Pintos Project 0-2

Pintos Data Structure

담당 교수 : 김영재 교수
학번 : 20160366
이름 : 김지수

Contents

* required functions에 대한 구현, 기타 필요에 의해 만든 functions와 main.c의 구현은 GitHub의 commit 번호: 89790768b25aa9e13b74f28a6d09c3b90daeb649 에서 한번에 확인할 수 있습니다.

I. Additional Implementation

- **void list_swap(struct list_elem *a, struct list_elem *b)**

parameter	위치를 바꾸고자 하는 item의 struct list_elem 포인터가 두 개 인자로 전달된다.
return	없음.
function	a와 b가 가리키는 item의 위치를 바꾸기 위해 a->prev->next와 a->next->prev가 b를 가리키고, b->prev->next와 b->next->prev가 a를 가리키도록 포인터 값을 변경한다. a->prev값과 a->next값을 a_prev, a_next 변수로 저장하여 순차적으로 포인터 값을 변경할 때 오류가 생기지 않도록 하며, a와 b가 연달아 존재하는 item인 경우를 테스트하여 일부 값 계산에 예외를 둔다.

- **void list_shuffle(struct list *list)**

parameter	element들의 위치를 섞고자 하는 list 포인터가 전달된다.
return	없음.
function	list_elem 포인터 a와 b를 선언하여 리스트의 첫 원소를 가리키고 시작한다. a는 한번씩 원소를 이동하고, b는 rand값으로 정해진 만큼 원소의 위치를 이동하여 자리를 바꾸고, 이는 a가 tail을 가리키기 전까지 반복된다.

- **unsigned hash_int_2 (int i)**

parameter	hash값을 얻고자 하는 int 값이 전달된다.
return	구해진 hash값을 unsigned로 반환한다.
function	4-byte로 구성된 i 값을, byte단위로 prime number(임의로 2,5,7,13을 weight로 사용)를 곱한 후 ^을 통해 값을 포개어 최종적으로 1byte짜리 unsigned 값을 만들어 반환한다.

- **struct bitmap * bitmap_expand(struct bitmap *bitmap, int size)**

parameter	bitmap 포인터와 int 타입의 값이 전달된다.
return	size가 연장된 bitmap의 포인터가 반환된다.
function	bitmap 포인터가 가리키는 bitmap의 bit_cnt를 새로운 size(기존 bit_cnt + size)로 업데이트 시키고, bits가 가리키는 공간을 그 사이즈로 새롭게 할당(realloc)한다. 그리고 새로 추가된 영역을 false로 초기화 한다.

· **about "main.c"**

global variables	**int list_recent_idx, int hash_recent_idx, int bm_recent_idx** → 각 자료구조에 대해 find함수를 통해 배열에서 원하는 이름의 자료를 찾는 순간 저장되는 위치 정보. 이는 delete 명령어 수행 시 최근에 찾은 원소를 배열에 접근하여 바로 삭제하는 용도로 활용된다. **struct list lists[10], struct hash hashtables[10], strut bitmap *bitmaps[10]** → 최대 10개까지 들어올 수 있는 size 10으로 자료를 저장하는 배열. bitmap은 bitmap.c에 구조체가 정의되어 있어 bitmap*의 배열로 구현하였다. 새로 자료를 추가하는 것 이외의 모든 연산은 배열에 접근하는 것으로 시작하며, 원하는 자료를 저장된 이름으로 찾을 수 있도록 **각각의 구조체(struct list, struct hash, struct bitmap)에 char name[10]의 field를 추가하였다.** **int list_available[10], int hash_available[10], int bitmap_available[10]** → 원소가 없는 자리에 find함수로 접근하는 경우 발생하는 segmentation fault를 방지하고자 원소가 저장되어 있는지 여부를 기록하는 배열이다. 원소가 있으면 1, 없으면 0으로 관리한다.
global methods	**char *rtrim(char *s), char *ltrim(char *s), char *trim(char *s)** → 문자열로 된 자료들의 명칭을 저장하고 접근하여 검색하는 경우 오류를 방지하고자 공백을 제거하는 함수들이다. 자세한 내용은 가장 처음 사용되는 LIST 파트에서 설명되어 있다. **int find_lists_idx(), int find_hashtables_idx(), int find_bitmaps_idx()** → 각 자료구조에 해당하는 배열에서 새로 추가하고자 하는 원소가 들어갈 index를 반환하는 함수. 자세한 내용은 보고서의 해당 자료구조 파트에서 설명되어 있다. **struct list *find_list (char *finding_name), struct hash *find_hashtable (char *finding_name), struct bitmap *find_list (char *finding_name)** → 각 자료구조에 해당하는 배열에서 원하는 이름을 가진 원소를 찾아서 그 포인터 값을 반환하는 함수이다. 자세한 내용은 보고서의 해당 자료구조 파트에 설명되어 있다.
main function 동작 원리	int main()함수 내에서는 기존에 존재하거나 다른 곳에서 정의한 함수들을 사용하였다. **char *getline(&line, &len, stdin)** → stdin으로 입력되는 명령어를 한번에 받아 char * line에 넣는다. **char *strtok(line, " ")** → line의 값을 공백 기준으로 잘라 앞부분을 char *command에 저장하여, 이 command에 받아진 명령어를 기준으로 if문 블록이 실행된다. **char *strtok(NULL, " ")** → 동작을 명령하는 구문 외의 추가적인 정보를 if 블록 내에서 받아 동작을 실행하면서 활용한다. **int strcmp(command, "create")** → 입력된 명령어에 따라 해당 if문 블록을 수행할지 결정할 때 사용한다. !strcmp(command, "명령어")가 true가 되면 해당 if문이 수행된다. **while문**을 돌며 명령어 **quit이 들어오기 전까지** ₩n를 기준으로 명령어가 한 줄 단위로 실행된다.

II. List

① Functions Implemented Additionally to Complement List Functions

<in main.c>

- **char *rtrim (char *s)**

parameter	문자열 s를 인자로 전달된다.
return	오른쪽 공백이 제거된 문자열이 반환된다.
function	전달받은 문자열의 오른쪽 공백을 제거하여 반환한다.

