Project #1 : MyLib

담당 교수 : 김영재 교수님

학번 : 20191243

이름 : 김태규

반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.

I. Additional Implementation

I-1. Additional Data Structure Implementation

1) `list, hashtable, bitmap` 각각의 포인터 주소를 저장하는 `array`를 전역변수로 구현 (`#define MAX_ooo_NUM 10`)

```
struct list *list_arr[MAX_LIST_NUM] = { NULL };
struct hash *hash_arr[MAX_HASH_NUM] = { NULL };
struct bitmap *bitmap_arr[MAX_BITMAP_NUM] = { NULL };
```

2) `struct list_item` 구현

```
/* List item. */
struct list_item {
  struct list_elem elem;
  int data;
};
```

3) `struct list`에 < `int item_cnt` > 필드 추가

```
/* List. */
struct list {
  struct list_elem head;      /* List head. */
  struct list_elem tail;      /* List tail. */
  int item_cnt;  // # of items
};
```

I-2. Function Implementation

1) List

Prototype	struct list *create_list(char *tok)
Parameter	char *tok
Return	list_arr[idx]가 이미 차 있다면 NULL 반환 list_arr[idx]가 비어 있다면 새로 만든 list의 포인터 주소 반환
Function	tok라는 문자열을 받아 list를 하나 생성 - list_init(struct *list) list의 head와 tail을 initialize 해주는 함수

```
struct list *create_list(char *tok){
    int idx = atoi(&tok[4]);
    if (list_arr[idx] != NULL){
        printf("CAN'T CREATE: %s ALREADY CREATED\n", tok);
        return NULL;
    }
    else{
        struct list *new_list = (struct list *)malloc(sizeof(struct list));
        list_init(new_list);
        list_arr[idx] = new_list;
        return new_list;
    }
}
```

Prototype	bool delete_list(char *tok)
Parameter	char *tok
Return	list_arr[idx]가 차 있다면 true (삭제 o) 반환 list_arr[idx]가 비어 있다면 false (삭제 x) 반환
Function	tok라는 문자열을 받아 해당 list의 메모리 해제 후 list_arr에 채워져 있던 해당 list의 포인터 주소를 NULL로 초기화

```
bool delete_list(char *tok){
    int idx = atoi(&tok[4]);
    if (list_arr[idx] == NULL){ // create 된 적이 없는 list 라면
```

```
        return false;
    }
    else{
        free(list_arr[idx]);
        list_arr[idx] = NULL;
        return true;
    }
}
```

Prototype	void dumpdata_list(char *tok)
Parameter	char *tok
Return	X
Function	tok라는 문자열을 받아 해당 list의 모든 data를 출력

```
void dumpdata_list(char *tok){
    int idx = atoi(&tok[4]);
    if (list_arr[idx] == NULL){
        printf("CAN'T DUMPDATA: %s NOT CREATED\n", tok);
        return;
    }
    else{
        int flag = 0;
        struct list_elem *temp = list_begin(list_arr[idx]);
        while (temp->next != NULL){ // temp가 tail이 아니면 반복문 실행
            int dump = list_entry(temp, struct list_item, elem)->data;
            printf("%d ", dump);
            temp = list_next(temp);
            flag = 1;
        }
        if (flag){
            printf("\n");
        }
        return;
    }
}
```

Prototype	void list_swap(struct list_elem *elem1, struct list_elem *elem2)
Parameter	struct list_elem *elem1, struct list_elem *elem2

