Project #1 : MyLib

담당 교수 : 이영민

학번 : 20190808

이름 : 방지혁

반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.

I. Additional Implementation

Prototype	int main();
Parameter	parameter를 받지 않는다.
Return	종료 시 0을 return한다.
Function	표준 입력을 받는데, getline 함수를 사용하여 한 줄씩 입력을 받는다. "quit"을 입력하면 반복문 탈출을 하며 종료한다. 또한, 각 입력된 문자열 끝의 개행 문자('₩n')를 널 문자('₩0')로 변경한다. strtok 함수를 사용하여 입력된 한 줄의 명령을 공백(' ')을 기준으로 토큰환한다. arg[0]은 주 명령어(create, delete, dumpdata 등), arg[1]: 자료 구조 타입(list, bitmap, hashtable), arg[2]는 추가 매개변수들 이런 식이다. 명령어가 "create", "delete", "dumpdata"인 경우 두 번째 인자인 arg[1]를 확인하여 자료구조를 판단하고 해당 자료구조에 따른 적절한 함수를 호출한다. 그 외의 경우에는 명령어의 구조를 파악하여 첫번째 인자의 명령 형식을 이용해서, list, hash, bitmap 글자수만큼 문자열을 비교하여 적절한 함수를 호출한다. 마지막으로 quit으로 반복문을 탈출하면, line을 free해준다.

Prototype	void list_functions(char arg[][30]);;
Parameter	char 배열을 parameter로 받는다.
Return	void 형으로 값을 return 하지 않는다.
Function	list table를 create 하는 경우, 새로운 list_table 구조체와 list 구조체를 할당하고, 리스트 이름을 char 배열에 저장하고, linked list 방식으로 list_table_head에 새로운 list_table을 추가한다. 여기서 2가지 방식으로 나뉘는데 list_table_head가 비어있었거나, 아닌 경우이다. 전자의 경우 새로 malloc한 list_table pointer를 list_table_ehad에 넣고, 후자의 경우 맨 뒤에 삽입한다. dumpdata(값들을 표시)의 경우, list_table linked list를 순회하다

	가 같은 이름을 찾으면 탈출해서 해당 리스트의 모든 요소를 순회한다. 비어있는 리스트인 경우 아무것도 출력하지 않기에 바로 종료하고 그 외의 경우 for문에서 list_begin, list_end, list_next 함수를 사용하여 처음과 끝 그리고 다음 pointer를 지정해주고 list_entry 함수로 새로운 포인터가 list_elem 가 아니라, list_item을 가리키도록 만들어서 값을 출력한다. delete의 경우 비어있지 않는 경우 리스트의 모든 항목을 먼저 제거하고, list_table 구조체를 연결 리스트에서 제거한다. 그리고 명령에 따라 이외의 다른 내부적인 function들을 수행하는데 이에 앞서 먼저 operation을 수행할 list_table을 찾는다.

Prototype	void hash_functions(char arg[][30]);
Parameter	char 배열을 parameter로 받는다.
Return	void 형으로 값을 return 하지 않는다.
Function	hash_table을 create하는 경우, 새로운 hash_table 구조체와 hash 구조체를 할당하고, hash_init 함수를 호출하여 해시 함수와 비교 함수를 설정한 후 이름을 저장한다. 이후 hash_table_head에 연결 리스트 방식으로 추가하는데, hash_table_head가 NULL이면 바로 할당하고 그렇지 않으면 마지막 노드를 찾아 연결한다. 해시 테이블의 dumpdata 수행 시에는 먼저 지정된 이름의 hash_table을 찾고, 해시 테이블이 비어 있지 않은 경우 hash_iterator를 사용하여 모든 요소를 순회하면서 값을 출력한다. 이 과정에서 hash_first와 hash_next 함수를 사용하여 요소들에 순차적으로 접근한다. delete 시에는 hash_destroy 함수로 해시 테이블 내부의 모든 요소를 해제한 후, hash_table 구조체를 연결 리스트에서 제거한다. 이때도 첫 번째 노드인 경우와 그렇지 않은 경우를 구분하여 처리한다. 다른 경우 먼저 지정된 이름의 hash_table을 찾은 후, 해당 해시 테이블에 대해 hash_insert, hash_find, hash_delete 등의 함수를 호출하여 작업을 수행한다. 특히 해시 테이블에 요소를 추가하거나 검색할 때는 다른 자료 구조들과 다르게 새로운 hash_elem 구조체를 할

	당하여 값을 설정한 후 해당 작업을 수행한다.