- **char * ltrim (char *s)**

parameter	문자열 s를 인자로 전달된다.
return	왼쪽 공백이 제거된 문자열이 반환된다.
function	전달받은 문자열의 왼쪽 공백을 제거하여 반환한다.

- **char *trim (char *s)**

parameter	문자열 s를 인자로 전달된다.
return	양쪽 공백이 제거된 문자열이 반환된다.
function	전달받은 문자열에 ltrim과 rtrim을 차례로 실행하여 양쪽 공백을 제거하여 반환한다.

- **int find_lists_idx ()**

parameter	없음.
return	새로 만들어진 list를 저장할 위치 인덱스(int)를 반환한다.
function	main.c에서는 struct list lists[10]의 배열 형태로, create된 list를 한 인덱스를 지정해 저장한다. 이 함수는 배열에서 새로 만들어진 list가 저장될 수 있는 인덱스를 찾아 반환하는 기능을 수행한다.

- **struct list * find_list(char *finding_name)**

parameter	lists 배열에서 찾고자 하는 list의 이름 문자열이 전달된다.
return	찾은 list의 포인터가 반환된다.

function	찾고자 하는 이름을 trim으로 공백을 제거한 후 lists들의 리스트를 하나씩 접근하여 같은 이름을 가진 리스트를 찾는다. 찾지 못한 경우에는 NULL을 반환한다. 저장되었다가 지운 리스트가 있는 경우, 해당 배열 인덱스 접근 시 발생하는 오류를 방지하기 위해 list_available 배열을 유지하여 아무것도 저장 되어있지 않은 경우에는 해당 인덱스로의 접근을 막는다. 또한, list_recent_idx라는 int값으로 일치하는 이름의 리스트를 찾아낸 인덱스 값을 보관한다. 이는 후에 delete로 찾아진 리스트를 삭제할 때 활용된다.

<in list.c>

- **struct list_elem * list_find_elem_by_index(struct list * l, int index)**

parameter	list 포인터와 int 값이 전달된다.
return	해당 index에 위치한 list_elem 포인터가 반환된다.
function	list에서 원하는 위치(index)에 있는 list_elem으로 이동하여 그 값을 반환한다.

- **bool less_list (const struct list_elem * a, const struct list_elem * b, void * aux)**

parameter	list_elem 포인터 두 개(a, b)와 보조 데이터 포인터 aux가 전달된다.
return	두 element의 대소 비교를 한 결과 Boolean 값이 반환된다.
function	list_elem 포인터 a와 b가 가리키고 있는 element를 list_item으로 캐스팅한 후, 둘의 data값의 크기를 비교하여, a의 data가 b의 data보다 작은 경우 true를, 아닌 경우 false를 반환한다.

② Functions Originally Implemented in list.c

< in list.h >

- **#define list_entry(LIST_ELEM, STRUCT, MEMBER) ₩**
  ((STRUCT *)((uint8_t *)&(LIST_ELEM)->next ₩
  - offsetof (STRUCT, MEMBER.next))

parameter	list_elem 포인터, 캐스팅을 원하는 구조체, 맨 앞의 인자가 해당하는 멤버 field명이 전달된다.
return	list_elem 포인터가 가리키는 element가 구조체로 캐스팅된, 해당 구조체의 포인터가 반환된다.
function	list_elem 포인터가 가리키는 element의 주소를 해당 구조체(list_item)의 포인터가 가리키도록 캐스팅하여, 해당 주소 값으로 list_item의 data field를 활용할 수 있도록 한다.

< in list.c >

- **static inline bool is_head(struct list_elem * elem)**

parameter	list_elem 포인터가 전달된다.
return	해당 element가 head인지 아닌지의 결과 Boolean 값이 반환된다.
function	elem이 가리키는 element가 Null이 아니고 head라면 true, 아니면 false값을 반환한다.

- **static inline bool is_interior(struct list_elem * elem)**

parameter	list_elem 포인터가 전달된다.
return	해당 element가 interior인지 아닌지의 결과 Boolean 값이 반환된다.
function	elem이 가리키는 element가 Null, head나 tail이 아닌 element라면 true, 아니면 false값을 반환한다.

- **static inline bool is_tail(struct list_elem * elem)**

parameter	list_elem 포인터가 전달된다.
return	해당 element가 tail인지 아닌지의 결과 Boolean 값이 반환된다.
function	elem이 가리키는 element가 Null이 아닌 tail이라면 true, 아니면 false값을 반환한다.

- **void list_init(struct list * list)**

parameter	초기화를 원하는 list의 포인터가 전달된다.
return	없음.
function	list 포인터가 할당 받은 공간에, head값과 tail값을 초기화한다. head->prev와 tail->next는 NULL로, head->next는 tail, tail->prev는 head를 가리키게 한다.

- **struct list_elem * list_begin(struct list * list)**

parameter	element의 시작 주소를 알고 싶은 list 포인터가 전달된다.
return	list의 시작 list_elem의 포인터가 반환된다.
function	전달받은 list포인터의 list의 head가 아닌, head 다음의 첫 원소의 위치를 반환한다.

- **struct list_elem * list_prev(struct list_elem * elem)**

parameter	직전 element를 알고자 하는 기준점 list_elem 포인터가 전달된다.

return	list_elem 포인터가 반환된다.
function	전달받은 list_elem포인터의 prev가 가리키는 list_elem 포인터를 반환한다. 전달받은 list_elem 포인터 그 자체가 head이면 결과는 undefined이다.

- **struct list_elem * list_next(struct list_elem * elem)**

parameter	다음 element를 알고자 하는 기준점 list_elem 포인터가 전달된다.
return	주어진 list_elem의 다음에 존재하는 list_elem 포인터가 반환된다.
function	전달받은 list_elem포인터의 next가 가리키는 list_elem 포인터를 반환한다. 전달받은 list_elem 포인터 그 자체가 tail이면 결과는 undefined이다.

