소프트웨어 공학

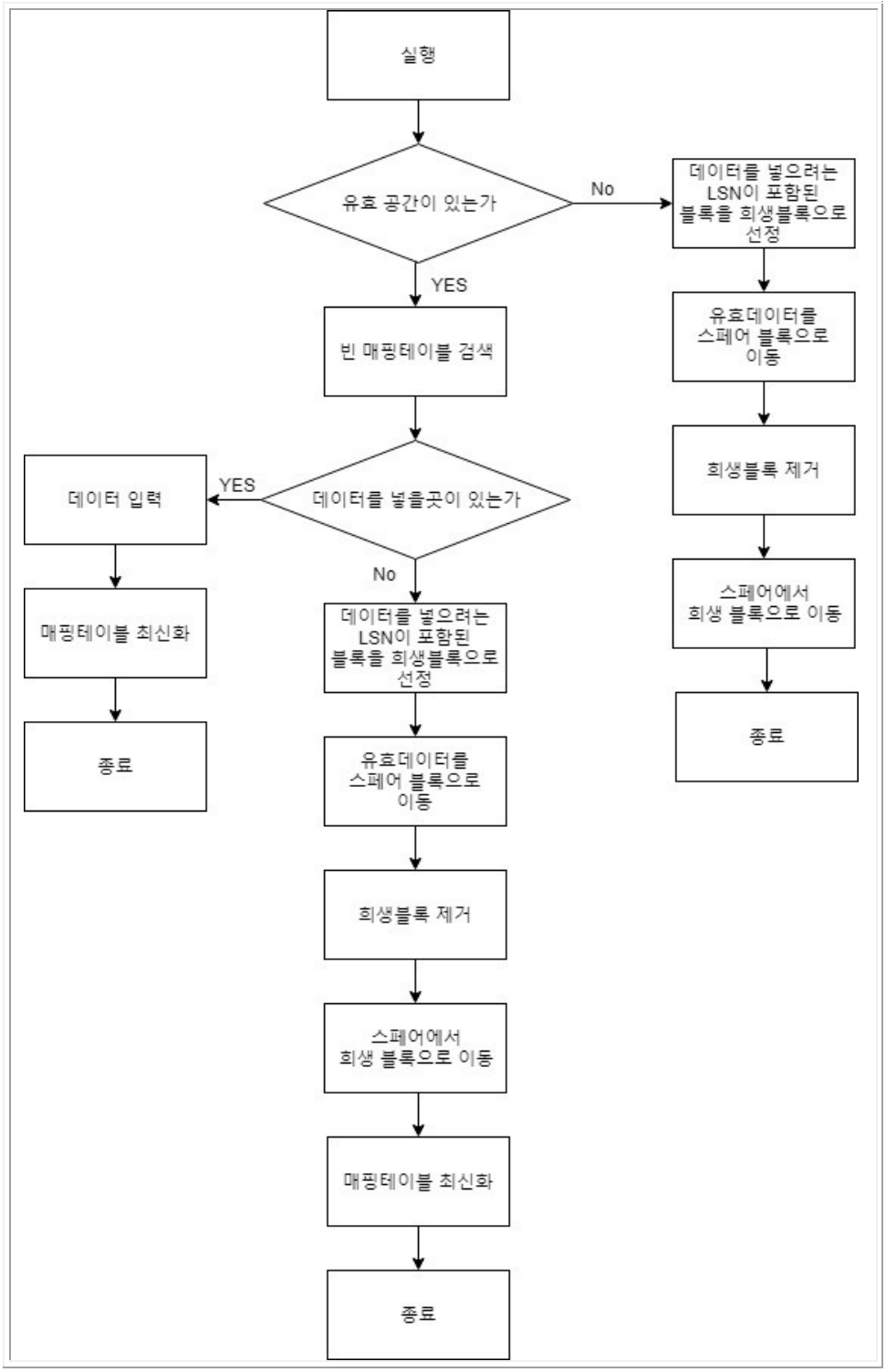
FTL

학번 : 201521920
이름 : 황찬솔
제출일 : 2018- 12 - 18
담당교수 : 권세진 교수님

1. 설계

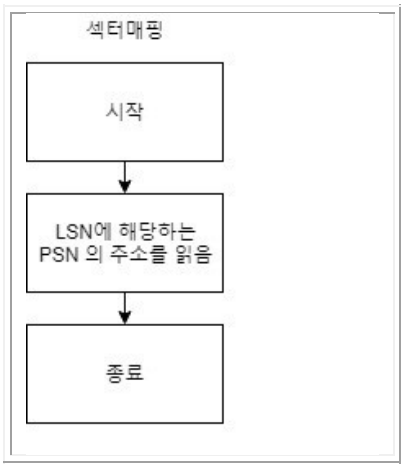
1) Sector Mapping

(1) FTL_write()



다음은 섹터 매핑 알고리즘의 쓰기 함수이다. 동적으로 할당하는 매핑 테이블에 맞추어서 이용되고 있지 않은 PSN 의 수를 따로 관리한다. FTL_write 가 호출되면 이 유효값을 확인한다. 