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

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 교수님 |
| 학번 : | 20150555 |
| 이름 : | 남민혁 |
|  |  |

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | int namecheck(char\* name); |
| **Parameter** | 조사하고자하는 자료구조의 이름 |
| **Return** | 사용자가 생성한 자료들의 array에서 해당 이름이 존재하면 해당 자료구조가 저장된 인덱스를, 없으면 -1 |
| **Function** | 사용자라 질의한 자료구조의 이름의 존재 유무를 판단하고, 그 위치를 array에서 찾아서 인덱스를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void create(struct request req); |
| **Parameter** | 사용자의 명령을 담은 array |
| **Return** | 없음 (void 함수) |
| **Function** | 사용자의 명령에서 첫번째 명령구문이 ‘create’일 경우, 생성을 원하는 자료구조를 생성하여 자료들의 array에 저장하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdata(struct request req); |
| **Parameter** | 사용자의 명령을 담은 array |
| **Return** | 없음 (void 함수) |
| **Function** | 사용자의 명령에서 첫번째 명령구문이 ‘dumpdata’일 경우, 사용자가 확인을 원하는 자료구조의 element들을 콘솔창에 보여주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void delete(struct request req); |
| **Parameter** | 사용자의 명령을 담은 array |
| **Return** | 없음 (void 함수) |
| **Function** | 사용자의 명령에서 첫번째 명령구문이 ‘delete’일 경우, 사용자가 지우기를 원하는 자료구조를 array에서 없애는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void delete\_all(); |
| **Parameter** | 없음 |
| **Return** | 없음 (void 함수) |
| **Function** | 사용자의 명령에서 첫번째 명령구문이 ‘quit’일 경우, 그동안 생성한 모든 자료구조를 없애고, 이를 관리하는 메모리 영역을 해제하는 함수. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry(LIST\_ELEM, STRUCT, MEMBER) |
| **Parameter** | 실제 저장된 데이터를 알고 싶은 List\_elem의 포인터, 알고 싶은 구조체와 해당 구조체에서 list\_elem의 이름, 즉, 조사를 원하는 list\_elem의 포인터와 함께 STRUCT에는 struct list\_item, MEMBER에는 elem으로 지정. |
| **Return** | List\_elem의 데이터인 list\_item의 포인터 |
| **Function** | 실제 데이터가 저장되어 있는 list\_item의 구조체 포인터를 return하는 매크로 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_head (struct list\_elem \*elem) |
| **Parameter** | 조사하고 싶은 리스트의 element 포인터 |
| **Return** | 해당 element가 head이면 true, 아니면 false |
| **Function** | 해당 element가 유효할떄, head이면 true, 아니면 false를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_interior (struct list\_elem \*elem) |
| **Parameter** | 조사하고 싶은 리스트의 element 포인터 |
| **Return** | 해당 element가 head나 tail이 아니면 true, 맞으면 false |
| **Function** | 해당 element가 유효할떄, head나 tail이 아니면 true, 맞으면 false를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_tail (struct list\_elem \*elem) |
| **Parameter** | 조사하고 싶은 리스트의 element 포인터 |
| **Return** | 해당 element가 tail이면 true, 아니면 false |
| **Function** | 해당 element가 유효할떄, tail이면 true, 아니면 false를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | 초기화를 원하는 리스트의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 리스트를 초기화한다. Head 다음에 Tail element가 오도록 리스트를 구성한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | 첫번째 element 알아낼 리스트의 포인터 |
| **Return** | 리스트의 첫번째 element의 주소 (head.next) |
| **Function** | 주어진 리스트의 head가 가리키는 다음 element의 포인터로, 리스트의 첫 번째 element의 포인터를 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*); |
| **Parameter** | 다음 element 구하고 싶은 리스트의 element 포인터 |
| **Return** | 파라미터로 보낸 포인터 다음에 저장된 element를 가리키는 포인터 |
| **Function** | 파리미터로 보낸 포인터가 가리키는 element가 리스트 내부에 있거나 head이면 주어진 list element의 다음 element를 구하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*); |
| **Parameter** | Tail 구하고자하는 리스트의 포인터 |
