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

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 교수님 |
| 학번 : | 20191300 |
| 이름 : | 남유정 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | element가 셔플될 리스트 1개 |
| **Return** | None |
| **Function** | 하나의 리스트의 랜덤한 element 2개씩 교환해가며 셔플. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | 한 리스트의 2개의 교환될 element |
| **Return** | None |
| **Function** | 하나의 리스트의 두 element \*a, \*b가 swap됨 |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | hash를 가리키는 숫자 i |
| **Return** | hash value |
| **Function** | integer의 hash value를 반환한다. |

|  |  |  |
| --- | --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap\*bitmap, int size) | |
| **Parameter** | 늘리고 싶은 비트맵과 확장 후 비트맵 사이즈 | |
| **Return** | struct bitmap(확장된 비트맵) | |
| **Function** | 기존 bitmap내부의 bit는 유지되고, size로 크기를 변화한 후, 추가된 bit들에 대해서는 0을 넣는다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_head (struct list\_elem \*elem); |
| **Parameter** | head node인지 확인할 list element |
| **Return** | head node인지 아닌지 결과 |
| **Function** | element가 head node인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b, list\_less\_func \*less, void \*aux); |
| **Parameter** | 정렬 순서인지 확인할 두개의 list element |
| **Return** | 정렬되어 있는지 결과 |
| **Function** | list의 두 element가 순서대로 위치한 지 확인 |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_interior (struct list\_elem \*elem); |
| **Parameter** | 내부 node인지 확인할 list element |
| **Return** | 내부 node인지 아닌지 결과 |
| **Function** | element가 list 내부 node인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_tail (struct list\_elem \*elem); |
| **Parameter** | tail node인지 확인할 list element |
| **Return** | tail node인지 아닌지 결과 |
| **Function** | element가 tail node인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | 만들 리스트 |
| **Return** | None |
| **Function** | 새 리스트를 만들고 head node와 tail node를 초기화 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | 첫번째 list element를 확인할 리스트 |
| **Return** | 첫번째 list\_elem |
| **Function** | 입력 받은 리스트의 첫번째 element node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*); |
| **Parameter** | 다음 list element를 확인할 list element |
| **Return** | 입력된 list element의 다음 list element |
| **Function** | list element를 받아서 연결된 다음 list element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*); |
| **Parameter** | tail node를 확인할 리스트 |
| **Return** | tail node인 list element |
| **Function** | 입력 받은 리스트의 tail node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*); |
| **Parameter** | 마지막 list element를 얻을 리스트 |
| **Return** | 마지막 element node |
| **Function** | 리스트를 받아서 그 리스트의 마지막 list element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*); |
| **Parameter** | 앞의 list element를 확인할 리스트 element |
| **Return** | 입력된 element의 앞에 위치한 element |
| **Function** | list\_element\* node를 받아서 연결된 이전 node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*); |
| **Parameter** | head node를 얻을 리스트 |
| **Return** | 리스트의 head node |
| **Function** | 입력 받은 리스트의 head node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*); |
| **Parameter** | head node를 반환할 리스트 |
| **Return** | 리스트의 head node |
| **Function** | 입력 받은 리스트의 head node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*); |
| **Parameter** | tail node를 반환할 리스트 |
| **Return** | 리스트의 tail node |
| **Function** | 입력 받은 리스트의 tail node를 반환 |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem); |
| **Parameter** | 새로 넣을 list element 와 바로 다음에 올 list element before |
| **Return** | None |
| **Function** | 삽입할 list element를 입력 받아 기존 노드 앞쪽에 삽입 |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last); |
| **Parameter** | splice 할 범위를 나타내는 list element 2개, splice한 sublist를 삽입할 위치를 나타내는 list element |
| **Return** | None |
| **Function** | first 부터 last 직전 element까지 element를 지우고 이들을 before element 바로 전에 삽입 |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*, struct list\_elem \*); |
| **Parameter** | 삽입할 리스트와 삽입할 list element |
| **Return** | None |
