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

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 교수님 |
| 학번 : | 20171662 |
| 이름 : | 이나연 |
|  |  |

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size) |
| **Parameter** | struct bitmap \*bitmap: 확장하고자 하는 bitmap  int size: 확장하고자 하는 비트 수 |
| **Return** | 확장 성공시 확장된 bitmap, 실패할 시 NULL |
| **Function** | 주어진 bitmap을 size만큼 뒤로 확장한다. 확장된 비트에는 모두 0으로 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | int i: hashing될 정수 |
| **Return** | 주어진 정수의 해시값 |
| **Function** | 주어진 정수를 (sqrt(5)-1)/2로 곱한 후 2^4를 곱하여 해당 숫자의 정수 부분만 hash\_bytes 함수로 넘겨 해시값을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | swap 될 list element a와 b |
| **Return** | - |
| **Function** | 주어진 두 list element들을 swap한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | element들을 섞을 list |
| **Return** | - |
| **Function** | 주어진 list내의 list element들을 임의의 수만큼 shuffle한다. list\_swap()함수를 이용한다. |

|  |  |
| --- | --- |
| **Prototype** | void square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | data를 제곱할 hash element |
| **Return** | - |
| **Function** | 주어진 hash element가 들어있는 hash item을 찾아 해당 item의 data를 제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | data를 세제곱할 hash element |
| **Return** | - |
| **Function** | 주어진 hash element가 들어있는 hash item을 찾아 해당 item의 data를 세제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux) |
| **Parameter** | data의 대소를 비교할 두 list element |
| **Return** | a의 data가 더 작으면 true, 아니면 false |
| **Function** | 주어진 list element a, b가 들어있는 list item을 찾아 해당 item들의 data의 대소를 비교한다. |

|  |  |
| --- | --- |
| **Prototype** | int find\_index\_by\_name(char\*\*name\_list, char\* name, int cnt) |
| **Parameter** | 찾을 list/bitmap/hash table의 이름, 이름이 저장되어 있는 list, 저장되어 있는 이름의 수 |
| **Return** | 주어진 name list상에서의 이름이 저장되어 있는 배열의 index |
| **Function** | 주어진 이름이 주어진 name list의 몇 번째 인덱스에 저장되어 있는지를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void destructor(struct hash\_elem \*e, void \*aux) |
| **Parameter** | free할 hash element |
| **Return** | - |
| **Function** | 주어진 hash element가 들어있는 hash item을 할당해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_dump\_all(struct hash\_elem \*e, void \*aux) |
| **Parameter** | data를 출력할 hash element |
| **Return** | - |
| **Function** | 주어진 hash element를 담고 있는 hash item을 찾은 후 그 item의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_hash(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | 해시값을 찾을 data에 해당하는 hash element |
| **Return** | data에 해당하는 해시값 |
| **Function** | 주어진 hash element가 들어있는 hash item을 찾은 후 해당 item의 data의 해시값을 구한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux) |
| **Parameter** | data의 대소를 비교할 hash element a, b |
| **Return** | a의 data가 더 작으면 true, 아니라면 false |
| **Function** | hash element a, b를 담고 있는 hash item을 찾은 뒤 해당 item안에 들어있는 data의 대소를 비교하여 a에 해당하는 data가 더 작다면 true, 아니면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | int find\_name(char\* name,char\*\* list\_name,char\*\* hash\_name,char\*\* bitmap\_name,int list\_cnt, int hash\_cnt, int bitmap\_cnt){ |
| **Parameter** | 찾을 이름, list, hash table, bitmap의 이름들을 저장하는 list와 각 list에 저장되어 있는 이름의 수 |
| **Return** | 만약 list의 이름이라면 10+i, hash table의 이름이라면 20+i, bitmap의 이름이라면 30+i, i는 이름이 저장되어 있는 인덱스 번호 |
| **Function** | 주어진 이름이 어떤 type인지 (list, bitmap, hash table 중 하나), name list에서의 index는 몇 번인지를 구한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | list\_entry(LIST\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next \  - offsetof (STRUCT, MEMBER.next))) |
| **Parameter** | list item을 찾을 list element, struct 구조체 명(list\_item), 해당 구조체 내에서 list element가 들어있을 member명 |
| **Return** | list element가 들어있는 list item의 주소 |
| **Function** | 주어진 list element가 포함되어 있는 list item의 주소값을 구한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_head (struct list\_elem \*elem) |
