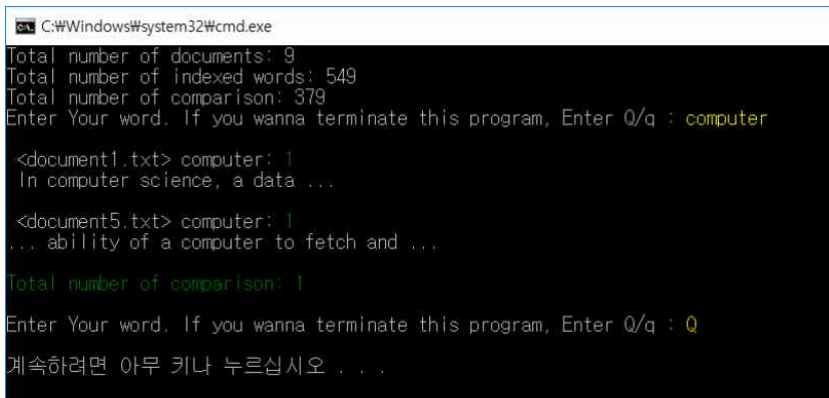


III. 설계과제 최종보고서

설계과제명 : 간단한 검색엔진(Simple Search Engine) 구현

교과목명	자료구조와 실습
담당교수	주해종 교수님
팀 원	2017112080 김혜수
	2017112078 정지연

설계과제 요약서

과 제 요 약 서	
설계과제명	간단한 검색엔진(Simple Search Engine) 구현
주요기술용어 (5~7개 단어)	파일입출력, 해싱, 길이비교, 체이닝
<div>1. 과제 목표</div> <p>주어진 document1~document9의 txt문서들을 읽고, 주어진 제한요소들에 따라 각 파일에 있는 단어들을 색인하고, 그 단어들을 검색 및 출력할 수 있는 간단한 검색엔진을 구현한다. 단, 적절한 자료구조와 알고리즘을 이용하여 가능한 한 효율적인 동작을 하는 프로그램을 만든다.</p> <div>2. 수행 내용 및 방법</div> <p>(1) 주어진 설계과제 지침서를 읽고 과제에서 구현해야 할 기능 및 제한요소 읽어보기.</p> <p>(2) txt 파일 읽고 문자열에 저장. 해싱. 체이닝</p> <ul style="list-style-type: none">- 메뉴 출력하기- 구현 <p>(3) 세부 알고리즘 수정</p> <ul style="list-style-type: none">- 효율을 높이기 위함 (문자열의 길이 고려) <p>(4) 결과 확인 및 분석</p> <p>(5) 보고서 작성</p> <div>3. 수행 결과</div> <div>1. 소문자 입력</div> <div></div>	

2. 대문자 입력

```
Enter Your word. If you wanna terminate this program, Enter Q/q : STRUCTURE

<document6.txt> STRUCTURE: 4
... of a data structure usually requires writing ...
... instances of that structure. The efficiency of ...
... of a data structure cannot be analyzed ...
... type, a data structure that is defined ...

<document1.txt> STRUCTURE: 2
... science, a data structure is a data ...
... precisely, a data structure is a collection ...

<document2.txt> STRUCTURE: 2
... on a data structure and the computational ...
... comparison, a data structure is a concrete ...

<document5.txt> STRUCTURE: 1
... items within the structure itself. Many data ...

<document8.txt> STRUCTURE: 1
... that allows data structure implementations to be ...

Total number of comparison: 1
```

3. 존재하지 않는 단어 입력

```
Enter Your word. If you wanna terminate this program, Enter Q/q : AAAAAAAA

"aaaaaaaa" does not exist.
Enter Your word. If you wanna terminate this program, Enter Q/q :
```

4. 결과 분석

주어진 제한요소를 고려한 문서의 색인과 검색, 출력이 정상적인 것을 확인했으며, 단어의 길이를 먼저 비교하면서 무의미한 비교를 최소화하여 효율성을 높일 수 있었다. 또한 정해지지 않은 입력이 주어지는 경우는 입력을 다시 받도록 구현하여 입력 실수로 인한 시간낭비를 없앴다.