| **Return** | 리스트의 tail 포인터 |
| **Function** | 리스트의 tail을 가리키는 포인터를 구하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*); |
| **Parameter** | 마지막 element 구하고자하는 리스트의 포인터 |
| **Return** | 마지막 element를 가리키는 포인터 |
| **Function** | Tail의 앞에 저장된 실제 데이터 중 마지막 element의 포인터를 구하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*); |
| **Parameter** | 이전 element를 구하고 싶은 리스트의 element 포인터 |
| **Return** | 파라미터로 보낸 포인터 이전에 저장된 element를 가리키는 포인터 |
| **Function** | 파리미터로 보낸 포인터가 가리키는 element가 리스트 내부에 있거나 tail이면 주어진 list element의 이전 element를 구하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*); |
| **Parameter** | Head를 구하고자 하는 리스트의 포인터 |
| **Return** | 주어진 리스트의 head를 가리키는 포인터 |
| **Function** | 주어진 리스트가 유효하면, head의 포인터를 구하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*); |
| **Parameter** | Head를 구하고자 하는 리스트의 포인터 |
| **Return** | 주어진 리스트의 head를 가리키는 포인터 |
| **Function** | 주어진 리스트가 유효하면, head의 포인터를 구하는 함수 (list\_rend 함수와 동일) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*); |
| **Parameter** | Tail 구하고자하는 리스트의 포인터 |
| **Return** | 리스트의 tail 포인터 |
| **Function** | 리스트의 tail을 가리키는 포인터를 구하는 함수. (list\_end 함수와 동일) |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*, struct list\_elem \*); |
| **Parameter** | 삽입하고 싶은 위치 이전에 있는 element의 포인터 “before”와삽입할 element의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 위치의 기준이 되는 element가 리스트의 내부에 있거나 tail이고, 리스트가 유효하면 삽입할 element를 before 이전 element로 삽입하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last); |
| **Parameter** | 새로운 값을 삽입될 기존 리스트의 위치 바로 앞인 element와 잘려서 다른 곳에 추가될 리스트의 element들 중 첫번째와 마지막 element의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 삽입될 리스트 A와 잘릴 리스트 B가 있을 때, 삽입될 위치를 나타내는 리스트A의 한 element, before가 내부에 있고 tail인 경우에 작동한다. 리스트 A에 삽입될 element는 리스트 B의 first부터 last 까지 A의 before 앞에 잘려서 삽입된다. 이에 따라 리스트 B는 first 전과 last 다음의 element만 남도록 하는 함수다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*, struct list\_elem \*); |
| **Parameter** | 새로운 element 삽입할 리스트와 새로운 element의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 리스트 맨 앞에 element를 삽입하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*); |
| **Parameter** | 새로운 element 삽입할 리스트와 새로운 element의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 리스트 맨 뒤에 element를 삽입하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*); |
| **Parameter** | 리스트에서 제거할 element의 포인터 |
| **Return** | 제거한 element 다음 element를 가리키는 포인터 |
| **Function** | 리스트에서 원하는 element를 제거하고 제거한 element 다음 element를 가리키는 포인터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*); |
| **Parameter** | Pop할 리스트 |
| **Return** | 리스트에서 pop된 맨 앞 element의 포인터 |
| **Function** | 리스트에서 맨 앞 element를 제거하고 제거된 element를 가리키는 포인터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*); |
| **Parameter** | Pop할 리스트 |
| **Return** | 리스트에서 pop된 맨 뒤 element의 포인터 |
| **Function** | 리스트에서 맨 뒤 element를 제거하고 제거된 element를 가리키는 포인터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*); |
| **Parameter** | 첫번째 element 알아낼 리스트의 포인터 |
| **Return** | 리스트의 첫번째 element의 주소 (head.next) |
| **Function** | 주어진 리스트의 head가 가리키는 다음 element의 포인터로, 리스트의 첫 번째 element의 포인터를 return하는 함수 (list\_begin과 비슷한 함수; list가 비어있는 경우 정의되지 않음) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*); |
| **Parameter** | 마지막 element 구하고자하는 리스트의 포인터 |
