Report

제목 : Smart Search System

학 과 : 전자공학과

교 과 : 자료구조

교 수 : 최은만 교수님

학 번 : 2007111921

이 름 : 임학수

제출일 : 11 .06 .14

과 제 요 약 서

설계 과제 명	단일 검색어기반 단어검색 엔진설계

1. Smart Search System 이란?

- Smart Search System은 text의 단어를 효율적인 자료구조로 저장하고, 가장 효율적인 방법으로 검색을 하는데 목적을 둔 프로그램이다.

❑ 특 징

- hash, binary search tree를 이용한 단어의 저장으로 검색의 시간 복잡도를 줄일 수 있습니다.
- 검색한 단어의 Line과 Order를 동시에 출력하여 text에서의 단어위치를 보다 빠르게 찾을 수 있습니다.
- OrSearch, AndSearch 기능으로 두 개의 단어를 동시에 검색 하거나, 두 개의 단어가 들어있는 Line을 출력할 수 있어, 검색의 활용도를 높였습니다.
- 현재 저장되어 있는 자료들을 hash index값을 이용해서 모두 출력 할 수 있어, 현재 저장되어 있는 데이터 들을 한눈에 알아 볼 수 있게 하였습니다.

2. 프로그램 구성 함수

to_Lower_String	입력받은 string을 소문자로 바꿔주는 함수
Build_data_structure	text file을 parsing하여 얻은 단어를 저장하는 함수
insert_data	hash및 bst를 이용해 저장하는 함수
bst_show_inorder	inorder를 이용한 bst 출력 함수
PrintAllWord	현재 저장되어 있는 모든 단어를 출력하는 함수
loading	현재는 장식일 뿐이지만 자료가 커질 경우 thread를 이용해 이용자에게 진행상황을 보여주게 할 수 있는 함수
ShowMenu	main의 menu를 출력하는 함수
SearchWord	검색을 위한 함수
SearchMenu	검색을 위한 menu
NormalSearch	검색할 단어가 한 개일 때의 함수
OrSearch	단어 두 개가 들어있는 Line 및 Order출력
AndSearch	단어 두 개 모두 들어있는 Line출력
main	메뉴를 선택하는 함수

3. 자료구조 및 변수

☐ 변 수

rfp	file을 읽기위한 file 포인터 변수
line	단어의 line을 계산하여 저장
reserve_cnt	parsing을 한 단어의 끝이 '₩n', '.' 인 경우 line을 증가시키기 위한 변수
hash_index	단어의 hash값을 저장
tok	parsing한 값을 임시로 저장
order	단어가 몇 번째 단어인지 계산하여 저장
time	시간 복잡도를 계산하여 저장
tree_pointer	tree_node 구조체를 가리키는 포인터 변수
frequency	단어의 빈도수를 계산하여 저장

☐ 배 열

fname	파일이름을 저장하는 배열
hash_set	hash table을 저장하는 배열
buffer	파일의 한 문장을 잠시 저장하는 배열
loading_table	loading의 진행상황을 나타내기 위한 배열
instruction	사용자가 입력하는 instruction을 저장하기 위한 배열

☐ 구 조 체

tree_node	배 열	
	word	단어를 저장하기 위한 배열
	변 수	
	line	단어의 line을 저장하기 위한 변수
	order	몇 번째 단어인지 저장하기 위한 변수
	left	bst의 왼쪽 노드를 가리키는 포인터 변수
	right	bst의 오른쪽 노드를 가리키는 포인터 변수

제 1 장 서 론

▢ 설계과제 내용

- text file의 문장들을 parsing하여 얻은 단어들을 hashing과 bst를 이용하여 저장하고, 저장되어 있는 단어를 단일 단어 검색, 복합 단어 검색(OrSearch, AndSearch)을 이용하여 효율적으로 검색한다.

▽ 세부 내용 1

Text file의 parsing 및 line, order 계산

- fgets함수로 text file의 한 line을 read하여 구분자를 " "(space) 로 하는 strtok함수를 이용하여 parsing을 합니다. 이 때 문장 끝에 '₩n', '.' 가 오게 되면 line값을 1 증가시키고, parsing을 할 때 마다 order변수 값을 1 증가 시켜 line 및 order변수 값을 계산합니다.

▽ 세부 내용 2

Parsing된 단어 자료구조에 삽입

- parsing된 단어를 동적 할당하여 hashing, bst기법으로 저장합니다. hashing 함수는 문자열의 길이를 5로 모듈 연산한 값을 hash index로 하여 hash_set 포인터로 가리킵니다. 이때 hash_set에 저장되는 단어는 bst로 다시 저장하는데, bst는 strcmp함수를 이용하여 사전 순으로 저장하여 찾기 쉽게 하였습니다. 따라서 bst의 개수는 hash_set[0] ~ hash_set[4] 까지 총 5개가 됩니다.

