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

|  |  |
| --- | --- |
| 담당 교수 : | 박 성 용 교수님 |
| 학번 : | 20161595 |
| 이름 : | 배성현 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation(in main.c)**

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem\* a, const struct list\_elem\* b, void\* aux); |
| **Parameter** | const struct list\_elem\* a(비교하고자 하는 element), const struct list\_elem\* b(비교하고자 하는 element), void\* aux |
| **Return** | bool(element a의 data값이 element b의 data값보다 작으면 true, 그렇지 않으면 false를 반환) |
| **Function** | 파라미터로 넘어온 element의 대소관계를 비교하는 기능을 하는 함수로, element a의 data값이 element b의 data값보다 작으면 true, 그렇지 않으면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func(const struct hash\_elem\* e, void\* aux); |
| **Parameter** | const struct hash\_elem\* e(해시연산을 적용하려는 element), void\* aux |
| **Return** | unsigned(해시연산이 적용된 해시값) |
| **Function** | 파라미터로 넘어온 element의 데이터 값에 hash\_int()함수 해시 연산을 적용한 해시 값을 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem\* a, const struct hash\_elem\* b, void\* aux) |
| **Parameter** | const struct hash\_elem\* a(비교하고자 하는 element), const struct hash\_elem\* b(비교하고자 하는 element), void\* aux |
| **Return** | bool(element a의 data값이 element b의 data값보다 작으면 true, 그렇지 않으면 false를 반환) |
| **Function** | 파라미터로 넘어온 element들의 대소관계를 비교하는 기능을 하는 함수로, element a의 data값이 element b의 data값보다 작으면 true, 그렇지 않으면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_delete\_func(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e(삭제하려는 element), void\* aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 삭제하려는 element인 e를 메모리에서 deallocate하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square\_func(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e(data의 값을 2배하려는 element), void\* aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 element e의 data를 2배로 바꾸는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple\_func(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e(data의 값을 3배하려는 element), void\* aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 element e의 data를 3배로 바꾸는 기능을 한다. hash\_hash\_action |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear\_func(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e(삭제하려는 element), void\* aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 삭제하려는 element인 e를 메모리에서 deallocate하는 기능을 한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list\*); |
| **Parameter** | struct list \*list (초기화하려는 리스트) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트를 빈 리스트로 initialize하는 기능을 한다.(리스트의 head.next가 tail을 가리키게 하고, 리스트의 tail.prev가 head를 가리키게 함으로써) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | struct list \*list (확인할 리스트) |
| **Return** | struct list\_elem \*(list의 가장 첫 element) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 가장 첫 element(head.next)를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*); |
| **Parameter** | struct list\_elem \*elem(리스트의 특정한 element) |
| **Return** | struct list\_elem \* (parameter로 들어온 element의 다음 element) |
| **Function** | 파라미터로 넘어온 특정한 element의 다음 element를 반환하여 주는 기능을 하게 되며 만약 파라미터로 넘어온 element가 마지막 element라면 list의 tail을 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*); |
| **Parameter** | struct list \*list(확인할 리스트) |
| **Return** | struct list\_elem \*(리스트의 tail) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 tail을 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*); |
| **Parameter** | struct list \*list(확인할 리스트) |
| **Return** | struct list\_elem \*(리스트의 가장 마지막 element) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 가장 마지막 element(tail.prev, list를 거꾸로 순회했을 때의 가장 첫 element)를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*); |
