



순천향대학교
SOON CHUN HYANG
UNIVERSITY

자료구조2 실습 5주차

자료구조2 실습

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 10월 5일

| 목 차 |

1. 이진 트리 영어 사전 재귀함수

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

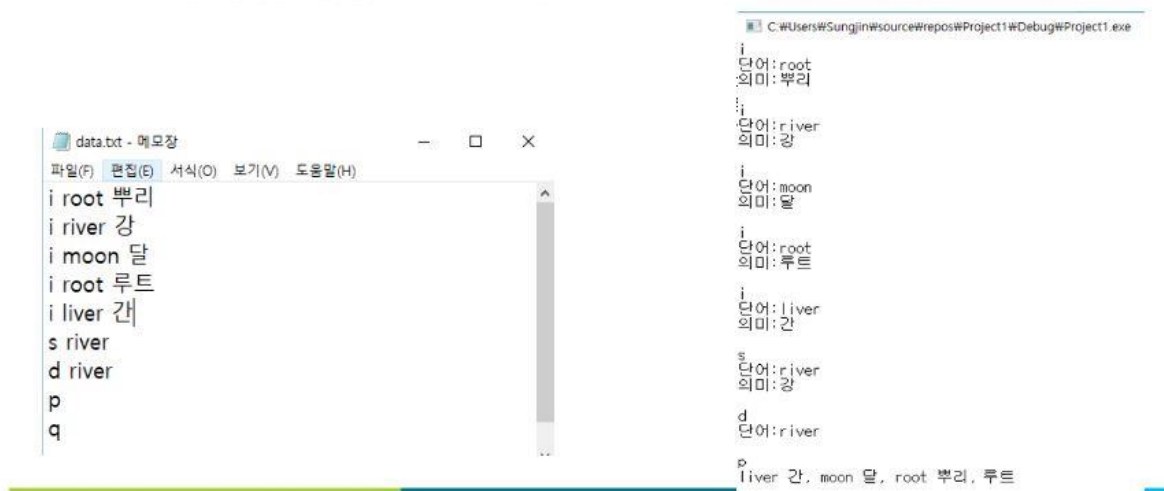
2. 이진 트리 영어 사전 반복

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

1.1 문제 분석

이진 트리

- P. 312의 프로그램 8.14를 활용하여 이진 탐색 트리를 이용한 영어 사전 프로그램을 작성하시오.
- data.txt 파일에 양식, 단어, 뜻이 저장되어 있음
- i는 데이터를 삽입, d는 데이터 삭제, s는 데이터의 탐색을 의미
- p는 전체 데이터의 출력, q는 프로그램 종료를 의미
- 중복되는 데이터가 삽입되는 경우 기존의 단어에 뜻이 추가되어야 함. 뜻은 연결 리스트의 형태로 크기에 따라 동적할당으로 구현



이번 문제는 저번 과제와 비슷한 문제로 앞에 있는 알파벳을 읽어 각 알파벳에 맞게 프로그램이 실행되면 되는 과제이다. 여기다가 트리를 이용해 영어 사전을 만들어야 한다.

우선 단어 구조체를 만들어야 한다. 이번에는 단어와 뜻이 들어가야 하고 2개는 모두 동적할당으로 단어의 크기에 맞게 공간이 할당되어야 하기 때문에 문자열 포인터로 선언을 해야 한다.

다음으로는 트리 구조체를 만들어야 한다. 트리 구조체에는 단어 구조체 data와 트리의 좌 우 연결 링크가 필요하다.

다음으로는 단어의 크기를 비교하는 함수가 필요하다. 영어 사전에서는 알파벳이 빠른 것이 앞으로 오기 때문에 단어 크기를 비교해서 strcmp를 이용해 함수를 만든다

단어 사전 프린트 함수가 필요하다. 중위 순회로 제작을 하고 마지막 단어이면 ,
가 빠진 상태로 출력을 하는 방식을 고려해야 한다.

다음은 탐색 함수이다. 우선 탐색 함수는 반복이 아닌 순환으로 제작한다. 트리와
단어를 받은 후 둘을 비교해 같을 경우는 출력 아닐 경우는 둘의 크기를 비교해
트리 노드의 좌 혹은 우로 이동하면서 단어를 찾을 수 있는 함수를 만든다. 없을
경우는 없다 라고 표시까지 나와야 한다.

다음으로는 새로운 노드 생성 함수가 필요하다. 새로운 노드를 동적할당 해준 후
받은 값을 넣어 준 후 좌 우를 null로 설정해 준 후 반환하는 함수가 필요하다.

다음으로는 노드 삽입 함수가 필요하다. 공백일 경우는 새로운 노드 생성 함수를
호출하고 아닐경우는 노드의 단어와 매개변수로 받은 단어를 비교해 좌 혹은 우
로 이동해 순환하는 함수를 제작한다. 만약 같을 경우는 뜻을 추가 해 준 후 종
료하게 해야 한다.

제거 함수를 위한 단말 노드 찾는 함수를 만들어야 한다. Node를 받은 후 왼쪽
단말 노드를 찾으러 계속 내려가는 함수이다.

다음으로는 제거 함수를 만들어야 한다. 이번에는 반복이 아닌 순환으로 제작한
다. 크기를 비교해서 왼쪽 오른쪽으로 이동한 후 재귀를 시켜준다. 이렇게 하다가
같은 단어를 찾을 경우 우선 왼쪽 혹은 오른쪽이 널일 경우는 temp를 만들어 가
리키게 한 후 free를 통해 제거한다. 만약 2가지가 아닐 경우는 맨 왼쪽으로 내려
가는 함수를 호출 한 후 중위 순회시 후계 노드를 삭제한다.

메인 함수에서는 우선 단어의 크기만큼 동적할당을 해줘야 하고 while 문을 이용
해 파일의 끝까지 읽으면서 해당하는 알파벳에 맞게 동작을 시켜주는 반복문을
제작해야 한다.

1.2 소스 코드

```
1 //-----
2 // 제작일 : 21년 09월 29일 ~ 10월 4일
3 // 제작자 : 20204005 김필중
4 // 프로그램명 : 이진트리 영어 사전 프로그램
5 //-----
6
7 // 필요한 헤더파일을 선언한다.
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include <memory.h>
12
13 // 오류 방지 구문을 선언한다
14 #pragma warning (disable : 4996)
15
16 // 전역 변수를 선언한다
17 // 사전 전체 출력시 마지막 단어일 경우 , 제거를 위함
18 int number1 = 0;
19 int number2 = 1;
20
21 // Word 구조체를 선언한다
22 typedef struct word
23 {
24     char *word; // 단어의 크기만큼 동적할당 하기 위해 포인터 선언
25     char *mean; // 뜻의 크기만큼 동적할당 하기 위해 포인터 선언
26 }Word;
27
28 // Tree 구조체 선언한다
29 typedef struct TreeNode
30 {
31     Word data; // 단어 구조체 data 변수 선언
32     struct TreeNode *left, *right; // 자기참조를 통해 트리의 좌우 링크 설정
33 }TreeNode;
34
```

```

35 // 단어 2개의 크기를 비교하는 함수
36 int compare(Word e1, Word e2)
37 {
38     return strcmp(e1.word, e2.word); // 받아온 단어 2개의 크기를 비교해 리턴
39 }
40
41 // 단어 사전 프린트 함수
42 void display(TreeNode *p)
43 {
44     if (p != NULL) // p가 NULL 이 아닐경우
45     {
46         display(p->left); // p->left의 재귀함수를 호출한다
47         if (number1 == number2) // 만약 number1(사전의 총 단어의 개수) == number 2(현재 출력한 단어의 개수) 가 일치하면
48         {
49             printf("%s %s", p->data.word, p->data.mean); // , 없이 출력한다
50             return; // 함수를 종료한다
51         }
52         else // 아닐경우는
53         {
54             printf("%s %s, ", p->data.word, p->data.mean); // , 가 있게 출력을 한다
55         }
56         number2++; // 현재 출력 단어 +1을 해준다
57         display(p->right); // p->right의 재귀함수를 호출한다
58     }
59 }
60
61 // 탐색 함수
62 TreeNode * search(TreeNode *root, Word data)
63 {
64     TreeNode * p = root; // p를 root에 가리키게 한다
65     while (p != NULL) // p가 null이 아닐때까지 반복한다
66     {
67         if (compare(data, p->data) == 0) // 만약 compare 함수를 통해 값이 0이 나올경우 같다는 것이다
68             return p; // p를 반환한다
69         else if (compare(data, p->data) < 0) // 만약 < 0 일 경우는
70             p = p->left; // 왼쪽으로 이동한다
71         else if (compare(data, p->data) > 0) // 만약 > 0 일 경우는
72             p = p->right; // 오른쪽으로 이동한다
73     }
74     return p; // 탐색에 실패했을 경우 NULL 반환
75 }
76
77 // 새로운 노드 생성 함수
78 TreeNode * new_node(Word item)
79 {
80     TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드를 크기만큼 동적할당 한다.
81     temp->data = item; // 받아온 값을 data에 넣어준다
82     temp->left = temp->right = NULL; // 노드의 좌 우 값을 null로 설정한다
83     number1++; // 총 단어의 개수를 증가시킨다
84     return temp; // 반환한다
85 }
86

```

```

87 // 트리 노드 삽입함수
88 TreeNode * insert_node(TreeNode * node, Word data)
89 {
90     // 트리가 공백이면 새로운 노드를 반환한다.
91     if (node == NULL)
92         return new_node(data);
93
94     // 그렇지 않으면 순환적으로 트리를 내려간다.
95     if (compare(data, node->data) < 0) // 만약 현재 단어가 매개변수의 단어보다 작다면
96     {
97         node->left = insert_node(node->left, data); // 왼쪽으로 재귀함수를 호출한다
98     }
99
100     else if (compare(data, node->data) > 0) // 만약 현재 단어가 매개변수의 단어보다 크다면
101     {
102         node->right = insert_node(node->right, data); // 오른쪽으로 재귀함수를 호출한다
103     }
104
105
106     else if (compare(data, node->data) == 0) // 만약 단어가 같다면
107     {
108         // 뜻을 이어 붙여 준다
109         strcat(node->data.mean, ", ");
110         strcat(node->data.mean, data.mean);
111         return node; // 루트 포인터를 반환한다.
112     }
113
114     // 루트 포인터를 반환한다.
115     return node;
116 }

