# Project #1 : MyLib


담당 교수 : 박성용 교수님

학번 : 20201654

이름 : 최호진

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

# I. Additional Implementation

| | |
|---|---|
| **Prototype** | int main(); |
| **Parameter** | parameter를 따로 입력 받지 않는다. |
| **Return** | 프로그램이 정상적으로 종료될 때 0을 return한다. |
| **Function** | 메인 함수로서 입력으로 "quit"을 입력 받을 때까지, 계속해서 command를 입력 받는다. command에 따라서 알맞은 함수를 call 하여 작업을 수행한다. |

| | |
|---|---|
| **Prototype** | void list_func(char arg[][30], int option); |
| **Parameter** | 입력 받은 문자열을 띄어쓰기 단위로 잘라 한 단어씩 저장된 arg배열과 option을 인자로 받는다. option은 그 값에 따라 서로 다른 기능을 수행한다.<br>(1: create, 2: dump, 3: delete) |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | doubly linked list type 타입에 대한 create, dumpdata, delete command를 수행한다. |

| | |
|---|---|
| **Prototype** | void bitmap_func(char arg[][30], int option); |
| **Parameter** | 입력 받은 문자열을 띄어쓰기 단위로 잘라 한 단어씩 저장된 arg배열과 option을 인자로 받는다. option은 그 값에 따라 서로 다른 기능을 수행한다.<br>(1: create, 2: dump, 3: delete) |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | bitmap 타입에 대한 create, dumpdata, delete command를 수행한다. |

| **Prototype** | void hash__func(char arg[][30], int option); |
|---|---|
| **Parameter** | 입력 받은 문자열을 띄어쓰기 단위로 잘라 한 단어씩 저장된 arg배열과 option을 인자로 받는다. option은 그 값에 따라 서로 다른 기능을 수행한다.<br>(1: create, 2: dump, 3: delete) |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | hash 타입에 대한 create, dumpdata, delete command를 수행한다. |

| **Prototype** | void create(char* command); |
|---|---|
| **Parameter** | 키보드로 입력 받은 command 문자열 전체를 parameter로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 입력 받은 command를 띄어쓰기 단위로 잘라 2차원 배열에 저장하고, 데이터 타입에 따라 알맞은 자료구조의 함수로 연결한다. |

| **Prototype** | void dumpdata(char* command); |
|---|---|
| **Parameter** | 키보드로 입력 받은 command 문자열 전체를 parameter로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 입력 받은 command를 띄어쓰기 단위로 잘라 2차원 배열에 저장하고, 데이터 타입에 따라 알맞은 자료구조의 함수로 연결한다. |

| **Prototype** | void delete(char* command); |
|---|---|
| **Parameter** | 키보드로 입력 받은 command 문자열 전체를 parameter로 전달받는다. |

| | |
|---|---|
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 입력 받은 command를 띄어쓰기 단위로 잘라 2차원 배열에 저장하고, 데이터 타입에 따라 알맞은 자료구조의 함수로 연결한다. |

| | |
|---|---|
| **Prototype** | void list_swap(struct list_elem *e1, struct list_elem *e2); |
| **Parameter** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체 두 개를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list_elem의 위치를 서로 swap한다. |

| | |
|---|---|
| **Prototype** | void list_shuffle(struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list 내부의 원소들을 random으로 순서를 섞는다. |

| | |
|---|---|
| **Prototype** | struct list_table* list_find(char* name); |
| **Parameter** | 찾고자하는 list의 이름 문자열을 파라미터로 전달받는다. |
| **Return** | llist의 이름과 list 간의 관계를 나타내는 포인터를 지닌 struct list_table을 반환한다. 즉, list를 원소로 지닌 struct이다. 또한 반환하는 값은 list_table 내에 name과 동일한 이름을 지닌 list 원소이다. 없다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 name과 동일한 이름인 list를 찾아 반환한다. |

| **Prototype** | bool compare_func (const struct list_elem *a, const struct list_elem *b, void *aux); |
|---|---|
| **Parameter** | 비교할 list의 두 원소(a, b)와 aux를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, a의 data가 b의 data보다 값이 작다면 true, 그 외엔 false를 return한다. |
| **Function** | 파라미터로 전달받은 두 list의 원소들을 비교하여 a가 b보다 더 작은 값을 가지는지 확인하고 결과를 반환한다. |

| **Prototype** | void list_dump(struct list* list); |
|---|---|
| **Parameter** | dump할 list를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list의 모든 원소를 dump한다. 이때 list_entry 매크로 함수를 이용해 list를 구성하는 원소를 구해 출력한다. |

| **Prototype** | void list_command(char* command); |
|---|---|
| **Parameter** | 키보드로 입력받은 command 한 줄을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 전달받은 파라미터를 띄어쓰기 단위로 잘라 문자열 배열에 저장하고, 미리 command 명들을 저장해둔 문자열 배열과 비교해 각 command에 해당하는 배열 내 위치를 구한다. 이 값을 이용해 switch case문에서 각 command에 맞는 작업을 수행한다. |