▽ 세부 내용 3

Normal, Or, And Search

- 이렇게 저장된 자료구조를 search 하는 것이 이 프로그램의 주목적인데, search 하는 방법은 Normal, Or, And Search가 있습니다. 먼저 Normal Search는 단어 하나를 입력하여 그 단어가 속해 있는 line, order와 함께 그 단어가 출현한 빈도수, time complexity를 출력해 줍니다. Or Search는 두 단어 각각이 속해 있는 line, order, 빈도수, time complexity를 출력해 줍니다. 마지막으로 And Search는 두 단어가 동시에 들어있는 line을 출력하는데, 역시 빈도수와 time complexity도 출력합니다.

❑ 진행 일정

▽ 진행일정

세부 개발내용	세부 추진일정				비 고
	주				
	1	2	3	4	
세밀한 작동 환경 계획					20%
구현을 위한 도구 및 방법론 조사					10%
진행 상황 피드백					30%
프로그램 구현					30%
보고서 작성					10%
주간 진도	20%	30%	50%	0%	100%

제 2장 본론

▢ 사용한 헤더파일

▽ stdlib.h → system 명령어(cls, pause), exit 명령어, malloc, free
▽ conio.h → 문자를 입력받기 위한 header file(getch)
▽ string.h → 문자열 편집을 위한 header file(strcpy, strcmp, strlen)
▽ stdio.h → 표준 입출력을 위한 header file(printf, scanf, fgets, gets, puts, fputs, fopen, fclose)
▽ Window.h → Sleep함수 사용하기 위한 header file

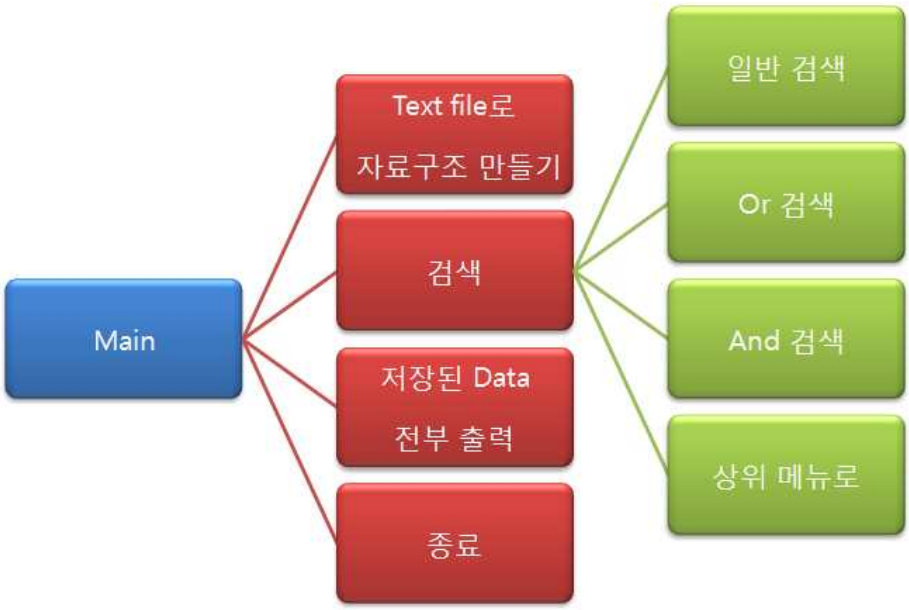
▢ 내부 저장 모양

검색할 텍스트	내부 저장 모양

Line 1: i i i you me
Line 2: me me you me
Line 3: and me

단어	hash index	노드 내용			
		라인 넘버	몇 번째 단어	왼쪽 노드	오른쪽 노드
me(1)	2	1	5	me(2)	NULL
me(2)	2	2	1	me(3)	NULL
me(3)	2	2	2	me(4)	NULL
me(4)	2	2	4	me(5)	NULL
me(5)	2	3	2	NULL	NULL
i(1)	1	1	1	i(2)	NULL
i(2)	1	1	2	i(3)	NULL
i(3)	1	1	3	NULL	NULL
and(1)	3	3	1	NULL	NULL
you(1)	3	1	4	you(2)	NULL
you(2)	3	2	3	and(1)	NULL

▢ 시스템 작동 알고리즘



▽ Noraml Search

- 단어를 입력 받고 strcmp를 이용해서 현재 자료구조에 저장되어 있는 단어를 검색한다. 단어의 길이를 5로 모듈 연산한 값을 index로 하는 hash기법을 이용해 시간을 줄였다. 이때 입력된 단어는 to_Lower_String함수를 통해 단어를 소문자로 변경해준다. 이는 소문자와 대문자의 아스키코드 값이 다르기 때문이다. 해당 hash table에는 다시 bst로 단어가 저장되어 있는데, 사전 배열순으로 같거나 이전 단어는 왼쪽에, 이후 단어는 오른쪽에 저장되어 있어 검색 시간을 이중으로 줄였다. 일치하는 단어를 찾았을 경우에는 해당 단어의 line수와 order, 빈도수, 시간복잡도를 출력하게 되는데, 찾은 단어의 수가 많을 경우 console에서 보기 힘들기 때문에 한 줄에 6단어씩 출력하게 하였다. 일치하는 단어가 없을 경우에는 Not exist라는 메시지를 출력한다.

