**Pintos Project 0-2: Pintos Data Structure**

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 |
| 학번 : | 20170101 |
| 이름 : | 이은지 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | Int findIndex(int mode, char \*string) |
| **Parameter** | Data structure and the string including the index |
| **Return** | Integer of index |
| **Function** | mode = 0, 비트맵은 1, 해쉬는 2로 설정하여 자료구조를 알 수 있게 한다. 그리고 입력된 문자열에서 인덱스를 추출한다. 예를 들어 mode가 0이라면 자료구조가 리스트임을 알 수 있고 , 입력된 문자열이 ‘list2’일 경우에는 해당 문자열 ‘list’에서 string[4]를 사용하여 ‘2’를 분리할 수 있다. 아스키코드를 활용하여 문자를 정수형 숫자로 변환하고(문자 – ‘0’) 인덱스값을 리턴한다. 입력되는 자료구조가 최대 10개임을 인지하고 자료구조에 따라 list/bmhash보다 각각 문자열의 길이가 2개 증가되어있다면 인덱스는 반드시 10임이기 때문에 별도 처리한다. |

|  |  |
| --- | --- |
| **Prototype** | Void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | struct list\_elem \*a, struct list\_elem \*b |
| **Return** | None |
| **Function** | Swap two list elements in parameters. 포인터를 활용하여 a와 b가 이웃한 경우, 그렇지 않은 경우를 별도 처리한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct \*list) |
| **Parameter** | List that will be shuffled |
| **Return** | None |
| **Function** | Shuffle elements of List in the parameter. 적절한 난수를 활용하여 해당 갯수만큼 위 list\_swap함수를 호출한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*a, struct list\_elem \*b, void \*aux) |
| **Parameter** | Two list elements |
| **Return** | Return true if the value of a is less than b, or return false if a is greater than or equal to b. |
| **Function** | Compare the values of two list elements A and B |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | Integer that will be hashed |
| **Return** | Hash value of integer i |
| **Function** | Using division(제산법), find the address with the remainder |

|  |  |
| --- | --- |
| **Prototype** | Bool hash\_less(const struct hash\_elem \*a,const struct hash\_elem \*b, void \*aux) |
| **Parameter** | Two hash elements |
| **Return** | Return true if the value of a is less than b, or return false if a is greater than or equal to b. |
| **Function** | Compare the values of two list elements A and B |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | Hash element and auxiliary |
| **Return** | The hash value of data |
| **Function** | Applying the hash function |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | Hash element and a string aux |
| **Return** | Computed data of hash element |
| **Function** | Performs some operation on hash element, given aux. aux는 문자열 포인터로 활용하였고, 만약 ‘square’가 입력되면 hash\_elem의 데이터를 제곱하고, ‘triple’이 입력되면 세제곱 처리한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size) |
| **Parameter** | Bitmap that you want to expand and the size of it |
| **Return** | Expanded bitmap if succeed, NULL, if fail |
| **Function** | Expand the given Bitmap to the size. 새로운 비트맵 구조체를 선언하고 메모리를 할당하여 데이터를 옮긴다. 그 후 이전 구조체는 메모리 해제한다. 그리고 새로운 확장된 비트맵 구조체를 리턴한다. |