| **Return** | 마지막 element를 가리키는 포인터 |
| **Function** | Tail의 앞에 저장된 실제 데이터 중 마지막 element의 포인터를 구하는 함수. (list\_rbegin과 비슷한 함수; list가 비어있는 경우 저의되지 않음) |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*); |
| **Parameter** | 사이즈를 구하고자 하는 리스트의 포인터 |
| **Return** | 리스트의 사이즈 |
| **Function** | 리스트의 사이즈를 구해서 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*); |
| **Parameter** | 비었는지 여부를 확인하고자하는 리스트의 포인터 |
| **Return** | 리스트가 비어있는지 여부에 따라 true, false의 Boolean value |
| **Function** | 주어진 리스트가 비었는지 여부를 확인하여 결과를 반환하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static void swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | 위치를 바꾸고 싶은 리스트 내의 두 element 포인터의 주소 |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 두 element의 주소를 서로 뒤바꿈으로써 list 내에서 element의 위치를 swap하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*); |
| **Parameter** | 순서를 반대로 만들고자하는 리스트의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 리스트가 비어있지 않으면 element 순서를 반대로 만들어 주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 리스트 내에서 오름차순으로 확인하고 싶은 부분의 시작과 끝인 element 포인터 두개와 element의 크기를 비교해주는 함수 list\_less\_function과 auxiliary 데이터인 aux |
| **Return** | Element a부터 b까지 모두 오름차순으로 정렬된 경우 true, 아니면 false |
| **Function** | 리스트안에서 특정 부분의 element 들이 오름차순으로 정렬되었는지 확인하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*find\_end\_of\_run (struct list\_elem \*a,  struct list\_elem \*b, list\_less\_func \*less, void \*aux) |
| **Parameter** | 리스트 내에서 오름차순으로 확인하고 싶은 부분의 시작과 끝인 element 포인터 두개와 element의 크기를 비교해주는 함수 list\_less\_function과 auxiliary 데이터인 aux |
| **Return** | Element a부터 b까지 중 이전에 있는 element보다 다음 element가 크거나 같지 않은 원소를 발견하면 해당 elemnt 포인터, 아니면 마지막 원소의 포인터 |
| **Function** | Element a부터 b까지 less 함수를 이용하여 non-decreasing order를 처음으로 만족하지 않는 list element의 포인터를 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | static void inplace\_merge (struct list\_elem \*a0,  struct list\_elem \*a1b0, struct list\_elem \*b1,  list\_less\_func \*less, void \*aux) |
| **Parameter** | Merge하고자 하는 리스트의 element a0, a1b0, b1와 두개의 element 크기를 비교하는 함수 list\_less\_func, auxiliary 데이터인 aux |
| **Return** | 없음 (void 함수) |
| **Function** | a0~a1b0와 a1b0~b1의 element 범위를 merge하는 함수. 이떄 반드시 모든 element가 유효해야하며, non-decreasing order로 정렬되어 있어야 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*,  list\_less\_func \*, void \*aux); |
| **Parameter** | 정렬할 리스트의 포인터와 리스트 내 element를 비교하는 함수 list\_less\_func의 포인터와 auxiliary 데이터인 aux |
| **Return** | 없음 (void 함수) |
| **Function** | list\_less\_func\* less를 이용해서 리스트의 element들을 오름차순으로 정렬해주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*, struct list\_elem \*,  list\_less\_func \*, void \*aux); |
| **Parameter** | 삽입할 대상이 되는 리스트의 포인터와 삽입할 element의 포인터, 리스트 내 element를 비교하는 함수 list\_less\_funct의 포인터와 auxiliary 데이터인 aux |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 element를 정렬된 순서를 유지하며 리스트에 오름차순에 맞게 삽입시켜주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates,  list\_less\_func \*, void \*aux); |
| **Parameter** | 중복 여부를 확인하고자하는 리스트, 중복된 element들을 넣을 duplicates list, element 비교를 위한 함수인 list\_\_less\_function, auxiliary 데이터인 aux |
| **Return** | 없음 |
