

Hashing

학번 : 201002513

이름 : 최 혁수

☐ 실습 6. Hashing

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

☐ 소감문 & 한계성

☐ All Source Code

실습 6. Hashing

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

Hashing

□ 개요

해쉬(hash)의 기본적인 알고리즘을 이해하고 단일 링크드리스트(Single Linked List)을 이용하여 각 해쉬테이블(hash_table)을 연결하는 기법을 구현한다.

□ 접근방법

- 키(Key)값 산출 : 데이터의 각 문자를 ASCII 숫자를 이용하여 덧셈 후 Division 방법으로 mod 연산을 수행
- Insert function : 입력한 element를 갖는 list_pointer를 hash_table에 연결
- Search function : 해당 해쉬테이블에 연결된 리스트 탐색을 통해 데이터 검출
- Delete function : Search function으로 찾은 리스트의 이전과 다음을 연결

□ Division algorithm

해당 Key 값을 특정한 값 x로 Mod 연산을 수행하여 구한다. ($= \text{mod } x$)

* ex) 숫자의 경우 : $1+2+3+4+5 = 15 \text{ mod } x$, if $x=13$, result=2

Hashing(1/2)

□ head.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Windows.h> // console size & clear 기능 사용

//linked list, chaining, division
#define MAX_CHAR 50
#define TABLE_SIZE 13

typedef struct{
    char Key_Num[MAX_CHAR]; // 제품의 번호
    char description[20]; // 제품의 설명
    int stored_num; // 제품의 재고량
    int reorder_level; // 제품의 재주문 단계
}element;

typedef struct list *list_pointer;
typedef struct list{ // item & link를 가진 struct list 선언
    element item;
    list_pointer link;
}List;

list_pointer hash_table[TABLE_SIZE]; // hash_table[13]

void Init_table(); // Init & allocate hash_table[]
int hash(char *key); // find key. (using mod x)
int transform(char *key); // ASCII값으로 각 문자를 ADD
void Insert(list_pointer pt); // hash_table[key] 끝에 list 삽입
void Show(list_pointer pt); // element 출력
int search(char *key); // key_Num[]에 따른 검색
void del(char *key); // 삭제
```

□ void hash(char *), int transform(char *)

```
int hash(char *key){ // find key. (mod)
    return (transform(key) % TABLE_SIZE);
}

int transform(char *key){ // ASCII 이용하여 각 문자 덧셈
    int number=0;
    while(*key) number += *key++;
    return number;
}
```

Hashing(2/2)

❑ void Insert(list_pointer)

```
void Insert(list_pointer pt){ // hash_table[key] 끝에 list 삽입
    list_pointer first;
    list_pointer temp = (List *)malloc(sizeof(struct list));

    strcpy(temp->item.Key_Num, pt->item.Key_Num);
    strcpy(temp->item.description, pt->item.description);
    temp->item.reorder_level = pt->item.reorder_level;
    temp->item.stored_num = pt->item.stored_num;
    temp->link = NULL;

    first = hash_table[hash(temp->item.Key_Num)];
    while(first->link != NULL){ // NULL 까지 수행
        first = first->link;
        Show(first); // 기존 element 출력
    }
    first->link = temp;
    Show(temp); // 입력된 element 출력
}
```

❑ int search(char *, int

```
int search(char *key){

    list_pointer search_node = (list_pointer)malloc(sizeof(struct list));
    // hash_table[]의 끝까지 검색
    for(search_node=hash_table[hash(key)]; ; search_node = search_node->link){
        if(search_node==NULL){ // empty? break;
            break;
        }
        if(!strcmp(key, search_node->item.Key_Num)){ // key와 동일하면 출력
            Show(search_node);
        }
    }
}
```

❑ void del(char *)

```
void del(char *key){
    list_pointer search_node = (list_pointer)malloc(sizeof(struct list));
    list_pointer temp;
    // hash_table[] 끝까지 검색
    for(search_node=hash_table[hash(key)]; ; search_node = search_node->link){
        if(!strcmp(key, search_node->link->item.Key_Num)){ // 값을 찾으면
            search_node->link = search_node->link->link; // 이전 link와 다음 link를 연결
            break;
        }
    }
}
```

실행결과 분석 & 구현상의 오류 및 한계

□ 실행 결과화면

○ Insert

```
C:\Windows\system32\cmd.exe

***** print *****
Key_Num : 123
description : keyboard
stored_num : 2
reorder_level : 1
***** print *****
Key_Num : 231
description : monitor
stored_num : 3
reorder_level : 2
        select the option
*****
1.input 2.search 3.delete 4.prog exit
*****
function : _
```

○ Search

```
C:\Windows\system32\cmd.exe

Key_Num : 123
***** print *****
Key_Num : 123
description : keyboard
stored_num : 2
reorder_level : 1
        select the option
*****
1.input 2.search 3.delete 4.prog exit
*****
function :
```

○ Delete