제 1 장 서론

제 1 절 설계과제 목적

구상한 자료구조를 이용하여 ASCII코드로 이루어진 txt파일을 읽어와 해당 파일의 영단어들을 색인하고, 단어를 검색할 수 있는 검색엔진을 주어진 제한요소들을 고려하여 구현해 본다.

제 2 절 설계과제 내용

(1) 기능

이 검색엔진은 크게 파일입출력, 색인, 검색 이렇게 세 가지 기능을 갖고 있다. 실행의 편의를 위해 while(1)을 사용하여 입력을 계속 받는다.

(2) 성능

단어를 비교하는 횟수를 최소화한 것이 가장 효율적인 알고리즘이기 때문에 이 횟수를 최소화하는 것을 중점으로 생각한다. 단, 입력파일이 9개가 아닐 경우 혹은 단어의 총 개수가 다른 경우 등의 주어지지 않은 제한요소에 관한 부분은 임의로 설정한다.

(3) 제한요소

단어를 나누는 기준은 공백(" ")이다.

단어는 영문자로만 이루어져 있다.(그러므로 검색 시 영어만 입력받는다)

검색할 때 영어의 대소문자는 구분하지 않는다.

검색할 때 검색단어를 포함한 단어(예를 들어 data를 검색하면 database는 검색 x)

색인 시 총 문서의 개수, 색인된 단어의 개수, 비교횟수를 출력하도록 한다.

검색 시 검색된 문서명, 검색단어를 포함한 전후 3단어, 비교횟수를 출력하도록 한다.

또한 단어가 가장 많이 존재하는 문서부터 검색 결과를 출력한다.

(4) 파일입출력

C언어의 표준 파일 입출력을 사용한다.

(5) 색인과정

받아온 파일에서 모든 알파벳들을 소문자로 바꾸고, 공백이 단어와 단어사이의 구분자이므로 공백을 기준으로 하여 한 묶음으로 만든다.

묶은 것의 맨 뒤에 '\0'을 붙이고, 각 단어들을 해싱으로 색인한다.

동시에 단어의 길이, 문서 내 단어의 위치, 중복여부, 비교횟수 등을 저장한다.

해시 충돌이 일어날 경우 체이닝을 이용해 단어를 동적할당한다.

결과를 출력한다.

(6) 검색과정

검색할 단어를 입력받는다(양식에 어긋날 경우 예외처리)

입력받은 단어를 모두 소문자로 바꾸고 해시코드를 만든다.

코드에 해당하는 해시테이블을 찾는다.(이 때, 문자열을 비교하기 전에 길이를 먼저 비교한다.)

단어의 길이가 같다면 해당 linked-list의 단어와 strcmp함수를 이용하여 비교한다.

같은 단어일 경우는 다음 다른 단어가 나올 때 까지 연속된 노드를 출력한다.

다른 단어일 경우 단어길이 비교부터 다시 수행한다.

검색을 완료한 후, 색인 시 저장한 단어출현빈도를 통해 출력 우선순위를 만들고,

우선순위에 따라 출력한다.

