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

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 |
| 학번 : | 20170364 |
| 이름 : | 김수빈 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux); |
| **Parameter** | Const struct list\_elem \*a: 비교하고자 하는 첫 번째 list\_elem const struct list\_elem \*b: 비교하고자 하는 두 번째 list\_elem |
| **Return** | a와 b의 데이터값을 비교하여 b가 a보다 큰 경우 true, 그렇지 않은 경우 false를 반환한다. |
| **Function** | a와 b의 값을 비교한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux); |
| **Parameter** | Const struct hash\_elem \*a: 비교하고자 하는 첫 번째 hash \_elem const struct hash\_elem \*b: 비교하고자 하는 두 번째 hash \_elem |
| **Return** | a와 b의 데이터값을 비교하여 b가 a보다 큰 경우 true, 그렇지 않은 경우 false를 반환한다. |
| **Function** | a와 b의 값을 비교한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func(const struct hash\_elem \*e, void \*aux); |
| **Parameter** | Const struct hash\_elem \*e: hash 를 생성할 hash\_elem void \*aux: 부가 기능 |
| **Return** | e에 대한 hash value를 반환한다. |
| **Function** | e에 대한 hash value를 생성하는 hash function을 call한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list \* find\_named\_list(named\_list \*, char \*); |
| **Parameter** | Named\_list \*head: 리스트 이름의 head char \*name: 찾고자 하는 리스트 이름 |
| **Return** | Name에 해당하는 이름을 가진 list 반환 |
| **Function** | Name에 해당하는 이름을 가진 list를 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* find\_elem\_by\_index(struct list \*, int); |
| **Parameter** | Struct list \*l: 대상 list int idx: 찾고자 하는 인덱스 |
| **Return** | L의 idx번째 인덱스에 해당하는 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 l의idx번째 인덱스에 저장되어 있는 list\_elem을 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash \* find\_named\_hash(named\_hash \*, char \*); |
| **Parameter** | Named\_list \*head: hash 이름의 head char \*name: 찾고자 하는 hash 이름 |
| **Return** | Name에 해당하는 이름을 가진 hash 반환 |
| **Function** | Name에 해당하는 이름을 가진 hash를 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \* find\_named\_bitmap(named\_bitmap \*, char \*); |
| **Parameter** | Named\_list \*head: bitmap 이름의 head char \*name: 찾고자 하는 bitmap 이름 |
| **Return** | Name에 해당하는 이름을 가진 bitmap 반환 |
| **Function** | Name에 해당하는 이름을 가진 bitmap을 찾는다. |

|  |  |
| --- | --- |
| **Prototype** | void free\_cmd(command \*); |
| **Parameter** | command \*cmd: 실행이 종료되어 삭제할 command |
| **Return** | 없음 |
| **Function** | struct \_command 에 할당한 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | command \* get\_cmd(); |
| **Parameter** | 없음 |
| **Return** | 입력받은 명령어와 argument정보를 저장한 구조체 command를 반환한다. |
| **Function** | Standard input으로부터 명령어를 받아 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_quit(command \*new\_cmd, name\_map \*names); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command name\_map \*names: 자료형의 이름이 저장되어 있는 name\_map |
| **Return** | 없음 |
| **Function** | 실행을 종료한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_create(command \*new\_cmd, name\_map \*names); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command name\_map \*names: 자료형의 이름이 저장되어 있는 name\_map |
| **Return** | 없음 |
| **Function** | 해당하는 자료형에 대한 create를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_delete(command \*new\_cmd, name\_map \*names); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command name\_map \*names: 자료형의 이름이 저장되어 있는 name\_map |
| **Return** | 없음 |
| **Function** | 해당하는 자료형에 대한 delete를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_dumpdata(command \*new\_cmd, name\_map \*names); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command name\_map \*names: 자료형의 이름이 저장되어 있는 name\_map |
