

# CSED101. Programming & Problem solving

## Fall, 2015

### Programming Assignment #4 (70 points)

김인한(kiminhan@postech.ac.kr)

■ **Due:** 2015.11.28 23:59

■ **Development Environment:** Windows Visual Studio 2010

#### ■ 제출물

- C Code files (**mystring.h, mystring.c, assn4\_1.c, assn4\_2.c**)
  - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 **주석**을 붙일 것
- **보고서 파일** (.doc(x) or .hwp) 예) assn4.doc(x) 또는 assn4.hwp
  - AssnReadMe.pdf 를 참조하여 작성할 것.
  - 보고서는 Problem2, Problem3에 대해서만 작성 할 것.
  - 각각의 기능에 대한 프로그램 실행 화면을 캡처하여 보고서에 포함시키고 간단히 설명할 것.
- 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.

#### ■ 주의사항

- 각 문제에 해당하는 요구사항을 반드시 지킬 것. (각 문제에 주어진 조건을 만족하지 못하면 감점 처리된다.)
- 모든 문제의 출력 형식은 아래의 예시들과 동일해야 하며, 같지 않을 시는 감점이 된다.
- 각 문제에 제시되어 있는 파일이름으로 제출 할 것. 그 외의 다른 이름으로 제출하면 감점 또는 0점 처리된다.
- 컴파일 & 실행이 안되면 무조건 0점 처리된다.
- 하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)
- 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
- 이번 과제에서 추가 기능 구현에 대한 점수가 없습니다.
- **과제 작성시 전역변수는 사용할 수 없으며, 요구되는 기능을 사용자 정의 함수로 적절히 분리하여 코드를 작성하여야 합니다. (main() 함수만 사용하는 경우 감점이 됩니다.)**

## ■ Problem 1: 문자열 처리 함수 (5점)

### (문제)

C 언어에서 제공되는 문자열 함수는 "string.h" 라이브러리에서 제공된다. 문자열 라이브러리에 포함되어 있는 함수 중 일부를 직접 작성해 보자.

### (설명)

제공된 assn4.zip 의 압축을 풀면 **mystring.h**와 **mystring.c**가 주어진다.

mystring.h 는 아래와 같은 문자열 처리 함수의 선언을 포함하고 있다. (그대로 사용하고 변경하지 말 것)

```
#ifndef MY_STRING_H
#define MY_STRING_H

int mystrlen(char *str);
char *mystrcpy(char *toStr, char *fromStr);
int mystrcmp(char *str1, char *str2);
char *mytolower(char *str);

#endif
```

위 함수의 구현부를 포함한 mystring.c 를 작성하라. 각 함수의 정의는 다음과 같다.

#### (1) int mystrlen(char \*str)

NULL 문자를 제외한 문자열의 길이를 반환한다. 빈 문자열의 경우 0 을 반환한다.

예제)

```
printf("%d\n", mystrlen("cs101")); // 결과: 5
```

#### (2) char \*mystrcpy(char \*toStr, char \*fromStr)

문자열 복사 함수로 NULL 문자를 포함한 문자열 fromStr 를 문자열 toStr 에 복사한 후, 문자열 toStr 의 시작 주소를 반환한다.

예제)

```
char str[256];
printf("%s\n", mystrcpy(str, "Good Day")); // 결과: Good Day
printf("%s\n", mystrcpy(str, "Hello"));    // 결과: Hello
```

#### (3) int mystrcmp(char \*str1, char \*str2)

문자열 str1 과 str2 의 대소를 비교한다(대소문자 구분). 비교기준은 아스키코드표의 값을 기준으로 한다.

각 문자열의 첫 번째 문자부터 비교를 시작한다. 만일 같다면 두 문자가 다를 때까지나 NULL 에 도달할 때까지 계속 비교를 수행한다.

- 비교 중 str1 의 문자가 작을 경우 -1, 클 경우 1 을 반환한다.
- 문자열이 길이가 같고 모든 문자가 같을 경우, 0 을 반환한다.
- 비교 중 하나의 문자열이 먼저 끝에 도달할 경우, 끝난 문자열을 작다고 판단한다.

예제)

```
printf("%d\n", mystrcmp("csed101", "csed103")); // 결과: -1
printf("%d\n", mystrcmp("csed", "Csed")); // 결과: 1
printf("%d\n", mystrcmp("csed", "cse")); // 결과: 1
printf("%d\n", mystrcmp("csed", "csed103")); // 결과: -1
```

#### (4) char \*mytolower(char \*str)

문자열 str 을 소문자로 변환한 후, 문자열 str 의 시작주소를 반환한다.

예제)

```
char str[256] = "Hello World! 123";
printf("%s\n", mytolower(str)); // 결과: hello world! 123
```

## ■ Problem 2: 영한 사전 프로그램 (30점)

(목적) 영한 사전 프로그램을 problem1 에서 만든 문자열 처리 함수와 동적 할당의 개념을 이용하여, 구현 하며 각각의 개념을 익힌다.

### (문제)

문제와 함께 제공된 "word.txt" 파일은 "영어단어, 단어의 뜻(한글)"의 조합으로 이루어진 영한 사전이다. 이 파일을 이용한 영한 사전 프로그램을 작성하자.