| **Parameter** | struct list\_elem \*elem(리스트의 특정한 element) |
| **Return** | struct list\_elem \* (parameter로 들어온 element의 이전 element) |
| **Function** | 파라미터로 넘어온 특정한 element의 이전 element를 반환하여 주는 기능을 하게 되며 만약 파라미터로 넘어온 element가 첫 element라면 list의 head를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*); |
| **Parameter** | struct list \*list(확인할 리스트) |
| **Return** | struct list\_elem \*(리스트의 head) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 head(해당 리스트를 거꾸로 순회했을 때의 end)를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*); |
| **Parameter** | struct list \*list(확인할 리스트) |
| **Return** | struct list\_elem \*(리스트의 head) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 head 를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*); |
| **Parameter** | struct list \*list(확인할 리스트) |
| **Return** | struct list\_elem \*(리스트의 tail) |
| **Function** | 파라미터로 넘어온 list가 NULL이 아닐 때에 해당 리스트의 tail 을 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*, struct list\_elem \*); |
| **Parameter** | struct list\_elem \* before(elem를 삽입하고자 하는 위치에 있는 element), struct list\_elem \* elem(삽입하려고 하는 element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 before(삽입하고자 하는 위치에 있는 element) 이전에 elem(삽입하려는 element)을 삽입하는 기능을 한다. 즉 before가 있는 위치에 elem이 삽입되게 되고 before 는 한 칸 뒤로 밀린다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last); |
| **Parameter** | struct list\_elem \*before(특정 list element), struct list\_elem \*first(특정 list element), struct list\_elem \*last(특정 list element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 first부터 last까지의 element들을 리스트에서 삭제하고 삭제한 element들을 before의 이전에 삽입하는 기능을 한다. 즉 first부터 last까지의 element들이 삭제되고 삭제된 element들이 before가 있는 위치에 삽입된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*, struct list\_elem \*); |
| **Parameter** | struct list \*list(삽입할 list), struct list\_elem \*elem(삽입할 element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list의 가장 처음에 elem을 삽입하여 주는 기능을 한다.(elem이 list의 가장 첫 element가 되게 된다.) |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*); |
| **Parameter** | struct list \*list(삽입할 list), struct list\_elem \*elem(삽입할 element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list의 가장 마지막에 elem을 삽입하여 주는 기능을 한다.(elem이 list의 가장 마지막 element가 되게 된다.) |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*); |
| **Parameter** | struct list\_elem \*elem(삭제할 element) |
| **Return** | struct list\_elem \*(삭제된 element의 다음 element, 즉 삭제된 위치에 오게 된 element) |
| **Function** | 파라미터로 넘어온 elem을 리스트에서 삭제하여 주고, 삭제된 element가 원래 가리키고 있던 다음 element를 반환하여 주는 기능을 한다. 만약 삭제할 element가 마지막 element라면 list의 tail을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*); |
| **Parameter** | struct list\* list(첫 element를 삭제하고자 하는 list) |
| **Return** | struct list\_elem \*(삭제된 element, 즉 삭제되기 전의 list에서 가장 첫 element) |
| **Function** | 파라미터로 넘어온 list의 가장 첫 element를 삭제한 뒤 삭제한 element를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*); |
| **Parameter** | struct list\* list(마지막 element를 삭제하고자 하는 list) |
| **Return** | struct list\_elem \*(삭제된 element, 즉 삭제되기 전의 list에서 가장 마지막 element) |
| **Function** | 파라미터로 넘어온 list의 가장 마지막 element를 삭제한 뒤 삭제한 element를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*); |
| **Parameter** | struct list\* list(확인할 list) |
| **Return** | struct list\_elem \*(list의 첫 element) |
| **Function** | 파라미터로 넘어온 list가 비어있지 않을 때에, list의 가장 첫 element를 반환하여주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*); |
| **Parameter** | struct list\* list(확인할 list) |
| **Return** | struct list\_elem \*(list의 마지막 element) |
