시스템SW실습2 과제2키

2016310936 우승민

이번 과제는 hash function에 의해 나온 각 단어의 key값에 따라 분류하여 hash table을 구성하는 과제입니다.

```
typedef struct db {
        char* word;
        int count;
        struct db *next;

} db_t;
```

우선 제가 만든 구조체의 구성은 넣어줄 단어, 그 단어의 개수, 다음 단어와 연결시키기 위한 포인터로 구성하였습니다.

그 후 main.c 코드를 보면 3가지를 입력해야 하는 것을 확인할 수 있습니다.

```
 if (argc != 2) {
         printf("Usage: %s size₩n", argv[0]);
         return -1;
 }

 size = atoi(argv[1]);
 DB = db_open(size);
 if (DB == NULL) {
         printf("DB not opened₩n");
         return -1;
 }
 printf("DB opened₩n");
```

첫번째는 파일 wordcount이고, 2번째는 구조체의 size, 마지막으로 추가할 단어들을 입력해 주어야합니다. size까지 입력이 완료되면 구조체를 열게 됩니다.

```
int t;

db_t *db_open(int size)
{
        db_t *db = NULL;
        db = (db_t*)malloc(sizeof(db_t)*size);
        t=size;
        int i;
        for(i=0; i<size; i++){
                db[i].word=NULL;
                db[i].next = NULL;
                }
        return db;
}
```

t 정수값을 전역변수로 사용한 것은 size의 값을 close함수에서 사용하기 위해 따로 저장한 것입니다.

open함수에서 size 크기에 해당하는 db를 만들어 주고, 각 db의 i번째 구성 값들을 초기화 시켜주었습니다.

다시 main함수로 돌아가 while문의 조건을 보면 scanf함수를 통해 단어를 key로 입력받는 것을 확인 할 수 있습니다. 만약 ret=-1일 때, 즉 파일이 끝날 때 while문이 종료됩니다.

```
while ((ret = scanf("%s", key)) != -1) {
        key_len = strlen(key);
        val = db_get(DB, key, key_len, &val_len);
        if (val == NULL) {
                printf("GET [%s] [NULL]₩n", key);
                cnt = 1;
        } else {
                printf("GET [%s] [%d]₩n", key, *((int *)val));
                cnt = *((int*)val) + 1;
                free(val);
        }
```

Char형 포인터인 val에 db_get 함수의 return값을 저장하고 값에 따라 출력하는 내용이 정해집니다. Db_get 함수를 보면 저는 hash function을 입력받는 단어들의 첫 알파벳의 ascii코드에 따라 분류하였습니다.

```
char *db_get(db_t *db, char *key, int key_len,
                        int *val_len)
{
        char *value = NULL;
        int x;
        x=key[0];
        db_t * cur = &db[x];
        if(db[x].word==NULL) return value;
        while(1){
                if(cur->next == NULL){
                        break;
                }
                if(strcmp(key,cur->word)!=0)
                        cur = cur->next;
                else
                        break;
        }
```

그 후 구조체 cur 포인터를 hash function 결과의 해당하는 db의 주소로 만들어주고, 만약 처음 입력받아 저장된 단어가 NULL이면 바로 value를 return하게 해주었습니다.

만약 처음받는 것이 아니면 구조체 안의 next 포인터를 이용하여 linked list중 해당 단어(key)의 위치에서 멈추도록 하였습니다.

만약 해당 단어(key)를 입력받은 적이 있으면 value를 int크기의 char형 포인터로 만들고

```
if(strcmp(key,cur->word)==0){
        value = (char*)malloc(sizeof(int));
        *(int*)value = cur -> count;


        }
```

구조체 안의 count값을 main 함수안의 출력값과 동일하게 *(int *)형으로 바꾸어 넣어주었습니다.

```
printf("GET [%s] [%d]₩n", key, *((int *)val));
```

```
else{
        if(cur->next == NULL)
                return value;
}
return value;
```

만약 입력받은 적이 없어 linked list 의 끝까지 간다면 새로 추가해주는 것이므로 바로 value 를 return 하게 해주었습니다.