### (설명 및 요구사항)

- 파일이름은 확장자를 포함하여 "assn4\_1.c"로 한다.
- Problem 1 에서 작성한 mystring.h 를 include 하여 사용한다. (표준 헤더 파일 <string.h>를 include 하여 사용할 수 없다.)
- 모든 입력 받은 문자열(ex: 명령어, 단어, 파일명 등)에는 공백이 없고, 100byte 를 넘지 않는다고 가정한다.
- 이 프로그램은 사용자로부터 5 개의 명령어(single, multi, recent, quit, help)를 입력 받아 각 기능을 수행하게 된다. 이 때, 각 명령어를 수행하는 별도의 함수를 구현하여 사용해야 한다.
- 명령어 recent 는 최근 검색 단어를 최대 5 개까지, 구조체 배열(recentTable)을 이용하여 저장하여 관리하도록 한다. 이 때 각 단어의 정보는 아래의 구조체 형식을 선언하여 사용한다.

```
typedef struct{
    int num;           // 사전 파일 "word.txt"에서 단어의 위치(몇 번째 단어)
    char *word;        // 영어 단어
    char *meaning;     // 단어의 뜻(한글)
}WORDINFO;
WORDINFO recentTable[5]; // main 함수에 선언하여 관리할 것
```

- 구조체(WORDINFO)의 단어(word)와 뜻(meaning) 정보는 동적으로 할당 된 문자열을 가리키도록 한다. 아래와 같이 필요한 길이만큼 메모리를 동적 할당을 받아 사용하여 메모리에 낭비가 없도록 한다.
- 아래는 입력 받은 문자열을 메모리 낭비 없이 동적으로 할당하여 사용한 예 이다.  
문자열을 입력 받을 때, 입력 받는 길이는 미리 알 수 없기 때문에 충분한 크기의 배열(str)을 선언하여 그곳에 입력 받고, 입력 받은 문자열의 길이만큼만 메모리를 동적으로 할당하여 사용 하여야 한다 (참고로, 한글 문자 1개는 2byte의 저장공간이 필요하다.).
- 사용이 끝난 동적 할당된 메모리는 free로 할당 해제 한다.

```
char str[100];           // 문자열을 입력 받을 배열
char *strP;             // 동적 메모리 할당 받을 포인터
scanf("%s", str );      // "abcde"를 입력 받은 경우, "abcde\0"을 저장하는 공간을 제외한 나머지
                        // 94칸의 메모리가 낭비.
strP = (char *)malloc(sizeof(char)*(strlen(str)+1)); // str에 저장된 문자열의 크기와 동일한 공간 할당
strcpy(strP, str);      // strP가 가리키는 공간에 문자열 복사 기능 수행
...
free(strP);            // strP 을 모두 사용 후, 메모리 할당 해제
// 이 과제에서는 strcpy, strlen 함수 대신 구현한 mystrcpy, mystrlen 함수를 사용할 것
```

## (1) 초기 화면

- 프로그램을 실행시키면 아래와 같이 사용 가능한 명령어 목록이 출력된 후, 그 아래에 명령어 입력을 대기하는 ">>" 표시가 뜨며, 이 상태에서 사용자는 명령어를 입력할 수 있다.

```

C:\Windows\system32\cmd.exe

-----
single: 한 단어 검색
multi : 여러 단어 검색
recent: 최근 검색 단어
quit  : 프로그램 종료
help  : 명령어 목록 보기
-----
>>
  
```

- single, multi, recent, quit, help** 는 각 메뉴의 명령어이고, 명령어를 입력 하였을 때만 기능이 실행된다. 이 명령어는 사용자가 명령어 입력 시, 대소문자를 구분하지 않고 동일한 명령어의 기능을 수행하도록 작성한다. 예를 들면, help, HELP, Help, helP 는 동일한 동작을 수행한다.
- 메뉴 출력에 사용되는 구분 선은 아래의 printf 문을 사용한다.

```
printf("-----Wn");
```

- 위의 5 가지 명령어 이외의 잘못된 명령어 입력 시, 예러 메시지 없이 아래와 같이 다시 명령어를 입력 받을 준비를 한다. (노란색으로 표시된 문자는 사용자 입력에 해당한다.)

```

C:\Windows\system32\cmd.exe

>> find
>>
  
```

## (2) 한 단어 검색: single

- single 입력 시, 아래의 화면과 같이 검색할 단어를 입력 받도록 한다.

```

C:\Windows\system32\cmd.exe

>> single
검색할 단어:
  
```

- 검색할 단어를 입력하면, 제공된 사전("word.txt" 파일)에서 검색한다. 파일의 내용 및 구성은 아래와 같다. (아래는 "word.txt" 파일의 일부이다.)

wild	야생의
wet	젖은
blind	눈먼
dumb	병어리의
sharp	날카로운

- 영어 단어와 뜻(한글)은 Pair 로 구성되어 있으며 영어 단어와 뜻(한글) 사이는 **탭**으로([영어단어][wt][단어의뜻]), 단어와 단어 사이는 **줄 바꿈 문자**로 구분된다.
- 입력한 단어가 사전에 있으면, 아래의 화면과 같이 검색 성공 문구와 함께 "사전에 위치한 순서.", "단어", "뜻" 순으로 출력한다.

```

C:\Windows\system32\cmd.exe

>> single
검색할 단어: delight
[검색 성공] 672.delight : 기쁨
>>
  
```

- 입력한 단어가 사전에 없는 경우에는 아래와 같은 결과를 출력 하도록 한다.

```

C:\Windows\system32\cmd.exe
>> single
검색할 단어 : superman
[검색 실패] superman 사전에 없는 단어
>>