Return	X
Function	elem1과 elem2이 떨어져 있는 경우와 인접해 있는 경우를 나눠서 elem swap 구현 main.c에서 elem1이 elem2보다 앞서는 elem가 되도록 처리하여 list_swap 함수에 변수로 넣음

```c
void list_swap(struct list_elem *elem1, struct list_elem *elem2){
  if (elem1 == elem2 || elem1 == NULL || elem2 == NULL){
    printf("CAN'T SWAP\n");
    return;
  }

  if (elem1->next != elem2){  // elem1 과 elem2 이 떨어져 있는 경우
    elem1->prev->next = elem2;
    elem1->next->prev = elem2;

    elem2->prev->next = elem1;
    elem2->next->prev = elem1;

    struct list_elem *temp = elem1->next;
    elem1->next = elem2->next;
    elem2->next = temp;

    temp = elem1->prev;
    elem1->prev = elem2->prev;
    elem2->prev = temp;
  }
  else{                       // elem1 과 elem2 가 인접해 있는 경우
    elem1->prev->next = elem2;
    elem2->next->prev = elem1;

    elem1->next = elem2->next;
    elem2->prev = elem1->prev;
    elem1->prev = elem2;
    elem2->next = elem1;
  }
  return;
}

// main.c 에서 list_swap 을 처리하는 코드
        else if (strcmp(tok[0], "list_swap") == 0){
            int idx = atoi(&tok[1][4]);
            int iter1 = atoi(tok[2]);
            int iter2 = atoi(tok[3]);
```

```
            if (iter1 > iter2){
                int n = iter1;
                iter1 = iter2;
                iter2 = n;
            }

            struct list_elem *temp = list_begin(list_arr[idx]);
            for (int i = 0; i < iter1; i++){
                temp = list_next(temp);
            }
            struct list_elem *target_elem1 = temp;

            temp = list_begin(list_arr[idx]);
            for (int i = 0; i < iter2; i++){
                temp = list_next(temp);
            }
            struct list_elem *target_elem2 = temp;

            list_swap(target_elem1, target_elem2);
        }
```

Prototype	void list_shuffle(struct list *list)
Parameter	struct list *list
Return	X
Function	list의 elem들의 순서를 무작위로 위치를 바꿈 (1) int iter: 난수 생성을 통해 list_swap 횟수 정하기 (2) int rand_num1, rand_num2: 난수 생성을 통해 list_swap의 대상이 될 elem 2개 정하기

```
void list_shuffle(struct list *list){
  if (&(list->head) == NULL || &(list->tail) == NULL){
    printf("EMPTY LIST\n");
    return;
  }
  struct list_elem *target1 = NULL, *target2 = NULL;
  int rand_num1, rand_num2;
  int num = list->item_cnt;

  // 현재 시간을 마이크로초 단위로 표현하여 시드 생성
  unsigned long long seed = get_current_microseconds();
  srand(seed);
```

```
  int iter = rand() % 9 + 1;
  for (int i = 0; i < iter; i++){
    rand_num1 = rand() % num;
    do{
      rand_num2 = rand() % num;
    } while(rand_num1 == rand_num2);

    if (rand_num1 > rand_num2){
      int n = rand_num1;
      rand_num1 = rand_num2;
      rand_num2 = n;
    }

    target1 = list_begin(list);
    for (int j = 0; j < rand_num1; j++){
      target1 = list_next(target1);
    }
    target2 = list_begin(list);
    for (int j = 0; j < rand_num2; j++){
      target2 = list_next(target2);
    }

    list_swap(target1, target2);
  }
  return;
}
```

Prototype	bool list_less(const struct list_elem *a, const struct list_elem *b, void *aux)
Parameter	const struct list_elem *a, const struct list_elem *b, void *aux
Return	a의 data가 b의 data보다 작으면 true, 크면 false 반환
Function	elem 2개의 data를 대소 비교하는 함수

```
bool list_less(const struct list_elem *a, const struct list_elem *b, void
*aux){
  return (list_entry(a, struct list_item, elem)->data <
                                list_entry(b, struct list_item, elem)->data)
                                                        ? true : false;
}
```

2) Hash

Prototype	struct hash *create_hash(char *tok)
Parameter	char *tok
Return	hash_arr[idx]가 비어 있다면 new_hash 반환 hash_arr[idx]가 차 있다면 NULL 반환
Function	tok라는 문자열을 받아 hashtable 생성

```
struct hash *create_hash(char *tok){
    int idx = atoi(&tok[4]);
    if (hash_arr[idx] != NULL){
        printf("CAN'T CREATE: %s ALREADY CREATED\n", tok);
        return NULL;
    }
    else{
        void *aux = NULL;
        struct hash *new_hash = (struct hash *)malloc(sizeof(struct hash));
        hash_init(new_hash, hash_func, hash_less, aux);
        hash_arr[idx] = new_hash;
        return new_hash;
    }
}
```

Prototype	bool delete_hash(char *tok)
Parameter	char *tok
Return	hash_arr[idx]가 비어 있다면 false 반환 hash_arr[idx]가 차 있다면 true 반환
Function	tok라는 문자열을 받아 hashtable 메모리 해제 후 hash_arr에 채워져 있던 해당 hashtable의 주소를 NULL로 초기화

```
bool delete_hash(char *tok){
    int idx = atoi(&tok[4]);
    if (hash_arr[idx] == NULL){ // create 된 적이 없는 hashtable 이라면
        return false;
    }
    else{
        hash_destroy(hash_arr[idx], destructor);
        hash_arr[idx] = NULL;
        return true;
```

```
    }
}
```

