



순천향대학교  
SOON CHUN HYANG  
UNIVERSITY

# 자료구조2 실습 14주차

## 자료구조2 실습

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 12월 07일

# | 목 차 |

## 1. 보간 탐색 구현

- 1.1 문제 분석
- 1.2 소스 코드
- 1.3 소스 코드 분석
- 1.4 실행 결과
- 1.5 느낀점

## 2. AVL 트리 구현

- 2.1 문제 분석
- 2.2 소스 코드
- 2.3 소스 코드 분석
- 2.4 실행 결과
- 2.5 느낀점

## 3. 선형 조사법으로 해시 테이블 구현

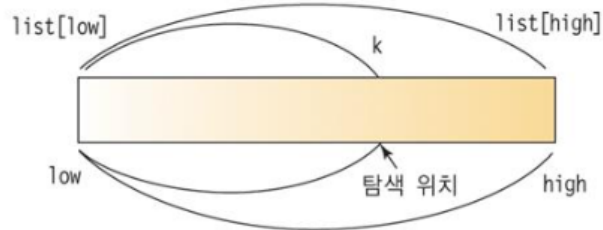
- 3.1 문제 분석
- 3.2 소스 코드
- 3.3 소스 코드 분석
- 3.4 실행 결과
- 3.5 느낀점

## 1.1 문제 분석

### ■ 보간 탐색 구현

- data.txt에서 데이터를 읽어와 509페이지의 프로그램 13.7의 보간 탐색 프로그램을 아래와 같이 실행되도록 구현하시오.

$(list[high] - list[low]) : (k - list[low]) = (high - low) : \text{탐색 위치} - low$



$$\text{탐색 위치} = \frac{(k - list[low])}{list[high] - list[low]} * (high - low) + low$$

cmd C:\WINDOWS\system32\cmd.exe

```
데이터 개수: 10000000
찾고자 하는 정수를 입력하세요 : 6666666
탐색 성공
6666667 번째에 저장되어 있음
보간탐색 실행 속도 : 0.000000
계속하려면 아무 키나 누르십시오 . . .
```

이 문제는 보간 탐색 식을 이용해 10000000 개의 숫자 데이터를 입력한 후 보간 탐색을 해 데이터를 찾는 문제이다. 가장 중요한 부분은 보간 탐색 식을 이용해서 데이터를 찾는 문제이다. 여기다가 보간 탐색 실행 속도도 측정을 해야한다.

우선 보간 탐색 계산 함수를 만들어서 키 값을 가지고 위에 있는 식을 이용해 key값이 작을 경우는 low값 증가 클 경우 high 값을 감소 시키는 함수를 만들어서 만약 있을 경우 값을 반환하고 없으면 그냥 -1을 보내 탐색 실패를 한다.

메인 함수에서 시간 변수와 int형 리스트를 만들어 파일의 크기만큼 동적할당을 시켜 데이터를 넣는다. 그리고 시간을 측정한 후 시간을 출력하고 값을 탐색한 후 종료한다.

## 1.2 소스 코드

```
1 //=====
2 // 제작기간: 21년 11월 30일 ~ 12월 6일
3 // 제작자: 20204005 김필중
4 // 프로그램명: 보안 탐색 구현
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <time.h>
11
12 // 오류 방지 구문을 설정한다
13 #pragma warning(disable : 4996)
14
15 // 보안 탐색 계산 함수
16 int search_interpolation(int key, int n, int list[])
17 {
18     int low, high, j; // high, low, j 변수 선언
19     low = 0; // low는 0
20     high = n - 1; // high는 n-1로
21     while ((list[high] >= key) && (key > list[low])) // high번째 값이 키값 이상 그리고 low번째 값이 키값 보다 작을 경우 반복을 한다
22     {
23         j = ((float)(key - list[low]) / (list[high] - list[low]) * (high - low)) + low; // j값에 탐색 위치 공식을 계산해준다.
24         if (key > list[j]) // 만약 j번째 값이 key값 보다 작을 경우
25             low = j + 1; // low값은 j + 1을 해준다
26         else if (key < list[j]) // j번째 값이 key값 보다 클 경우
27             high = j - 1; // high값은 j - 1을 해준다
28         else // 만약 같을 경우
29             low = j; // low값을 j값을 설정한다
30     }
31     if (list[low] == key) // 만약 low번째 값이 key값과 동일하면
32         return(low); // 반환
33     else // 아닐경우 실패
34         return -1;
35 }
36
```

```

37 // 메인 함수
38 int main(void)
39 {
40     FILE *fp; // 파일 포인터를 선언한다
41     int *list; // int형 list 포인터를 선언한다
42     // 필요한 변수들을 선언한다
43     int i = 0;
44     int cnt = 0;
45     int tmp;
46     // 시간 측정 변수
47     clock_t start_1, end_1;
48     float res;
49
50     // 파일을 열고 실패 시 종료한다
51     fp = fopen("data01.txt", "r");
52
53     if (fp == NULL)
54     {
55         printf("파일 오픈 실패\n");
56         return 0;
57     }
58
59     // 파일의 개수를 카운트 한다.
60     while (!feof(fp))
61     {
62         fscanf(fp, "%d", &tmp);
63         cnt++;
64     }
65
66     rewind(fp); // 파일 포인터를 처음으로 돌린다
67
68     // list를 파일의 개수만큼 동적할당을 한다
69     list = (int *)malloc(sizeof(int) * cnt);
70
71     // 개수 만큼 반복한다
72     for (int n = 0; n < cnt; n++)
73         fscanf(fp, "%d ", &list[n]); // 동적할당한 공간에 파일의 값들을 집어넣는다
74
75     printf("데이터의 개수 : %d\n", cnt);
76     printf("찾고자 하는 정수를 입력하십시오: ");
77     scanf("%d", &tmp);
78
79     start_1 = clock(); // 측정 시작
80     i = search_interpolation(tmp, cnt, list); // 함수 호출
81     end_1 = clock(); // 측정 종료
82
83     res = (float)(end_1 - start_1) / CLOCKS_PER_SEC; // 시간 계산
84

```

```
85     if (i >= 0)
86     {
87         printf("탐색 성공\n%d 번째에 저장되어있음\n", i);
88     }
89     else
90     {
91         printf("탐색 실패\n");
92     }
93
94     printf("보간탐색 실행 속도 : %f\n", res);
95
96     // 파일을 닫고 동적할당을 해제시키고 종료한다
97     fclose(fp);
98     free(list);
99     return 0;
100 }
101
102
103
```

## 1.3 소스 코드 분석

```
//=====
// 제작기간: 21년 11월 30일 ~ 12월 6일
// 제작자: 20204005 김필중
// 프로그램명: 보간 탐색 구현
//=====

// 필요한 헤더파일을 선언한다
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 오류 방지 구문을 설정한다
#pragma warning(disable : 4996)
```

1. 필요한 헤더파일을 선언한다

2. 오류 방지 구문을 설정한다

```
// 보간 탐색 계산 함수
int search_interpolation(int key, int n, int list[])
{
    int low, high, j; // high, low, j 변수 선언
    low = 0; // low는 0
    high = n - 1; // high는 n-1로
    while ((list[high] >= key) && (key > list[low])) // high번째 값이 키값 이상 그리고 low번째 값이 키값 보다 작을 경우 반복을 한다
    {
        j = ((float)(key - list[low]) / (list[high] - list[low]) * (high - low)) + low; // j값에 탐색 위치 공식을 계산해준다.
        if (key > list[j]) // 만약 j번째 값이 key값 보다 작을 경우
            low = j + 1; // low값은 j + 1을 해준다
        else if (key < list[j]) // j번째 값이 key값 보다 클 경우
            high = j - 1; // high값은 j - 1을 해준다
        else // 만약 같을 경우
            low = j; // low값을 j값을 설정한다
    }
}
```

3. 보간 탐색 계산 함수를 선언한다

4. 우선 high, low, j 변수를 선언하고 low는 0, high는 크기 - 1 값으로 정한다

5. high번째 값이 키 값 이상이고 low번째 값이 키값보다 작을 경우에 계속 반복한다

6. j에 보간 탐색 식을 이용해 계산한 값을 저장한다

7. 만약 j번째 배열의 값이 키값보다 크다면 low 값을 j + 1로 저장한다

8. 반대라면 high 값을 j-1로 해준다.