- **struct list_elem * list_end(struct list * list)**

parameter	끝지점을 알고자 하는 list의 포인터가 전달된다.
return	list의 가장 끝에 있는 list_elem 포인터가 반환된다.
function	전달받은 list 포인터의 tail의 위치를 반환한다. 이는 정 방향 iterate시 반복문의 조건문에 사용될 수 있다.

- **struct list_elem * list_rbegin(struct list * list)**

parameter	reverse begin을 알고자 하는 list의 포인터가 전달된다.
return	list의 끝에서 시작한 첫 list_elem의 포인터가 반환된다.
function	전달받은 list 포인터의 tail 바로 앞 list_elem 포인터, tail->prev의 위치를 반환한다. reverse iterate시 첫 시작점을 반환하는 것이다.

- **struct list_elem * list_rend(struct list * list)**

parameter	reverse end를 알고자 하는 list의 포인터가 전달된다.
return	list의 reverse end지점에 있는 list_elem의 포인터가 반환된다.
function	전달받은 list 포인터의 head의 위치를 반환한다. reverse iterate시 마지막 지점을 확인하는 조건으로 사용된다.

- **struct list_elem * list_head(struct list * list)**

parameter	head 위치를 알고자 하는 list의 포인터가 전달된다.
return	head의 포인터가 list_elem 포인터 타입으로 반환된다.

function	전달받은 list 포인터의 head의 위치를 반환한다.

- **struct list_elem * list_tail(struct list * list)**

parameter	tail 위치를 알고자 하는 list의 포인터가 전달된다.
return	tail의 포인터가 list_elem 포인터 타입으로 반환된다.
function	전달받은 list 포인터의 tail의 위치를 반환한다.

- **void list_insert(struct list_elem * before, struct list_elem * elem)**

parameter	list_elem 포인터가 두 개(before, elem) 전달된다.
return	없음.
function	before가 가리키는 element 앞에 elem이 가리키는 element를 삽입한다. before->prev->next가 elem을 가리키고, before->prev도 elem을 가리키도록 설정한다.

- **void list_splice(struct list_elem * before, struct list_elem * first, , struct list_elem * last)**

parameter	list_elem 포인터가 세 개(before, first, last) 전달된다.
return	없음.
function	before가 가리키는 element 앞에, first가 가리키는 element부터 last->prev가 가리키는 element까지 통째로 삽입한다. before->prev->next가 first를 가리키고, before->prev가 last->prev를 가리키도록 설정한다.

- **void list_push_front(struct list * list, struct list_elem * elem)**

parameter	list 포인터와 그 list의 맨 앞에 추가하고자 하는 list_elem 포인터가 하나씩 전달된다.
return	없음.
function	해당 list의 begin과 elem을 인자로 list_insert함수를 수행하여, list의 begin 바로 앞, 즉 list의 head 바로 뒤에 elem가 가리키는 element를 삽입한다.

- **void list_push_back(struct list * list, struct list_elem * elem)**

parameter	list 포인터와 그 list의 맨 뒤에 추가하고자 하는 list_elem 포인터가 하나씩 전달된다.
return	없음.

function	해당 list의 end와 elem을 인자로 list_insert함수를 수행하여, list의 end 바로 앞, 즉 list의 tail 바로 앞에 elem가 가리키는 element를 삽입한다.

- **struct list_elem * list_remove(struct list_elem * elem)**

parameter	제거하고자 하는 list_elem 포인터가 전달된다.
return	제거한 element의 바로 뒤 list_elem 포인터가 반환된다.
function	해당 list_elem의 prev->next와 next->prev를 수정하여 elem을 해당 리스트에서 사라지게 하고, elem->next를 반환한다. elem이 list의 한 요소가 아닌 경우 undefined 한 행동이다.

- **struct list_elem * list_pop_front(struct list * list)**

parameter	가장 앞의 element를 꺼내고자 하는 list 포인터가 전달된다.
return	가장 앞에 있던 list_elem 포인터가 반환된다.
function	해당 list의 가장 앞의 element를 list에서 삭제하고, 그 element를 가리키는 list_elem을 반환한다.

- **struct list_elem * list_pop_back(struct list * list)**

parameter	가장 뒤의 element를 꺼내고자 하는 list 포인터가 전달된다.
return	가장 뒤에 있던 list_elem 포인터가 반환된다.
function	해당 list의 가장 뒤의 element를 list에서 삭제하고, 그 element를 가리키는 list_elem을 반환한다.

- **struct list_elem * list_front(struct list * list)**

parameter	가장 앞의 element를 알고자 하는 list 포인터가 전달된다.
return	가장 앞에 있는 list_elem의 포인터가 반환된다.
function	해당 list의 가장 앞의 element, 즉 head->next에 있는 list_elem 포인터를 반환한다.

- **struct list_elem * list_back(struct list * list)**

parameter	가장 뒤의 element를 알고자 하는 list 포인터가 전달된다.
return	가장 뒤에 있는 list_elem의 포인터가 반환된다.
function	해당 list의 가장 뒤의 element, tail->prev에 있는 list_elem 포인터를 반환한다.

- **size_t list_size(struct list * list)**

parameter	size를 알고자 하는 list 포인터가 전달된다.
return	list의 element의 개수가 size_t 타입(unsigned long)의 값으로 반환된다.
function	해당 list의 element 개수를 head->next 부터 tail->prev까지 세어서 총 개수를 반환한다.

- **bool list_empty(struct list * list)**

parameter	empty 여부를 확인하려는 list 포인터가 전달된다.
return	empty 확인의 결과 Boolean값이 반환된다.
function	head->next가 tail과 같은 지 확인하고, 같다면 해당 list의 원소 개수가 하나도 없는 것이므로 true를 반환하고, 그렇지 않으면 empty가 아니므로 false를 반환한다.

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

parameter	swap하고자 하는 list_elem 포인터의 포인터가 두 개(a, b) 전달된다.
return	없음.
function	a와 b가 가리키는 list_elem포인터를 서로 바꿔서 가리키도록 수행한다.

- **void list_reverse(struct list *list)**

parameter	역순으로 순서를 바꾸고자 하는 list 포인터가 전달된다.
return	없음.
function	iteration을 돌며 swap을 수행하여 list의 element순서를 역순으로 저장한다.