| **Function** | 파라미터로 넘어온 list가 비어있지 않을 때에, list의 가장 마지막 element를 반환하여주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*); |
| **Parameter** | struct list\* list(사이즈를 확인할 list) |
| **Return** | size\_t(list의 사이즈) |
| **Function** | 파라미터로 넘어온 list가 가지고 있는 element의 개수를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*); |
| **Parameter** | struct list\* list(비어있는지를 확인할 list) |
| **Return** | bool(list가 비어있으면 true, 비어있지 않으면 false) |
| **Function** | 파라미터로 넘어온 list가 비어있는지 비어있지 않은지를 체크하여 만약 list가 비어있으면 true를 반환하고, 비어있지 않다면 false를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*); |
| **Parameter** | struct list\* list(거꾸로 바꿀 list) |
| **Return** | 없음 |
| **Function** | 파라미터로 들어온 list의 순서를 거꾸로 바꾸어 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | struct list \*list(정렬할 list), list\_less\_func \*less(element의 대소관계를 나타내는 less 함수), void \*aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 less함수에 따라 list를 정렬시켜주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*, struct list\_elem \*, list\_less\_func \*, void \*aux); |
| **Parameter** | struct list \*list(삽입할 list), struct list\_elem \*elem(삽입할 element), list\_less\_func \*less(element의 대소관계를 나타내는 less함수), void \*aux |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list를 첫 element부터 순회하여 less함수에서true를 반환하는 element의 위치에 elem을 삽입하여 주는 기능을 한다. 만약 파라미터로 넘어온 list가 less함수에 따라 정렬된 list라면 less함수에 따라 정렬된 적절한 위치에 elem을 삽입하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates, list\_less\_func \*, void \*aux); |
| **Parameter** | struct list \*list(순회할 리스트), struct list \*duplicates(제거된 element들을 추가할 리스트), list\_less\_func \*less(element의 대소관계를 나타내는 less함수), void \*aux |
| **Return** | 없음 |
| **Function** | 파라미터로 들어온 list에서 같은 값을 가지는 연속된 element들중 첫 번째 element만을 제외하고 모두 제거한다. 만약 duplicates가 null이 아니라면 제거된 element들이 duplicates에 추가되게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | struct list \*list(확인할 list), list\_less\_func \*less(element의 대소관계를 나타내는 less함수), void \*aux |
| **Return** | struct list\_elem \*(가장 큰 data를 가지고 있는 element) |
| **Function** | 파라미터로 넘어온 list안에서 less함수에 따라 가장 큰 data를 가지고 있는 element를 반환하여 주는 기능을 한다. 만약 가장 큰 데이터 값이 여러 개 있다면, 가장 빠른 element를 반환하여준다. 또한 list가 비어있다면 list의 tail을 반환하여준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux); |
| **Parameter** | struct list \*list(확인할 list), list\_less\_func \*less(element의 대소관계를 나타내는 less함수), void \*aux |
| **Return** | struct list\_elem \*(가장 작은 data를 가지고 있는 element) |
| **Function** | 파라미터로 넘어온 list안에서 less함수에 따라 가장 작은 data를 가지고 있는 element를 반환하여 주는 기능을 한다. 만약 가장 작은 데이터 값이 여러 개 있다면, 가장 빠른 element를 반환하여준다. 또한 list가 비어있다면 list의 tail을 반환하여준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem\* a, struct list\_elem\* b); |
| **Parameter** | struct list\_elem\* a, struct list\_elem\*b(swap될 두 list element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list\_elem a와 b를 서로 위치를 바꾸어 주는(swap해주는) 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list\* list); |
| **Parameter** | struct list \* list(shuffle 될 리스트) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 list를 무작위로 섞어주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b, list\_less\_func \*less, void \*aux); |
| **Parameter** | struct list\_elem \*a(정렬되었는지 확인할 시작점 element), struct list\_elem \*b(정렬되었는지 확인할 끝점 element), list\_less\_func \*less(element의 대소관계를 비교하는 함수), void \*aux |
| **Return** | bool(element a부터 b까지의 원소가 less함수에 맞게 잘 정렬되어있으면 true를 반환하고, 아니라면 false반환) |
| **Function** | 파라미터로 넘어온 element a부터 b까지가 less함수에 맞게 정렬되어있는지의 여부를 확인하는 기능을 한다. 잘 정렬되어있다면 true를 반환하고 아니라면, false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_head (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem(list의 헤드인지 확인할 element) |
