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

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 |
| 학번 : | 20181202 |
| 이름 : | 김수미 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.  
( 파란색 : 직접 작성한 함수 / 회색 : 제공된 코드의 함수 )**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | int main( ) |
| **Parameter** | NONE |
| **Return** | NONE |
| **Function** | 사용자로부터 input값을 받아 적절한 함수를 실행시키고, ‘quit’를 입력 받을 때 까지 이를 반복한다. 모든 입력은 standard input으로 받는다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\_node |
| **Parameter** | NONE (함수가 아닌 구조체이다) |
| **Return** | NONE (함수가 아닌 구조체이다) |
| **Function** | Bitmap을 여러 개 생성하는 경우, 이를 연결 리스트 형태로 관리하기 위한 구조체이다. 각 Bitmap의 이름(char title[10]) 및 다음 Bitmap을 가리키는 포인터, Bitmap 자료구조를 멤버로 가진다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_node |
| **Parameter** | NONE (함수가 아닌 구조체이다) |
| **Return** | NONE (함수가 아닌 구조체이다) |
| **Function** | Hashtable을 여러 개 생성하는 경우, 이를 연결 리스트 형태로 관리하기 위한 구조체이다. 각 Hashtable의 이름(char title[10]) 및 다음 Hashtable을 가리키는 포인터, Hashtable 자료구조를 멤버로 가진다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_node |
| **Parameter** | NONE (함수가 아닌 구조체이다) |
| **Return** | NONE (함수가 아닌 구조체이다) |
| **Function** | List을 여러 개 생성하는 경우, 이를 연결 리스트 형태로 관리하기 위한 구조체이다. 각 List의 이름(char title[10]) 및 다음 List를 가리키는 포인터, List 자료구조를 멤버로 가진다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 6글자가 ‘create’ 인 경우 실행되는 함수이다. ‘create bitmap <title>’ , ‘create hashtable <title>’ , ‘create list <title>’의 총 세가지 경우가 존재하므로 input string을 전달받아 create 뒤에 나오는 문자열을 읽어 해당 <title>을 가지는 bitmap/hashtable/list 중 하나를 생성 및 초기화하는 과정을 수행한다 |

|  |  |
| --- | --- |
| **Prototype** | void dumpdata\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 8글자가 ‘dumpdata’ 인 경우 실행되는 함수이다. ‘dumpdata <title>’의 한가지 경우만이 존재하지만 <title>의 제목을 가지는 자료구조가 bitmap 인지, hashtable인지, list인지 모르기 때문에 현재 존재하는 자료구조들 사이에서 <title>의 제목을 가지는 자료구조를 찾고, 그것이 bitmap 인지, hashtable인지, list인지에 따라서 그 안에 저장되어 있는 값들을 출력하는 알맞은 과정을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void delete\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 6글자가 ‘delete’ 인 경우 실행되는 함수이다. ‘delete <title>’의 한가지 경우만이 존재하지만 <title>의 제목을 가지는 자료구조가 bitmap 인지, hashtable인지, list인지 모르기 때문에 현재 존재하는 자료구조들 사이에서 <title>의 제목을 가지는 자료구조를 찾고, 그것이 bitmap 인지, hashtable인지, list인지에 따라서 그 안에 저장되어 있는 값들을 destroy 및 free 시키는 알맞은 과정을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void my\_bitmap\_dump(struct bitmap \*bitmap) |
| **Parameter** | dumpdata의 대상이 되는 bitmap |
| **Return** | NONE |
| **Function** | Bitmap 안에 저장되어 있는 데이터들을 출력하는 함수이다. void dumpdata\_cmd(char\*) 함수 내부에서 호출되며 Parameter로 받은 bitmap 내부에 저장된 값을 print 한다. |

|  |  |
| --- | --- |
| **Prototype** | void my\_hash\_dump(struct hash \*hash) |
| **Parameter** | dumpdata의 대상이 되는 hashtable |
| **Return** | NONE |
| **Function** | Hashtable 안에 저장되어 있는 데이터들을 출력하는 함수이다. void dumpdata\_cmd(char\*) 함수 내부에서 호출되며 Parameter로 받은 hashtable 내부에 저장된 값을 print 한다. |

|  |  |
| --- | --- |
| **Prototype** | void my\_list\_dump(struct list \*list) |
| **Parameter** | dumpdata의 대상이 되는 list |
| **Return** | NONE |
| **Function** | List 안에 저장되어 있는 데이터들을 출력하는 함수이다. void dumpdata\_cmd(char\*) 함수 내부에서 호출되며 Parameter로 받은 List 내부에 저장된 값을 print 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 6글자가 ‘bitmap’ 인 경우 실행되는 함수이다. Bitmap과 관련된 작업을 수행하는 다양한 함수들을 처리하는 함수이며, 총 17가지의 Bitmap 관련 함수를 수행한다. (bitmap\_mark, bitmap\_all, bitmap\_any, bitmap\_contains  , bitmap\_count, bitmap\_dump, bitmap\_expand, bitmap\_set  , bitmap\_set\_all, bitmap\_set\_multiple, bitmap\_flip, bitmap\_none  , bitmap\_reset, bitmap\_scan\_and\_flip, bitmap\_scan, bitmap\_size  , bitmap\_test) |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\_node \*my\_bitmap\_find(char \*name) |
| **Parameter** | 특정 bitmap의 title string(char array) |
| **Return** | bitmap\_node 구조체 포인터 |