| **Prototype** | unsigned hash_int_2 (int i); |
|---|---|
| **Parameter** | 변환할 값을 파라미터로 전달받는다. |
| **Return** | unsigned 타입을 return하는데, 함수를 통해 변환된 값을 반환한다. |

| | |
|---|---|
| **Function** | 파라미터로 전달받은 값들이 하나의 bucket에만 중복되어 들어가지 않도록 적절한 연산을 통해 값을 변환한다. |

| | |
|---|---|
| **Prototype** | void hash_action_triple(struct hash_elem *e, void *aux); |
| **Parameter** | data와 struct list_elem 구조체를 지닌 hash_elem과 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 element를 list_elem_to_hash_elem을 이용해 data 값을 찾아 세제곱한다. |

| | |
|---|---|
| **Prototype** | void hash_action_square(struct hash_elem *e, void *aux); |
| **Parameter** | data와 struct list_elem 구조체를 지닌 hash_elem과 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 element를 list_elem_to_hash_elem을 이용해 data 값을 찾아 제곱한다. |

| | |
|---|---|
| **Prototype** | void hash_dump(struct hash* hash); |
| **Parameter** | dump할 hash를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 hash의 전체 원소를 return한다. 이때 list_elem_to_hash_elem 매크로 함수를 이용해 hash의 원소를 찾는데 이 함수는 list_entry 매크로 함수를 hash에 맞게 다시 정의한 함수이다. |

| | |
|---|---|
| **Prototype** | struct hash_table* hashtable_find(char* name); |
| **Parameter** | 찾고자하는 hash의 이름 문자열을 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | hash의 이름과 hash 간의 관계를 나타내는 포인터를 지닌 struct hash_table을 반환한다. 즉, list를 원소로 지닌 struct이다. 또한 반환하는 값은 hash_table 내에 name과 동일한 이름을 지닌 hash 원소이다. 없다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 name과 동일한 이름인 hash를 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | unsigned hash_function(const struct hash_elem *e, void *aux); |
| **Parameter** | data의 값을 구할 hash_elem을 파라미터로 전달받는다. aux는 parameter로 받지만 역할이 없다 |
| **Return** | unsigned type을 return하는데, 이때 return하는 값은 e의 data 값이다. |
| **Function** | 파라미터로 전달받은 element를 list_elem_to_hash_elem을 이용해 data 값을 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | bool hash_less (const struct hash_elem *a, const struct hash_elem *b, void *aux); |
| **Parameter** | 함수기능을 수행할 hash_elem 두 개와 aux를 파라미터로 전달받는다. |
| **Return** | bool type으로 a와 b의 데이터 값을 비교해 a가 작다면 true, 크다면 false를 return한다. |
| **Function** | a와 b의 데이터 값을 비교해 bool type으로 반환한다. |

| | |
|---|---|
| **Prototype** | void hash_free (struct hash_elem* e, void *aux); |
| **Parameter** | free할 hash_elem과 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 받은 hash_elem을 free한다. |

| **Prototype** | void hash_command(char* command); |
|---|---|
| **Parameter** | 키보드로 입력받은 command 한 줄을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 전달받은 파라미터를 띄어쓰기 단위로 잘라 문자열 배열에 저장하고, 미리 command 명들을 저장해둔 문자열 배열과 비교해 각 command에 해당하는 배열 내 위치를 구한다. 이 값을 이용해 switch case문에서 각 command에 맞는 작업을 수행한다. |

| **Prototype** | struct bitmap *bitmap_expand(struct bitmap *bitmap, int size); |
|---|---|
| **Parameter** | 함수를 수행(expand)할 비트맵과 그 크기(size)를 파라미터로 전달받는다. |
| **Return** | struct bitmap 타입을 return하는데, 이때 return하는 값은 전달받은 bitmap의 사이즈가 expand된 bitmap이다. |
| **Function** | 파라미터로 전달받은 bitmap의 기존 크기에 size를 더해 확장된 bitmap을 return한다. |

| **Prototype** | struct bitmap_table *bitmap_find(char* name); |
|---|---|
| **Parameter** | 찾고자하는 bitmap의 이름 문자열을 파라미터로 전달받는다. |
| **Return** | bitmap의 이름과 bitmap 간의 관계를 나타내는 포인터를 지닌 struct bitmap_table을 반환한다. 즉, list를 원소로 지닌 struct이다. 또한 반환하는 값은 bitmap_table 내에 name과 동일한 이름을 지닌 bitmap 원소이다. 없다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 name과 동일한 이름인 bitmap를 찾아 반환한다. |

| **Prototype** | void bitmap_dumpdata(struct bitmap* bitmap); |
|---|---|

| | |
|---|---|
| **Parameter** | dump할 bitmap을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 받은 bitmap을 구성하는 전체 bit를 dump한다. true는 1, false는 0으로 출력한다. |

| | |
|---|---|
| **Prototype** | void bitmap_command(char* command); |
| **Parameter** | 키보드로 입력받은 command 한 줄을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 전달받은 파라미터를 띄어쓰기 단위로 잘라 문자열 배열에 저장하고, 미리 command 명들을 저장해둔 문자열 배열과 비교해 각 command에 해당하는 배열 내 위치를 구한다. 이 값을 이용해 switch case문에서 각 command에 맞는 작업을 수행한다. |

