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

|  |  |
| --- | --- |
| 담당 교수 : | 김영재 |
| 학번 : | 20170212 |
| 이름 : | 윤다영 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | int find\_avail(bool list[10]); |
| **Parameter** | 자료구조의 각 인덱스가 사용가능한지(비었는지)를 저장하는 bool[10] array |
| **Return** | 사용가능한 자료구조의 인덱스 번호 (int) |
| **Function** | parameter, 즉 list에서 가장 처음으로 사용가능한 인덱스를 반환 |

|  |  |
| --- | --- |
| **Prototype** | int find\_list(struct list\_list ll, char\* list\_name); |
| **Parameter** | 각각의 list와 list 이름, struct list 배열에서 각 인덱스 사용여부를 저장하는 struct list\_list, 검색할 list의 이름 (char\*) |
| **Return** | 해당 이름을 가진 list의 인덱스 번호 (int) |
| **Function** | parameter로 주어진 list 이름을 list\_list에서 찾아 줌 |

|  |  |
| --- | --- |
| **Prototype** | int find\_bitmap(struct bitmap\_list bl, char\* bitmap\_name); |
| **Parameter** | 각각의 bitmap과 bitmap 이름, struct bitmap\* 배열에서 각 인덱스 사용여부를 저장하는 struct bitmap\_list, 검색할 bitmap의 이름 (char\*) |
| **Return** | 해당 이름을 가진 bitmap의 인덱스 번호 (int) |
| **Function** | parameter로 주어진 bitmap 이름을 bitmap\_list에서 찾아 줌 |

|  |  |
| --- | --- |
| **Prototype** | int find\_hashtable(struct hashtable\_list hl, char\* hashtable\_name); |
| **Parameter** | 각각의 hashtbale과 hashtable 이름, struct hash 배열에서 각 인덱스 사용여부를 저장하는 struct hashtable\_list, 검색할 hashtable의 이름 (char\*) |
| **Return** | 해당 이름을 가진 hashtable의 인덱스 번호 (int) |
| **Function** | parameter로 주어진 hashtable 이름을 hashtable\_list에서 찾아 줌 |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | bool user\_list\_less\_func(const struct list\_elem\* a, const struct list\_elem\* b, void\* aux); |
| **Parameter** | 해당 list\_elem을 포함하는 list\_item의 data 값을 비교할 두 list\_elem\* 포인터 a, b와 함수에 전달할 보조 parameter aux |
| **Return** | a의 data 값이 더 작은지를 구분하는 bool |
| **Function** | a의 data 값과 b의 data 값을 비교하여 a의 data 값이 b의 data 값보다 작을 경우에 true 반환, 같거나 클 경우에는 false 반환 |

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | void(return값 없음) |
| **Function** | parameter로 받은 list를 초기화 시켜준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_begin(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 가장 첫번째 interior element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_next(struct list\_elem\*); |
| **Parameter** | 해당 함수를 적용할 list\_elem\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list\_elem이 다음으로 가리키는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_end(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 tail에 해당하는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_rbegin(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 가장 마지막 interior element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_prev(struct list\_elem\*); |
| **Parameter** | 해당 함수를 적용할 list\_elem\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list\_elem의 바로 앞(이전) list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_rend(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 head에 해당하는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_head(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 head에 해당하는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_tail(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 tail에 해당하는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert(struct list\_elem\*, struct list\_elem\*); |
| **Parameter** | list에 새로 삽입할 원소의 포인터 list\_elem\* elem과, 삽입할 원소의 위치의 바로 뒤 원소의 포인터 list\_elem\* before |
| **Return** | void(return값 없음) |
| **Function** | before 앞에 elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice(struct list\_elem\* before, struct list\_elem\* first, struct list\_elem\* last); |