| **Function** | 입력 받은 리스트의 가장 앞 list element 앞에 새로운 element를 삽입 |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*); |
| **Parameter** | 삽입할 리스트와 삽입할 list element |
| **Return** | None |
| **Function** | 입력 받은 리스트의 가장 끝 element 뒤에 새로운 element를 삽입 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*); |
| **Parameter** | 삭제할 list element |
| **Return** | 삭제된 list element |
| **Function** | 해당 element를 리스트에서 삭제한 뒤 해당 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*); |
| **Parameter** | 가장 앞의 element를 삭제할 리스트 |
| **Return** | 삭제된 list element |
| **Function** | 입력 받은 리스트의 가장 앞 element를 리스트에서 삭제한 뒤 해당 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*); |
| **Parameter** | 가장 뒤의 element를 삭제할 리스트 |
| **Return** | 삭제된 list element |
| **Function** | 입력 받은 리스트의 가장 뒤 element를 리스트에서 삭제한 뒤 해당 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*); |
| **Parameter** | 가장 앞의 element를 반환할 리스트 |
| **Return** | 리스트의 가장 앞의 element |
| **Function** | 입력 받은 리스트의 가장 앞의 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*); |
| **Parameter** | 가장 뒤의 element를 반환할 리스트 |
| **Return** | 리스트의 가장 뒤의 element |
| **Function** | 입력 받은 리스트의 가장 뒤의 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*); |
| **Parameter** | 크기를 얻으려는 리스트 |
| **Return** | 리스트의 element 수 |
| **Function** | 입력 받은 리스트의 element 수 |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*); |
| **Parameter** | empty 유무를 조사할 리스트 |
| **Return** | 리스트의 empty 여부 |
| **Function** | 입력 받은 리스트가 비었는지 확인 후 비어있으면 true를 반환 |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*); |
| **Parameter** | 순서를 역순으로 할 리스트 |
| **Return** | None |
| **Function** | 입력 받은 리스트의 순서를 역순으로 뒤집는다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*  find\_end\_of\_run (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux); |
| **Parameter** | non-decreasing 하게 배열되어 있는 범위의 element 2개, 대소관계 확인 함수 |
| **Return** | non-decreasing 배열의 마지막 element |
| **Function** | 입력 받은 리스트의 element가 non decreasing 순서대로 정렬되어 있는 범위의 마지막 element를 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** | 병합할 두 리스트의 list element |
| **Return** | None |
| **Function** | 두 리스트의 공통된 element를 기준으로 두 리스트의 일부분을 병합 |

|  |  |
| --- | --- |
| **Prototype** | void  list\_sort (struct list \*list, list\_less\_func \*less, void \*aux); |
| **Parameter** | 정렬할 리스트, 대소 관계 함수, 보조 정보 |
| **Return** | None |
| **Function** | 입력 받은 리스트의 element를 크기 순서대로 정렬 |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*,  struct list\_elem \*, list\_less\_func \*, void \*aux); |
| **Parameter** | 정렬되어 있는, 삽입할 element, 대소 관계 함수, 보조 정보 |
| **Return** | None |
| **Function** | 이미 정렬된 리스트를 입력 받아 새로운 element를 정렬 순서에 맞추어 삽입 |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates,  list\_less\_func \*, void \*aux); |
| **Parameter** | 복제할 리스트, 복사본 리스트, 대소 관계 함수, 보조 정보 |
| **Return** | None |
| **Function** | 원본 리스트의 모든 element를 중복 없이 복사 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | 최댓값을 찾을 리스트, 대소 관계 함수, 보조 정보 |
| **Return** | 리스트의 가장 큰 element |
| **Function** | 입력 받은 리스트의 최댓값 element를 찾아서 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | 최솟값을 찾을 리스트, 대소 관계 함수, 보조 정보 |
| **Return** | 리스트의 가장 작은 element |
| **Function** | 입력 받은 리스트의 최솟값 element를 찾아서 반환 |

|  |  |
| --- | --- |
| **Prototype** | static void  swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | 위치를 바꿀 두 개의 노드 |
| **Return** | None |
| **Function** | 두 개의 노드를 받아서 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list); |
| **Parameter** | 순서를 랜덤하게 바꿀 리스트 |
| **Return** | None |
| **Function** | 리스트 내의 element 배열을 랜덤하게 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry(LIST\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next \  - offsetof (STRUCT, MEMBER.next))) |
| **Parameter** | 내부 struct 주소, 외부의 struct, 내부의 struct 멤버 변수 이름 |
| **Return** | 감싸고 있는 struct의 주소 |
| **Function** | 내부 struct 주소를 받아 그 struct를 감싸고 있는 외부의 struct 주소를 반환 |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem \  - offsetof (STRUCT, MEMBER.list\_elem))) |