## II. List

| | |
|---|---|
| **Prototype** | list_entry(LIST_ELEM, STRUCT, MEMBER); |
| **Parameter** | list_elem과, 바꾸고자 하는 struct type과 struct 내에 선언된 list_elem의 변수명을 파라미터로 전달받는다. |
| **Return** | list_elem을 원하는 struct로 바꿔서 return해준다. |
| **Function** | list_elem을 member로 가지고 있는 list_item을 찾는 함수로 list_elem의 data값을 알 수 있다. |

| | |
|---|---|
| **Prototype** | static inline bool is_head (struct list_elem *elem); |
| **Parameter** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | Bool type의 함수로 true 또는 false를 return한다.<br>전달받은 파라미터가 list의 head라면 true, head가 아니라면 false를 return한다. |

| | |
|---|---|
| **Function** | 파라미터로 받은 elem이 list의 head인지 확인한다. |

| | |
|---|---|
| **Prototype** | static inline bool is_interior (struct list_elem *elem); |
| **Parameter** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | Bool type의 함수로 true 또는 false를 return한다.<br>전달받은 파라미터가 list의 head 또는 tail이 아니라면 true, head 또는 tail이라면 false를 return한다. |
| **Function** | 파라미터로 받은 elem이 list의 내부 원소인지 (head 또는 tail이 아닌지) 확인한다. |

| | |
|---|---|
| **Prototype** | void list_init (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list를 초기화한다. list의 head의 prev와 tail의 next를 NULL로 하고, head와 tail을 연결한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_begin (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 시작 element(head의 next)를 return한다. |
| **Function** | 파라미터로 전달받은 list의 첫번째 element(head의 next)를 찾아서 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_next (struct list_elem *elem); |

| | |
|---|---|
| **Parameter** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 파라미터로 전달받은 element의 다음 element를 return한다. |
| **Function** | 파라미터로 전달받은 원소의 다음 원소를 return한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_end (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 tail을 return한다. |
| **Function** | 파라미터로 전달받은 list의 tail을 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_rbegin (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 마지막 element(tail의 prev, 오른쪽에서 첫번째)를 return한다. |
| **Function** | 파라미터로 전달받은 list의 마지막 element(tail의 prev)을 찾아서 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_prev (struct list_elem *elem); |
| **Parameter** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 |

| | |
|---|---|
| | struct list_elem 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 파라미터로 전달받은 element의 prev element를 return한다. |
| **Function** | 파라미터로 전달받은 원소의 이전 원소를 return한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_rend (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 첫번째 element(head의 next, 오른쪽에서 마지막)를 return한다. |
| **Function** | 파라미터로 전달받은 list의 첫번째 element(head의 next)를 찾아서 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_head (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 head를 return한다. |
| **Function** | 파라미터로 전달받은 list의 head를 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_tail (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list의 element간의 관계를 나타내는 prev, next 포인터를 지닌 |

| | |
|---|---|
| | struct list_elem 타입의 구조체를 return 하는데, 이 때 return하는 값은 list의 tail를 return한다. |
| **Function** | 파라미터로 전달받은 list의 tail를 반환한다. |

| | |
|---|---|
| **Prototype** | void list_insert (struct list_elem *before, struct list_elem *elem); |
| **Parameter** | insert할 원소의 위치의 before element와, insert할 element를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 before element의 다음 자리에 새 element를 추가한다. |

| | |
|---|---|
| **Prototype** | void list_splice (struct list_elem *before, struct list_elem *first, struct list_elem *last); |
| **Parameter** | splice될 list 자리의 before element와 splice할 list의 시작과 끝 element를 각각 before, first, last의 변수명으로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 before의 다음 자리에 다른 list의 first부터 last 전까지 원소를 추가한 뒤 삭제한다. |

| | |
|---|---|
| **Prototype** | void list_push_front (struct list *list, struct list_elem *elem); |
| **Parameter** | elem을 추가할 list와 추가할 elem을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list의 head를 찾아 그 뒤에 elem을 insert한다. 다시 말해 list의 가장 앞에 원소를 추가한다. |

| | |
|---|---|
| **Prototype** | void list_push_back (struct list *list, struct list_elem *elem); |
| **Parameter** | elem을 추가할 list와 추가할 elem을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list의 tail을 찾아 그 앞에 elem을 insert한 |

| | |
|---|---|
| | 다. 다시 말해 list의 가장 뒤에 원소를 추가한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_remove (struct list_elem *elem); |
| **Parameter** | 제거하고자 하는 list의 elem을 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 제거한(파라미터로 전달받은) 원소의 다음 원소이다. |
| **Function** | 파라미터로 전달받은 원소를 list에서 제거하고 그 다음 원소를 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_pop_front (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list의 첫 원소(head의 next)이다. |
| **Function** | 파라미터로 전달받은 list의 front를 찾고 이를 제거한다. 그 다음 찾은 값을 반환해준다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_pop_back (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list의 마지막 원소(tail의 prev)이다. |
| **Function** | 파라미터로 전달받은 list의 back을 찾고 이를 제거한다. 그 다음 찾은 값을 반환해준다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_front (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list의 가장 첫 원소(head의 next)이다. |
| **Function** | 파라미터로 전달받은 list의 front를 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_back (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list의 마지막 원소(tail의 prev)이다. |
| **Function** | 파라미터로 전달받은 list의 back를 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | size_t list_size (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이 때 return하는 값은 list가 보유한 원소의 개수이다. |
| **Function** | 파라미터로 전달받은 list의 원소의 개수(list_begin부터 end까지)를 반환한다. |