| **Function** | list에서 중복으로 들어간 데이터에서 하나만 남기고 제거하며 제거하는 데이터들은 duplicates list에 insert해주는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | 최댓값을 구하고자하는 리스트의 포인터, element 비교를 위한 함수 list\_less\_funct, auxiliary 데이터인 aux |
| **Return** | 주어진 리스트 데이터에서 가장 큰 element를 가리키는 포인터 |
| **Function** | 리스트에서 가장 큰 값을 가지고 있는 element의 포인터를 return 하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | 최솟값을 구하고자하는 리스트의 포인터, element 비교를 위한 함수 list\_less\_funct, auxiliary 데이터인 aux |
| **Return** | 주어진 리스트 데이터에서 가장 작은 element를 가리키는 포인터 |
| **Function** | 리스트에서 가장 작은 값을 가지고 있는 element의 포인터를 return 하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_swap(struct list\_elem \*, struct list\_elem \*); |
| **Parameter** | 순서를 바꾸고자하는 element의 포인터 2개 |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 주어진 element의 순서를 바꿔주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* nth\_elem(struct list \*, size\_t); |
| **Parameter** | N번째 element를 구하고자하는 리스트의 포인터와 원하는 순서 |
| **Return** | 파라미터로 보내진 리스트에서 N번째 element의 포인터 |
| **Function** | 주어진 리스트에서 N번째 element를 찾아 포인터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*) |
| **Parameter** | 순서를 무작위로 섞고자하는 리스트의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 리스트의 element 개수만큼 두개의 element를 무작위로 두개씩 골라 순서를 바꾸어주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*,  const struct list\_elem \*, void \*); |
| **Parameter** | 크기를 비교하고 싶은 특정 리스트 내의 element 포인터 두개와 사용자 정의 데이터 |
| **Return** | 두개의 element 중에서 앞쪽의 element가 작은 경우 true, 반대의 경우에는 false |
| **Function** | 리스트 내의 element 크기를 비교하여 결과를 return하는 함수. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) |
| **Parameter** | 실제 저장된 데이터를 알고 싶은 hash\_elem의 포인터, 알고 싶은 구조체와 해당 구조체에서 list\_elem의 이름, 즉, 조사를 원하는 hash\_elem의 포인터와 함께 STRUCT에는 struct hash\_item, MEMBER에는 elem으로 지정. |
| **Return** | hash\_elem의 데이터인 hash\_item의 포인터 |
| **Function** | 실제 데이터가 저장되어 있는 hash\_item의 구조체 포인터를 return하는 매크로 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | 해시테이블과, hash를 위한 함수인 hash\_hash\_func, element의 크기 비교를 위한 hash\_less\_func과 auxiliary 데이터 aux |
| **Return** | 주어진 해시테이블의 bucket이 NULL이 아니면 true, NULL이면 false |
| **Function** | 주어진 해시테이블을 초기화해주는 함수 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 해시테이블과, element 삭제를 위한 destructor 함수 |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블 내의 모든 element를 삭제하고 bucket을 초기화하는 함수로, destructor가 NULL이 아니면 모든 element에 대해서 destructor를 호출하여 해시테이블을 초기화시켜주는 함수 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 해시테이블과, element 삭제를 위한 destructor 함수 |
| **Return** | 없음 (void 함수) |
| **Function** | 내부적으로 hash\_clear 함수가 호출되어서 해시테이블을 없애고, bucket를 해제하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 해시테이블과 insert할 해시테이블의 새로운 element |
| **Return** | 삽입 시 원래 있는 element이면 그 element를, 새로운 element이면 NULL |
| **Function** | 새로운 element를 해시테이블에에 삽입하는 함수로 같은 값이 이미 들어 있으면 넣지 않고, 없는 값이면 해시테이블에 삽입하고 rehash해준다 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 해시테이블과, insert할 해시테이블의 새로운 element |
| **Return** | 삽입 시 원래 있는 element이면 그 element를, 새로운 element이면 NULL |
| **Function** | 새로운 element와 동일한 element가 해시테이블에 이미 있다면 기존 것을 삭제하고 새로운 것을 삽입한다. 그리고 원래 있는 element라면 해당 element를, 새로운 element면 NULL을 return하는 함수. Hash\_insert는 동일한 element를 무시하는 것에 비해, hash\_replace는기존 동일한 element를 제거하고 새로운 것을 insert 한다는 점에서 차이가 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 해시테이블과 찾을 element의 포인터 |