다시 main 문으로 돌아가 다음 함수인 put 함수로 이동하면,

```
db_put(DB, key, key_len, (char *)&cnt, sizeof(int));
printf("PUT [%s] [%d]₩n", key, cnt);
```

우선 temp라는 db_t 포인터를 만들어주고 그 안에 입력받는 key를 넣어야 하므로 key 길이에 해당하는 size를 malloc해주었습니다. Count는 처음 입력받는다는 가정하에 1로 설정해주고 next는 마지막이 될 수 있으므로 NULL로 만들어 주었습니다.

```
void db_put(db_t *db, char *key, int key_len,
                        char *val, int val_len)
{
        db_t *temp = (db_t *)malloc(sizeof(db_t));
        temp -> word = (char*)malloc(sizeof(char)*(key_len+1));
        int k;
        temp -> count=1;
        temp -> next = NULL;
        for(k=0; k<key_len; k++)
                temp -> word[k]= key[k];
        int x= key[0];
        db_t *cur = &db[x];
```

Cur 포인터는 get함수와 동일하게 key의 첫 알파벳에 해당하는 위치의 db 주소를 넣어주었습니다.

```
if(*(int*)val==1){
        if(db[x].word==0){
                db[x].word=temp->word;
                db[x].count=temp->count;
                db[x].next=NULL;
                free(temp);
                return ;
                }
```

만약 입력받은 것이 처음이고 처음 db에 단어가 없으면 해당 위치의 db에 temp에 넣어준 것과 동일한 값을 넣어주고 데이터 손실을 방지하기 위해 temp를 free해주었습니다.

```
else{
        while(1){
                if(cur -> next == NULL)
                        break;
                cur = cur ->next;
                }
        cur -> next = temp;
        }
}
```

.

처음 db에 단어가 있었다면, cur를 마지막 위치까지 이동시키고 다음 next주소로 temp를 연결시켜 주었습니다

다음으로 val값이 1이 아닐 때 즉 입력받은 적이 있는 단어라면, 해당 단어 위치가 될 때까지 cur의 위치를 옮겨주고 해당 위치의 count 값에 1을 더해주었습니다.

```
else{
        while(1){
                if(strcmp(cur->word,key)!=0)
                        cur = cur ->next;
                else
                        break;
                }
        cur->count = cur->count+1;
}
```

```
db_close(DB);
printf("DB closed₩n");
return 0;
```

Db_put함수를 나오고 main문으로 다시 가서 만약 파일입력이 끝나 while문이 종료되면 db_close를 통해 사용했던 구조체를 free시켜 메모리 낭비를 방지해줍니다.

```
void db_close(db_t *db)
{
        int i=0;
        db_t* temp = (db_t *)malloc(sizeof(db_t));
        db_t* cur = (db_t *)malloc(sizeof(db_t));
        for(i=0; i<t; i++){
                if(db[i].word == NULL)
                        continue;
```

여기서 t는 전역변수로 size와 같은 값을 가지고 있어 db구조체를 끝까지 확인하기 위해 for문을 사용하였습니다.

만약 i번째 word가 NULL이면, 즉 hash function을 통해 나온 적이 없는 값(위치)이면 메모리를 사용하지 않았기 때문에 continue해주고

```
if(db[i].next != NULL){
        cur = db[i].next;
        while(cur->next != NULL){
                temp = cur ->next;
                free(cur->word);
                free(cur);
                cur = temp;
        }
        temp = cur;
        free(temp->word);
        free(temp);
}
```

NULL이 아니면 해당 db에 단어들이 있는 것임으로, cur에 바로 다음으로 오는 구조체를 넣어주고, while문을 통하여 구조체 마지막 위치 전까지 free 시켜주었습니다. 그 후 마지막 위치는 while밖에서 따로 free해 주었습니다.

for문을 통하여 db 전체에 연결된 list들을 free해준 후 각 db의 첫번째 head부분은 word만 for문으로 free해주고, 전체를 한번에 free시켜 주었습니다.

```
for(i=0; i<t; i++){
        if(db[i].word != NULL)
                free(db[i].word);
        }
free(cur);
free(db);
```