실행 방법 그리고 마지막 6장은 결론으로 구성되어 있다.

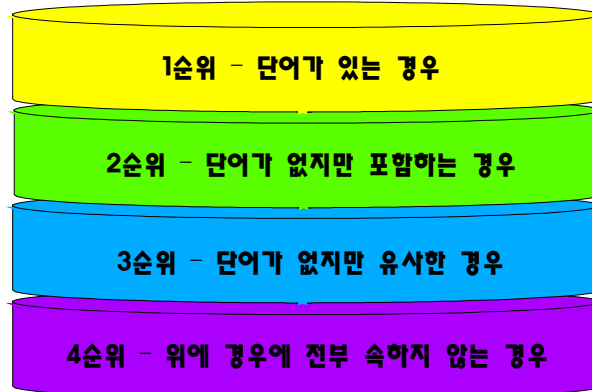
## 1.4 기대효과

기존 프로그램과 다르게 이 프로그램에서는 해당 단어가 있는 경우뿐만이 아니라, 단어가 없는 경우에도 그 유사도에 따라 상위에 랭크 되도록 설정하여 검색의 정확성을 높였다. 이 프로그램을 더 발전시켜 더 많고 다양한 정보를 검색하고 데이터를 처리 할 수 있기를 기대해 본다.

## 제 2장. 프로그램 설계

### 2.1 랭킹의 기준

세부 기능을 구현하기 전, 랭킹의 기준을 제일 먼저 정했다.



<1순위> 단어가 있는 파일의 종류가 많다면 해당 단어의 빈도가 높을수록 순위를 앞쪽으로 매긴다.

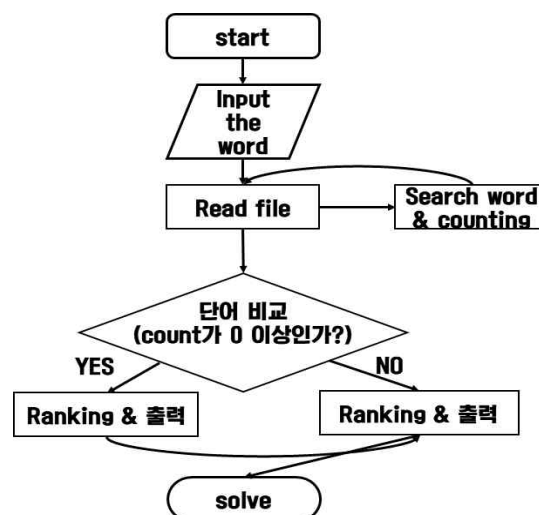
<2순위> 예를 들어, 이 경우는 검색 단어가 hand일 때, 파일 내에 hand라는 단어는 없지만 secondhand라는 단어가 있는 경우다.

마찬가지로 해당 경우인 파일의 종류가 많다면 빈도가 높을수록 순위를 앞쪽으로 매긴다.

<3순위> 이 경우는 예를 들어 검색 단어가 moon일 때, 파일 내에 moon라는 단어는 없지만 mood라는 단어가 있을 때 발생한다. 이 경우 단어의 앞쪽부터 세 글자(m,o,o)가 일치하므로 일치하는 글자는 3이다.

일치하는 글자 수가 같은 다양한 종류의 파일이 있다면, 빈도가 높을수록 순위를 앞쪽으로 매긴다.

### 2.2 동작 알고리즘



[그림 2-2] 단어 검색 프로그램의 알고리즘

## 2.3 검색 가능한 단어의 종류

1) 단어의 시작이 ( - " ' 인 경우

ex. (this -tel "hello 'I think

2) 단어의 끝이 , - . ) ! ~ ? " '인 경우

ex. next, information- end. problem) hi! hi~ what? you" programming'

3) 모두 대문자로 구성된 단어

4) 모두 소문자로 구성된 단어

5) 첫 글자만 대문자로 구성된 단어(나머지는 소문자)

## 제 3장 프로그램의 구현

### 3.1 단어 찾기

파일 내에서 단어를 찾는 방법에 대한 프로그램을 만드는 것을 가장 먼저 작성했다.

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[30], word[30], b1[30], b2[30];
    int count = 0, size, i;

    printf("원하는 단어를 입력하시오. : ");
    scanf("%s", str);

    size = strlen(str);
    if (96 < str[0] && str[0] < 123)
    {
        strcpy(b1, str);
        b1[0] = str[0] - 32;
        strcpy(b2, str);
        strupr(b2);
    }
}
```

[그림 3-1-1]

그림 3-1-1의 if절 이하의 코드는 단어의 경우를 3가지로 만들어 주는 것이다. 입력한 단어의 첫 글자가 소문자인 경우에는 그 단어 전체가 소문자로 구성되어 있다고 판단하여 모두 대문자인 경우, 첫 글자만 대문자인 경우를 만들어 주었다.

```
else if (64 < str[0] && str[0] < 91)
{
    if (64 < str[1] && str[1] < 91)
    {
        strcpy(b1, str);
        strlwr(b1);
        b1[0] = b1[0] - 32;
        strcpy(b2, str);
        strlwr(b2);
    }
    else if (96 < str[1] && str[1] < 123)
    {
        strcpy(b1, str);
        b1[0] = str[0] + 32;
        strcpy(b2, str);
        strupr(b2);
    }
}
}
```

[그림 3-1-2]

그림 3-1-2는 첫 글자가 대문자로 시작하는 단어의 경우에 해당한다. 이때는 두 번째 글자도 비교하여 두 번째 글자가 대문자면 그 단어는 모두 대문자, 소문자이면 첫 글자만 대문자라고 판단하여 각각의 경우에 알맞게 또 다른 문자열을 생성한다.

```

void frontchecker(char strr[100])
{
    int size, i;
    char front;

    size = strlen(strr);

    front = strr[0];

    switch (front)
    {
        case '(':
            for (i = 0; i < size - 1; i++)
                strr[i] = strr[i + 1];
            strr[size - 1] = '\0';
            break;
        case '-':
            for (i = 0; i < size - 1; i++)
                strr[i] = strr[i + 1];
            strr[size - 1] = '\0';
            break;
        case '\\':
            for (i = 0; i < size - 1; i++)
                strr[i] = strr[i + 1];
            strr[size - 1] = '\0';
            break;
        case '\':
            for (i = 0; i < size - 1; i++)
                strr[i] = strr[i + 1];
            strr[size - 1] = '\0';
            break;
        default:
            break;
    }
}

```

[그림 3-1-3]

그림 3-1-3은 파일 내에서 불러온 단어가 알파벳이 아닌 다른 단어로 시작할 때, 그 부분을 제거하기 위해 만든 함수이다.(예, "hello ~~) 시작 가능한 종류는 (, -, ", '의 4가지가 있다고 판단하여 4가지의 경우에 대한 코드를 작성하였다.

```

void lastchecker(char st[100])
{
    int size; //st의 크기
    char last;

    size = strlen(st);

    last = st[size - 1];

    switch (last)
    {
        case ',':
            st[size - 1] = '\0';
            break;
        case '-':
            st[size - 1] = '\0';
            break;
        case '.':
            st[size - 1] = '\0';
            break;
        case ')':
            st[size - 1] = '\0';
            break;
        case '!':
            st[size - 1] = '\0';
            break;
        case '~':
            st[size - 1] = '\0';
            break;
        case '?':
            st[size - 1] = '\0';
            break;
        case '\\':
            st[size - 1] = '\0';
            break;
        case '\':
            st[size - 1] = '\0';
            break;
        default:
            break;
    }
}

```

[그림 3-1-4]

그림 3-1-4는 그림 3-1-3과 비슷한 코드이다. 이번에는 단어의 끝에 알파벳이 아닌 경우가 올 때 그 부분을 제거하기 위한 함수이다.(예, hello!) 끝으로 가능한 종류는 ,와 -, 그리고 ), !, ~, ?, ", '의 9가지가 있다고 판단하여 9가지 경우에 대한 코드를 작성하였다.