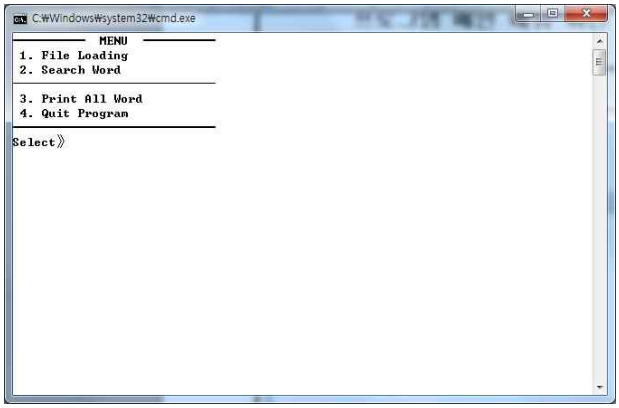
▽ Or Search

- 단어를 공백을 통해 두 개 입력 받은 후, Normal Search와 같은 방법으로 검색한 후 출력한다. 따라서 빈도수와 시간복잡도, line, order는 단어 각각을 Normal Search했을 때 값의 합을 출력하게 된다.

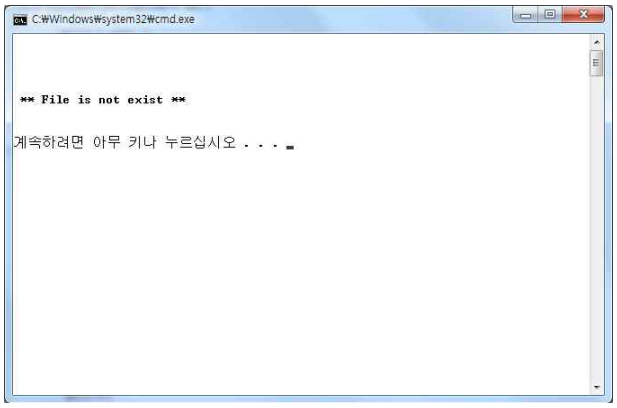
▽ And Search

- And Search의 경우는 두 개의 단어가 존재하는 line을 출력한다. 전반적인 방법은 위의 방법과 동일하고, 다른 점은 중첩 반복문을 이용한 점인데, 이 때문에 시간복잡도가 많이 증가하게 된다. 따라서 이를 조금이라도 줄이고자 searchPoint라는 node pointer를 선언하여 두 번째 반복문이 시작하는 포인트를 설정하게 하였다.