| | |
|---|---|
| **Prototype** | bool list_empty (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | bool type을 return하는데, list가 비어있을 땐 true, 비어있지 않다면 false를 return한다. |
| **Function** | 파라미터로 전달받은 list가 비어있는지 확인한다. list의 begin과 end를 비교해 이들이 같은 지 확인함으로써 수행한다. |

| | |
|---|---|
| **Prototype** | static void swap (struct list_elem **a, struct list_elem **b); |

| | |
|---|---|
| **Parameter** | swap할 list_elem 들의 주소를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 element들을 swap한다. |

| | |
|---|---|
| **Prototype** | void list_reverse (struct list *list); |
| **Parameter** | list의 head와 tail의 정보를 가지고 있는 struct list 타입의 구조체를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list의 원소의 순서를 거꾸로 만든다. |

| | |
|---|---|
| **Prototype** | static bool is_sorted (struct list_elem *a, struct list_elem *b, list_less_func *less, void *aux); |
| **Parameter** | sort 되어있는지 확인할 list의 시작 element(a), 끝 element(b)를 파라미터로 받고 이들 사이 원소들의 대소를 비교할 함수를 less로 전달받는다. 또한 aux를 파라미터로 전달받는데, 이때 aux는 less 함수의 인자로 사용된다. |
| **Return** | bool type을 return하는데, 처음부터 마지막 원소까지 오름차순으로 정렬되어 있다면 true, 그렇지 않다면 false를 return한다. |
| **Function** | a, b 사이의 원소를 less함수를 이용해 정렬되어 있는지 확인한다. 첫 원소(a)부터 마지막 원소(b)까지 list_next를 이용해 이동하며 a와 a의 prev를 비교한다. |

| | |
|---|---|
| **Prototype** | static struct list_elem *find_end_of_run (struct list_elem *a, struct list_elem *b, list_less_func *less, void *aux); |
| **Parameter** | 함수를 수행하고자 하는 list의 시작 원소(a)와 끝 원소(b), 대소를 비교할 함수 less와 less에 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | list_elem 타입의 구조체를 return하는데, 이때 return하는 값은 a와 b 사이에서 오름차순이 만족되지 않는 첫번째 원소를 return |

| | |
|---|---|
| | 한다. 만약 오름차순으로 정렬되어 있다면 b를 return한다. |
| **Function** | list 내의 원소 a부터 b까지 less 함수를 이용해 비교하며 오름차순으로 정렬되어 있는지 확인한다. 만약 정렬되어 있지 않다면 이를 만족하지 않는 첫 원소를 return한다. |

| | |
|---|---|
| **Prototype** | static void inplace_merge (struct list_elem *a0, struct list_elem *a1b0, struct list_elem *b1, list_less_func *less, void *aux); |
| **Parameter** | 함수를 수행하고자 하는 list의 시작 원소(a)와 중간(a1b0), 끝 원소(b), 대소를 비교할 함수 less와 less에 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | a0에서 a1b0의 원소 사이에 a1b0에서 b1까지의 원소를 넣는다. 이때 list는 모두 오름차순으로 정렬되어 있다. |

| | |
|---|---|
| **Prototype** | void list_sort (struct list *list, list_less_func *less, void *aux); |
| **Parameter** | sort할 list와 insert할 elem, 대소를 비교할 함수 less와 less에 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list를 오름차순으로 정렬한다. find_end_of_run과 implace_merge 함수를 이용해 이를 수행한다. |

| | |
|---|---|
| **Prototype** | void list_insert_ordered (struct list *list, struct list_elem *elem, list_less_func *less, void *aux); |
| **Parameter** | insert할 list와 insert할 elem, 대소를 비교할 함수 less와 less에 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받는 원소를 list 내에 insert하는데 이때 insert되는 위치는 작은 순서대로 정렬되어 insert되도록, elem보다 큰 |

| | |
|---|---|
| | 리스트 내 다른 원소들의 앞에 오도록 한다. |

| | |
|---|---|
| **Prototype** | void list_unique (struct list *list, struct list *duplicates, list_less_func *less, void *aux); |
| **Parameter** | 중복된 원소를 제거하고자 하는 list와 제거된 원소를 저장할 duplicate list, 그리고 대소를 비교할 함수 less와 less에 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 list의 중복된 원소들을 제거한다. 또한 duplicate에 NULL을 전달받는다면 제거만 하지만, 다른 list를 전달받는다면 제거된 element들을 순서대로 저장한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_max (struct list *list, list_less_func *less, void *aux); |
| **Parameter** | max를 찾고자 하는 list와 원소들의 대소를 비교할 함수 less, less 함수에 사용될 aux를 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list 내의 가장 큰 data를 갖는 원소이다. |
| **Function** | list 내의 가장 큰 data를 가진 원소를 less 함수를 통해 찾아 return한다. |

| | |
|---|---|
| **Prototype** | struct list_elem *list_min (struct list *list, list_less_func *less, void *aux); |
| **Parameter** | min를 찾고자 하는 list와 원소들의 대소를 비교할 함수 less, less 함수에 사용될 aux를 파라미터로 전달받는다. |
| **Return** | list_elem 타입을 return하는데, 이 때 return하는 값은 list 내의 가장 작은 data를 갖는 원소이다. |
| **Function** | list 내의 가장 작은 data를 가진 원소를 less 함수를 통해 찾아 return한다. |