9. 만약 같을 경우는 low값을 j값으로 저장한다

```

if (list[low] == key) // 만약 low번째 값이 key값과 동일하면
    return(low); // 반환
else // 아닐경우 실패
    return -1;

```

10. 만약 low번째 값과 key값이 동일하다면 low를 반환한다

11. 아닐 경우는 값이 없으므로 -1을 반환한다

```

// 메인 함수
int main(void)
{
    FILE *fp; // 파일 포인터를 선언한다
    int *list; // int형 list 포인터를 선언한다
    // 필요한 변수들을 선언한다
    int i = 0;
    int cnt = 0;
    int tmp;
    // 시간 측정 변수
    clock_t start_1, end_1;
    float res;

```

12. 메인함수에서 파일 포인터와 int형 list 포인터를 선언한다

13. 필요한 변수들을 선언한다

14. 시간 측정 변수를 선언한다

```

// 파일을 열고 실패 시 종료한다
fp = fopen("data01.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return 0;
}

// 파일의 개수를 카운트 한다.
while (!feof(fp))
{
    fscanf(fp, "%d", &tmp);
    cnt++;
}

```

15. 파일을 열고 실패하면 종료한다



16. 파일의 개수를 카운트를 하기위해 파일 끝까지 반복한다

```
rewind(fp); // 파일 포인터를 처음으로 돌린다

// list를 파일의 개수만큼 동적할당을 한다
list = (int *)malloc(sizeof(int) * cnt);
```

17. 파일 포인터를 처음으로 돌린다.

18. list를 카운트한 개수만큼 동적할당을 한다

```
// 개수 만큼 반복한다
for (int n = 0; n < cnt; n++)
    fscanf(fp, "%d ", &list[n]); // 동적할당한 공간에 파일의 값들을 집어넣는다

printf("데이터의 개수 : %d\n", cnt);
printf("찾고자 하는 정수를 입력하십시오: ");
scanf("%d", &tmp);

start_1 = clock(); // 측정 시작
i = search_interpolation(tmp, cnt, list); // 함수 호출
end_1 = clock(); // 측정 종료

res = (float)(end_1 - start_1) / CLOCKS_PER_SEC; // 시간 계산
```

19. 개수 만큼 반복하면서 동적할당한 list에 값들을 저장한다

20. 측정을 시작하고 바로 입력받은 값을 넣고 탐색 호출을 한다

21. 탐색이 끝나면 측정을 종료하고 시간 계산을 한다

```
if (i >= 0)
{
    printf("탐색 성공\n%d 번째에 저장되어있음\n", i);
}
else
{
    printf("탐색 실패\n");
}

printf("보간탐색 실행 속도 : %f\n", res);

// 파일을 닫고 동적할당을 해제시키고 종료한다
fclose(fp);
free(list);
return 0;
```

22. i값이 0이 아니라면 저장된 위치를 출력하고 아니면 탐색 실패를 한다

23. 파일을 닫고 동적할당을 해제하고 종료한다.

## 1.4 실행 결과

Microsoft Visual Studio 디버그 콘솔

데이터의 개수 : 10000000  
찾고자 하는 정수를 입력하시오: 7777777  
탐색 성공  
7777777 번째에 저장이 되어있음  
보간탐색 실행 속도 : 0.000000

C:\Users\korca\Desktop\20204005\_김필중\_자료구조실습\_14주차 소  
xe(15264 프로세스)이(가) 0 코드로 인해 종료되었습니다.  
이 창을 닫으려면 아무 키나 누르세요.

data01 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

0 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 :  
10 311 312 313 314 315 316 317 318 319 320 321 322 323  
66 567 568 569 570 571 572 573 574 575 576 577 578 579  
22 823 824 825 826 827 828 829 830 831 832 833 834 835  
062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072  
1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 127  
1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 148  
1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 168  
1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 189  
2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 209  
2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 230  
2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 250  
2702 2703 2704 2705 2706 2707 2708 2709 2710 2711 271  
2907 2908 2909 2910 2911 2912 2913 2914 2915 2916 291  
3112 3113 3114 3115 3116 3117 3118 3119 3120 3121 312  
3317 3318 3319 3320 3321 3322 3323 3324 3325 3326 332  
3522 3523 3524 3525 3526 3527 3528 3529 3530 3531 353  
3727 3728 3729 3730 3731 3732 3733 3734 3735 3736 373  
3932 3933 3934 3935 3936 3937 3938 3939 3940 3941 394  
4137 4138 4139 4140 4141 4142 4143 4144 4145 4146 414  
A2A7 A2A8 A2A9 A2AA A2AB A2AC A2AD A2AE A2AF A2B0 A2B1 A2B2 A2B3 A2B4 A2B5 A2B6 A2B7 A2B8 A2B9 A2BA A2BB A2BC A2BD A2BE A2BF A2C0 A2C1 A2C2 A2C3 A2C4 A2C5 A2C6 A2C7 A2C8 A2C9 A2CA A2CB A2CC A2CD A2CE A2CF A2D0 A2D1 A2D2 A2D3 A2D4 A2D5 A2D6 A2D7 A2D8 A2D9 A2DA A2DB A2DC A2DD A2DE A2DF A2E0 A2E1 A2E2 A2E3 A2E4 A2E5 A2E6 A2E7 A2E8 A2E9 A2EA A2EB A2EC A2ED A2EE A2EF A2F0 A2F1 A2F2 A2F3 A2F4 A2F5 A2F6 A2F7 A2F8 A2F9 A2FA A2FB A2FC A2FD A2FE A2FF

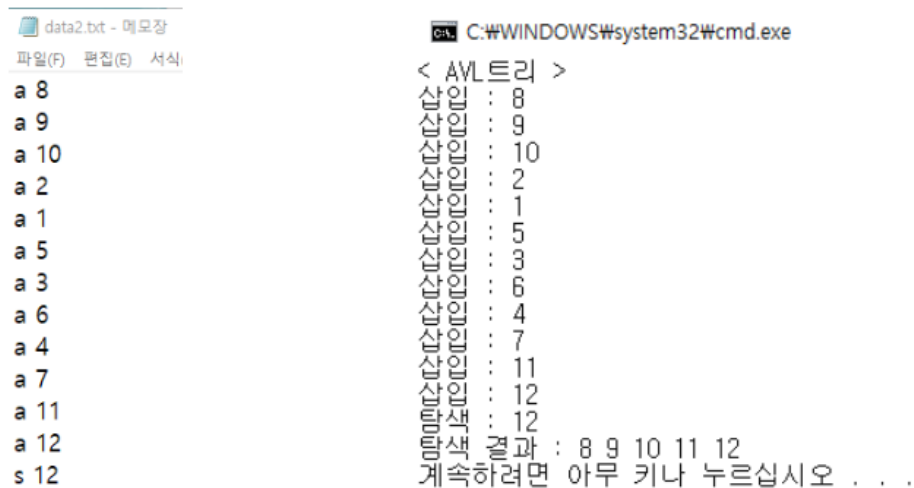
## 1.5 느낀점

처음에 보간탐색을 보고 나서 식을 보고 복잡하겠다 라는 생각을 했었다. 하지만 실제로 코드를 구현하고 나니 식만 잘 만들고 조건들만 만족하게 하면 금방 구현이 가능하다는 사실을 알게되었다. 보간 탐색의 큰 특징은 탐색이 앞에 있으면 앞에서 찾고 이렇게 단번에 못찾아도 탐색의 위치가 찾는 데이터와 가까워서 탐색 대상을 줄이는 속도가 빠르다. 그래서 보간 탐색도 여러곳에서 쓰일 수도 있다는 생각을 하게 되었다.