```
C:\Windows\system32\cmd.exe

Key_Num : 123
        select the option
*****
1.input 2.search 3.delete 4.prog exit
*****
function :
```

□ 구현상의 오류 및 한계

실습과제(PDF)에서 Key_Num = 12345 일 때, Key값 산출이 $(1+2+3+4+5)\%13 = 2$ 로 된다고 했는데, element에서 선언한 Key_Num은 char형으로 각 index를 받고 있으므로 주어진 transform()에서는 숫자 또한 문자의 형식으로 ASCII값으로 덧셈이 수행된다. 즉 key가 2가 나올수 없다.

이를 해결하기 위해서는 주어진 transform()을 사용자가 입력한 Key_Num값을 숫자 및 문자를 구별하는 조건식을 추가해야 한다.

소감문 & 한계성

소감문 & 한계성

□ 소감문

해쉬(Hash)의 이론적 알고리즘을 일정한 element을 가진 struct를 구현하여 table을 구성하고 이를 single linked list로 연결하는 각 알고리즘의 복합체로 표현하는 방법을 익혔으며, 이를 Division이란 방법으로 Key값을 찾아 Table[]의 index로 쓰일수 있다는 것 또한 알수 있었다.

□ 한계성

Element의 Key_Num[]을 문자, 숫자를 구별해서 key값을 산출하기 위해선 구별할수 있는 조건식을 추가해야 한다.

또한 링크드 리스트를 이용하여 해쉬 테이블을 무한적으로 늘릴수 있지만, hash_table은 13개로 정해져있기 때문에 이를 동적으로 늘리기 위해선 realloc을 이용해야 한다.

MAX_CHAR 50로 문자열의 개수 또한 정해져있으므로 50개 이상이 되는 문자열의 입력시 오류가 발생하는 점을 가지고 있다.

All Source Code

Hashing(1/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Windows.h> // console size & clear 기능 사용

//linked list, chaining, division
#define MAX_CHAR 50
#define TABLE_SIZE 13

typedef struct{
    char Key_Num[MAX_CHAR]; // 제품의 번호
    char description[20]; // 제품의 설명
    int stored_num; // 제품의 재고량
    int reorder_level; // 제품의 재주문 단계
}element;

typedef struct list *list_pointer;
typedef struct list{ // item & link를 가진 struct list 선언
    element item;
    list_pointer link;
}List;

list_pointer hash_table[TABLE_SIZE]; // hash_table[13]

void Init_table(); // Init & allocate hash_table[]
int hash(char *key); // find key. (using mod x)
int transform(char *key); // ASCII값으로 각 문자를 ADD
void Insert(list_pointer pt); // hash_table[key] 끝에 list 삽입
void Show(list_pointer pt); // element 출력
int search(char *key); // key_Num[]에 따른 검색
void del(char *key); // 삭제

void Init_table(){ // allocate & Init the hash_table[]
    int i;
    for(i=0; i<TABLE_SIZE; i++){
        hash_table[i] = (list_pointer)malloc(sizeof(struct list));
        hash_table[i]->link = NULL;
    }
}

int hash(char *key){ // find key. (mod)
    return (transform(key) % TABLE_SIZE);
}

int transform(char *key){ // ASCII 이용하여 각 문자 덧셈
    int number=0;
    while(*key) number += *key++;
    return number;
}
```

Hashing(2/3)

```
void Insert(list_pointer pt){ // hash_table[key] 끝에 list 삽입
    list_pointer first;
    list_pointer temp = (List *)malloc(sizeof(struct list));

    strcpy(temp->item.Key_Num, pt->item.Key_Num);
    strcpy(temp->item.description, pt->item.description);
    temp->item.reorder_level = pt->item.reorder_level;
    temp->item.stored_num = pt->item.stored_num;
    temp->link = NULL;

    first = hash_table[hash(temp->item.Key_Num)];
    while(first->link != NULL){ // NULL 까지 수행
        first = first->link;
        Show(first); // 기존 element 출력
    }
    first->link = temp;
    Show(temp); // 입력된 element 출력
}

int search(char *key){

    list_pointer search_node = (list_pointer)malloc(sizeof(struct list));
    // hash_table[]의 끝까지 검색
    for(search_node=hash_table[hash(key)]; ; search_node = search_node->link){
        if(search_node==NULL){ // empty? break;
            break;
        }
        if(!strcmp(key, search_node->item.Key_Num)){ // key와 동일하면 출력
            Show(search_node);
        }
    }
}

void Show(list_pointer pt){ // print the element
    printf("***** print *****\n");
    printf("Key_Num : %s\n", pt->item.Key_Num);
    printf("description : %s\n", pt->item.description);
    printf("stored_num : %d\n", pt->item.stored_num);
    printf("reorder_level : %d\n", pt->item.reorder_level);
}

void del(char *key){

    list_pointer search_node = (list_pointer)malloc(sizeof(struct list));
    list_pointer temp;
    // hash_table[] 끝까지 검색
    for(search_node=hash_table[hash(key)]; ; search_node = search_node->link){
        if(!strcmp(key, search_node->link->item.Key_Num)){ // 값을 찾으면
            search_node->link = search_node->link->link; // 이전 link와 다음 link를 연결
            break;
        }
    }
}
```

Hashing(3/3)

```
int main(){
    list_pointer temp = (List*)malloc(sizeof(struct list));
    int i=1;
    system("mode con cols=41 lines=30"); // console size 조절
    Init_table(); // Init & allocate
    while(i!=4){
        printf("          select the option          \n");
        printf("*****\n");
        printf("1.input  2.search  3.delete  4.prog exit\n");
        printf("*****\n");
        printf("function : ");
        scanf("%d", &i);
        system("cls"); // console Clear
        switch(i){
            case 1: // Input function
                printf("***** input *****\n");
                printf("Key_Num : ");
                scanf("%s", &temp->item.Key_Num);
                printf("Description : ");
                scanf("%s", &temp->item.description);
                printf("stored_num : ");
                scanf("%d", &temp->item.stored_num);
                printf("reorder_level : ");
                scanf("%d", &temp->item.reorder_level);
                system("cls");
                Insert(temp);
                break;
            case 2: // Search function
                printf("Key_Num : ");
                scanf("%s", &temp->item.Key_Num);
                search(temp->item.Key_Num, i);
                break;
            case 3: // Delete function
                printf("Key_Num : ");
                scanf("%s", &temp->item.Key_Num);
                del(temp->item.Key_Num);
                break;
            case 4: // End of program(while)
                break;
        }
    }
}
```