| **Function** | 특정 bitmap의 title을 사용자가 입력했을 때 이에 해당하는 bitmap\_node를 bitmap 연결 리스트에서 찾아 포인터를 반환한다. 포인터 변수를 가지고 해당 bitmap에 대한 작업을 수행할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 4글자가 ‘hash’ 인 경우 실행되는 함수이다. Hashtable과 관련된 작업을 수행하는 다양한 함수들을 처리하는 함수이며, 총 17가지의 Hashtable 관련 함수를 수행한다. (hash\_insert, hash\_apply, hash\_delete, hash\_empty  , hash\_size, hash\_clear, hash\_find, hash\_replace) |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_node \*my\_hash\_find(char \*name) |
| **Parameter** | 특정 hashtable의 title string(char array) |
| **Return** | hash\_node 구조체 포인터 |
| **Function** | 특정 hashtable의 title을 사용자가 입력했을 때 이에 해당하는 hash\_node를 hashtable 연결 리스트에서 찾아 포인터를 반환한다. 포인터 변수를 가지고 해당 hashtable에 대한 작업을 수행할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_cmd(char \*input) |
| **Parameter** | 사용자가 입력한 input string(char array) |
| **Return** | NONE |
| **Function** | 사용자가 입력한 input string의 첫 4글자가 ‘list’ 인 경우 실행되는 함수이다. List와 관련된 작업을 수행하는 다양한 함수들을 처리하는 함수이며, 총 19가지의 List 관련 함수를 수행한다. (list\_front, list\_back, list\_pop\_front, list\_pop\_back, list\_push\_front  , list\_push\_back, list\_insert\_ordered, list\_insert, list\_empty, list\_size  , list\_max, list\_min, list\_remove, list\_reverse, list\_shuffle, list\_sort  , list\_splice, list\_swap, list\_unique) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_node \*my\_list\_find(char\* name) |
| **Parameter** | 특정 list의 title string(char array) |
| **Return** | list\_node 구조체 포인터 |
| **Function** | 특정 list 의 title을 사용자가 입력했을 때 이에 해당하는 list \_node를 list 연결 리스트에서 찾아 포인터를 반환한다. 포인터 변수를 가지고 해당 list 에 대한 작업을 수행할 수 있다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | 서로 자리를 바꿀 두 개의 list elements |
| **Return** | NONE |
| **Function** | Paremeter로 받은 두 개의 list element의 자리를 서로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | Shuffle(무작위로 섞음)의 대상이 되는 list |
| **Return** | NONE |
| **Function** | Parameter로 받은 list의 elements들의 순서를 무작위로 섞는다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux) |
| **Parameter** | list의 prev, next 포인터를 저장하고 있는 list\_elem 구조체 두 개. Void 타입 변수 한개 역시 parameter로 받지만 해당 함수 내에서 사용되지 않는다(없으면 에러 발생하여 포함시킴). |
| **Return** | a의 data값이 b의 data값 보다 작으면 true(Boolean type)를, 크거나 같으면 false(Boolean type)를 return한다. |
| **Function** | parameter로 받은 두 개의 list\_elem 자료형 변수를 포함하는 list\_item을 찾아 각 data값을 비교하여 a가 b보다 작은 값을 가지고 있는지를 체크해 결과값을 return 한다. / List와 관련된 작업을 수행하는 여러 함수들 중 list\_less 함수를 parameter로 요구하는 함수들이 있다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | NONE |
| **Function** | Initializes LIST as an empty list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the beginning of LIST. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | True/False |
| **Function** | Returns the front element in LIST. Undefined behavior if LIST is empty. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Removes the front element from LIST and returns it. Undefined behavior if LIST is empty before removal. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Removes the back element from LIST and returns it. Undefined behavior if LIST is empty before removal. |
| **Prototype** | struct list\_elem \*list\_front (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the front element in LIST. Undefined behavior if LIST is empty. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the back element in LIST. Undefined behavior if LIST is empty. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | size\_t variable |
| **Function** | Returns the number of elements in LIST. Runs in O(n) in the number of elements. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | True/False |
| **Function** | Returns true if LIST is empty, false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | NONE |
| **Function** | Reverses the order of LIST. |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | NONE |
| **Function** | Sorts LIST according to LESS given auxiliary data AUX, using a natural iterative merge sort that runs in O(n lg n) time and O(1) space in the number of elements in LIST. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux |
| **Return** | NONE |