Prototype	void dumpdata_hash(char *tok)
Parameter	char *tok
Return	X
Function	tok라는 문자열을 받아 해당 hashtable에 있는 모든 hash_elem의 value 출력

```
void dumpdata_hash(char *tok){
    int idx = atoi(&tok[4]);
    if (hash_arr[idx] == NULL){
        printf("CAN'T DUMPDATA: %s NOT CREATED\n", tok);
        return;
    }
    else{
        int flag = 0;
        struct hash_iterator *iter = (struct hash_iterator *)
                                     malloc(sizeof(struct hash_iterator));
        hash_first(iter, hash_arr[idx]);
        hash_next(iter);
        struct list_elem *temp = &hash_cur(iter)->list_elem;
        while (temp != NULL){
            int dump = list_elem_to_hash_elem(temp)->value;
            printf("%d ", dump);
            temp = &hash_next(iter)->list_elem;
            flag = 1;
        }
        if (flag){
            printf("\n");
        }
        return;
    }
}
```

Prototype	unsigned hash_int_2(int i)
Parameter	int i
Return	hash

Function	정수 i를 받아 아래와 같은 연산 과정 거친 후 해시값 반환

```
unsigned hash_int_2(int i){
  unsigned hash = i;
  hash ^= (hash >> 20) ^ (hash >> 12);
  hash += ~(hash << 15);
  hash ^= (hash >> 10);
  hash += (hash << 3);
  hash ^= (hash >> 6);
  hash ^= (hash >> 7) ^ (hash >> 4);
  hash = hash % 16;
  return hash;
}
```

Prototype	unsigned hash_func(const struct hash_elem *elem, void *aux)
Parameter	const struct hash_elem *elem, void *aux
Return	hash_int 함수에 해당 elem의 value을 넣었을 때의 결과값
Function	literally hash function value를 해시값으로 바꿔줌

```
unsigned hash_func(const struct hash_elem *elem, void *aux){
  return hash_int(elem->value);
}
```

Prototype	bool hash_less(const struct hash_elem *a, const struct hash_elem *b, void *aux)
Parameter	const struct hash_elem *a, const struct hash_elem *b, void *aux
Return	a의 value가 b의 value 보다 작으면 true 반환, 크면 false 반환
Function	elem 2개의 value 대소 비교하는 함수

```
bool hash_less(const struct hash_elem *a, const struct hash_elem *b, void
*aux){
  return (a->value < b->value) ? true : false;
}
```

Prototype	void destructor(struct hash_elem *hash_elem, void *aux)
Parameter	struct hash_elem *hash_elem, void *aux
Return	X
Function	hash_elem의 메모리 해제

```c
void destructor(struct hash_elem *hash_elem, void *aux){
  free(hash_elem);
}
```

Prototype	void square(struct hash_elem *elem, void *aux)
Parameter	struct hash_elem *elem, void *aux
Return	X
Function	hash_elem의 value를 제곱해준다.

```c
void square(struct hash_elem *elem, void *aux){
  int n = elem->value;
  elem->value = n * n;
}
```

Prototype	void triple(struct hash_elem *elem, void *aux)
Parameter	struct hash_elem *elem, void *aux
Return	X
Function	hash_elem의 value를 세제곱해준다.

```c
void triple(struct hash_elem *elem, void *aux){
  int n = elem->value;
  elem->value = n * n * n;
}
```

3) Bitmap

Prototype	struct bitmap *create_bitmap(char *tok1, char *tok2)
Parameter	char *tok1, char *tok2
Return	bitmap_arr[idx]가 차 있다면 NULL 반환 bitmap_arr[idx]가 비어 있다면 new_bitmap의 포인터 주소 반환
Function	tok1과 tok2를 받아 tok2만큼의 bit 수를 가진 bitmap 생성

```
struct bitmap *create_bitmap(char *tok1, char *tok2){
    int idx = atoi(&tok1[2]);
    int bit_cnt = atoi(tok2);
    if (bitmap_arr[idx] != NULL){
        printf("CAN'T CREATE: %s ALREADY CREATED\n", tok1);
        return NULL;
    }
    else{
        struct bitmap *new_bitmap = bitmap_create(bit_cnt);
        bitmap_arr[idx] = new_bitmap;
        return new_bitmap;
    }
}
```

