1.

```c
#include <stdio.h>
#include <stdlib.h>

int HASH_idx = 0;
int HASH_Arr[6] = {7000, 6997, 12000, 11117, 22000, 22307};
int HASH_PRIME;

void HASH_init()
{
    HASH_PRIME = HASH_Arr[HASH_idx];
    HASH_idx++;
}

int hash(char key[5])
{
    int i;
    long long x;
    x=0;
    for (i=0; i<4; i++) {
        x=x+key[i];
        x=x<<8;
    }
    x=x+key[4];
    return x%HASH_PRIME;
}
int main() {

    int n = 1;
    char words[5757][5];
    char data[100];

    FILE *f;

    f = fopen( filename: "words.dat",  mode: "r");
    for (int i = 0; i < 5762; i++) {
        fgets(data, sizeof(data), f);
        if(data[0] == '*')
        {
            continue;
        }
        for (int j = 0; j < 5; j++) {
            words[i - 4][j] = data[j];
        }
    }
    fclose(f);
    while(n<7)
    {
        HASH_init();
        int *num = calloc(HASH_PRIME, sizeof(int));
        char arr[5];
        int sum = 0;

        for(int i = 0; i < 5757; i++)
        {
            for(int j = 0; j < 5; j++)
            {
                arr[j] = words[i][j];
            }
            int idx = hash(arr);
            num[idx]++;
        }

        for (int k = 0; k < HASH_PRIME; k++) {
            if (num[k] > 1) {
                sum += num[k] - 1;
            }
        }
        printf("%d이 HASH_PRIME 일 때 M%d의 충돌횟수 : %d\n",HASH_PRIME, n, sum);
        n++;
        free(num);
    }
    return 0;
}
```

1번 문제는 주어진 수를 HASH_PRIME을 사용하여 각 시행의 충돌 횟수를 구하는 문제이다. 6개의 HASH_PRIME을 실행하기 위해 전역변수로 HASH_Arr의 배열을 만들고 Main 함수에서 시행이 반복될때 마다 HASH_PRIME의 값을 초기화 시켜주기 위해 HASH_init 함수를 만들었다. main 함수에서 words.dat 파일 각 행의 5개의 문자를 받기위해 words 2차원 배열을 만들어 초기화 시켜주었다. 그 다음 calloc을

이용하여 HASH_PRIME * 4byte의 공간을 동적 배열 num을 만들어 값을 0으로 초기화시키고 words에 저장된 데이터를 행마다 hash 값을 계산하여 나온 결괏값을 index로 하여 해당하는 num[idx]에 1씩 더했다. 이 작업을 수행하면 num 배열의 저장된 값은 0이면 한번이라도 해당하는 hash값이 나오지 않은것 1이면 충돌이 발생하지 않은 값 2이상이면 충돌이 발생한 값인 것을 알 수 있다. 이를 이용하여 K(index)=0 부터 HASH_PRIME의 값 이전까지 num[K]가 2이상일 경우에는 뺀 num[K]값을 모두 더하면 총 충돌횟수를 알 수 있다.

```
7000이 HASH_PRIME 일 때 M1의 충돌횟수 : 2305
6997이 HASH_PRIME 일 때 M2의 충돌횟수 : 1821
12000이 HASH_PRIME 일 때 M3의 충돌횟수 : 2733
11117이 HASH_PRIME 일 때 M4의 충돌횟수 : 1235
22000이 HASH_PRIME 일 때 M5의 충돌횟수 : 1359
22307이 HASH_PRIME 일 때 M6의 충돌횟수 : 655
```

결과를 통해 알 수 있는 점은 HASH_PRIME의 값이 소수(6997, 11117, 22307)가 합성수(7000, 12000, 22000)보다 충돌횟수가 적은 것을 볼 수 있다. 이유를 생각해 보면 예를 들어 10, 20, 30, 40, 50 배열이 있을때 합성수 4를 hash_Prime으로 잡고 mod 연산을 하면 10⇒2, 20⇒0, 30⇒2 40⇒0, 50⇒2로 2가 3번, 0은 2번 나와 충돌이 3번 발생하지만 만약 3이라면 10⇒1, 20⇒2, 30⇒0, 40⇒1, 50⇒2로 1의 값으로 딱 한번의 충돌이 발생하는 것을 보아 소수가 충돌을 최소화 하기에 좋다. 단, 만약 소수 5를 잡을 경우 배열의 모든 수가 5의 배수로 10⇒0, 20⇒0 30⇒0, 40⇒0, 50⇒0으로 4번의 충돌이 발생하므로 어떠한 소수를 선택할 지도 잘 고려할 부분인 것 같다.