```

```

117 //
118
119 TreeNode * min_value_node(TreeNode *node)
120 {
121     TreeNode * current = node; // current 노드가 node 를 가리키게 한다
122     // 맨 왼쪽 단말 노드를 찾으려 내려감
123     while (current->left != NULL)
124         current = current->left;
125     return current;
126 }

```



```

128 // 제거 함수 재귀함수 버전
129 TreeNode* delete_node(TreeNode* root, Word key)
130 {
131     if (root == NULL)
132         return root;
133
134     // 만약 키가 루트보다 작으면 왼쪽 서브 트리에 있는 것임
135     if (compare(key, root->data) < 0)
136         root->left = delete_node(root->left, key);
137     // 만약 키가 루트보다 크면 오른쪽 서브 트리에 있는 것임
138     else if (compare(key, root->data) > 0)
139         root->right = delete_node(root->right, key);
140     // 키가 루트와 같으면 이 노드를 삭제하면 됨
141     else
142     {
143         // 첫 번째나 두 번째 경우
144         if (root->left == NULL)
145         {
146             TreeNode* temp = root->right; // temp를 root->right 를 가리키게 한다
147             free(root); // root 를 제거한다
148             return temp; // temp를 리턴한다
149         }
150         else if (root->right == NULL)
151         {
152             TreeNode* temp = root->left; // temp를 root->left 를 가리키게 한다
153             free(root); // root 를 제거한다
154             return temp; // temp를 리턴한다
155         }
156         // 세 번째 경우
157         TreeNode* temp = min_value_node(root->right);
158
159         // 중위 순회시 후계 노드를 복사한다.
160         root->data = temp->data;
161         // 중위 순회시 후계 노드를 삭제한다.
162         root->right = delete_node(root->right, temp->data);
163     }
164     return root;
165 }
166

```



```

168 // 메인 함수
169 int main(void)
170 {
171     FILE *fp; // 파일 포인터 fp를 선언한다
172     // strcpy를 위한 문자열 변수를 선언한다
173     char temp_check[100];
174     char temp_word[100];
175     char temp_mean[100];
176     Word *test; // test 구조체 포인터를 선언한다
177     TreeNode * root = NULL; // root 노드를 선언한다
178     TreeNode * tmp; // 임시 저장 노드를 선언한다
179     // 정수형 변수를 선언한다
180     int size = 0;
181     int i = 0;
182     int cnt = 0;
183
184     fp = fopen("data.txt", "r"); // 파일을 오픈한다
185
186     // 파일 오픈 실패시 오류 메시지를 출력하고 종료한다
187     if (fp == NULL)
188     {
189         printf("파일 오픈 실패\n");
190         return 0;
191     }
192
193     // 파일 끝까지 반복한다
194     while (!feof(fp))
195     {
196         fgets(temp_check, 100, fp); // 한줄을 읽는다
197         cnt++; // cnt 크기 증가
198     }
199     rewind(fp); // 파일포인터를 처음으로 돌린다
200     test = (Word*)malloc(sizeof(Word) * cnt); // cnt 개수만큼 동적할당을 해준다
201
202

```

```

203 // 파일 끝까지 반복을 한다
204 while (!feof(fp))
205 {
206     fscanf(fp, "%s ", temp_check); // 처음 값을 입력받는다
207     // 만약 처음 값이 i 일 경우는
208     if (strcmp(temp_check, "i") == 0)
209     {
210         fscanf(fp, "%s %s\n", temp_word, temp_mean); // 단어의 이름과 뜻을 저장한다
211         printf("%s\n", temp_check); // i 를 출력한다
212         size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
213         test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
214         size = strlen(temp_mean); // 단어의 뜻의 크기를 계산한다
215         test[i].mean = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 뜻의 크기 + 1 만큼 동적할당한다
216         // 단어 출력
217         printf("단어 : %s\n", temp_word);
218         printf("의미 : %s\n", temp_mean);
219         // strcpy로 구조체에 넣어준다
220         strcpy(test[i].word, temp_word);
221         strcpy(test[i].mean, temp_mean);
222         // insert_node 를 이용해 root에 데이터를 삽입한다
223         root = insert_node(root, test[i]);
224         printf("\n");
225     }
226
227     // 만약 처음 값이 p 일 경우는
228     if (strcmp(temp_check, "p") == 0)
229     {
230         printf("%s\n", temp_check); // p를 출력한다
231         display(root); // root를 display로 사전에 모두 출력한다
232         printf("\n");
233     }

```

```

234 // 만약 처음 값이 d 일 경우는
235 if (strcmp(temp_check, "d") == 0)
236 {
237     fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
238     printf("%s\n", temp_check); // d를 출력한다
239     printf("단어 : %s\n", temp_word); // 단어를 출력한다
240     size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
241     test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
242     strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
243     root = delete_node(root, test[i]); // delete_node 를 이용해 root에 데이터를 제거한다
244     printf("\n");
245     number1--; // 총 단어 개수 -1을 한다
246 }
247 // 만약 처음 값이 s 일 경우는
248 if (strcmp(temp_check, "s") == 0)
249 {
250     fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
251     printf("%s\n", temp_check); // s를 출력한다
252     printf("단어 : %s\n", temp_word); // 단어를 출력한다
253     size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
254     test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
255     strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
256     tmp = search(root, test[i]); // search 를 이용해 데이터를 찾는다
257     if (tmp != NULL) // 만약 tmp 가 null 이 아닐경우에는
258     {
259         printf("의미 : %s\n", tmp->data.mean); // 뜻을 출력한다
260     }
261     else
262         printf("단어를 찾을 수 없습니다\n");
263
264     printf("\n");
265 }
266 // 만약 처음 값이 q 일 경우는
267 if (strcmp(temp_check, "q") == 0)
268 {
269     printf("\n%s\n", temp_check); // q를 출력하고
270     break; // 종료한다
271 }
272
273 i++; // i값을 증가시킨다

```

```

279 fclose(fp); // 파일을 닫는다
280 free(test); // test를 동적할당 해제를 해준다
281 free(root); // root를 동적할당 해제를 해준다
282 return 0;
283 }

```

1.3 소스 코드 분석

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

1. 필요한 헤더파일을 선언한다.

```
// 오류 방지 구문을 선언한다  
#pragma warning (disable : 4996)  
  
// 전역 변수를 선언한다  
// 사전 전체 출력시 마지막 단어일 경우 , 제거를 위함  
int number1 = 0;  
int number2 = 1;
```

2. 오류 방지 구문을 선언한다

3. 사전 프린트 함수에서 마지막을 찾기 위해 전역변수를 선언한다

```
// Word 구조체를 선언한다  
typedef struct word  
{  
    char *word; // 단어의 크기만큼 동적할당 하기 위해 포인터 선언  
    char *mean; // 뜻의 크기만큼 동적할당 하기 위해 포인터 선언  
}Word;
```

4. Word 구조체를 선언한다

5. 구조체 안에 있는 문자열 변수들은 모두 포인터로 선언해서 동적할당을 하기 위한다.

```
// Tree 구조체 선언한다
typedef struct TreeNode
{
    Word data; // 단어 구조체 data 변수 선언
    struct TreeNode *left, *right; // 자기참조를 통해 트리의 좌우 링크 설정
}TreeNode;
```

6. 트리 구조체를 선언한다

7. 단어 구조체 data를 선언하고

8. 자기 참조를 통해 트리의 좌 우 링크를 설정한다.

```
// 단어 2개의 크기를 비교하는 함수
int compare(Word e1, Word e2)
{
    return strcmp(e1.word, e2.word); // 받아온 단어 2개의 크기를 비교해 리턴
}
```

9. 단어 2개를 비교하는 함수를 만든다

10. 단어 2개를 받아와 strcmp를 통해 나오는 값을 반환한다.

```
// 단어 사전 프린트 함수
void display(TreeNode *p)
{
    if (p != NULL) // p가 NULL 이 아닐경우
    {
        display(p->left); // p->left의 재귀함수를 호출한다
        if (number1 == number2) // 만약 number1(사전의 총 단어의 개수) == number 2(현재 출력한 단어의 개수) 가 일치하면
        {
            printf("%s %s", p->data.word, p->data.mean); // , 없이 출력한다
            return; // 함수를 종료한다
        }
        else // 아닐경우는
        {
            printf("%s %s, ", p->data.word, p->data.mean); // , 가 있게 출력을 한다
        }
        number2++; // 현재 출력 단어 +1을 해준다
        display(p->right); // p->right의 재귀함수를 호출한다
    }
}
```

11. p가 null 이 아닐경우 p->left의 재귀 함수를 호출한다

12. 만약 number1 == number2가 일치하면 마지막이라는 뜻이므로 , 없이 호출하고 종료한다.

13. 아닐 경우는 그냥 출력한다

14. p->right 의 재귀함수를 호출한다.

```
// 탐색 함수
TreeNode * search(TreeNode *root, Word data)
{
    TreeNode * p = root; // p를 root에 가리키게 한다
    while (p != NULL) // p가 null이 아닐때까지 반복한다
    {
        if (compare(data, p->data) == 0) // 만약 compare 함수를 통해 값이 0이 나올경우 같다는 것이다
            return p; // p를 반환한다
        else if (compare(data, p->data) < 0) // 만약 < 0 일 경우는
            p = p->left; // 왼쪽으로 이동한다
        else if (compare(data, p->data) > 0) // 만약 > 0 일 경우는
            p = p->right; // 오른쪽으로 이동한다
    }
    return p; // 탐색에 실패했을 경우 NULL 반환
}
```

15. 우선 p가 root를 가리키게 한다.