| **Return** | 없음 |
| **Function** | 해당하는 자료형에 대한 dumpdata를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_list\_push(command \*, struct list \*); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command struct list \*l: 대상 list |
| **Return** | 없음 |
| **Function** | l에 대하여 list\_push\_front 또는 list\_push\_back을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void call\_list\_pop(command \*, struct list \*); |
| **Parameter** | command \*new\_cmd: 실행하고자 하는 command struct list \*l: 대상 list |
| **Return** | 없음 |
| **Function** | l에 대하여 list\_pop\_front 또는 list\_pop\_back을 수행한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | Struct list \*l: 초기화해서 생성하고자 하는 리스트 |
| **Return** | 없음 |
| **Function** | Head와 tail을 가진 리스트로 초기화한다. 이 때 head의 prev와 tail의 next는 NULL을 가리키고, head의 next는 tail을, tail의 prev는 head를 가리킨다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 시작 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 head가 가리키고 있는 list\_elem, 즉 리스트의 첫 번째 list\_elem 을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*); |
| **Parameter** | Struct list\_elem \*elem: 다음 list\_elem 을 찾고자 하는 list\_elem |
| **Return** | Parameter로 받은 list\_elem elem의 다음 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list\_elem elem이 next로 가리키고 있는 list\_elem, 즉 다음 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 tail을 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 tail, 즉 리스트의 맨 끝 list\_elem 을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 오른쪽 기준 처음을 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 tail을 가리키고 있는 list\_elem, 즉 리스트의 tail의 prev를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 list\_elem elem의 이전 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list\_elem elem이 prev로 가리키고 있는 list\_elem, 즉 이전 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 오른쪽 기준 맨 끝을 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 head, 즉 리스트의 맨 처음 list\_elem 을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 head를 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 head, 즉 리스트의 맨 처음 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*); |
| **Parameter** | Struct list \*list: 탐색하고자 하는 대상 리스트 |
| **Return** | Parameter로 받은 리스트 list의 tail을 반환한다. |
| **Function** | Parameter로 받은 리스트 list 내부의 tail, 즉 리스트의 맨 마지막 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*, struct list\_elem \*); |
| **Parameter** | Struct list\_elem \*before: 삽입의 기준점이 되는 list\_elem struct list\_elem \*elem: 새로 리스트에 삽입할 list\_elem |
| **Return** | 없음 |
| **Function** | Before의 바로 직전 list\_elem 으로 elem을 리스트에 삽입한다. 즉, before->prev = elem, elem->next = before 가 되게끔 새로운 list\_elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last); |
| **Parameter** | Struct list\_elem \*before: 삽입의 기준점 struct list\_elem \*first: 삭제할 list\_elem의 시작점 struct list\_elem \*last: 삭제할 list\_elem의 끝점 |
| **Return** | 없음 |
| **Function** | first부터 last까지의 list\_elem을 잘라낸 후, 해당 list\_elem들을 before의 바로 직전에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*, struct list\_elem \*); |
| **Parameter** | Struct list \*list: list\_elem을 삽입할 list struct list\_elem \*elem: 새로 삽입할 list\_elem |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list의 맨 처음에 elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*); |
| **Parameter** | Struct list \*list: list\_elem을 삽입할 list struct list\_elem \*elem: 새로 삽입할 list\_elem |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list의 맨 끝에 elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*); |