**II. List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list \*) |
| **Parameter** | A list |
| **Return** | None |
| **Function** | Initialize list as an empty list and set the pointers |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin(struct list\*) |
| **Parameter** | A list |
| **Return** | The beginning of list |
| **Function** | 리스트의 head 포인터의 next가 가리키는 element를 가져온다. 즉 첫번째 element 리턴 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next(struct list\_elem \*) |
| **Parameter** | A list element |
| **Return** | Return the element after the parameter in its list. |
| **Function** | Return the element after the parameter in its list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end(struct list \*) |
| **Parameter** | A list |
| **Return** | Return the tail of the list |
| **Function** | Return the tail of the list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin(struct list \*) |
| **Parameter** | A list |
| **Return** | Return the last element of the list (just before the tail) |
| **Function** | Return the list’s reverse beginning, that is, the last element of the list (just before the tail) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev(struct list\_elem \*) |
| **Parameter** | A list element |
| **Return** | Return the element before the parameter in its list. |
| **Function** | Return the element before the parameter in its list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend(struct list \*) |
| **Parameter** | A list |
| **Return** | Return the head of the list |
| **Function** | Return the list’s reverse end, that is, the first element of the list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head(struct list \*) |
| **Parameter** | A list |
| **Return** | Return the head of the list |
| **Function** | Return the head of the list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail(struct list \*) |
| **Parameter** | A list |
| **Return** | Return the tail of the list |
| **Function** | Return the tail of the list |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert(struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | A list element to be attached(before) and a list element to be inserted(elem) |
| **Return** | None |
| **Function** | Insert a second element just before the first element. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice(struct list\_elem \*before, struct list\_elem \*first,struct list\_elem \*last) |
| **Parameter** | A list element to be attached(before) and the elements in the list(first, last) |
| **Return** | None |
| **Function** | Remove elements first through last from their current list, and then inserts them just before first element(before) |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front(struct list\*, struct list\_elem \*) |
| **Parameter** | A list and a list element to be inserted |
| **Return** | None |
| **Function** | Insert elem at the beginning of list, so that it becomes the front in the list |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back(struct list\_elem \*, struct list\_elem \*) |
| **Parameter** | A list and a list element to be inserted |
| **Return** | None |
| **Function** | Insert elem at the end of list, so that it becomes the back in the list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove(struct list\_elem \*) |
| **Parameter** | A element to be removed |
| **Return** | Return removed element. |
| **Function** | Iterate the list and find out the element and remove it. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front(struct list \*) |
| **Parameter** | A list |
| **Return** | Return removed element. |
| **Function** | Remove the front element from list and return it |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back(struct list \*) |
| **Parameter** | A list |
| **Return** | Return removed element. |
| **Function** | Remove the back element from list and return it |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front(struct list \*) |
| **Parameter** | A list |
| **Return** | The front element in list |
| **Function** | The front element in list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back(struct list \*) |
| **Parameter** | A list |
| **Return** | The back element in list |
| **Function** | The back element in list |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size(struct list \*) |
| **Parameter** | A list |
| **Return** | The size of list |
| **Function** | Count the number of list\_element in the list |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty(struct list\*) |
| **Parameter** | A list |
| **Return** | Return true if the list is empty, false otherwise. |
| **Function** | Compare the values of two list elements A and B |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort(struct list\*,list\_less\_func \*,void \*aux) |
| **Parameter** | A list, the function pointer that compares the element and auxiliary data |
| **Return** | None |
| **Function** | Sort the list |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort(struct list\*,list\_less\_func \*,void \*aux) |
| **Parameter** | A list, the function pointer that compares the element and auxiliary data |
| **Return** | None |
| **Function** | Sort the list |

|  |  |
| --- | --- |
| **Prototype** | static void swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | 리스트의 포인터를 swap한다. |
| **Return** | None |
| **Function** | 포인터 자체를 swap하는 것이기 때문에 내부 포인터 조정이 필요없다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | A list |
| **Return** | None |
| **Function** | 리스트를 돌면서 element마다 prev와 next포인터를 swap하며 리스트의 앞뒤를 전환한다. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux) |
| **Parameter** | Two elements, the function pointer that compares the element and auxiliary data |
| **Return** | Return true only if the list elements a through b are in order |
| **Function** | 순서대로 나열되어있으면 true 반환. a와 a이전의 값을 비교하여 만약 a가 list\_prev(a)보다 작으면 즉각적으로 false 반환. 즉 비내림차순이어야 true반환. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*find\_end\_of\_run (struct list\_elem \*a, struct list\_elem \*b, list\_less\_func \*less, void \*aux) |
| **Parameter** | Two elements, the function pointer that compares the element and auxiliary data |
| **Return** | Return the (exclusive) end of the run |
| **Function** | 비내림차순임을 확인함과 동시에 b를 탐색할 때까지 a는 list\_next함수를 이용하여 리스트를 순회한다. |