```

FILE *fp = fopen("marktest2.txt", "rt");

if (fp == NULL)
{
    printf("이 파일은 비어있습니다!");
    return 1;
}
do
{
    fscanf(fp, "%s", word);
    frontchecker(word);
    lastchecker(word);
    frontchecker(word);
    lastchecker(word);

    if (strlen(word) == size)
    {
        if (strstr(word, str) != NULL)
            count++;
        else if (strstr(word, b1) != NULL)
            count++;
        else if (strstr(word, b2) != NULL)
            count++;
    }
    else
        continue;
} while (!feof(fp));

printf("단어의 개수 : %d \n", count);
fclose(fp);

```

[그림 3-1-5]

그림 3-1-5에서는 파일을 열고 파일을 단어 단위로 읽어오며 사용자와 입력한 단어와 일치한지 파악한다. 파일에서 읽어온 단어의 시작과 끝을 일단 확인 해 준다. 확인은 두 번을 거쳤는데 알파벳이 아닌 것이 앞이나 뒤에 두 번 오는 경우도 있다고 생각해서다.(예를들어, --air 같은 경우)

strstr함수는 단어가 포함된 경우(예를 들어 at을 검색했는데 파일에서 that이라는 단어를 읽어 온 경우)도 조건을 통과시켜 버리기 때문에 파일에서 불러온 단어와 사용자가 입력한 단어가 크기가 일치할 경우에 strstr함수를 사용하여 단어의 일치를 파악하도록 했다. 그리고 일치하다고 파악이 되면 횟수를 셸다.

```

if (count == 0)
{
    fseek(fp, 0L, SEEK_SET);
    do
    {
        fscanf(fp, "%s", word);
        frontchecker(word);
        lastchecker(word);
        frontchecker(word);
        lastchecker(word);

        if (size < strlen(word))
        {
            if (strstr(word, str) != NULL)
                include++;
            else if (strstr(word, b1) != NULL)
                include++;
            else if (strstr(word, b2) != NULL)
                include++;
        }
        else
            continue;
    } while (!feof(fp));
}

```

[그림 3-1-6]

그림 3-1-6은 파일 내에 단어가 없는 경우에, 파일 내의 단어가 사용자가 입력한 단어를 포함하고 있는지 파악하는 코드다. 위에서 언급한 strstr 함수의 특성을 이용하였다. 그리고 단어를 포함하는 경우에는 그 횟수를 셸다.



```

if (count == 0 && include == 0)
{
    fseek(fp, 0L, SEEK_SET);
    do
    {
        fscanf(fp, "%s", word);
        frontchecker(word);
        lastchecker(word);
        frontchecker(word);
        lastchecker(word);
        noword = 0;

        if (strncmp(str, word, 2) == 0)
        {
            noword = 2;
            for (num = 2; num < size; num++)
            {
                if (str[num] == word[num])
                    noword++;
                else
                    break;
            }
        }

        else if (strncmp(b1, word, 2) == 0)
        {
            noword = 2;
            for (num = 2; num < size; num++)
            {
                if (str[num] == word[num])
                    noword++;
                else
                    break;
            }
        }

        else if ((strncmp(b2, word, 2)) == 0)
        {
            noword = 2; //이미 2개 일치하므로
            for (num = 2; num < size; num++)
            {
                if (str[num] == word[num])
                    noword++;
                else
                    break;
            }
        }

        if (noword > nowordmax)
        {
            nowordmax = noword;
            nowordmaxc = 1; //max값이 바뀌었으니까
        }
        else if (noword == nowordmax)
        {
            nowordmaxc++; //같은 경우가 많을 수록 앞에
        }

    } while (!feof(fp));
}
fclose(fp);
printf("단어를 포함하는 경우 : %d \n", include);
if (include == 0)
{
    printf("일부분이 단어와 비슷한 경우 : %d / %d번", nowordmax, nowordmaxc);
}

```

[그림 3-1-7]

그림 3-1-7은 파일 내에 단어가 없고, 포함되는 단어도 없는 경우에 해당하는 코드다. 앞에서부터 2글자 이상 같을 때 유사하다고 할 수 있다고 판단하여 strncmp를 이용하여 조건을 설정했다. 그리고 최대 유사한 글자 수가 같은 단어의 횟수를 셸다.

### 3.2 폴더 내의 파일 열고 닫기

이 프로그램은 한 폴더내의 여러 개의 파일에서 사용자가 입력한 단어를 찾는 것이다. 단어 찾는 코드를 작성한 다음, 폴더내의 파일을 순차적으로 열고 닫는 코드를 작성하게 되었다.

```
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <string.h>

typedef struct name
{
    char name[30];
}bello;

void main()
{
    struct finddata_t fd;
    long handle;
    int result = 1;
    int count = 0;
    char word[30];
    int i;
    char strChangeDir[60] = { "c:testfile" };

    bello filename[6];

    handle = _findfirst("c:testfile\\*.\"", &fd);
    if (handle == -1) return;
    while (result != -1)
    {
        if ((strcmp(".", fd.name)) != 0 && (strcmp("..", fd.name)) != 0)
        {
            printf("파일명 : %s, 크기: %d\n", fd.name, fd.size);
            strcpy(filename[count].name, fd.name);
            count++;
        }
        result = _findnext(handle, &fd);
    }
    findclose(handle);
}
```

[그림 3-2-1]

그림 3-2-1은 작업 중인 디렉토리 내의 testfile이라는 폴더 내의 파일을 읽어 오는 코드다.

```
printf("\n총 개수 : %d\n", count);

for (i = 0; i < count; i++)
{
    printf("파일 이름 : %s \n", filename[i].name);
}

if (chdir(strChangeDir) == 0)
{
    for (i = 0; i < count; i++)
    {
        printf("\n\n파일[%d]", i+1);
        FILE *fp = fopen(filename[i].name, "rt");

        if (fp == NULL)
        {
            printf("error to opening file. \n");
            break;
        }

        while (!feof(fp))
        {
            fscanf(fp, "%s", &word);
            printf("%s ", word);
        }

        fclose(fp);
    }
}

return 0;
```

[그림 3-2-2]

그림 3-2-2는 fopen을 통해 폴더 내의 파일들이 잘 열리는지 확인하기 위한 코드다.

### 3.3 랭킹 매기기

단어의 유무와 횟수에 대한 코드와 파일을 열고 닫고 읽어오는 코드를 모두 작성한 다음 프로젝트의 핵심인 랭킹을 매겼다.

```
typedef struct hello
{
    char filename[30];
    int yescount;
    int contain;
    int coin;
    int nocount;
}bello;
typedef struct yes //단어 있는 경우
{
    char yesname[30];
    int yescount;
}Yes;
typedef struct include //없는데 포함하는 경우
{
    char inname[30];
    int contain;
}In;
typedef struct coincide //없는데 앞글자 일치 정도(유사도)
{
    char coinname[30];
    int coin;
    int coincount;
}Co;
```

[그림 3-3-1]

그림 3-3-1은 랭킹에 필요한 요소들을 저장하기 위해 만든 구조체이다.

-> bello구조체에서 단어가 있는 경우에는 그 횟수를 변수 yescount에 저장한다.

-> 단어가 없는데 포함되는 경우에는 그 횟수를 변수 contain에 저장한다.

-> 단어가 없는데 포함도 안되는 경우에는 일치하는 글자 수를 변수 coin에 저장하고 그 횟수를 변수 nocount에 저장한다.

-> 구조체 Yes는 단어가 있는 경우에, In은 포함되는 경우에, Co는 단어가 없고 포함이 되지 않는 경우에 사용하게 될 구조체다.