## 2.1 문제 분석

### ■ AVL 트리 구현

- 520 페이지에 있는 프로그램 13.11의 AVL 트리 프로그램을 구현 하시오. 이때 데이터는 data.txt에서 읽어오며 a는 add(삽입) 그리고 s는 search(탐색)이다.



```
data2.txt - 메모장
파일(F) 편집(E) 서식
a 8
a 9
a 10
a 2
a 1
a 5
a 3
a 6
a 4
a 7
a 11
a 12
s 12

C:\WINDOWS\system32\cmd.exe
< AVL트리 >
삽입 : 8
삽입 : 9
삽입 : 10
삽입 : 2
삽입 : 1
삽입 : 5
삽입 : 3
삽입 : 6
삽입 : 4
삽입 : 7
삽입 : 11
삽입 : 12
탐색 : 12
탐색 결과 : 8 9 10 11 12
계속하려면 아무 키나 누르십시오 . . .
```

이번 과제는 avl 트리를 구현해서 탐색을 하는 문제이다. 스스로 트리의 균형을 잡는 이진 트리이므로 계속해서 회전하는게 특징이다. 또한 트리를 만든 후 탐색을 구현해야 한다.

먼저 avl 트리 노드를 정의해서 값과 링크 구성을 한다

그리고 트리 높이 반화나 함수를 만들어서 트리 높이를 반환하는 함수를 만들어야 한다.

노드의 균형 인수를 반환하는 함수를 만들어 트리 높이의 차를 반환하는 함수를 만들어야 한다.

동적 노드를 생성하는 함수를 만들어서 노드를 동적할당을 한 후 키값을 대입하고 링크 연결을 한 후 반환하는 함수를 만들어야 한다.

오른쪽 왼쪽을 돌리는 회전 함수를 만들어서 조건에 부합한다면 회전을 하는 함수를 만들어 균형을 맞춰주는 함수를 만들어야 한다.

avl트리에 노드를 추가하는 함수를 만들어서 조건을 다 고려해서 트리가 왼쪽으로 회전해야한다면 왼쪽 호출, 오른쪽 호출을 조건을 만들어서 삽입되고 난 후 트리의 균형이 깨진다면 조건을 통해 트리의 균형을 유지하는 함수를 만든다.

검색 함수를 만들어서 원하는 값을 탐색하면서 탐색을 하면서 방문하는 노드를 출력하는 함수를 만든다.

메인 함수에는 루트를 선언하고 파일 포인터를 선언하고 파일을 읽어서 앞의 값이 a라면 삽입 s라면 탐색하는 반복문을 만든다. 그리고 s값을 저장한 후 탐색 결과를 호출한다.

그 후 값을 출력하고 파일을 닫고 종료한다.

## 2.2 소스 코드

```
1 //=====
2 // 제작기간: 21년 11월 30일 ~ 12월 6일
3 // 제작자: 20204005 김필중
4 // 프로그램명: AVL트리 구현
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include<stdio.h>
9 #include<stdlib.h>
10 #include <string.h>
11
12 // 오류 방지 구문
13 #pragma warning(disable : 4996)
14
15 // max 매크로 선언
16 #define MAX(a, b) (a)
17
18
19 // AVL 트리 노드 정의
20 typedef struct AVLNode
21 {
22     int key; // 키값
23     struct AVLNode *left; // 왼쪽 링크 구성
24     struct AVLNode *right; // 오른쪽 링크 구성
25 } AVLNode;
26
27 // 트리의 높이를 반환
28 int get_height(AVLNode *node)
29 {
30     int height = 0; // 높이값 0으로 선언
31
32     if (node != NULL) // 노드가 null이 아니라면
33         height = 1 + max(get_height(node->left), get_height(node->right)); // 1+ 왼쪽 노드, 오른쪽 노드 높이 호출해서 큰 값 이용
34
35     return height; // 높이 반환
36 }
37
38 // 노드의 균형인수를 반환
39 int get_balance(AVLNode* node)
40 {
41     if (node == NULL) // 노드가 null이라면 0 반환
42         return 0;
43
44     return get_height(node->left) - get_height(node->right); // 왼쪽 높이 - 오른쪽 높이 반환
45 }
46
```

```

47 // 노드를 동적으로 생성하는 함수
48 AVLNode* create_node(int key)
49 {
50     AVLNode* node = (AVLNode*)malloc(sizeof(AVLNode)); // 노드 동적할당
51     node->key = key; // 키값을 대입
52     node->left = NULL; // 왼쪽 링크 null로 연결
53     node->right = NULL; // 오른쪽 링크 null로 연결
54     return(node); // 노드 반환
55 }
56
57 // 오른쪽으로 회전시키는 함수
58 AVLNode *rotate_right(AVLNode *parent)
59 {
60     AVLNode* child = parent->left; // child를 부모 노드의 왼쪽에 연결
61     parent->left = child->right; // 부모의 왼쪽을 자식의 오른쪽과 연결
62     child->right = parent; // 자식의 오른쪽을 부모를 연결
63
64     // 새로운 루트를 반환
65     return child;
66 }
67
68 // 왼쪽으로 회전시키는 함수
69 AVLNode *rotate_left(AVLNode *parent)
70 {
71     AVLNode *child = parent->right; // child를 부모 노드의 오른쪽에 연결
72     parent->right = child->left; // 부모의 오른쪽을 자식의 왼쪽과 연결
73     child->left = parent; // 자식의 왼쪽을 부모를 연결
74
75     // 새로운 루트 반환
76     return child;
77 }
78
79 // AVL 트리에 새로운 노드 추가 함수
80 // 새로운 루트를 반환한다.
81 AVLNode* insert(AVLNode* node, int key)
82 {
83     // 이진 탐색 트리의 노드 추가 수행
84     // 노드가 null이면
85     if (node == NULL)
86         return(create_node(key)); // 반환한다
87
88     // 만약 키값이 노드의 키보다 작으면
89     if (key < node->key)
90         node->left = insert(node->left, key); // 왼쪽에 값 삽입
91     // 만약 키 값이 노드 보다 크다면
92     else if (key > node->key)
93         node->right = insert(node->right, key); // 오른쪽에 값 삽입
94     else // 동일한 키는 허용되지 않음
95         return node;
96 }

```

```

90
97 // 노드들의 균형인수 재계산
98 int balance = get_balance(node);
99
00 // LL 타입 처리
01 // 균형이 1 이상이고 키 값이 노드의 왼쪽의 키 값보다 작을 경우 회전
02 if (balance > 1 && key < node->left->key)
03     return rotate_right(node);
04
05 // RR 타입 처리
06 // 균형이 -1 이하이고 키 값이 노드의 오른쪽의 키 값보다 클 경우 회전
07 if (balance < -1 && key > node->right->key)
08     return rotate_left(node);
09
10 // LR 타입 처리
11 // 균형이 1 이상이고 키 값이 노드의 왼쪽의 키 값보다 클 경우
12 if (balance > 1 && key > node->left->key)
13 {
14     // 노드의 왼쪽을 오른쪽으로 회전 시키고
15     node->left = rotate_right(node->left);
16     // 노드를 오른쪽으로 돌린다
17     return rotate_right(node);
18 }
19
20 // RL 타입 처리
21 // 균형이 -1 이하이고 키 값이 노드의 오른쪽의 키 값보다 작을 경우
22 if (balance < -1 && key < node->right->key)
23 {
24     // 노드의 오른쪽을 오른쪽으로 회전 시키고
25     node->right = rotate_right(node->right);
26     // 왼쪽으로 돌린다
27     return rotate_left(node);
28 }
29 // 노드 반환
30 return node;
31 }
32