☐ 실행 화면

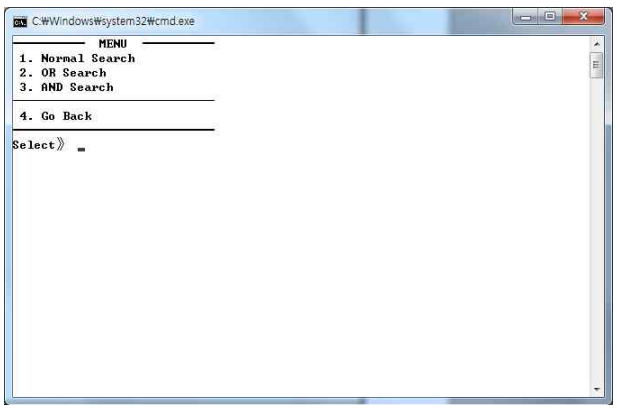


프로그램 메인 메뉴 화면



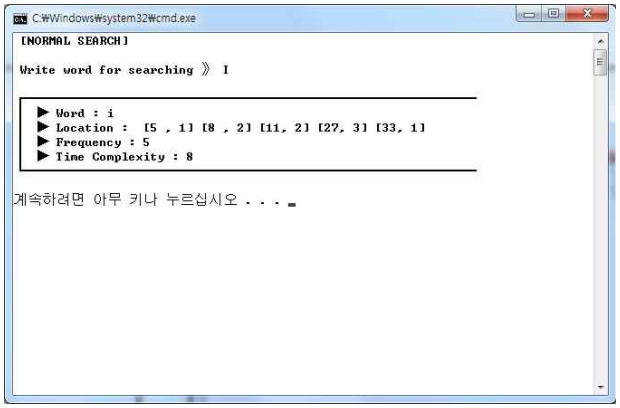
1. File Loading 선택
-> File이 없을 경우

1. File Loading 선택
-> File이 있을 경우

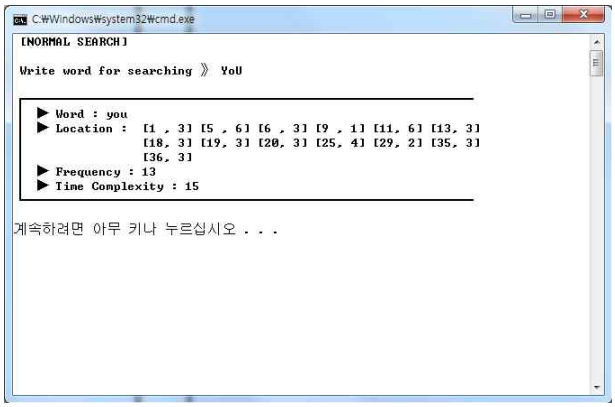


2. Search Word 선택
-> Search Menu

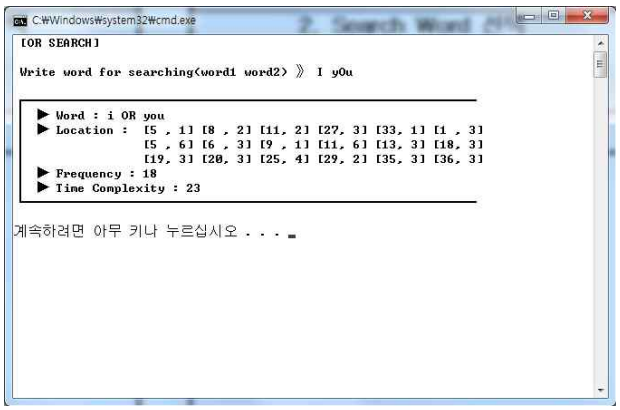
☐ 실행 화면



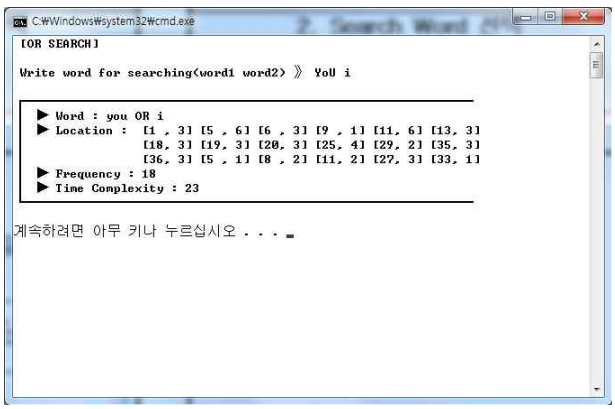
2. Search Word 선택
-> 1. Normal Search 선택
-> 키워드 "I" 검색 화면



