**Project #1 : MyLib**

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 교수님 |
| 학번 : | 20201614 |
| 이름 : | 이민석 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | struct list\_elem \*a : swap할 첫 번째 포인터  struct list\_elem \*b : swap할 첫 번째 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | list\_elem으로 이루어진 doubly linked list의 두 개 node의 위치를 바꾸는 기능을 한다. 세 가지 경우를 나누어 처리하며, 자세한 내용은 다음과 같다. case 1 : a의 다음 요소가 b일 경우, case 2 : a의 다음 요소가 b일 경우, case 3 : a와 b 사이에 두 개 이상의 요소가 있을 경우 |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | struct list \*list : 섞을 list의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 인자로 받은 포인터가 가리키는 list의 요소들을 무작위로 섞는다. List에 있는 모든 요소에 대해 랜덤으로 위치를 결정하여 해당위치에 있는 요소와 그 다음 요소와의 위치를 list\_swap()을 이용하여 교환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool less\_func\_list(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux) |
| **Parameter** | const struct list\_elem \*a : 비교할 첫 번째 요소의 포인터  const struct list\_elem \*b : 비교할 두 번째 요소의 포인터  void \*aux : 추가적인 비교에 사용될 매개변수 |
| **Return** | bool : ‘a’의 data가 ‘b’의 data보다 작으면 ‘true’, 그렇지 않으면 ‘false’를 반환한다. |
| **Function** | 두 개의 list 요소를 가리키는 ‘a’, ‘b’의 포인터를 인자로 받아 ‘list\_entry’ 매크로를 이용하여 list\_item 구조체로 변환한 후, data 값을 비교하여 ‘a’의 data가 ‘b’의 data보다 작은지의 여부를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2 (int i) |
| **Parameter** | int i: 해시 값을 생성할 정수 |
| **Return** | Unsigned : ‘i'의 해시 값 |
| **Function** | 입력받은 ‘i'를 주어진 규칙에 맞게 계산하여 해시 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool less\_func\_hash(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux) |
| **Parameter** | const struct hash\_elem \*a: 비교할 첫 번째 요소의 포인터  const struct hash\_elem \*b: 비교할 두 번째 요소의 포인터  void \*aux: 추가적인 비교에 사용될 매개변수 |
| **Return** | bool : ‘a’의 data가 ‘b’의 data보다 작으면 ‘true’, 그렇지 않으면 ‘false’를 반환한다. |
| **Function** | 두 개의 hash\_elem의 포인터를 인자로 받아, ‘a’의 data와 ‘b’의data를 비교한다. ‘a’의 data가 ‘b’의 data보다 작으면 ‘true’, 그렇지 않으면 ‘false’를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func (const struct hash\_elem \*e, void \*aux) |
| **Parameter** | const struct hash\_elem \*e: hash값을 생성할 포인터  void \*aux: 추가적인 비교에 사용될 매개변수 |
| **Return** | unsigned : hash\_int를 통해 생성된 해시 값 |
| **Function** | 인자로 받은 hash\_elem의 e->data의 해시 값을 hash\_int()함수를 이용하여 계산한다. 생성된 해시 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destructor (struct hash\_elem \*e, void \*aux) |
| **Parameter** | struct hash\_elem \*e: 제거할 요소의 포인터  void \*aux: 추가적인 비교에 사용될 매개변수 |
| **Return** | 반환 값 없음. |
| **Function** | free()를 통해 e을 free 처리한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | struct hash\_elem \*e: 데이터를 제곱할 요소의 포인터  void \*aux: 추가적인 비교에 사용될 매개변수 |
| **Return** | 반환 값 없음. |
| **Function** | hash\_elem인 e->data를 제곱하여 이를 다시 e->data에 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | struct hash\_elem \*e: 데이터를 세제곱할 요소의 포인터  void \*aux: 추가적인 비교에 사용될 매개변수 |
| **Return** | 반환 값 없음. |
| **Function** | hash\_elem인 e->data를 세제곱하여 이를 다시 e->data에 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap,int size) |
| **Parameter** | struct bitmap \*bitmap: 크기를 늘릴 비트맵의 포인터  int size: 추가로 늘릴 비트맵의 크기 |
| **Return** | struct bitmap \*: 새로 늘린 비트맵의 포인터 |