| **Parameter** | 해쉬 element의 주소, 그것을 감싸고 있는 구조체 종류, 해쉬 element의 멤버 변수 이름 |
| **Return** | 해쉬 element를 감싸고 있는 구조체의 주소 |
| **Function** | 해쉬 element를 받아서 그 element를 감싸고 있는 구조체 주소를 반환 |

|  |  |
| --- | --- |
| **Prototype** | #define list\_elem\_to\_hash\_elem(LIST\_ELEM) \  list\_entry(LIST\_ELEM, struct hash\_elem, list\_elem) |
| **Parameter** | 리스트 element |
| **Return** | 해당 element를 감싸고 있는 해쉬 element의 주소 |
| **Function** | 리스트 element를 받아서 그것을 감싸고 있는 해쉬 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | 새로 할당한 해쉬의 주소, 해쉬 함수로 사용할 함수, 해쉬 elemen의 대소를 비교할 함수, 보조 정보 |
| **Return** | 초기화가 잘 됐는지 여부 |
| **Function** | 해쉬 테이블을 초기화 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 비울 해쉬의 주소, element에 행할 함수 |
| **Return** | None |
| **Function** | 해쉬 테이블의 모든 bucket의 element를 비운다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 제거할 해쉬 구조체의 주소, element에 행할 함수 |
| **Return** | None |
| **Function** | 해쉬 테이블의 element를 없애고 해쉬 테이블 메모리 free |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*); |
| **Parameter** | element 삽입할 해쉬의 주소, 삽입할 해쉬 element |
| **Return** | 삽입된 해쉬 element의 주소 |
| **Function** | 새로운 해쉬 element를 받아서 해쉬 테이블에 넣고, 해당 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*); |
| **Parameter** | element가 삽입될 해쉬 구조체의 주소, 삽입할 해쉬 element |
| **Return** | 삽입된 해쉬 element의 주소 |
| **Function** | 해쉬에 새로운 hash element를 넣는다. 이미 해당 element가 해쉬에 있다면 해당 element를 제거한 후 그 자리를 replace |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 값을 찾을 해쉬 구조체, 찾는 값을 가진 element |
| **Return** | 찾는 값을 가진 해쉬 element의 주소 |
| **Function** | 입력된 해쉬 element와 같은 값을 갖는 element가 해쉬 테이블에 있는지 찾고 있으면 그 element의 위치를 반환 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 값을 삭제할 해쉬 구조체, 삭제할 값을 가진 해쉬 element |
| **Return** | 삭제된 해쉬 element |
| **Function** | 입력된 해쉬 element와 같은 값을 갖는 element가 해쉬 테이블에 있는지 보고 있으면 그 element를 해쉬 테이블에서 삭제, 리턴. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 함수를 적용할 해쉬 구조체, 각 element에 적용할 함수 |
| **Return** | None |
| **Function** | 해쉬 테이블의 모든 element에 함수를 적용. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*); |
| **Parameter** | element의 위치를 받아올 iterator, element의 위치를 알고자 하는 해쉬 구조체 |
| **Return** | None |
| **Function** | iterator를 받아서 해쉬 테이블의 가장 첫 번째 element로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*); |
| **Parameter** | 가리키는 element를 변경할 iterator |
| **Return** | 변경된 iterator 포인터가 가리키는 해쉬 element의 주소 |
| **Function** | iterator가 현재 가리키는 element의 다음 element를 가리킨다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*); |
| **Parameter** | 가리키고 있는 element를 알고자 하는 iterator |
| **Return** | iterator가 가리키고 있는 해쉬 element |
| **Function** | iterator가 가리키는 해쉬 element를 반환 |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*); |
| **Parameter** | 크기를 알고자 하는 해쉬 구조체 |
| **Return** | 해쉬 테이블 내의 element 개수 |
| **Function** | 현재 해쉬 테이블의 element 개수를 리턴 |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*); |
| **Parameter** | 비었는지 알고자 하는 해쉬 구조체 |
| **Return** | 해쉬 테이블이 비었는지의 여부 |
| **Function** | 현재 해쉬 테이블이 비었는지 여부를 불리언 형태로 반환. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*); |
| **Parameter** | 키 값을 얻고자 하는 문자열 |
| **Return** | 문자열의 해쉬 함수의 키 |
| **Function** | 문자열의 해쉬 함수 키를 반환. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int); |
| **Parameter** | 키 값을 얻고자 하는 정수형 |
| **Return** | 정수 데이터의 해쉬 함수의 키 |
| **Function** | 정수 데이터의 키를 알려주는 해쉬 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_value\_elem(const struct hash\_elem\* e, void\* aux); |
| **Parameter** | 키 값을 보려는 해쉬 element, 보조 정보 |
| **Return** | 해당 element의 아이템의 데이터에 대한 키 값 |
| **Function** | 데이터가 정수인 element에 대해 값을 볼 수 있도록 하ㅁ |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less\_value(const struct hash\_elem \*a,  const struct hash\_elem \*b,  void \*aux); |
| **Parameter** | 대소 비교할 해쉬 element 2개 |
| **Return** | a element가 b element보다 작은 지의 여부 |
| **Function** | 두 개의 해쉬 element를 받아서 대소관계를 비교 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_free(struct hash\_elem \*e, void \*aux); |
| **Parameter** | free시킬 해쉬 element, 보조 정보 |