| **Return** | 찾고자하는 element가 있으면 해당 element를 return하고 없으면 NULL |
| **Function** | 찾고자하는 element가 있으면 해당 element를 return하고 없으면 NULL을 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*); |
| **Parameter** | 해시테이블과 삭제할 element 포인터 |
| **Return** | Element와 같은 것이 해시테이블에 있으면 삭제하고 삭제한 element를, 없으면 NULL |
| **Function** | Element와 같은 것이 해시테이블에 있으면 삭제하고 삭제한 element를, 없으면 NULL을 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*); |
| **Parameter** | 해시테이블과 각 element에 적용할 action 내용을 담은 함수 |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블 내의 모든 element에 대해서 action 함수를 적용시켜 element 값을 바꾸는 함수 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*); |
| **Parameter** | Iteration을 통해 해시테이블 내에서 현재 가리키는 element를 담는 hash\_iterator와 해시테이블 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블의 iterator를 초기화해주는 함수로서, iterator가 해시테이블의 제일 앞을 가리키도록 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*); |
| **Parameter** | Iteration을 통해 해시테이블 내에서 현재 가리키는 element를 담는 hash\_iterator |
| **Return** | 현재 element 다음에 있는 element를 return하되, 없으면 NULL |
| **Function** | Hash\_iterator에서 해시테이블 내의 다음 element를 return하되, 없으면 NULL을 return하도록 하는 함수로, 반드시 hash\_first를 사용한 후에 사용해야 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*); |
| **Parameter** | Iteration을 통해 해시테이블 내에서 현재 가리키는 element를 담는 hash\_iterator |
| **Return** | Iterator가 현재 가리키는 element 속의 데이터 |
| **Function** | Iterator가 현재 가리키는 element 속의 데이터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*); |
| **Parameter** | 사이즈 조사하고자하는 해시테이블 |
| **Return** | 해시테이블의 element 개수 |
| **Function** | 파라미터로 보낸 해시테이블의 element 개수를 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*); |
| **Parameter** | 비었는지 여부 조사하고자하는 해시테이블 |
| **Return** | 해시테이블이 비어있는 경우 true, element가 있으면 false |
| **Function** | 해시테이블 내부가 비었는지 여부를 조사하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*, size\_t); |
| **Parameter** | 특정 hash 값을 변환할 값이 들어 있는 buffer의 포인터와 그 크기 |
| **Return** | Buffer 내의 데이터를 hash function으로 찾은 값 |
| **Function** | Buffer 내의 데이터에 hash function을 적용하여 특정 byte size 만큼의 hash 값를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*); |
| **Parameter** | Hash 할 문자열 |
| **Return** | 문자열의 hash 값 |
| **Function** | 문자열의 hash value를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int i); |
| **Parameter** | Hash 할 정수 |
| **Return** | 정수의 hash 값 |
| **Function** | hash\_bytes(&i, sizeof i)를 호출하여 정수의 hash 값을 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int); |
| **Parameter** | Hash 할 정수 |
| **Return** | 정수의 hash 값 |
| **Function** | 새롭게 정의한 hash\_int 함수로서, 주어진 정수의 절댓값을 5로 나누어 그것보다 크거나 같은 최소의 정수 (e.g., 1-5를 1, 6-10을 2를 부여; 자연수 범위에서의 상한)를 hash 값으로 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_hash(const struct hash\_elem \*e, void \*aux); |
| **Parameter** | 해시테이블 내의 element와 auxiliary 데이터 aux |
| **Return** | 해시테이블 내의 element의 데이터의 hash 값 |
| **Function** | Hash function을 이용하여 element의 데이터 값의 hash 값을 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux); |
| **Parameter** | 크기를 비교할 해시테이블 내의 element 두개와 auxiliary 데이터 aux |
| **Return** | a의 값이 b의 값보다 작으면 true, 크면 false |
| **Function** | 해시테이블 내의 두개의 element의 크기를 비교하여 결과를 return하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_square(struct hash\_elem \*e, void \*aux); |