2. Search Word 선택
-> 1. Normal Search 선택
-> 키워드 "YoU" 검색 화면

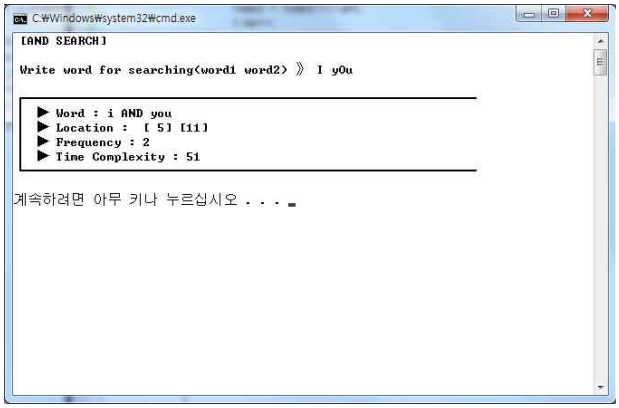


2. Search Word 선택
-> 2. Or Search 선택
-> 키워드 "I yOu" 검색 화면

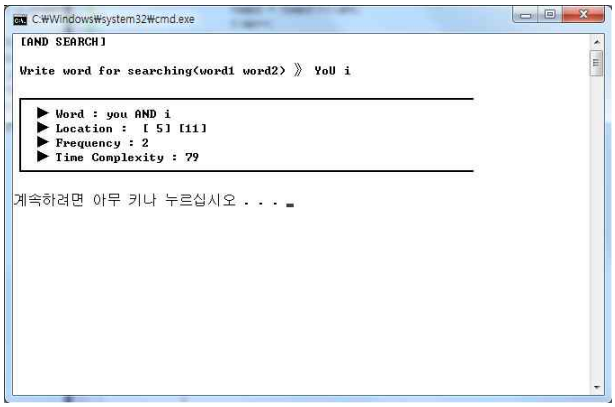


2. Search Word 선택
-> 2. Or Search 선택
-> 키워드 "YoU i" 검색 화면

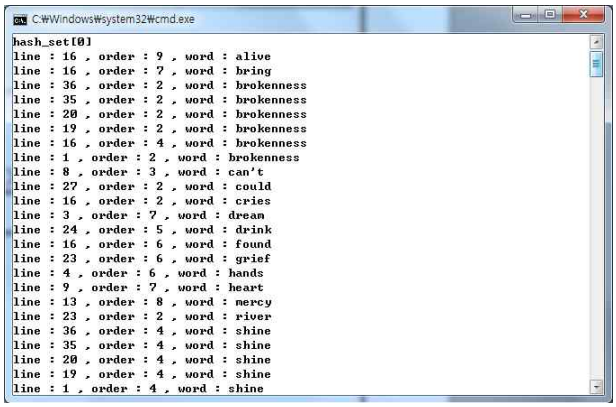
☐ 실행 화면



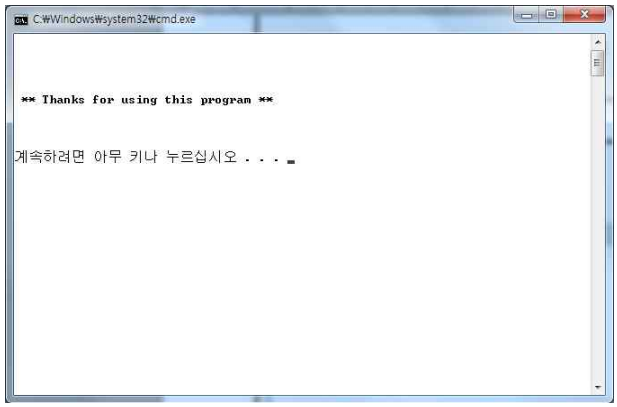
2. Search Word 선택
-> 3. And Search 선택
-> 키워드 "I yOu" 검색 화면



2. Search Word 선택
-> 3. And Search 선택
-> 키워드 "YoU i" 검색 화면



3. Print All Word 선택
-> 각 Hash Index 별로 출력



4. Quit Program 선택
-> 약간의 animation과 함께 종료

제 3 장 결론

☐ 소감

- 자료구조 두 번째 프로젝트. 시험 전 마지막 과제. 프로젝트를 경험하며 여러 함수 레퍼런스 습득과 프로그래밍 실력의 향상이 극대화 되는 기회라는 것을 알기에 최대한 현실에 적용 할 수 있게 구현하려고 노력했다.

기말 시험공부를 조금은 제쳐두고 하는 프로젝트라 불안한 마음도 있었지만 자료구조를 조금씩 수정할 때 마다 줄어드는 시간복잡도를 보면서 미소를 짓곤 했다. 지금 프로그램은 자료의 수가 상당히 적어서 느끼지 못하겠지만 자료가 방대해진다면 시간의 절약을 피부로 느낄 수 있을 거라 생각한다.

지금 내 실력으론 실무에는 닿을 수 없겠지만 언젠가 가게 될 그곳에서의 나를 상상할 수 있는 귀중한 시간이었다.

제 4 장 참고자료

▽ 네이버 지식 in

▽ C프로그래밍(FREELEC, 윤성우)

▽ winapi.co.kr

제 5 장 코드

☐ Header Code

```
#ifndef __SEARCH_ENGINE_H__
#define __SEARCH_ENGINE_H__

#include<Windows.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>


typedef struct tree_node *tree_pointer;
typedef struct tree_node
{
     char word[100];
     int line;
     int order;
     tree_pointer left;
     tree_pointer right;
```

```
}tree_node;

FILE* rfp;

tree_node* hash_set[5] = {NULL,};

void to_Lower_String(char* str);
void Build_data_structure();
void insert_data(int index, char *word, int line, int order);
void bst_show_inorder(tree_pointer ptr);
void PrintAllWord();
void QuitProgram();
void loading();
void ShowMenu();
void SearchWord();
void SearchMenu();
void NormalSearch();
void OrSearch();
void AndSearch();

#endif
```