| **Parameter** | head인지 확인할 list element |
| **Return** | elem이 head라면 true, 아니면 false |
| **Function** | 주어진 list element가 head인지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_interior (struct list\_elem \*elem) |
| **Parameter** | interior인지 확인할 list element |
| **Return** | elem이 interior element라면 true, 아니면 false |
| **Function** | 주어진 list element가 내부 element인지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_tail (struct list\_elem \*elem) |
| **Parameter** | tail인지 확인할 list element |
| **Return** | elem이 tail이라면 true, 아니면 false |
| **Function** | 주어진 list element가 tail인지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*list) |
| **Parameter** | 초기화할 list |
| **Return** | - |
| **Function** | 주어진 list를 빈 list로 초기화해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*list) |
| **Parameter** | 첫 element를 찾을 list |
| **Return** | 주어진 list의 head의 다음 노드 |
| **Function** | 주어진 list의 첫 element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*elem) |
| **Parameter** | 다음 노드를 찾고자 하는 list element |
| **Return** | 주어진 list element의 next list element |
| **Function** | 주어진 list element의 다음 list element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*list) |
| **Parameter** | tail을 찾고자 하는 list |
| **Return** | 주어진 list의 tail |
| **Function** | 주어진 list의 tail을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*list) |
| **Parameter** | 오른쪽 시작 노드를 찾을 list |
| **Return** | 주어진 list의 반대 beginning element |
| **Function** | list를 반대로 돌기 위하여 사용할 수 있음. list의 오른쪽 시작, 즉 제일 마지막 노드를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*elem) |
| **Parameter** | 이전 노드를 찾을 list element |
| **Return** | elem의 직전 list element |
| **Function** | 주어진 list element의 직전 list element를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*list) |
| **Parameter** | 오른쪽으로 부터 끝 노드를 찾을 list |
| **Return** | 주어진 list의 반대 last element, 즉 list의 head |
| **Function** | list를 반대로 돌기 위하여 사용할 수 있음. list의 오른쪽 끝, 즉 head 노드를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*list) |
| **Parameter** | head를 찾을 list |
| **Return** | list의 head element |
| **Function** | 주어진 list의 head를 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*list) |
| **Parameter** | tail을 찾을 list |
| **Return** | list의 tail element |
| **Function** | 주어진 list의 tail을 찾아준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | 새로 삽입할 list element, 새로 삽입할 element의 next가 될 list element |
| **Return** | - |
| **Function** | before의 직전에 elem을 삽입한다. 이 때 before는 interior혹은 tail이어야 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | first, last: 잘라낼 list element들의 첫 element와 마지막 element의 다음 element (last element는 제외된다.)  before: 잘라낸 element들을 before의 직전에 삽입한다. |
| **Return** | - |
| **Function** | first, last가 존재하던 list에서 first ~ last 직전 node까지를 빼낸 다음 before의 직전에 삽입한다. 이 때 before는 interior 혹은 tail이어야 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | 새로운 노드 elem, elem이 삽입될 list |
| **Return** | - |
| **Function** | 주어진 list의 제일 처음에 elem을 삽입한다. 이는 list\_insert()함수를 이용하여 list\_begin(list)과 elem을 전달하여 구현한다. 그렇게 되면 현재 list의 begin element의 직전에 elem이 삽입되게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | 새로운 노드 elem, elem이 삽입될 list |
| **Return** | - |
| **Function** | 주어진 list의 제일 마지막에 elem을 삽입한다. 이는 list\_insert()함수를 이용하여 list\_end(list)과 elem을 전달하여 구현한다. 그렇게 되면 현재 list의 tail element의 직전에 elem이 삽입되게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*elem) |
| **Parameter** | 제거하고자 하는 list element |
| **Return** | 제거하고자 한 list element |
| **Function** | elem을 list에서 제거한 후 해당 elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*list) |
| **Parameter** | element를 pop하고자 하는 list |
| **Return** | pop된 list element |
| **Function** | 주어진 list의 제일 첫 list element를 pop한 후 해당 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*list) |
| **Parameter** | element를 pop하고자 하는 list |
| **Return** | pop된 list element |