```

### (3) 여러 단어 검색: multi

- 여러 단어 검색 기능은 txt 파일을 이용하여 사전 내의 여러 단어를 한 번에 검색하는 기능이다. 이 기능에 사용될 txt 파일의 예시는 "word\_multi.txt" 파일로 제공되며 파일의 내용 및 구성은 아래와 같다. (아래는 "word\_multi.txt" 파일의 일부이다.)

```

creature
weapon
apple
wonderful
show
tongue
ostrich
chain

```

- 검색하고자 하는 단어가 나열되어 있으며, 단어는 줄 바꿈 문자로 구분된다.
- multi 입력 시, 파일명을 입력 받는 곳이 나오고, 파일명을 입력 하였을 경우 파일 내의 모든 단어에 대해서 번호, 단어, 뜻을 출력한다. (파일명에는 공백이 없으며 그 길이는 100byte 를 넘지 않는다.)

```

C:\Windows\system32\cmd.exe
>> multi
파일명 입력 : word_multi.txt
[검색 성공] 123.creature : 동물
[검색 성공] 146.weapon : 무기
[검색 실패] apple 사전에 없는 단어
[검색 성공] 49.wonderful : 놀랄만한
[검색 실패] show 사전에 없는 단어
[검색 성공] 833.tongue : 혀
[검색 성공] 863.ostrich : 타조
[검색 성공] 825.chain : 사슬

```

... 중략 ...

```

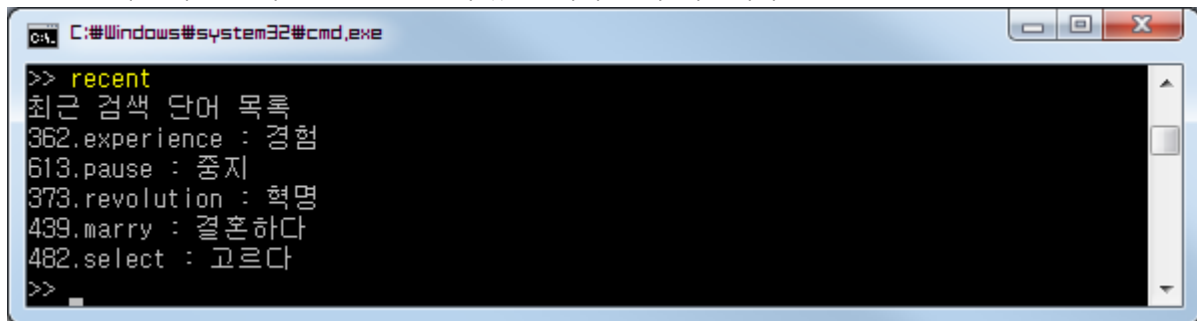
C:\Windows\system32\cmd.exe
[검색 성공] 482.select : 고르다
[검색 성공] 439.marry : 결혼하다
[검색 성공] 373.revolution : 혁명
[검색 성공] 613.pause : 중지
[검색 성공] 362.experience : 경험
>>

```

위의 두 화면은 입력 받은 word\_multi.txt 파일에 대한 출력예시의 일부이다. 실제 프로그램은 파일 내의 모든 단어의 검색 결과를 출력한다.

#### (4) 최근 검색 단어: recent

- recent 입력 시, 가장 최근에 검색 된(실패한 단어 제외) 최대 5 개까지의 단어를 출력해 준다. 단, 위쪽으로 갈수록 최신 단어이다. 아래의 예시는 제공한 word\_multi.txt 파일을 가지고 multi 명령어를 수행한 후, recent 를 입력 했을 때의 결과 화면이다.

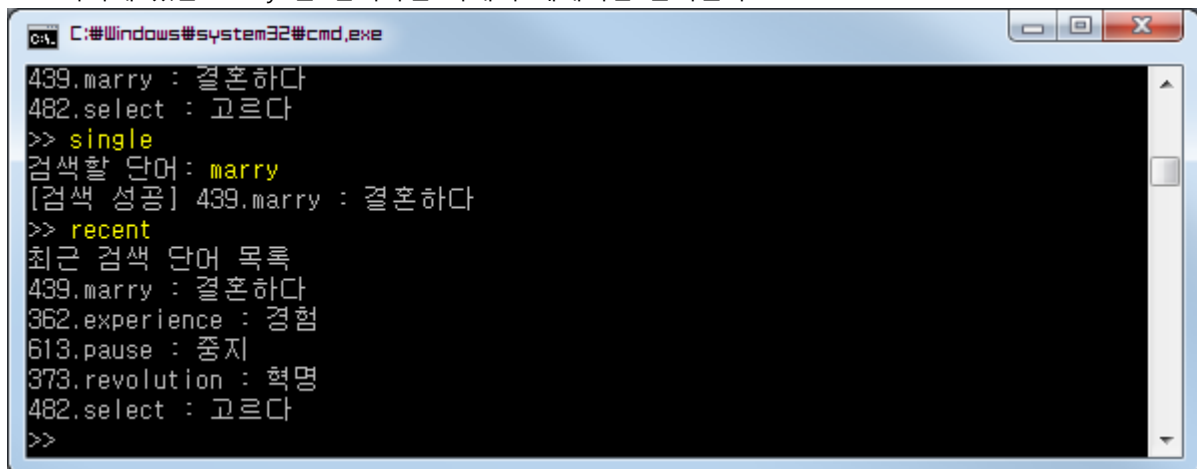


```
C:\Windows\system32\cmd.exe
>> recent
최근 검색 단어 목록
362.experience : 경험
613.pause : 중지
373.revolution : 혁명
439.marry : 결혼하다
482.select : 고르다
>>
```

- 이 기능을 구현 시, 크기 5 의 구조체 배열을 이용하여 최근에 검색된 단어 정보를 저장하여 관리하도록 한다.