☐ Source Code

```
#include "search_engine.h"

enum{FILE_LOADING=1, SEARCH_WORD, PRINT_ALL_WORD, QUIT_PROGRAM};
```

```
int main()
{
	int inputMenu = 0;

	while(1)
	{
		ShowMenu();
		inputMenu = getch();
		inputMenu -= '0';

		switch(inputMenu)
		{
		case FILE_LOADING:
			Build_data_structure();
			break;

		case SEARCH_WORD:
			SearchWord();
			break;

		case PRINT_ALL_WORD:
			PrintAllWord();
			break;

		case QUIT_PROGRAM:
			QuitProgram();
			exit(0);
		}
	}
	return 0;
}

void to_Lower_String(char* str) // 소문자로 바꿔주는 함수
{
	while ( *str )
	{
		if ( *str >= 'A' && *str <= 'Z' )
				*str += 32;
		++str;
	}
}

void Build_data_structure() // 파일의 내용을 hash, bst에 저장
{
	char buffer[100];
```

```
 char *tok;
 char temp[100];
 int line=1;
 int reserve_cnt=0;
 int hash_index;
 int order=1;
 char fname[20] = "searchlist.txt";

 if((rfp = fopen(fname, "r")) == NULL)
 {
        system("cls");
    puts(""),puts(""),puts(""),puts("");
        puts(" ** File is not exist ** ");
        puts(""),puts("");
        exit(0);
}

 while(fgets(buffer,100, rfp) > 0)
 {
        tok = strtok(buffer, " ");
        strcpy(temp, tok);

        if(temp[strlen(temp)-1] == '₩n')
        {
                if(strlen(temp) == 1)
                {
                        line++;
                        continue;
                }

                reserve_cnt++;
                temp[strlen(temp)-1] = '₩0';
                if(temp[strlen(temp)-1] == '.')
                {
                        temp[strlen(temp)-1] = '₩0';
                }
                hash_index = strlen(temp) % 5;
                to_Lower_String(temp);
                insert_data(hash_index, temp, line, order);
        }
        else if(temp[strlen(temp)-1] == '.')
        {
                reserve_cnt++;
                temp[strlen(temp)-1] = '₩0';
                hash_index = strlen(temp) % 5;
```

```
		to_Lower_String(temp);
		insert_data(hash_index, temp, line, order);
	}
	else if(temp[strlen(temp)-1] == ',')
	{
		temp[strlen(temp)-1] = '₩0';
		hash_index = strlen(temp) % 5;
		to_Lower_String(temp);
		insert_data(hash_index, temp, line, order);
	}
	else if(tok)
	{
		hash_index = strlen(tok) % 5;
		to_Lower_String(tok);
		insert_data(hash_index, tok, line, order);
	}
	if(reserve_cnt == 1)
	{
		line++;
		reserve_cnt=0;
	}

	order++;

	while(tok =  strtok(NULL, " "))
	{
		strcpy(temp, tok);

		if(temp[strlen(temp)-1] == '₩n')
		{
			if(strlen(temp) == 1)
			{
				line++;
				order=1;
				continue;
			}
		}
		if(temp[strlen(temp)-1] == '₩n')
		{
			reserve_cnt++;
			temp[strlen(temp)-1] = '₩0';
			if(temp[strlen(temp)-1] == '.')
			{
				temp[strlen(temp)-1] = '₩0';
			}
```

```
                        hash_index = strlen(temp) % 5;
                        to_Lower_String(temp);
                        insert_data(hash_index, temp, line, order);
                }
                else if(temp[strlen(temp)-1] == '.')
                {
                        reserve_cnt++;
                        temp[strlen(temp)-1] = '₩0';
                        hash_index = strlen(temp) % 5;
                        to_Lower_String(temp);
                        insert_data(hash_index, temp, line, order);
                }
                else if(temp[strlen(temp)-1] == ',')
                {
                        temp[strlen(temp)-1] = '₩0';
                        hash_index = strlen(temp) % 5;
                        to_Lower_String(temp);
                        insert_data(hash_index, temp, line, order);
                }
                else if(tok)
                {
                        hash_index = strlen(tok) % 5;
                        to_Lower_String(tok);
                        insert_data(hash_index, tok, line, order);
                }
                if(reserve_cnt == 1)
                {
                        line++;
                        reserve_cnt=0;
                }
                order++;
        }

        order = 1;
    }
    fclose(rfp);
    loading();
}

void insert_data(int index, char* word, int line, int order) // hash, bst에 저장
{
    tree_node* new_node;
    tree_node* temp = hash_set[index];
    tree_node* pre_node;
```

```
	new_node = (tree_node*)malloc(sizeof(tree_node));

	new_node->line = line;
	new_node->order = order;
	strcpy(new_node->word, word);
	new_node->left = NULL;
	new_node->right = NULL;

	if(hash_set[index]==NULL)
	{
		hash_set[index] = new_node;
	}
	else
	{
		while(temp)
		{
			if(strcmp(new_node->word, temp->word) > 0)
			{
				pre_node = temp;
				temp = temp->right;
			}
			else