```
int filecounter()
{
    struct _finddata_t fd1;
    long handle;
    int result = 1;
    int filecount = 0;

    handle = _findfirst("c:\\dataset-2nd\\*.*", &fd1);
    if (handle == -1) return;
    while (result != -1)
    {
        if ((strcmp(".", fd1.name)) != 0 && (strcmp("..", fd1.name)) != 0)
        {
            filecount++;
        }
        result = _findnext(handle, &fd1);
    }
    _findclose(handle);
    return filecount;
}
```

[그림 3-3-2]

그림 3-3-2는 단어를 찾고자 하는 폴더 내의 파일 개수를 세기 위해 새로 만든 함수이다.

```

printf("%d \n", filecounter());

if ((score = (bello*)malloc(sizeof(bello)*filecounter())) == NULL)
{
    printf("score에 자리없음!");
    exit(1);
}

if ((have = (Yes*)malloc(sizeof(Yes)*filecounter())) == NULL)
{
    printf("have에 자리없음!");
    exit(1);
}

if ((in = (In*)malloc(sizeof(In)*filecounter())) == NULL)
{
    printf("in에 자리없음!");
    exit(1);
}

if ((coin = (Co*)malloc(sizeof(Co)*filecounter())) == NULL)
{
    printf("coin에 자리없음!");
    exit(1);
}

```

[그림 3-3-3]

그림 3-3-3은 파일의 개수만큼 구조체 포인터 변수에 메모리를 할당시키는 코드다.

```

for (i = 0; i < filecount; i++)
{
    if (score[i].yescount > 0)
        printf("파일명 : %s에는 단어가 있습니다!  있는 개수 : %d \n", score[i].filename, score[i].yescount);
    else if (score[i].yescount == 0)
    {
        if (score[i].contain > 0)
            printf("파일명 : %s에는 단어가 없습니다!, 포함 횟수 : %d \n", score[i].filename, score[i].contain);
        else if (score[i].contain == 0)
        {
            if (score[i].nocount > 0)
                printf("파일명 : %s에는 단어가 없습니다!, 일치하는 단어 수 : %d, 횟수 : %d \n", score[i].filename, score[i].coin, score[i].nocount);
            else if (score[i].nocount == 0)
                printf("파일명 : %s에는 그 어떤 비슷한 단어도 없습니다!\n", score[i].filename);
        }
    }
}

```

[그림 3-3-4]

그림 3-3-4는 bello 구조체에 값이 제대로 입력되었는지 확인하기 위한 코드다.(최종 코드에는 없는 부분이다.)

```

printf("\n\n새로운 구조체에 넣은 뒤\n\n");

j = 0, n = 0, m = 0;
for (i = 0; i < filecount; i++)
{
    if (score[i].yescount > 0)
    {
        strcpy(have[j].yesname, score[i].filename);
        have[j].yescount = score[i].yescount;
        j++;
    }

    if (score[i].contain > 0)
    {
        strcpy(in[n].iname, score[i].filename);
        in[n].contain = score[i].contain;
        n++;
    }

    if (score[i].nocount > 0)
    {
        strcpy(coin[m].coinname, score[i].filename);
        coin[m].coin = score[i].coin;
        coin[m].coincount = score[i].nocount;
        m++;
    }
}

```

[그림 3-3-5]

그림 3-3-5는 랭킹을 매기기 위해 위에서 선언한 구조체 Yes, In, Co에 값을 넣는 코드다.

```

for (i = 0; i < filecount; i++)
{
    if (have[i].yescount > 0)
    {
        printf("파일명 %s에는 단어가 %d번 있습니다.\n", have[i].yesname, have[i].yescount);
    }
}

for (i = 0; i < filecount; i++)
{
    if (in[i].contain > 0)
    {
        printf("파일명 %s에는 단어가 %d번 포함되어 있습니다.\n", in[i].iname, in[i].contain);
    }
}

for (i = 0; i < filecount; i++)
{
    if (coin[i].coincount > 0)
    {
        printf("파일명 %s에는 단어와 글자가 %d개 일치하는 것이 %d번 있습니다.\n", coin[i].coinname, coin[i].coin, coin[i].coincount);
    }
}

for (i = 0; i < filecount; i++)
{
    if (!(score[i].yescount > 0) && !(score[i].contain > 0))
    {
        if (!(score[i].nocount > 0))
        {
            printf("파일명 %s에는 일치하거나 유사한 단어가 없습니다!\n", score[i].filename);
        }
    }
}

```

[그림 3-3-6]

그림 3-3-6은 구조체 Yes, In, Co에 값이 제대로 들어갔는지 확인하기 위한 코드다.(최종 코드에는 없는 부분이다.)

```

printf("\n\n-----결과-----\n\n");

j = 0, n = 0, m = 0;
for (i = 0; i < filecount; i++)
{
    if ((score + i)->yescount > 0)
    {
        strcpy((have + j)->yesname, (score + i)->filename);
        (have + j)->yescount = (score + i)->yescount;
        j++;
    }

    if ((score + i)->contain > 0)
    {
        strcpy(in[n].iname, (score + i)->filename);
        (in + n)->contain = (score + i)->contain;
        n++;
    }

    if ((score + i)->nocount > 0)
    {
        strcpy((coin + m)->coinname, (score + i)->filename);
        (coin + m)->coin = (score + i)->coin;
        (coin + m)->coincount = (score + i)->nocount;
        m++;
    }
}

```

[그림 3-3-7]

그림 3-3-7은 구조체 bello에 저장된 값을 bubble sort를 효율적으로 실행하기 위해 각각의 경우에 따라 나뉜 구조체 Yes, In, Co에 앞에서부터 값을 채워 넣는 코드다.

```

for (k = 0; k < filecount - 1; k++)
{
    for (h = 0; h < filecount - k - 1; h++)
    {
        if ((have + h)->yescount < (have + h + 1)->yescount)
        {
            temp1 = have[h];
            have[h] = have[h + 1];
            have[h + 1] = temp1;
        }
        else if (!(have[h].yescount > 0))
            break;
    }
}

```

[그림 3-3-8]

그림 3-3-8은 bubble sort를 이용해 구조체 Yes를 오름차순으로 재정렬하는 코드다.

```

for (i = 0; i < filecount; i++)
{
    if ((in + i)->contain > 0)
    {
        for (k = 0; k < filecount - 1; k++)
        {
            for (h = 0; h < filecount - k - 1; h++)
            {
                if ((in + h)->contain < (in + h + 1)->contain)
                {
                    temp2 = in[h];
                    in[h] = in[h + 1];
                    in[h + 1] = temp2;
                }
            }
        }
    }
}

```

[그림 3-3-9]

그림 3-3-9는 bubble sort를 이용해 구조체 In을 오름차순으로 재정렬하는 코드다.

```

for (i = 0; i < filecount; i++)
{
    if ((coin + i)->coincount > 0)
    {
        for (k = 0; k < filecount - 1; k++)
        {
            for (h = 0; h < filecount - k - 1; h++)
            {
                if ((coin + h)->coin < (coin + h + 1)->coin)
                {
                    temp3 = coin[h];
                    coin[h] = coin[h + 1];
                    coin[h + 1] = temp3;
                }
                else if ((coin + h)->coin == (coin + h + 1)->coin)
                {
                    if ((coin + h)->coincount < (coin + h + 1)->coincount)
                    {
                        temp3 = coin[h];
                        coin[h] = coin[h + 1];
                        coin[h + 1] = temp3;
                    }
                }
            }
        }
    }
}

```

[그림 3-3-10]

그림 3-3-10은 bubble sort를 이용해 구조체 Co를 오름차순으로 재정렬하는 코드다.