| **Parameter** | Struct list\_elem \*elem: 삭제할 list\_elem |
| **Return** | 삭제한 list\_elem의 다음 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list\_elem을 list에서부터 삭제한 후, 삭제한 list\_elem의 다음 list\_elem, 즉 elem->next 에 해당하던 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*); |
| **Parameter** | Struct list \*list: list\_elem을 꺼낼 대상 list |
| **Return** | 리스트에서부터 꺼낸 list\_elem 을 반환한다. |
| **Function** | Parameter로 받은 list에서 맨 처음 list\_elem을 pop한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*); |
| **Parameter** | Struct list \*list: list\_elem을 꺼낼 대상 list |
| **Return** | 리스트에서부터 꺼낸 list\_elem 을 반환한다. |
| **Function** | Parameter로 받은 list에서 맨 뒤 list\_elem을 pop한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*); |
| **Parameter** | Struct list \*list: 탐색할 대상 list |
| **Return** | list의 맨 처음 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list의 첫 번째 element, 즉 head->next 에 해당하는 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*); |
| **Parameter** | Struct list \*list: 탐색할 대상 list |
| **Return** | list의 맨 마지막 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list의 마지막 element, 즉 tail->prev 에 해당하는 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*); |
| **Parameter** | Struct list \*list: 탐색할 대상 list |
| **Return** | List의 크기를 unsigned datatype인 size\_t로 반환한다. |
| **Function** | Parameter로 받은 list에 들어있는 element의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*); |
| **Parameter** | Struct list \*list: 탐색할 대상 list |
| **Return** | List가 빈 리스트인지 여부를 true/false로 반환한다. |
| **Function** | Parameter로 받은 list가 빈 list인지, 즉 list의 begin과 list의 end가 동일한지를 판단하여 boolean값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*); |
| **Parameter** | Struct list \*list: 탐색할 대상 list |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list의 element들의 순서를 역순으로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | Struct list \*list: 탐색할 대상 list list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list의 list\_elem들을, 대소 비교 함수 less와 aux에 의해 정렬한다. |

|  |  |
| --- | --- |
| **Prototype** | static bool  is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux); |
| **Parameter** | Struct list\_elem \*a: 탐색의 시작점 struct list\_elem \*b: 탐색의 끝점 list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | a부터 b까지의 list\_elem이 정렬되어 있는지 여부를 true/false로 반환한다. |
| **Function** | Parameter로 받은 list\_elem a부터 b까지를 탐색하여 a에서 b까지의 데이터가 정렬되어 있는지를 판단한다. 이 때 정렬 여부 판단은 less function의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*  find\_end\_of\_run (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux); |
| **Parameter** | Struct list\_elem \*a: 탐색의 시작점 struct list\_elem \*b: 탐색의 끝점 list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | a부터 b까지의 list\_elem들 중, a에서부터 시작하여 오름차순(non-decreasing)으로 정렬되어 있는 부분집합의 마지막 list\_elem을 반환한다. 이 때 정렬 여부 판단은 less function의 정의에 의하여 이루어진다. |
| **Function** | Parameter로 받은 list\_elem a에서부터 시작하여 오름차순(non-decreasing)으로 정렬되어 있는 최대 길이의 부분집합의 마지막 list\_elem (end of run)을 반환한다. 이 때 정렬 여부 판단은 less function과 aux의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | static void  inplace\_merge (struct list\_elem \*a0, struct list\_elem \*a1b0,  struct list\_elem \*b1,  list\_less\_func \*less, void \*aux) |
| **Parameter** | Struct list\_elem \*a0: 병합할 부분집합 a의 시작점 struct list\_elem \*a1b0: 병합할 부분집합 a의 끝점, b의 시작점 struct list\_elem \*b1: 병합할 부분집합 b의 끝점 list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 a0~a1b0, a1b0~b1에 해당하는 두 부분집합을 병합한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*, struct list\_elem \*,  list\_less\_func \*, void \*aux); |