| **Parameter** | 새로운 위치에 삽입될 원소의 시작 포인터 list\_elem\* first, 끝지점 포인터 last와, 삽입할 원소들의 위치의 바로 뒤 원소의 포인터 list\_elem\* before |
| **Return** | void(return값 없음) |
| **Function** | list에서 first부터 last까지 추출한 후 삭제하고, 그 원소들을 새로운 위치인 before 앞에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front(struct list\*, struct list\_elem\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터와 list\_elem\* 포인터 |
| **Return** | void(return값 없음) |
| **Function** | parameter로 주어진 list의 가장 앞에 주어진 list\_elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back(struct list\*, struct list\_elem\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터와 list\_elem\* 포인터 |
| **Return** | void(return값 없음) |
| **Function** | parameter로 주어진 list의 가장 뒤에 주어진 list\_elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_remove(struct list\_elem\*); |
| **Parameter** | list에서 삭제할 list\_elem\* 포인터 |
| **Return** | 함수의 실행 결과로 삭제된 list\_elem |
| **Function** | parameter로 주어진 list\_elem을 list에서 제거하고 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_pop\_front(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수의 실행 결과로 삭제된 list\_elem |
| **Function** | parameter로 주어진 list의 가장 처음 list\_elem을 삭제하고 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_pop\_back(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수의 실행 결과로 삭제된 list\_elem |
| **Function** | parameter로 주어진 list의 가장 마지막 list\_elem을 삭제하고 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_front(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 가장 첫번째 interior element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_back(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | parameter로 받은 list의 가장 마지막 interior element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수의 결과값 (size\_t) |
| **Function** | parameter로 받은 list의 크기를 계산하여 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | 함수의 결과를 저장하는 bool값 |
| **Function** | parameter로 받은 list가 비었는지를 확인하여 비었으면 true, 한 개라도 원소가 존재하면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void swap(struct list\_elem\*\* a, struct list\_elem\*\* b); |
| **Parameter** | 해당 함수를 적용시킬 list\_elem\*\* a, list\_elem\*\* b |
| **Return** | void(return값 없음) |
| **Function** | 주어진 a와 b가 가리키는 list\_elem\*을 서로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* 포인터 |
| **Return** | void(return값 없음) |
| **Function** | parameter로 받은 list안 원소의 순서를 역순으로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted(struct list\_elem\* a, struct list\_elem\* b, list\_less\_func\* less, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list의 시작 위치 가리키는 list\_elem\* a, 끝지점 가리키는 list\_elem\* b, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | 함수의 결과를 저장하는 bool |
| **Function** | less 함수 포인터가 가리키는 함수에 따라 a부터 b가 제대로 정렬되었는지 확인하여 정렬되어 있다면 true, 아니라면 false를 반환 |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem\* find\_end\_of\_run(struct list\_elem\* a, struct list\_elem\* b, list\_less\_func\* less, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list의 시작 위치 가리키는 list\_elem\* a, 끝지점 가리키는 list\_elem\* b, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | 함수의 결과인 run의 가장 마지막 원소 가리키는 list\_elem\* |
| **Function** | a부터 시작해서 b 혹은 b의 이전에 있는 원소들 중 하나의 원소까지 nondecreasing 순서로 나열된 run을 찾아 해당 run의 가장 마지막 원소를 반환 |

|  |  |
| --- | --- |