16. p가 null이 아닐때까지 반복한다

17. 만약 compare 함수에서 0이 반환되면 같다는 뜻이므로 p를 반환한다

18. 아닐 경우 < 0 이면 왼쪽으로 이동한다

19. > 0 이면 오른쪽으로 이동한다

20. 이렇게 해도 없을 경우는 null을 반환한다

```
// 새로운 노드 생성 함수
TreeNode * new_node(Word item)
{
    TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드를 크기만큼 동적할당 한다.
    temp->data = item; // 받아온 값을 data에 넣어준다
    temp->left = temp->right = NULL; // 노드의 좌 우 값을 null로 설정한다
    number1++; // 총 단어의 개수를 증가시킨다
    return temp; // 반환한다
}
```

21. temp 노드를 크기만큼 동적할당을 한다.

22. 받아온 item 값을 data에 삽입한다

23. 노드의 좌 우 값을 null로 설정한다

24. 총 단어의 개수를 증가 시키고 반환한다


```

// 트리 노드 삽입함수
TreeNode * insert_node(TreeNode * node, Word data)
{
    // 트리가 공백이면 새로운 노드를 반환한다.
    if (node == NULL)
        return new_node(data);

    // 그렇지 않으면 순환적으로 트리를 내려간다.
    if (compare(data, node->data) < 0) // 만약 현재 단어가 매개변수의 단어보다 작다면
    {
        node->left = insert_node(node->left, data); // 왼쪽으로 재귀함수를 호출한다
    }

    else if (compare(data, node->data) > 0) // 만약 현재 단어가 매개변수의 단어보다 크다면
    {
        node->right = insert_node(node->right, data); // 오른쪽으로 재귀함수를 호출한다
    }

    else if (compare(data, node->data) == 0) // 만약 단어가 같다면
    {
        // 뜻을 이어 붙여 준다
        strcat(node->data.mean, ", ");
        strcat(node->data.mean, data.mean);
        return node; // 루트 포인터를 반환한다.
    }

    // 루트 포인터를 반환한다.
    return node;
}

```

25. 만약 처음에 트리가 공백이라면 새로운 노드를 삽입한 후 반환한다
26. 아닐경우 compare 함수를 호출한 후 < 0 일 경우 노드의 왼쪽에 insert_node 함수를 재귀 호출한다
27. > 0 일 경우 노드의 오른쪽에 insert_node 함수를 재귀 호출한다
28. 만약 같을 경우는 strcat를 이용해 뜻을 붙여 준 후 루트 포인터를 반환한다

```

// 단말 노드 찾기 함수
TreeNode * min_value_node(TreeNode *node)
{
    TreeNode * current = node; // current 노드가 node 를 가리키게 한다
    // 맨 왼쪽 단말 노드를 찾으려 내려감
    while (current->left != NULL)
        current = current->left;
    return current;
}

```

29. 단말 노드를 찾기 위해서 current 노드가 node를 가리키게 한 후

30. 맨 왼쪽 단말 노드를 찾으러 내려간다.
31. current의 왼쪽이 널이 아닐 경우 계속 내려간다.
32. null이라면 current를 반환한다.

```
// 제거 함수 재귀함수 버전
TreeNode* delete_node(TreeNode* root, Word key)
{
    if (root == NULL)
        return root;

    // 만약 키가 루트보다 작으면 왼쪽 서브 트리에 있는 것임
    if (compare(key, root->data) < 0)
        root->left = delete_node(root->left, key);
    // 만약 키가 루트보다 크면 오른쪽 서브 트리에 있는 것임
    else if (compare(key, root->data) > 0)
        root->right = delete_node(root->right, key);
    // 키가 루트와 같으면 이 노드를 삭제하면 됨
    else
    {
        // 첫 번째나 두 번째 경우
        if (root->left == NULL)
        {
            TreeNode* temp = root->right; // temp를 root->right 를 가리키게 한다
            free(root); // root 를 제거한다
            return temp; // temp를 리턴한다
        }
        else if (root->right == NULL)
        {
            TreeNode* temp = root->left; // temp를 root->left 를 가리키게 한다
            free(root); // root 를 제거한다
            return temp; // temp를 리턴한다
        }
        // 세 번째 경우
        TreeNode* temp = min_value_node(root->right);

        // 중위 순회시 후계 노드를 복사한다.
        root->data = temp->data;
        // 중위 순회시 후계 노드를 삭제한다.
        root->right = delete_node(root->right, temp->data);
    }
    return root;
}
```

33. 만약 root가 null이라면 그냥 반환한다.
34. 만약 키가 루트 보다 작다면 왼쪽 서브트리로 이동한다

35. 이동 한 후 delete_node를 재귀 호출한다.
36. 만약 키가 루트 보다 크다면 오른쪽 서브트리로 이동한다
37. 이동 한 후 delete_node를 재귀 호출한다.
38. 둘 다 아닐 경우에는 맨 왼쪽 단말노드에 간 후
39. 중위 순회를 하면서 후위 노드를 복사한 후 제거한다.

```
// 메인 함수
int main(void)
{
    FILE *fp; // 파일 포인터 fp를 선언한다
    // strcpy를 위한 문자열 변수를 선언한다
    char temp_check[100];
    char temp_word[100];
    char temp_mean[100];
    Word *test; // test 구조체 포인터를 선언한다
    TreeNode * root = NULL; // root 노드를 선언한다
    TreeNode * tmp; // 임시 저장 노드를 선언한다
    // 정수형 변수를 선언한다
    int size = 0;
    int i = 0;
    int cnt = 0;

    fp = fopen("data.txt", "r"); // 파일을 오픈한다

    // 파일 오픈 실패시 오류 메시지를 출력하고 종료한다
    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return 0;
    }
}
```

40. 파일 포인터 fp를 선언한다.
41. 문자열을 삽입하기 위해서는 strcpy를 통해 삽입해야 하기 때문에 문자열 변수 3개를 만들어 준다.
42. test 구조체 포인터를 선언한다.

43. root 노드를 만든 후 null로 지정한다
44. 임시 저장 노드 tmp를 만든다.
45. 정수형 변수들을 선언한다.
46. 파일을 오픈 한 후 만약 오픈이 실패하면 오류 출력을 한 후 종료한다.

```
// 파일 끝까지 만족한다
while (!feof(fp))
{
    fgets(temp_check, 100, fp); // 한줄을 읽는다
    cnt++; // cnt 크기 증가
}
rewind(fp); // 파일포인터를 처음으로 돌린다
test = (Word*)malloc(sizeof(Word) * cnt); // cnt 개수만큼 동적할당을 해준다
```

47. 파일 끝까지 반복하면서 한줄 씩 읽는다.
48. 크기를 증가 시켜준다.
49. 파일 포인터를 처음으로 돌려 준 후 cnt개수만큼 동적할당을 해준다.

```
// 파일 끝까지 반복을 한다
while (!feof(fp))
{
    fscanf(fp, "%s ", temp_check); // 처음 값을 입력받는다
    // 만약 처음 값이 i 일 경우는
    if (strcmp(temp_check, "i") == 0)
    {
        fscanf(fp, "%s %s\n", temp_word, temp_mean); // 단어의 이름과 뜻을 저장한다
        printf("%s\n", temp_check); // i 를 출력한다
        size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
        test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
        size = strlen(temp_mean); // 단어의 뜻의 크기를 계산한다
        test[i].mean = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 뜻의 크기 + 1 만큼 동적할당한다
        // 단어 출력
        printf("단어 : %s\n", temp_word);
        printf("의미 : %s\n", temp_mean);
        // strcpy로 구조체에 넣어준다
        strcpy(test[i].word, temp_word);
        strcpy(test[i].mean, temp_mean);
        // insert_node 를 이용해 root에 데이터를 삽입한다
        root = insert_node(root, test[i]);
        printf("\n");
    }
}
```

50. 파일 끝까지 반복한다
51. 처음 알파벳을 읽어 온다.

52. 만약 i라면 삽입하라는 뜻이기 때문에 단어와 뜻을 저장한다.
53. 단어와 뜻의 크기를 파악한 후 각 크기 + 1 만큼 구조체 i번째에 동적할당을 한다.
54. 단어를 출력한 후 strcpy로 구조체에 넣어 준다.
55. insert_node 함수를 호출하고 종료한다.

```
// 만약 처음 값이 p 일 경우는
if (strcmp(temp_check, "p") == 0)
{
    printf("%s\n", temp_check); // p를 출력한다
    display(root); // root를 display로 사전을 모두 출력한다
    printf("\n");
}
```

56. 만약 p일 경우는 root를 display로 보내준다.

```
// 만약 처음 값이 d 일 경우는
if (strcmp(temp_check, "d") == 0)
{
    fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
    printf("%s\n", temp_check); // d를 출력한다
    printf("단어 : %s\n", temp_word); // 단어를 출력한다
    size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
    test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
    strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
    root = delete_node(root, test[i]); // delete_node 를 이용해 root에 데이터를 제거한다
    printf("\n");
    number1--; // 총 단어 개수 -1을 한다
}
```

57. 만약 처음 값이 d일 경우는 txt 파일에서 단어를 읽어 온다
58. 단어의 크기를 계산한 후 i번째 구조체에 크기 + 1 만큼 동적할당을 해준다
59. strcpy로 구조체에 넣어 준 후 delete_node를 호출한다.
60. 총 데이터 개수에서 -1을 한다.

```

// 만약 처음 값이 d 일 경우는
if (strcmp(temp_check, "s") == 0)
{
    fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
    printf("%s\n", temp_check); // s를 출력한다
    printf("단어 : %s\n", temp_word); // 단어를 출력한다
    size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
    test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
    strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
    tmp = search(root, test[i]); // search 를 이용해 데이터를 찾는다
    if (tmp != NULL) // 만약 tmp 가 null 이 아닐 경우에는
    {
        printf("의미 : %s\n", tmp->data.mean); // 뜻을 출력한다
    }
    else
        printf("단어를 찾을 수 없습니다\n");

    printf("\n");
}

```

61. 처음 값이 s라면 txt 파일에서 단어를 읽은 후
62. 단어의 크기를 계산한 후 i번째 구조체에 크기 + 1 만큼 동적할당을 해준다
63. strcpy로 구조체에 넣어 준 후 serach를 호출해준다.
64. 만약 tmp가 null이 아니라면 뜻을 출력하고 null 이라면 못찾는다는 출력한다.

```

// 만약 처음 값이 q 일 경우는
if (strcmp(temp_check, "q") == 0)
{
    printf("\n%s\n", temp_check); // q를 출력하고
    break; // 종료한다
}

i++; // i값을 증가시킨다

```

65. q가 들어오면 종료한다
66. 이 과정들이 끝나고 반복문을 다시 시작하기 전에 i 값을 증가시킨다.