(7) 예외처리

파일을 불러올 수 없는 경우

검색할 단어가 없는 경우

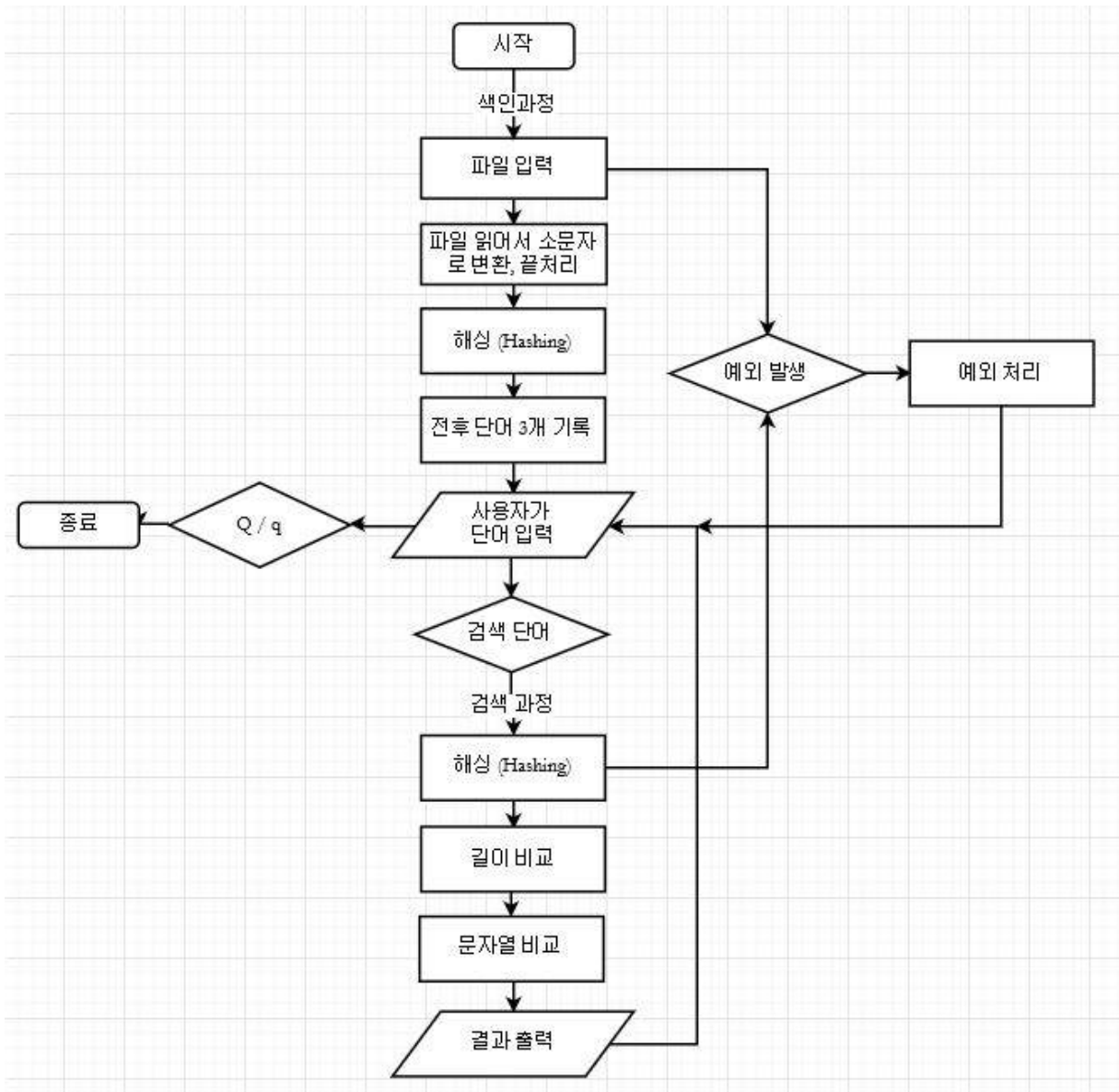
진행을 중단한 뒤, 오류를 출력하고 콘솔메뉴로 복귀한다.

제 3 절 진행 일정 및 개인별 담당분야

성명	담당 분야	참여도(%)
김혜수	아이디어 도출. 알고리즘 구상. 프로그램 코딩. 설계계획서 작성. 설계결과보고서 작성.	50%
정지연	아이디어 도출. 알고리즘 구상. 프로그램 코딩. 설계계획서 작성. 설계결과보고서 작성.	50%

제 2 장 프로그램의 구성

제 1 절 전체 구성도



제 2 절 프로그램 세부 구성

<색인>

파일을 불러와 파일에 있는 문장들을 한 글자씩 읽어와 공백마다 하나의 단어로 만든다. 이때, 단어의 끝을 알리기 위해 단어의 마지막 부분에는 NULL값을 저장해준다. 그리고 단어를 저장하기 전에 읽은 단어를 hash 함수를 통해 key값을 부여하고 해시테이블에 저장해준다.

저장된 단어는 해싱 과정을 거치게 된다. 단어의 길이가 같거나 다음 테이블이 비어있을 때까지 다음 테이블로 이동하고, 단어가 채워지고 같은 단어가 아닐 때까지 비교횟수의 연산이 일어난다. 이 과정 중에서 충돌 시 동적 할당이 진행되며 이때 비교 횟수가 증가된다.

그리고 단어 위치를 플레이스에 기록한다. 파일 내에 같은 단어가 있다면 해당 단어는 연속된 노드의 끝에 저장되며 다른 단어는 노드의 마지막 위치에 삽입된다.