```

rank = 1;
printf("-----단어 있는 경우----- \n");
for (i = 0; i < filecount; i++)
{
    if ((have + i)->yescount > 0)
    {
        printf(" [순위%d] - %s, %d번\n", rank, (have + i)->yesname, (have + i)->yescount);
        rank++;
    }
}

printf("-----단어는 없지만 포함하는 경우----- \n");
for (i = 0; i < filecount; i++)
{
    if ((in + i)->contain > 0)
    {
        printf(" [순위%d] - %s, %d번\n", rank, (in + i)->inname, (in + i)->contain);
        rank++;
    }
}

```

[그림 3-3-11]

```

printf("-----단어가 없지만 유사한 단어는 있는 경우----- \n");
for (i = 0; i < filecount; i++)
{
    if ((coin + i)->coincount > 0)
    {
        printf(" [순위%d] - %s %d글자 %d번\n ", rank, (coin + i)->coinname, (coin + i)->coin, (coin + i)->coincount);
        rank++;
    }
}

printf("-----포함하거나 유사한 단어조차도 없는 경우----- \n");
for (i = 0; i < filecount; i++)
{
    if (!((score + i)->yescount > 0) && !((score + i)->contain > 0))
    {
        if (!((score + i)->nocount > 0))
        {
            printf(" [순위%d] - %s\n ", rank, (score + i)->filename);
            rank++;
        }
    }
}