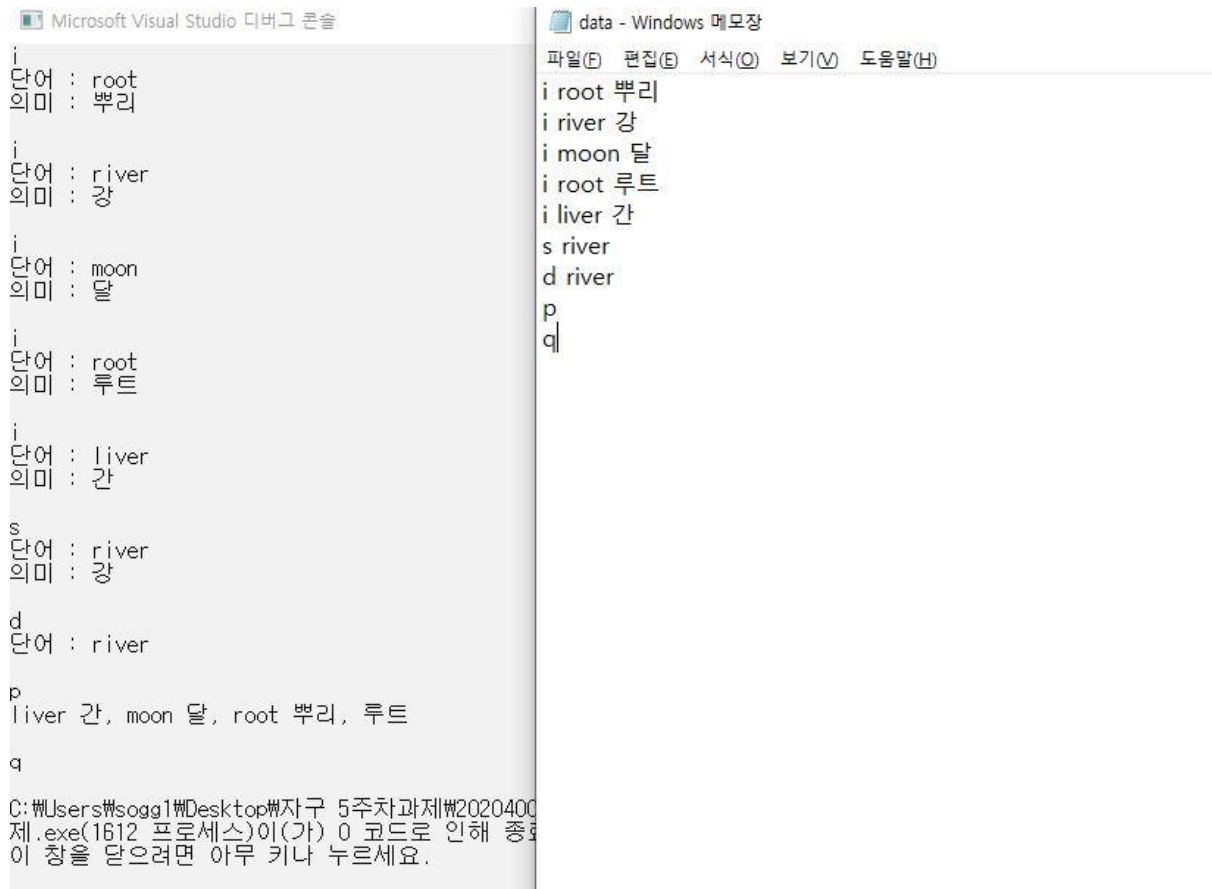
```

fclose(fp); // 파일을 닫는다
free(test); // test를 동적할당 해제를 해준다
free(root); // root를 동적할당 해제를 해준다
return 0;

```

69. 파일을 닫은 후 동적할당을 해제하고 종료한다.

1.4 실행 화면



The image shows two windows side-by-side. The left window is titled 'Microsoft Visual Studio 디버그 콘솔' (Microsoft Visual Studio Debug Console) and displays the following text:

```
i
단어 : root
의미 : 뿌리

i
단어 : river
의미 : 강

i
단어 : moon
의미 : 달

i
단어 : root
의미 : 루트

i
단어 : liver
의미 : 간

s
단어 : river
의미 : 강

d
단어 : river

p
liver 간, moon 달, root 뿌리, 루트

q
```

The right window is titled 'data - Windows 메모장' (data - Windows Notepad) and displays the following text:

```
i root 뿌리
i river 강
i moon 달
i root 루트
i liver 간
s river
d river
p
q
```

At the bottom of the Visual Studio window, a status bar shows the file path: C:\Users\sogg1\Desktop\자구 5주차과제\2020400제.exe(1612 프로세스)이(가) 0 코드로 인해 종료되었습니다. 이 창을 닫으려면 아무 키나 누르세요.

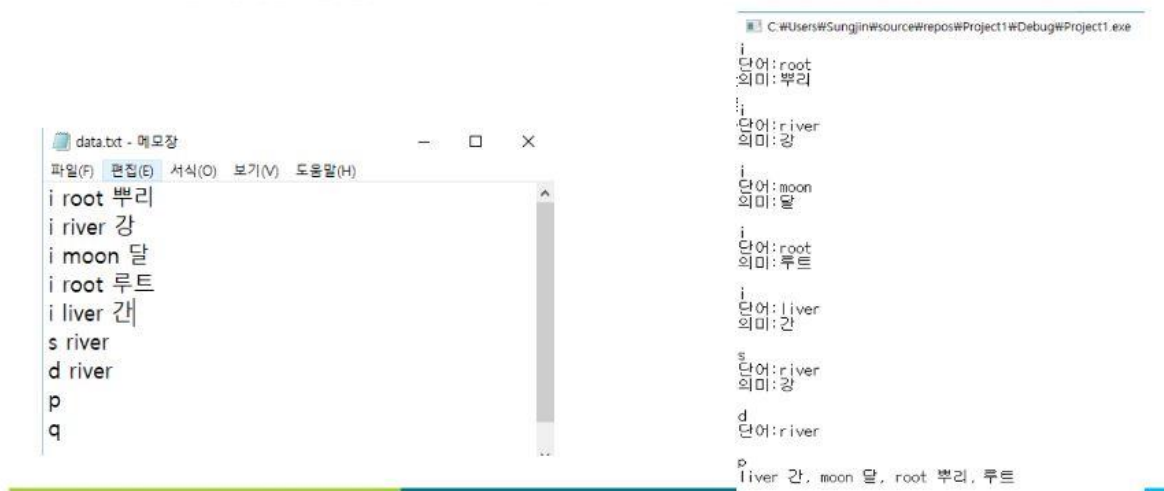
1.5 느낀점

이번 과제는 계속 언급하셨던 단어의 크기만큼 동적할당 하는 것을 메인으로 잡아야 한다고 생각했다. 또한 수업시간에 설명해 주셨던 코드들을 응용해서 적용시키는 방법을 계속 생각했다. 단어가 아닌 그냥 숫자로 대입을 하면 쉬웠으나 단어에다가 동적할당까지 추가하려고 하니 더 복잡해졌다. 그래서 이번에는 그림을 조금만 그려보고 최대한 코드를 수정하면서 디버깅 작업을 더 많이 했다. 코드는 복잡하더라도 다행히 실행화면이 출력되었다. 이번 과제에서 제일 중요한 부분은 트리를 만든 후 크기에 따라 어디에 배치할 것인지. 또한 트리를 이용해 크기를 비교해가면서 찾아가는 탐색, 제거 부분이 가장 중요하다고 느꼈다. 이것을 통해 추후 다른곳에 적용 시킬 수 있다고 생각했다. 지금 한 과제는 순환으로 만든 과제이다. 다음에는 반복으로 만들어야겠다고 생각했다.

2.1 문제 분석

이진 트리

- P. 312의 프로그램 8.14를 활용하여 이진 탐색 트리를 이용한 영어 사전 프로그램을 작성하시오.
- data.txt 파일에 양식, 단어, 뜻이 저장되어 있음
- i는 데이터를 삽입, d는 데이터 삭제, s는 데이터의 탐색을 의미
- p는 전체 데이터의 출력, q는 프로그램 종료를 의미
- 중복되는 데이터가 삽입되는 경우 기존의 단어에 뜻이 추가되어야 함. 뜻은 연결 리스트의 형태로 크기에 따라 동적할당으로 구현



앞에서 한 과제와 문제는 동일하다. 이번에는 삽입, 탐색, 제거를 순환이 아닌 반복을 통해 구현하는 것이 문제이다.

우선 단어 구조체를 만들어야 한다. 이번에는 단어와 뜻이 들어가야 하고 2개는 모두 동적할당으로 단어의 크기에 맞게 공간이 할당되어야 하기 때문에 문자열 포인터로 선언을 해야 한다.