			{
				pre_node = temp;
				temp = temp->left;
			}
		}
		if(strcmp(new_node->word, pre_node->word)>0)
		{
			pre_node->right = new_node;
		}
		else
		{
			pre_node->left = new_node;
		}
	}
}

void bst_show_inorder(tree_pointer ptr) // bst의 inorder 탐색
{
	if(ptr)
	{
		bst_show_inorder(ptr->left);
		printf("line : %d , order : %d , word : %s₩n", ptr->line, ptr->order, ptr->word);
		bst_show_inorder(ptr->right);
```

```
	}
}

void PrintAllWord() // 저장되 있는 모든 단어 출력
{
	int i;

	system("cls");

	for(i=0; i<5; i++)
	{
		printf("₩nhash_set[%d]₩n", i);
		bst_show_inorder(hash_set[i]);
	}
	getch();
}

void QuitProgram() // 프로그램 종료
{
	char endingMessage[40];
	int i;
	strcpy(endingMessage, " ** Thanks for using this program ** ");

	system("cls");
	puts(""),puts(""),puts(""),puts("");
	for(i=0; i<strlen(endingMessage); i++)
	{
		printf("%c", endingMessage[i]);
		Sleep(50);
	}
	puts(""),puts(""),puts(""),puts("");
}

void loading() // 가벼운 loading animation
{
	int loading_table[10] = {0,};
	int i, j;

	system("cls");
	for(i=0; i<10; i++)
	{
		puts(""), puts(""), puts(""), puts(""), puts("");
		fputs("   ", stdout);
		for(j=0; j<10; j++)
		{
```

```
                if(loading_table[j] == 0)
                        fputs("□", stdout);
                else
                        fputs("■", stdout);
        }

        if(i%3==0)
        {
                puts(""), puts(""), puts("");
                fputs(" Loading. ", stdout);
        }
        else if(i%3==1)
        {
                puts(""), puts(""), puts("");
                fputs(" Loading.. ", stdout);
        }
        else if(i%3==2)
        {
                puts(""), puts(""), puts("");
                fputs(" Loading... ", stdout);
        }

        loading_table[i] = 1;
        Sleep(200);
        system("cls");
    }

    puts(""), puts(""), puts(""), puts(""), puts("");
    puts("  ■■■■■■■■■■");
    puts(""), puts(""), puts("");
    fputs(" ** File Loading Is Completed ** ", stdout);
    getch();
}

void ShowMenu() // 메인 메뉴
{
    system("cls");

    printf("————————  MENU  ————————₩n");
    printf(" 1. File Loading₩n");
    printf(" 2. Search Word₩n");
    printf("————————————————————₩n");
    printf(" 3. Print All Word₩n");
    printf(" 4. Quit Program₩n");
    printf("————————————————————₩n");
```

```
	printf("Select》  ");
}

void SearchWord()
{
	enum{NORMAL_SEARCH=1, OR_SEARCH, AND_SEARCH, GO_BACK};

	int inputMenu = 0;

	while(1)
	{
		SearchMenu();
		inputMenu = getch();
		inputMenu -= '0';

		switch(inputMenu)
		{
		case NORMAL_SEARCH:
			NormalSearch();
			break;

		case OR_SEARCH:
			OrSearch();
			break;

		case AND_SEARCH:
			AndSearch();
			break;

		case GO_BACK:
			return;
		}
	}
}

void SearchMenu() // search 메인 메뉴
{
	system("cls");

	printf("――――――  MENU  ――――――₩n");
	printf(" 1. Normal Search₩n");
	printf(" 2. OR Search₩n");
	printf(" 3. AND Search₩n");
	printf("――――――――――――――――――――₩n");
	printf(" 4. Go Back₩n");
```

```
	printf("──────────────────────────₩n");
	printf("Select》  ");
}

void NormalSearch() // 일반적인 search
{
	char word[20];
	int index;
	tree_node* temp;
	int frequency=0;
	int time=0;

	system("cls");
	puts(" [NORMAL SEARCH]"),puts("");
	fputs(" Write word for searching 》  ", stdout);
	gets(word);
	to_Lower_String(word);
	index = strlen(word) % 5;
	temp = hash_set[index];

	puts("");
	printf(" ┌──────────────────────────────────────────────────────────₩n");
	printf(" │  ▶ Word : %s ₩n", word);
	printf(" │  ▶ Location : ");

	while(temp)
	{
		if(strcmp(temp->word, word)<0)
		{
			temp = temp->right;
			time++;
		}
		else if(strcmp(temp->word, word)>0)
		{
			temp = temp->left;
			time++;
		}
		else
		{
			if(frequency%6 == 0 && frequency!=0)
			{
				printf("₩n │                  ");
			}
			frequency++;
			printf(" [%-2d,%2d]", temp->line, temp->order);
```

```
                temp = temp->left;
                time++;
        }
    }

    if(frequency)
    {
        puts("");
        printf("▏ ▶ Frequency : %d ₩n", frequency);
        printf("▏ ▶ Time Complexity : %d ₩n", time);
        printf(" └──────────────────────────────────────────────────₩n₩n");
    }
    else
    {
        printf(" Not exist₩n");
        printf(" └──────────────────────────────────────────────────₩n₩n");
    }


    system("pause");
}

void OrSearch() // 단어 두개 or search
{
    char instruction[60];
    char *word1;
    char *word2;
    int index;
    tree_node* temp;
    int frequency=0;
    int time=0;

    system("cls");
    puts(" [OR SEARCH]"), puts("");
    fputs(" Write word for searching(word1 word2) 》 ", stdout);