| **Function** | 주어진 list의 제일 끝 list element를 pop한 후 해당 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*list) |
| **Parameter** | 제일 앞에 있는 element를 찾고자 하는 list |
| **Return** | list의 제일 앞 element |
| **Function** | 주어진 list의 제일 첫 list element를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*list) |
| **Parameter** | 제일 뒤에 있는 element를 찾고자 하는 list |
| **Return** | list의 제일 뒤 element |
| **Function** | 주어진 list의 제일 뒤 list element를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*list) |
| **Parameter** | size를 알고자 하는 list |
| **Return** | 주어진 list의 size |
| **Function** | 주어진 list의 내부 element의 개수를 세어 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | 비어있는지 확인하고자 하는 list |
| **Return** | list가 비어있다면 true, 아니라면 false |
| **Function** | 주어진 list가 비어있는지 확인한다. head의 next가 tail인지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | static void swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | swap하고자 하는 두 element\* |
| **Return** | - |
| **Function** | a와 b가 가리키는 `struct list\_elem \*'를 서로 swap한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | 뒤집고자 하는 list |
| **Return** | - |
| **Function** | 주어진 list의 내부 element들의 순서를 뒤집는다. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux) |
| **Parameter** | sorted되어있는지 확인할 구간의 첫 element a, 마지막 element의 직후 element b, 대소비교에 쓰이는 함수 less |
| **Return** | a~b 직전까지의 구간의 data가 오름차순으로 정렬되어 있으면 true, 아니면 false |
| **Function** | 주어진 구간의 data가 오름차순으로 정렬되어 있는지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*find\_end\_of\_run (struct list\_elem \*a, struct list\_elem \*b, list\_less\_func \*less, void \*aux) |
| **Parameter** | 오름차순으로 정렬되어 있는지 확인할 구간의 처음 a, 마지막 직후 b, 대소비교에 쓰일 함수 less |
| **Return** | 오름차순으로 정렬되어 있는 구간의 바로 다음 list element |
| **Function** | a element부터 시작하여 b element 직전까지 오름차순으로 정렬되어 있는 구간을 확인한 후 오름차순 구간이 끝난 직후의 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void inplace\_merge (struct list\_elem \*a0, struct list\_elem \*a1b0, struct list\_elem \*b1, list\_less\_func \*less, void \*aux) |
| **Parameter** | 첫 list element 구간의 첫 element a0, 마지막 직후의 노드 a1b0, 두번째 list element 구간의 첫 element a1b0, 마지막 직후의 노드 b1, 대소 비교용 함수 less |
| **Return** | - |
| **Function** | 정렬되어 있는 두 구간 a0~a1b0(제외), a1b0~b1(제외)을 병합한다. 병합 역시 병합 이후에 해당 구간이 오름차순의 형태로 정렬되어 있어야 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 정렬할 list, 대소비교용 함수 less |
| **Return** | - |
| **Function** | 주어진 list를 오름차순으로 정렬한다. 이는 merge sort의 자료구조를 따른다. inplace\_merge()함수를 이용하여 구현한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 삽입할 element elem, elem을 삽입할 list, 대소비교용 함수 less |
| **Return** | - |
| **Function** | 주어진 elem을 주어진 list내에 삽입한다. 이 때, list는 이미 오름차순으로 정렬되어 있는 상태여야 하고 새로 삽입하는 elem역시 오름차순 정렬을 해치지 않고 삽입된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 각 element의 data가 unique한지 확인할 list, unique하지 않은 data가 들어 있는 element들의 list가 저장될 duplicates, 대소비교용 함수 less |
| **Return** | - |
| **Function** | 주어진 list내의 data들이 unique한지 확인한다. 만약 중복되는 data가 있다면 해당 element를 remove하고 대신 duplicate list에 삽입한다. 즉 해당 함수의 마지막에 list엔 중복되지 않는 data를 가지는 element들만 남게 될 것이고 제외된 element들은 모두 duplicate list에 들어가 있을 것이다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 제일 큰 data를 포함하는 element를 찾을 list, 대소비교용 함수 less |
| **Return** | 제일 큰 data를 포함하는 element |
| **Function** | 주어진 list의 element들 중 가장 큰 data를 가지고 있는 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 제일 작은 data를 포함하는 element를 찾을 list, 대소비교용 함수 less |
| **Return** | 제일 작은 data를 포함하는 element |
| **Function** | 주어진 list의 element들 중 가장 작은 data를 가지고 있는 element를 반환한다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | 초기화 할 hash table h, hash element의 해시값을 구해주는 함수 hash, hash element끼리의 대소비교용 함수 less |
| **Return** | 성공적으로 초기화가 되면 true, 아니면 false |
| **Function** | hash table h의 모든 member들을 초기화한다. 또한 해시 함수 hash, 대소 비교용 함수 less도저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor |
| **Parameter** | element를 모두 제거할 hash table h, hash element들의 메모리를 할당 해제하는 함수 destructor |
| **Return** | - |
| **Function** | h의 모든 element들을 제거 및 할당 해제한다. destructor가 NULL이 아니라면 hash의 모든 element에 대하여 실행된다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | 제거할 hash table h, hash element들의 메모리를 할당 해제하는 함수 destructor |
| **Return** | - |
| **Function** | hash table h를 제거한다. destructor가 NULL이 아니라면 hash의 모든 element에 대하여 실행된다. element들이 모두 제거되고 난 후 bucket의 head도 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | 새로운 element를 삽입할 hash table h, 새로 삽입될 hash element new |
| **Return** | 만약 새로운 element가 hash table내의 element와 중복이라면 중복되는 element를 return, 없다면 NULL |
| **Function** | 새로운 hash element new를 hash table h에 삽입한 후 이미 존재하는 element와 중복이 아니라면 NULL을 반환한다. 만약 중복이 된다면 삽입을 하지 않는다. |

|  |  |
| --- | --- |
| **Prototype** | hash\_entry(HASH\_ELEM, STRUCT, MEMBER)  ((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem  - offsetof (STRUCT, MEMBER.list\_elem))) |
| **Parameter** | 자기 자신이 포함되어 있는 hash item의 포인터를 찾고자 하는 hash element HASH\_ELEM, hash item의 구조체명 STRUCT, hash item내에서 hash element의 멤버명 MEMBER |
| **Return** | HASH\_ELEM이 포함되어 있는 hash item의 주소값 |
| **Function** | hash element HASH\_ELEM의 주소값을 사용하여 HASH\_ELEM이 들어 있는 구조체 hash item의 주소값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | 새로운 element가 삽입될 hash table h, 삽입될 새로운 element new |
| **Return** | new와 중복되는 element가 존재한다면 중복되는 element return, 아니라면 NULL return |
| **Function** | new를 h에 삽입한다. 만약 중복되는 element가 존재한다면 기존에 존재하던 element를 삭제한 후 new를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | element를 찾을 hash table h, 찾을 hash element e |
| **Return** | 찾은 element가 있다면 해당 element를 return, 아니면 NULL |
| **Function** | hash table h내에서 e와 동일한 element를 찾은 후 반환한다. 없다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | element를 제거할 hash table h, 제거할 element e |
| **Return** | 제거된 element e, e가 h내에 존재하지 않았다면 NULL |
| **Function** | h내에서 e와 동일한 hash element를 찾은 다음 제거한 후 해당 element를 반환한다. 만약 없다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | 어떠한 함수를 적용할 hash table h, h에 적용할 함수 action |
| **Return** | - |
| **Function** | h내의 모든 element에 대하여 action내의 함수를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | hash table의 iteration에 사용되는 iterator i, hash table h |
| **Return** | - |
| **Function** | h내의 element들에 대해서 iterate하기 위해 i를 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*i) |
| **Parameter** | hash table내의 element를 iterate하기 위한 iterator i |
| **Return** | 현재 element의 다음 element, 없다면 NULL |
| **Function** | i를 다음 단계로 보내 hash element의 다음 element에 접근하게 한 후 해당 element를 반환한다. 없다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*i) |
| **Parameter** | hash table내의 iterator i |
| **Return** | iterator에 현재 있는 hash element |
| **Function** | 현재 iterator가 가리키고 있는 hash element를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*h) |
| **Parameter** | 크기를 알고자 하는 hash table h |
| **Return** | hash table의 hash element 수 |
| **Function** | h의 element들의 개수를 세어 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | 비어있는지 확인할 hash table h |
| **Return** | 비어있다면 true, 아니면 false |
| **Function** | h가 아무 element를 가지고 있지 않는지 확인한다. elem\_cnt의 수를 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*buf\_, size\_t size) |
| **Parameter** | hashing할 data buf, buf에 몇 bytes가 있는지 저장하는 size |
| **Return** | buf의 hash 값 |
| **Function** | size 만큼의 bytes의 크기를 가지는 buf의 hash 를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*s\_) |
| **Parameter** | hash 값을 찾고자 하는 string s |
| **Return** | s의 hash 값 |
| **Function** | s의 hash 값을 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int i) |
| **Parameter** | hash 값을 찾을 정수 i |
| **Return** | i의 hash 값 |
| **Function** | i의 hash 값을 반환한다. 이는 hash\_bytes()로 구현한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*find\_bucket (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | element를 찾아낼 hash table, 들어있는 bucket을 찾고자 하는 hash element e |