다음으로는 트리 구조체를 만들어야 한다. 트리 구조체에는 단어 구조체 data와 트리의 좌 우 연결 링크가 필요하다.

다음으로는 단어의 크기를 비교하는 함수가 필요하다. 영어 사전에서는 알파벳이 빠른 것이 앞으로 오기 때문에 단어 크기를 비교해서 strcmp를 이용해 함수를 만든다

단어 사전 프린트 함수가 필요하다. 중위 순회로 제작을 하고 마지막 단어이면 ,

가 빠진 상태로 출력을 하는 방식을 고려해야 한다.

다음은 탐색 함수이다. 우선 탐색 함수는 순환이 아닌 반복으로 제작한다. 트리와 단어를 받은 후 둘을 비교해 같을 경우는 출력 아닐 경우는 둘의 크기를 비교해 트리 노드의 좌 혹은 우로 이동하면서 단어를 찾을 수 있는 함수를 만든다. 없을 경우는 없다 라고 표시까지 나와야 한다. 순환과 다르게 node를 다시 호출하는 것이 아닌 직접적으로 옮긴다.

다음으로는 노드 삽입 함수가 필요하다. 노드 삽입은 우선 빈 노드일 경우는 노드를 동적할당해 데이터의 값을 넣어주고 만약 아닐 경우는 크기를 비교해 좌 우 링크로 이동해 데이터를 넣어주는 함수를 만들어 준다.

제거 함수를 위한 단말 노드 찾는 함수를 만들어야 한다. Node를 받은 후 왼쪽 단말 노드를 찾으러 계속 내려가는 함수이다.

다음으로는 제거 함수를 만들어야 한다. 이번에는 반복으로 제작한다. 우선 필요한 노드들을 만들어 준 후 t가 null이 아니면서 동시에 t->data와 데이터가 같지 않을 때 까지 반복하면서 크기에 따라 좌 우로 이동시킨다. 그 후 단말노드 혹은 자식이 하나이거나 두개일 때 모두 경우를 나눠서 반복을 하면서 제거를 한 후 해제시키는 함수이다.

메인 함수에서는 우선 단어의 크기만큼 동적할당을 해줘야 하고 while 문을 이용해 파일의 끝까지 읽으면서 해당하는 알파벳에 맞게 동작을 시켜주는 반복문을 제작해야 한다.

2.2 소스 코드

```
1  //-----
2  // 제작일 : 21년 09월 29일 ~ 10월 4일
3  // 제작자 : 20204005 김필중
4  // 프로그램명 : 이진트리 영어 사전 프로그램 반복 이용
5  //-----
6
7  // 필요한 헤더파일을 선언한다.
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <memory.h>
12
13 // 오류 방지 구문을 선언한다
14 #pragma warning (disable : 4996)
15
16 // 전역 변수를 선언한다
17 // 사전 전체 출력시 마지막 단어일 경우 , 제거를 위함
18 int number1 = 0;
19 int number2 = 1;
20
21 // Word 구조체를 선언한다
22 typedef struct word
23 {
24     char *word; // 단어의 크기만큼 동적할당 하기 위해 포인터 선언
25     char *mean; // 뜻의 크기만큼 동적할당 하기 위해 포인터 선언
26 }Word;
27
28 // Tree 구조체 선언한다
29 typedef struct TreeNode
30 {
31     Word data; // 단어 구조체 data 변수 선언
32     struct TreeNode *left, *right; // 자기참조를 통해 트리의 좌우 링크 설정
33 }TreeNode;
34
35 // 단어 2개의 크기를 비교하는 함수
36 int compare(Word e1, Word e2)
37 {
38     return strcmp(e1.word, e2.word); // 받아온 단어 2개의 크기를 비교해 리턴
39 }
40
```

```

41 // 단어 사전 프린트 함수
42 void display(TreeNode *p)
43 {
44     if (p != NULL) // p가 NULL 이 아닐경우
45     {
46         display(p->left); // p->left의 재귀함수를 호출한다
47         if (number1 == number2) // 만약 number1(사전의 총 단어의 개수) == number 2(현재 출력한 단어의 개수) 가 일치하면
48         {
49             printf("%s %s", p->data.word, p->data.mean); // , 없이 출력한다
50             return; // 함수를 종료한다
51         }
52     }
53     else // 아닐경우는
54     {
55         printf("%s %s, ", p->data.word, p->data.mean); // , 가 있게 출력을 한다
56     }
57     number2++; // 현재 출력 단어 +1을 해준다
58     display(p->right); // p->right의 재귀함수를 호출한다
59 }
60
61 // 탐색 함수 반복
62 TreeNode * search(TreeNode *node, Word data)
63 {
64     while (node != NULL) // node 가 null 일때까지 반복한다
65     {
66         if (node != NULL) // node 가 null 이 아니라면
67         {
68             if (compare(data, node->data) == 0) // 2개의 단어를 비교해서 같으면
69                 return node; // 현재 노드 반환
70             else if (compare(data, node->data) < 0) // 단어를 비교해 작을 경우는
71                 node = node->left; // 왼쪽으로 이동
72             else // 아닐경우는
73                 node = node->right; // 오른쪽으로 이동
74         }
75     }
76     return NULL; // 없을 경우는 null 반환
77 }
78
79

```

```

80 // 트리 노드 삽입함수
81 void insert_node(TreeNode ** node, Word data)
82 {
83     TreeNode *p, *t; // p는 부모노드 t는 현재노드
84     TreeNode *n; // n은 새로운 노드
85
86     t = *node; // t를 node를 가리키게 한다
87     p = NULL; // p는 null로 잡는다
88
89
90     while (t != NULL) // t 노드가 null이 아닐경우에 반복
91     {
92         if (compare(data, t->data) == 0) // 만약 현재 단어가 같을경우는
93         {
94             // 뜻을 이어 붙여 준다
95             strcat(t->data.mean, ", ");
96             strcat(t->data.mean, data.mean);
97             return; // 반환한다
98         }
99         p = t; // 부모노드가 현재 노드를 가리키게 한다
100         if (compare(data, t->data) < 0) // 만약 작을 경우는
101             t = t->left; // 왼쪽으로 이동
102         else // 아닐경우는
103             t = t->right; // 오른쪽으로 이동
104     }
105
106     n = (TreeNode *)malloc(sizeof(TreeNode)); // 동적할당을 해준다
107     // 동적할당 확인 후 실패시 종료
108     if (n == NULL)
109         return;
110
111     n->data = data; // 새로운 노드에 데이터 삽입
112     n->left = n->right = NULL; // 좌 우는 널값으로
113
114     if (p != NULL) // p가 널이 아닐경우는
115     {
116         if (compare(data, p->data) < 0) // 작을 경우
117             p->left = n; // 왼쪽으로 연결
118         else // 클 경우는
119             p->right = n; // 오른쪽으로 연결
120     }

```

```

119         p->right = n; // 오른쪽으로 연결
120     }
121
122     else
123     {
124         *node = n; // *node를 n을 가리키게 한다
125         number1++; // 총 개수를 더한다
126     }
127
128     //
129     TreeNode * min_value_node(TreeNode *node)
130     {
131         TreeNode * current = node; // current 노드가 node 를 가리키게 한다
132         // 맨 왼쪽 단말 노드를 찾으려 내려감
133         while (current->left != NULL)
134             current = current->left;
135         return current;
136     }
137
138     // 제거 함수 반복 버전
139     void delete_node(TreeNode **root, Word key)
140     {
141         TreeNode *p, *child, *succ, *succ_p, *t;
142         //key를 갖는 노드 t를 탐색, p는 t의 부모노드
143
144         p = NULL; // p를 nil값으로 한다
145         t = *root; // t를 *root로 한다
146
147         // t가 null이 아니면서 t->data와 key가 같지 않을때 까지 반복한다
148         while (t != NULL && compare(t->data, key) != 0)
149         {
150             p = t; // p가 t를 가리키기 한다
151             t = (compare(key, t->data) < 0) ? t->left : t->right; // t는 key t->data와 비교해 작으면 t->left로 이동 아니면 오른쪽으로 이동한다
152         }

```

```

153         if (t == NULL) // t가 null일 경우는
154         {
155             //탐색 트리에 없는 키
156             return;
157         }
158
159         //단말노드인 경우
160         if ((t->left == NULL) && (t->right == NULL)) // t의 좌 우가 없을경우
161         {
162             if (p != NULL) // p가 null이 아닐경우
163             {
164                 if (p->left == t) // p의 왼쪽이 t랑 같을 경우
165                     p->left = NULL; // p의 왼쪽을 null로 한다
166                 else // 아닐경우
167                     p->right = NULL; // p의 오른쪽을 null로 한다
168             }
169             else //부모노드가 없으면 루트
170                 *root = NULL;
171         }
172
173         //하나의 자식만 가지는 경우
174         else if ((t->left == NULL) || (t->right == NULL)) // 둘 중 하나만 null일 경우
175         {
176             child = (t->left != NULL) ? t->left : t->right; // child를 t->left가 null이 아니라면 왼쪽 아닐경우 오른쪽을 가리키게 한다
177             if (p != NULL) // 만약 null이 아닐경우
178             {
179                 if (p->left == t) // 왼쪽과 p가 같을 경우
180                     p->left = child; // 왼쪽을 child를 가리키게 한다
181                 else
182                     p->right = child; // 오른쪽을 child를 가리키게 한다
183             }
184             else //부모노드가 없으면 루트
185                 *root = child;
186         }
187     }