비어 있는 PSN 이 없으면 입력 받은 LSN 에 해당되어 있던 PSN 이 있는 블록을 정리한 후 해당 위치에 데이터를 넣는다. 유효값을 확인했을 때 아직 테이블이 비어 있으면 빈 테이블을 메모리에서 순차적으로 확인하며 데이터 입력을 시도한다. 만약 끝까지 데이터를 입력하지 못했으면, 희생 블록을 하나 선정해 정리한 후 데이터를 입력한다.

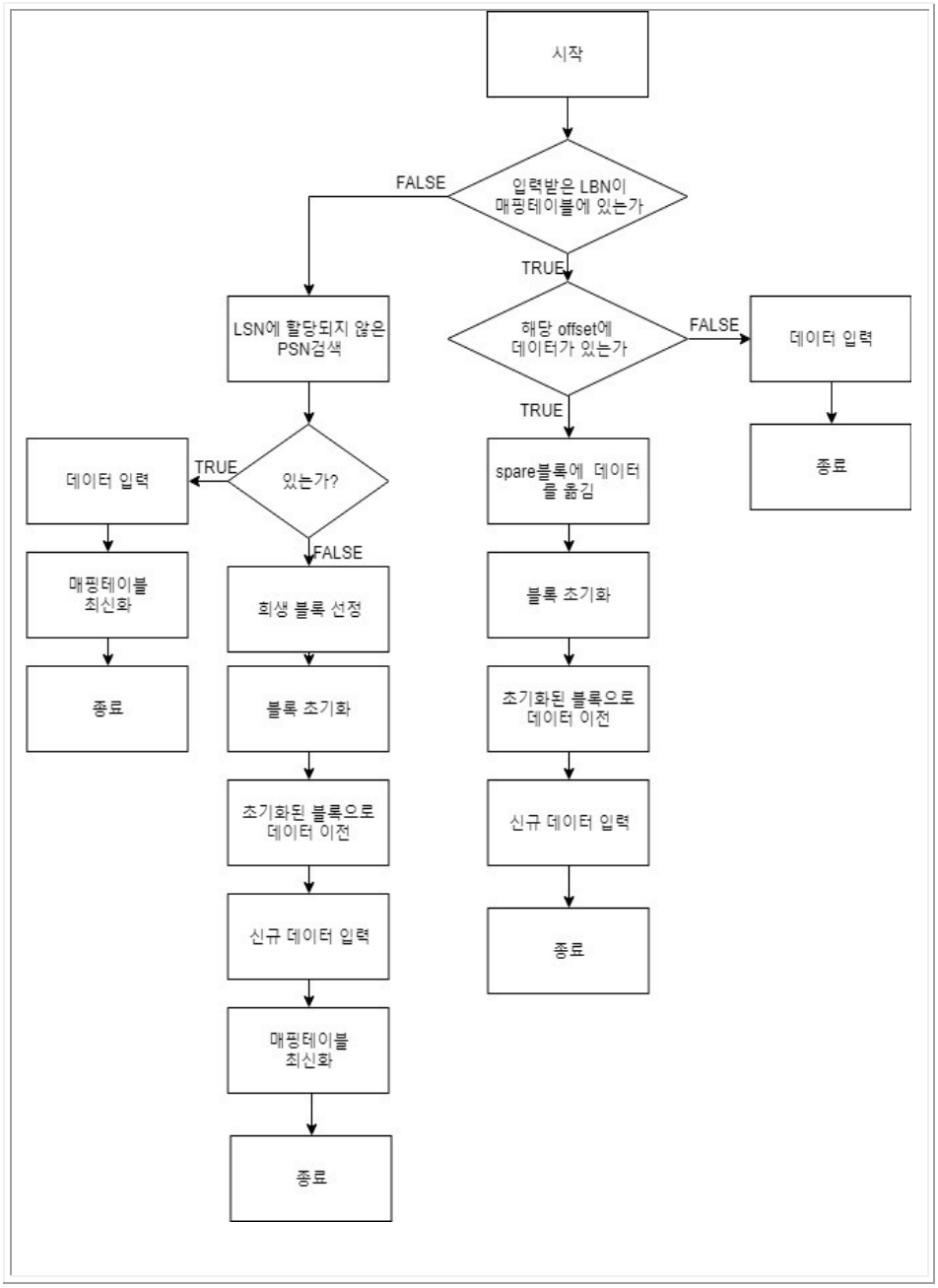
(2) FTL_read()