| **Prototype** | static void inplace\_merge(struct list\_elem\* a0, struct list\_elem\* a1b0, struct list\_elem\* b1, list\_less\_func\* less, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list의 시작 위치 가리키는 list\_elem\* a0, merge의 중간점 list\_elem\* a1b0, 끝지점 가리키는 list\_elem\* b1, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | less 함수에 따라 정렬되어 있는 a0부터 a1b0까지의 원소와 a1b0부터 b1까지의 원소를 병합하여 다시 less 함수에 따라 정렬된 a0부터 b1까지의 구간을 만들어준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort(struct list\*, list\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list\* list, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | less 함수에 따라 주어진 list를 정렬해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered(struct list\*, struct list\_elem\*, list\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list\* list, 새로 삽입할 list\_elem\* elem, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | less 함수에 따라 주어진 list에서 elem의 위치를 찾아 삽입해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique(struct list\*, struct list\* duplicates, list\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list\* list, 삭제될 원소들이 삽입될 list\* dumplicates, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | list에서 less함수에 의해 동일한 값들을 갖는 원소들 중 제일 첫번째 원소를 제외한 원소들을 삭제해주고, duplicates가 NULL을 가리키지 않는다면 삭제된 값들을 duplicates에 삽입해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_max(struct list\*, list\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list\* list, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | list에서 less함수에 의해 가장 큰 값을 가진 가장 첫번째 원소를 찾아 반환한다. empty list의 경우 해당 list의 tail을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_min(struct list\*, list\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 list\* list, 크기비교 logic을 담는 함수의 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | list에서 less함수에 의해 가장 작은 값을 가진 가장 첫번째 원소를 찾아 반환한다. empty list의 경우 해당 list의 tail을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* find\_list\_index(struct list\*, int); |
| **Parameter** | 해당 함수를 적용할 list\* list, 리스트 원소의 인덱스 번호(int) |
| **Return** | 함수에서 찾은 원소에 해당하는 list\_elem\* 포인터 |
| **Function** | list에서 주어진 인덱스 번호에 해당하는 위치에 있는 list\_elem을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem\*, struct list\_elem\*); |
| **Parameter** | 해당 함수를 수행할 두 list\_elem\* 포인터 a, b |
| **Return** | void(return값 없음) |
| **Function** | 두 list\_elem의 위치를 서로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list\*); |
| **Parameter** | 해당 함수를 적용할 list\* list |
| **Return** | void(return값 없음) |
| **Function** | 주어진 list의 원소들의 순서를 임의로 뒤섞는다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | unsigned user\_hash\_hash\_func(const struct hash\_elem\* e, void\* aux); |
| **Parameter** | hash 함수를 적용한 hash 값을 추출할 hash\_elem\* 포인터 e, 함수에 보조적인 내용을 전달할 parameter aux |
| **Return** | e에 해당하는 hash 값 (unsigned int) |
| **Function** | hash.h에 선언된 hash\_int 함수를 사용하여 주어진 e의 hash value를 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | bool user\_hash\_less\_func(const struct hash\_elem\* a, const struct hash\_elem\* b, void\* aux); |
| **Parameter** | 해당 hash\_elem을 포함하는 hash\_item의 data 값을 비교할 두 hash\_elem\* 포인터 a, b와 함수에 전달할 보조 parameter aux |
| **Return** | a의 data 값이 더 작은지를 구분하는 bool |
| **Function** | a의 data 값과 b의 data 값을 비교하여 a의 data 값이 b의 data 값보다 작을 경우에 true 반환, 같거나 클 경우에는 false 반환 |

|  |  |
| --- | --- |
| **Prototype** | void user\_hash\_square\_func(struct hash\_elem\* e, void\* aux); |
| **Parameter** | hash\_item으로 변환했을 때 data 값에 함수를 적용시킬 hash\_elem\* e, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | 주어진 e에 해당하는 hash\_item의 data 값을 원래의 값에서 제곱한 값으로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void user\_hash\_triple\_func(struct hash\_elem\* e, void\* aux); |