| **Return** | bool(elem이 list의 head이면 true, head가 아니면 false를 반환) |
| **Function** | 파라미터로 넘어온 elem이 list의 head인지의 여부를 확인하는 기능을 한다. elem이 head라면 true를, head가 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_interior (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem(내부 요소인지 확인할 element) |
| **Return** | bool(elem이 내부 요소이면 true, 내부요소가 아니면 false를 반환) |
| **Function** | 파라미터로 넘어온 elem이 내부 요소(head도 아니고, tail도 아닌 list에 속한 element)인지의 여부를 확인하는 기능을 한다. 만약 내부요소라면 true를 아니라면, false를 반환하여 준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_tail (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem(list의 tail인지 확인할 element) |
| **Return** | bool(elem이 list의 tail이면 true, tail이 아니면 false를 반환) |
| **Function** | 파라미터로 넘어온 elem이 list의 tail인지의 여부를 확인하는 기능을 한다. elem이 tail이라면 true를, tail이 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | struct list\_elem \*\*a, struct list\_elem \*\*b(swap할 list element 포인터를 가리키는 포인터들) |
| **Return** | 없음. |
| **Function** | 파라미터로 넘어온 a와 b가 가리키고 있는 list element 포인터를 swap하는 기능을 한다. |

|  |  |
| --- | --- |
| **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(순회 시작점을 의미하는 element), struct list\_elem \*b(순회 끝점을 의미하는 element), list\_less\_func \*less(element의 대소관계를 비교하는 함수), void \*aux |
| **Return** | struct list\_elem \*(처음 감소하는 값을 가지는 element가 있으면 해당 element를 반환하고 없으면 b를 반환) |
| **Function** | a에서부터 b까지의 element들에서 less함수에 따라 처음 값이 감소할 때까지 순회하거나 b까지 갈 때까지 순회하는 기능을 한다. 처음 감소하는 값을 가지는 element가 있으면 해당 element를 반환하며, 없다면 최종적으로 a=b가 되기 때문에 b가 반환되게 된다. |

|  |  |
| --- | --- |
| **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(시작점을 나타내는 element), struct list\_elem \*a1b0(중간점을 나타내는 element), struct list\_elem \*b1(끝점을 나타내는 element), list\_less\_func \*less(element의 대소관계를 비교하는 함수), void \*aux |
| **Return** | 없음 |
| **Function** | a0에서부터 a1b0(제외)까지의 element들과, a1b0부터 b1(제외)까지의 element들을 less함수에 따라 감소하지 않는 순서로 병합하는 기능을 한다. 이 때 두 범위는 비어있지 않아야 하며, less함수에 따라 감소하지 않는 순서로 정렬되어 있어야 한다. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry(LIST\_ELEM, STRUCT, MEMBER) ((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next - offsetof (STRUCT, MEMBER.next))) |
| **Parameter** | LIST\_ELEM(자신을 담고 있는 구조체의 포인터를 알고자 하는 list\_elem), STRUCT(list\_elem을 담고 있는 구조체의 이름), MEMBER(해당 구조체에서의 list\_elem을 나타내는 멤버 이름) |
| **Return** | 파라미터로 들어온 LIST\_ELEM을 담고 있는 구조체의 포인터를 반환 |
| **Function** | 파라미터로 들어온 LIST\_ELEM의 데이터에 접근하기 위하여 LIST\_ELEM을 담고 있는 구조체의 포인터를 반환하여주는 기능을 한다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | struct hash \*h(초기화하려는 해시테이블), hash\_hash\_func \*hash(적용할 해시 함수), hash\_less\_func \*less(element의 대소관계를 나타내는 less 함수), void \*aux |
| **Return** | bool(해시테이블 초기화를 성공하면 true, 실패하면 false를 반환함.) |
| **Function** | 파라미터를 통해 넘어온 해시테이블 h를 초기화 해주고 해시테이블 초기화를 성공하면 true, 실패하면 false를 반환한다. 해당 해시테이블 h는 hash\_hash\_func \*hash함수를 통하여 hash값을 계산하게 되고 hash\_less\_func \*less 함수를 통하여 대소관계를 비교하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*); |
| **Parameter** | struct hash \*h(clear할 해시테이블), hash\_action\_func \*destructor(element가 포함된 hash\_item의 메모리를 deallocate해주는 함수) |
| **Return** | 없음. |
| **Function** | 파라미터로 들어온 hashtable h의 모든 element들을 제거해주는 기능을 한다. 이 때 destructor함수가 NULL이 아니라면 destructor함수를 element 각각에 대하여 호출하여 element들의 메모리를 deallocate해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*); |
| **Parameter** | struct hash \*h(제거할 해시테이블), hash\_action\_func \*destructor(element가 포함된 hash\_item의 메모리를 deallocate해주는 함수) |
| **Return** | 없음 |
| **Function** | 파라미터로 들어온 hashtable h를 제거하는 기능을 한다. 이 때 destructor함수가 NULL이 아니라면 hashtable h안의 모든 element들의 메모리를 deallocate해주게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(삽입할 해시테이블), struct hash\_elem \*new(삽입할 element) |