섹터 매핑의 읽기 함수는 단순하다. 입력 받은 LSN 을 참조하여 매핑 테이블에서 매칭되어 있는 PSN 을 찾아 해당 위치의 데이터를 찾아오기만 하면 된다.

2) Block Mapping

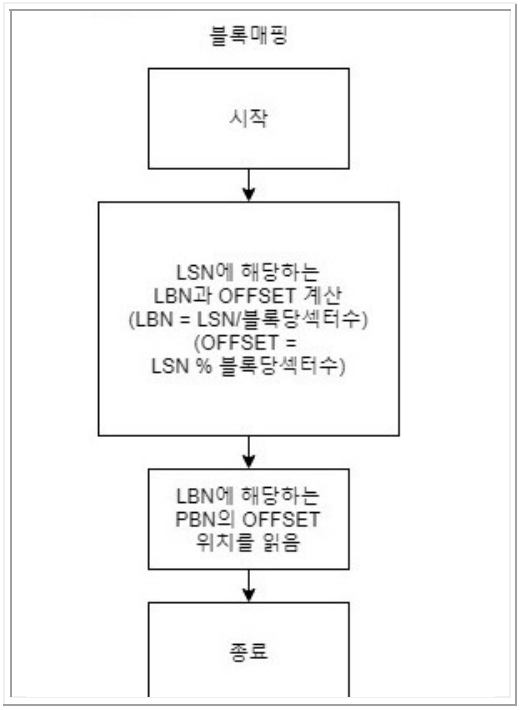
(1) FTL_write()



블록 매핑에서의 쓰기 함수이다. 우선 입력 받은 LSN 을이용하여 LBN 을 구하고 해당 LBN 과 매칭되어 있는 PBN 이 있는지 확인한다. PBN 이 존재하면 해당 LSN 을 나머지 계산하여 얻은 OFFSET 을 이용하여 해당 위치에 데이터 입력을 시도한다. 만약 해당 위치에 데이터가 존재하면 그 데이터를 제외한 다른 데이터를 모두 Spare 블록에 옮기고 블록을 초기화 한 후 데이터를 다시 제자리로 옮겨주고 입력 받은 데이터를 저장한다.

만약 LBN 에 연결된 PBN 이 없으면 비어 있는 PBN 을 찾아 데이터를 입력하고 매핑 테이블을 업데이트 시켜준다.