| **Parameter** | hash\_item으로 변환했을 때 data 값에 함수를 적용시킬 hash\_elem\* e, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | 주어진 e에 해당하는 hash\_item의 data 값을 원래의 값에서 세제곱한 값으로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void user\_hash\_destruct\_func(struct hash\_elem\* e, void\* aux); |
| **Parameter** | 해당 함수를 적용시킬 hash\_elem\* e, 함수에 전달할 보조 parameter aux |
| **Return** | void(return값 없음) |
| **Function** | 주어진 e의 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init(struct hash\*, hash\_hash\_func\*, hash\_less\_func\*, void\* aux); |
| **Parameter** | 해당 함수를 적용할 hash\* hash, hash함수를 가리키는 포인터 hash, 크기 비교를 해주는 함수 포인터 less, 함수에 전달할 보조 parameter aux |
| **Return** | hash 초기화에 성공여부를 저장하는 bool |
| **Function** | parameter로 받은 hash의 hash함수로 지정한 hash 함수를, 크기 비교를 위한 함수로 지정한 less 함수를 설정해주고 hash를 초기화 시켜준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear(struct hash\*, hash\_action\_func\*); |
| **Parameter** | 해당 함수를 적용할 hash\* hash, 해당 함수에서 각 hash 원소의 메모리를 해제해줄 함수 포인터 destructor |
| **Return** | void(return값 없음) |
| **Function** | destructor이 NULL을 가리키지 않을 경우 hash의 각 hash 원소를 제거하고 메모리를 해제해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy(struct hash\*, hash\_action\_func\*); |
| **Parameter** | 해당 함수를 적용할 hash\* hash, 해당 함수에서 각 hash 원소의 메모리를 해제해줄 함수 포인터 destructor |
| **Return** | void(return값 없음) |
| **Function** | hash의 모든 hash 원소를 제거, 메모리 해제한 후 hashtable hash도 제거, 메모리 해제해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_insert(struct hash\*, struct hash\_elem\*); |
| **Parameter** | 해당 함수를 적용할 hash\* h, 새로 삽입할 원소 hash\_elem\* new |
| **Return** | 해당 함수의 결과 hashtable 원소를 반환하는 hash\_elem\* |
| **Function** | 해당 h에 new를 삽입한다. 만약 이미 같은 값을 갖는 원소가 h에 있다면 new를 삽입하지 않고, 해당 hash\_elem\*을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_replace(struct hash\*, struct hash\_elem\*); |
| **Parameter** | 해당 함수를 적용할 hash\* h, 새로 삽입할 원소 hash\_elem\* new |
| **Return** | 해당 함수의 결과 hashtable 원소를 반환하는 hash\_elem\* |
| **Function** | h에 new의 값과 같은 값을 갖는 원소가 있을 경우 new로 대체하고 기존의 대체된 원소를 반환한다. new의 값과 같은 값을 갖는 원소가 없다면 new를 삽입하고 NULL 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_find(struct hash\*, struct hash\_elem\*); |
| **Parameter** | 해당 함수를 적용할 hash\* h, 원소 hash\_elem\* new |
| **Return** | 함수의 결과에 해당하는 원소를 가리키는 hash\_elem\* 포인터 |
| **Function** | h에서 주어진 new의 data값과 같은 값을 갖는 원소를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_delete(struct hash\*, struct hash\_elem\*); |
| **Parameter** | 해당 함수를 적용할 hash\* h, 삭제할 data값을 가진 hash\_elem\* e |
| **Return** | 삭제된 hash\_elem\* (삭제 실패 시 NULL 포인터) |
| **Function** | hashtable h에서 주어진 e의 data값과 같은 data값을 갖는 원소를 찾아 삭제하고 반환해준다. 만약 e와 같은 data값을 갖는 원소가 없다면 NULL 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply(struct hash\*, hash\_action\_func\*); |
| **Parameter** | 해당 함수를 적용할 hash\* hash, 해당 함수에서 각 hash 원소에 적용해줄 작업을 수행하는 함수 포인터 hash\_action\_func\* action |
| **Return** | void(return값 없음) |
| **Function** | h의 모든 원소에 대하여 action 함수를 적용해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first(struct hash\_iterator\*, struct hash\*); |
| **Parameter** | hash\_iterator를 새롭게 생성할 주소 hash\_iterator \*i, 해당 함수를 적용할 hash\* h |
| **Return** | void(return값 없음) |
| **Function** | hash\_iterator의 주소를 담는 포인터 i에 h의 정보를 담는 hash\_iterator를 새롭게 생성해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_next(struct hash\_iterator\*); |
| **Parameter** | 해당 함수를 수행시킬 hash\_iterator\* i |
| **Return** | 함수의 수행 결과로 찾은 hash\_elem\* |
| **Function** | i가 현재 가리키는 원소의 바로 다음 hashtable의 원소를 반환한다. 더 이상 뒤에 남은 원소가 없다면 NULL 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_cur(struct hash\_iterator\*); |
| **Parameter** | 해당 함수를 수행시킬 hash\_iterator\* i |