| **Parameter** | 해시테이블 내의 값을 바꿀 element와 auxiliary 데이터 aux |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 해시테이블의 element의 값을 제곱하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_triple(struct hash\_elem \*e, void \*aux); |
| **Parameter** | 해시테이블 내의 값을 바꿀 element와 auxiliary 데이터 aux |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 해시테이블의 element의 값을 3승하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_destructor(struct hash\_elem\* e, void\* aux); |
| **Parameter** | 해시테이블 내의 값을 바꿀 element와 auxiliary 데이터 aux |
| **Return** | 없음 (void 함수) |
| **Function** | 주어진 해시테이블의 element를 hash\_entry를 통해 element 속 hash\_item을 구한 후 메모리 해제하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_elem\_to\_hash\_elem(LIST\_ELEM) |
| **Parameter** | 실제 저장된 데이터를 알고 싶은 hash\_elem (list\_elem)의 포인터, 알고 싶은 구조체와 해당 구조체에서 hash\_elem (list\_elem)의 이름, 즉, 조사를 원하는hash\_item (list\_item)의 포인터와 함께 STRUCT에는 struct hash\_item, MEMBER에는 elem으로 지정. |
| **Return** | hash\_elem (list\_elem)의 데이터인 hash\_item (list\_item)의 포인터 |
| **Function** | 실제 데이터가 저장되어 있는 hash\_item (list\_item)의 구조체 포인터를 return하는 매크로 함수. 사실상 해시테이블이 리스트로 구현되어 있어, 내부적으로는 list\_entry 함수를 활용함. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \* find\_bucket (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 조사를 원하는 해시테이블 h와 bucket. 값을 찾고자 하는 element |
| **Return** | 특정 해시테이블 h에서 파라미터로 보내진 element의 해시값 |
| **Function** | 특정 해시테이블 h에서 파라미터로 보내진 element의 해시값을 찾아 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*find\_elem (struct hash \*h,  struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | 조사를 원하는 해시테이블 h, bucket과 찾고자하는 element |
| **Return** | 원하는 element를 찾으면 해당 element의 포인터를, 존재하지 않으면 NULL |
| **Function** | 해시테이블 h의 bucket에서 원하는 element를 찾는 함수로, 찾으면 해당 element의 포인터를, 존재하지 않으면 NULL을 reutrn하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x) |
| **Parameter** | 조사를 원하는 숫자 x |
| **Return** | X의 lowest-order 비트가 1인 것을 0으로 만든 것을 return |
| **Function** | X의 lowest-order 비트가 1인 것을 0으로 만든 것을 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t is\_power\_of\_2 (size\_t x) |
| **Parameter** | 조사를 원하는 숫자 x |
| **Return** | X가 2의 제곱꼴이면 true, 아니면 false |
| **Function** | X가 2의 제곱꼴인지 확인하여 결과를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h) |
| **Parameter** | Bucket의 개수를 조정하고자하는 해시테이블 h |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블 h에서 가장 적합한 bucket의 개수로 바꿔주는 함수로, 메모리의 부족으로 해시가 제대로 생성되지 않을 수 있으나, 이는 단지 해시테이블이 덜 효율적으로 이루어질 뿐 문제를 발생시키지는 않는다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket,  struct hash\_elem \*e) |
| **Parameter** | 해시테이블 h, 넣고자하는 element e와 넣으려고하는 bucket |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블 h의 bucket에 특정 element e를 넣는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static void remove\_elem (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 해시테이블과 제거하고하는 element의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 해시테이블 h에서 특정 element를 제거하는 함수. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | #define ELEM\_BITS (sizeof (elem\_type) \* CHAR\_BIT) |
| **Parameter** | 조사를 원하는 element type |
| **Return** | 파라미터로 보낸 element의 비트 개수 |
| **Function** | 파라미터로 보낸 element의 비트 개수를 return하는 매크로 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_idx (size\_t bit\_idx) |