return 0;

```

[그림 3-3-12]

그림 3-3-11과 그림 3-3-12는 최종 rank를 출력하는 코드다. 단어가 있는 경우, 없는데 포함하는 경우, 없고 포함하지 않는데 유사한 경우, 모두 아닌 경우 순이다. 출력 될 때 마다 변수 rank의 값을 올려줌으로써 ranking을 표현했다.

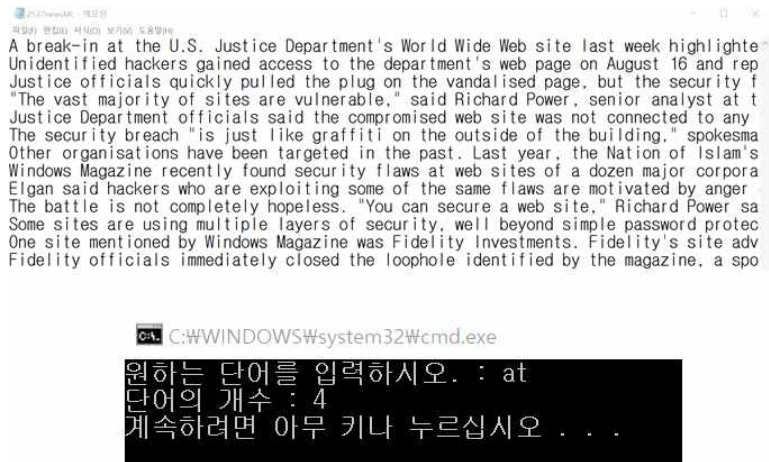


## 제 4장. 프로그램의 적용 및 분석

### 4.1 단어 찾기



[그림 4-1-1]



[그림 4-1-2]



[그림 4-1-3]

그림 4-1-1, 그림 4-1-2과 그림 4-1-3은 그림 3-1-1에서 3-1-5까지의 코드를 실행 해봤을 때의 결과다. 이 결과를 통해 코드가 원하는 데로 잘 실행됨을 알 수 있다.

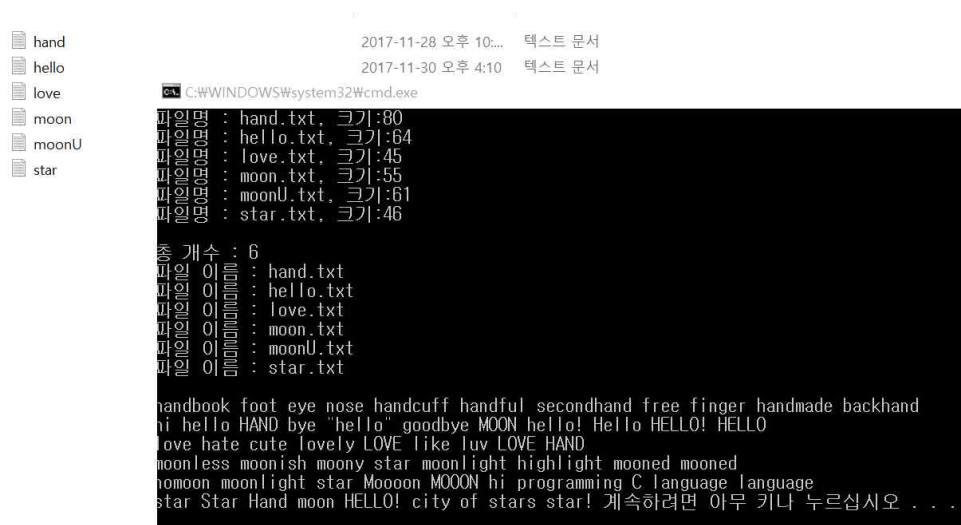




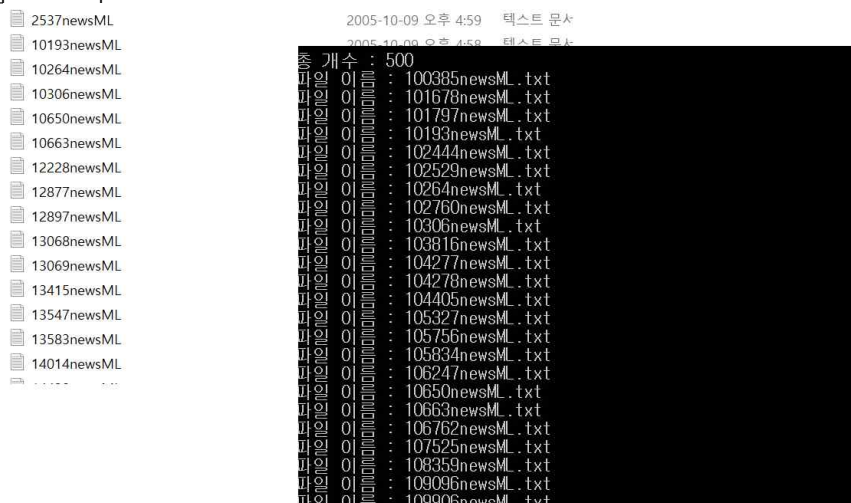
[그림 4-1-4]

그림 4-1-4를 통해 그림 3-1-6과 그림 3-1-7의 코드가 원하는 대로 잘 실행됨을 알 수 있다.

## 4.2 폴더 내의 파일 열고 닫기



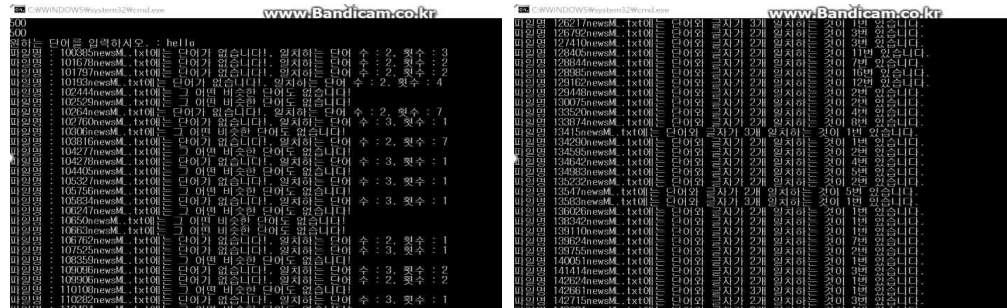
[그림 4-2-1]



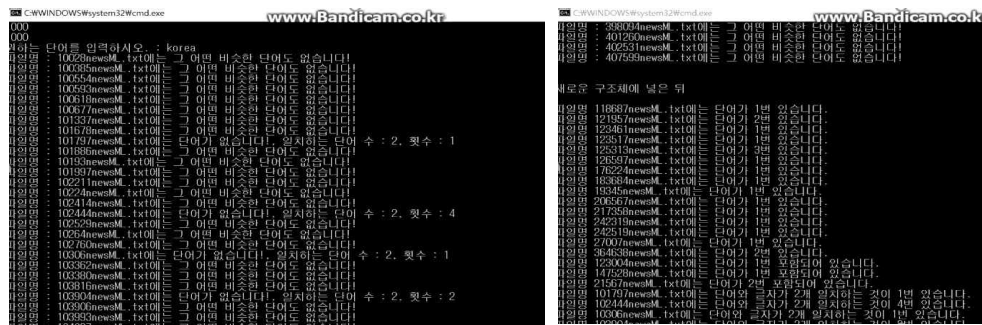
[그림 4-2-2]

그림 4-2-1과 그림 4-2-2를 통해 그림 3-2-1과 그림 3-2-2의 코드가 정상적으로 작동하는지 확인 할 수 있다.

### 4.3 랭킹 매기기



[그림 4-3-1]

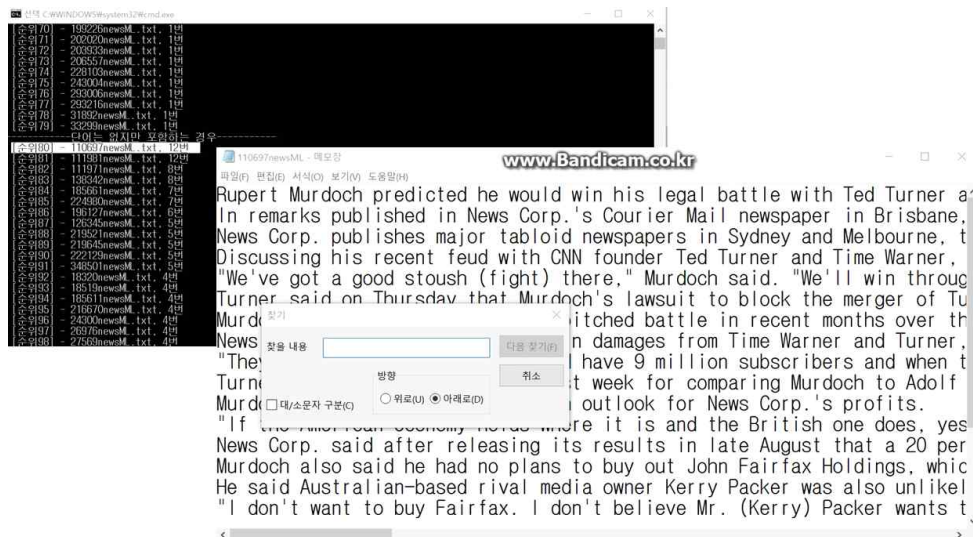


[그림 4-3-2]

그림 4-3-1은 dataset파일로 그림 3-3-1에서 3-3-6까지의 코드를 실행한 결과이고 그림 4-3-2는 dataset-2nd파일로 그림 3-3-1에서 3-3-6까지의 코드를 실행한 결과이다. 그림 4-3-2를 통해 korea라는 단어를 입력했을 때, 단어가 있는 경우와 단어를 포함하는 경우를 합쳐 총 18번이라는 결과가 나오는 것을 알 수 있다.(단어를 포함하는 경우에 해당하는 파일을 확인 해 보니 해당 파일에서는 korea라는 단어는 없고 korean이 있었다.) 그리고 직접 해당 텍스트 파일을 열어 확인해 본 결과 위의 그림과 동일한 결과가 나오는 것을 확인할 수 있었다. 이를 통해 코드가 정상적으로 작동한다는 것을 알 수 있다.



[그림 4-3-3]



[그림 4-3-4]

그림 4-3-3과 그림 4-3-4는 그림 3-3-7에서 3-3-11까지의 코드를 실행한 결과이다. 위의 두 그림은 air라는 단어를 입력했을 때의 결과를 보여주고 있다. 해당 텍스트 파일을 열어 개수를 세어보니 위의 결과와 일치함을 확인 할 수 있었다. 이를 통해 단어 찾기 프로그램이 정상적으로 실행됨을 알 수 있다.(영상을 보고 싶다면 final presentation ppt파일을 보면 된다.)

#### 4.4 프로그램의 한계점

- 1) 파일에서 읽어 온 단어가 앞이나 뒤에 알파벳이 아닌 문자가 3개 이상일 경우.  
ex. 사용자가 검색하고자 하는 단어 -> air  
파일에서 읽어 온 단어 -> ---air  
=> 이런 경우 파일에 단어가 있음에도 단어를 포함하는 경우가 된다.
- 2) 파일에서 읽어 온 단어의 앞이나 뒤에 연구자가 가정하지 않은 문자가 있는 경우.  
=> 이런 경우 파일에 단어가 있음에도 단어를 포함하는 경우가 된다.
- 3) 폴더 경로 설정을 해주지 않으면 올바르게 측정이 되지 않는다.  
=> 프로그램을 실행하기 전 사용자가 경로를 설정해 줘야하는 번거로움이 있다.

## 제 5장. 최종 코드와 프로그램 실행 방법

### 5.1 최종 코드

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <io.h>
4  #include <conio.h>
5  #include <stdlib.h>
6
7  //파일 경로 총 3군데 지정해줘야 함!(Specify the path to the folder you want to search!)
8  //line 55, line 92, line 421
9
10 typedef struct hello
11 {
12     char filename[30];
13     int yescount;
14     int contain;
15     int coin;
16     int nocount;
17 }bello;
18
19 typedef struct yes
20 {
21     char yesname[30];
22     int yescount;
23 }Yes;
24
25 typedef struct include
26 {
27     char inname[30];
28     int contain;
29 }In;
30
31 typedef struct coincide
32 {
33     char coinname[30];
34     int coin;
35     int coincount;
36 }Co;
37
38 int filecounter();
39 void lastchecker(char st[100]);
40 void frontchecker(char strr[100]);
41
42 int main()
43 {
44     char str[100], word[100], b1[100], b2[100];
45     int count = 0, size, i, num = 0, j, m, n, k, h;
46     int include = 0;
47     int noword = 0;
48     int nowordmax = 2;
49     int nowordmaxc = 0;
50     struct _finddata_t fd;
51     long handle;
52     int result = 1;
53     int filecount = 0;
54     char filename[30];
55     char ChangeDir[60] = { "c:dataset-2ndWW" }; //경로 설정1(Setting path 1)
56     int rank;
57
58     bello *score;
59     Yes *have;
```

```

60     In *in;
61     Co *coin;
62
63     Yes temp1;
64     In temp2;
65     Co temp3;
66
67
68     if ((score = (bello*)malloc(sizeof(bello)*filecounter())) == NULL)
69     {
70         printf("score에 자리없음!");
71         exit(1);
72     }
73
74     if ((have = (Yes*)malloc(sizeof(Yes)*filecounter())) == NULL)
75     {
76         printf("have에 자리없음!");
77         exit(1);
78     }
79
80     if ((in = (In*)malloc(sizeof(In)*filecounter())) == NULL)
81     {
82         printf("in에 자리없음!");
83         exit(1);
84     }
85
86     if ((coin = (Co*)malloc(sizeof(Co)*filecounter())) == NULL)
87     {
88         printf("coin에 자리없음!");
89         exit(1);
90     }
91
92     handle = _findfirst("c:\\dataset-2nd\\WW*.\"", &fd); //경로 설정2(Setting path2)
93     if (handle == -1) return;
94     while (result != -1)
95     {
96         if ((strcmp(".", fd.name)) != 0 && (strcmp("..", fd.name)) != 0)
97         {
98             strcpy(score[filecount].filename, fd.name);
99             filecount++;
100         }
101         result = _findnext(handle, &fd);
102     }
103     _findclose(handle);
104
105     printf("원하는 단어를 입력하시오. : ");
106     scanf("%s", str);
107
108     size = strlen(str);
109
110     if (96 < str[0] && str[0] < 123)
111     {
112         strcpy(b1, str);
113         b1[0] = str[0] - 32;
114         strcpy(b2, str);
115        strupr(b2);
116     }
117
118     else if (64 < str[0] && str[0] < 91)
119     {
120         if (64 < str[1] && str[1] < 91)
121         {
122             strcpy(b1, str);