<검색>

입력받은 단어를 임시로 저장한 뒤, 임시로 저장한 단어의 끝까지 소문자는 소문자 그대로, 대문자는 소문자로 변경해준다. 그리고 역시 임시로 저장한 단어의 끝에는 NULL 값을 부여한다.

단어는 해시 함수를 통해 key값으로 검색이 되는데, 테이블에서 맞는 단어를 찾을 때 우선, 글자 수가 맞지 않으면 비교 자체가 이루어지지 않는다. 이 과정에서는 길이가 같을 때까지 링크를 계속 뒤로 밀면서 글자 수가 같은지 같지 않은지, 판단한다.

테이블에서 찾는 단어가 없을 경우 다음 테이블을 탐색하고 이때 비교횟수가 증가한다. 그리고 같은 문자열이 있는 경우, 해당 노드부터 같은 결과의 노드를 모두 출력한다. 그리고 찾는 단어가 많이 등장하는 파일부터 출력하기 위해 파일들에 우선순위를 부여한다.

제 3 장 결과 및 토의

제 1 절 프로그램 테스트 결과

1. 소문자 입력

```
C:\Windows\system32\cmd.exe
Total number of documents: 9
Total number of indexed words: 549
Total number of comparison: 379
Enter Your word. If you wanna terminate this program, Enter Q/q : computer

<document1.txt> computer: 1
In computer science, a data ...

<document5.txt> computer: 1
... ability of a computer to fetch and ...

Total number of comparison: 1
Enter Your word. If you wanna terminate this program, Enter Q/q : Q
계속하려면 아무 키나 누르십시오 . . .
```

2. 대문자 입력

```
Enter Your word. If you wanna terminate this program, Enter Q/q : STRUCTURE

<document6.txt> STRUCTURE: 4
... of a data structure usually requires writing ...
... instances of that structure. The efficiency of ...
... of a data structure cannot be analyzed ...
... type, a data structure that is defined ...

<document1.txt> STRUCTURE: 2
... science, a data structure is a data ...
... precisely, a data structure is a collection ...

<document2.txt> STRUCTURE: 2
... on a data structure and the computational ...
... comparison, a data structure is a concrete ...

<document5.txt> STRUCTURE: 1
... items within the structure itself. Many data ...

<document8.txt> STRUCTURE: 1
... that allows data structure implementations to be ...

Total number of comparison: 1
```


3. 존재하지 않는 단어 입력

```
Enter Your word. If you wanna terminate this program, Enter Q/q : AAAAAAAA
"aaaaaaa" does not exist.
Enter Your word. If you wanna terminate this program, Enter Q/q :
```

제 2 절 수행 결과에 대한 토의

1. '색인과정'에 대한 토의 결과

	분석	평가
단어 생성	파일의 내용을 받아오고 문자열로 만드는 과정이 순차 탐색의 알고리즘을 따르기 때문에 파일의 크기가 커진다면 느려질 수 있다.	9 / 10
비교 횟수	해싱과 chaining을 이용했기 때문에 최악의 경우에도 $O(n)$ 의 시간 복잡도를 갖는다.	10 / 10
확장성	한글과 같은 전각문자의 경우 'ASCII코드'가 아니므로 확장성에 분명한 한계가 존재한다.	8 / 10
메모리	구조체를 여러 개 사용하고, 그 안에 들어가는 내용 또한 상대적으로 많기 때문에 공간 복잡도는 다소 떨어진다.	6 / 10

2. '검색과정'에 대한 토의 결과

	분석	평가
예외 처리	대부분의 제한요소에 해당하는 경우에 대해 처리가 가능하다. 확장성에 따라서도 코드를 수정할 수 있다.	10 / 10
비교 횟수	해싱과 chaining을 이용했기 때문에 최악의 경우에도 $O(n)$ 의 시간 복잡도를 갖는다. 이 때, 해시 테이블 내의 노드를 모두 비교하는 것이 아니라 단어의 길이를 먼저 확인하기 때문에 이론적으로 $O(n)$ 보다 훨씬 더 낮은 시간 복잡도를 갖는다.	10 / 10
확장성	한글과 같은 전각문자의 경우 'ASCII코드'가 아니므로 확장성에 분명한 한계가 존재한다.	8 / 10
메모리	구조체를 여러 개 사용하고, 그 안에 들어가는 내용 또한 상대적으로 많기 때문에 공간 복잡도는 다소 떨어지는 편이다.	6 / 10

제 3 절 기타

<구현하고 싶었지만 하지 못한 부분>

현재 검색엔진에서 단어에 특수문자가 포함될 경우 검색이 제대로 수행되지 않는 경우가 있다.