| **Function** | 비트맵 ‘bitmap’의 크기를 기존 크기에 ‘size’만큼 더 늘리는 역할을 하며, 늘린 비트들은 모두 0으로 설정한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | list\_init (struct list \*list) |
| **Parameter** | struct list \*list : 초기화할 list의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 인자로 받는 list의 구조체를 초기화 한다. 초기 구조인 head와 tail을 next, prev 포인터를 이용하여 연결하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*list) |
| **Parameter** | struct list \*list : 찾고자 하는 시작 요소가 담긴 list의 포인터 |
| **Return** | struct list\_elem \*: list의 첫 번째 요소(헤드 다음)의 포인터 |
| **Function** | head->next의 포인터를 반환하여, 인자로 받는 list의 head 다음 요소의 포인터를 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next(struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem : 다음 요소를 얻기 위한 현재 list 요소의 포인터 |
| **Return** | struct list\_elem \*: 주어진 요소의 다음 요소를 가리키는 포인터 |
| **Function** | 인자로 받는 요소의 다음을 가리키는 포인터를 반환하는 기능을 수행하며, 만약 list의 마지막 요소라면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*list) |
| **Parameter** | struct list \*list : 크기를 측정할 list의 포인터 |
| **Return** | size\_t : list 내 요소의 개수(list의 사이즈)를 나타내는 값 |
| **Function** | list를 순회하며 요소의 개수를 센다. list\_begin(), list\_end()를 사용하여 첫 번째 요소부터 마지막 요소까지 순회하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back(struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list: 요소를 추가하는 list의 포인터  struct list\_elem \*elem: list에 추가할 새로운 요소의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | list\_end()함수를 사용하여 list의 마지막 요소 정보를 얻게 되고, 그 뒤에 list\_insert()를 사용하여 elem을 list에 추가하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list: 요소를 추가하는 list의 포인터  struct list\_elem \*elem: 리스트에 추가할 새로운 요소의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | list\_begin()함수를 사용하여 list의 가장 앞 요소 정보를 얻게 되고, 그 앞에 list\_insert()를 사용하여 elem을 list에 추가하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*list) |
| **Parameter** | struct list \*list: 첫 번째 요소를 검색할 list의 포인터 |
| **Return** | struct list\_elem \*: list의 첫 번째 요소의 포인터 |
| **Function** | list\_empty()를 이용하여 list가 비어있지 않은 경우, list의 첫번째 요소의 포인터를 반환한다. Head의 다음 요소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*list) |
| **Parameter** | struct list \*list: 마지막 요소를 검색할 list의 포인터 |
| **Return** | struct list\_elem \*: list의 마지막 요소의 포인터 |
| **Function** | list\_empty()를 이용하여 list가 비어있지 않은 경우, list의 마지막 요소의 포인터를 반환한다. tail의 이전 요소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*list) |
| **Parameter** | struct list \*list: 첫 번째 요소를 제거할 list의 포인터 |
| **Return** | struct list\_elem \*: 제거된 첫 번째 요소의 포인터 |
| **Function** | list\_front()함수를 사용하여 list의 첫번째 요소를 찾은 후, list\_remove()함수를 사용하여 해당 요소를 list에서 제거한다. 제거된 요소의 포인터를 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*list) |
| **Parameter** | struct list \*list: 마지막 요소를 제거할 list의 포인터 |
| **Return** | struct list\_elem \*: 제거된 마지막 요소의 포인터 |
| **Function** | list\_back()함수를 사용하여 list의 마지막 요소를 찾은 후, list\_remove()함수를 사용하여 해당 요소를 list에서 제거한다. 제거된 요소의 포인터를 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*before: 삽입할 위치의 다음에 오는 요소의 포인터  struct list\_elem \*elem: before 이전 위치에 삽입되는 새로운 요소를 가리키는 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | Assert를 통해 에러가 나는 경우인지 검증한 후, elem의 요소를 before의 앞에 삽입하게 된다. 이 때 적절히 elem, before의 prev, next를 연결한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem,  list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list: 새 요소를 삽입할 list의 포인터  struct list\_elem \*elem: list에 삽입할 새로운 요소의 포인터  list\_less\_func \*less: 두 list 요소를 비교하는 함수의 포인터void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | 반환 값 없음 |