| **Return** | 함수의 수행 결과로 찾은 hash\_elem\* |
| **Function** | i가 현재 가리키는 hashtable의 원소를 반환한다. 더 이상 남은 원소가 없다면 NULL 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size(struct hash\*); |
| **Parameter** | 해당 함수를 적용시킬 hash\* h |
| **Return** | hastable 원소들의 개수를 저장하는 size\_t |
| **Function** | 현재 h에 있는 hashtable 원소들의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty(struct hash\*); |
| **Parameter** | 해당 함수를 적용시킬 hash\* h |
| **Return** | 해당 함수의 결과 값을 저장하는 bool |
| **Function** | 현재 h가 비어 있는지 확인하고, 비어 있다면 true를, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes(const void\*, size\_t); |
| **Parameter** | hashing할 데이터의 포인터 const void \*buf\_, 데이터의 크기 size\_t size |
| **Return** | buf\_가 가리키는 데이터의 hash 값 |
| **Function** | buf\_가 가리키는 size 크기의 데이터를 hashing하여 그 결과 hash 값을 반환 |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string(const char\*); |
| **Parameter** | hashing할 string을 가리키는 포인터 const char \*s\_ |
| **Return** | s\_가 가리키는 string의 hash 값 |
| **Function** | s\_가 가리키는 string을 hashing하여 그 결과 hash 값을 반환(size만큼 hash \* FNV\_32\_PRIME의 값과 \*buf++의 값을 계속하여 XOR 연산 시켜준다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int(int); |
| **Parameter** | hashing할 int형 데이터 i |
| **Return** | i의 hash 값 |
| **Function** | hash\_bytes를 호출하여 i를 hash해준다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \* find\_bucket (struct hash \*h, struct hash\_elem \*e); |
| **Parameter** | 해당 함수를 적용시킬 hash\* h, hashtable의 원소 hash\_elem\* e |
| **Return** | e가 있는 bucket의 위치를 가리키는 list\* |
| **Function** | h안에서 e가 소속된 bucket의 위치를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \* find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e); |
| **Parameter** | 해당 함수를 적용시킬 hash\* h, hashtable의 bucket을 가리키는 포인터 list\* bucket, hashtable의 원소 hash\_elem\* e |
| **Return** | 함수의 결과로 찾은 원소의 포인터 hash\_elem\* |
| **Function** | 주어진 h의 주어진 bucket 안에서 e와 동일한 data값을 갖는 원소를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x); |
| **Parameter** | 해당 함수를 적용할 size\_t x |
| **Return** | x에 해당 함수를 적용한 결과값 size\_t |
| **Function** | x에서 1로 set된 가장 낮은 자리 수의 값을 0으로 바꿔주고 그 값을 반환 |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t is\_power\_of\_2 (size\_t x); |
| **Parameter** | 해당 함수를 적용할 size\_t x |
| **Return** | x에 해당 함수를 적용한 결과값 size\_t |
| **Function** | x가 2의 제곱이면 true를, 아니면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h); |
| **Parameter** | 해당 함수를 적용할 hash\* h |
| **Return** | void(return값 없음) |
| **Function** | buckets의 개수를 다시 이상적인 상태로 조정해준다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e); |
| **Parameter** | 해당 함수를 적용할 hash\* h, list\* bucket, 새로 삽입할 hash\_elem\* e |
| **Return** | void(return값 없음) |
| **Function** | 주어진 h의 bucket에 새로운 원소 e를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e); |
| **Parameter** | 해당 함수를 적용할 hash\* h, list\* bucket, 삭제할 hash\_elem\* e |
| **Return** | void(return값 없음) |
| **Function** | 주어진 h의 bucket에서 e에 해당하는 원소를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int); |
| **Parameter** | hashing할 int형 데이터 i |
| **Return** | i의 hash 값 |
| **Function** | hash\_int와 다른 방법으로 i를 hashing해준다. (i의 size만큼 각 byte의 값과 hash\*hashmul 값을 계속하여 홀수번째(1, 3, 5…)에는 XOR 연산을, 짝수번째에는 XNOR 연산을 해준다.) |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_idx (size\_t bit\_idx); |
| **Parameter** | 해당 함수를 적용할 size\_t bit\_idx |
| **Return** | 함수를 적용한 결과 인덱스 번호를 저장하는 size\_t |
| **Function** | 주어진 bit\_idx에 해당하는 bitmap의 원소의 인덱스를 반환 |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx); |
| **Parameter** | 해당 함수를 적용할 size\_t bit\_idx |