```

```

123     strlwr(b1);
124     b1[0] = b1[0] - 32;
125     strcpy(b2, str);
126     strlwr(b2);
127 }
128
129     else if (96 < str[1] && str[1] < 123)
130     {
131         strcpy(b1, str);
132         b1[0] = str[0] + 32;
133         strcpy(b2, str);
134         strupr(b2);
135     }
136 }
137
138 if (chdir(ChangeDir) == 0)
139 {
140     for (i = 0; i < filecount; i++)
141     {
142         count = 0;
143         include = 0;
144         nowordmax = 2;
145         nowordmaxc = 0;
146         FILE *fp = fopen(score[i].filename, "rt");
147
148         if (fp == NULL)
149         {
150             printf("이 파일은 비어있습니다!");
151             return 1;
152         }
153         do
154         {
155             fscanf(fp, "%s", word);
156             frontchecker(word);
157             lastchecker(word);
158             frontchecker(word);
159             lastchecker(word);
160
161             if (strlen(word) == size)
162             {
163                 if (strstr(word, str) != NULL)
164                     count++;
165                 else if (strstr(word, b1) != NULL)
166                     count++;
167                 else if (strstr(word, b2) != NULL)
168                     count++;
169             }
170             else
171                 continue;
172         } while (!feof(fp));
173
174         (score + i)->yescount = count;
175
176         if ((score + i)->yescount == 0)
177         {
178             fseek(fp, 0L, SEEK_SET);
179             do
180             {
181                 fscanf(fp, "%s", word);
182                 frontchecker(word);
183                 lastchecker(word);
184                 frontchecker(word);
185                 lastchecker(word);

```

```

186
187     if (size < strlen(word))
188     {
189         if (strstr(word, str) != NULL)
190             include++;
191         else if (strstr(word, b1) != NULL)
192             include++;
193         else if (strstr(word, b2) != NULL)
194             include++;
195     }
196     else
197         continue;
198 } while (!feof(fp));
199 (score + i)->contain = include;
200 }
201
202 if ((score + i)->yescount == 0 && (score + i)->contain == 0)
203 {
204     fseek(fp, 0L, SEEK_SET);
205     do
206     {
207         fscanf(fp, "%s", word);
208         frontchecker(word);
209         lastchecker(word);
210         frontchecker(word);
211         lastchecker(word);
212         noword = 0;
213
214         if (strncmp(str, word, 2) == 0)
215         {
216             noword = 2;
217             for (num = 2; num < size; num++)
218             {
219                 if (str[num] == word[num])
220                     noword++;
221                 else
222                     break;
223             }
224         }
225
226         else if (strncmp(b1, word, 2) == 0)
227         {
228             noword = 2;
229             for (num = 2; num < size; num++)
230             {
231                 if (str[num] == word[num])
232                     noword++;
233                 else
234                     break;
235             }
236         }
237
238         else if ((strncmp(b2, word, 2)) == 0)
239         {
240             noword = 2;
241             for (num = 2; num < size; num++)
242             {
243                 if (str[num] == word[num])
244                     noword++;
245                 else
246                     break;
247             }
248         }

```

```

249
250         if (noword > nowordmax)
251         {
252             nowordmax = noword;
253             nowordmaxc = 1;
254         }
255         else if (noword == nowordmax)
256         {
257             nowordmaxc++;
258         }
259     } while (!feof(fp));
260     (score + i)->coin = nowordmax;
261     (score + i)->nocount = nowordmaxc;
262 }
263 fclose(fp);
264 }
265 }
266 }
267
268 printf("\n\n-----결과-----\n\n");
269
270 j = 0, n = 0, m = 0;
271 for (i = 0; i < filecount; i++)
272 {
273     if ((score + i)->yescount > 0)
274     {
275         strcpy((have + j)->yesname, (score + i)->filename);
276         (have + j)->yescount = (score + i)->yescount;
277         j++;
278     }
279
280     if ((score + i)->contain > 0)
281     {
282         strcpy(in[n].iname, (score + i)->filename);
283         (in + n)->contain = (score + i)->contain;
284         n++;
285     }
286
287     if ((score + i)->nocount > 0)
288     {
289         strcpy((coin + m)->coinname, (score + i)->filename);
290         (coin + m)->coin = (score + i)->coin;
291         (coin + m)->coincount = (score + i)->nocount;
292         m++;
293     }
294 }
295
296 for (k = 0; k < filecount - 1; k++)
297 {
298     for (h = 0; h < filecount - k - 1; h++)
299     {
300         if ((have + h)->yescount < (have + h + 1)->yescount)
301         {
302             temp1 = have[h];
303             have[h] = have[h + 1];
304             have[h + 1] = temp1;
305         }
306         else if (!(have[h].yescount > 0))
307             break;
308     }
309 }
310
311

```



```

312     for (i = 0; i < filecount; i++)
313     {
314         if ((in + i)->contain > 0)
315         {
316             for (k = 0; k < filecount - 1; k++)
317             {
318                 for (h = 0; h < filecount - k - 1; h++)
319                 {
320                     if ((in + h)->contain < (in + h + 1)->contain)
321                     {
322                         temp2 = in[h];
323                         in[h] = in[h + 1];
324                         in[h + 1] = temp2;
325                     }
326                 }
327             }
328         }
329     }
330
331     for (i = 0; i < filecount; i++)
332     {
333         if ((coin + i)->coincount > 0)
334         {
335             for (k = 0; k < filecount - 1; k++)
336             {
337                 for (h = 0; h < filecount - k - 1; h++)
338                 {
339                     if ((coin + h)->coin < (coin + h + 1)->coin)
340                     {
341                         temp3 = coin[h];
342                         coin[h] = coin[h + 1];
343                         coin[h + 1] = temp3;
344                     }
345                     else if ((coin + h)->coin == (coin + h + 1)->coin)
346                     {
347                         if ((coin + h)->coincount < (coin + h + 1)->coincount)
348                         {
349                             temp3 = coin[h];
350                             coin[h] = coin[h + 1];
351                             coin[h + 1] = temp3;
352                         }
353                     }
354                 }
355             }
356         }
357     }
358
359     rank = 1;
360     printf("-----단어 있는 경우----- \n");
361     for (i = 0; i < filecount; i++)
362     {
363         if ((have + i)->yescount > 0)
364         {
365             printf(" [순위%d] - %s, %d번\n", rank, (have + i)->yesname, (have + i)->yescount);
366             rank++;
367         }
368     }
369
370     printf("-----단어는 없지만 포함하는 경우----- \n");
371     for (i = 0; i < filecount; i++)
372     {
373         if ((in + i)->contain > 0)
374         {