| **Function** | list의 처음부터 탐색하여, elem이 삽입될 위치를 찾게 된다. for문을 사용하여 위치를 찾은 후, 적절한 위치에 list\_insert()를 사용하여 elem을 추가하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty(struct list \*list) |
| **Parameter** | struct list \*list: 비어 있는지 확인할 list의 포인터 |
| **Return** | bool: list가 비어 있으면 true, 그렇지 않으면 false를 반환한다. |
| **Function** | list\_begin()의 반환 값과 list\_end()의 반환 값을 비교하여 같으면 ‘true’, 다르면 ‘false’를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max(struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list: 최댓값을 찾을 list의 포인터  list\_less\_func \*less: 두 list 요소를 비교하는 함수 포인터  void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | struct list\_elem \*: list 내 최댓값을 가진 요소의 포인터 |
| **Function** | list의 첫 번째 요소 data 값으로 max을 초기화 한 후, list를 순회하며, 각 요소들을 max값과 비교한다. 이 때 less\_func을 사용하며 max값을 최신화한다. 리스트의 마지막 요소까지 비교한 후, max 값을 가진 요소의 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min(struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list: 최솟값을 찾을 list의 포인터  list\_less\_func \*less: 두 list 요소를 비교하는 함수 포인터  void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | struct list\_elem \*: list 내 최솟값을 가진 요소의 포인터 |
| **Function** | list의 첫 번째 요소 data 값으로 min을 초기화 한 후, list를 순회하며, 각 요소들을 min값과 비교한다. 이 때 less\_func을 사용하며 min값을 최신화한다. list 의 마지막 요소까지 비교한 후, min 값을 가진 요소의 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem: list 에서 제거할 요소의 포인터 |
| **Return** | struct list\_elem \*: 제거된 요소의 다음 요소의 포인터 |
| **Function** | elem의 이전 요소를 elem의 다음 요소와 연결하여 elem을 list 에서 제거하게 된다. 제거한 요소의 다음 요소 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse(struct list \*list) |
| **Parameter** | struct list \*list: 순서를 반대로 바꿀 list 의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | list가 비어 있지 않은 지 확인한 후, list의 첫번째 요소부터 마지막 요소까지 각 요소들의 prev와 next를 swap()을 통해 바꾸게 된다. 마지막으로 head와 tail 또한 적절히 수정한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list: 정렬할 list의 포인터  list\_less\_func \*less: 두 list 요소를 비교하는 함수 포인터  void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | 반환 값 없음. |
| **Function** | Divide & Conquer 알고리즘을 이용하여 list 를 반복하여 두 부분으로 나누고 각 부분을 정렬한다. 다시 병합하여 전체가 정렬되게끔 한다. 이 때 find\_end\_of\_run()을 사용하여 두 개의 연속적으로 증가하는 sequence들을 찾고, inplace\_merge()를 사용하여 병합한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice(struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | struct list\_elem \*before: 삽입할 위치의 바로 앞 요소의 포인터  struct list\_elem \*first: 이동할 요소들의 시작을 알려주는 요소의 포인터  struct list\_elem \*last: 이동할 요소들의 마지막의 다음 요소 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | First와 last 사이의 요소들을 제거한 후, before 바로 앞에 삽입한다. 이 때 before는 list의 내부 요소나 tail이 될 수 있다. 리스트의 특정 부분을 다른 위치로 이동하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates,  list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list: 중복 요소를 제거할 대상 list의 포인터  struct list \*duplicates: 중복된 요소를 저장할 list의 포인터  list\_less\_func \*less: 두 리스트 요소를 비교하는 함수 포인터  void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | 반환 값 없음 |