- **static bool is_sorted(struct list_elem * a, struct list_elem * b, list_less_func *less, void * aux)**

parameter	list_elem 포인터 두 개(a, b)와 less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	Boolean 값이 반환된다.
function	a가 가리키는 list_elem부터 b->prev가 가리키는 list_elem까지 aux 값에 따른 nondecreasing 순서로 정렬되어 있다면 true, 정렬되지 않았다면 false를 반환한다.

- **static struct list_elem * find_end_of_run(struct list_elem * a, struct list_elem * b, list_less_func *less, void * aux)**

parameter	list_elem 포인터 두 개(a, b)와 less function 포인터, 보조 데이터 aux 포인터가 전달된다.

return	nondecreasing 부분 리스트의 exclusive end 지점 list_elem의 포인터가 반환된다.
function	a가 가리키는 list_elem부터 b->prev가 가리키는 list_elem까지 순회를 돌다가, aux에 따른 nondecreasing 순서를 따르지 않는 element를 발견하면 그것(nondecreasing 부분 list의 exclusive end element의 포인터)을 반환한다.

- **static void implace_merge (struct list_elem * a0, struct list_elem * a1b0, struct list_elem * b1, list_less_func *less, void * aux)**

parameter	list_elem 포인터 세 개(a0, a1b0, b1)와 less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	없음.
function	a0가 가리키는 element부터 a1b0->prev가 가리키는 element까지의 부분 리스트와, a1b0이 가리키는 element부터 b1->prev가 가리키는 element까지의 부분 리스트를 병합한다. 병합 기준은 aux 값에 따른 nondecreasing 순서로, a0가 병합된 결과의 시작점이 된다.

- **void list_sorted(struct list * list, list_less_func *less, void * aux)**

parameter	list 포인터와 less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	없음.
function	해당 list의 element들을 merge sort 방식으로, aux값에 따른 nondecreasing 순서로 정렬한다.

- **void list_insert_ordered (struct list * list, struct list_elem * elem, list_less_func *less, void * aux)**

parameter	list 포인터와 list_elem 포인터, less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	없음.
function	해당 list를 순회하다 aux값에 따른 nondecreasing 순서에 알맞은 위치를 발견하여 elem이 가리키는 element를 삽입한다.

- **void list_unique (struct list * list, struct list * duplicates, list_less_func *less, void * aux)**

parameter	list 포인터 두 개(list, duplicates), less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	없음.
function	list를 순회하다 중복되는 element들을 발견하면 list에서는 삭제하고 duplicates에 push_back한다.

- **struct list_elem * list_max(struct list * list, list_less_func *less, void * aux)**

parameter	list 포인터, less function 포인터, 보조 데이터 aux 포인터가 전달된다.

return	max값을 가진 list_elem의 포인터가 반환된다.
function	list를 순회하며 찾은 가장 큰 값을 가진 list_elem의 포인터를 반환한다.

- **struct list_elem * list_min(struct list * list, list_less_func *less, void * aux)**

parameter	list 포인터, less function 포인터, 보조 데이터 aux 포인터가 전달된다.
return	min값을 가진 list_elem의 포인터가 반환된다.
function	list를 순회하며 찾은 가장 작은 값을 가진 list_elem의 포인터를 반환한다.

III.Hash Table

① Functions Implemented Additionally to Complement Hash Functions

<in main.c>

- **int find_hashtables_idx ()**

parameter	없음.
return	새로 만들어진 hash table를 저장할 위치 인덱스(int)를 반환한다.
function	main.c에서는 struct hash hashtables[10]의 배열 형태로 생성된 hash table들을 저장한다. 이 배열에서 다음에 새로 만들어진 hash table가 저장될 수 있는 인덱스를 찾아 반환해주는 기능을 수행한다.

- **struct list * find_hashtable(char *finding_name)**

parameter	hashtables 배열에서 찾고자 하는 hash table의 이름 문자열이 전달된다.
return	찾은 hash table의 포인터가 반환된다.
function	찾고자 하는 이름을 trim으로 공백을 제거한 후 hashtables의 리스트를 하나씩 접근하여 같은 이름을 가진 리스트를 찾는다. 찾지 못한 경우에는 NULL을 반환한다. 저장되었다가 지운 리스트가 있는 경우, 해당 배열 인덱스 접근 시 발생하는 오류를 방지하기 위해 hash_available 배열을 유지하여 아무것도 저장 되어있지 않은 인덱스로의 접근을 막는다. 또한, hash_recent_idx라는 int값으로 일치하는 이름의 리스트를 찾아낸 인덱스 값을 보관한다. 이는 후에 delete로 찾아진 리스트를 삭제할 때 활용된다.

<in hash.c>

- **unsigned int hash_function(const struct hash_elem * e, void * aux)**

parameter	hash elem 포인터와 보조 데이터 포인터 aux가 전달된다.
return	e에 주어진 hash_int를 수행한 결과값 hash가 unsigned int값으로 반환된다.
function	hash_elem 포인터가 가리키는 element에 대해 hash_int 함수를 실행한 hash값을 반환한다.

- **bool less_hash(const struct hash_elem * a, const struct hash_elem * b, void * aux)**

parameter	hash elem 포인터 두 개(a, b)와 보조 데이터 포인터 aux가 전달된다.
return	a와 b의 데이터에 대한 대소 비교 결과값 Boolean 값이 반환된다.
function	hash_elem 포인터 a와 b가 가리키고 있는 element의 data값의 크기를 비교하여, a의 data가 b의 data보다 작은 경우 true를, 아닌 경우 false를 반환한다.

- **void square (struct hash_elem * a, void * aux)**

parameter	hash_elem 포인터와 보조 데이터 aux가 전달된다.
return	없음.
function	hash_elem 포인터가 가리키는 element의 데이터를 제곱하여 저장한다.

- **void triple (struct hash_elem * a, void * aux)**

parameter	hash_elem 포인터와 보조 데이터 aux가 전달된다.
return	없음.
function	hash_elem 포인터가 가리키는 element의 데이터를 세제곱하여 저장한다.