## III.Hash Table

| | |
|---|---|
| **Prototype** | bool hash_init (struct hash *h, hash_hash_func *hash, hash_less_func *less, void *aux); |
| **Parameter** | 초기화하고자 하는 hash(h)와 hash의 원소의 값을 구할 때 사용된 hash_hash_func(hash)과 hash_less_func, 이들의 파라미터로 사용되는 aux를 파라미터로 전달받는다. |
| **Return** | bool type을 return하는데, 이때 return하는 값은 초기화가 성공 성공적 이루어졌을 때 true, 그렇지 않을 때 false를 return한다. |
| **Function** | 파라미터로 받은 hash 내의 member 변수들을 모두 초기화한다. |

| | |
|---|---|
| **Prototype** | void hash_clear (struct hash *h, hash_action_func *destructor); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 destructor 함수를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 destructor 함수를 이용해 전달받은 hash(h)의 hash_elem들을 지우고 bucket을 초기화한다. |

| | |
|---|---|
| **Prototype** | void hash_destroy (struct hash *h, hash_action_func *destructor); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 destructor 함수를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 hash를 hash_clear함수를 통해 지우고 free하여 제거한다. |

| | |
|---|---|
| **Prototype** | struct hash_elem * hash_insert (struct hash *h, struct hash_elem *new); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 insert할 hash_elem 구조체를 파라미터로 전달받는다. |
| **Return** | hash_elem 구조체를 return하는데, hash 내에 이미 new와 같은 |

| | |
|---|---|
| | hash_elem(old)이 있다면 old를 return하고 없다면 null을 return 한다. |
| **Function** | 파라미터로 전달받은 hash 내부의 적절한 bucket에 hash_elem을 insert한다. 이미 동일한 hash_elem이 존재한다면 insert하지 않는다. |

| | |
|---|---|
| **Prototype** | struct hash_elem *hash_replace (struct hash *h, struct hash_elem *new); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 replace할 hash_elem 구조체를 파라미터로 전달받는다. |
| **Return** | hash_elem 구조체를 return하는데, 파라미터로 전달받은 new가 insert되면서 remove된 old hash_elem을 return한다. |
| **Function** | 파라미터로 전달받은 hash 내부에 new를 insert하고 이미 존재하는 hash_elem은 제거한다. |

| | |
|---|---|
| **Prototype** | struct hash_elem *hash_find (struct hash *h, struct hash_elem *e); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 찾고자 하는 hash_elem 구조체를 파라미터로 전달받는다. |
| **Return** | hash_elem 구조체를 return하는데, 파라미터로 전달받은 e를 return한다. 만약 hash에 존재하지 않는다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 hash 내부에 hash_elem e가 있는지 찾는다. |

| | |
|---|---|
| **Prototype** | struct hash_elem *hash_delete (struct hash *h, struct hash_elem *e); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 제거하고자 하는 hash_elem 구조체를 파라미터로 전달받는다. |
| **Return** | hash_elem 구조체를 return하는데, 파라미터로 전달받은 e가 hash 내부에 있다면 e를 return 없다면 NULL을 return한다. |

| | |
|---|---|
| **Function** | 파라미터로 전달받은 hash 내부에 hash_elem e를 제거한다. |

| | |
|---|---|
| **Prototype** | void hash_apply (struct hash *h, hash_action_func *action); |
| **Parameter** | hash table의 정보를 지닌 struct hash와 수행하고자 하는 함수의 포인터인 hash_action_func을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 hash 내부의 모든 hash_elem에 대하여 action 함수를 수행한다. |

| | |
|---|---|
| **Prototype** | void hash_first (struct hash_iterator *i, struct hash *h); |
| **Parameter** | hash table과 bucket, hash_elem을 지닌 구조체 hash_iterator와 hash table의 정보를 지닌 struct hash를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | hash_iterator를 i가 파라미터로 전달받은 hash h의 정보를 지니도록 초기화한다. |

| | |
|---|---|
| **Prototype** | struct hash_elem *hash_next (struct hash_iterator *i); |
| **Parameter** | 함수를 수행할 iterator의 값을 지닌 hash_iterator를 파라미터로 전달받는다. |
| **Return** | hash_elem 타입을 return하는데, i가 지닌 현재 hash_elem이 아닌 다음 elem을 return한다. 만약 다음 hash_elem이 존재하지 않는다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 iterator가 지닌 hash_elem의 다음 hash_elem을 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | struct hash_elem *hash_cur (struct hash_iterator *i); |
| **Parameter** | 함수를 수행할 iterator의 값을 지닌 hash_iterator를 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | hash_elem 타입을 return하는데, 현재 iterator가 지닌 hash_elem을 return한다. 만약 현재 hash_elem이 hash table의 마지막이라면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 iterator의 현재 hash_elem을 반환한다. |