(2) FTL_read()



블록 매핑에서의 읽기 함수는 입력 받은 LSN 을 LBN 과 OFFSET 으로 구분한 뒤 매핑 테이블을 이용하여 데이터를 읽어온다.

2. 구현

1) 정의

(1) 매핑 테이블의 데이터 형

매핑 테이블은 2byte 로 관리한다. 이 때문에 섹터 매핑일 때 65,504 섹터, 블록 매핑일 때 65,504 블록 까지 관리할 수 있다. 65,536 이 아닌 이유는 이 프로그램에서 1 개의 블록을 여분 블록으로 관리하기 때문이다.

(2) 매핑 테이블의 저장위치

본 프로그램에서는 매핑 테이블을 별도의 메모리에 넣었다고 가정하여 작성하였다.

(3) 매핑 테이블의 초기화

매핑 테이블은 처음부터 PSN 이 주어지지 않고 사용되는 LSN 이 생길 때 PSN 을 매칭 시켜준다. 사용하지 않은 LSN 의 값은 각 스페어 블록의 주소로 초기화되어 있다.

2) 공통 함수

(1) Flash_read(unsigned short PSN)

메모리에서 데이터를 읽어오는 기본적인 함수이다. PSN 을 인수로 입력 받고 주어진 PSN 의 데이터를 읽어서 반환 한다. 만약 데이터가 존재하지 않으면 NULL 을 반환한다.

(2) Flash_write(unsigned short PSN, char data)

입력 받은 PSN 에 데이터를 저장한다. 만약 데이터가 이미 있었거나 다른 이유로 데이터 입력에 실패하면 1 을 반환하고, 정상적으로 데이터가 저장되면 0 을 반환한다.

(3) Flash_erase(unsigned short PBN)

입력 받은 PBN 의 메모리를 초기화한다.

(4) init (unsigned int Mbyte)

메모리와 매핑 테이블을 생성, 초기화 하는 함수이다. 섹터 매핑에서는 매핑 테이블이 들어간 저장공간에 총 섹터수와 사용된 PSN 을 입력하고 매핑 테이블의 PSN 을 저장한다. 매핑 테이블의 LSN 은 주소 값의 OFFSET 을 이용하기 때문에 저장하지 않는다. 데이터의 크기는 전체 섹터 수(2byte) + 유효 섹터수(2byte) + 매핑 테이블 (2byte * 섹터 수) 이다. 블록 매핑에서는 전체 블록 수(2byte) + 매핑 테이블(2byte * 블록 수) 로 저장된다.

(5) upload_table(unsigned short **maptable, FTL_INFO *info)

매핑 테이블을 파일에서 읽어오는 함수이다. 매핑 테이블 파일에서 매핑 테이블에 대한 정보를 불러와 FTL_INFO형 구조체(구조체에는 쓰이는 정보에 따라 2byte 크기로 변수가 정의되어 들어간다)에 입력하고 입력 받은 정보에 따라 매핑 테이블을 동적으로 메모리를 할당 받아 작성한다.

(6) update_table(unsigned short *maptable, FTL_INFO *info)

매핑 테이블을 최신화 하는 함수이다. 매핑 테이블이 변경되면 호출하여 변경된 정보를 매핑 테이블 파일에 저장한다.

3) Sector Mapping

(1) FTL_write(unsigned short *,FTL_INFO , char ,Counter)

FTL 을 이용한 파일 읽기 함수이다. Counter 형 구조체는 지우기 횟수와 쓰기 횟수를 계산하기 위한 구조체이다.