| **Parameter** | 변환을 원하는 비트맵의 인덱스 bit\_idx |
| **Return** | 실제 element에서 bit\_idx번째의 인덱스 |
| **Function** | 실제 element에서 bit\_idx번째의 인덱스로 변환하여 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx) |
| **Parameter** | 비트맵의 인덱스 bit\_idx |
| **Return** | Bit\_idx에서만 1인 비트맵(비트마스크) |
| **Function** | Bit\_idx 자리에서만 1이고, 나머지는 0인 비트맵(비트마스크)를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt (size\_t bit\_cnt) |
| **Parameter** | 비트맵의 인덱스 bit\_idx |
| **Return** | Bit\_cnt 비트를 위해서 필요한 element의 개수 |
| **Function** | Bit\_cnt 비트를 위해서 필요한 element의 개수를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt (size\_t bit\_cnt) |
| **Parameter** | 비트맵의 인덱스 bit\_idx |
| **Return** | Bit\_cnt 비트를 위해서 필요한 바이트의 개수 |
| **Function** | Bit\_cnt 비트를 위해서 필요한 바이트의 개수를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type last\_mask (const struct bitmap \*b) |
| **Parameter** | 비트맵의 포인터 b |
| **Return** | 비트맵 b에서 마지막 element만 1이고, 나머지는 0이 되도록 하는 비트마스크 |
| **Function** | 비트맵 b의 마지막 element만 1이고, 나머지는 0이 되도록 하는 비트마스크를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | 초기화하고자하는 비트맵의 크기 bit\_cnt |
| **Return** | 초기화를 성공하면 비트맵의 포인터를 실패하면 NULL |
| **Function** | Bit\_cnt 만큼의 크기를 가진 비트맵을 생성하고, 0으로 초기화하여, 그 포인터를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*, size\_t byte\_cnt); |
| **Parameter** | 생성할 비트맵의 크기 bit\_cnt, 비트맵 동적할당 대신 사용할 block, block의 크기 block\_size |
| **Return** | 새로 생성한 비트맵의 포인터 |
| **Function** | 비트맵을 생성하되 주어진 block을 이용해서 생성하고 0으로 초기화해주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | Buffer에 저장할 비트맵의 크기 bit\_cnt |
| **Return** | Bit\_cnt 만큼의 비트맵을 저장하기 위한 buffer의 크기 |
| **Function** | 주어진 bit\_cnt만큼의 비트맵을 저장하기 위한 buffer의 크기를 return해주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*); |
| **Parameter** | 해제할 비트맵 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵을 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*); |
| **Parameter** | 크기를 알고 싶은 비트맵의 포인터 |
| **Return** | 비트맵의 크기 |
| **Function** | 파라미터로 보낸 비트맵의 포인터를 참조하여, 해당 비트맵의 크기를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool); |
| **Parameter** | Set하고자하는 비트맵의 포인터와 set하고자하는 비트의 인덱스 idx와 set하고 싶은 값을 담은 bool |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵의 특정 비트를 원하는 bool 값으로 set하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx); |
| **Parameter** | 비트맵의 포인터와 값을 1로 바꾸고 싶은 해당 비트맵의 비트의 인덱스 idx |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵의 특정 비트를 1로 바꾸어 주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx); |
| **Parameter** | 비트맵의 포인터와 값을 0으로 바꾸고 싶은 해당 비트맵의 비트의 인덱스 idx |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵의 특정 비트를 0으로 바꾸어 주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | 비트맵의 포인터와 값을 반대로 바꾸고 싶은 해당 비트맵의 비트의 인덱스 idx |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵의 특정 비트를 반대로 바꾸어 주는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx); |
| **Parameter** | 비트맵의 포인터와 값을 조사하고 싶은 비트의 인덱스 idx |
| **Return** | 인덱스 idx에 들어 있는 값 |
| **Function** | 파라미터로 보낸 비트맵의 특정 비트의 값을 조사하여 returnㅎ하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool); |
| **Parameter** | 비트맵의 포인터와 바꾸고 싶은 값 bool |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵의 모든 비트의 값을 bool 값으로 바꾸는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 비트맵의 포인터, 특정 값으로 바꾸고 싶은 비트들의 시작 인덱스 idx와 그것부터의 개수, 바꿀 값 bool |