`WORDINFO recentTable[5]; // main 함수에 선언하여 관리할 것`

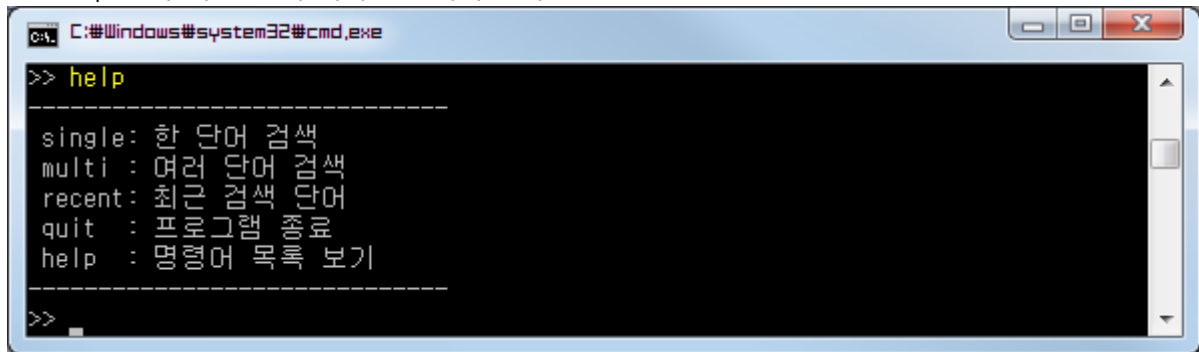
- 구조체 배열 내의 단어와 뜻을 저장하는 문자열 포인터는 사용 될 때 마다 동적으로 저장될 문자열의 크기만큼 할당 받아 사용한다. 예를 들어, 기존에 할당 받은 문자열 포인터에 새로운 문자열이 저장 될 경우에도, 할당 받았던 메모리는 해제 한 후, 새로운 문자열을 위한 메모리 공간을 새로 할당 받아 사용한다.
- 검색 단어 목록 내에 있는 단어가 다시 검색되면, 그 검색단어는 가장 최신 검색단어로 기존 목록의 위치에서 제외 된 후, 최상단에 위치하도록 구현한다. 위 예제의 상황에서 최근 검색 단어 목록에 있는 marry 를 검색하면 아래의 예제처럼 출력된다.



```
C:\Windows\system32\cmd.exe
439.marry : 결혼하다
482.select : 고르다
>> single
검색할 단어: marry
[검색 성공] 439.marry : 결혼하다
>> recent
최근 검색 단어 목록
439.marry : 결혼하다
362.experience : 경험
613.pause : 중지
373.revolution : 혁명
482.select : 고르다
>>
```

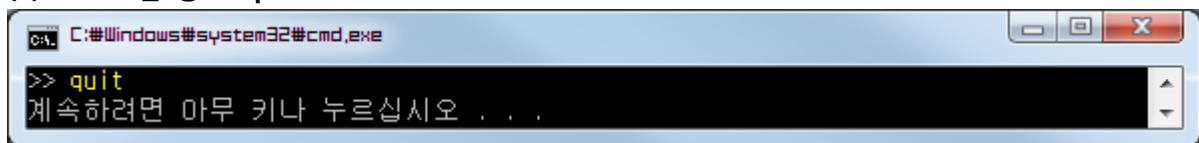
#### (5) 명령어 목록 보기: help

- help 입력 시, 명령어 목록을 출력해 준다.



```
C:\Windows\system32\cmd.exe
>> help
-----
single: 한 단어 검색
multi : 여러 단어 검색
recent: 최근 검색 단어
quit  : 프로그램 종료
help  : 명령어 목록 보기
-----
>>
```

#### (6) 프로그램 종료: quit



```
C:\Windows\system32\cmd.exe
>> quit
계속하려면 아무 키나 누르십시오 . . .
```

- quit 명령어 입력 시, 최근 검색 단어를 저장하는 구조체 배열(recentTable)에 동적으로 할당 받은 공간을 할당 해제 하고 프로그램을 종료한다. 즉, 프로그램을 종료하기 전, 동적할당을 받아 사용했던 모든 메모리를 반드시 할당해제 하도록 한다.