| **Return** | struct hash\_elem \*(삽입할 element new가 해시테이블 h안에 없다면 NULL을 반환, 있다면 hashtable안에 있는 element를 반환) |
| **Function** | 파라미터로 들어온 hashtable h에 삽입할 원소 new를 삽입하는 기능을 한다. 만약 삽입할 element new와 data값이 같은 element가 이미 hashtable h안에 있다면 new를 삽입하지 않고 hastable h안에 있는 그 element를 반환하게 되고, 없다면 NULL을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(삽입할 해시테이블), struct hash\_elem \*new(삽입할 element) |
| **Return** | struct hash\_elem \*(삽입할 element new가 해시테이블 h안에 없다면 NULL을 반환, 있다면 hashtable안에 있는 element를 반환) |
| **Function** | 파라미터로 들어온 hashtable h에 new를 삽입하는 기능을 한다. 만약 new와 data값이 같은 element가 이미 hashtable h안에 있다면 그 element를 제거 후 new를 hashtable h안에 넣고 제거한 element를 반환하게 되고, 없다면 NULL을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(확인할 해시테이블), struct hash\_elem \*e(찾을 element) |
| **Return** | struct hash\_elem \*(찾을 element가 해시테이블에 있으면 그 element를 반환하고, 없으면 NULL반환) |
| **Function** | 파라미터로 들어온 hashtable h에서 e와 같은 값을 가지는 element를 찾아주는 기능을 한다. 만약 e와 data값이 같은 element가 hashtable안에 있다면 그 element를 반환하게 되고, 없다면 NULL을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(확인할 해시테이블), struct hash\_elem \*e(지울 element) |
| **Return** | struct hash\_elem \*(지울 element가 해시테이블에 있다면 그 element를 반환하고, 없다면 NULL을 반환) |
| **Function** | 파라미터로 들어온 hashtable h에서 e와 data값이 같은 element를 지워주는 기능을 한다. 만약 지울 element가 hashtable안에 있다면 그 element를 반환하게 되고 없다면 NULL을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*); |
| **Parameter** | struct hash \*h(적용할 해시테이블), hash\_action\_func \*action(적용할 action 함수) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 hashtable h의 모든 element에 대해 action함수 내에 구현되어 있는 action이 적용되게 하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*); |
| **Parameter** | struct hash\_iterator \*i(초기화할 해시 iterator), struct hash \*h(확인할 해시테이블) |
| **Return** | 없음 |
| **Function** | 파라미터로 들어온 hash iterator i를 hashtable h를 순회할 수 있게끔 초기화 해주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*); |
| **Parameter** | struct hash\_iterator \*i(특정한 해시 iterator) |
| **Return** | struct hash\_elem \* |
| **Function** | 파라미터로 들어온 hash iterator i가 현재 가리키고 있는 element의 다음 element를 가리키도록 해주고 또 그 element를 반환하여 주는 기능을 한다. 만약 다음 element가 없다면 null을 반환하여 준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*); |
| **Parameter** | struct hash\_iterator \*i(특정한 해시 iterator) |
| **Return** | struct hash\_elem \*(해시 iterator i가 가리키고 있는 element를 반환) |
| **Function** | 파라미터로 넘어온 해시 iterator i가 가리키고 있는 element를 반환해주는 기능을 한다. 만약 해시테이블의 마지막이라면 null을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*); |
| **Parameter** | struct hash \*h(사이즈를 확인할 해시테이블) |
| **Return** | size\_t(해시테이블의 element의 개수를 반환) |
| **Function** | 파라미터로 넘어온 hashtable h가 가지고 있는 element의 개수를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*); |
| **Parameter** | struct hash \*h(비어있는지 확인할 해시테이블) |
| **Return** | bool(해시테이블이 비어있으면 true, 비어있지 않으면 false) |
| **Function** | 파라미터로 넘어온 hashtable h가 비어있는지 비어있지 않은지를 체크하여 만약 비어있으면 true를 반환하고, 비어있지 않다면 false를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*, size\_t); |
| **Parameter** | const void \*buf\_(bytes를 저장하고 있는 주소), size\_t size(buf\_의 사이즈) |
| **Return** | unsigned(해시연산이 적용된 해시 값을 반환) |
| **Function** | buf\_가 가리키는 bytes에 대해 해시 연산을 적용하여 그 해시 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*); |
| **Parameter** | const char \*s\_(해시함수를 적용할 string) |
| **Return** | unsigned(해시연산이 적용된 해시 값을 반환) |
| **Function** | 파라미터로 넘어온 문자열 s\_에 대해 대해 해시 연산을 적용하여 그 해시 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int); |
| **Parameter** | int i(해시함수를 적용할 integer) |