```



```

133 // 검색 함수
134 void Search(AVLNode *root, int search_num)
135 {
136     while(1)
137     {
138         if (root == NULL) // 루트가 null이면 탐색 실패
139         {
140             printf("값 탐색 실패\n");
141             break;
142         }
143         // 만약 루트값이 찾는값과 동일하면 출력 후 반복문 종료
144         if (root->key == search_num)
145         {
146             printf("[%d] ", root->key);
147             break;
148         }
149         // 아닐 경우는 그냥 출력
150         printf("[%d] ", root->key);
151         // 찾는 값이 루트 값보다 클 경우 오른쪽으로
152         if (search_num > root->key)
153             root = root->right;
154         // 아닐 경우는 왼쪽으로
155         else
156             root = root->left;
157     }
158 }
159
160
161 // 메인 함수
162 int main(void)
163 {
164     AVLNode *root = NULL; // 루트를 선언한다
165     FILE *fp; // 파일 포인터를 선언한다
166     // 필요한 함수를 선언한다
167     int tmp;
168     int save;
169     char check[10];
170
171     // 파일을 열고 실패 시 종료한다
172     fp = fopen("data02.txt", "r");
173
174     if (fp == NULL)
175     {
176         printf("파일 오픈 실패\n");
177         return 0;
178     }
179

```

```

179
180 //트리 구축
181 printf("< AVL 트리 >\n");
182 // 파일 끝까지 반복
183 while (!feof(fp))
184 {
185     // 만약 앞 글자가 a 라면
186     fscanf(fp, "%s", check);
187     if(strcmp(check, "a") == 0)
188     {
189         // 값을 읽은 후 노드에 삽입
190         fscanf(fp, "%d", &tmp);
191         printf("삽입 : %d\n", tmp);
192         root = insert(root, tmp);
193     }
194     else
195         fscanf(fp, "%d", &save);
196 }
197
198 printf("탐색 결과 : ");
199 // save 값을 탐색한다
200 Search(root, save);
201
202 // 파일을 닫고 종료한다
203 fclose(fp);
204 return 0;
205 }
206

```

## 2.3 소스 코드 분석

```
//=====
// 제작기간: 21년 11월 30일 ~ 12월 6일
// 제작자: 20204005 김필중
// 프로그램명: AVL트리 구현
//=====

// 필요한 헤더파일을 선언한다
#include<stdio.h>
#include<stdlib.h>
#include <string.h>

// 오류 방지 구문
#pragma warning(disable : 4996)

// max 매크로 선언
#define MAX(a, b) (a)
```

1. 필요한 헤더파일을 선언한다.
2. 오류 방지 구문을 선언한다
3. max 매크로를 선언한다

```
// AVL 트리 노드 정의
typedef struct AVLNode
{
    int key; // 키값
    struct AVLNode *left; // 왼쪽 링크 구성
    struct AVLNode *right; // 오른쪽 링크 구성
} AVLNode;
```

4. 트리 노드를 정의해서 키값, 왼쪽 링크, 오른쪽 링크를 구성하는 구조체를 선언한다

```
// 트리의 높이를 반환
int get_height(AVLNode *node)
{
    int height = 0; // 높이값 0으로 선언

    if (node != NULL) // 노드가 null이 아니라면
        height = 1 + max(get_height(node->left), get_height(node->right)); // 1+ 왼쪽 노드, 오른쪽 노드 높이 호환해서 큰 값 이용

    return height; // 높이 반환
}
```

5. 트리의 높이를 반환하는 함수를 만든다

6. 높이를 0으로 선언 후 노드가 null이 아니라면 높이를 1+ 왼쪽 or 오른쪽 높이 중 높은 값을 더해준 후 높이를 반환한다

```
// 노드의 균형인수를 반환
int get_balance(AVLNode* node)
{
    if (node == NULL) // 노드가 null이라면 0 반환
        return 0;

    return get_height(node->left) - get_height(node->right); // 왼쪽 높이 - 오른쪽 높이 반환
}
```

7. 노드 균형인수를 반환하는 함수를 만든다

8. 노드가 null이라면 종료한다

9. 아닐 경우는 왼쪽 높이 - 오른쪽 높이를 계산한 값을 반환한다

```
// 노드를 동적으로 생성하는 함수
AVLNode* create_node(int key)
{
    AVLNode* node = (AVLNode*)malloc(sizeof(AVLNode)); // 노드 동적할당
    node->key = key; // 키값을 대입
    node->left = NULL; // 왼쪽 링크 null로 연결
    node->right = NULL; // 오른쪽 링크 null로 연결
    return(node); // 노드 반환
}
```

10. 동적 생성 함수를 만든다

11. 노드를 동적할당 한 후 키값을 대입, 오른쪽 왼쪽 링크를 null로 연결을 한다.

12. 노드를 반환한다.

```

// 오른쪽으로 회전시키는 함수
AVLNode *rotate_right(AVLNode *parent)
{
    AVLNode* child = parent->left; // child를 부모 노드의 왼쪽에 연결
    parent->left = child->right; // 부모의 왼쪽을 자식의 오른쪽과 연결
    child->right = parent; // 자식의 오른쪽을 부모를 연결

    // 새로운 루트를 반환
    return child;
}

```

13. 오른쪽 회전 함수를 만든다

14. child를 부모 노드의 왼쪽과 연결하는 노드를 한 후

15. 부모 노드의 왼쪽을 child의 오른쪽 노드로 연결한다

16. 자식의 오른쪽을 부모를 연결한 후 새로운 child를 반환한다

```

// 왼쪽으로 회전시키는 함수
AVLNode *rotate_left(AVLNode *parent)
{
    AVLNode *child = parent->right; // child를 부모 노드의 오른쪽에 연결
    parent->right = child->left; // 부모의 오른쪽을 자식의 왼쪽과 연결
    child->left = parent; // 자식의 왼쪽을 부모를 연결

    // 새로운 루트 반환
    return child;
}

```

17. 왼쪽 회전 함수를 만든다

18. child를 부모 노드의 오른쪽과 연결하는 노드를 한 후

19. 부모 노드의 오른쪽을 child의 왼쪽 노드로 연결한다

20. 자식의 왼쪽을 부모를 연결한 후 새로운 child를 반환한다

```

// AVL 트리에 새로운 노드 추가 함수
// 새로운 루트를 반환한다.
AVLNode* insert(AVLNode* node, int key)
{
    // 이전 탐색 트리의 노드 추가 수행
    // 노드가 null이면
    if (node == NULL)
        return(create_node(key)); // 반환한다

    // 만약 키값이 노드의 키보다 작으면
    if (key < node->key)
        node->left = insert(node->left, key); // 왼쪽에 값 삽입
    // 만약 키 값이 노드 보다 크다면
    else if (key > node->key)
        node->right = insert(node->right, key); // 오른쪽에 값 삽입
    else // 동일한 키는 허용되지 않음
        return node;

    // 노드들의 균형인수 재계산
    int balance = get_balance(node);
}

```

21. 노드 추가 함수를 선언한다
22. 노드가 만약 null이라면 새로운 노드를 생성한 후 반환한다
23. 만약 키값이 노드의 키보다 작다면 왼쪽에 값을 삽입한다
24. 만약 키값이 노드의 키보다 크다면 오른쪽에 값을 삽입한다
25. 동일하면 노드를 반환한다.
26. 노드들의 균형인수 재계산한다.

```

// LL 타입 처리
// 균형이 1 이상이고 키 값이 노드의 왼쪽의 키 값보다 작을 경우 회전
if (balance > 1 && key < node->left->key)
    return rotate_right(node);

// RR 타입 처리
// 균형이 -1 이하이고 키 값이 노드의 오른쪽의 키 값보다 클 경우 회전
if (balance < -1 && key > node->right->key)
    return rotate_left(node);