|  |  |
| --- | --- |
| **Prototype** | static void inplace\_merge (struct list\_elem \*a0, struct list\_elem \*a1b0, struct list\_elem \*b1, list\_less\_func \*less, void \*aux) |
| **Parameter** | Three elements, the function pointer that compares the element and auxiliary data |
| **Return** | None |
| **Function** | Merges A0 through A1B0 (exclusive) with A1B0 through B1  (exclusive) to form a combined range also ending at B1  (exclusive). |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered(struct list\*,struct list\_elem \*,list\_less\_func \*, void \*aux) |
| **Parameter** | A list, the function pointer that compares the element and auxiliary data |
| **Return** | None |
| **Function** | Insert element in the proper position in list |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique(struct list\*,struct list \*duplicates,list\_less\_func \*, void \*aux) |
| **Parameter** | Two lists, the function pointer that compares the element and auxiliary data |
| **Return** | None |
| **Function** | Remove the duplicates after comparing each element, and insert second list by list\_push\_back() |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | A list, the function pointer that compares the element and auxiliary data |
| **Return** | Return the element in the list with the largest value |
| **Function** | 리스트를 돌면서 가장 큰 값을 가지는 element를 탐색한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | A list, the function pointer that compares the element and auxiliary data |
| **Return** | Return the element in the list with the smallest value |
| **Function** | 리스트를 돌면서 가장 작은 값을 가지는 element를 탐색한다. |