| | |
|---|---|
| **Prototype** | size_t hash_size (struct hash *h); |
| **Parameter** | size를 계산하고자 하는 hash 구조체를 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, hash의 size(hash_elem의 수)를 return한다. |
| **Function** | 파라미터로 전달받은 hash의 hash_elem의 개수를 구하여 반환한다. |

| | |
|---|---|
| **Prototype** | bool hash_empty (struct hash *h); |
| **Parameter** | 확인하고자 하는 hash 구조체를 파라미터로 전달받는다. |
| **Return** | bool type을 return하는데, 만약 hash가 비어있다면(hash_elem이 없다면) true, 비어있지 않다면 false를 return한다. |
| **Function** | 파라미터로 전달받은 hash가 empty hash인지 확인하여 결과를 반환한다. |

| | |
|---|---|
| **Prototype** | unsigned hash_bytes (const void *buf_, size_t size); |
| **Parameter** | 함수를 수행할 buffer와 buffer의 size를 파라미터로 전달받는다. |
| **Return** | unsigned 타입을 return하는데, 이때 return하는 값은 hash value이다. |
| **Function** | 파라미터로 전달받은 buf의 (size) bytes의 hash value를 구하여 반환한다. |

| | |
|---|---|
| **Prototype** | unsigned hash_string (const char *s_); |
| **Parameter** | 함수를 수행할 문자열을 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | unsigned 타입을 return하는데, 이때 return하는 값은 hash value이다. |
| **Function** | 파라미터로 전달받은 s의 hash value를 구한다. |

| | |
|---|---|
| **Prototype** | unsigned hash_int (int i); |
| **Parameter** | 함수를 수행할 integer를 파라미터로 전달받는다. |
| **Return** | unsigned 타입을 return하는데, 이때 return하는 값은 hash_bytes 함수를 통해 변환된 hash value이다. |
| **Function** | 파라미터로 전달받은 i의 hash value를 구한다. |

| | |
|---|---|
| **Prototype** | static struct list *find_bucket (struct hash *h, struct hash_elem *e); |
| **Parameter** | 찾고자하는 bucket을 지닌 hash와 찾고자하는 bucket이 보유한 hash_elem을 파라미터로 전달받는다. |
| **Return** | static struct list 타입을 return하는데, 이때 return되는 값은 e를 지닌 bucket이다. |
| **Function** | 파라미터로 전달받은 hash 내부에 hash_elem을 보유한 bucket을 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | static struct hash_elem *find_elem (struct hash *h, struct list *bucket, struct hash_elem *e); |
| **Parameter** | 찾고자하는 hash_elem을 지닌 hash와 bucket, 찾고자하는 hash_elem을 파라미터로 전달받는다. |
| **Return** | static struct hash_elem 타입을 return하는데, 파라미터로 전달받은 e와 같은 hash_elem이 존재하면 그것을 return하고 있지 않다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 hash의 bucket 내부에 e와 같은 hash_elem이 있는지 찾는다. |

| **Prototype** | static inline size_t turn_off_least_1bit (size_t x); |
|---|---|
| **Parameter** | size_t 타입의 변수 x를 파라미터로 전달받는다. |
| **Return** | bit operation을 이용해 x와 (x-1)의 & 연산 결과를 return한다. |
| **Function** | x의 1로 설정된 비트 중 가장 아래 비트를 0으로 바꾼다. |

| **Prototype** | static inline size_t is_power_of_2 (size_t x); |
|---|---|
| **Parameter** | size_t 타입의 변수 x를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, 이때 return하는 값은 x가 2의 거듭제곱이라면 true, 아니라면 false를 return한다. |
| **Function** | 파라미터로 전달받은 값이 2의 거듭제곱인지 확인한다. |

| **Prototype** | static void rehash (struct hash *h); |
|---|---|
| **Parameter** | 함수를 수행할 hash를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 받은 hash의 bucket의 수를 이상적으로 바꾼다. |

| **Prototype** | static void insert_elem (struct hash *h, struct list *bucket, struct hash_elem *e); |
|---|---|
| **Parameter** | insert하고자 하는 hash_elem을 지닌 hash와 list타입의 bucket 그리고, insert할 hash_elem을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 hash의 bucket의 가장 앞에 hash_elem을 insert한다. |

| **Prototype** | static void remove_elem (struct hash *h, struct hash_elem *e); |
|---|---|
| **Parameter** | 제거하고자 하는 hash_elem을 지닌 hash와 hash_elem을 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 hash 내의 hash_elem e를 제거한다. |