#### (평가조건)

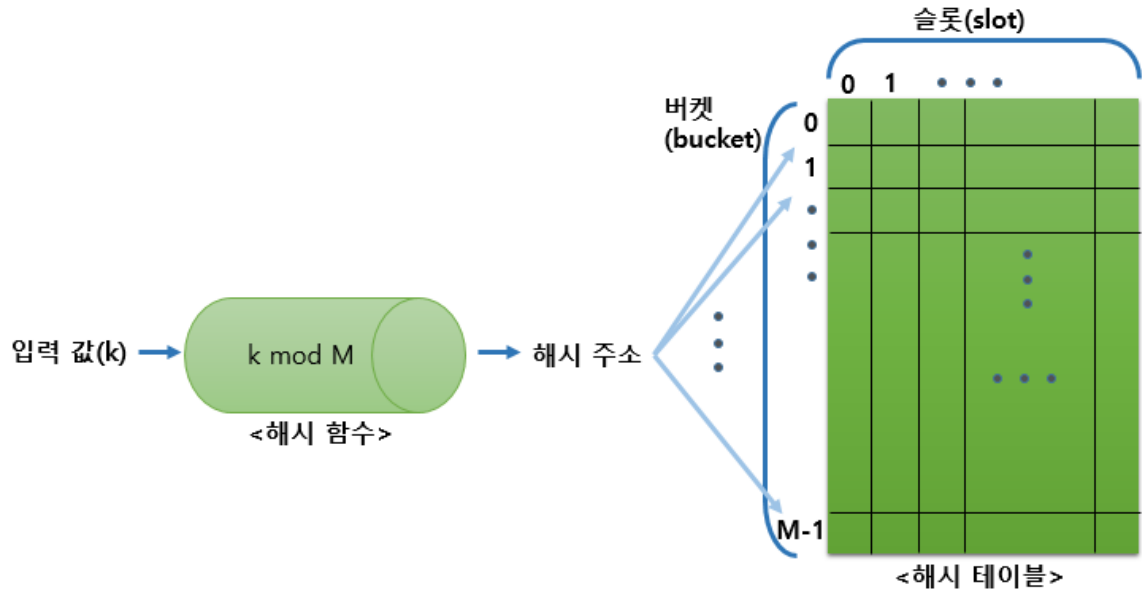
- 문제의 요구사항을 잘 지켰는가?
- 설명된 기능을 이해하고 정확히 구현하였는가?
- 메모리 동적 할당 및 해제를 구현하였는가?
- 명령어를 수행하는 함수를 명령어 단위로 적절히 구현한 후 사용했는가?

### ■ Problem 3: 해싱(Hashing)을 이용한 영한 사전 프로그램 (35점)

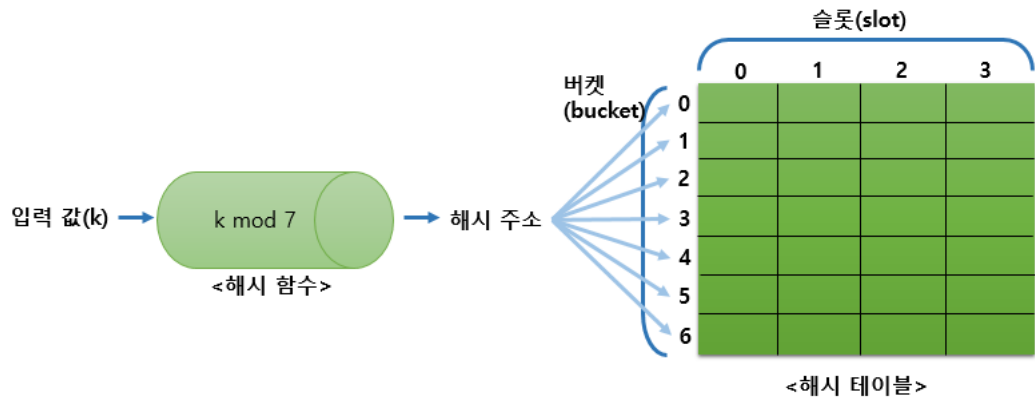
(목적) 다양한 분야에서 많이 사용되는 자료구조 중 하나인 해싱(Hashing)의 개념을 이용하여 영어 단어 검색 기능을 수행해 본다.

#### (개념 설명)

- 해싱이란 많은 데이터를 효율적으로 처리하는 자료구조 중 하나로써, 크게 **해시 키(Hash key)**, **해시 함수(Hash function)**, **해시 주소(Hash Address)**, **해시 테이블(Hash Table)**로 나뉜다.



- 입력 값을 작은 정수로 **사상(mapping)**시키는 함수를 수학적 함수를 **해시 함수**라 부르고, 해시 함수의 연산 결과로 나오는 값을 **해시 주소**라 부른다.
- 해시 테이블은 **버킷(bucket)**과 **슬롯(slot)**으로 구성되는데, 이는 **배열의 행과 열**을 의미 한다. 즉, 2차원 배열과 동일한 구조를 가진다.
- 해시 함수를 통하여 나온 값을 **해시 주소**라 부르고, 이 주소는 **버킷의 인덱스**를 나타낸다.  
**예)** 해시 테이블에 데이터를 저장 할 경우, 해시 함수를 통해 연산된 해시 주소가 3 이라고 한다면, 배열의 행 인덱스가 3인 곳에 저장 된다.
- 두 개 이상의 입력 값이 **같은 해시 주소를 가질 경우 이를 충돌(collision)**이라 부른다. 해시 테이블에 데이터를 저장 할 때, 충돌이 발생 하면 같은 버킷의 다른 슬롯에 데이터를 저장한다.
- 충돌이 발생 할 때마다 같은 버킷 내의 비어있는 슬롯에 데이터를 저장 하는데, 더 이상 저장할 슬롯이 버킷 내에 존재하지 않으면 **오버플로우(overflow)**라 한다.
- 예를 들어, 위의 그림에서처럼  $k \bmod M$  을 해시 함수로 사용해 보자. 이 때, 입력 값  $k$ 는 **양의 정수**라고 가정하고, 해시 함수의  $M$ 은 **7**로 지정한다. 즉, 해시 함수를 통하여 나오는 결과값인 **해시 주소**는 **0~6**이 된다. **슬롯의 개수**는 **4개**로 가정하자. 즉, 아래의 그림과 같은 모양의 전체적인 해싱 구조가 나타난다.