**III. Hash Table**

|  |  |
| --- | --- |
| **Prototype** | Bool hash\_init (struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | Hash table, hashing function that compute hash values, comparing function and auxiliary data |
| **Return** | Return true if initializing the hash table successes, or false otherwise |
| **Function** | 해시테이블의 멤버 변수를 초기화하고 bucket의 메모리를 할당한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | Hash Table, auxiliary function pointer |
| **Return** | None |
| **Function** | Remove all the elements from hash table |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | Hash Table, auxiliary function pointer |
| **Return** | None |
| **Function** | Remove the hash table and free memories |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_insert(struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | Hash table and a hash element to be inserted |
| **Return** | The inserted hash element |
| **Function** | 해당 hash element에 적절한 bucket을 탐색한 뒤, 만약 bucket에 동일한 element가 없는 경우에만 삽입을 처리한다. 그리고 이에 맞춰 해시테이블의 정보도 수정한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | Hash table and a hash element to replace |
| **Return** | The replacing hash element |
| **Function** | 해당 hash element에 적절한 bucket을 탐색한 뒤, 만약 bucket에 동일한 element가 없는 경우에는 insert와 동일하게 삽입을 처리한다. 아닐 경우에는 기존의 element를 삭제하고 삽입을 처리한다. 그리고 이에 맞춰 해시테이블의 정보도 수정한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | Hash table and a hash element to find |
| **Return** | The hash element to find |
| **Function** | 찾고자 하는 element의 bucket에서 해당 element를 찾으면 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | Hash table and a hash element to delete |
| **Return** | The hash element deleted |
| **Function** | Hash Table에서 hash\_find를 이용하여 해당 element를 찾은 후 remove한다. 마찬가지로 이에 맞춰 해시테이블의 정보도 수정한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply(struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | Hash table and hash\_action\_function |
| **Return** | None |
| **Function** | Applying the hash action(like square , triple) to the data of element. And delivery the action through the void pointer |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first(struct hash\_iterator \*I, struct hash \*h) |
| **Parameter** | Iterator pointer and hash table |
| **Return** | None |
| **Function** | Set the iterator to the hash table |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next(struct hash\_iterator \*i) |
| **Parameter** | Iterator pointer |
| **Return** | The hash\_elem next to the current hash\_elem |
| **Function** | Return the next hash element |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur(struct hash\_iterator \*i) |
| **Parameter** | Iterator pointer |
| **Return** | The current hash\_elem |
| **Function** | Return the current hash\_elem that parameter is pointing |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_elem \*hash\_size(struct hash \*h) |
| **Parameter** | Hash table |
| **Return** | The number of elements of the hash table |
| **Function** | Return the number of elements of the hash table |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty(struct hash \*h) |
| **Parameter** | Hash table |
| **Return** | Return true if the hash is empty, or false otherwise |
| **Function** | Return true if the hash is empty, or false otherwise |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes(const void \*buf\_, size\_t size) |
| **Parameter** | Buffer and the size of buffer |
| **Return** | Return a hash of the size bytes in buffer |
| **Function** | 해시 함수의 핵심이 되는 함수이다. 입력되는 키의 주소를 버퍼로 받아와 임의의 상수와 연산을 한 후 그 값을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string(const char \*s\_) |
| **Parameter** | A string |
| **Return** | Return a hash of string |
| **Function** | 위는 숫자 계열의 키를 연산한다면, 해당함수는 동일한 값을 이용하나 문자열을 연산하여 해시를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int(int i) |
| **Parameter** | Integer that will be hashed |
| **Return** | Hash value of integer i |
| **Function** | 입력된 숫자의 주소값을 hash\_bytes로 전달하는 중간 매개역할의 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*find\_elem(struct hash \*h, struct list\*bucket, struct hash\_elem \*e) |
| **Parameter** | Hash table, bucket, and element to find |
| **Return** | Hash element to find, or NULL if there’s no element in the bucket |
| **Function** | List형식으로 구현된 bucket에서 순회하면서 element의 데이터값과 일치하는지 확인하고 일치한다면(어느 한 쪽도 적지 않다면 일치한다는 의미) 리턴하고 그렇지 않다면 NULL을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_bit(size\_t x) |
| **Parameter** | Size\_t 형식의 수 |
| **Return** | A number whose LSB is 0 |
| **Function** | 비트 연산으로 &연산자를 활용하여 LSB를 0로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | Static inline size\_t is\_power\_of\_2(size\_t x) |
| **Parameter** | Size\_t 형식의 수 |
| **Return** | Return true if X is a power of 2, otherwise false |
| **Function** | turn\_off\_least\_bit함수를 활용하여 2의 제곱수를 판별할 수 있다. 2의 제곱수인 이진수들은 MSB만 1이고 나머지 자릿수들은 모두 0이다. 따라서 2의 제곱수를 turn\_off\_least\_bit함수 처리하면 반드시 결과값은 0이 된다. 0은 2의 제곱수가 아님에도 불구하고 해당 함수의 결과가 0이 나오므로 이 경우를 제외하고 처리하여 2의 제곱수를 판별할 수 있다.  [참고]  If x=4, in the turn\_off\_least\_bit function  100 = 4  011 = 3  -------------  000 |

|  |  |
| --- | --- |
| **Prototype** | Static void rehash(struct hash \*h) |
| **Parameter** | Hash table |
| **Return** | None |
| **Function** | Check and change the number of buckets in hash table to match the current condition. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem(struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | Hash table, bucket, and hash element |
| **Return** | None |
| **Function** | Insert element into the bucket using list\_push\_front |

|  |  |
| --- | --- |
| **Prototype** | static void remove\_elem(struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | Hash table, hash element |
| **Return** | None |
| **Function** | Hash element 내에 list element가 내포되어있음을 활용하여 element를 삭제한다. |