Prototype	bool delete_bitmap(char *tok)
Parameter	char *tok
Return	bitmap_arr[idx]가 비어 있다면 false (삭제 x) 반환 bitmap_arr[idx]가 차 있다면 true (삭제 o) 반환
Function	tok라는 문자열을 받아 해당 bitmap 메모리 해제 후 bitmap_arr에 채워져 있던 해당 bitmap 포인터 주소를 NULL로 초기화

```
bool delete_bitmap(char *tok){
    int idx = atoi(&tok[2]);
    if (bitmap_arr[idx] == NULL){ // create 된 적이 없는 bitmap 이라면
        return false;
    }
    else{
        bitmap_destroy(bitmap_arr[idx]);
```

```
        bitmap_arr[idx] = NULL;
        return true;
    }
}
```

Prototype	void dumpdata_bitmap(char *tok)
Parameter	char *tok
Return	X
Function	tok라는 문자열을 받아 해당 bitmap의 bit 정보 출력 < bit 출력 방법 > ① bitmap->bits[i / (sizeof(elem_type) * 8)] 비트맵 배열에서 i번째 비트가 포함된 워드를 선택. sizeof(elem_type)은 워드의 크기를 byte 단위로 나타내고, 8을 곱해줌으로써 bit로 변환 ② >> (i % (sizeof(elem_type) * 8)) 선택된 워드에서 i번째 비트의 위치를 계산. i % (sizeof(elem_type)*8)은 i를 워드 내에서의 오프셋으로 변환 “>>” bit shift 연산자를 사용하여 해당 bit가 최하위 bit가 되도록 이동 ③ & 1 & 1을 통해 최하위 비트만 유지하여 해당 위치의 bit 값이 1인지 0인지 확인 가능.

```
void dumpdata_bitmap(char *tok){
    int idx = atoi(&tok[2]);
    struct bitmap *bitmap = bitmap_arr[idx];
    if (bitmap == NULL){
        printf("CAN'T DUMPDATA: %s NOT CREATED\n", tok);
        return;
    }
    else{
```

```
        for (int i = 0; i < bitmap->bit_cnt; i++){
             printf("%ld", (bitmap->bits[i / (sizeof(elem_type) * 8)]
                                        >> (i % (sizeof(elem_type) * 8))) & 1);
        }
        printf("\n");
        return;
    }
}
```

Prototype	struct bitmap *bitmap_expand(struct bitmap *bitmap, int size)
Parameter	struct bitmap *bitmap, int size
Return	bitmap의 포인터 주소
Function	원래 있던 bitmap의 bit 수를 size만큼 늘리고 추가된 비트를 0으로 설정함

```
struct bitmap *bitmap_expand(struct bitmap *bitmap, int size){
  bitmap->bit_cnt += size;
  bitmap_set_multiple (bitmap, bitmap->bit_cnt-size, size, false);
  return bitmap;
}
```

II. List

Prototype	`void list_init (struct list *list)`
Parameter	struct list *
Return	X
Function	list의 head와 tail의 initialization 작업

Prototype	`struct list_elem *list_begin (struct list *list)`
Parameter	struct list *
Return	`return list->head.next;`
Function	list의 첫 번째 elem의 주소 반환 (head 바로 다음 elem)

Prototype	`struct list_elem *list_next (struct list_elem *elem)`
Parameter	struct list_elem *
Return	`return elem->next;`
Function	다음 elem의 주소 반환

Prototype	`struct list_elem *list_end (struct list *list)`
Parameter	struct list *
Return	`return &list->tail;`
Function	list의 tail 주소 반환

Prototype	`struct list_elem *list_prev (struct list_elem *elem)`
Parameter	struct list_elem *
Return	`return elem->prev;`
Function	이전 elem의 주소 반환

Prototype	`void list_insert (struct list_elem *before, struct list_elem *elem)`
Parameter	struct list_elem *before, struct list_elem *elem
Return	X
Function	BEFORE 이전 위치에 elem 노드를 삽입

Prototype	`void list_splice (struct list_elem *before, struct list_elem *first, struct list_elem *last)`
Parameter	struct list_elem *before, struct list_elem *first, struct list_elem *last
Return	X
Function	first부터 last까지의 elem들을 BEFORE 이전 위치에 삽입

Prototype	`void list_push_back (struct list *list, struct list_elem *elem)`
Parameter	
Return	X
Function	list의 제일 마지막 순서로 elem 삽입 (tail 바로 앞)

Prototype	`struct list_elem * list_pop_front (struct list *list)`
Parameter	
Return	`return front;`
Function	list의 제일 앞에 있는 elem를 list에서 pop 시킨 후 pop 시킨 elem를 반환

Prototype	`void list_unique (struct list *list, struct list *duplicates, list_less_func *less, void *aux)`
Parameter	
Return	X
Function	list에서 중복되는 값을 가진 elem들을 duplicates에 옮김.