| **Function** | list를 순회하며 각 인접한 요소들 중 동일한 요소들을 찾아내어 첫 번째 요소를 제외한 나머지 동일한 요소들을 제거한다. 이 때 duplicates가 NULL이 아니라면 제거된 요소들은 duplicates에 저장된다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | Bool hash\_init(struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | struct hash \*h: 초기화할 해시 테이블의 포인터  hash\_hash\_func \*hash: 해시 테이블에 사용할 해시 함수  hash\_less\_func \*less: 해시 테이블 내 요소들을 비교하는 함수 포인터  void \*aux: less\_func 함수에 전달될 추가적인 매개변수 |
| **Return** | bool: 초기화 과정이 제대로 작동하면 ‘true’, 그렇지 않다면 ‘false’를 반환한다. |
| **Function** | 해시 테이블 h의 elem\_cnt를 0, bucket\_cnt를 4으로 초기화 하며malloc을 사용하여 버킷 배열에 대한 메모리를 할당한다. 이 때 할당된 크기는 bucket\_cnt에 의해 결정된다. 해시 함수, 비교 함수, 그리고 추가적인 데이터(aux)를 해시 테이블 구조체에 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first(struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | struct hash\_iterator \*i: 초기화할 해시 테이블 iterator의 포인터  struct hash \*h: 순회할 해시 테이블의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 해시 테이블 iterator i를 순회할 해시 테이블, 버킷 배열의 첫번째 버킷, 리스트 요소들로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*i) |
| **Parameter** | struct hash\_iterator \*i: 현재 위치의 해시 테이블의 iterator 포인터 |
| **Return** | struct hash\_elem \*: 해시 테이블 내의 다음 요소의 포인터 |
| **Function** | Iterator i의 현재 요소를 현재 버킷의 다음 요소로 업데이트 하며 이 때 list\_next()를 사용한다. 만약 현재 버킷 내에 순회할 요소가 없다면 다음 버킷으로 이동한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*i) |
| **Parameter** | struct hash\_iterator \*i: 현재 위치의 해시 테이블의 iterator 포인터 |
| **Return** | struct hash\_elem \*: 현재 위치의 해시 테이블의 iterator가 가리키는 요소의 포인터 |
| **Function** | 현재 해시 요소의 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h: 소멸할 해시 테이블의 포인터  hash\_action\_func \*destructor: 해시 테이블의 각 요소를 제거하는 소멸자 함수 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 사전에 정의된 hash\_destructor()를 이용하여 해시 테이블의 모든 요소를 제거한다. 또한 할당된 버킷 배열의 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h: 요소를 삽입할 해시 테이블의 포인터  struct hash\_elem \*new: 새로 삽입할 hash\_elem 포인터 |
| **Return** | 해시 테이블 내에 삽입되는 요소와 동일한 key를 가진 요소가 있다면 해당 기존 요소의 포인터를 반환, 그렇지 않다면 NULL을 반환 |
| **Function** | find\_bucket()을 이용하여 new 요소를 적절한 버킷에 삽입하게 된다. 만약 기존 요소와 겹친다면 old 변수에 저장되며, 그렇지 않다면 insert\_elem()을 통해 해당 버킷에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | struct hash \*h: hash\_action\_func을 진행할 해시 테이블의 포인터  hash\_action\_func \*action: 해시 테이블의 각 요소에 대해 수행할 정보를 담고 있는 함수 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 해시 테이블의 모든 버킷을 순회하며, 버킷 내 모든 리스트 요소들을 순회한다. 모든 요소들을 hash\_action\_func 인자로 넘겨 주어 목표하는 기능을 수행하도록 한다.. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h: 요소를 삭제할 해시 테이블의 포인터  struct hash\_elem \*e: 삭제하고자 하는 요소의 포인터 |
| **Return** | Struct hash\_elem\*: 삭제된 요소의 포인터 |
| **Function** | find\_elem()을 사용하여 해시 테이블 내, 삭제하고자 하는 요소 e를 찾는다. 이후 삭제하려는 요소를 remove\_elem()을 통해 해시 테이블에서 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | Bool hash\_empty (struct hash \*h) |
| **Parameter** | struct hash \*h: 비어있는지 확인하려는 해시 테이블의 포인터 |
| **Return** | Bool: 만약 h가 비어있다면 ‘true’, 그렇지 않다면 ‘false’를 반환 |
| **Function** | 해시 테이블 내 요소의 총 개수를 나타내는 ‘h’의 elem\_cnt 값을 확인하여 해시 테이블의 비어있는 여부를 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*h) |
| **Parameter** | struct hash \*h: 사이즈를 확인하려는 해시 테이블의 포인터 |