- **void destructor (struct hash_elem * a, void * aux)**

parameter	hash_elem 포인터와 보조 데이터 포인터 aux가 전달된다.
return	없음.
function	hash_elem 포인터가 가리키는 element를 해당 리스트에서 삭제하기 위해 prev와 next의 포인터를 수정하고, 해당 element의 할당영역을 free시킨다.

② Functions Originally Implemented in list.c

<in hash.h>

- `#define hash_entry(HASH_ELEM, STRUCT, MEMBER) ₩`
  `((STRUCT *)((uint8_t *)&(HASH_ELEM)->list_elem ₩`
  `- offsetof (STRUCT, MEMBER.list_elem))`

parameter	hash_elem 포인터, 캐스팅을 원하는 구조체, 맨 앞의 인자가 해당하는 멤버 field명이 전달된다.
return	hash_elem 포인터가 가리키는 element가 구조체로 캐스팅된, 해당 구조체의 포인터가 반환된다.
function	hash_elem 포인터가 가리키는 element의 주소를 해당 구조체(hash_item)의 포인터가 가리키도록 캐스팅하여, 해당 주소 값으로 hash_item의 data field를 활용할 수 있도록 한다.

<in hash.c>

- `#define list_elem_to_hash_elem(LIST_ELEM) ₩`
  `list_entry(LIST_ELEM, struct hash_elem, list_elem)`

parameter	hash_elem으로 캐스팅하고자 하는 list_elem 포인터가 전달된다.
return	hash_elem 포인터 타입이 반환된다.
function	list_elem을 list_entry에 넣은 기능을 수행하는 매크로이다. list_elem 포인터가 가리키는 element의 주소를 해당 구조체(struct hash_elem)의 포인터가 가리키도록 캐스팅하여 반환한다.

- `bool hash_init(struct hash * h, hash_hash_func *hash, hash_less_func *less, void * aux)`

parameter	hash 포인터, hash_hash_func 함수 포인터, hash_less_func 함수 포인터와 보조 데이터 포인터 aux가 전달됨
return	init의 결과값 Boolean값을 반환한다.
function	hash 포인터로 받은 공간에 주어진 hash 함수, less 함수와 aux 값으로 초기화한다. 초기의 bucket개수는 4개로 설정하여 buckets 리스트에 그만큼의 공간을 할당한다. buckets 공간이 Null 이 아니면 hash_clear 후 true를 반환하고, 그렇지 않으면 false를 반환한다.

- `void hash_clear(struct hash * h, hash_action_func *destructor)`

parameter	clear하고자 하는 hash의 포인터와 clear에 사용될 hash_action_func 함수 포인터가 전달된다.
return	없음.
function	주어진 hash 포인터가 가리키는 hash의 element를 모두 제거하고 buckets에 저장된 list포인터를 초기화한다. 이 때 사용되는 action함수로 사용되는 destructor함수는 앞서 정의한 void destructor (struct hash_elem * a, void * aux)를 활용할 수 있다.

- **void hash_destroy(struct hash * h, hash_action_func *destructor)**

parameter	destroy하고자 하는 hash의 포인터와 destroy에 사용될 hash_action_func 함수 포인터가 전달된다.
return	없음.
function	주어진 hash 포인터가 가리키는 hash에 hash_clear element를 실행하여 element를 모두 제거하고 buckets에 저장된 list포인터 공간까지 free시킨다. 이 때 사용되는 action함수로 사용되는 destructor함수는 앞서 정의한 void destructor (struct hash_elem * a, void * aux)를 활용할 수 있다.

- **struct hash_elem * hash_insert(struct hash * h, struct hash_elem *new)**

parameter	hash 포인터와 그에 삽입할 hash_elem의 포인터가 전달된다.
return	hash_elem 포인터가 반환된다.
function	새로운 hash_elem가 가리키는 element를 hash포인터로 주어진 h에 추가한다. 동일한 element가 기존에 추가되어 있는 경우 그 위치를 반환하고, 기존에 없는 경우 new element를 추가한다.

- **struct hash_elem * hash_replace(struct hash * h, struct hash_elem *new)**

parameter	hash 포인터와 그에 추가할 hash_elem의 포인터가 전달된다.
return	hash_elem 포인터가 반환된다.
function	새로운 hash_elem가 가리키는 element를 hash포인터로 주어진 h에 추가하고 rehash를 실행한다. 동일한 element가 기존에 추가되어 있는 경우 기존의 것을 삭제하고 새로운 것을 추가하며, 동일한 element가 있다면 그 주소를, 없다면 NULL을 반환한다.

- **struct hash_elem * hash_find(struct hash * h, struct hash_elem *e)**

parameter	hash 포인터와 찾고자 하는 hash_elem의 포인터가 전달된다.
return	hash_elem 포인터가 반환된다.
function	h와 e로 find_bucket을 하고, 이 셋(h, e와 find_bucket의 결과)의 정보를 인자로 넘겨 find_elem을 수행한다. 해당 hash h에서, hash_elem포인터가 가리키는 element와 동일한 element를 bucket에서 발견하면 그 element의 주소를 반환하고, 발견하지 못하면 NULL을 반환한다.

- **struct hash_elem * hash_delete(struct hash * h, struct hash_elem *e)**

parameter	hash 포인터와 삭제하고자 하는 hash_elem의 포인터가 전달된다.
return	hash_elem 포인터가 반환된다.
function	h와 e로 find_bucket을 하고, 이 셋(h, e와 find_bucket의 결과)을 인자로 넘겨 find_elem을 수행한다. e가 가리키는 것과 동일한 element를 발견한다면 그것을 삭제 후 반환하고, 발견하지 못했다면 NULL을 반환한다.

- **void hash_apply(struct hash * h, hash_action_func *action)**

parameter	hash 포인터와 적용될 연산에 사용할 hash_action_func 포인터가 전달된다.
return	없음.
function	hash 포인터 h가 가리키는 hash의 모든 element에 대해 action함수를 적용하여 그 결과를 저장한다. action 함수로는 앞서 정의한 triple과 square를 사용할 수 있다.