Prototype	
Parameter	
Return	
Function	

Prototype	
Parameter	
Return	
Function	

Prototype	
Parameter	
Return	
Function	

III.Hash Table

Prototype	`bool hash_init (struct hash *h, hash_hash_func *hash, hash_less_func *less, void *aux)`
Parameter	
Return	hashtable initialization 성공하면 true, 실패하면 false
Function	새 hashtable initialization 작업

Prototype	`void hash_destroy (struct hash *h, hash_action_func *destructor)`
Parameter	
Return	X
Function	hashtable 메모리 해제 작업

Prototype	`struct hash_elem * hash_insert (struct hash *h, struct hash_elem *new)`
Parameter	
Return	삽입하고자 하는 elem가 이미 존재하는 elem라면 해당 elem를, 새로운 elem라면 NULL 반환
Function	새 elem를 hashtable에 삽입. 이미 같은 value를 갖고 있는 elem가 존재한다면 아무 것도 하지 않음.

Prototype	`struct hash_elem * hash_replace (struct hash *h, struct hash_elem *new)`
Parameter	
Return	삽입하고자 하는 elem가 이미 존재하는 elem라면 해당 elem를, 새로운 elem라면 NULL 반환
Function	새 elem를 hashtable에 삽입. 이미 같은 value를 갖고 있는 elem가 존재한다면 기존의 것을 새 elem로 대체함.

Prototype	`struct hash_elem *` `hash_cur (struct hash_iterator *i)`
Parameter	
Return	hash iterator의 현재 elem를 반환
Function	현재 hash iterator에 저장되어 있는 elem를 반환함.

Prototype	`void` `hash_first (struct hash_iterator *i, struct hash *h)`
Parameter	
Return	X
Function	hash iterator에 hashtable의 첫 번째 elem가 오도록 함.

Prototype	`struct hash_elem *` `hash_next (struct hash_iterator *i)`
Parameter	
Return	next hash elem를 반환
Function	hash iterator에 저장되어 있는 hash elem를 다음 elem로 바꾸고 그 elem를 반환함.

Prototype	`struct hash_elem *` `hash_find (struct hash *h, struct hash_elem *e)`
Parameter	
Return	찾고자 하는 elem가 있다면 그 elem의 pointer를, 없다면 NULL 반환
Function	hash_elem 찾기

Prototype	`void` `hash_apply (struct hash *h, hash_action_func *action)`
Parameter	
Return	X
Function	hash의 모든 elem에 action을 취함 (ex. square, triple)

IV. Bitmap

Prototype	`struct bitmap *` `bitmap_create (size_t bit_cnt)`
Parameter	
Return	bitmap 메모리 할당 성공하면 해당 bitmap의 pointer를, 실패하면 NULL 반환
Function	bitmap 새로 만들고 initialization 작업

Prototype	`void` `bitmap_set (struct bitmap *b, size_t idx, bool value)`
Parameter	
Return	X
Function	해당 bitmap idx의 bit를 value(true/false)로 설정함 void bitmap_set_all과 void bitmap_set_muliple도 비슷한 방식으로 진행됨

Prototype	`void` `bitmap_mark (struct bitmap *b, size_t bit_idx)`
Parameter	
Return	X
Function	해당 bitmap idx의 bit를 true로 설정함

Prototype	`void` `bitmap_flip (struct bitmap *b, size_t bit_idx)`
Parameter	
Return	X
Function	해당 bitmap idx의 bit를 toggle함.

Prototype
Parameter
	`size_t`
	`bitmap_scan (const struct bitmap *b, size_t start,`
	`size_t cnt, bool value)`

Return	group의 시작 idx 반환, group이 없으면 BITMAP_ERROR라는 숫자 반환 `(18446744073709551615UL)`
Function	START idx부터 cnt 비트만큼 value(T/F) 값을 가지는 group이 있는지 확인.

Prototype **Parameter**	`size_t` `bitmap_count (const struct bitmap *b, size_t start, size_t cnt, bool value)`
Return	비트맵에서 주어진 범위 내의 bit 중에서 value와 일치하는 bit의 개수를 반환
Function	이하 동문