```

```

375         printf(" [순위%d] - %s, %d번\n", rank, (in + i)->iname, (in + i)->contain);
376         rank++;
377     }
378 }
379
380 printf("-----단어가 없지만 유사한 단어는 있는 경우----- \n");
381 for (i = 0; i < filecount; i++)
382 {
383     if ((coin + i)->coincount > 0)
384     {
385         printf(" [순위%d] - %s %d글자 %d번\n ", rank, (coin + i)->coinname, (coin + i)->coin, (coin + i)->coincount);
386         rank++;
387     }
388 }
389
390 printf("-----포함하거나 유사한 단어조차도 없는 경우----- \n");
391 for (i = 0; i < filecount; i++)
392 {
393     if (!(score + i)->yescount > 0) && !(score + i)->contain > 0)
394     {
395         if (!(score + i)->nocount > 0)
396         {
397             printf(" [순위%d] - %s\n ", rank, (score + i)->filename);
398             rank++;
399         }
400     }
401 }
402
403 free(score);
404 free(have);
405 free(in);
406 free(coin);
407
408 getchar();
409 getchar();
410
411 return 0;
412 }
413
414 int filecounter()
415 {
416     struct _finddata_t fd1;
417     long handle;
418     int result = 1;
419     int filecount = 0;
420
421     handle = _findfirst("c:\\dataset-2nd\\*.txt", &fd1); //경로 설정3(Setting path3)
422     if (handle == -1) return;
423     while (result != -1)
424     {
425         if ((strcmp(".", fd1.name)) != 0 && (strcmp("..", fd1.name)) != 0)
426         {
427             filecount++;
428         }
429         result = _findnext(handle, &fd1);
430     }
431     _findclose(handle);
432
433     return filecount;
434 }
435
436 void lastchecker(char st[100])
437 {

```

```

438     int size; //st의 크기
439     char last;
440
441     size = strlen(st);
442
443     last = st[size - 1];
444
445     switch (last)
446     {
447     case ';':
448         st[size - 1] = '\0';
449         break;
450     case '-':
451         st[size - 1] = '\0';
452         break;
453     case ':':
454         st[size - 1] = '\0';
455         break;
456     case ')':
457         st[size - 1] = '\0';
458         break;
459     case '!':
460         st[size - 1] = '\0';
461         break;
462     case '~':
463         st[size - 1] = '\0';
464         break;
465     case '?':
466         st[size - 1] = '\0';
467         break;
468     case '"':
469         st[size - 1] = '\0';
470         break;
471     case '\':
472         st[size - 1] = '\0';
473         break;
474     default:
475         break;
476     }
477 }
478
479 void frontchecker(char strr[100])
480 {
481     int size, i;
482     char front;
483
484     size = strlen(strr);
485
486     front = strr[0];
487
488     switch (front)
489     {
490     case '(':
491         for (i = 0; i < size - 1; i++)
492             strr[i] = strr[i + 1];
493         strr[size - 1] = '\0';
494         break;
495     case '-':
496         for (i = 0; i < size - 1; i++)
497             strr[i] = strr[i + 1];
498         strr[size - 1] = '\0';
499         break;
500     case '"':

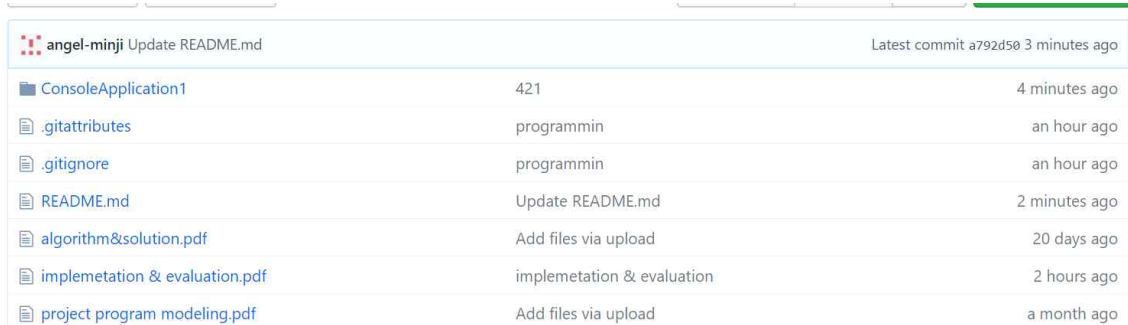
```

```
501     for (i = 0; i < size - 1; i++)
502         strr[i] = strr[i + 1];
503     strr[size - 1] = 'W0';
504     break;
505 case 'W':
506     for (i = 0; i < size - 1; i++)
507         strr[i] = strr[i + 1];
508     strr[size - 1] = 'W0';
509     break;
510 default:
511     break;
512 }
513 }
```

## 5.2 프로그램 실행 방법

### 5.2.1 github를 이용하여 프로그램 다운.

->github 주소 - <https://github.com/angel-minji/SWPROGRAMMING-2>

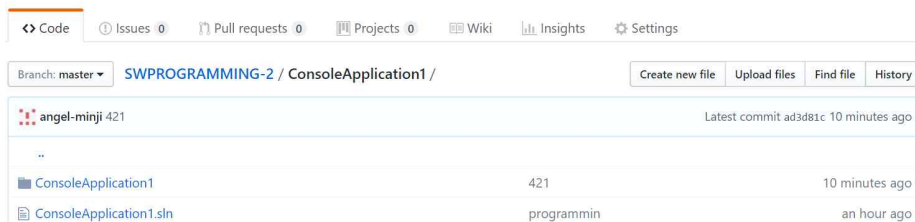


angel-minji Update README.md		Latest commit a792d50 3 minutes ago
ConsoleApplication1	421	4 minutes ago
.gitattributes	programmin	an hour ago
.gitignore	programmin	an hour ago
README.md	Update README.md	2 minutes ago
algorithm&solution.pdf	Add files via upload	20 days ago
implemetation & evaluation.pdf	implemetation & evaluation	2 hours ago
project program modeling.pdf	Add files via upload	a month ago

[그림 5-2-1-1]

그림 5-2-1-1은 저자의 github 주소로 들어갔을 때 볼 수 있는 화면이다. 여기서 전체 파일을 바로 다운 받을 수도 있겠지만 여러 대의 컴퓨터에서 다운 받은 뒤 프로그램을 실행 시켜봤을 때 어떤 컴퓨터는 정상적으로 작동 되고 어떤 컴퓨터는 개발 환경과의 차이 때문에 발생하는 것으로 추측되는 에러가 발생했다. 그래서 실행시킬 때 새로운 프로젝트를 만들어서 소스 파일을 붙여 넣기 하는 것이 제일 좋은 방법인 것 같다.