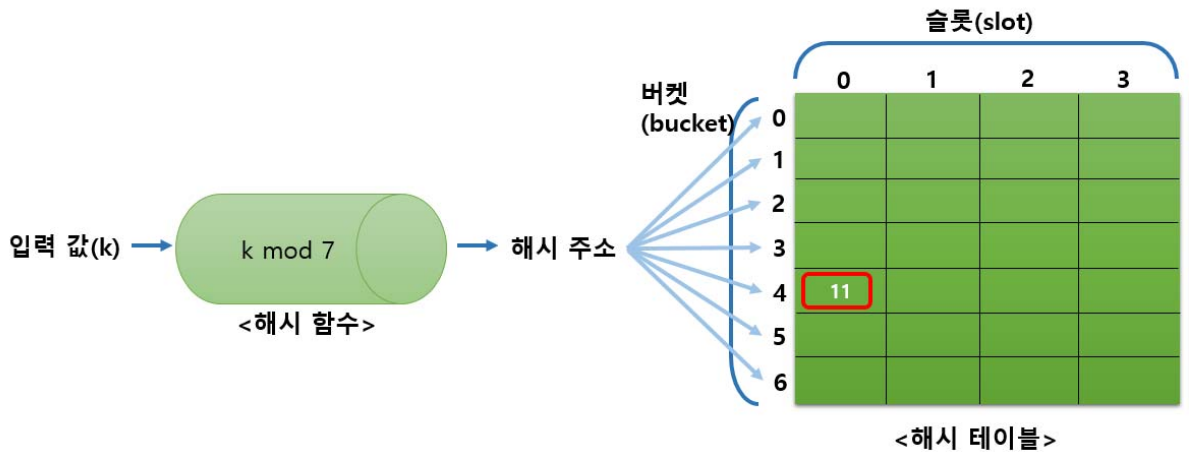


A. 해시 테이블에 저장 하고자 하는 입력 값 k가 11이라 하면,

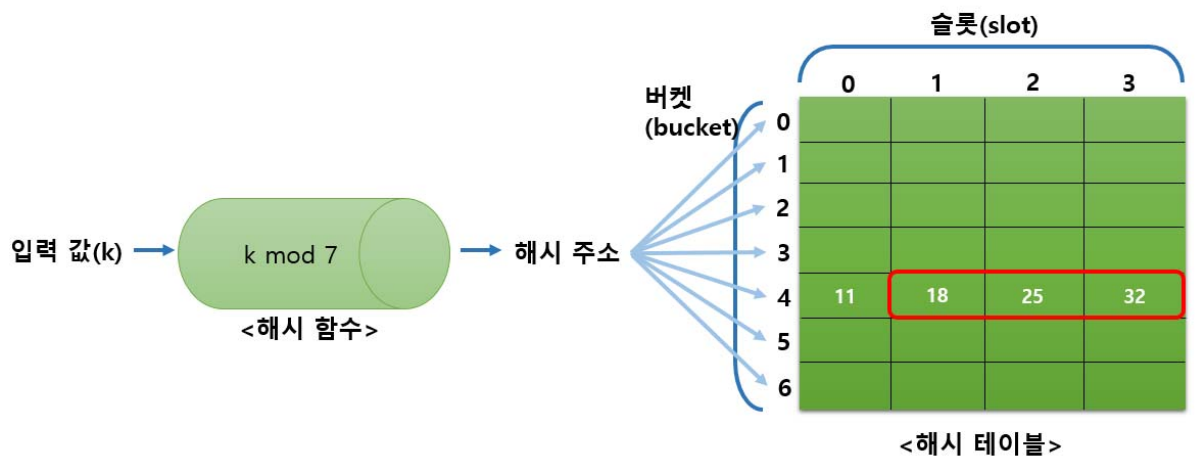
$$11 \bmod 7$$

연산이 수행되고, **해시 주소는 4**가 된다.

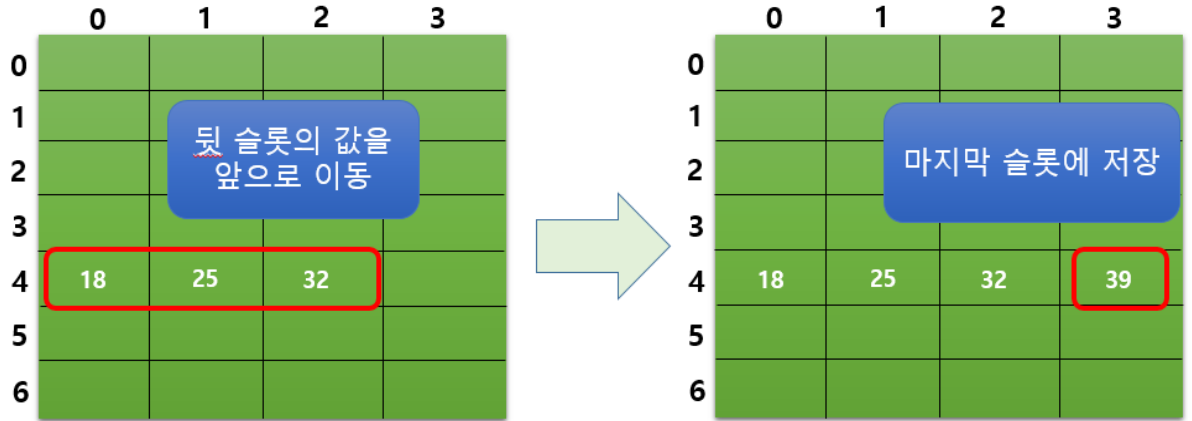
결과적으로, 11은 버킷 4의 0번째 슬롯에 저장된다.