**Ⅳ. Bitmap**

|  |  |
| --- | --- |
| **Prototype** | Static inline size\_t elem\_idx(size\_t bit\_idx) |
| **Parameter** | index of the bit |
| **Return** | The index of the element that contains the bit numbered bit\_idx |
| **Function** | Bits / the size of bits = index |

|  |  |
| --- | --- |
| **Prototype** | Static inline elem\_type bit\_mask(size\_t bit\_idx) |
| **Parameter** | index of the bit |
| **Return** | Return an elem\_type where only the bit corresponding to bit\_idx is turned on |
| **Function** | Using the shift, turn on the bit on the index and return it in elem\_type |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt(size\_t bit\_cnt) |
| **Parameter** | Number of bits |
| **Return** | Return the number of elements required for bit\_cnt bits |
| **Function** | Elements # = bit\_cnt / elem\_bits(number of bits in element) 의 올림값 |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt(size\_t bit\_cnt) |
| **Parameter** | Number of bits |
| **Return** | Return the number of bytes required for bit\_cnt bits |
| **Function** | Bytes # = elem\_type \* elem\_cnt |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t last\_mask(const struct bitmap \*b) |
| **Parameter** | Bitmap |
| **Return** | Return a bit mask in which the bits actually used in the last element of bitmap’s bits are set to 1 and the rest are set to 0. |
| **Function** | Elements # = bit\_cnt / elem\_bits(number of bits in element) 의 올림값임을 감안하여 elements에 쓰이지 않은 남은 비트를 last\_bits로 정의한다. 만약 last\_bits가 존재한다면 좌측, 존재하지 않는다면 우측 값을 적용한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create(size\_t bit\_cnt) |
| **Parameter** | Size of bitmap |
| **Return** | bitmap |
| **Function** | Initialize bitmap of bit\_cnt bits and sets all of its bits to false(0) |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_create\_in\_buf(size\_t bit\_cnt, void \*block, size\_t block\_size) |
| **Parameter** | Size of bitmap, preallocated storage |
| **Return** | bitmap |
| **Function** | Create and return a bitmap with bit\_cnt bits in the block size bytes of storage prellocated at block |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy(struct bitmap \*b) |
| **Parameter** | bitmap |
| **Return** | None |
| **Function** | Free the bits and free the struct bitmap |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set(struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | Bitmap and index to set value |
| **Return** | None |
| **Function** | Turn the idx to elem\_idx and set the bit by masking |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark(struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | Bitmap and index to set to 1 |
| **Return** | None |
| **Function** | Turn the idx to elem\_idx and set the bit as 1 by masking |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset(struct bitmap \*b, size\_t idx) |
| **Parameter** | Bitmap and index to set to 0 |
| **Return** | None |
| **Function** | Turn the idx to elem\_idx and set the bit as 0 by masking |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip(struct bitmap \*b, size\_t idx) |
| **Parameter** | Bitmap and index to flip |
| **Return** | None |
| **Function** | Turn the idx to elem\_idx and set the bit 1 to 0, 0 to 1 by masking |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_test(struct bitmap \*b, size\_t idx) |
| **Parameter** | Bitmap and index to get the value |
| **Return** | None |
| **Function** | &연산자를 이용하여 추출한다. Bits 중 idx에 해당하는 elem에서 bit\_mask를 이용하여 비트 위치에 해당하는 비트를 추출할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple(struct bitmap \*b, size\_t start, size\_cnt, bool value) |
| **Parameter** | Bitmap and set the value from start to start +cnt |
| **Return** | None |
| **Function** | Bitmap\_set을 start부터 start+cnt 까지 반복 호출한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | Bitmap, the index of start and cnt, value |