| **Parameter** | Struct list \*list: 삽입할 대상 list struct list\_elem \*elem: 삽입할 새로운 list\_elem list\_less func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list에, elem을 적절한 위치에 정렬되도록 삽입한다. 이 때 정렬 여부 판단은 less function과 aux의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates,  list\_less\_func \*, void \*aux); |
| **Parameter** | Struct list \*list: 탐색할 대상 list struct list \*duplicates: 중복 list\_elem을 저장할 list list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list를 탐색하며 중복된 data를 가진 인접한 list\_elem에 대하여 두 개의 해당 중복 list\_elem들 중 첫 번째 list\_elem만 남겨두고 나머지 list\_elem들은 list에서 삭제하여 duplicates에 저장한다. 이 때 값이 동일한지 여부의 판단은 less function과 aux의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | Struct list \*list: 탐색할 대상 list list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 최대값에 해당하는 data를 가진 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list를 탐색하며 data의 값이 최대인 list\_elem을 반환한다. 이 때 대소 판단은 less function과 aux의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | Struct list \*list: 탐색할 대상 list list\_less\_func \*less: 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | 최소값에 해당하는 data를 가진 list\_elem을 반환한다. |
| **Function** | Parameter로 받은 list를 탐색하며 data의 값이 최소인 list\_elem을 반환한다. 이 때 대소 판단은 less function과 aux의 정의에 의하여 이루어진다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b); |
| **Parameter** | Struct list\_elem \*a: 바꿀 첫 번째 list\_elem  Struct list\_elem \*b: 바꿀 두 번째 list\_elem |
| **Return** | 없음 |
| **Function** | Parameter로 받은 두 개의 list\_elem a와 b의 위치를 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list); |
| **Parameter** | Struct list \*list: 셔플을 적용할 대상 list |
| **Return** | 없음 |
| **Function** | Parameter로 받은 list에 대하여 list내의 list\_elem들의 순서를 무작위로 섞는다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | Struct hash \*h: 초기화할 대상 hash hash\_hash\_func \*hash: hash function hash\_less\_func: \*less: hash 대소 비교 함수 void \*aux: 부가 기능 |
| **Return** | Hashtable의 초기화에 성공했으면 true를 반환, hashtable이 이미 NULL인 경우 등 초기화를 하지 못했으면 false를 반환한다. |
| **Function** | Parameter로 받은 h에 대하여 h의 모든 원소를 삭제하여 초기화를 진행한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*); |
| **Parameter** | Struct hash \*h: 초기화할 대상 hash hash\_action\_func \*destructor: 메모리 해제 함수 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 h에 대하여 h의 모든 원소를 삭제한다. 이 때 hash\_action\_func인 destructor가 NULL이 아닌 경우 h의 원소들을 제거하면서 destructor의 정의에 따라 할당된 메모리도 함께 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*); |
| **Parameter** | Struct hash \*h: 삭제할 대상 hash hash\_action\_func \*destructor: 메모리 해제 함수 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 h에 대하여 해시 테이블 h 자체를 제거한다. 이 때 hash\_action\_func인 destructor가 NULL이 아닌 경우 h의 원소들을 제거하면서 destructor의 정의에 따라 할당된 메모리도 함께 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*); |
| **Parameter** | Struct hash \*h: hash\_elem을 삽입할 대상 hash struct hash\_elem \*new: 삽입할 새로운 hash\_elem |
| **Return** | 삽입하고자 하는 hash\_elem의 data가 이미 h에 포함되어 있는 경우 기존의 hash\_elem을 반환하고, 포함되어 있지 않은 경우 new를 삽입하고 NULL을 반환한다. |
| **Function** | Parameter로 받은 h에 new에 해당하는 data가 존재하지 않는 경우 new를 삽입하고, 존재하는 경우 삽입하지 않는다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*); |
| **Parameter** | Struct hash \*h: hash\_elem을 삽입할 대상 hash struct hash\_elem \*new: 삽입할 새로운 hash\_elem |
| **Return** | Parameter로 받은 h에 new를 삽입하고, new와 동일한 데이터를 가진 hash\_elem 을 h에서 꺼내 반환한다. |
| **Function** | Parameter로 받은 h에 new에 해당하는 data가 존재하는 경우 new로 교체하고, 기존의 hash\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*); |