처음 info 에 저장된 Number_of_avail 값을 이용하여 테이블이 가득 찼을 경우를 판단하고 만약 모든 데이터가 유효데이터로 차 있으면 입력 받은 LSN 에 대한 PSN 을 지우고 해당 위치에 데이터를 입력한다. 데이터가 가득 찬 경우가 아니라면 이 함수는 PSN 의 0 번부터 데이터가 저장되어 있는지

매핑 테이블을 이용하여 검색한다.

```
for (existPSN = 0; existPSN < info->Number_of_Sector; existPSN++) {
    for (count = 0; count < info->Number_of_Sector; count++) {
        if (maptable[count] == existPSN) {   //psn이 매핑테이블에 존재하면 다음psn으로 넘어감
            break;
        }
    }
    if (count == info->Number_of_Sector) {  //existPSN에 해당하는 PSN이 검색되지 않았을 때 실행
        count_avail++;
        if (Flash_write(existPSN, data) == 0) { //빈 공간이 발견되어 정상적으로 저장되었으면
            // test@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
            counter->Wcounter++;
            if (maptable[LSN] != info->Number_of_Sector)
                info->Number_of_avail++;
            maptable[LSN] = existPSN;       //매핑테이블 변경
            update_table(maptable, info);   //매핑테이블 업데이트
            return 0;
        }
    }
    if (count_avail + info->Number_of_avail == info->Number_of_Sector) {    // 매핑테이블은 비어있으나
        break;                                                              // 쓸수 있는 공간이 없을때
    }
}
// 여기까지 넘어오면 공간 확보가 필요하다고 판단.
// 희생블록 탐색
```

위 사진은 매핑 테이블에서 사용되지 않은 PSN 을 검색하는 코드이다. 0 번 PSN 부터 순차적으로 접근하며 하나의 PSN 을 검색하기 위해 매핑 테이블에서 해당 PSN 이 발견될 때 까지 매핑 테이블을 순차검색 한다. exisetPSN 과 일치하는 매핑 테이블이 발견되지 않으면 데이터를 입력한다. 만약 데이터 입력에 실패하면 데이터를 입력할 공간을 확보하기 위해 희생될 블록을 선정한다. 선정 방식은 메모리의 0 번주소부터 순차적으로 검색하여 처음으로 나오는 매핑 테이블에 존재하지 않는 섹터가 포함된 블록의 주소를 알아내어 해당 블록의 유효 데이터를 spare에 옮기고 블록을 정리한 뒤 spare 에 옮겼던 데이터를 원 블록으로 다시 복귀시킨다. 모든 처리가 완료되면 데이터를 빈 공간에 입력한 뒤 매핑 테이블을 업데이트 시켜준다.

(2) FTL_read(unsigned short *, FTL_INFO, unsigned LSN)

```
char FTL_read(unsigned short *maptable, FTL_INFO *info, unsigned short LSN) {
    if (LSN < 0 || LSN >= info->Number_of_Sector||maptable[LSN]==info->Number_of_Sector) {
        printf("> 해당 영역은 유효하지 않습니다₩n");
        return NULL;
    }
    printf(">%hd번 PSN에 저장되어있습니다. ₩n", maptable[LSN]);
    return Flash_read(maptable[LSN]);
}
```

입력 받은 LSN 이 유효한 범위인지 판단하고 Flash_read()를 호출하여 원하는 LSN 의 데이터를 반환한다.

4) Block Mapping

(1) FTL_write(unsigned short *, FTL_INFO, unsigned short, char, Counter)

이 함수는 블록 매핑에서의 데이터 입력 함수이다. 함수가 호출되면 크게 두 분기로 나누어진다. 입력 받은 LSN 에 해당하는 LBN 을 구하고 LBN 에 해당하는 매핑 테이블에 PBN 이 존재 하는지 확인한다. 매핑 테이블에서 PBN 을 찾는 알고리즘은 섹터 매핑의 매핑 테이블 탐색과 같다. 매핑 테이블을 순차적으로 확인하며 해당하는 PBN 이 존재하는지를 검사한다.