2.

```c
#include <stdio.h>
#include <stdlib.h>

#define HASH_PRIME 11117

int hash(char key[5])
{
    int i;
    long long x;
    x=0;
    for (i=0; i<4; i++) {
        x=x+key[i];
        x=x<<8;
    }
    x=x+key[4];
    return x%HASH_PRIME;
}

int main()
{
    char words[5757][5];
    char data[100];

    FILE *f;
    f = fopen( filename: "words.dat",  mode: "r");
    for (int i = 0; i < 5762; i++) {
        fgets(data, sizeof(data), f);
        if(data[0] == '*')
        {
            continue;
        }
        for (int j = 0; j < 5; j++) {
            words[i - 4][j] = data[j];
        }
    }
    fclose(f);

    int *num = calloc(HASH_PRIME, sizeof(int));
    char arr[5];
    int sum = 0;
    int cmp = 0;

    for(int i = 0; i < 5757; i++)
    {
        for(int j = 0; j < 5; j++)
        {
            arr[j] = words[i][j];
        }
        int idx = hash(arr);
        num[idx]++;
        printf("%s의 해쉬값 : %d 비교횟수 : %d\n",arr, idx, num[idx]);

        cmp += num[idx];

    }

    for (int k = 0; k < HASH_PRIME; k++) {
        if(num[k] != 0)
        {
            printf("해쉬테이블[%d]의 chain길이 : %d\n", k, num[k]);
        }
        if (num[k] > 1) {
            sum += num[k] - 1;
        }
    }
    printf("총 충돌 횟수 : %d\n", sum); // 1235
    printf("words.dat파일 데이터들의 총 비교 횟수 : %d\n", cmp);
    free(num);
    return 0;
}
```

2번은 11117을 HASH_PRIME으로 잡고 chaining을 이용하여 해쉬테이블의 각 원소별 체인의 길이를 통해 words.dat 파일 속 데이터를 모두 한번씩 찾을때 총 몇번의 비교가 발생하는지 구하는 문제다. 코드의 대부분은 1번코드와 비슷하지만 좀더 자세히 알아보기 위해 각 데이터별 해쉬값과 현재 hash table 안에 서 라선이 들어갔을 경우의 비교횟수를 구해보았다.

```
aargh의 해쉬값 : 6352 비교횟수 : 1
abaca의 해쉬값 : 4572 비교횟수 : 1
abaci의 해쉬값 : 4580 비교횟수 : 1
aback의 해쉬값 : 4582 비교횟수 : 1
abaft의 해쉬값 : 5359 비교횟수 : 1
abase의 해쉬값 : 8672 비교횟수 : 1
abash의 해쉬값 : 8675 비교횟수 : 1
abate의 해쉬값 : 8928 비교횟수 : 1
abbey의 해쉬값 : 3942 비교횟수 : 1
abbot의 해쉬값 : 6497 비교횟수 : 1
abeam의 해쉬값 : 10525 비교횟수 : 1
abend의 해쉬값 : 2727 비교횟수 : 1
abets의 해쉬값 : 4278 비교횟수 : 1
abhor의 해쉬값 : 10616 비교횟수 : 1
abide의 해쉬값 : 6621 비교횟수 : 1
```

처음에는 대부분 hash table 값이 비어 있기 때문에 1만 출력되는 것을 볼 수 있고 이후 결과를 살펴보면 halts 문자열이 처음 들어갔을 때 비교횟수는 1이라면

```
halts의 해쉬값 : 6310 비교횟수 : 1
```

zowie가 들어가면 동일한 hash값으로 비교횟수가 그로 증가하는 것을 볼 수 있다.

```
zowie의 해쉬값 : 6310 비교횟수 : 2
```

이를 통해 해쉬테이블 6310번째에는 두개의 원소가 chaining이 되어 chain의 길이는 2임을 알 수 있다.

```
해쉬테이블[6310]의 chain길이 : 2
```

```
해쉬테이블[11055]의 chain길이 : 1
해쉬테이블[11058]의 chain길이 : 2
해쉬테이블[11063]의 chain길이 : 1
해쉬테이블[11064]의 chain길이 : 1
해쉬테이블[11067]의 chain길이 : 1
해쉬테이블[11069]의 chain길이 : 1
해쉬테이블[11074]의 chain길이 : 1
해쉬테이블[11075]의 chain길이 : 2
해쉬테이블[11079]의 chain길이 : 2
해쉬테이블[11082]의 chain길이 : 1
해쉬테이블[11087]의 chain길이 : 1
해쉬테이블[11090]의 chain길이 : 1
해쉬테이블[11097]의 chain길이 : 3
해쉬테이블[11101]의 chain길이 : 1
해쉬테이블[11102]의 chain길이 : 1
해쉬테이블[11105]의 chain길이 : 1
해쉬테이블[11111]의 chain길이 : 1
해쉬테이블[11114]의 chain길이 : 1
해쉬테이블[11116]의 chain길이 : 3
총 충돌 횟수 : 1235
words.dat파일 데이터들의 총 비교 횟수 : 7216
```

최종 결과를 보면 각 테이블별 chain의 길이를 알 수 있고 총 비교횟수 7216을 확인할 수 있다.