| **Parameter** | Struct hash \*h: 탐색할 대상 hash struct hash\_elem \*e: 찾고자 하는 대상 hash\_elem |
| **Return** | 찾고자 하는 hash\_elem 을 반환한다. |
| **Function** | Parameter로 받은 h에 e에 해당하는 hash\_elem을 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*); |
| **Parameter** | Struct hash \*h: 탐색할 대상 hash struct hash\_elem \*e: 찾고자 하는 대상 hash\_elem |
| **Return** | Hash\_elem e를 반환하고, h에 e가 존재하지 않는 경우 NULL을 반환한다. |
| **Function** | E를 찾아 h에서부터 제거하고 제거한 e를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*); |
| **Parameter** | Struct hash \*h: 대상 hash hash\_action\_func \*action: 적용할 액션 함수 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 hash h의 hash\_elem들에 hash\_action\_func를 적용한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*); |
| **Parameter** | Struct hash\_iterator \*i: hash 탐색 인덱스 i struct hash \*h: 대상 h |
| **Return** | 없음 |
| **Function** | Parameter로 받은 hash h를 탐색하여 첫 번째 hash\_elem에 대한 정보를 i에 저장한다. 이 함수를 통해 i를 hash 탐색 인덱스로 사용할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*); |
| **Parameter** | Struct hash\_iterator \*i: hash 탐색 인덱스 i |
| **Return** | 인덱스 i가 가리키고 있는 다음 hash\_elem을 반환한다. |
| **Function** | Hash를 탐색하는 과정에서 탐색 인덱스 i가 다음으로 탐색할 hash\_elem에 대한 정보를 반환한다. For loop에서 일종의 증가연산자로 사용할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*); |
| **Parameter** | Struct hash\_iterator \*i: hash 탐색 인덱스 i |
| **Return** | 인덱스 i가 가리키고 있는 현재 hash\_elem을 반환한다. |
| **Function** | Hash를 탐색하는 과정에서 탐색 인덱스 i가 현재 가리키고 있는 hash\_elem에 대한 정보를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*); |
| **Parameter** | Struct hash \*h: 탐색할 대상 hash |
| **Return** | h에 들어있는 hash\_elem의 개수, 즉 hash의 크기를 unsigned datatype인 size\_t 로 반환한다. |
| **Function** | Parameter로 받은 h의 크기를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*); |
| **Parameter** | Struct hash\* h: 탐색할 대상 hash |
| **Return** | h가 비어있는지 여부를 true/false의 boolean값으로 반환한다. |
| **Function** | Parameter로 받은 h가 비어있는지 여부를 판단하여 비어있다면 true, 비어있지 않고 hash\_elem이 존재한다면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*, size\_t); |
| **Parameter** | const void \*buf\_: 데이터 버퍼 size\_t size: 데이터의 사이즈 |
| **Return** | Buf\_ 에 저장되어있는 size 크기의 byte data에 대한 hash value를 unsigned 형으로 반환한다. |
| **Function** | Buf\_에서 size크기만큼의 byte data를 key로 하여 그에 해당하는 hash value를 반환하는 hash function이다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*); |
| **Parameter** | Const char \*s\_: 문자열 데이터 |
| **Return** | s에 저장되어있는 문자열에 대한 hash value를 unsigned 형으로 반환한다. |
| **Function** | Parameter로 받은 s에 저장된 문자열을 key로 하여 그에 해당하는 hash value를 반환하는 hash function이다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int); |
| **Parameter** | Int i: 정수 데이터 |
| **Return** | i에 저장되어있는 정수에 대한 hash value를 unsigned 형으로 반환한다. |
| **Function** | Parameter로 받은 i에 저장된 정수를 key로 하여 그에 해당하는 hash value를 반환하는 hash function이다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2 (int); |
| **Parameter** | Int i: 정수 데이터 |
| **Return** | i에 저장되어있는 정수에 대한 hash value를 unsigned 형으로 반환한다. |
| **Function** | Parameter로 받은 i에 저장된 정수를 key로 하여 그에 해당하는 hash value를 반환하는 hash function이다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*  find\_bucket (struct hash \*h, struct hash\_elem \*e); |
| **Parameter** | Struct hash \*h: 대상 hash struct hash\_elem \*e: 찾고자 하는 hash\_elem |
| **Return** | e가 속해 있는 h의 bucket을 반환한다. |
| **Function** | Parameter로 받은 e가 h의 어떤 bucket에 속한 데이터인지를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*  find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e); |