| **Return** | h내에서 e가 들어있는 bucket |
| **Function** | h내에서 e가 들어있는 bucket을 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | element를 찾아낼 hash table h, element를 찾아낼 h내의 bucket, 찾고자 하는 hash element e |
| **Return** | bucket에서 e와 동일한 element, 없다면 NULL |
| **Function** | h내의 bucket에서 e와 동일한 element를 찾아낸 후 반환한다. 없다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x) |
| **Parameter** | 1로 set되어 있는 lowest order bit을 0으로 만들 수 x |
| **Return** | x AND (x-1) |
| **Function** | x에서 1로 set되어 있는 lowest order bit을 0으로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t is\_power\_of\_2 (size\_t x) |
| **Parameter** | 2의 거듭제곱인지 확인할 수 x |
| **Return** | x가 2의 거듭제곱이라면 true, 아니면 false |
| **Function** | x가 2의 거듭제곱인지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h) |
| **Parameter** | bucket의 수 및 element배치를 바꿀 hash table h |
| **Return** | - |
| **Function** | 제일 최적의 hash table로 h를 설정하기 위해 h의 bucket수를 바꾼다. 또한 element들이 새로 설정된 bucket에 제대로 들어가게끔 한다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | element를 삽입할 hash table h, element가 들어갈 bucket, 새로 삽입될 element e |
| **Return** | - |
| **Function** | h내의 bucket에 e를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | static void remove\_elem (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | element를 제거할 hash table h, 제거될 element e |
| **Return** | - |
| **Function** | h에서 e를 제거한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_idx (size\_t bit\_idx) |
| **Parameter** | 해당 bit가 들어있는 element를 구하려 하는 bit의 index |
| **Return** | bit\_idx번째 bit가 들어있는 element의 index |
| **Function** | bit\_idx를 사용하여 bit가 몇 번째 element에 들어있는지 구한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx) |
| **Parameter** | bit의 index bit\_idx |
| **Return** | bit\_idx번째 bit만 turned on되어 있는 수 |
| **Function** | bit\_idx번째 bit만 turned on되어 있는 수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt (size\_t bit\_cnt) |
| **Parameter** | bit의 수 |
| **Return** | bit\_cnt만큼의 bit가 있을 때 필요한 element의 수 |
| **Function** | bit\_cnt만큼의 bit가 있을 때 필요한 element의 수를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt (size\_t bit\_cnt) |
| **Parameter** | bit의 수 |
| **Return** | bit\_cnt만큼의 bit가 있을 때 필요한 byte의 수 |
| **Function** | bit\_cnt만큼의 bit가 있을 때 필요한 byte의 수를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type last\_mask (const struct bitmap \*b) |
| **Parameter** | bit mask를 생성하는데 필요한 bitmap b |
| **Return** | b의 마지막 element에서 실제로 사용되는 bit들이 1로 set되어있고 나머지는 0으로 설정되어 있는 bit mask |
| **Function** | b의 마지막 element에서 실제로 사용되는 bit들이 1로 set되어있고 나머지는 0으로 설정되어 있는 bit mask를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | bit의 수 bit\_cnt |
| **Return** | bit\_cnt만큼의 bit를 가지는 bitmap, 메모리 할당을 못하면 NULL |
| **Function** | bitmap을 총 bit\_cnt만큼의 bit를 가지게 하고 모든 bit를 false로 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*block, size\_t block\_size ) |
| **Parameter** | 희망하는 bitmap의 bit수 bit\_cnt, 새로 만들어진 bitmap이 들어있을 block, block의 size block\_size |
| **Return** | bit\_cnt만큼의 bit를 가지는 bitmap |
| **Function** | bit\_cnt만큼의 bit를 가지는 bitmap을 생성한 후 반환한다. 해당 bitmap은 block\_size만큼의 byte를 가지는 block에 들어있게 된다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt) |
| **Parameter** | bitmap의 bit수 bit\_cnt |
| **Return** | bit\_cnt만큼의 bit를 가지는 bitmap을 만드는데 필요한 byte수 |
| **Function** | bit\_cnt만큼의 bit를 가지는 bitmap을 만드는데 필요한 byte수를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | 제거하고자 하는 bitmap b |
| **Return** | - |
| **Function** | b내의 메모리를 모두 할당해제한 후 b를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | size를 알고자 하는 bitmap b |
| **Return** | b의 bit의 수 |
| **Function** | b의 bit의 수를 bit\_cnt를 사용하여 구한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | bitmap b, 바꾸고자 하는 b내의 index idx, idx번째 bit를 설정할 Boolean value |
| **Return** | - |