```

27. LL타입 데이터를 처리한다. 만약 균형이 1이상이고 키 값이 노드의 왼쪽 키값보다 작을경우 오른쪽 회전

28. RR타입 데이터를 처리한다. 만약 균형이 -1이하이고 키 값이 노드의 오른쪽 키값보다 클 경우 왼쪽 회전

```
// LR 타입 처리
// 균형이 1 이상이고 키 값이 노드의 왼쪽의 키 값보다 클 경우
if (balance > 1 && key > node->left->key)
{
    // 노드의 왼쪽을 오른쪽으로 회전 시키고
    node->left = rotate_right(node->left);
    // 노드를 오른쪽으로 돌린다
    return rotate_right(node);
}

// RL 타입 처리
// 균형이 -1 이하이고 키 값이 노드의 오른쪽의 키 값보다 작을 경우
if (balance < -1 && key < node->right->key)
{
    // 노드의 오른쪽을 오른쪽으로 회전 시키고
    node->right = rotate_right(node->right);
    // 왼쪽으로 돌린다
    return rotate_left(node);
}

// 노드 반환
return node;
```

29. LR타입 데이터를 처리한다.

만약 균형이 1이상이고 키 값이 노드의 왼쪽 키값보다 클 경우

30. 노드의 왼쪽을 오른쪽으로 회전시키고 노드를 오른쪽으로 돌린다

31. RL타입 데이터를 처리한다.

만약 균형이 -1이하이고 키 값이 노드의 오른쪽 키값보다 작을 경우

32. 노드의 오른쪽을 오른쪽으로 회전시키고 노드를 왼쪽으로 돌린다

```

// 검색 함수
void Search(AVLNode *root, int search_num)
{
    while(1)
    {
        if (root == NULL) // 루트가 null이면 탐색 실패
        {
            printf("값 탐색 실패\n");
            break;
        }
        // 만약 루트값이 찾는값과 동일하면 출력 후 반복문 종료
        if (root->key == search_num)
        {
            printf("[%d] ", root->key);
            break;
        }
        // 아닐 경우는 그냥 출력
        printf("[%d] ", root->key);
        // 찾는 값이 루트 값보다 클 경우 오른쪽으로
        if (search_num > root->key)
            root = root->right;
        // 아닐 경우는 왼쪽으로
        else
            root = root->left;
    }
}

```

33. 검색 함수를 선언한다

34. 루트가 NULL이면 탐색을 실패하고 종료한다

35. 아닐 경우 루트값이 찾는값과 같으면 값을 표시하고 종료한다

36. 찾는 값이 아닐 경우는 크기를 비교해 조건에 맞게 오른쪽 혹은 왼쪽으로 이동한 후 다시 반복한다



```

// 메인 함수
int main(void)
{
    AVLNode *root = NULL; // 루트를 선언한다
    FILE *fp; // 파일 포인터를 선언한다
    // 필요한 함수를 선언한다
    int tmp;
    int save;
    char check[10];

    // 파일을 열고 실패 시 종료한다
    fp = fopen("data02.txt", "r");

    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return 0;
    }
}

```

37. 메인 함수에서 루트를 선언하고 파일 포인터를 선언한다

38. 필요한 함수들을 선언한다

39. 파일을 열고 실패 시 종료한다

```

//트리 구축
printf("< AVL 트리 >\n");
// 파일 끝까지 반복
while (!feof(fp))
{
    // 만약 앞 글자가 a 라면
    fscanf(fp, "%s", check);
    if(strcmp(check, "a") == 0)
    {
        // 값을 읽은 후 노드에 삽입
        fscanf(fp, "%d", &tmp);
        printf("삽입 : %d\n", tmp);
        root = insert(root, tmp);
    }
    else
        fscanf(fp, "%d", &save);
}

```

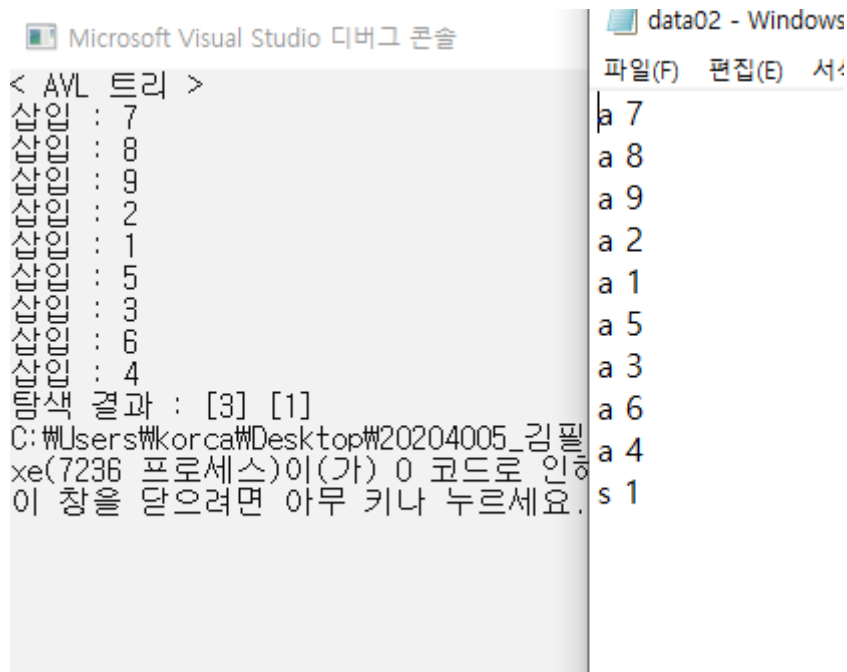
40. 트리를 구축하기 위해 파일 끝까지 반복을 하고 만약 앞 글자가 A라면 값을 읽은 후 노드에 삽입을 한다

41. 아닐 경우는 SAVE에 저장을 한다

```
printf("탐색 결과 : ");  
// save 값을 탐색한다  
Search(root, save);  
  
// 파일을 닫고 종료한다  
fclose(fp);  
return 0;
```

42. 탐색 결과를 호출 한 후 파일을 닫고 종료한다.

## 2.4 실행 결과



The screenshot shows two windows from Microsoft Visual Studio. The left window, titled 'Microsoft Visual Studio 디버그 콘솔', displays the output of an AVL tree search. It shows a series of '삽입' (insert) operations with values 7, 8, 9, 2, 1, 5, 3, 6, and 4. Below these, it shows '탐색 결과 : [3] [1]' and a system message about a process (xe(7236 프로세스)) being terminated. The right window, titled 'data02 - Windows', shows a list of values: a 7, a 8, a 9, a 2, a 1, a 5, a 3, a 6, a 4, and s 1.

```
< AVL 트리 >
삽입 : 7
삽입 : 8
삽입 : 9
삽입 : 2
삽입 : 1
삽입 : 5
삽입 : 3
삽입 : 6
삽입 : 4
탐색 결과 : [3] [1]
C:\Users\korca\Desktop\20204005_김필
xe(7236 프로세스)이(가) 0 코드로 인해
이 창을 닫으려면 아무 키나 누르세요.
```

data02 - Windows

파일(F) 편집(E) 서

a 7  
a 8  
a 9  
a 2  
a 1  
a 5  
a 3  
a 6  
a 4  
s 1

## 2.5 느낀점

이번 과제는 AVL 트리를 만들어서 탐색하는 과제였다. 이진 트리를 이용하는 것이기 때문에 탐색을 할 때 크기를 비교하면서 내려가는 것이 가능해 빠르다. 다만 AVL 트리를 처음에 봤을 때 돌리는 것이 가장 어려웠지만 계속 그림을 그리다 보니 이해가 조금씩 가서 다행이었다는 느낌을 받았다. 이것을 이용해 트리의 균형을 계속해서 유지하면서 한다면 더 빠른 트리 탐색이 가능하다고 생각하고 다양한 분야에서 사용이 가능하다고 생각했다.