| **Parameter** | Struct hash \*h: 대상 hash struct list \*bucket: 대상 bucket struct hash\_elem \*e: 찾고자 하는 hash\_elem |
| **Return** | E를 찾아 e를 반환하고, 만약 e가 없는 경우 NULL을 반환한다. |
| **Function** | Parameter로 받은 h의 bucket에서 e를 찾아 반환하고, 만약 e가 h에 존재하지 않는 경우 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x); |
| **Parameter** | Size\_t x: 대상 데이터 |
| **Return** | x의 가장 낮은 자리 비트를 1에서 0으로 바꾼 값을 반환한다. |
| **Function** | Parameter로 받은 x의 가장 낮은 자리 비트를 1에서 0으로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  is\_power\_of\_2 (size\_t x) |
| **Parameter** | Size\_t x: 대상 데이터 |
| **Return** | x가 제곱수인 경우 true를, 그렇지 않은 경우 false를 반환한다. |
| **Function** | Parameter로 받은 x가 제곱수인지 여부를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | static void  rehash (struct hash \*h) |
| **Parameter** | Struct hash \*h: 대상 hash |
| **Return** | 없음 |
| **Function** | Parameter로 받은 hash h의 bucket 수를 바꾸고 보다 더 효율적인 탐색을 할 수 있도록 다시 해싱을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | static void  insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | Struct hash \*h: 삽입할 대상 hash struct list \*bucket: 삽입할 대상 bucket struct hash\_elem \*e: 삽입하고자 하는 hash\_elem |
| **Return** | 없음 |
| **Function** | Parameter로 받은 e를 h의 bucket에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | static void  remove\_elem (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | Struct hash \*h: 대상 hash struct hash\_elem \*e: 제거하고자 하는 hash\_elem |
| **Return** | 없음 |
| **Function** | Parameter로 받은 h에서 e를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_destructor (struct hash\_elem \*, void \*); |
| **Parameter** | Struct hash\_elem \*e: 대상 hash\_elem void \*aux: 부가기능 |
| **Return** | 없음 |
| **Function** | Parameter로 받은 hash\_elem e에 할당되어있는 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_apply (struct hash\_elem \*, void \*); |
| **Parameter** | Struct hash\_elem \*e: 대상 hash\_elem void \*aux: 부가 기능 |
| **Return** | 없음 |
| **Function** | Aux로 문자열을 받아 해당 문자열의 조건에 부합하는 기능을 e에 적용한다. “square”이 aux로 들어오는 경우 데이터 값에 제곱을, “triple”이 들어오는 경우 세제곱을 수행한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | Size\_t bit\_cnt: 생성하고자 하는 비트맵의 크기 |
| **Return** | 생성한 bitmap을 반환한다. |
| **Function** | Paramter로 받은 bit\_cnt크기의 bitmap을 생성한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*, size\_t byte\_cnt); |
| **Parameter** | Size\_t bit\_cnt: bitmap의 크기 void \*block: 이미 할당되어 있어야 하는 block buffer size\_t block\_size: block에 저장된 데이터의 크기 |
| **Return** | 생성한 bitmap을 반환한다. |
| **Function** | Parameter로 받은 block의 block\_size만큼의 데이터를 버퍼에 유지한채로 bit\_cnt의 크기만큼 bitmap 을 생성한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | Size\_t bit\_cnt: bitmap의 크기 |
| **Return** | Bit\_cnt크기의 bitmap생성을 위해 필요한 byte 수를 반환한다. |
| **Function** | Paramter로 받은 bit\_cnt크기의 bitmap을 생성하기 위해 몇 바이트가 필요한지를 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap |
| **Return** | 없음 |
| **Function** | Paramter로 받은 bitmap b를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*); |
| **Parameter** | Const struct bitmap \*b: 대상 bitmap |
| **Return** | Bitmap b의 크기를 unsigned datatype인 size\_t로 반환한다. |
| **Function** | Parameter로 받은 b의 크기를 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t idx: 대상 인덱스 bool value: 변경하고자 하는 Boolean 값 |
| **Return** | 없음 |
| **Function** | B의 idx번째 비트를 value로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t idx: 대상 인덱스 |
| **Return** | 없음 |
| **Function** | B의 idx번째 비트를 1로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t idx: 대상 인덱스 |
| **Return** | 없음 |