| **Function** | b내의 idx번째 bit를 value로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, b내의 bit index bit\_idx |
| **Return** | - |
| **Function** | b내의 bit\_idx번째 bit를 true로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, b내의 bit index bit\_idx |
| **Return** | - |
| **Function** | b내의 bit\_idx번째 bit를 false로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, b내의 bit index bit\_idx |
| **Return** | - |
| **Function** | b내의 bit\_idx번째 bit를 반대 (0->1, 1->0)로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*b, size\_t idx) |
| **Parameter** | bitmap b, b내의 bit index bit\_idx |
| **Return** | b내의 bit\_idx번째 bit |
| **Function** | b내의 bit\_idx번째 bit를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*b, bool value) |
| **Parameter** | bitmap b, boolean value value |
| **Return** | - |
| **Function** | bitmap내의 모든 bit를 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, boolean value value, b내의 index start, set할 bit의 수 cnt |
| **Return** | - |
| **Function** | b내의 start번째 bit부터 cnt개의 bit를 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, boolean value value, b내의 index start, count할 bit의 수 cnt |
| **Return** | b내의 start번째 bit로부터 cnt개의 bit중 value로 check되어 있는 bit의 수 |
| **Function** | b내의 start번째 bit로부터 cnt개의 bit중 value로 check되어 있는 bit의 수를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, boolean value value, b내의 index start, 확인할 bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 cnt개의 bit중 하나라도 value로 set되어 있다면 true, 아니면 false |
| **Function** | b내에서 start번째 bit부터 cnt개의 bit중 하나라도 value로 set되어 있는지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, b내의 index start, 확인할 bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 cnt개의 bit중 하나라도 true로 set되어 있다면 true, 아니면 false |
| **Function** | b내에서 start번째 bit부터 cnt개의 bit중 하나라도 true로 set되어 있는지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, b내의 index start, 확인할 bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 cnt개의 bit중 모두 false로 set되어 있다면 true, 아니면 false |
| **Function** | b내에서 start번째 bit부터 cnt개의 bit중 모두 false로 set되어 있는지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, b내의 index start, 확인할 bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 cnt개의 bit중 모두 true로 set되어 있다면 true, 아니면 false |
| **Function** | b내에서 start번째 bit부터 cnt개의 bit중 모두 true로 set되어 있는지 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, boolean value value, b내의 index start, 확인할 연속된bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 연속된 cnt개의 bit가 모두 value로 set되어 있을 시 해당 구간의 첫번째 index. 없다면 BITMAP\_ERROR를 return한다. |
| **Function** | b내에서 start번째 bit부터 연속된 cnt개의 bit가 모두 value로 set되어 있을 시 해당 구간의 첫번째 index를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, boolean value value, b내의 index start, 확인할 연속된bit의 수 cnt |
| **Return** | b내에서 start번째 bit부터 연속된 cnt개의 bit가 모두 value로 set되어 있을 시 해당 구간의 첫번째 index. 없다면 BITMAP\_ERROR를 return한다. |
| **Function** | b내에서 start번째 bit부터 연속된 cnt개의 bit가 모두 value로 set되어 있을 시 해당 구간의 첫번째 index를 구한 후, 해당 구간의 bit를 모두 반대로 돌린다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size (const struct bitmap \*b) |
| **Parameter** | 저장하기를 희망하는 bitmap b |
| **Return** | bitmap b가 file에 저장되기 위하여 필요한 byte의 수 |
| **Function** | bitmap b가 file에 저장되기 위하여 필요한 byte의 수를 구한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read (struct bitmap \*b, struct file \*file) |
| **Parameter** | 읽을 bitmap b, b를 담고 있는 file |
| **Return** | b를 성공적으로 file로부터 읽어들였으면 true, 아니면 false |
| **Function** | b를 file로부터 읽어들인다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write (const struct bitmap \*b, struct file \*file) |
| **Parameter** | file에 write할 bitmap b, b를 write할 file |
| **Return** | b를 성공적으로 file로 write했으면 true, 아니면 false |
| **Function** | b를 file로 write한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*b) |
| **Parameter** | 내부 data를 모두 dump할 bitmap b |
| **Return** | - |
| **Function** | b의 내부를 console에 16진수로 dump한다. |