| **Return** | unsigned(해시연산이 적용된 해시 값을 반환) |
| **Function** | 파라미터로 넘어온 정수 i에 대해 대해 해시 연산을 적용하여 그 해시 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i); |
| **Parameter** | int i(해시함수를 적용할 integer) |
| **Return** | unsigned(해시연산이 적용된 해시 값을 반환) |
| **Function** | 파라미터로 넘어온 정수 i에 대해 대해 해시 연산을 적용하여 그 해시 값을 반환한다. 해시연산은 먼저 0<A<1의 범위를 가지는 상수 A(0.618934)에 해시 연산을 적용할 integer i를 곱하고, 곱한 수에서 소수 부분만을 추출한 뒤에, 여기에 상수 m(216)값을 곱한 뒤 이를 내림 연산하였다[H(i)=floor((i\*A % 1)\*m)]. 이 때, A값은 Knuth가 제안한 (=0.618934값(이 값을 사용하면 값이 고르게 분포된다고 제안)을 이용하였고 상수 m값은 2n중 임의로 선택하였다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*find\_bucket (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(element e가 속한 hashtable), struct hash\_elem \*e(확인할 hash element) |
| **Return** | struct list\*(e가 속한 (hashtable h의)bucket을 반환) |
| **Function** | 파라미터로 넘어온 e가 속한 (hashtable h의) bucket을 찾아주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*find\_elem (struct hash \*, struct list \*,struct hash\_elem \*); |
| **Parameter** | struct hash \*h(검색할 hashtable), struct list \*bucket(검색할 bucket), struct hash\_elem \*e(찾고자 하는 데이터를 나타내는 element) |
| **Return** | hash\_elem(element e와 같은 데이터 값을 가지는 element를 bucket에서 찾아 반환, 없으면 NULL 반환) |
| **Function** | 파라미터로 넘어온 e와 같은 데이터 값을 가지는 bucket안의 element를 찾아주는 기능을 한다. 만약 없다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*, struct list \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(element를 추가할 hashtable), struct list \*bucket(element가 들어가게 될 bucket), struct hash\_elem \*e(추가할 element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 hashtable h의 bucket에 element e를 추가하는 기능을 하게 된다. 즉 hashtable h에 element e를 추가하게 된다. 이때 element e는 해당 bucket의 가장 첫 번째 element로 추가된다. |

|  |  |
| --- | --- |
| **Prototype** | static void remove\_elem (struct hash \*, struct hash\_elem \*); |
| **Parameter** | struct hash \*h(삭제할 element가 포함된 hashtable), struct hash\_elem \*e(삭제할 element) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 element e를 hashtable h에서 삭제해 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x); |
| **Parameter** | size\_t x(가장 마지막 bit를 끌 수 x) |
| **Return** | size\_t(x에서 가장 마지막 bit를 끈 수) |
| **Function** | 파라미터로 넘어온 x에서 가장 마지막 bit가 켜져 있을 때에 마지막 bit를 끈 수를 반환하는 기능을 한다. 이 때 x가 0이거나 2의 거듭 제곱이라면 0을 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t is\_power\_of\_2 (size\_t x); |
| **Parameter** | size\_t x(2의 거듭제곱인지 확인할 수 x) |
| **Return** | size\_t(x가 2의 거듭제곱이라면 1을 반환하고 아니라면 0을 반환) |
| **Function** | 파라미터로 넘어온 x가 2의 거듭제곱인지 확인하여 주는 기능을 한다. 만약 x가 2의 거듭제곱이라면 1을 반환하고 아니라면 0을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h); |
| **Parameter** | struct hash \*h(rehash할 해시 테이블) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 hastable h의 bucket의 수를 이상적으로 매칭시키는 기능을 한다. 이는 Out-of-memory때문에 실패할 수도 있으나, 단지 해시에 대한 접근을 비효율적으로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) ((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem - offsetof (STRUCT, MEMBER.list\_elem))) |
| **Parameter** | HASH\_ELEM(자신을 담고 있는 구조체의 포인터를 알고자 하는 hash\_elem), STRUCT(hash\_elem을 담고 있는 구조체의 이름), MEMBER(해당 구조체에서의 hash\_elem을 나타내는 멤버) |
| **Return** | 파라미터로 들어온 HASH\_ELEM을 담고 있는 구조체의 포인터를 반환 |
| **Function** | 파라미터로 들어온 HASH\_ELEM의 데이터에 접근하기 위하여 HASH\_ELEM을 담고 있는 구조체의 포인터를 반환하여주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_elem\_to\_hash\_elem(LIST\_ELEM) list\_entry(LIST\_ELEM, struct hash\_elem, list\_elem) |
| **Parameter** | LIST\_ELEM(자신을 담고 있는 hash element를 알고자 하는list\_elem)) |
| **Return** | 파라미터로 넘어온 list\_elem을 담고 있는 hash element를 반환 |
| **Function** | 파라미터로 넘어온 list\_elem를 가지고 이에 해당하는 hash element를 반환하여 주는 기능을 한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt(생성할 bitmap의 사이즈) |