## IV. Bitmap

| | |
|---|---|
| **Prototype** | static inline size_t elem_idx (size_t bit_idx); |
| **Parameter** | 찾고자 하는 bit의 index를 파라미터로 전달받는다. |
| **Return** | static inline size_t 타입을 return하는데, 이때 return하는 값은 element의 index이다. |
| **Function** | 파라미터로 전달받은 bit의 index를 지닌 element의 index를 찾는다. |

| | |
|---|---|
| **Prototype** | static inline elem_type bit_mask (size_t bit_idx); |
| **Parameter** | 함수를 수행할 bit의 index를 파라미터로 전달받는다. |
| **Return** | static inline elem_type을 타입을 return하는데, 이때 elem type은 bit로 구성된 bitmap의 원소이다. 또한 return하는 값은 파라미터로 받은 index의 비트가 true인 elem type을 return한다. |
| **Function** | 파라미터로 전달받은 index에 해당하는 bit만 1(true)인 elem_type을 찾아 반환한다. |

| | |
|---|---|
| **Prototype** | static inline size_t elem_cnt (size_t bit_cnt); |
| **Parameter** | 함수를 수행할 bit의 개수를 파라미터로 전달받는다. |
| **Return** | static inline size_t 타입을 return하는데, 이때 return하는 값은 파라미터로 전달받은 bit 개수에 필요한 element으 ㅣ수이다. |
| **Function** | 파라미터로 전달받은 bit 개수에 필요한 element의 수를 반환한다. |

| | |
|---|---|
| **Prototype** | static inline size_t byte_cnt (size_t bit_cnt); |
| **Parameter** | 함수를 수행할 bit의 개수를 파라미터로 전달받는다. |

| | |
|---|---|
| **Return** | static inline size_t 타입을 return하는데, 이때 return하는 값은 파라미터로 전달받은 bit 개수에 필요한 byte의 수이다. |
| **Function** | 파라미터로 전달받은 bit 개수에 필요한 byte의 개수를 반환한다. |

| | |
|---|---|
| **Prototype** | static inline elem_type last_mask (const struct bitmap *b); |
| **Parameter** | bit의 개수와 bitmap의 값을 나타내는 elem type의 변수를 지닌 struct bitmap을 파라미터로 전달받는다. |
| **Return** | static inline elem_type을 return하는데 이때 return하는 값은 마지막 비트만 1(ture)인 elem type의 변수이다. |
| **Function** | 파라미터로 전달받은 비트맵의 마지막 element의 bit만 1이고 나머지는 0으로 설정한다. |

| | |
|---|---|
| **Prototype** | struct bitmap *bitmap_create (size_t bit_cnt); |
| **Parameter** | 만들고자 하는 bit의 개수를 파라미터로 전달받는다. |
| **Return** | struct bitmap 타입의 bitmap을 return하는데, 만약 bitmap을 만드는데 실패한다면 NULL을 return한다. |
| **Function** | 파라미터로 전달받은 bit 개수(bit_cnt)만큼의 bit를 지닌 bitmap을 만들고 초기화한다. 이때 bit의 초기값은 모두 false(0)로 설정한다. |

| | |
|---|---|
| **Prototype** | struct bitmap *bitmap_create_in_buf (size_t bit_cnt, void *block, size_t block_size ); |
| **Parameter** | 만들고자 하는 bitmap의 크기(bit_cnt)와 buffer block(block), buffer block의 크기(block_size)를 파라미터로 전달받는다. |
| **Return** | struct bitmap 타입의 bitmap을 return한다. |
| **Function** | 파라미터로 전달받은 bit의 크기(bit_cnt)만큼의 bit를 지닌 bitmap을 만들고 초기화한다. 이때 block에 미리 할당된 메모리를 이용해 만든다. |

| **Prototype** | size_t bitmap_buf_size (size_t bit_cnt); |
|---|---|
| **Parameter** | buffer의 size를 구할 bit_cnt를 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 파라미터로 전달받은 bit의 개수(bit_cnt)만큼의 bit를 지닌 bitmap을 만들기 위한 buffer의 size이다. |
| **Function** | 파라미터로 전달받은 bit의 개수(bit_cnt)만큼 bit를 지닌 bitmap을 만들기 위한 buffer의 size를 구한다. |

| **Prototype** | void bitmap_destroy (struct bitmap *b); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap을 free하여 제거한다. |

| **Prototype** | size_t bitmap_size (const struct bitmap *b); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap을 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 bitmap(bit array)의 크기, 즉 bit의 개수이다. |
| **Function** | 파라미터로 전달받은 bitmap을 구성하는 bit의 개수를 구해 반환한다. |

| **Prototype** | void bitmap_set (struct bitmap *b, size_t idx, bool value); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 bit의 index와 설정하고자 하는 값(value)을 파라미터로 전달받는다 |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 index번째 bit를 value 값으로 설정한다. |

| **Prototype** | void bitmap_mark (struct bitmap *b, size_t bit_idx); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 bit의 index를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 index번째 bit를 true(1)로 설정한다. |

| **Prototype** | void bitmap_reset (struct bitmap *b, size_t bit_idx); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 bit의 index를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 index번째 bit를 false(0)으로 설정한다. |

| **Prototype** | bool bitmap_test (const struct bitmap *b, size_t idx); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 bit의 index를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, 이때 return하는 값은 index 위치의 bit의 값이다. |
| **Function** | 파라미터로 전달받은 bitmap의 idx번째 bit의 값을 return한다. |