| **Function** | B의 idx번째 비트를 0으로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t bit\_idx: 대상 인덱스 |
| **Return** | 없음 |
| **Function** | B의 bit\_idx번째 인덱스를 1->0 또는 0->1로 뒤집는다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t idx: 대상 인덱스 |
| **Return** | 해당 비트 값을 true(1), false(0)로 반환한다. |
| **Function** | B의 idx번째 비트가 1인지 0인지를 파악하여 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap bool value: 변경하고자 하는 boolean값 |
| **Return** | 없음 |
| **Function** | B의 모든 비트를 value로 변환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 bool value: 변경하고자 하는 Boolean 값 |
| **Return** | 없음 |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지의 비트를 value로 변경한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 bool value: 파악하고자 하는 Boolean 값 |
| **Return** | Value에 해당하는 비트의 개수를 unsigned datatype인 size\_t로 반환한다. |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지 사이에서 value에 해당하는 비트의 개수를 세어 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 bool value: 파악하고자 하는 Boolean 값 |
| **Return** | Value를 포함하고 있는 경우 true, 그렇지 않은 경우 false 반환 |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지 사이에서 value에 해당하는 비트가 존재하는지를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 |
| **Return** | 1을 포함하고 있는 경우 true, 그렇지 않은 경우 false 반환 |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지 사이에서 true(1)에 해당하는 비트가 존재하는지를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 |
| **Return** | 1을 포함하고 있지 않은 경우 true, 그렇지 않은 경우 false 반환 |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지 사이에서 true(1)에 해당하는 비트가 존재하는지를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 |
| **Return** | 0을 포함하고 있지 않은 경우 true, 그렇지 않은 경우 false 반환 |
| **Function** | b에서 start에 해당하는 인덱스부터 cnt개까지 사이에서 false(0)에 해당하는 비트가 존재하는지를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 bool value: 파악하고자 하는 Boolean 값 |
| **Return** | Value에 해당하는 cnt개의 연속된 값이 존재하는 부분집합의 첫 번째 인덱스를 반환한다. 만약 존재하지 않는 경우 BITMAP\_ERROR를 반환한다. |
| **Function** | b에서 cnt개만큼의 value가 연속적으로 존재하는 곳이 있는지를 판단한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap size\_t start: 시작 인덱스 size\_t cnt: 길이 bool value: 파악하고자 하는 Boolean 값 |
| **Return** | Value에 해당하는 cnt개의 연속된 값이 존재하는 부분집합을 찾아 해당 비트들을 모두 뒤집고, 첫 번째 인덱스를 반환한다. 만약 존재하지 않는 경우 BITMAP\_ERROR를 반환한다. |
| **Function** | b에서 cnt개만큼의 value가 연속적으로 존재하는 곳이 있는지를 판단하고 해당 값들을 뒤집는다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size (const struct bitmap \*); |
| **Parameter** | Const struct bitmap \*b: 생성하고자 하는 bitmap |
| **Return** | B를 파일에 생성하는데 필요한 바이트 수를 반환한다. |
| **Function** | B를 파일에 생성하는데 필요한 바이트 수를 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read (struct bitmap \*, struct file \*); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap  Struct file \*file: 읽어들이고자 하는 file |
| **Return** | file로부터 b를 읽어들이는데 성공하면 true, 실패하면 false를 반환한다. |
| **Function** | file에서부터 b를 읽고 성공 여부를 파악한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write (const struct bitmap \*, struct file \*); |
| **Parameter** | Const struct bitmap \*b: 대상 bitmap  Struct file \*file: 읽어들이고자 하는 file |
| **Return** | B를 file에 쓰고, 성공하면 true, 실패하면 false를 반환한다. |
| **Function** | File에 b를 쓰고 성공 여부를 파악한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*); |
| **Parameter** | Const struct bitmap \*b: 대상 bitmap |
| **Return** | 없음 |
| **Function** | B에 저장되어 있는 데이터를 16진수로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size); |
| **Parameter** | Struct bitmap \*b: 대상 bitmap int size: 늘리고자 하는 bitmap 길이 |
| **Return** | 길이를 연장한 새로운 b를 반환하고, 연장에 실패한 경우 NULL을 반환한다. |
| **Function** | Bitmap b를 size의 크기만큼 기존의 데이터를 유지하며 연장한다. |