이를 isalpha()함수를 이용해서 단어를 색인할 때 알파벳 외의 문자는 모두 삭제하고 저장하려고 의도했으나 런타임 오류와 시간부족 인하여 완성하지 못했다.

제 4 장 부록

<project.c>

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <windows.h>
#include "project.h"

void main()
{
    char search_word[50];

    build("document1.txt");
    build("document2.txt");
    build("document3.txt");
    build("document4.txt");
    build("document5.txt");
    build("document6.txt");
    build("document7.txt");
    build("document8.txt");
    build("document9.txt");
    printf("Total number of documents: %d \n", indexed_doc);
    printf("Total number of indexed words: %d \n", indexed_num);
    printf("Total number of comparison: %d \n", comp);

    while (1)
    {
        printf("Enter Your word. If you wanna terminate this program, Enter Q/q : ");
        textcolor(14);
        scanf("%s", search_word);
        textcolor(7);

        if (strcmp(search_word, "Q") == 0 || strcmp(search_word, "q") == 0) {
            printf("\n");
            exit(1);
        }
        else {
```

```

        search(search_word);
    }
}
return 0;
}

```

```

char to_small(char *str)
{
    if (96 < *str && *str < 123)
        return *str;
    else if (64 < *str && *str < 91)
        return *str + 32;
    else return NULL;
}

```

```

void create_table(hash_link *hlp, char *alphabat)
{
    int i = 0;
    hash_link n_table = (hash_link)malloc(sizeof(hash_table));
    n_table->next_tablelink = NULL;
    n_table->strlen = 0;

    for (i = 0; i < 10; i++) {
        n_table->freq[i] = 0;
        n_table->existence[i] = false;
        n_table->place[i].rear_cursor = 0;
    }

    strcpy(n_table->word, alphabat);
    ((*hlp)->next_tablelink) = n_table;
    (*hlp) = n_table;
}

```

```

void create_place(place_link *wtp)
{
    place_link n_place = (place_link)malloc(sizeof(place));
    (*wtp)->next_place = n_place;
    n_place->next_place = NULL;
    (*wtp) = n_place;
    (*wtp)->rear_cursor = 0;
}

```

```

int hash(char *key)
{
    return(transform(key) % TABLE_SIZE);
}

int transform(char *key)
{
    int number = 0;
    while (*key) {
        number += *key++;
    }

    return number;
}

void search(char* alphabet)
{
    hash_link hlp;
    int priority[10];
    boolean table_check[10] = { 0 };
    int i = 0, j = 0, encl_word = indexed_doc;
    char temp[100];
    comp = 0;
    strcpy(temp, alphabet);

    while (temp[i] != '\0') {
        alphabet[j++] = to_small(temp + i);
        i++;
    }
    alphabet[j] = '\0';

    hlp = &table[hash(alphabet)];

    while (hlp->strlen != j && hlp->next_tablelink) {
        hlp = hlp->next_tablelink;
        comp++;
    }

    while (strcmp(hlp->word, alphabet) != 0) {
        if (!hlp->next_tablelink) {
            textcolor(2);
            printf("\n\"%s\" does not exist.\n", alphabet);

```

```

        textcolor(7);
        return;
    }

    hlp = hlp->next_tablelink;
    comp++;
}

for (i = 0; i < indexed_doc; i++) {
    int max_top = INT_MAX;
    int top, top_index;
    top = 0;

    for (j = 0; j < indexed_doc; j++) {
        if ((hlp->freq[j] > top) && table_check[j] == 0) {
            top = hlp->freq[j];
            top_index = j;
        }
    }

    if (hlp->freq[i] == 0)
        encl_word--;

    priority[i] = top_index;
    table_check[top_index] = 1;
    max_top = top;
}

//printf("\n");
for (i = 0; i < encl_word; i++) {
    place_link plp = &hlp->place[priority[i]];
    FILE *ifp = fopen(doc_name[priority[i]], "r");
    char p_word[100];
    printf("\n <%s> %s: ", doc_name[priority[i]], temp);
    textcolor(2);
    printf("%d\n", hlp->freq[priority[i]]);
    textcolor(7);

    while (1) {
        int s = 0;

        fseek(ifp, plp->front_cursor, SEEK_SET);
        if (plp->front_cursor != 0)

```