| **Return** | size\_t: ‘h’에 저장된 요소의 총 개수 |
| **Function** | 해시 테이블 내 요소의 총 개수를 나타내는 ‘h’의 elem\_cnt 값을 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h: 초기화할 해시 테이블의 포인터  hash\_action\_func \*destructor: 해시 테이블의 각 요소를 제거하는 소멸자 함수 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | ‘h’의 모든 버킷을 순회하며, 버킷 내 모든 요소들을 destructor를 통해 제거한다. 또한 list\_init()을 통해 버킷들을 초기화 한 후, elem\_cnt = 0으로 초기화 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h: new가 삽입될 해시 테이블 포인터  struct hash\_elem \*new: 삽입하는 요소의 포인터 |
| **Return** | struct hash\_elem \*: 새롭게 삽입된 요소에 의해 대체된 요소의 포인터 |
| **Function** | ‘new'를 ‘h’에 삽입하게 되며, 이때 삽입되는 요소와 동일한 key를 가진 요소가 있다면 그것을 대체하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h: 찾고자 하는 요소를 담고있는 해시 테이블 포인터  struct hash\_elem \*e: 찾고자 하는 요소의 정보를 담고있는 포인터 |
| **Return** | struct hash\_elem \*: 찾고자 하는 요소와 동일한 요소의 포인터 |
| **Function** | Find\_elem()을 통해 찾고자 하는 요소와 동일한 정보를 담고 있는 요소를 찾게 되며, 만약 없다면 NULL을 반환한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | size\_t bit\_cnt: 생성할 비트맵의 비트 수 크기 |
| **Return** | struct bitmap \*: 생성된 비트맵의 포인터 |
| **Function** | struct bitmap 구조체에 대한 메모리를 동적 할당하며, 비트를 저장할 배열에 대해 메모리를 할당한다. 이를 성공하거나, bit\_춧 == 0 일 경우, bitmap\_set\_all()을 통해 비트맵의 모든 비트를 ‘false’로 초기화 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | const struct bitmap \*b: 사이즈를 확인하려는 비트맵의 포인터 |
| **Return** | size\_t: 비트맵 사이즈 |
| **Function** | 비트맵 크기의 정보를 담고있는 비트맵 구조체의 bit\_cnt를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | Bool bitmap\_test (const struct bitmap \*b, size\_t idx) |
| **Parameter** | const struct bitmap \*b: 살펴볼 비트맵의 포인터  size\_t idx: ‘true’ 또는 ‘false’인지 살펴볼 비트의 인덱스 |
| **Return** | Bool: idx에 위치한 비트가 1이면 ‘true’ 그렇지 않으면 ‘false’ |
| **Function** | Assert 매크로를 사용하여 ‘b’와 ‘idx’가 유효한지 확인한 후, bit\_mask()를 이용하여 idx에 위치한 비트의 마스크를 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | struct bitmap \*b: 제거할 비트맵의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 비트맵 ‘b’를 제거하며, free()를 통해 b와 b->bits에 할당된 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b: 비트를 set할 비트맵의 포인터  size\_t bit\_idx: set할 비트의 인덱스 |
| **Return** | 반환 값 없음. |
| **Function** | 비트맵 ‘b’의 ‘bit\_idx’에 위치하는 비트를 1(=’true’)로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | Bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b: 모든 비트가 1인지 확인할 비트맵 포인터  size\_t start: 확인할 비트맵의 시작 bit 인덱스  size\_t cnt: start로부터 확인할 비트의 수 |
| **Return** | Bool: ‘start’~’start+cnt'까지 bit가 모두 1이면 true, 그렇지 않으면 ‘false’를 반환 |
| **Function** | Bitmap\_contains()를 사용하여 ‘start’~’start+cnt'까지 모든 비트들이 set(‘1’)인지, 그렇지 않은지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | Bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b: 확인할 비트맵의 포인터  size\_t start: 확인할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수 |
| **Return** | Bool: ‘start’~’start+cnt'까지 bit가 하나이상 1이면 true, 그렇지 않으면 ‘false’를 반환 |
| **Function** | Bitmap\_contains()를 사용하여 ‘start’~’start+cnt'까지 최소 한 개 이상의 비트가 set(‘1’)인지, 그렇지 않은지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | Bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b: 확인할 비트맵의 포인터  size\_t start: 확인할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수  bool value: 검사할 비트 값의 Boolean 값 |