| **Return** | struct bitmap \*(생성에 성공하면 생성한 bitmap을, 실패하면 NULL을 반환) |
| **Function** | 파라미터로 넘어온 bit\_cnt만큼의 크기의 bitmap을 생성하고, 생성한 bitmap의 모든 bit를 false로 초기화하는 기능을 한다. 생성에 성공하면 생성한 bitmap을, 실패하면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*, size\_t block\_size); |
| **Parameter** | size\_t bit\_cnt(생성할 bitmap의 사이즈), void \*block(특정 버퍼), size\_t block\_size(block의 사이즈) |
| **Return** | struct bitmap \*(생성된 bitmap 반환) |
| **Function** | 파라미터로 넘어온 이미 allocate된 block에 bit\_cnt만큼의 크기의 bitmap을 생성하고, 생성한 bitmap의 모든 bit를 false로 초기화한 후 이를 반환하는 기능을 한다. 이 때 block\_size는 생성할 bitmap의 buffer size보다 같거나 커야 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt(비트맵의 bit 수) |
| **Return** | size\_t(비트맵의 모든 bit를 저장할 수 있게 하는 byte의 수를 반환) |
| **Function** | 파라미터로 넘어온 비트맵의 bit\_cnt를 이용하여 bit\_cnt 만큼의 bit를 모두 저장할 수 있게 하는 byte의 수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*); |
| **Parameter** | struct bitmap \*b(제거할 비트맵) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵을 free하여 제거하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*); |
| **Parameter** | const struct bitmap \*b (비트의 수를 확인할 비트맵) |
| **Return** | size\_t(비트맵의 비트의 수를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b가 가지고 있는 비트의 수를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), size\_t idx(설정하고자 하는 인덱스), bool value(설정 값) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 idx번 째에 해당하는 비트를 value값으로 설정해주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), size\_t idx(설정하고자 하는 인덱스) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 idx번째에 해당하는 비트를 true로 설정해주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), size\_t idx(설정하고자 하는 인덱스) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 idx번째에 해당하는 비트를 false로 설정해주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), size\_t idx(설정하고자 하는 인덱스) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 idx번째에 해당하는 비트를 현재 비트와 반대로 설정해주는 역할(토글해주는)을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t idx(특정 비트의 인덱스) |
| **Return** | bool (해당 비트의 값을 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 idx번째에 해당하는 비트의 값을 반환하여주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), bool value(설정 값) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 모든 비트들을 value값으로 설정하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | struct bitmap \*b(설정할 비트맵), size\_t start(설정을 시작할 인덱스), size\_t cnt(시작지점으로부터 개수), bool value(설정 값) |
| **Return** | 없음 |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 cnt개의 비트들을 모두 value값으로 설정하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(시작지점으로부터 개수), bool value(찾을 값) |
| **Return** | size\_t(value와 같은 값을 가지고 있는 비트의 개수를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 start+cnt인덱스까지의 비트들에서 value와 같은 값을 가지고 있는 비트의 개수를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(시작지점으로부터 개수), bool value(찾을 값) |
| **Return** | bool(value와 같은 값을 가지고 있는 비트가 하나라도 있다면 true, 하나도 없다면 false 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 start+cnt인덱스까지의 비트들에서 value와 같은 값을 가지고 있는 비트가 하나라도 있다면 true, 하나도 없으면 false를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(시작지점으로부터 개수) |
| **Return** | bool(true가 하나라도 있으면 true, 하나도 없다면 false 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 start+cnt인덱스까지의 비트들에서 true 값을 가지고 있는 비트가 하나라도 있다면 true, 하나도 없으면 false를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(시작지점으로부터 개수) |
| **Return** | bool(true가 하나라도 있으면 false, 하나도 없다면 true 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 start+cnt인덱스까지의 비트들에서 true 값을 가지고 있는 비트가 하나라도 있다면 false, 하나도 없으면 true를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(시작지점으로부터 개수) |