| **Function** | Inserts ELEM in the proper position in LIST, which must be sorted according to LESS given auxiliary data AUX. Runs in O(n) average case in the number of elements in LIST. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates,list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, struct list \*duplicates,list\_less\_func \*less, void \*aux |
| **Return** | NONE |
| **Function** | Iterates through LIST and removes all but the first in each set of adjacent elements that are equal according to LESS given auxiliary data AUX. If DUPLICATES is non-null, then the elements from LIST are appended to DUPLICATES. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the element in LIST with the largest value according to LESS given auxiliary data AUX. If there is more than one maximum, returns the one that appears earlier in the list. If the list is empty, returns its tail. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the element in LIST with the smallest value according to LESS given auxiliary data AUX. If there is more than one minimum, returns the one that appears earlier in the list. If the list is empty, returns its tail. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* variable |
| **Function** | Return's LIST's head. list\_head() can be used for an alternate style of iterating through a list. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last |
| **Return** | NONE |
| **Function** | Removes elements FIRST though LAST (exclusive) from their current list, then inserts them just before BEFORE, which may be either an interior element or a tail. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem\* variable |
| **Function** | Removes ELEM from its list and returns the element that followed it. Undefined behavior if ELEM is not in a list. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | NONE |
| **Function** | Inserts ELEM at the beginning of LIST, so that it becomes the front in LIST. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | NONE |
| **Function** | Inserts ELEM at the end of LIST, so that it becomes the back in LIST. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*elem |
| **Return** | NONE |
| **Function** | Inserts ELEM just before BEFORE, which may be either an interior element or a tail. The latter case is equivalent to list\_push\_back(). |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem\* variable |
| **Function** | Returns the element after ELEM in its list. If ELEM is the last element in its list, returns the list tail. Results are undefined if ELEM is itself a list tail. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | struct my\_hash\_elem |
| **Parameter** | NONE (함수가 아닌 구조체이다) |
| **Return** | NONE (함수가 아닌 구조체이다) |
| **Function** | hash\_elem 구조체 변수와 정수형 데이터값을 멤버로 가지는 구조체이다. Pintos 자료구조의 특징 때문에 hash element 와 데이터 값을 동시에 저장하는 구조체 자료형을 선언해 주었다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int); |
| **Parameter** | Hash의 대상이 되는 integer(정수형 변수) |
| **Return** | Parameter로 받은 integer의 hash value |
| **Function** | Hashtable을 관리함에 있어서, key값이 주어졌을 때 key 값이 저장되어있는 bucket의 index를 찾을 때 사용되는 함수이다. 강의자료를 참고하여 간단하게 Key 값이 x 일 때 bucket의 index number는 x를 100으로 나눈 몫을 bucket의 index로 반환할 수 있도록 함수를 작성했다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned my\_hash\_function(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | data값을 구하고자 하는 hash\_elem 자료형 변수와 void 자료형 변수. Void 자료형 변수는 parameter로 받지만 해당 함수 내에서 사용되지 않는다. |
| **Return** | Parameter로 받은 hash\_elem 자료형 변수의 data 값을 return. |
| **Function** | parameter로 받은 hash\_elem 자료형 변수를 hash\_entry 함수를 사용하여 my\_hash\_elem 구조체 자료형으로 변환한 뒤 hash\_int 함수를 사용하여 hash\_elem 자료형 변수의 hash된 데이터 값을 return 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less (const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux); |
| **Parameter** | hash\_elem 구조체 타입 변수 두 개. Void 자료형 변수 aux는 parameter로 받지만 함수 내에서 사용되지 않는다. |
| **Return** | a의 data값이 b의 data값 보다 작으면 true(Boolean type)를, 크거나 같으면 false(Boolean type)를 return한다. |