매핑 테이블에 PBN 이 존재하면 해당 PBN 에 입력 받은 LSN 을 이용해 구한 OFFSET 을 이용하여 데이터를 입력한다. 만약 해당 섹터에 데이터가 존재하면 해당 섹터를 제외한 나머지 섹터를 전부 spare 영역에 저장하고 블록을 지운 뒤 다시 spare 에 있는 데이터를 이동시킨다.

```
for (count_PBN = 0; count_PBN < info->Number_of_Block; count_PBN++) {
    for (count = 0; count < info->Number_of_Block; count++) {
        if (maptable[count] == count_PBN) {
            break;      //해당 PBN은 매핑테이블에 존재하므로 다음으로 이동
        }
    }
    if (count == info->Number_of_Block) {   // count_PBN에 해당하는 PBN이 매핑테이블에 존재하지 않으면
        if (Flash_write(count_PBN*BtoS+LSN%BtoS, data) == 0) {  // 해당 위치에 데이터 입력. overwrite에러가 나지 않으면 종료
            counter->Wcounter++;
            maptable[LSN/BtoS] = count_PBN;
            update_table(maptable, info);
            return 0;
        }
    }   //입력에 실패하면 다음 빈 블록 검색
}
```

위 코드는 매핑 테이블에 할당되지 않은 PBN 을 검색하는 소스이다. 전체적인 과정은 섹터 매핑과 같으며 원하는 offset 에 데이터를 넣을 수 있으면서 동시에 매핑 테이블에 할당되지 않은 블록을 찾을 때 까지 반복된다. 해당 과정에 실패하면 희생블록을 선정하여 해당 블록을 정리하는 과정을 수행한다.

(2) FTL_read(unsigned short, FTL_INFO)

입력 받은 LSN 을 섹터당 블록수로 나누어 LBN 을 구하고 그 나머지를 OFFSET 으로 설정하여 LBN 에 해당하는 PBN*(블록당 섹터 수) + OFFSET 을 이용하여 데이터를 읽어온다.

3. 테스트 결과

1) Sector Mapping

```
>Number of Sector : 2048                Number of ableSector : 0
>
>       <Flash Memory Operating System>> This program use SectorMapping System
>
> Sector's data type is character
> Block consist of 32 sectors
> If want Exit the Program, commanding "end"
> Option type [ Write data ] : w (LSN) (data)                    ex) w 4 a
>
> Option type [ read data ] : w (PSN)                            ex) r 4
>
> Option type [ initializing Flash Memory ] : init (megabyte)    ex) init 4
>
> Option type [ Show table ] : t (Block number)                  ex) t
>
>> init 1
1 megabytes virtual flash memory are created
> successfully upload table
```

실행 후 init 1 로 1MB 의 메모리를 할당한다.

크기:	2.03KB (2,080 바이트)	크기:	4.00KB (4,100 바이트)

메모리의 크기는(좌측) 2048(1 바이트의 2048 섹터) + 32(여분블록)만큼 생성, 매핑 테이블의 크기(우측) 는 2048 * 2byte(매핑 테이블의 크기)+4(정보) 생성

```
> Option type [ Show table ] : t (Block number)                ex) t
>
>> t
----------------매핑 테이블----------------------
LSN             PSN             LSN             PSN             LSN             PSN

0               2048            1               2048            2               2048
3               2048            4               2048            5               2048
6               2048            7               2048            8               2048
9               2048            10              2048            11              2048
12              2048            13              2048            14              2048
15              2048            16              2048            17              2048
18              2048            19              2048            20              2048
21              2048            22              2048            23              2048
24              2048            25              2048            26              2048
27              2048            28              2048            29              2048
30              2048            31              2048            32              2048
33              2048            34              2048            35              2048
36              2048            37              2048            38              2048
39              2048            40              2048            41              2048
42              2048            43              2048            44              2048
45              2048            46              2048            47              2048
48              2048            49              2048            50              2048
51              2048            52              2048            53              2048
54              2048            55              2048            56              2048
```

테이블을 확인하면 위와 같이 PSN 값이 2048 로 초기화 된 것을 알 수 있다.

```
87              2048            88              2048            89              2048
90              2048            91              2048            92              2048
93              2048            94              2048            95              2048
96              2048            97              2048            98              2048
99              2048            100             0               101             2048
102             2048            103             2048            104             2048
105             2048            106             2048            107             2048
108             2048            109             2048            110             2048
111             2048            112             2048            113             2048
```