B. 이 후, 18, 25, 32를 저장 하려 한다면, 각각의 해시 주소는 해시 함수에 의해 4가 되고, 0번 버킷에 이미 동일한 해시 주소를 가지는 값들이 존재하기 때문에, 충돌이 발생하여 같은 버킷의 다음 빈 슬롯에 입력 값이 입력 순서대로 저장 된다.



- C. 해시 테이블이 위와 같은 상황에서 39 가 해시 테이블에 저장 된다고 하자. 39의 해시 주소 또한 4 이고, 더 이상 **저장할 슬롯이 버킷4에 없기 때문에, 오버플로우가 발생한다.**
- D. 오버플로우가 발생 하였을 때, 버킷 4에 가장 오래 저장 되어 있었던 11을 삭제하고, 뒷 슬롯의 값을 앞으로 한 칸씩 이동 후, 제일 마지막 슬롯에 39를 저장한다.



#### (문제 설명)

1. Problem2에서는 사전(word.txt)에서만 원하는 단어를 찾았다.
2. 만약, 사람들이 fat이라는 단어를 자주 검색 한다고 하자. 이럴 경우 problem2에서는 fat을 찾기 위해 사전을 처음부터 탐색 하여야 한다. 이는 비효율적이며, 자주 검색되는 단어를 빨리 찾을 수 있게 하여 사전 프로그램의 속도를 향상 시키기 위해 해싱의 개념이 사용될 수 있다.
3. Problem3에서는 사전을 검색 하기 전에, 해시 테이블을 먼저 검색하게 된다. 만약 해시 테이블에 검색하려는 단어가 있다면, 사전을 모두 탐색할 필요가 없으며 빠른 시간 내에 단어와 뜻을 찾을 수 있다.
4. 해시 테이블에 검색한 단어가 없을 경우에는 사전을 탐색하여 단어의 번호, 단어, 뜻을 출력하며, 검색된 단어는 해시 테이블에 저장된다.
5. Problem3의 순서는 아래와 같다.
  - A. 검색할 단어 입력.
  - B. 해시 테이블에 단어가 존재하는지 확인.
  - C. 있다면, 해시 테이블에 저장된 단어 출력.
  - D. 없다면, 사전의 내용을 검색하여 출력하고, 해시 테이블에 검색된 단어 저장.
6. 이러한 순서로 프로그램이 만들어져 있다면, 자주 검색 되는 단어는 해시 테이블에 존재 할 것이기 때문에 빠른 검색 결과를 얻을 수 있다.
7. Problem3에서는 해싱의 입력 값(k)으로 검색을 위해 입력되는 영어 단어의 아스키코드 값의 합을 사용하며, 해시 테이블의 크기는 위의 설명과 같다고 한다. 또한, 해시 테이블에는 단어의 번호, 단어, 뜻이 저장 된다.

예를 들어, fat이 입력 되었다면  $k = (102 + 97 + 116)$  이 되며, 해시 주소 값은

$$315 \bmod 7$$

을 연산하여 0이 된다.

fat은 제공된 사전 파일의 10 번째에 있고, 해시 테이블이 비어 있었다고 가정하자

그러면, 해시 테이블의 버킷 0의 0번째 슬롯에는 "10, fat, 살찐" 이 저장된다.

8. 해시 테이블은 problem2에서 사용된 구조체를 사용한 7행, 4열의 구조체 포인터 배열로 선언되며, 새로운 값을 저장 할 경우 번호, 단어, 뜻을 저장할 수 있는 구조체를 동적으로 할당하여 해시 테이블의 포인터가 가리키도록 한다.

#### (요구사항)

- 파일이름은 확장자를 포함하여 "asn4\_2.c"로 한다.
- Problem 1 에서 작성한 mystring.h 를 include 하여 사용한다. (표준 헤더 파일 <string.h>를 include 하여 사용할 수 없다.
- 메인 함수에 해시 테이블에 사용 될 구조체 포인터 배열(7행 4열)을 선언하고, 다른 함수에서 필요할 경우 파라미터로 넘겨서 사용 할 것. (Problem 2에서 사용한 구조체 WORDINFO 사용할 것)

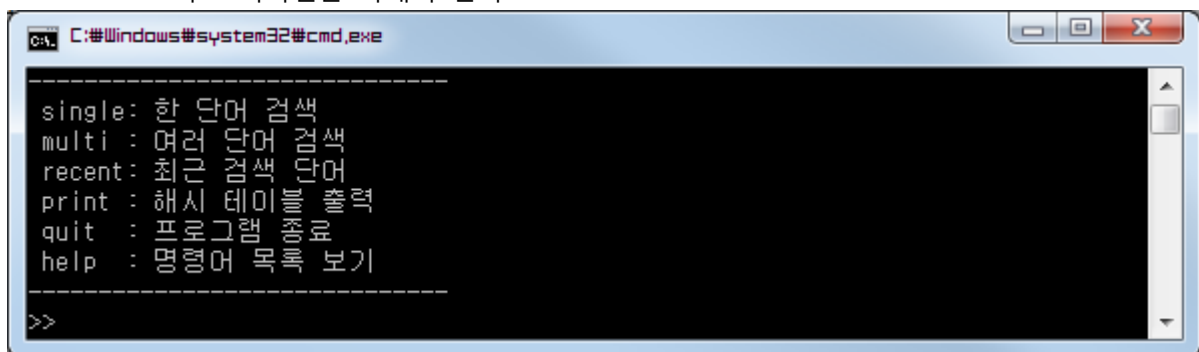
예시) 메인 함수에 선언된 해시 테이블

```
#define MOD_VAL 7
#define MAX_COL 4
...
WORDINFO *hashTable[MOD_VAL][MAX_COL]; // main 함수에 선언하여 관리할 것
```

- 해시 테이블에 새로운 단어 정보를 추가 할 경우에는 메모리를 동적 할당 받은 후, 포인터로 주소를 가리키도록 해서 사용 할 것.
- 메모리를 동적 할당 받은 후, 모든 사용이 끝나면 (예: 오버플로우로 인한 삭제) free를 이용하여 메모리를 반환할 것.
- 사용자로부터 6 개의 명령어(**single, multi, recent, print, quit, help**)를 입력 받아 각 기능을 수행하게 된다. 이 때, 각 명령어를 수행하는 별도의 함수를 구현하여 사용해야 한다.