| **Function** | parameter로 받은 두 개의 hash\_elem 자료형 변수의 data값을 비교하여 a의 데이터값이 b의 데이터 값 보다 작은지를 체크하고, 그 결과값을 return 한다. / Hashtable과 관련된 작업을 수행하는 여러 함수들 중 hash\_less함수를 parameter로 요구하는 함수들이 있다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_free (struct hash\_elem \*e, void \*aux) |
| **Parameter** | Free시키고자 하는 hash\_elem 자료형 변수. Void 자료형 변수 aux는 parameter로 받지만 함수 내에서 사용되지 않는다. |
| **Return** | NONE |
| **Function** | parameter로 받은 hash\_elem 자료형 변수를 멤버로 가지는 구조체 변수를 찾아 할당된 구조체 변수를 free 시킨다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash\_elem 자료형 변수. Void 자료형 변수 aux는 parameter로 받지만 함수 내에서 사용되지 않는다. |
| **Return** | NONE |
| **Function** | hash\_apply <title> square를 사용자가 입력했을 때 이를 처리하기 위해 사용되는 함수이다. parameter로 받은 hash\_elem 자료형 변수를 hash\_entry를 사용하여 my\_hash\_elem 구조체 타입으로 변환한 뒤 해당 구조체 변수의 data값을 제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash\_elem 자료형 변수. Void 자료형 변수 aux는 parameter로 받지만 함수 내에서 사용되지 않는다. |
| **Return** | NONE |
| **Function** | hash\_apply <title> triple을 사용자가 입력했을 때 이를 처리하기 위해 사용되는 함수이다. parameter로 받은 hash\_elem 자료형 변수를 hash\_entry를 사용하여 my\_hash\_elem 구조체 타입으로 변환한 뒤 해당 구조체 변수의 data값을 세제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*h, hash\_hash\_func \*hash  , hash\_less\_func \*less, void \*aux) |
| **Parameter** | struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux |
| **Return** | True/False |
| **Function** | Initializes hash table H to compute hash values using HASH and compare hash elements using LESS, given auxiliary data AUX. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | struct hash \*h |
| **Return** | True/False |
| **Function** | Returns true if H contains no elements, false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | struct hash\_iterator \*i, struct hash \*h |
| **Return** | NONE |
| **Function** | Initializes I for iterating hash table H. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*i) |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | struct hash\_elem\* variable |
| **Function** | Advances I to the next element in the hash table and returns it. Returns a null pointer if no elements are left. Elements are returned in arbitrary order. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*i) |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | struct hash\_elem\* variable |
| **Function** | Returns the current element in the hash table iteration, or a null pointer at the end of the table. Undefined behavior after calling hash\_first() but before hash\_next(). |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | struct hash\_elem\* variable |
| **Function** | Inserts NEW into hash table H and returns a null pointer, if no equal element is already in the table. If an equal element is already in the table, returns it without inserting NEW. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | struct hash \*h, hash\_action\_func \*action |
| **Return** | NONE |
| **Function** | Calls ACTION for each element in hash table H in arbitrary order. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | struct hash\_elem\* variable |
| **Function** | Finds, removes, and returns an element equal to E in hash table H. Returns a null pointer if no equal element existed in the table. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | struct hash \*h |
| **Return** | True/False |
| **Function** | Returns true if H contains no elements, false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*h) |
| **Parameter** | struct hash \*h |
| **Return** | size\_t variable |
| **Function** | Returns the number of elements in H. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | NONE |
| **Function** | Removes all the elements from H. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | Finds and returns an element equal to E in hash table H, or a null pointer if no equal element exists in the table. |
| **Function** | Finds and returns an element equal to E in hash table H, or a null pointer if no equal element exists in the table. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | struct hash\_elem\* variable |