### 3.1 문제 분석

#### ■ 선형조사법을 이용한 HashTable

- data.txt에 저장된 정수들을 선형 조사법을 이용한 HashTable에 저장하라. 저장된 정수들의 앞의 문자에 따라 저장하거나 HashTable에 있는지 검색하여 있으면 HashTable에 저장된 위치를 출력하고 검색이 되지 않는다면 아래와 같은 검색 실패 문구를 출력하라.

- Mod = 7, TableSize = 10

- i = insert, s = search

```
C:\WINDOWS\system32\cmd.exe
< HashTable Size = [10] >
```

```
< Data Insert Finish >
data = 91 저장 도중 HashTable : 0 에서 충돌 감지 - index = 1로 증가하였습니다.
data = 91 저장 도중 HashTable : 1 에서 충돌 감지 - index = 2로 증가하였습니다.
data = 106 저장 도중 HashTable : 1 에서 충돌 감지 - index = 2로 증가하였습니다.
data = 106 저장 도중 HashTable : 2 에서 충돌 감지 - index = 3로 증가하였습니다.
data = 106 저장 도중 HashTable : 3 에서 충돌 감지 - index = 4로 증가하였습니다.
data = 106 저장 도중 HashTable : 4 에서 충돌 감지 - index = 5로 증가하였습니다.
data = 106 저장 도중 HashTable : 5 에서 충돌 감지 - index = 6로 증가하였습니다.

< Find Data Location >
74 는 HashTable : 4 에서 검색되었습니다.
94 는 HashTable : 3 에서 검색되었습니다.
64 는 HashTable : 1 에서 검색되었습니다.
입력하신 값 90 은 HashTable에서 검색되지 않았습니다.
12 는 HashTable : 5 에서 검색되었습니다.
70 는 HashTable : 0 에서 검색되었습니다.
106 는 HashTable : 6 에서 검색되었습니다.
입력하신 값 51 은 HashTable에서 검색되지 않았습니다.
< Finish >
```

계속하려면 아무 키나 누르십시오 . . .

data - 메모장

파일(F) 편집(E) 서식(O) 보기(V)

i 74

i 94

i 64

i 70

i 12

i 91

i 106

s 74

s 94

s 64

s 90

s 12

s 70

s 106

s 51

이 문제는 선형 조사법을 이용해 해시 테이블을 만드는 문제이다. 가장 중요한 것은 테이블 사이즈는 10이고 나누는 수는 7로 나누어 나머지를 가지고 테이블에 저장하는 것이다. 그러기 위해서는 계산하는 함수 그리고 저장하는 함수를 가지고 만들어야 한다. 또한 저것이 만약 글자일 수도 있기 때문에 문자열을 숫자로 변환하는 atoi 를 이용할 예정이다.

먼저 element 구조체를 선언해서 char형 키값을 선언한다

Element 해싱 테이블을 선언하고 테이블 초기화 함수를 만든다

문자로 된 키를 숫자로 변환하는 함수를 만들어 atoi를 이용해 값을 숫자로 바꿔 반환하게 한다.

계산 함수를 만들어서 위 함수를 이용해 숫자로 바꾼 값을 mod로 나누어 남은 나머지를 반환한다

선형 조사법 식을 이용해서 테이블에 키를 삽입하는 함수를 만든다.

위에서 선언한 함수들을 이용해서 값을 얻은 후 만약 받은 값의 해싱테이블이 비어있지 않다면 인덱스 값을 하나씩 증가시키면서 빈 공간을 찾게 한다. 그마저도 없으면 해싱테이블이 꽉 찬것이기 때문에 종료한다 만약 해싱테이블이 비어있으면 그 공간에 값을 저장한다

선형 조사법 탐색 함수를 만들어서 위 함수와 마찬가지로 값을 보낸 후 계산된 값을 리턴 받는다. 그리고 비어 있지 않다면 만약 키값과 탐색된 키값이 같으면 출력 후 종료 아닐 경우 해싱 테이블을 하나 증가시켜 탐색한다 만약 계속 없으면 없는 것이기 때문에 없음을 출력하고 종료한다

메인 함수에서는 파일 포인터를 선언하고 구조체 포인터를 선언한다

데이터의 개수를 카운트 한 후 카운트 한만큼 동적할당을 하고 다시 파일을 끝까지 읽으면서 동적할당한 리스트에 저장한다. 그리고 리스트에 있는 값들을 해싱 테이블 삽입 함수를 이용해 해싱테이블에 넣고 그 후 들어오는 값들을 탐색을 호출해서 값들을 탐색하게 할 예정이다.

## 3.2 소스 코드

```
1 //=====
2 // 제작기간: 21년 11월 30일 ~ 12월 6일
3 // 제작자: 20204005 김필중
4 // 프로그래밍: 선형 조사법을 이용한 해시테이블
5 //=====
6
7 // 필요한 헤더파일 선언
8 #include <stdio.h>
9 #include <string.h>
10 #include <stdlib.h>
11
12 // 오류 방지 구문
13 #pragma warning(disable : 4996)
14
15 #define KEY_SIZE 10 // 탐색키의 최대길이
16 #define TABLE_SIZE 10 // 해싱 테이블의 크기
17 #define MOD 7 // 나누는 값 설정
18
19 // element 구조체 설정
20 typedef struct
21 {
22     char key[KEY_SIZE]; // 키값 설정
23 } element;
24
25 element hash_table[TABLE_SIZE]; // 해싱 테이블 선언
26
27 // 테이블 초기화
28 void init_table(element ht[])
29 {
30     int i;
31     // 테이블 사이즈 만큼 초기화를 한다
32     for (i = 0; i < TABLE_SIZE; i++)
33     {
34         ht[i].key[0] = NULL;
35     }
36 }
37
38 // 문자로 된 키를 숫자로 변환
39 int transform1(char *key)
40 {
41     int number = atoi(key); // 숫자 변환
42     return number; // 숫자 반환
43 }
44
45 // 제산 함수를 사용한 해싱 함수
46 int hash_function(char *key)
47 {
48     // 키를 숫자로 변환한 다음 MOD의 크기로 나누어 나머지를 반환
49     return transform1(key) % MOD;
50 }
51
```

```

52 // 필요한 매크로 선언
53 #define empty(item) (strlen(item.key)==0) // 비어 있는지 확인
54 #define equal(item1, item2) (!strcmp(item1.key, item2.key)) // 똑같은지 확인
55
56 // 선형 조사법을 이용하여 테이블에 키를 삽입하고, 테이블이 가득 찬 경우는 종료
57 void hash_lp_add(element item, element ht[])
58 {
59     // 필요한 변수 선언
60     int i, hash_value, number;
61     // hash_function을 호출해서 나온 값을 i에 저장
62     hash_value = i = hash_function(item.key);
63     // number에 transform 함수에서 나온 값을 저장
64     number = transform1(item.key);
65     // 만약 i번째 해시테이블이 비어있지 않다면 반복
66     while (!empty(ht[i]))
67     {
68         // 만약 item의 키값과 해시테이블의 키값이 같다면 종료
69         if (equal(item, ht[i]))
70         {
71             fprintf(stderr, "탐색키가 중복되었습니다\n");
72             exit(1);
73         }
74         // 아닐 경우
75         i = (i + 1) % MOD; // i에 1을 더해준 후 mod를 나눠 나온 나머지를 저장
76         printf("data = %d 저장 도중 HashTable : %d에서 충돌 감지 - index = %d로 증가하였습니다.\n", number, i - 1, i);
77         // 만약 i값과 해시 밸류의 값이 같으면 다한것이기 때문에 종료
78         if (i == hash_value)
79         {
80             fprintf(stderr, "테이블이 가득찼습니다\n");
81             exit(1);
82         }
83     }
84     // i번째 해시 테이블에 값 저장
85     ht[i] = item;
86 }
87
88 // 선형조사법을 이용하여 테이블에 저장된 키를 탐색
89 void hash_lp_search(element item, element ht[])
90 {
91     // 필요한 변수 선언
92     int i, hash_value;
93     // hash_function을 호출해서 나온 값을 i에 저장
94     hash_value = i = hash_function(item.key);
95     // 비어있지 않다면
96     while (!empty(ht[i]))
97     {
98         // 값이 같을 경우
99         if (equal(item, ht[i]))
100         {
101             fprintf(stderr, "%s는 HashTable : %d 에서 검색되었습니다\n", item.key, i); // 출력 후 종료
102             return;
103         }
104         // 같지 않다면 i에 1을 더해준 후 mod를 나눠 나온 나머지를 저장
105         i = (i + 1) % MOD;
106         // 만약 i값과 해시 밸류의 값이 같으면 검색 실패
107         if (i == hash_value)
108         {
109             fprintf(stderr, "입력하신 값 %s 은 HashTable에서 검색되지 않습니다\n", item.key);
110             return;
111         }
112     }
113     // 비어있으면 검색 실패
114     fprintf(stderr, "입력하신 값 %s 은 HashTable에서 검색되지 않습니다\n", item.key);
115 }
116
117