#### (설명 및 실행 예제)

- Problem3의 초기화면은 아래와 같다.



- Problem2처럼 명령어 목록은 초기 실행 시, 한 번만 출력 되고, 명령어 help를 입력 하였을 때만, 다시 출력된다.
- single 입력 시, 먼저 해시 테이블에서 찾고자 하는 단어가 있는지 검색 한 후, 없으면 사전 파일에서 검색한다.  
프로그램을 시작하고 처음 검색 시, 해시 테이블에는 정보가 없기 때문에, 사전 파일에서

단어를 검색 하여야 하며, 아래와 같이 검색 성공 문구와 함께 사전에 저장된 순서에 따른 번호, 단어, 뜻을 출력한다.

그리고, 해시 테이블에 검색된 결과를 저장하고, 결과를 출력한다.

```
C:\Windows\system32\cmd.exe
>> single
검색할 단어: sharp
[검색 성공(파일)] 5.sharp : 날카로운
[저장 완료(해시 버킷 3)] 5.sharp : 날카로운
>>
```

- sharp의 해시 주소는 3이 되고, 버킷 3에 저장 되었다는 문구와 함께 위의 예시처럼 결과를 출력 한다.
- **print** 입력 시, 해시 테이블의 내용을 출력 한다. 현재는 sharp의 정보만 있으므로, 아래와 같이 출력된다.

```
C:\Windows\system32\cmd.exe
>> print
버킷 0:
버킷 1:
버킷 2:
버킷 3: 5.sharp : 날카로운
버킷 4:
버킷 5:
버킷 6:
>>
```

- 다시 한 번 sharp를 검색하면, 해시 테이블에 sharp에 대한 정보가 있기 때문에, 사전 파일을 읽을 필요 없이, 해시 테이블에 있는 정보를 출력해 준다.

```
C:\Windows\system32\cmd.exe
>> single
검색할 단어: sharp
[검색 성공(해시)] 5.sharp : 날카로운
>>
```

- 오버플로우가 발생하는 예를 설명하기 위하여, 해시 주소가 0인 wet, fat, thirsty, famous, strange를 명령어 single을 이용하여 검색한다.

먼저, wet, fat, thirsty, famous를 순서대로 검색한 경우의 해시 테이블을 출력한 결과이다.

```
C:\Windows\system32\cmd.exe
>> print
버킷 0: 2.wet : 젖은 10.fat : 살찐 11.thirsty : 목마른 14.famous : 유명한
버킷 1:
버킷 2:
버킷 3: 5.sharp : 날카로운
버킷 4:
버킷 5:
버킷 6:
>>
```

- 이 때, 해시 주소가 0인 strange를 검색하면, 버킷 0에 있는 wet이 삭제 되고 가장 마지막 슬롯에 strange가 저장 되어야 하며, 아래와 같이 출력된다.

```
C:\Windows\system32\cmd.exe
>> single
검색할 단어: strange
[검색 성공(파일)] 21.strange : 이상한
[저장 실패(해시)] overflow
[삭제 완료(해시)] 2.wet : 젖은
[저장 완료(해시 버킷 0)] 21.strange : 이상한
>>
```

이 때, 해시 테이블의 정보를 print 명령어로 출력하면, 아래와 같다.

```
C:\Windows\system32\cmd.exe
>> print
버킷 0: 10.fat : 살찐 11.thirsty : 목마른 14.famous : 유명한 21.strange : 이상한
버킷 1:
버킷 2:
버킷 3: 5.sharp : 날카로운
버킷 4:
버킷 5:
버킷 6:
>>
```

- multi 명령어**는 problem2와 같은 역할을 수행하나, 출력되는 결과는 single 명령어를 사용했을 때 출력되었던 해시와 관련된 결과도 함께 출력하도록 한다.
- recent 명령어**는 problem2와 같은 기능을 수행하도록 한다.
- 사전에 없는 단어를 single 혹은 multi 명령어를 이용해 검색** 하였을 경우에도 problem2와 같이 출력한다.

```
C:\Windows\system32\cmd.exe
>> single
검색할 단어: superman
[검색 실패] superman 사전에 없는 단어
>>
```

- quit 명령어** 입력시, 해시테이블(hashTable)과 최근 검색 단어를 저장하는 구조체 배열(recentTable)에 동적으로 할당 받은 공간을 할당 해제 하고 프로그램을 종료한다.

```
C:\Windows\system32\cmd.exe
>> quit
계속하려면 아무 키나 누르십시오 . . .
```

#### (평가조건)

- 해시의 기능을 정확히 이해하고 구현 하였는가?
- 명령어를 수행하는 함수를 명령어 단위로 적절히 구현하여 사용하였는가?
- 문제의 요구사항을 잘 지켰는가?
- 설명된 기능을 이해하고 정확히 구현하였는가?
- 메모리 동적 할당 및 해제를 구현하였는가?