| **Function** | Inserts NEW into hash table H, replacing any equal element already in the table, which is returned. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | NONE |
| **Function** | Destroys hash table H. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand (struct bitmap\*, int) |
| **Parameter** | Size를 expand(확장)하고자 하는 Bitmap과, 얼마나 확장할 것인지를 나타내는 integer(정수형) 변수 |
| **Return** | Expand가 완료된 Bitmap. Expand에 실패하는 경우 NULL 값을 return 한다. |
| **Function** | Parameter로 받은 Bitmap의 크기를 parameter로 받은 interger 변수 만큼 확장한다(backward expansion). 예를 들어 기존 Bitmap의 크기가 10 이고 parameter로 해당 Bitmap과 정수 2를 받은 경우, 그 Bitmap의 크기를 12로 확장시킨다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | size\_t bit\_cnt |
| **Return** | struct bitmap\* variable |
| **Function** | Initializes B to be a bitmap of BIT\_CNT bits and sets all of its bits to false. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*b, size\_t idx) |
| **Parameter** | const struct bitmap \*b, size\_t idx |
| **Return** | True/False |
| **Function** | Returns the value of the bit numbered IDX in B. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | NONE |
| **Function** | Atomically sets the bit numbered BIT\_IDX in B to true. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | True/False |
| **Function** | Returns true if every bit in B between START and START + CNT, exclusive, is set to true, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | True/False |
| **Function** | Returns true if any bits in B between START and START + CNT, exclusive, are set to true, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | True/False |
| **Function** | Returns true if any bits in B between START and START + CNT, exclusive, are set to VALUE, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t variable |
| **Function** | Returns the number of bits in B between START and START + CNT, exclusive, that are set to VALUE. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*b) |
| **Parameter** | const struct bitmap \*b |
| **Return** | NONE |
| **Function** | Dumps the contents of B to the console as hexadecimal. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*b, bool value) |
| **Parameter** | struct bitmap \*b, bool value |
| **Return** | NONE |
| **Function** | Sets all bits in B to VALUE. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | NONE |
| **Function** | Sets the CNT bits starting at START in B to VALUE. |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | struct bitmap \*b, size\_t idx, bool value |
| **Return** | NONE |
| **Function** | Atomically sets the bit numbered IDX in B to VALUE. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | NONE |
| **Function** | Atomically toggles the bit numbered IDX in B : if it is true, makes it false, and if it is false, makes it true. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | True/False |
| **Function** | Returns true if no bits in B between START and START + CNT, exclusive, are set to true, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | NONE |
| **Function** | Atomically sets the bit numbered BIT\_IDX in B to false. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t variable |
| **Function** | Finds the first group of CNT consecutive bits in B at or after START that are all set to VALUE, flips them all to !VALUE, and returns the index of the first bit in the group. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t variable |
| **Function** | Finds and returns the starting index of the first group of CNT consecutive bits in B at or after START that are all set to VALUE. If there is no such group, returns BITMAP\_ERROR. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | const struct bitmap \*b |
| **Return** | size\_t variable |
| **Function** | Returns the number of bits in B. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | struct bitmap \*b |
| **Return** | NONE |
| **Function** | Destroys bitmap B, freeing its storage. |