| **Return** | 없음 (void 함수) |
| **Function** | 파라미터로 보낸 비트맵에서 idx부터 cnt개만큼 bool의 값으로 바꾸는 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 비트맵의 포인터와 특정 값의 개수를 세고 싶은 비트들의 시작 인덱스 idx와 조사를 원하는 비트들의 개수 cnt, 조사할 값 bool |
| **Return** | 특정 비트맵에서 idx부터 cnt개만큼의 비트 중 bool 값을 가진 바트의 개수 |
| **Function** | 특정 비트맵의 idx 비트부터 cnt개 만큼의 비트 중 bool 값을 가진 비트의 개수를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 비트맵의 포인터와 특정 값의 유무를 조사하고 싶은 비트들의 시작 인덱스 idx와 조사를 원하는 비트들의 개수 cnt, 조사할 값 bool |
| **Return** | 특정 비트맵에서 idx부터 cnt개만큼의 비트 중 bool 값을 가지고 있으면 true, 없으면 false |
| **Function** | 특정 비트맵의 idx 비트부터 cnt개 만큼의 비트 중 bool 값의 유무를 조사하여 그 결과를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 비트맵의 포인터와 1의 유무를 조사하고 싶은 비트들의 시작 인덱스 idx와 조사를 원하는 비트들의 개수 cnt |
| **Return** | 특정 비트맵에서 idx부터 cnt개만큼의 비트 중 1을 가지고 있으면 true, 없으면 false |
| **Function** | 특정 비트맵의 idx 비트부터 cnt개 만큼의 비트 중 1의 유무를 조사하여 그 결과를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 비트맵의 포인터와 특정 값의 유무를 조사하고 싶은 비트들의 시작 인덱스 idx와 조사를 원하는 비트들의 개수 cnt |
| **Return** | 특정 비트맵에서 idx부터 cnt개만큼의 비트 모두가 0을 가지고 있으면 true, 아니면 false |
| **Function** | 특정 비트맵의 idx 비트부터 cnt개 만큼의 비트 중 모두 0의 값을 가지는지 조사하여, 그 결과를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | 비트맵 포인터와 조사를 원하는 부분의 시작 인덱스 start와 개수 cnt |
| **Return** | 특정한 비트맵의 start부터 start\_cnt 까지 모두 1이면 true, 아니면 false |
| **Function** | 특정한 비트맵의 start부터 start\_cnt 까지 모두 1이면 true, 아니면 false를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 조사를 원하는 비트맵과 조사하고자 하는 비트의 시작 인덱스 start, 비트의 개수 cnt, 원하는 값 bool |
| **Return** | 특정 비트맵의 start 인덱스부터 연속되게 cnt개 만큼 bool의 값을 가진 그룹이 있으면 시작 index를, 아니면 BITMAP\_ERROR |
| **Function** | 특정 비트맵의 start 인덱스부터 연속되게 cnt개 만큼 bool의 값을 가진 그룹이 있으면 시작 index를, 아니면 BITMAP\_ERROR를 return하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | 조사를 원하는 비트맵과 조사하고자 하는 비트의 시작 인덱스 start, 비트의 개수 cnt, 원하는 값 bool |
| **Return** | 특정 비트맵의 start 인덱스부터 연속되게 cnt개 만큼 bool의 값을 가진 그룹이 있으면 시작 index를, 아니면 BITMAP\_ERROR |
| **Function** | 특정 비트맵의 start 인덱스부터 연속되게 cnt개 만큼 bool의 값을 가진 그룹이 있으면 시작 index를, 아니면 BITMAP\_ERROR를 return하되, 해당 그룹의 값을 반전시키는 기능도 가진 함수. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size (const struct bitmap \*); |
| **Parameter** | 비트맵의 포인터 |
| **Return** | 비트맵에 저장한 바이트 크기 |
| **Function** | 특정 비트맵에 저장한 바이트의 크기를 조사하여 reurn하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read (struct bitmap \*, struct file \*); |
| **Parameter** | 비트맵의 포인터와 파일 포인터 |
| **Return** | 파일로부터 비트맵을 읽어와서 성공하면 true, 실패하면 false |
| **Function** | 파일로부터 비트맵을 읽어와서 성공여부를 reurn하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write (const struct bitmap \*, struct file \*); |
| **Parameter** | 비트맵의 포인터와 파일 포인터 |
| **Return** | 파일로부터 비트맵을 써서 성공하면 true, 실패하면 false |
| **Function** | 파일로부터 비트맵을 써서 성공여부를 reurn하는 함수. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*); |
| **Parameter** | 비트맵의 포인터 |
| **Return** | 없음 (void 함수) |
| **Function** | Hex\_dump 함수를 활용하여 비트맵을 콘솔에 16진법(한줄에 16개씩)으로 dump하는 함수 |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand (struct bitmap \*, int); |
| **Parameter** | 비트맵 크기를 바꾸고 싶은 비트맵의 포인터와 추가하고 싶은 개수 |
| **Return** | 비트맵 크기를 int 만큼 늘린 비트맵의 포인터 |
| **Function** | 파라미터로 받은 비트맵을 int만큼 늘려주고 0으로 초기화하는 함수. |