```

        printf("...");
    for (j = 0; j < 9; j++) {
        if (fscanf(ifp, "%s", p_word) == 0)
            break;

        printf(" %s", p_word);

        long ft = ftell(ifp);
        if (ftell(ifp) >= plp->rear_cursor)
            break;
    }
    if (fscanf(ifp, "%s", p_word) > 0)
        printf(" ...");

    printf("\n");

    if (plp->next_place != NULL)
        plp = plp->next_place;
    else
        break;
}

fclose(ifp);
}

textcolor(2);
printf("\nTotal number of comparison: %d\n\n", comp);
textcolor(7);
}

```

```

void build(char* fname)
{
    FILE *ifp;
    int file_pointer[4] = { 0 };
    int file_pointer_index = 4;
    int i;
    place_link cursor[4] = { 0 };
    int cursor_index = 4;
    hash_link hlp;
    place_link plp;
    long ft = 0;
    char string[100];

```

```

char alphabet[100];

if (indexed_doc == 10) {
    return;
}

strcpy(doc_name[indexed_doc], fname);

if ((ifp = fopen(fname, "r")) == NULL) {
    printf("\nFiles not founded\n", fname);
    return;
}

while (fscanf(ifp, "%s", string) == 1) {
    indexed_num++;
    int i, j;
    i = j = 0;

    while (string[i] != '\0') {
        alphabet[j++] = to_small(string + i);
        if (alphabet[j - 1] == NULL) {
            j--;
        }

        i++;
    }

    if (!j) continue;
    alphabet[j] = '\0';

    hlp = &table[hash(alphabet)];
    while (hlp->strlen != j && hlp->next_tablelink) {
        hlp = hlp->next_tablelink;
    }
    comp++;

    while (hlp->strlen == 0 || strcmp(alphabet, hlp->word) != 0) {
        if (hlp->strlen == 0) {
            comp--;
        }

        if (!(hlp->next_tablelink)) {

```

```

        create_table(&hlp, alphabet);
        break;

    }

    hlp = hlp->next_tablelink;
    comp++;
}

hlp->freq[indexed_doc]++;
hlp->existence[indexed_doc] = true;
hlp->strlen = j;
plp = &hlp->place[indexed_doc];

while (plp->rear_cursor != 0) {
    if (plp->next_place)
        plp = plp->next_place;
    else
        break;
}

if (plp->rear_cursor != 0) {

    create_place(&plp);
}

plp->next_place = NULL;
cursor[cursor_index++ % 4] = plp;
plp->front_cursor = file_pointer[(file_pointer_index - 4) % 4];

if (cursor_index > 7)
    cursor[cursor_index % 4]->rear_cursor = ftell(ifp);
file_pointer[file_pointer_index++ % 4] = ftell(ifp);
}

for (i = 0; i < 4; i++){
    cursor[cursor_index++ % 4]->rear_cursor = ftell(ifp);
}

indexed_doc++;
fclose(ifp);
}

```



```
void textcolor(int color_number)
{
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
color_number);
}
```

<project.h>

```
#pragma once
#define _CRT_NO_SECURE_WARNINGS
#define boolean unsigned char
#define true 1
#define false 0
#define TABLE_SIZE 20000

int comp = 0;
int indexed_num = 0;
int indexed_doc = 0;
char doc_name[10][100];

typedef struct place *place_link;
typedef struct place
{
    int front_cursor;
    int rear_cursor;
    place_link next_place;
}place;

typedef struct hash_table *hash_link;
typedef struct hash_table
{
    int strlen;
    int freq[10];
    boolean existence[10];
    char word[100];

    place place[10];
    hash_link next_tablelink;
}hash_table;
hash_table table[TABLE_SIZE];

void textcolor(int color_number);
void build(char *fname);
char to_small(char *str);
int hash(char *key);
int transform(char *key);
void create_table(hash_link *hlp, char *alphabet);
void create_place(place_link *wtp);
void search(char *alphabet);
```