W 100 A 를 통해 데이터를 입력하면 LSN 100 에 해당하는 PSN 이 0 으로 변경됨을 확인할 수 있다.

```
>> w 100 A
> successfully upDate table
> successfully upload table
> 데이터 쓰기에 성공했습니다
쓰기 : 1 회,  지우기 : 0회, 지워진 블록 :-1 번 블록
>
```

데이터의 쓰기 횟수가 1 번인 것은 비어있는 섹터에 다른 작업 없이 데이터를 입력했기 때문이다.

```
00000000  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000010  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000020  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000030  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
```

다음과 같이 모든 데이터가 x 로 입력되어 있는 더미 메모리로 테스트를 위해 교체한다. 매핑테이블은 모두 공백상태로 초기화했다.

```
>> w 100 a
>successfully erase
>successfully erase
> successfully upDate table
> successfully upload table
> 데이터 쓰기에 성공했습니다
쓰기 : 1 회,  지우기 : 2회, 지워진 블록 :0 번 블록
```

```
0B 0C 0D 0E 0F
20 20 20 20 20  a
20 20 20 20 20
58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58  XXXXXXXXXXXXXXXX
```

데이터가 차 있을때는 희생 블록을 지우고 해당 위치에 데이터를 쓰기 때문에 지우기가 2 회 발생하였다.

```
>> r 100
>0번 PSN에 저장되어있습니다.
>   100 주소에 저장된 값 : a
>
```

Flash read 를 호출하여 100 번 주소를 검색하면 a 가 반환되어 출력된다.
현재 a 는 PSN 0 에 저장되어 있기에 0 번 PSN 에 저장되어있음을 출력한다.

2) Block Mapping

```
> successfully upload table
>
>Number of Sector : 64
>
>       <Flash Memory Operating System>> This program use BlockMapping System
> Sector's data type is character
> Block consist of 32 sectors
> If want Exit the Program, commanding "end"
> Option type [ Write data ] : w (LBN) (data)                    ex) w 4 a
>
> Option type [ read data ] : w (PSN)                            ex) r 4
>
> Option type [ initializing Flash Memory ] : init (megabyte)    ex) init 4
>
> Option type [ Show table ] : t (Block number)                  ex) t
>
>> init 1
1 megabytes virtual flash memory are created
> successfully upload table
>
```

실행 후 init 1 로 1MB 의 메모리를 할당한다.

크기:	2.03KB (2,080 바이트)	크기:	130바이트 (130 바이트)

메모리의 크기는(좌측) 2048(1 바이트의 2048 섹터) + 32(여분블록)만큼 생성,
매핑 테이블의 크기(우측) 는 64 * 2byte(매핑 테이블의 크기)+2(정보) 생성

```
>> t
---------------매핑 테이블----------------------
LBN             PBN             LBN             PBN             LBN             PBN

0               64              1               64              2               64
3               64              4               64              5               64
6               64              7               64              8               64
9               64              10              64              11              64
12              64              13              64              14              64
15              64              16              64              17              64
18              64              19              64              20              64
21              64              22              64              23              64
24              64              25              64              26              64
27              64              28              64              29              64
30              64              31              64              32              64
33              64              34              64              35              64
36              64              37              64              38              64
39              64              40              64              41              64
42              64              43              64              44              64
45              64              46              64              47              64
48              64              49              64              50              64
51              64              52              64              53              64
54              64              55              64              56              64
57              64              58              64              59              64
60              64              61              64              62              64
63              64              ------------------------------------------------
```

테이블을 확인하면 위와 같이 PBN 값이 64 로 초기화 된 것을 알 수 있다.

```
0       64      1       64      2       64
3       0       4       64      5       64
6       64      7       64      8       64
9       64      10      64      11      64
12      64      13      64      14      64
```