- **void hash_first(struct hash_iterator * i, struct hash * h)**

parameter	hash_iterator 포인터와 iteration의 대상이 되는 hash 포인터가 전달된다.
return	없음.
function	hash의 element에 대해 iteration을 하기 위해 iterator를 초기화한다. iterator i가 가리키는 공간에 h의 hash, bucket, bucket의 첫번째 list의 head 주소를 저장한다.

- **struct hash_elem * hash_next(struct hash_iterator * i)**

parameter	현재 사용되고 있는 hash_iterator 포인터가 전달된다.
return	다음으로 사용될 hash_elem의 포인터가 반환된다.
function	hash_iterator 포인터가 가리키는 다음 hash_elem의 포인터를 반환한다.

- **struct hash_elem * hash_cur(struct hash_iterator * i)**

parameter	현재 사용되고 있는 hash_iterator 포인터가 전달된다.
return	현재 iterator가 가리키는 hash_elem의 포인터가 반환된다.
function	hash_iterator 포인터가 가리키는 현재의 hash_elem의 포인터가 반환된다. hash_first를 한 직후 iterator가 head를 가리킬 때, hash_cur하는 것은 undefined된 동작이다.

- **size_t * hash_size(struct hash * h)**

parameter	사이즈를 알고자 하는 hash의 포인터가 전달된다.
return	hash의 element 개수가, unsinged long으로 정의된 size_t 타입의 값으로 반환된다.
function	hash 포인터가 가리키는 hash의 elem_cnt를 반환한다.

- **bool hash_empty(struct hash * h)**

parameter	empty인지 알고자 하는 hash 포인터가 전달된다.

return	empty인지의 결과값 Boolean이 반환된다.
function	hash 포인터가 가리키는 hash의 elem_cnt가 0이면 true, 그렇지 않으면 false를 반환한다.

- **unsigned hash_bytes(const void * buf_, size_t size)**

parameter	hash_byte를 적용할 데이터를 가리키는 void 포인터와 그의 size_t 값이 전달된다.
return	hash_byte가 적용된 hash 결과가 unsigned으로 반환된다.
function	buf_에 있는 값을 1byte단위로 XOR를 적용하여 1 byte짜리 hash를 만들어 반환한다.

- **unsigned hash_string(const char * s_)**

parameter	hash_string을 적용할 문자열 s_값이 전달된다.
return	hash_string이 적용된 hash 결과가 unsigned으로 반환된다.
function	s_에 있는 값을 1byte단위로 XOR를 적용하여 1 byte짜리 hash를 만들어 반환한다.

- **unsigned hash_int(int i)**

parameter	hash값을 구하고자 하는 int i가 전달된다.
return	결과 hash 값이 unsigned으로 반환된다.
function	i의 위치와 i의 size를 인자로 한 hash_bytes의 값을 hash값으로 반환한다.

- **static struct hash_elem * find_elem(struct hash * h, struct list *bucket, struct hash_elem* e)**

parameter	hash 포인터, 인덱스 상 elem이 위치할 bucket의 list포인터, hash_elem 포인터가 전달된다.
return	hash_elem 포인터가 반환된다.
function	해당 hash h에서, hash_elem포인터가 가리키는 element와 동일한 element를 bucket에서 발견하면 그 element의 주소를 반환하고, 발견하지 못하면 NULL을 반환한다.

- **static inline size_t turn_off_least_1bit(size_t x)**

parameter	size_t type의 x가 전달된다.
return	size_t 값이 반환된다.

function	x와 (x-1)에 and 연산을 취함으로써 가장 마지막 bit를 1로 만든다. 그리고 반환되는 값의 나머지 비트에도 변화가 있을 수 있다.

- **static inline size_t is_power_of_2(size_t x)**

parameter	size_t type의 x가 전달된다.
return	size_t 값이 반환된다.
function	x가 2의 제곱수면 true, 아니면 false를 반환한다. x가 2의 제곱수인 것은, 0이 아니며 turn_off_least_1bit의 값이 0이 나오게 되는 것으로 판단할 수 있다.

- **static void rehash(struct hash * h)**

parameter	rehash를 하고자 하는 hash의 포인터가 전달된다.
return	없음.
function	새로운 bucket_cnt를, bucket당 최적의 element 개수(여기서는 2)가 되면서 bucket의 총 개수가 2의 제곱이 되도록 설정한다. bucket의 개수가 변한 경우 그만큼 새로운 공간을 할당하고 초기화한 후, 기존 bucket에 있던 element에 대해 find_bucket을 실행하여 hash에 새롭게 넣는다.

- **static void insert_elem(struct hash * h, struct list * bucket, struct hash_elem * e)**

parameter	hash 포인터, 인덱스 상 elem이 위치할 bucket의 list 포인터, hash_elem포인터가 전달된다.
return	없음.
function	해당 hash의 elem_cnt를 증가시키고, bucket이 가리키는 리스트의 맨 앞에 e가 가리키는 hash_elem을 삽입한다.

- **static void remove_elem(struct hash * h, struct hash_elem * e)**

parameter	hash 포인터, 제거하고자 하는 hash_elem의 포인터가 전달된다.
return	없음.
function	해당 hash의 elem_cnt를 감소시키고, e가 가리키는 hash_elem을 list_remove를 실행해 해당 리스트에서 제거한다.

IV.Bitmap

① Functions Implemented Additionally to Complement a Program

<in main.c>

- **int find_bitmaps_idx ()**

parameter	없음.
return	새로 만들어진 bitmap의 포인터를 저장할 위치 인덱스(int)를 반환한다.
function	main.c에서는 struct bitmap * bitmaps[10]의 배열 형태로 create된 bitmap의 포인터를 저장한다. 이 배열에서 다음에 새로 만들어진 bitmap의 포인트가 저장될 수 있는 인덱스를 찾아 반환해주는 기능을 수행한다.