| **Return** | 함수를 적용한 결과를 저장하는 elem\_type |
| **Function** | bit\_idx에 해당하는 비트만 1의 값을 갖는 elem\_type 반환 |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt (size\_t bit\_cnt); |
| **Parameter** | 해당 함수를 적용할 size\_t bit\_idx |
| **Return** | 함수를 적용한 결과 size를 저장하는 size\_t |
| **Function** | bit\_cnt 만큼의 비트 수를 갖도록 하는 bitmap의 원소의 개수를 반환 |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt (size\_t bit\_cnt); |
| **Parameter** | 해당 함수를 적용할 size\_t bit\_idx |
| **Return** | 함수를 적용한 결과 size를 저장하는 size\_t |
| **Function** | bit\_cnt 만큼의 비트 수를 갖도록 하는 bitmap에 필요한 byte 수를 반환 |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type last\_mask (const struct bitmap \*b); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b |
| **Return** | 함수의 결과 bit mask를 저장하는 elem\_type |
| **Function** | b의 마지막 원소의 비트들에서 사용되는 비트들만 1의 값을 갖고 나머지는 0의 값을 갖도록 하는 bit mask를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_create(size\_t bit\_cnt); |
| **Parameter** | 생성할 비트맵의 크기를 전달하는 size\_t bit\_cnt |
| **Return** | 생성된 bitmap\* |
| **Function** | bit\_cnt 만큼의 비트를 가진 비트맵을 생성하고, 비트맵의 모든 비트들을 false로 set한다. 생성에 실패하면 NULL 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_create\_in\_buf(size\_t bit\_cnt, void\*, size\_t block\_size); |
| **Parameter** | 생성할 비트맵의 크기를 전달하는 size\_t bit\_cnt |
| **Return** | 생성된 bitmap\* |
| **Function** | 미리 할당된 block의 block\_size bytes에 bit\_cnt 만큼의 비트를 가진 bitmap을 생성하고, 비트맵의 모든 비트들을 false로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size(size\_t bit\_cnt); |
| **Parameter** | 해당 함수를 적용할 size\_t bit\_cnt |
| **Return** | 함수의 결과 바이트 수를 저장하는 size\_t |
| **Function** | bit\_cnt 크기의 bitmap을 수용하는 데에 필요한 바이트 수를 반환 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy(struct bitmap\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b |
| **Return** | void(return값 없음) |
| **Function** | b의 메모리를 해제하고, b를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size(const struct bitmap\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b |
| **Return** | 함수 결과 비트 수를 저장하는 size\_t |
| **Function** | b의 비트 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set(struct bitmap\*, size\_t idx, bool); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 인덱스 번호 size\_t idx, 값 bool value |
| **Return** | void(return값 없음) |
| **Function** | b의 idx번째 비트를 value의 값으로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark(struct bitmap\*, size\_t idx); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 인덱스 번호 size\_t idx |
| **Return** | void(return값 없음) |
| **Function** | b의 idx번째 비트의 값을 true로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset(struct bitmap\*, size\_t idx); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 인덱스 번호 size\_t idx |
| **Return** | void(return값 없음) |
| **Function** | b의 idx번째 비트의 값을 false로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip(struct bitmap\*, size\_t idx); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 인덱스 번호 size\_t idx |
| **Return** | void(return값 없음) |
| **Function** | b의 idx번째 비트의 값이 원래 true였다면 false로, false였다면 true로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test(const struct bitmap\*, size\_t idx); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 인덱스 번호 size\_t idx |
| **Return** | 함수의 결과로 찾은 비트의 값을 저장하는 bool |
| **Function** | b의 idx번째 비트의 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all(struct bitmap\*, bool); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 값 bool value |
| **Return** | void(return값 없음) |
| **Function** | b의 모든 bit의 값을 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple(struct bitmap\*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt, 값 bool value |