| **Return** | None |
| **Function** | 해쉬 element를 받아서 hash table에서 제거하고, 이를 감싸고 있는 hash\_item을 free한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square (struct hash\_elem \*e, void \*aux); |
| **Parameter** | 함수를 행할 해쉬 element, 추가 정보 |
| **Return** | None |
| **Function** | action function임. 받은 해쉬 element의 데이터를 제곱한다.. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple (struct hash\_elem \*e, void \*aux); |
| **Parameter** | 함수를 행할 해쉬 element, 보조 정보 |
| **Return** | None |
| **Function** | action function. 받은 해쉬 element의 데이터를 세제곱한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | 생성하려는 비트맵의 크기 |
| **Return** | 생성한 비트맵 |
| **Function** | 비트맵 크기를 받아서 해당 크기의 비트맵을 생성 |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \* block, size\_t byte\_cnt); |
| **Parameter** | 비트맵 크기, 블럭의 주소, 블럭의 크기 |
| **Return** | 생성한 비트맵 |
| **Function** | BLOCK\_SIZE의 할당되어 있는 BLOCK에 받은 크기만큼 비트맵을 생성 |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | 크기 |
| **Return** | 바이트 수 |
| **Function** | bitmap\_create\_in\_buf를 사용할 때 필요한 크기를 반환 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*); |
| **Parameter** | 삭제하려는 비트맵 |
| **Return** | None |
| **Function** | 받은 비트맵의 구조체 멤버를 해제하고 해당 비트맵도 해제 |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*); |
| **Parameter** | 크기를 볼 비트맵 |
| **Return** | 비트맵의 크기 |
| **Function** | 압력된 비트맵의 크기 반환 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool); |
| **Parameter** | 행할 비트맵, set할 인덱스, set할 값 |
| **Return** | None |
| **Function** | 비트맵의 idx번째 비트에 bool 값을 입력 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx); |
| **Parameter** | mark할 bitmap, mark할 인덱스 |
| **Return** | None |
| **Function** | bitmap의 idx번째 비트를 true로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx); |
| **Parameter** | reset할 bitmap, reset할 인덱스 |
| **Return** | None |
| **Function** | bitmap의 idx번째 비트를 false로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | 값을 반전할 bitmap, 반전할 인덱스 |
| **Return** | None |
| **Function** | bitmap의 idx번째 비트를 true는 false로 false는 true로 반전. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx); |
| **Parameter** | 값을 확인할 bitmap과 idx |
| **Return** | bool |
| **Function** | bitmap의 idx번째 인덱스 값이 무엇인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool); |
| **Parameter** | set할 bitmap, set할 값 |
| **Return** | None |
| **Function** | bitmap의 모든 비트에 set할 값을 입력 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 값을 설정할 bitmap, 입력을 시작할 인덱스, set할 비트 개수, 설정할 bool값 |
| **Return** | None |
| **Function** | bitmap의 start번째 비트부터 cnt 개의 비트를 bool 값으로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 개수를 셀 bitmap, 확인을 시작할 인덱스, 확인할 비트 개수, 개수를 셀 값 |
| **Return** | 값을 가진 비트의 수 |
| **Function** | bitmap의 start부터 cnt개의 비트 안에 bool 값이 몇개 있는지 확인 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 값이 있는지 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 개수, 확인할 값 |
| **Return** | 값을 가지고 있는지의 여부 |
| **Function** | bitmap의 start부터 cnt개의 비트 안에 bool 값이 존재하는지 여부 확인 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 값을 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 수 |
| **Return** | true가 있는지의 여부 |
| **Function** | bitmap의 start부터 cnt개의 비트 중에 true가ㄴ 있는지 확인 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 수 |
| **Return** | true가 없는지의 여부 |
| **Function** | bitmap의 start부터 cnt개의 비트가 모두 false인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 수 |
| **Return** | 모두 true인지의 여부 |
| **Function** | bitmap의 start부터 cnt개의 비트가 모두 true인지 확인 |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 수, 확인할 값 |
| **Return** | 비트의 인덱스 |
| **Function** | bitmap의 start부터 cnt개의 비트를 확인하며 확인할 값이 있는 첫번째 인덱스를 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 확인할 bitmap, 확인을 시작할 인덱스, 확인할 비트의 수, 확인할 값 |
| **Return** | 반전시킨 비트의 첫 인덱스 |
| **Function** | bitmap의 start부터 cnt개의 비트를 확인할 값에 대해 scan하며 찾는다면 해당 위치부터 cnt개의 비트를 반전시킨다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*); |
| **Parameter** | 출력할 비트맵 |
| **Return** | None |
| **Function** | 비트맵을 출력한다. |