```

```

188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211

```

```

else
{ //두개의 자식을 가지는 경우
  //오른쪽 서브트리에서 후속자를 찾는다.
  succ_p = t;
  succ = t->right;
  //후속자를 찾아서 계속 왼쪽으로 이동한다.
  while (succ->left != NULL)
  {
    succ_p = succ;
    succ = succ->left;
  }
  //후속자의 부모와 자식을 연결
  if (succ_p->left = succ)
    succ_p->left = succ->right;
  else
    succ_p->right = succ->right;
  //후속자를 현재 노드로 이동한다.
  t->data = succ->data;
  t = succ;
}

free(t); // 해제시킨다
}

```



```

213 // 메인 함수
214 int main(void)
215 {
216     FILE *fp; // 파일 포인터 fp를 선언한다
217     // strcpy를 위한 문자열 변수를 선언한다
218     char temp_check[100];
219     char temp_word[100];
220     char temp_mean[100];
221     Word *test; // test 구조체 포인터를 선언한다
222     TreeNode * root = NULL; // root 노드를 선언한다
223     TreeNode * tmp; // 임시 저장 노드를 선언한다
224     // 정수형 변수를 선언한다
225     int size = 0;
226     int i = 0;
227     int cnt = 0;
228
229     fp = fopen("data.txt", "r"); // 파일을 오픈한다
230
231     // 파일 오픈 실패시 오류 메시지를 출력하고 종료한다
232     if (fp == NULL)
233     {
234         printf("파일 오픈 실패\n");
235         return 0;
236     }
237
238     // 파일 끝까지 반복한다
239     while (!feof(fp))
240     {
241         fgets(temp_check, 100, fp); // 한줄을 읽는다
242         cnt++; // cnt 크기 증가
243     }
244     rewind(fp); // 파일포인터를 처음으로 돌린다
245     test = (Word*)malloc(sizeof(Word) * cnt); // cnt 개수만큼 동적할당을 해준다
246
247

```

```

248 // 파일 끝까지 반복을 한다
249 while (!feof(fp))
250 {
251     fscanf(fp, "%s ", temp_check); // 처음 값을 입력받는다
252     // 만약 처음 값이 i 일 경우는
253     if (strcmp(temp_check, "i") == 0)
254     {
255         fscanf(fp, "%s %s\n", temp_word, temp_mean); // 단어의 이름과 뜻을 저장한다
256         printf("%s\n", temp_check); // i 를 출력한다
257         size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
258         test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
259         size = strlen(temp_mean); // 단어의 뜻의 크기를 계산한다
260         test[i].mean = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 뜻의 크기 + 1 만큼 동적할당한다
261         // 단어 출력
262         printf("단어 : %s\n", temp_word);
263         printf("의미 : %s\n", temp_mean);
264         // strcpy로 구조체에 넣어준다
265         strcpy(test[i].word, temp_word);
266         strcpy(test[i].mean, temp_mean);
267         // insert_node 를 이용해 root에 데이터를 삽입한다
268         insert_node(&root, test[i]);
269         printf("\n");
270     }
271
272     // 만약 처음 값이 p 일 경우는
273     if (strcmp(temp_check, "p") == 0)
274     {
275         printf("%s\n", temp_check); // p를 출력한다
276         display(root); // root를 display로 사건을 모두 출력한다
277         printf("\n");
278     }
279     // 만약 처음 값이 i 일 경우는

```



```

279 // 만약 처음 값이 d 일 경우는
280 if (strcmp(temp_check, "d") == 0)
281 {
282     fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
283     printf("%s\n", temp_check); // d를 출력한다
284     printf("단어 : %s\n", temp_word); // 단어를 출력한다
285     size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
286     test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
287     strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
288     delete_node(&root, test[i]); // delete_node 를 이용해 root에 데이터를 제거한다
289     printf("\n");
290     number1--; // 총 단어 개수 -1을 한다
291 }
292 // 만약 처음 값이 s 일 경우는
293 if (strcmp(temp_check, "s") == 0)
294 {
295     fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
296     printf("%s\n", temp_check); // s를 출력한다
297     printf("단어 : %s\n", temp_word); // 단어를 출력한다
298     size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
299     test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
300     strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
301     tmp = search(root, test[i]); // search 를 이용해 데이터를 찾는다
302     if (tmp != NULL) // 만약 tmp 가 null 이 아닐 경우에는
303     {
304         printf("의미 : %s\n", tmp->data.mean); // 뜻을 출력한다
305     }
306     else
307         printf("단어를 찾을 수 없습니다.\n");
308     printf("\n");
309 }
310

```

```

311 // 만약 처음 값이 q 일 경우는
312 if (strcmp(temp_check, "q") == 0)
313 {
314     printf("\n%s\n", temp_check); // q를 출력하고
315     break; // 종료한다
316 }
317
318 i++; // i값을 증가시킨다
319 }
320
321
322
323
324 fclose(fp); // 파일을 닫는다
325 free(test); // test를 동적할당 해제를 해준다
326 free(root); // root를 동적할당 해제를 해준다
327 return 0;
328 }

```

2.3 소스 코드 분석

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

1. 필요한 헤더파일을 선언한다.

```
// 오류 방지 구문을 선언한다  
#pragma warning (disable : 4996)  
  
// 전역 변수를 선언한다  
// 사전 전체 출력시 마지막 단어일 경우 , 제거를 위함  
int number1 = 0;  
int number2 = 1;
```

2. 오류 방지 구문을 선언한다

3. 사전 프린트 함수에서 마지막을 찾기 위해 전역변수를 선언한다

```
// Word 구조체를 선언한다  
typedef struct word  
{  
    char *word; // 단어의 크기만큼 동적할당 하기 위해 포인터 선언  
    char *mean; // 뜻의 크기만큼 동적할당 하기 위해 포인터 선언  
}Word;
```

4. Word 구조체를 선언한다

5. 구조체 안에 있는 문자열 변수들은 모두 포인터로 선언해서 동적할당을 하기 위한다.

```
// Tree 구조체 선언한다
typedef struct TreeNode
{
    Word data; // 단어 구조체 data 변수 선언
    struct TreeNode *left, *right; // 자기참조를 통해 트리의 좌우 링크 설정
}TreeNode;
```

6. 트리 구조체를 선언한다

7. 단어 구조체 data를 선언하고

8. 자기 참조를 통해 트리의 좌 우 링크를 설정한다.

```
// 단어 2개의 크기를 비교하는 함수
int compare(Word e1, Word e2)
{
    return strcmp(e1.word, e2.word); // 받아온 단어 2개의 크기를 비교해 리턴
}
```

9. 단어 2개를 비교하는 함수를 만든다

10. 단어 2개를 받아와 strcmp를 통해 나오는 값을 반환한다.

```
// 단어 사전 프린트 함수
void display(TreeNode *p)
{
    if (p != NULL) // p가 NULL 이 아닐경우
    {
        display(p->left); // p->left의 재귀함수를 호출한다
        if (number1 == number2) // 만약 number1(사전의 총 단어의 개수) == number 2(현재 출력한 단어의 개수) 가 일치하면
        {
            printf("%s %s", p->data.word, p->data.mean); // , 없이 출력한다
            return; // 함수를 종료한다
        }
        else // 아닐경우는
        {
            printf("%s %s, ", p->data.word, p->data.mean); // , 가 있게 출력을 한다
        }
        number2++; // 현재 출력 단어 +1을 해준다
        display(p->right); // p->right의 재귀함수를 호출한다
    }
}
```

11. p가 null 이 아닐경우 p->left의 재귀 함수를 호출한다

12. 만약 number1 == number2가 일치하면 마지막이라는 뜻이므로 , 없이 호출하고 종료한다.

13. 아닐 경우는 그냥 출력한다

14. p->right 의 재귀함수를 호출한다

```
// 탐색 함수 반복
TreeNode * search(TreeNode *node, Word data)
{
    while (node != NULL) // node 가 null 일때까지 반복한다
    {
        if (node != NULL) // node 가 null 이 아니라면
        {
            if (compare(data, node->data) == 0) // 2개의 단어를 비교해서 같으면
                return node; // 현재 노드 반환
            else if (compare(data, node->data) < 0) // 단어를 비교해 작을 경우는
                node = node->left; // 왼쪽으로 이동
            else // 아닐경우는
                node = node->right; // 오른쪽으로 이동
        }
    }
    return NULL; // 없을 경우는 null 반환
}
```

15. 우선 node 가 null 일 때까지 반복을 한다

16. null이 아니라면 2개의 단어를 비교해서 같으면 현재 노드 반환

17. 작을 경우는 왼쪽으로 이동 큰 경우는 오른쪽으로 이동한다

```

// 트리 노드 삽입함수
void insert_node(TreeNode ** node, Word data)
{
    TreeNode *p, *t; // p는 부모노드 t는 현재노드
    TreeNode *n; // n은 새로운 노드

    t = *node; // t를 node를 가리키게 한다
    p = NULL; // p는 null로 잡는다

    while (t != NULL) // t 노드가 null이 아닐경우에 반복
    {
        if (compare(data, t->data) == 0) // 만약 현재 단어가 같을경우는
        {
            // 뜻을 이어 붙여 준다
            strcat(t->data.mean, ", ");
            strcat(t->data.mean, data.mean);
            return; // 반환한다
        }
        p = t; // 부모노드가 현재 노드를 가리키게 한다
        if (compare(data, t->data) < 0) // 만약 작을 경우는
            t = t->left; // 왼쪽으로 이동
        else // 아닐경우는
            t = t->right; // 오른쪽으로 이동
    }
}