| **Return** | void(return값 없음) |
| **Function** | start 인덱스부터 b의 cnt 개수의 비트들을 value 값으로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count(const struct bitmap\*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt, 값 bool value |
| **Return** | 함수의 결과로 얻어진 비트 개수 size\_t |
| **Function** | start 인덱스부터 b의 cnt 개수의 비트들 중 value에 해당하는 값을 가진 비트의 개수를 세어 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains(const struct bitmap\*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt, 값 bool value |
| **Return** | 함수의 결과로 true/false 저장하는 bool |
| **Function** | start 인덱스부터 b의 cnt 개수의 비트들 중 value에 해당하는 값을 가진 비트가 하나라도 있다면 true를, 아니라면 false를 반환 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any(const struct bitmap\*, size\_t start, size\_t cnt); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt |
| **Return** | 함수의 결과로 true/false 저장하는 bool |
| **Function** | start 인덱스부터 b의 cnt 개수의 비트들 중 true 값을 가진 비트가 하나라도 있다면 true를, 아니라면 false를 반환 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none(const struct bitmap\*, size\_t start, size\_t cnt); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt |
| **Return** | 함수의 결과로 true/false 저장하는 bool |
| **Function** | start 인덱스부터 b의 cnt 개수의 비트들 중 true 값을 가진 비트가 하나도 없다면 true를, 아니라면 false를 반환 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all(const struct bitmap\*, size\_t start, size\_t cnt); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt |
| **Return** | 함수의 결과로 true/false 저장하는 bool |
| **Function** | start 인덱스부터 b의 cnt 개수의 모든 bit가 true 값을 가진다면 true를, 아니라면 false를 반환 |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan(const struct bitmap\*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt, 값 bool value |
| **Return** | 함수의 결과로 얻어진 비트들의 시작 인덱스를 저장하는 size\_t |
| **Function** | b에서 start 인덱스 이후(포함 가능) cnt개의 value에 해당하는 값을 가진 연속된 가장 첫번째 비트들을 찾아 반환한다. 만약 그런 비트들이 없다면 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip(struct bitmap\*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 해당 함수에 적용될 bitmap\* b, 인덱스 번호 size\_t start, 비트 개수 cnt, 값 bool value |
| **Return** | 함수의 결과로 얻어진 비트들의 시작 인덱스를 저장하는 size\_t |
| **Function** | b에서 start 인덱스 이후(포함 가능) cnt개의 value에 해당하는 값을 가진 연속된 가장 첫번째 비트들을 찾고 그들의 값을 모두 !value로 바꾼 후 그 위치를 반환한다. 만약 그런 비트들이 없다면 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size(const struct bitmap\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b |
| **Return** | 함수의 결과로 얻어진 바이트 수를 저장하는 size\_t |
| **Function** | b를 파일에 저장하기 위해 필요한 바이트의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read(struct bitmap\*, struct file\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, file\* file |
| **Return** | 파일 읽기의 성공 여부를 저장하는 bool |
| **Function** | file로부터 b를 읽어 온다. 파일 읽기가 성공하면 true를, 실패하면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write(const struct bitmap\*, struct file\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, file\* file |
| **Return** | 파일 쓰기의 성공 여부를 저장하는 bool |
| **Function** | b를 file에 기록한다. 파일 쓰기가 성공하면 true를, 실패하면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump(const struct bitmap\*); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b |
| **Return** | void(return값 없음) |
| **Function** | b의 내용을 콘솔에 16진수로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap\* bitmap, int size); |
| **Parameter** | 해당 함수를 적용할 bitmap\* b, 추가로 확장할 크기를 전달하는 int size |
| **Return** | 확장된 bitmap 가리키는 bitmap\* |
| **Function** | b의 크기를 원래 크기에 size를 더한 값으로 확장해준다. 만약 실패하면 NULL 포인터를 반환한다. |