W 100 A 를 통해 데이터를 입력하면 LBN 3 에 해당하는 PBN 이 0 으로 변경됨을 확인할 수 있다.((int)100 / 32 = 3)

```
>> w 100 A
> successfully upDate table
> successfully upload table
> 데이터 쓰기에 성공했습니다
쓰기 : 1 회,  지우기 : 0회, 지워진 블록 :-1 번 블록
>
```

비어있는 상태의 메모리에 데이터를 입력하는 작업은 섹터 매핑과 같은 성능을 보인다.

```
00000000  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000010  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000020  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00000030  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
```

다음과 같이 모든 데이터가 x 로 입력되어 있는 더미 메모리로 테스트를 위해 교체한다. 매핑테이블은 모두 공백상태로 초기화했다.

```
>> w 100 a
>successfully erase
> successfully upDate table
>  성공적으로 데이터를 입력했습니다.
> successfully upload table
> 데이터 쓰기에 성공했습니다
쓰기 : 1 회,  지우기 : 1회, 지워진 블록 :0 번 블록
>
```

```
0B 0C 0D 0E 0F

20 20 20 20 20      a
20 20 20 20 20
58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58  XXXXXXXXXXXXXXXX
```

블록 매핑에서 희생 블록이 데이터가 없는 상태라 판단했기 때문에 지우기가 1 번 발생했다

```
09 0A 0B 0C 0D 0E 0F
65 20 20 20 20 20 20      abcd e
20 20 20 20 20 20 20
58 58 58 58 58 58 58  XXXXXXXXXXXXXXX
58 58 58 58 58 58 58  XXXXXXXXXXXXXXX
```

테스트를 위해 데이터를 추가로 입력하고 이전에 사용했던 100 번주소에 데이터 P 를 입력한다.

```
>> w 100 P
>successfully erase
>successfully erase
>  성공적으로 데이터를 입력했습니다.
> successfully upload table
> 데이터 쓰기에 성공했습니다
쓰기 : 11 회,  지우기 : 2회, 지워진 블록 :0 번
>
```

아직 블록이 비어있음에도 불구하고 블록 매핑에서는 지우기 2 회와 입력되어 있는 데이터 *2 + 1(새 데이터)만큼의 쓰기가 사용되었다.

3) 분석

섹터 매핑과 블록 매핑 방법에 대한 FTL 알고리즘을 각각 구현하였다. 최대한 두 알고리즘이 구조가 비슷하도록 설계하였다. 두 방법을 비교했을 때 가장 두드러지게 나타난 차이는 매핑 테이블의 크기였다. 같은 1MB 의 메모리를 구현하면 섹터는 4096byte , 블록은 128byte 의 크기의 매핑 테이블이 생성되었다. 또다른 차이로는 같은 LSN 에 데이터를 입력하였을 때의 처리이다. 섹터 매핑은 동일한 LSN 이 입력되었을 경우 빈 섹터를 찾아 입력하고 매핑 테이블을 초기화 시켰다. 때문에 모든 메모리가 데이터로 차기 전 까지는 데이터를 입력하는데 지우기가 발생하지 않았다. 그러나 블록 매핑은 같은 주소에 데이터를 입력했을 경우 동일한 LBN 을 가지는 데이터를 위해 지우기, 쓰기 연산이 계속 사용되었다.

FTL 을 구현해 봤을 때 확실하게 데이터를 관리하는 것은 섹터에 대한 정보가 많은 방식이다. 그러나 정보가 많아질수록 매핑 테이블의 크기는 늘어나고 특정

섹터 수를 넘어갔을 때 정보를 관리하는 바이트의 크기를 늘려야 해서 매핑 테이블의 크기가 크게 커짐을 볼 수 있었다. 이를 보완해서 각종 정보를 offset 으로 처리하여 여러 블록도 offset 으로 관리하게 되면 섹터나 블록 수가 많아져도 해당 offset 정보를 활용해서 작은 데이터 형으로 관리 할 수 있을 것 같다.