| **Return** | Bool: ‘start’~’start+cnt'까지 bit가 하나이상 ‘value’와 일치하면 true, 그렇지 않으면 ‘false’를 반환 |
| **Function** | Bitmap\_test()을 이용하여 ‘start’~’start+cnt' 범위 내에 ‘value’와 값이 일치하는 bit가 있는지 여부를 출력하는 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b: 계산할 비트맵의 포인터  size\_t start: 계산할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수  bool value: 계산에 포함할 비트의 Boolean 값 |
| **Return** | size\_t: ‘start’~’start+cnt'까지 ‘value’와 일치하는 bit의 개수 |
| **Function** | Bitmap\_test()함수를 이용하여 ‘start’~’start+cnt' 범위 내에 ‘value’와 값이 일치하는 bit을 찾은 후, 그것들의 개수를 세어 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*b) |
| **Parameter** | const struct bitmap \*b: 내용을 출력할 비트맵의 포인터 |
| **Return** | 반환 값 없음. |
| **Function** | 비트맵 ‘b’의 값들을 16진수 형태로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*b, bool value) |
| **Parameter** | struct bitmap \*b: set\_all을 적용할 비트맵의 포인터  bool value: 모든 비트에 적용할 Boolean 값 |
| **Return** | 반환 값 없음 |
| **Function** | Bitmap\_set\_multiple()를 이용하여 비트맵 ‘b’의 값들을 ‘value’의 값과 동일하게 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b: 비트를 반전시킬 비트맵의 포인터  size\_t bit\_idx: 반전시킬 비트의 인덱스 |
| **Return** | 반환 값 없음 |
| **Function** | Bitmap\_mask()를 이용하여 bit\_idx의 비트 마스크를 계산한 후, Boolean expression을 하여 해당 인덱스의 비트를 반전시킨다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b: 확인할 비트맵의 포인터  size\_t start: 확인할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수 |
| **Return** | Bool: ‘start’~’start+cnt'까지 bit가 모두 reset(‘0’)이면 ‘true’, 그렇지 않으면 ‘false’를 반환 |
| **Function** | Bitmap\_contains()를 사용하여 ‘start’~’start+cnt'까지 모든 비트들이 reset(‘0’)인지, 그렇지 않은지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b: reset할 비트를 포함한 비트맵의 포인터  size\_t bit\_idx: reset할 비트의 인덱스 |
| **Return** | 반환 값 없음 |
| **Function** | 비트맵 ‘b’의 인덱스 ‘bit\_idx’에 위치한 비트의 값을 ‘false’(‘0’으로 설정하는 기능을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip(const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b: 검색할 비트맵의 포인터  size\_t start: 검색할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수  bool value: 찾고자 하는 비트의 Boolean 값 |
| **Return** | size\_t: 찾은 부분 비트 그룹의 첫번째 bit 인덱스를 반환 |
| **Function** | 비트맵 ‘b’에서 ‘value’와 값이 같은 연속된 부분 비트 그룹을 찾고, 그 비트들의 값을 반전시킨다. 만약 조건을 만족하는 그룹이 없다면, BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b: 검색할 비트맵의 포인터  size\_t start: 검색할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 확인할 비트의 수  bool value: 찾고자 하는 비트의 Boolean 값 |
| **Return** | size\_t: 찾은 부분 비트 그룹의 첫번째 bit 인덱스를 반환 |
| **Function** | 비트맵 ‘b’에서 ‘value’와 값이 같은 연속된 cnt개의 비트 그룹을 찾는다. 만약 조건을 만족하는 그룹이 없다면, BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple(const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b: set/reset 설정할 비트맵의 포인터  size\_t start: set/reset 설정할 비트맵의 비트 시작 인덱스  size\_t cnt: start로부터 set /reset 설정할 비트의 수  bool value: 모든 비트에 적용할 Boolean 값 |
| **Return** | 반환 값 없음. |
| **Function** | 비트맵 ‘b’내에서 ‘start’~’start+cnt'까지 모든 비트들을 ‘value’와같은 값으로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t bit\_idx, bool value) |
| **Parameter** | struct bitmap \*b: set/reset 설정할 비트맵의 포인터  size\_t bit\_idx: set/reset 설정할 비트맵의 비트 인덱스  bool value: 비트에 적용할 Boolean 값 |
| **Return** | 반환 값 없음 |
| **Function** | 비트맵 ‘b’내에서 ‘bit\_idx’에 위치한 bit을 ‘value’와 같은 값으로 설정한다. |