```

```

118 // 메인 함수
119 int main(void)
120 {
121     FILE *fp; // 파일 포인터
122     element e; // 임시 구조체
123     element *list; // 리스트 포인터 구조체
124     // 필요한 함수 선언
125     int cnt = 0;
126     char check[10];
127     int j = 0;
128
129     // 파일 오픈하고 실패시 종료
130     fp = fopen("data03.txt", "r");
131
132     if (fp == NULL)
133     {
134         printf("파일 오픈 실패\n");
135         return 0;
136     }
137
138     printf("< HashTable Size = [%d] >\n\n\n", TABLE_SIZE);
139
140     // 파일을 끝까지 반복한다
141     while (!feof(fp))
142     {
143         // 앞의 글자를 읽고 i일 경우 카운트 1개를 증가시킨다
144         fscanf(fp, "%s", check);
145         if (strcmp(check, "i") == 0)
146             cnt++;
147     }
148     // 파일 포인터를 맨앞으로 돌린다
149     rewind(fp);
150
151     // 카운트 한 개수만큼 동적할당을 한다
152     list = (element*)malloc(sizeof(element)*cnt);
153
154     // 파일 끝까지 반복한다
155     while (!feof(fp))
156     {
157         // 맨 앞글자를 읽고 i 라면
158         fscanf(fp, "%s", check);
159         if (strcmp(check, "i") == 0)
160         {
161             fscanf(fp, "%s", check); // 값을 읽고 리스트 배열에 저장한다
162             strcpy(list[j].key, check);
163             j++;
164         }
165     }
166     // 파일 포인터를 맨앞으로 돌린다
167     rewind(fp);
168

```



```

170 printf("< Data Insert Finish >\n");
171 // 개수 만큼 반복한다
172 for (int i = 0; i < cnt; i++)
173 {
174     // list배열에 값을 hash_lp_add에 호출한다
175     strcpy(e.key, list[i].key);
176     hash_lp_add(e, hash_table);
177 }
178
179 // 값 탐색 함수
180 printf("\n\n< Find Data Location >\n");
181 // 파일 끝까지 반복한다
182 while (!feof(fp))
183 {
184     // 파일 앞글자를 읽고 s면
185     fscanf(fp, "%s", check);
186     if (strcmp(check, "s") == 0)
187     {
188         // 값을 읽고 탐색 함수를 호출한다
189         fscanf(fp, "%s", check);
190         strcpy(e.key, check);
191         hash_lp_search(e, hash_table);
192     }
193 }
194
195 // 파일을 닫고 동적할당을 해제 시키고 종료한다
196 fclose(fp);
197 free(list);
198 printf("\n\n< Finish >\n");
199 return 0;
200 }

```

### 3.3 소스 코드 분석

```
//=====
// 제작기간: 21년 11월 30일 ~ 12월 6일
// 제작자: 20204005 김필중
// 프로그램명: 선형 조사법을 이용한 해시테이블
//=====

// 필요한 헤더파일 선언
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// 오류 방지 구문
#pragma warning(disable : 4996)

#define KEY_SIZE 10 // 탐색키의 최대길이
#define TABLE_SIZE 10 // 해싱 테이블의 크기
#define MOD 7 // 나누는 값 설정
```

1. 필요한 헤더파일을 선언한다
2. 오류 방지 구문을 선언한다
3. 필요한 값들을 정의한다

```
// element 구조체 설정
typedef struct
{
    char key[KEY_SIZE]; // 키값 설정
} element;

element hash_table[TABLE_SIZE]; // 해싱 테이블 선언
```

4. element 구조체를 선언한다
5. 구조체 안에는 char형 키값을 선언한다
6. 해싱 테이블을 선언한다

```

// 테이블 초기화
void init_table(element ht[])
{
    int i;
    // 테이블 사이즈 만큼 초기화를 한다
    for (i = 0; i < TABLE_SIZE; i++)
    {
        ht[i].key[0] = NULL;
    }
}

```

7. 테이블 초기화 함수를 만든다
8. 테이블 사이즈 만큼 초기화를 한다.

```

// 문자로 된 키를 숫자로 변환
int transform1(char *key)
{
    int number = atoi(key); // 숫자 변환
    return number; // 숫자 반환
}

```

9. 문자로 된 키를 숫자로 변환하는 함수를 만든다
10. number 변수에 atoi를 이용해 숫자로 변환한 후 숫자를 반환한다

```

// 제산 함수를 사용한 해싱 함수
int hash_function(char *key)
{
    // 키를 숫자로 변환한 다음 MOD의 크기로 나누어 나머지를 반환
    return transform1(key) % MOD;
}

```

11. 해싱 함수를 선언한다
12. 변환 함수를 호출해 받은 값을 위에서 선언한 MOD로 나누어 남은 나머지를 반환한다

```
// 필요한 매크로 선언
#define empty(item) (strlen(item.key)==0) // 비어 있는지 확인
#define equal(item1, item2) (!strcmp(item1.key, item2.key)) // 똑같은지 확인
```

13. EQUAL 매크로와, EMPTY 매크로를 선언한다

```
// 선형 조사법을 이용하여 테이블에 키를 삽입하고, 테이블이 가득 찬 경우는
void hash_ip_add(element item, element ht[])
{
    // 필요한 변수 선언
    int i, hash_value, number;
    // hash_function을 호출해서 나온 값을 i에 저장
    hash_value = i = hash_function(item.key);
    // number에 transform1 함수에서 나온 값을 저장
    number = transform1(item.key);
    .....
}
```

14. 선형 조사법을 이용한 키 삽입 함수를 선언한다

15. 필요한 변수들을 선언한다

16. hash\_value와 i 값을 해싱 함수를 호출해서 나온 값을 저장한다

17. number에 변환한 값을 저장한다

```
.....
// 만약 i번째 해시테이블이 비어있지 않다면 반복
while (!empty(ht[i]))
{
    // 만약 item의 키값과 해시테이블의 키값이 같다면 종료
    if (equal(item, ht[i]))
    {
        fprintf(stderr, "탐색키가 중복되었습니다\n");
        exit(1);
    }
    // 아닐 경우
    i = (i + 1) % MOD; // i에 1을 더해준 후 mod를 나눠 나온 나머지를 저장
    printf("data = %d 저장 도중 HashTable : %d에서 충돌 감지 - index = %d로 증가하였습니다.\n", number, i - 1, i);
    // 만약 i값과 해시 밸류의 값이 같으면 다찬것이기 때문에 종료
    if (i == hash_value)
    {
        fprintf(stderr, "테이블이 가득 찼습니다\n");
        exit(1);
    }
}
// i번째 해시 테이블에 값 저장
ht[i] = item;
```

18. 만약 i번째 해시테이블이 비어있지 않다면 반복문을 실행한다.

19. 만약 item의 키값과 해시 테이블의 키값이 같다면 종료한다.