| **Return** | Return the number of bits in bimap between start and start+cnt |
| **Function** | bitmap\_test로 비트의 값을 추출한 뒤 파라미터인 value와 비교하여 일치할 때마다 리턴값이 1씩 증가된다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains(const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | Bitmap and the index of start and cnt. value |
| **Return** | Return true if any bits in bitmap between start and start+cnt(exclusive) are set to value, and false otherwise. |
| **Function** | bitmap\_test로 비트의 값을 추출한 뒤 (start, start+cnt)구간 내에 파라미터인 value와 비교하여 한 개의 비트라도 일치하는 경우에는 false를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any(const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | Bitmap and the index of start and cnt. |
| **Return** | Return true if any bits in bitmap between start and start+cnt(exclusive) are set to value, and false otherwise. |
| **Function** | Bitmap\_contains의 value가 true인 경우로 고정된 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none(const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | Bitmap and the index of start and cnt. |
| **Return** | Return true if no bits in bitmap between start and start+cnt(exclusive) are set to value, and false otherwise. |
| **Function** | Bitmap\_contains를 부정 기능을 수행하는 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all(const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | Bitmap and the index of start and cnt. |
| **Return** | Return true if every bit in bitmap between start and start+cnt(exclusive) are set to value, and false otherwise. |
| **Function** | 한 비트라도 false라면(contain) false가 도출되고, 모두 false가 아니라면 true가 도출된다. 즉 논리적으로 value가 false인 bitmap\_contains의 부정기능을 수행하는 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan(const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | Bitmap and the index of start, cnt, and value |
| **Return** | Starting index of the first group of cnt consecutive bits in bitmap at or after start that are all set to value. |
| **Function** | Start부터 cnt만큼(bits를 벗어나지 않는 선에서) bitmap\_contains를 통하여 !value를 가지지 않는 구간을 탐색하여 해당 구간의 시작 인덱스를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip(const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | Bitmap and the index of start, cnt, and value |
| **Return** | Find the starting index of the first group of cnt consecutive bits in bitmap at or after start that are all set to value. And then flips them all to !value. If there is no such group, return error. If cnt is zero, return 0 |
| **Function** | Bitmap\_scan을 수행하여 탐색을 마친 뒤 해당 구간에 대하여 bitmap\_set\_multiple로 !value 처리한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size(const struct bitmap \*b) |
| **Parameter** | bitmap |
| **Return** | The number of bytes needed to store bitmap in a file |
| **Function** | 비트맵이 elem\_type으로 구성되어있음을 활용하여 elem\_type에 쓰이는 비트를 기준으로 비트맵에 소요되는 총 bits를 계산할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read(const struct bitmap \*b,struct file \*file) |
| **Parameter** | Bitmap, file |
| **Return** | Return true if reading file successes, false otherwise |
| **Function** | 비트맵에 총 소요되는 비트 크기를 size에 저장하여 해당 사이즈만큼 파일 읽기에 버퍼를 제공하여 일치하면 성공한 것이고, 그렇지 않다면 실패한 것이다. 그리고 elem으로 쓰이지 않는 비트를 비트맵의 last\_mask 함수 결과와 비트마스킹한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write(const struct bitmap \*b, struct file \*file) |
| **Parameter** | Bitmap, file |
| **Return** | Return true if writing file successes, false otherwise |
| **Function** | Reading과 마찬가지로 만약 write한 크기와 비트맵의 총 비트가 일치한 경우에 success고 이외에는 false로 쓰기 실패이다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump(const struct bitmap \*b) |
| **Parameter** | Bitmap |
| **Return** | None |
| **Function** | hex\_dump.c를 활용한 함수로 비트맵의 데이터를 16진수로 출력한다. hex\_dump (0,b->bits, byte\_cnt(b->bit\_cnt)/2, false)라고 호출하였다. 비트맵의 비트가 존재하는 버퍼를 입력받아 줄마다 최대 16개의 비트로 표현한다. 첫번째 비트부터 출력해야하므로 starting point를 알려주는 첫 번째 인수에 0을 전달한다. 그리고 따라서 버퍼의 크기를 출력할 비트 모두로 잡는 것이 아니라 효율적으로 데이터의 절반만 설정하도록 하였다. |