```

18. 부모노드 현재노드 새로운 노드를 만든다

19. t = *node를 가리키게 한 후 p는 null로 잡는다

20. t가 null이 아닐경우에 반복을 해준다. 만약 같으면 뜻을 이어 붙인다

21. 작을 경우는 왼쪽으로 이동 클 경우는 오른쪽으로 이동한다


```

n = (TreeNode *)malloc(sizeof(TreeNode)); // 동적할당을 해준다
// 동적할당 확인 후 실패시 종료
if (n == NULL)
    return;

n->data = data; // 새로운 노드에 데이터 삽입
n->left = n->right = NULL; // 좌 우는 널값으로

if (p != NULL) // p가 널이 아닐경우는
{
    if (compare(data, p->data) < 0) // 작을 경우
        p->left = n; // 왼쪽으로 연결
    else // 클 경우는
        p->right = n; // 오른쪽으로 연결
}

else
    *node = n; // *node를 n을 가리키게 한다
number1++; // 총 개수를 더한다

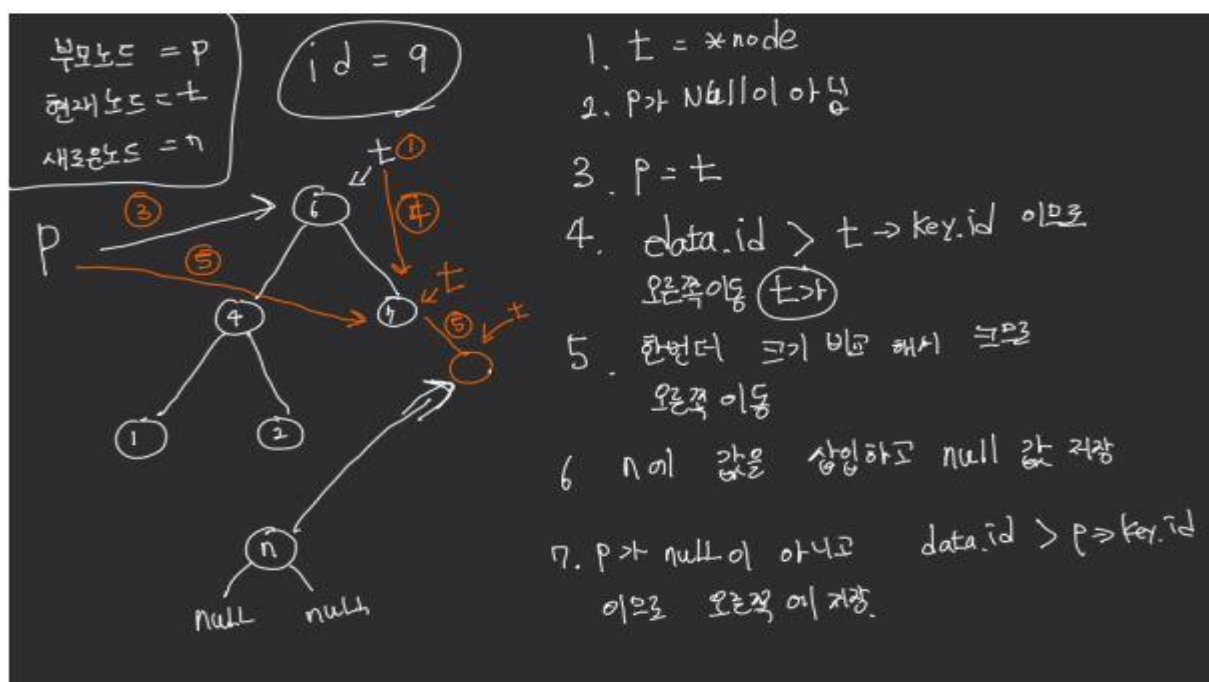
```

22. 동적할당을 해준 후 동적할당을 확인하고 실패 시 종료한다

23. 새로운 노드에 데이터를 삽입하고 좌 우는 null로 잡아준다

24. p가 null이 아니라면 작을 경우는 왼쪽으로 연결하고 큰 경우는 오른쪽으로 연결한다

25. 그 후 node가 n을 가리키게 한다. 전체 사전의 개수를 더한다



```

TreeNode * min_value_node(TreeNode *node)
{
    TreeNode * current = node; // current 노드가 node 를 가리키게 한다
    // 맨 왼쪽 단말 노드를 찾으러 내려감
    while (current->left != NULL)
        current = current->left;
    return current;
}

```

26. 단말 노드를 찾기 위해서 current 노드가 node를 가리키게 한 후

27. 맨 왼쪽 단말 노드를 찾으러 내려간다.

28. current의 왼쪽이 널이 아닐 경우 계속 내려간다.

29. null이라면 current를 반환한다.

```

// 제거 함수 반복 버전
void delete_node(TreeNode **root, Word key)
{
    TreeNode *p, *child, *succ, *succ_p, *t;
    //key를 갖는 노드 t를 탐색, p는 t의 부모노드

    p = NULL; // p를 널값으로 한다
    t = *root; // t를 *root로 한다

    // t가 null이 아니면서 t->data와 key가 같지 않을때 까지 반복한다
    while (t != NULL && compare(t->data, key) != 0)
    {
        p = t; // p가 t를 가리키기 한다
        t = (compare(key, t->data) < 0) ? t->left : t->right; // t는 key t->data와 비교해 작으면 t->left로 이동 아니면 오른쪽으로 이동한다
    }

    if (t == NULL) // t가 null일 경우는
    {
        //탐색 트리에 없는 키
        return;
    }
}

```

30. 필요한 노드들을 선언한다.

31. p를 null값으로 한 후 t = *root 로 한다

32. t가 null이 아니고 t->data가 key가 같지 않을 때까지 반복을 한다

33. p = t를 한 후 크기를 비교해 작을 경우 왼쪽 큰 경우는 오른쪽으로 이동한다

34. 만약 null이면 데이터가 없으므로 종료한다


```

//단말노드인 경우
if ((t->left == NULL) && (t->right == NULL)) // t의 좌 우가 없을 경우
{
    if (p != NULL) // p가 null이 아닐 경우
    {
        if (p->left == t) // p의 왼쪽이 t랑 같을 경우
            p->left = NULL; // p의 왼쪽을 null로 한다
        else // 아닐 경우
            p->right = NULL; // p의 오른쪽을 null로 한다
    }
    else //부모노드가 없으면 루트
        *root = NULL;
}

```

35. 만약 단말 노드인 경우는 좌 우가 없어야 한다.

36. p가 null이 아니고 p의 왼쪽이 t와 같은 경우는 p->left를 null로 바꾼다
아니라면 p->right를 null로 바꾼다

37. 혹은 부모 노드가 없으면 루트 반환을 해준다

```

//하나의 자식만 가지는 경우
else if ((t->left == NULL) || (t->right == NULL)) // 둘 중 하나만 null일 경우
{
    child = (t->left != NULL) ? t->left : t->right; // child를 t->left가 null이 아니라면 왼쪽 아닐 경우 오른쪽을 가리키게 한다
    if (p != NULL) // 만약 null이 아닐 경우
    {
        if (p->left == t) // 왼쪽과 p가 같을 경우
            p->left = child; // 왼쪽을 child를 가리키게 한다
        else
            p->right = child; // 오른쪽을 child를 가리키게 한다
    }
    else //부모노드가 없으면 루트
        *root = child;
}

```

38. 하나의 자식만 갖는 경우는 우선 둘 중 하나만 null이어야 한다.

39. child를 t->left가 null이 아니라면 왼쪽 아닐 경우는 오른쪽을 가리키게 한다

40. null이 아니라면 왼쪽을 child를 가리키게 하고 아니면 오른쪽을 child를 가리키게 한다

41. 부모 노드가 없으면 루트 반환을 해준다

```

else
{ //두개의 자식을 가지는 경우
//오른쪽 서브트리에서 후속자를 찾는다.
succ_p = t;
succ = t->right;
//후속자를 찾아서 계속 왼쪽으로 이동한다.
while (succ->left != NULL)
{
    succ_p = succ;
    succ = succ->left;
}
//후속자의 부모와 자식을 연결
if (succ_p->left = succ)
    succ_p->left = succ->right;
else
    succ_p->right = succ->right;
//후속자를 현재 노드로 이동한다.
t->data = succ->data;
t = succ;
}

free(t); // 해제시킨다
}

```

42. 이것도 아니면 2개를 가지고 있는 것이다
43. succ_p = t succ = t->right 를 가리키게 한 후
44. 후속자를 찾아서 계속 왼쪽으로 이동한다
45. 만약 null이 아니라면 계속 반복하면서 내려간다
46. null이라면 후속자의 부모와 자식을 연결한 후 현재 노드로 이동시킨다
47. 그리고 해제 시킨다.

```

// 메인 함수
int main(void)
{
    FILE *fp; // 파일 포인터 fp를 선언한다
    // strcpy를 위한 문자열 변수를 선언한다
    char temp_check[100];
    char temp_word[100];
    char temp_mean[100];
    Word *test; // test 구조체 포인터를 선언한다
    TreeNode * root = NULL; // root 노드를 선언한다
    TreeNode * tmp; // 임시 저장 노드를 선언한다
    // 정수형 변수를 선언한다
    int size = 0;
    int i = 0;
    int cnt = 0;

    fp = fopen("data.txt", "r"); // 파일을 오픈한다

    // 파일 오픈 실패시 오류 메시지를 출력하고 종료한다
    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return 0;
    }
}

```

48. 파일 포인터 fp를 선언한다.

49. 문자열을 삽입하기 위해서는 strcpy를 통해 삽입해야 하기 때문에 문자열 변수 3개를 만들어 준다.

50. test 구조체 포인터를 선언한다.

51. root 노드를 만든 후 null로 지정한다

52. 임시 저장 노드 tmp를 만든다.

53. 정수형 변수들을 선언한다.

54. 파일을 오픈 한 후 만약 오픈이 실패하면 오류 출력을 한 후 종료한다.

```
// 파일 끝까지 반복한다
while (!feof(fp))
{
    fgets(temp_check, 100, fp); // 한줄을 읽는다
    cnt++; // cnt 크기 증가
}
rewind(fp); // 파일포인터를 처음으로 돌린다
test = (Word*)malloc(sizeof(Word) * cnt); // cnt 개수만큼 동적할당을 해준다
```

55. 파일 끝까지 반복하면서 한줄 씩 읽는다.