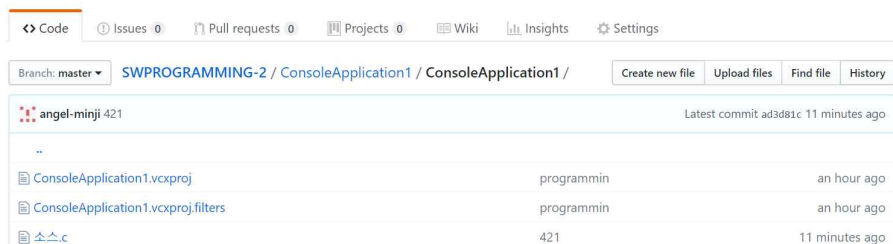
그림의 화면에서 제일 위에 보이는 폴더 ConsoleApplication1을 누른다.



angel-minji 421		Latest commit ad3d81c 10 minutes ago
..		
ConsoleApplication1	421	10 minutes ago
ConsoleApplication1.sln	programmin	an hour ago

[그림 5-2-1-2]

그러면 그림 5-2-1-2의 화면이 나타난다. 이때 한번 더 ConsoleApplication1 폴더에 들어가면



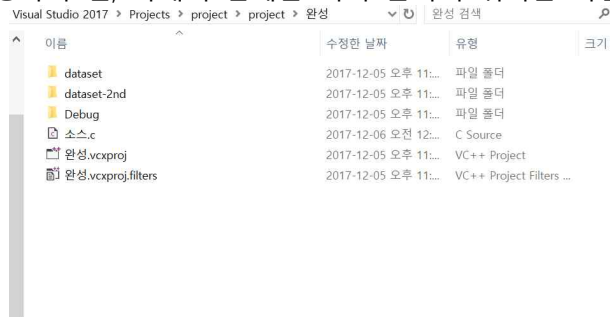
angel-minji 421		Latest commit ad3d81c 11 minutes ago
..		
ConsoleApplication1.vcxproj	programmin	an hour ago
ConsoleApplication1.vcxproj.filters	programmin	an hour ago
소스.c	421	11 minutes ago

[그림 5-2-1-3]

그림 5-2-1-3의 화면이 나타나고 여기서 소스.c를 들어가게 되면 소스 파일을 확인할 수 있다.

비주얼 스튜디오에서 새로운 프로젝트를 만들어(빈 프로젝트로 만들 것) 위에서 확인한 소스 파일을 붙여 넣어 준다.

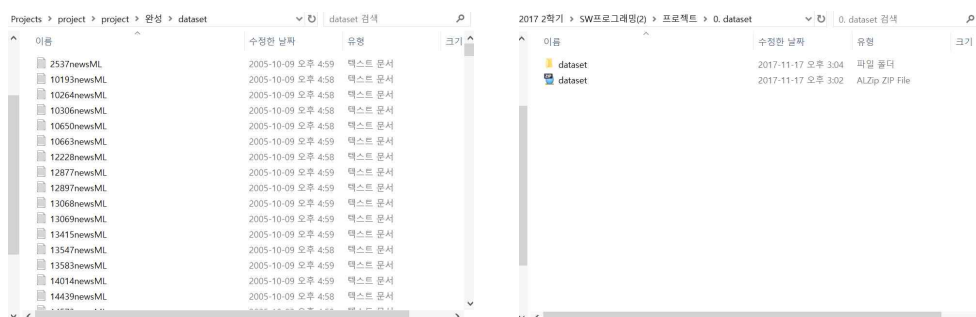
프로그램을 실행하기 전, 아래의 단계를 거쳐 폴더의 위치를 확인해야 한다.



[그림 5-2-1-1]

그림 5-2-1-1은 프로그램이 현재 작업 중인 디렉토리이다.

그림을 통해 작업 중인 디렉토리 내에 검색하고 싶은 파일들이 담긴 폴더가 있음을 확인 할 수 있다.



[그림 5-2-1-2] 맞는 예

[그림 5-2-1-3] 틀린 예

그림 5-2-1-2는 그림 5-2-1-1의 디렉토리에서 dataset이라는 폴더로 이동한 모습이다. 그림처럼 그 폴더에 들어가면 바로 단어 찾기에 사용 될 파일들이 있어야 한다. 그림 5-2-1-3처럼 dataset이라는 폴더에 들어갔을 때 바로 파일들이 나오는 것이 아닌 또 다른 폴더가 있으면 안 된다.

## 5.2.2 작업 경로의 설정

프로그램을 실행하기 전, 총 3부분의 작업 경로를 설정해 주어야 한다.(line 55, 92, 421)

```

41 int main()
42 {
43     char str[100], word[100], b1[100], b2[100];
44     int count = 0, size, i, num = 0, j, m, n, k, h;
45     int include = 0; // 사용자가 입력한 단어가 파일의 단어에 포함되는 경우
46     int noword = 0; // 사용자가 입력한 단어가 파일의 단어에 앞부분에 일부 있을 때
47     int nowordmax = 2; // 위의 숫자의 경우 중 제일 큰 경우(글자수가 가장 많이 일치할 경
48     int nowordmaxc = 0;
49     struct _finddata_t fd;
50     long handle;
51     int result = 1;
52     int filecount = 0;
53     char filename[30];
54     char ChangeDir[60] = { "c:\\dataset-2nd\\#" }; //경로 설정 f.
55     int rank;
56

```

[그림 5-2-2-1]

그림 5-2-2-1은 작업 경로를 설정 해 줘야하는 첫 번째 부분이다. 후에 fopen함수를 실행하는 데 쓰일 것이다. 그림의 예시는 dataset-2nd라는 폴더 내의 파일들에서 단어를 검색하기 위한 것이다.

```

90
91     handle = _findfirst("c:\\dataset-2nd\\*.*", &fd); //경로 설정 2
92     if (handle == -1) return;
93     while (result != -1)
94     {
95         if ((strcmp(".", fd.name)) != 0 && (strcmp("..", fd.name)) != 0)
96         {
97             strcpy(score[filecount], filename, fd.name);
98             filecount++;
99         }
100         result = _findnext(handle, &fd);
101     }
102     _findclose(handle);
103     printf("원하는 단어를 입력하시오. : ");
104     scanf("%s", str);
105

```

[그림 5-2-2-2]

그림 5-2-2-2는 작업 경로를 설정해 줘야하는 두 번째 부분이다. 폴더 내의 파일 이름을 저장하기 위해 사용되는 코드에서 필요한 것이다. 그림의 예시는 dataset-2nd라는 폴더 내의 파일들에서 단어를 검색하기 위한 것이다.

```

414 int filecounter()
415 {
416     struct _finddata_t fd1;
417     long handle;
418     int result = 1;
419     int filecount = 0;
420
421     handle = _findfirst("c:\\dataset-2nd\\*.*", &fd1); //경로 설정 3(Setting path 3)
422     if (handle == -1) return;
423     while (result != -1)
424     {
425         if ((strcmp(".", fd1.name)) != 0 && (strcmp("..", fd1.name)) != 0)
426         {
427             filecount++;
428         }
429         result = _findnext(handle, &fd1);
430     }
431     _findclose(handle);
432     return filecount;
433

```

[그림 5-2-2-3]

그림 5-2-2-3은 작업 경로를 설정해 줘야하는 세 번째 부분이다. 폴더 내의 파일 개수를 세기 위해 만든 함수인 filecounter 내에서 경로 설정이 필요하여 넣은 코드다. 그림의 예시는 dataset-2nd라는 폴더 내의 파일 개수를 세기 위한 것이다.

## 제 6장. 결론

본 논문에서 생각한 랭킹의 기준은 지극히 개인적이다. 따라서 더 좋은 기준이 있다면 그 기준에 따라서 더 효과적으로 결과를 도출 해 낼 수도 있을 것이다.

본 논문에서 쓰인 방법이 정보를 선별하는데 긍정적인 효과를 낳기를 바란다.



## 참고 문헌

1. 열혈 C 프로그래밍(윤성우)
2. Programming in Ansi C(E Balagurusamy)
3. [C언어] DIR 함수; 디렉토리 파일 검색; 와일드카드 지원; findfirst findnext  
(<http://mwultong.blogspot.com/2006/12/c-dir-findfirst-findnext.html>)
4. [C언어] 문자열 대소문자 변환 관련 함수 -strupr, strlwr(<http://shaeod.tistory.com/237>)