- **struct list * find_bitmap(char *finding_name)**

parameter	bitmaps 배열에서 찾고자 하는 bitmap의 이름 문자열이 전달된다.
return	찾은 bitmap의 포인터가 반환된다.
function	찾고자 하는 이름을 trim으로 공백을 제거한 후 bitmap 포인터들의 리스트를 하나씩 접근하여 같은 이름을 가진 리스트를 찾는다. 찾은 결과가 없는 경우 NULL을 반환한다. 저장되었다가 지운 리스트가 있는 경우, 해당 배열 인덱스 접근 시 발생하는 오류를 방지하기 위해 bitmap_available 배열을 유지하여 아무것도 저장 되어있지 않은 인덱스로의 접근을 막는다. 또한, bm_recent_idx라는 int값으로 일치하는 이름의 리스트를 찾아낸 인덱스 값을 보관한다. 이는 후에 delete로 찾아진 리스트를 삭제할 때 활용된다.

<in bitmap.c>

- **char bitmap_get_name(struct bitmap * b)**

parameter	이름을 알고자 하는 bitmap의 포인터가 전달된다.
return	bitmap의 name영역에 있는 문자열이 반환된다.
function	bitmap 구조체가 bitmap.c에 구현되어 있어 main에서 직접적으로 bitmap의 contents에 접근할 수 없어 만든 함수이다. 주어진 포인터가 가리키는 bitmap의 name영역의 문자열을 반환한다.

- **void bitmap_set_name(struct bitmap * b, char *new_name)**

parameter	이름을 설정하고자 하는 bitmap의 포인터가 전달된다.
return	없음.

function	bitmap 구조체가 bitmap.c에 구현되어 있어 main에서 직접적으로 bitmap의 contents에 접근할 수 없기 때문에 만든 helper 함수이다. 포인터가 가리키는 bitmap의 name영역에 인자로 전달받은 이름을 저장한다.

- **void bitmap_print(struct bitmap * b)**

parameter	출력하고자 하는 bitmap의 포인터가 전달된다.
return	없음
function	bitmap 구조체가 bitmap.c에 구현되어 있어 main에서 직접적으로 bitmap의 contents에 접근할 수 없기 때문에 만든 helper 함수이다. 주어진 포인터가 가리키는 bitmap의 bits가 가리키는 내용을, element단위로 접근한 후 비트단위로 get_a_bit함수를 실행하여 비트 하나씩 출력한다.

- **int get_a_bit(unsigned long x, int n)**

parameter	unsigned long 값과 int 값이 하나씩 전달된다.
return	int type 값을 반환한다.
function	bitmap 구조체의 bit의 한 element에 든 elem_type으로 정의된 unsigned long값과 원하는 인덱스의 비트를 반환한다.

② Functions Originally Implemented in list.c

<in bitmap.c>

- **static inline size_t elem_idx (size_t bit_idx)**

parameter	bit의 인덱스가 size_t 값으로 전달된다.
return	속하는 element의 인덱스가 size_t로 반환된다.
function	bit가 저장된 위치를 알기 위해 bit_idx를 element에 속한 비트의 개수(ELEM_BIT)로 나누어, 속한 element의 인덱스를 구하여 반환한다.

- **static inline elem_type bit_mask (size_t bit_idx)**

parameter	bit의 인덱스가 size_t 값으로 전달된다.
return	element 값이 elem_type으로 반환된다.
function	bit_idx로 지정한 비트만 1로 켜져 있고 나머지 비트는 0인 element를 반환한다.

- **static inline size_t elem_cnt (size_t bit_cnt)**

parameter	bit의 개수가 size_t 값으로 전달된다.
return	element의 개수가 elem_type으로 반환된다.
function	해당 bit개수를 모두 저장할 수 있는 element의 개수를, 반올림하여 구한 후 반환한다.

- **static inline size_t byte_cnt (size_t bit_cnt)**

parameter	bit의 개수가 size_t 값으로 전달된다.
return	필요한 byte의 개수가 size_t 값으로 반환된다.
function	해당 bit개수로 필요한 총 byte수를, 곱하여 계산한 후 반환한다.

- **static inline elem_type last_mask (const struct bitmap *b)**

parameter	bitmap 포인터 b가 전달된다.
return	element 값이 elem_type으로 반환된다.
function	해당 bitmap의 마지막 element에 사용되는 비트들은 1로, 아닌 것들은 0으로 설정되어 한 element의 값이 반환된다.

- **struct bitmap * bitmap_create (size_t bit_cnt)**

parameter	bit의 개수가 size_t 값으로 전달된다.
return	생성된 bitmap 포인터가 반환된다.
function	원하는 bit개수만큼 byte 값을 계산하여 공간을 할당하고 비트를 모두 false로 설정한 후, 그 bitmap의 위치를 반환한다. 메모리 할당에 실패한 경우 NULL을 반환한다.

- **struct bitmap * bitmap_create_in_buf (size_t bit_cnt, void * block, size_t block_size)**

parameter	bit의 개수가 size_t 값으로, block이 할당된 곳의 포인터가, block 사이즈가 size_t로 전달된다.
return	생성된 bitmap 포인터가 반환된다.
function	원하는 bit개수와 전체 bitmap이 block_size에 들어간다면, 미리 할당된 block 포인터의 할당 공간에 비트 개수를 설정하고 비트를 모두 false로 초기화한 후, 그 bitmap의 위치를 반환한다.

- **size_t bitmap_buf_size (size_t bit_cnt)**

parameter	bit의 개수가 size_t 값으로 전달된다.
return	필요한 byte의 개수가 size_t 값으로 반환된다.
function	해당 bit개수에 맞는 bitmap을 생성하기 위해 필요한 총 byte수를 반환한다.

- **void bitmap_destroy (struct bitmap *b)**

parameter	destroy하고자 하는 bitmap의 포인터가 전달된다.
return	없음.
function	해당하는 bitmap의 공간을 모두 free시킨다.

- **size_t bitmap_size (const struct bitmap *b)**

parameter	size를 알고자 하는 bitmap의 포인터가 전달된다.
return	bitmap에 저장된 총 bit의 개수가 size_t 값으로 반환된다.
function	bitmap 포인터가 가리키는 b의 bit_cnt를 반환한다.