56. 크기를 증가 시켜준다.

57. 파일 포인터를 처음으로 돌려 준 후 cnt개수만큼 동적할당을 해준다.

```
// 파일 끝까지 반복을 한다
while (!feof(fp))
{
    fscanf(fp, "%s ", temp_check); // 처음 값을 입력받는다
    // 만약 처음 값이 i 일 경우는
    if (strcmp(temp_check, "i") == 0)
    {
        fscanf(fp, "%s %s\n", temp_word, temp_mean); // 단어의 이름과 뜻을 저장한다
        printf("%s\n", temp_check); // i 를 출력한다
        size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
        test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
        size = strlen(temp_mean); // 단어의 뜻의 크기를 계산한다
        test[i].mean = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 뜻의 크기 + 1 만큼 동적할당한다
        // 단어 출력
        printf("단어 : %s\n", temp_word);
        printf("의미 : %s\n", temp_mean);
        // strcpy로 구조체에 넣어준다
        strcpy(test[i].word, temp_word);
        strcpy(test[i].mean, temp_mean);
        // insert_node 를 이용해 root에 데이터를 삽입한다
        insert_node(&root, test[i]);
        printf("\n");
    }
}
```

58. 파일 끝까지 반복한다

59. 처음 알파벳을 읽어 온다.

60. 만약 i라면 삽입하라는 뜻이기 때문에 단어와 뜻을 저장한다.

61. 단어와 뜻의 크기를 파악한 후 각 크기 + 1 만큼 구조체 i번째에 동적할당을 한다.

62. 단어를 출력한 후 strcpy로 구조체에 넣어 준다.

63. insert_node 함수를 호출하고 종료한다.

```
}  
// 만약 처음 값이 p 일 경우는  
if (strcmp(temp_check, "p") == 0)  
{  
    printf("%s\n", temp_check); // p를 출력한다  
    display(root); // root를 display로 사전에 모두 출력한다  
    printf("\n");  
}
```

64. 만약 p일 경우는 root를 display로 보내준다.

```
// 만약 처음 값이 d 일 경우는  
if (strcmp(temp_check, "d") == 0)  
{  
    fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다  
    printf("%s\n", temp_check); // d를 출력한다  
    printf("단어 : %s\n", temp_word); // 단어를 출력한다  
    size = strlen(temp_word); // 단어의 이름의 크기를 계산한다  
    test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다  
    strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다  
    delete_node(&root, test[i]); // delete_node 를 이용해 root에 데이터를 제거한다  
    printf("\n");  
    number1--; // 총 단어 개수 -1을 한다  
}
```

65. 만약 처음 값이 d일 경우는 txt 파일에서 단어를 읽어 온다

66. 단어의 크기를 계산한 후 i번째 구조체에 크기 + 1 만큼 동적할당을 해준다

67. strcpy로 구조체에 넣어 준 후 delete_node를 호출한다.

68. 총 데이터 개수에서 -1을 한다.

```

// 만약 처음 값이 d 일 경우는
if (strcmp(temp_check, "s") == 0)
{
    fscanf(fp, "%s\n", temp_word); // txt 파일에 단어를 읽는다
    printf("%s\n", temp_check); // s를 출력한다
    printf("단어 : %s\n", temp_word); // 단어를 출력한다
    size = strlen(temp_word); // 단어의 이름의 크기를 계산한다
    test[i].word = (char*)malloc(sizeof(char)*(size + 1)); // i번째 구조체에 단어의 크기 + 1 만큼 동적할당한다
    strcpy(test[i].word, temp_word); // strcpy로 구조체에 넣어준다
    tmp = search(root, test[i]); // search 를 이용해 데이터를 찾는다
    if (tmp != NULL) // 만약 tmp 가 null 이 아닐경우에는
    {
        printf("의미 : %s\n", tmp->data.mean); // 뜻을 출력한다
    }
    else
        printf("단어를 찾을 수 없습니다\n");

    printf("\n");
}

```

69. 처음 값이 s라면 txt 파일에서 단어를 읽은 후
70. 단어의 크기를 계산한 후 i번째 구조체에 크기 + 1 만큼 동적할당을 해준다
71. strcpy로 구조체에 넣어 준 후 serach를 호출해준다.
72. 만약 tmp가 null이 아니라면 뜻을 출력하고 null 이라면 못찾는다는 것을 출력한다.

```

// 만약 처음 값이 q 일 경우는
if (strcmp(temp_check, "q") == 0)
{
    printf("\n%s\n", temp_check); // q를 출력하고
    break; // 종료한다
}

i++; // i값을 증가시킨다
}

```

73. q가 들어오면 종료한다
74. 이 과정들이 끝나고 반복문을 다시 시작하기 전에 i 값을 증가시킨다.

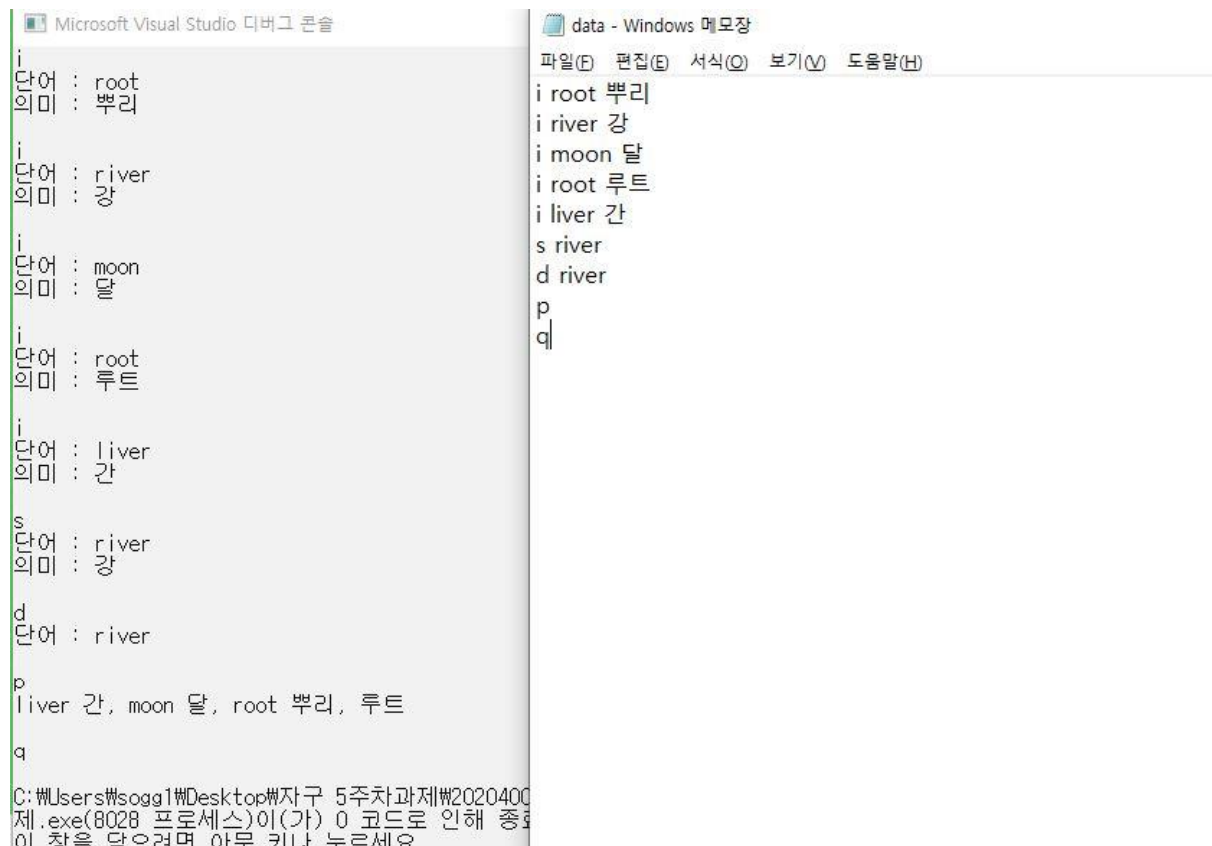
```

fclose(fp); // 파일을 닫는다
free(test); // test를 동적할당 해제를 해준다
free(root); // root를 동적할당 해제를 해준다
return 0;
}

```

75. 파일을 닫은 후 동적할당을 해제하고 종료한다.

2.4 실행 화면



```
i
단어 : root
의미 : 뿌리

i
단어 : river
의미 : 강

i
단어 : moon
의미 : 달

i
단어 : root
의미 : 루트

i
단어 : liver
의미 : 간

s
단어 : river
의미 : 강

d
단어 : river

p
liver 간, moon 달, root 뿌리, 루트

q

C:\Users\wsogg1\Desktop\자구 5주차과제\2020400
제.exe(8028 프로세스)이(가) 0 코드로 인해 종료
이 창을 닫으려면 아무 키나 누르세요
```

2.5 느낀점

이번에는 1번과는 결과가 같지만 중간에 반복문을 사용하는 프로그램을 만들었다. 확실히 전보다 코드가 길어지고 복잡해진 느낌이 든다. 하지만 반복에 제일 큰 장점이라면 순환보다 더 빠르다는 것이다. 순환은 계속 호출하고 리턴하고 하지만 반복은 그런거 없이 빠르게 반복하고 나간다는 것이다. 그러기에 코드가 복잡하지만 속도가 빠른 반복을 사용하는 것이 더 좋은 방법이라 생각한다. 이번에는 순환과는 다르게 그림을 그리면서 가는 것이 더 편했다. 내가 직접 숫자를 대입해 먼저 코드를 이해하고 접근한 후 단어로 연결하는 방식으로 코드를 작성했다. 앞으로는 순환 반복이 주어지면 몇번 더 순환 반복 두개씩 만들어 보고 확실히 체감이 될 때 그때 어떤 코드를 사용해야 할지 고민을 해야겠다고 느꼈다