| **Return** | bool(모두 true면 true를, 하나라도 false이면 false를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 start+cnt인덱스까지의 비트들이 모두 true값을 가지면 true를, 하나라도 false값을 가지면 false를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(연속된 개수), bool value(찾을 값) |
| **Return** | size\_t(value와 같은 값을 가지는 비트 cnt개가 연속되는 비트그룹들 중 첫 번째 그룹의 시작 index를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 value와 같은 값을 가지는 비트 cnt개가 연속되는 비트그룹들 중 첫 번째 그룹의 시작 index를 반환하는 기능을 한다. 만약 index가 범위 내에 없다면 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵), size\_t start(찾기를 시작할 인덱스), size\_t cnt(연속된 개수), bool value(찾을 값) |
| **Return** | size\_t(value와 같은 값을 가지는 비트 cnt개가 연속되는 비트그룹들 중 첫 번째 그룹의 시작 index를 반환, 범위 내에 없다면 BITMAP\_ERROR를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 start 인덱스부터 value와 같은 값을 가지는 비트 cnt개가 연속되는 비트그룹들 중 첫 번째 그룹의 value값들을 반대로 바꾸고, 그 그룹의 시작 index를 반환하는 기능을 한다. 만약 index가 범위 내에 없다면 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size (const struct bitmap \*); |
| **Parameter** | const struct bitmap \*b(확인할 비트맵) |
| **Return** | size\_t(비트맵을 파일에 저장하기 위한 바이트 수를 반환) |
| **Function** | 파라미터로 넘어온 비트맵을 파일에 저장하기 위한 바이트 수를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read (struct bitmap \*, struct file \*); |
| **Parameter** | struct bitmap \*b(저장할 비트맵), struct file \*file(읽어올 파일) |
| **Return** | bool(읽어오는데 성공하면 true를, 실패하면 false를 반환) |
| **Function** | 파라미터로 넘어온 파일로부터 비트맵을 비트맵 b로 읽어오는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write (const struct bitmap \*, struct file \*); |
| **Parameter** | const struct bitmap \*b(읽어올 비트맵), struct file \*file(저장할 파일) |
| **Return** | bool(쓰는데 성공하면 true를, 실패하면 false를 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b로부터 비트맵을 file에 쓰는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*); |
| **Parameter** | const struct bitmap \*b(출력할 비트맵) |
| **Return** | 없음. |
| **Function** | 비트맵 b의 내용을 console에 16진수로 출력하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap\* bitmap, int size); |
| **Parameter** | struct bitmap\* bitmap(연장할 비트맵), int size(연장할 사이즈) |
| **Return** | struct bitmap\*(연장을 성공하면 연장된 bitmap을, 실패하면 NULL을 반환) |
| **Function** | 기존의 bitmap의 크기에 size만큼 더 연장하여 성공할 시 연장된 bitmap을 반환하는 기능을 한다. 실패할 시에는 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_idx (size\_t bit\_idx); |
| **Parameter** | size\_t bit\_idx(element index를 확인하고자 하는 비트의 비트index) |
| **Return** | size\_t(bit\_idx의 비트를 담고 있는 element의 index를 반환) |
| **Function** | 파라미터로 넘어온 비트 번호 bit\_idx의 비트를 담고 있는 element의 index 를 반환하여 주는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx); |
| **Parameter** | size\_t bit\_idx(켜져있어야 하는 비트의 bit index) |
| **Return** | elem\_type(bit\_idx에 해당하는 비트만 켜져 있는 비트마스크를 반환) |
| **Function** | 파라미터로 넘어온 비트 번호 bit\_idx에 해당하는 비트만 켜져 있는 비트마스크를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt(필요한 element의 수를 계산하고자 하는 bit의 수) |
| **Return** | size\_t(bit\_cnt만큼의 비트들을 위해 필요한 element의 수를 반환) |
| **Function** | 파라미터로 넘어온 bit\_cnt만큼의 비트들을 위해 필요한 element의 수를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt (size\_t bit\_cnt) |
| **Parameter** | size\_t bit\_cnt(필요한 byte의 수를 계산하고자 하는 bit의 수) |
| **Return** | size\_t(bit\_cnt만큼의 비트들을 위해 필요한 byte의 수를 반환) |
| **Function** | 파라미터로 넘어온 bit\_cnt만큼의 비트들을 위해 필요한 byte의 수를 반환하는 기능을 한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type last\_mask (const struct bitmap \*b) |
| **Parameter** | const bitmap \*b(last\_mask를 만들기 위해 확인할 비트맵) |
| **Return** | elem\_type(b의 비트들의 마지막 element에서 실제 사용된 비트들은 1로 설정하고, 나머지는 0으로 설정하는 비트마스크 반환) |
| **Function** | 파라미터로 넘어온 비트맵 b의 비트들의 마지막 element에서 실제로 사용된 비트들은 1로 설정하고, 나머지는 0으로 설정하는 비트마스크를 반환하여 주는 기능을 한다. |