    gets(instruction);
    to_Lower_String(instruction);
    word1 = strtok(instruction, " ");
    word2 = strtok(NULL, " ");

    index = strlen(word1) % 5;
    temp = hash_set[index];

    puts("");
    printf(" ┌──────────────────────────────────────────────────₩n");
```

```
printf("▏ ▶ Word : %s OR %s ₩n", word1, word2);
printf("▏ ▶ Location : ");


while(temp)
{
        if(strcmp(temp->word, word1)<0)
        {
                temp = temp->right;
                time++;
        }
        else if(strcmp(temp->word, word1)>0)
        {
                temp = temp->left;
                time++;
        }
        else
        {
                if(frequency%6 == 0 && frequency!=0)
                {
                        printf("₩n ▏                    ");
                }
                frequency++;
                printf(" [%-2d,%2d]", temp->line, temp->order);
                temp = temp->left;
                time++;
        }
}

index = strlen(word2) % 5;
temp = hash_set[index];

while(temp)
{
        if(strcmp(temp->word, word2)<0)
        {
                temp = temp->right;
                time++;
        }
        else if(strcmp(temp->word, word2)>0)
        {
                temp = temp->left;
                time++;
        }
        else
```

```
		{
			if(frequency%6 == 0 && frequency!=0)
			{
				printf("₩n |                    ");
			}
			frequency++;
			printf(" [%-2d,%2d]", temp->line, temp->order);
			temp = temp->left;
			time++;
		}
	}

	if(frequency)
	{
		puts("");
		printf(" |  ▶ Frequency : %d ₩n", frequency);
		printf(" |  ▶ Time Complexity : %d ₩n", time);
		printf(" └──────────────────────────────────────────────────────────₩n₩n");
	}
	else
	{
		printf(" Not exist₩n");
		printf(" └──────────────────────────────────────────────────────────₩n₩n");
	}

	system("pause");
}

void AndSearch() // 단어 2개 and search
{
	char instruction[60];
	char *word1;
	char *word2;
	int index;
	tree_node* temp;
	tree_node* temp2;
	tree_node* searchPoint;
	int frequency=0;
	int time=0;
	int line;
	int cnt=0;


	system("cls");
	puts(" [AND SEARCH]"), puts("");
```

```
fputs(" Write word for searching(word1 word2) 》  ", stdout);
gets(instruction);
to_Lower_String(instruction);
word1 = strtok(instruction, " ");
word2 = strtok(NULL, " ");
searchPoint = hash_set[strlen(word2) % 5];

index = strlen(word1) % 5;
temp = hash_set[index];

puts("");
printf(" ┌──────────────────────────────────────────────────────────────₩n");
printf(" |  ▶ Word : %s AND %s ₩n", word1, word2);
printf(" |  ▶ Location : ");

while(temp)
{
        if(strcmp(temp->word, word1)<0)
        {
                temp = temp->right;
                time++;
        }
        else if(strcmp(temp->word, word1)>0)
        {
                temp = temp->left;
                time++;
        }
        else
        {
                line = temp->line;
                temp2 = searchPoint;

                while(temp2)
                {
                        if(strcmp(temp2->word, word2)<0)
                        {
                                temp2 = temp2->right;
                                time++;
                        }
                        else if(strcmp(temp2->word, word2)>0)
                        {
                                temp2 = temp2->left;
                                time++;
                        }
                        else
```

```
                {
                        if(line == temp2->line)
                        {
                                frequency++;
                                printf(" [%2d]", temp2->line);
                                temp2 = temp2->left;
                                time++;
                                cnt++;
                                if(cnt==1)
                                {
                                        searchPoint = temp2;
                                }
                                break;
                        }

                        temp2 = temp2->left;
                        time++;
                }
            }

            temp = temp->left;
            time++;
        }
    }

    if(frequency)
    {
        puts("");
        printf(" ┃  ▶ Frequency : %d ₩n", frequency);
        printf(" ┃  ▶ Time Complexity : %d ₩n", time);
        printf(" ┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━₩n₩n");
    }
    else
    {
        printf(" Not exist₩n");
        printf(" ┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━₩n₩n");
    }

    system("pause");
}
```