| **Prototype** | void bitmap_flip (struct bitmap *b, size_t bit_idx); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 bit의 index를 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 idx번째 bit의 값을 flip한다. |

| **Prototype** | void bitmap_set_all (struct bitmap *b, bool value); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 설정하고자 하는 값(value)를 파라미터 |

| | |
|---|---|
| | 로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 전체 bit를 value 값으로 설정한다. |

| | |
|---|---|
| **Prototype** | void bitmap_set_multiple (struct bitmap *b, size_t start, size_t cnt, bool value); |
| **Parameter** | 함수를 수행할 bitmap과 index(start), 개수(cnt)와 값(value)을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index에서부터 cnt 개수의 비트의 값을 value로 설정한다. |

| | |
|---|---|
| **Prototype** | size_t bitmap_count (const struct bitmap *b, size_t start, size_t cnt, bool value); |
| **Parameter** | 함수를 수행할 bitmap과 index(start), 개수(cnt)와 값(value)을 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 start에서부터 cnt 개수의 비트 사이에 value의 값을 가진 bit의 개수를 구해 return한다. |
| **Function** | 파라미터로 주어진 bitmap의 범위에 value값을 가진 비트의 개수를 세 반환한다. |

| | |
|---|---|
| **Prototype** | bool bitmap_contains (const struct bitmap *b, size_t start, size_t cnt, bool value); |
| **Parameter** | 함수를 수행할 bitmap과 index(start), 개수(cnt)와 값(value)을 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, start에서부터 cnt 개수의 비트 사이 |

| | |
|---|---|
| | 에 value가 있다면 true, 없다면 false를 return한다. |
| **Function** | 파라미터로 전달받은 비트맵의 범위에 value가 있는지 확인하고 결과를 반환한다. |

| | |
|---|---|
| **Prototype** | bool bitmap_any (const struct bitmap *b, size_t start, size_t cnt); |
| **Parameter** | 함수를 수행할 bitmap과 index (start), 그리고 개수(cnt)를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, start에서부터 cnt 개수의 비트 사이에 1이 있다면 true, 없다면 false를 return한다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index에서부터 cnt 개수의 비트에 1(true)이 있는지 확인한다. |

| | |
|---|---|
| **Prototype** | bool bitmap_none (const struct bitmap *b, size_t start, size_t cnt); |
| **Parameter** | 함수를 수행할 bitmap과 index (start), 그리고 개수(cnt)를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, start에서부터 cnt 개수의 비트 사이에 0이 있다면 true, 없다면 false를 return한다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index에서부터 cnt 개수의 비트에 0(false)이 있는지 확인한다. |

| | |
|---|---|
| **Prototype** | bool bitmap_all (const struct bitmap *b, size_t start, size_t cnt); |
| **Parameter** | 함수를 수행할 bitmap과 index (start), 그리고 개수(cnt)를 파라미터로 전달받는다. |
| **Return** | bool 타입을 return하는데, start에서부터 cnt 개수의 모든 값이 1이라면 true, 아니라면 false를 return한다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index에서부터 cnt개의 값이 모두 true(1)인지 확인하여 결과를 반환한다. |

| **Prototype** | size_t bitmap_scan (const struct bitmap *b, size_t start, size_t cnt, bool value); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 index(start), 개수(cnt)와 값(value)을 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 start index부터 cnt개수의 모든 비트가 파라미터로 전달받은 value라면 start index를 return하고 없다면 BITMAP_ERROR를 return한다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index부터 cnt개의 비트의 값이 value로 되어있는 group이 있는지 확인한다. |

| **Prototype** | size_t bitmap_scan_and_flip (struct bitmap *b, size_t start, size_t cnt, bool value); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap과 index(start), 개수(cnt)와 값(value)을 파라미터로 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 start index부터 cnt개수의 모든 비트가 파라미터로 전달받은 value라면 start index를 return하고 없다면 BITMAP_ERROR를 return, cnt가 0이라면 0을 return한다. |
| **Function** | 파라미터로 전달받은 bitmap의 start번째 index부터 cnt개의 비트의 값이 value로 되어있는 group이 있는지 확인한다. 만약 있다면 group의 bit를 모두 flip한다. |

| **Prototype** | size_t bitmap_file_size (const struct bitmap *b); |
|---|---|
| **Parameter** | 함수를 수행할 bitmap을 전달받는다. |
| **Return** | size_t 타입을 return하는데, 이때 return하는 값은 byte_cnt 함수에 파라미터로 전달받은 bitmap의 bit의 수를 인자로 줘서 계산한 결과이다. |
| **Function** | 파라미터로 전달받은 bitmap의 bit의 수를 바이트로 표현할 때 필요한 byte의 수를 구해 반환한다. |

| | |
|---|---|
| **Prototype** | void bitmap_dump (const struct bitmap *b); |
| **Parameter** | dump할 bitmap을 파라미터로 전달받는다. |
| **Return** | void 타입이므로, 따로 값을 return하지 않는다. |
| **Function** | 파라미터로 받은 bitmap을 16진수로 변환하여 dump한다. |