20. 아닐 경우는 i값을 하나 늘린 후 다시 mod로 나눈 나머지 값으로 변경한다

21. 그리고 충돌이 되었다는 프린트를 한 후 만약 i와 hash\_value가 같다면 해시 테이블이 가득찼다는 뜻이므로 종료하난

22. 만약 모두 아니면 i번째 해시테이블에 값을 삽입한다

```
// 선형조사법을 이용하여 테이블에 저장된 키를 탐색
void hash_lp_search(element item, element ht[])
{
    // 필요한 변수 선언
    int i, hash_value;
    // hash_function을 호출해서 나온 값을 i에 저장
    hash_value = i = hash_function(item.key);
    // 비어있지 않다면
    while (!empty(ht[i]))
    {
        // 값이 같을 경우
        if (equal(item, ht[i]))
        {
            fprintf(stderr, "%s는 HashTable : %d 에서 검색되었습니다\n", item.key, i); // 출력 후 종료
            return;
        }
        // 같지 않다면 i에 1을 더해준 후 mod를 나눠 나온 나머지를 저장
        i = (i + 1) % MOD;
        // 만약 i값과 해시 밸류의 값이 같으면 검색 실패
        if (i == hash_value)
        {
            fprintf(stderr, "입력하신 값 %s 은 HashTable에서 검색되지 않습니다\n", item.key);
            return;
        }
    }
    // 비어있으면 검색 실패
    fprintf(stderr, "입력하신 값 %s 은 HashTable에서 검색되지 않습니다\n", item.key);
}
```

23. 선형조사법을 이용한 탐색함수를 선언한다

24. 필요한 변수를 선언하고 hash\_value와 i 값을 해싱 함수를 호출해서 나온 값을 저장한다

25. i번째 해시테이블이 비어있지 않다면 계속 반복한다.

26. 만약 해시테이블의 값과 item의 값이 같을 경우 탐색 성공을 호출하고 호출한 값, 인덱스 번호를 알려준 후 종료한다.

27. 아닐 경우 i값을 하나 증가시키고 mod로 나눈 나머지 값을 i 값으로 정한다

28. 만약 i와 hash\_value 값이 같으면 탐색 실패를 말한다

29. 만약 해싱테이블이 비어있으면 없다는 뜻이므로 종료한다.

```
// 메인 함수
int main(void)
{
    FILE *fp; // 파일 포인터
    element e; // 임시 구조체
    element *list; // 리스트 포인터 구조체
    // 필요한 함수 선언
    int cnt = 0;
    char check[10];
    int j = 0;

    // 파일 오픈하고 실패시 종료
    fp = fopen("data03.txt", "r");

    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return 0;
    }

    printf("< HashTable Size = [%d] >\n\n", TABLE_SIZE);
```

30. 메인함수에서는 파일 포인터 임시 구조체, 리스트 포인터 구조체를 선언한다

31. 필요한 변수들을 선언한다

32. 파일을 오픈하고 실패하면 종료한다

```
// 파일을 끝까지 반복한다
while (!feof(fp))
{
    // 앞의 글자를 읽고 i일 경우 카운트 1개를 증가
    fscanf(fp, "%s", check);
    if (strcmp(check, "i") == 0)
        cnt++;
}
// 파일 포인터를 맨앞으로 돌린다
rewind(fp);

// 카운트 한 개수만큼 동적할당을 한다
list = (element*)malloc(sizeof(element)*cnt);
```

33. 파일을 끝까지 반복을 한다. 만약 앞의 글자를 읽고 i일 경우 카운트를 증가시킨다.

34. 파일 포인터를 앞으로 돌린다

35. 카운트한 개수만큼 동적할당을 한다.

```
// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 맨 앞글자를 읽고 i 라면
    fscanf(fp, "%s", check);
    if (strcmp(check, "i") == 0)
    {
        fscanf(fp, "%s", check); // 값을 읽고 리스트 배열에 저장한다
        strcpy(list[j].key, check);
        j++;
    }
}
// 파일 포인터를 맨앞으로 돌린다
rewind(fp);
```

36. 파일을 끝까지 반복하면서 맨 앞글자를 읽고 i라면 값을 읽고 리스트 배열에 저장을 한다

37. 다시 파일 포인터를 맨앞으로 돌린다

```
printf("< Data Insert Finish >\n");
// 개수 만큼 반복한다
for (int i = 0; i < cnt; i++)
{
    // list배열에 값을 hash_lp_add에 호출한다
    strcpy(e.key, list[i].key);
    hash_lp_add(e, hash_table);
}

// 값 탐색 함수
printf("\n\n< Find Data Location >\n");
// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 파일 앞글자를 읽고 s면
    fscanf(fp, "%s", check);
    if (strcmp(check, "s") == 0)
    {
        // 값을 읽고 탐색 함수를 호출한다
        fscanf(fp, "%s", check);
        strcpy(e.key, check);
        hash_lp_search(e, hash_table);
    }
}
```

38. 개수 만큼 반복하면서 값을 해싱테이블에 삽입하는 함수를 호출한다

39. 넣은 후 파일을 맨 끝까지 반복하면서 맨 앞글자가 s일 경우 그 뒤에 값을 읽어서 탐색 함수를 호출한다.

```
// 파일을 닫고 동적할당을 해제시키고 종료한다
fclose(fp);
free(list);
printf("#####< Finish >#####");
return 0;
```

40. 파일을 닫고 동적할당을 해제시키고 종료한다



## 3.4 실행 결과

```
Microsoft Visual Studio 디버그 콘솔
< HashTable Size = [10] >

< Data Insert Finish >
data = 91 저장 도중 HashTable : 0에서 충돌 감지 - index = 1로 증가하였습니다.
data = 91 저장 도중 HashTable : 1에서 충돌 감지 - index = 2로 증가하였습니다.
data = 106 저장 도중 HashTable : 1에서 충돌 감지 - index = 2로 증가하였습니다.
data = 106 저장 도중 HashTable : 2에서 충돌 감지 - index = 3로 증가하였습니다.
data = 106 저장 도중 HashTable : 3에서 충돌 감지 - index = 4로 증가하였습니다.
data = 106 저장 도중 HashTable : 4에서 충돌 감지 - index = 5로 증가하였습니다.
data = 106 저장 도중 HashTable : 5에서 충돌 감지 - index = 6로 증가하였습니다.

< Find Data Location >
74는 HashTable : 4 에서 검색되었습니다
34는 HashTable : 3 에서 검색되었습니다
34는 HashTable : 1 에서 검색되었습니다
입력하신 값 90 은 HashTable에서 검색되지 않습니다
12는 HashTable : 5 에서 검색되었습니다
70는 HashTable : 0 에서 검색되었습니다
106는 HashTable : 6 에서 검색되었습니다
입력하신 값 51 은 HashTable에서 검색되지 않습니다

< Finish >
C:\Users\korca\Desktop\20204005_김필중_자료구조실습_14주차_소스코드\Debug\20204005_김필중_자료구조실습_14주차.exe(14856 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

```
data03 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V)
i 74
i 94
i 64
i 70
i 12
i 91
i 106
s 74
s 94
s 64
s 90
s 12
s 70
s 106
s 51
```

## 3.5 느낀점

해시 테이블을 처음 보고 나서 이것을 왜 사용할까 하고 느꼈다. 하지만 해시 테이블을 잘 이용한다면 데이터를 충돌없이 삽입하고 탐색하기가 쉽다는 것을 느꼈다. 또한 나머지 값을 이용해 데이터를 저장하는 부분은 매우 흥미로워서 나중에는 다양한 나머지가 아니면 다른 방법이 있는지에 대해 공부를 더 해보고 싶다는 생각을 했다. 해시 테이블 저장 방법은 더 많다고 과제를 진행하면서 느꼈다. 추후 프로그램을 제작할 때 해시테이블을 이용하면 좋은 기능을 만들 수 있겠다는 생각을 했다.