- **void bitmap_set (struct bitmap *b, size_t idx, bool value)**

parameter	bitmap 포인터, 비트의 인덱스가 size_t로, 원하는 value값이 boolean 값으로 전달된다.
return	없음.
function	해당하는 bitmap에서 idx 위치에 있는 비트를 접근하여, 인자로 넘어온 value값에 따라 bitmap_mark 또는 bitmap_reset을 실행하여 idx 위치의 비트를 원하는 boolean 값으로 설정한다.

- **void bitmap_mark (struct bitmap *b, size_t bit_idx)**

parameter	bitmap 포인터, 비트의 인덱스가 size_t로 전달된다.
return	없음.
function	해당하는 bitmap에서 bit_idx 위치가 속하는 elem를 계산하여 접근한 후, 1로 설정한다.

- **void bitmap_reset (struct bitmap *b, size_t bit_idx)**

parameter	bitmap 포인터, 비트의 인덱스가 size_t로 전달된다.
return	없음.

function	해당하는 bitmap에서 bit_idx 위치가 속하는 elem를 계산하여 접근한 후, 0으로 설정한다.

- **void bitmap_flip (struct bitmap *b, size_t bit_idx)**

parameter	bitmap 포인터, 비트의 인덱스가 size_t로 전달된다.
return	없음.
function	해당하는 bitmap에서 bit_idx 위치가 속하는 elem를 계산하여 접근한 후, 기존에 0이면 1로, 기존에 1이면 0으로 설정한다.

- **bool bitmap_test (const struct bitmap *b, size_t idx)**

parameter	bitmap 포인터, 비트의 인덱스가 size_t로 전달된다.
return	해당 비트의 값을 Boolean 값을 반환한다.
function	해당하는 bitmap에서 idx 위치가 속하는 elem의 비트들과, 원하는 값만 on된 bit_mask(idx)값에 and 연산을 취하여 0이면 false을, 1이면 true를 반환한다.

- **void bitmap_set_all (struct bitmap *b, bool value)**

parameter	bitmap 포인터, 원하는 값이 boolean으로 전달된다.
return	없음.
function	해당하는 bitmap의 모든 비트를 vaule값으로 설정한다.

- **void bitmap_set_multiple (struct bitmap *b, size_t start, size_t cnt, bool value)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로, 원하는 값이 boolean으로 전달된다.
return	없음.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼을 모두 vaule값으로 설정한다.

- **size_t bitmap_count (struct bitmap *b, size_t start, size_t cnt, bool value)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로, 원하는 값이 boolean으로 전달된다.
return	조건에 맞는 비트의 개수를 size_t로 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼 비트를 순회하며 vaule와 같은 것의 개수를 반환한다.

- **bool bitmap_contains (const struct bitmap *b, size_t start, size_t cnt, bool value)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로, 원하는 값이 boolean으로 전달된다.
return	Boolean 값을 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼을 순회하며, 한 개의 비트라도 vaule값과 같은 것이 존재한다면 true, 아니라면 false를 반환한다.

- **bool bitmap_any (const struct bitmap *b, size_t start, size_t cnt)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로 전달된다.
return	Boolean 값을 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼을 순회하며, 한 개의 비트라도 true가 존재한다면 true, 아니라면 false를 반환한다.

- **bool bitmap_none (const struct bitmap *b, size_t start, size_t cnt)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로 전달된다.
return	Boolean 값을 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼을 순회하며, 모두 false라면 true, 하나라도 true가 있다면 false를 반환한다.

- **bool bitmap_all (const struct bitmap *b, size_t start, size_t cnt)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로 전달된다.
return	Boolean 값을 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 총 cnt개수 만큼을 순회하며, 모두 true라면 true, 하나라도 false가 있다면 false를 반환한다.

- **size_t bitmap_scan (struct bitmap *b, size_t start, size_t cnt, bool value)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로, 원하는 값이 boolean으로 전달된다.
return	조건에 맞는 index를 size_t로 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 순회를 하며, value와 연속해서 cnt개수만큼 일치하는 그룹이 있다면 그 시작 인덱스를 반환한다. 그런 그룹이 없다면 BITMAP_ERROR를 반환한다.

- **size_t bitmap_scan_flip (struct bitmap *b, size_t start, size_t cnt, bool value)**

parameter	bitmap 포인터, 시작 인덱스와 개수가 size_t로, 원하는 값이 boolean으로 전달된다.
return	조건에 맞는 index를 size_t로 반환한다.
function	해당하는 bitmap에의 start 위치의 비트부터 순회를 하며, value와 연속해서 cnt개수만큼 일치하는 그룹이 있다면 그 그룹의 비트를 모두 !value로 재설정한 후 그 시작 인덱스를 반환한다. 그런 그룹이 없다면 BITMAP_ERROR를 반환한다. cnt가 0이라면 0을 반환한다.

- **size_t bitmap_file_size (const struct bitmap *b)**

parameter	bitmap 포인터가 전달된다.
return	필요한 byte수를 size_t로 반환한다.
function	해당하는 bitmap을 저장하기 위해 필요한, bit_cnt에 맞는 총 byte 개수를 반환한다.

- **bool bitmap_read (const struct bitmap *b, struct file *file)**

parameter	bitmap 포인터와 file 포인터가 전달된다.
return	Boolean 값을 반환한다.
function	file이 가리키는 content를 기존의 bitmap b가 가진 bit_cnt만큼 bits배열에 옮겨 쓴다. 성공적으로 옮겨 쓰면 true를, 그렇지 않으면 false를 반환한다.

- **bool bitmap_write (const struct bitmap *b, struct file *file)**

parameter	bitmap 포인터와 file 포인터가 전달된다.
return	Boolean 값을 반환한다.
function	bitmap b가 가진 bit_cnt만큼 bits배열의 비트 값들을 file에 옮겨 쓴다. 성공적으로 옮겨 쓰면 true를, 그렇지 않으면 false를 반환한다.

- **void bitmap_dump (const struct bitmap *b)**

parameter	bitmap 포인터가 전달된다.
return	없음.
function	bitmap이 가리키는 contents의 비트들을 16진수로 콘솔에 출력한다.