Prototype	void bitmap_functions(char arg[][30]);
Parameter	char 배열을 parameter로 받는다.
Return	void 형으로 값을 return 하지 않는다.
Function	bitmap_table을 create하는 경우, 새로운 bitmap_table 구조체와 bitmap 구조체를 할당하고, 이름을 저장한 후 linked list 방식으로 bitmap_table_head에 추가한다. 이 과정도 두 가지 경우로 나뉘는데, bitmap_table_head가 NULL인 경우 새로 할당한 구조체를 bitmap_table_head에 직접 할당하고, 그렇지 않은 경우 마지막 노드까지 순회한 후 마지막 노드의 right 포인터에 새 구조체를 연결한다. Dumpdata의 경우 특별하게 bitmap_table 연결 리스트를 순회하며 지정된 이름의 비트맵을 찾고, 해당 비트맵의 모든 비트를 순회하면서 각 비트의 상태에 따라 0 또는 1을 출력한다. 이 과정에서 bitmap_test 함수를 사용하여 각 비트의 상태를 확인한다. Bitmap delete 시에는 bitmap_destroy 함수로 비트맵 자체를 먼저 해제하고, bitmap_table 구조체를 연결 리스트에서 제거한다. 이 과정에서도 첫 번째 노드인 경우와 중간 노드인 경우를 구분하여 처리한다. 다른 비트맵 작업을 수행할 때는 먼저 지정된 이름의 bitmap_table을 찾아 해당 비트맵에 접근한 후 bitmap_set, bitmap_mark, bitmap_reset 등 각종 비트맵 조작 함수를 호출하여 작업을 수행한다.

추가한 항목들	설명

struct list_table{
char name[20];
struct list* list;
struct list_table* right;
};
	list 구조체 pointer를 멤버로 갖는 list_table 구조체를 추가했다. Char 배열 name을 추가하여 입력 받은 명령과 저장된 이름을 비교해서 operation이 행해질 list를 찾을 수 있도록 했고, 다음 list_table pointer를 추가하여 list_table 리스트를 순

	회할 수 있도록 하였다.

struct hash_table{
  char name[20];
  struct hash* hash;
  struct hash_table *right;
};
	hash 구조체 pointer를 멤버로 갖는 hash_table 구조체를 추가했다. Char 배열 name을 추가하여 입력 받은 명령과 저장된 이름을 비교해서 operation이 행해질 hash을 찾을 수 있도록 했고, 다음 hash_table pointer를 추가하여 hash_table 리스트를 순회할 수 있도록 하였다.

struct bitmap_table{
  char name[20];
  struct bitmap *bitmap;
  struct bitmap_table *right;
};
	bitmap 구조체 pointer를 멤버로 갖는 bitmap_table 구조체를 추가했다. Char 배열 name을 추가하여 입력 받은 명령과 저장된 이름을 비교해서 operation이 행해질 bitmap을 찾을 수 있도록 했고, 다음 bitmap_table pointer를 추가하여 bitmap table 리스트를 순회할 수 있도록 하였다.

struct list_table* list_table_head = NULL;
 	
struct hash_table* hash_table_head = NULL;
 	
struct bitmap_table* bitmap_table_head = NULL;
	순회를 하기 위한 list_table, hash_table, bitmap_table 포인터의 가장 처음 주소로 전역 변수로 선언하였고, 처음에는 비어있기 때문에 NULL을 할당하였다.

II. List

Prototype	void list_swap(struct list_elem *a, struct list_elem *b);
Parameter	parameter로 교환할 요소인 list_elem 자료형의 pointer a와 b를 받는다.

Return	void형으로 return하지 않는다.
Function	NULL 포인터가 parameter로 입력되거나 두 요소가 동일하면 아무것도 하지 않고 종료한다. a가 b 앞에 있는 경우, b가 a 앞에 있는 경우, a와 b가 멀리 떨어져 있는 일반적인 경우에 대해 세 가지로 나누어 list_elem a와 b의 prev, next pointer를 업데이트하여 위치를 교환한다.

Prototype	void list_shuffle(struct list *list);
Parameter	parameter로 list 자료형의 pointer list를 받는다
Return	void형으로 return하지 않는다.
Function	우선 리스트 사이즈를 list_size함수를 이용하여 구해서 size_t size 변수에 넣는다. 섞을 요소가 1개 이하이면 종료한다. 난순생성기를 초기화 하고, 리스트 요소들의 포인터를 저장할 배열을 동적으로 할당한다. List_begin, list_end, list_next 함수를 사용하여 리스트의 모든 요소를 순회하며 해당 배열에 저장한다. 각 index j에 대해 임의의 위치 k를 정해서, list_swap함수로 위치를 교환한다. 정리하자면, list에 있는 list_elem들을 fisher-yates 알고리즘을 사용하여 shuffle한다.

Prototype	bool list_less (const struct list_elem *a, const struct list_elem *b, void *aux);
Parameter	list_elem형 ptr인 a와 b, 그리고 추가적인 parameter aux를 parameter로 받는다.
Return	boolean 자료형을 return한다.
Function	item_a의 data와 item_b의 data를 list_entry 매크로를 이용하여 비교하여 전자가 더 작으면 true를 return, 그외의 경우에는 false를 return한다.

III.Hash Table

Prototype	unsigned hash_hash(const struct hash_elem *e, void *aux);
Parameter	hash_elem pointer인 e와 aux인 보조데이터를 parameter로 받는다.
Return	해시값을 계산하여 unsigned 자료형을 return한다.
Function	해시 요소의 data 멤버에 대한 해시 값을 계산하는데, hash_int 함수를 호출하여 인자로 data를 넣고 이 hash_int함수에서는 hash_bytes 함수를 호출하여 parameter로 받은 정수 값에 대한 해시 값을 구한다.

Prototype	bool hash_less(const struct hash_elem *a, const struct hash_elem *b, void *aux);
Parameter	hash_elem pointer인 a와 b 그리고 보조데이터인 aux를 parameter로 받는다.
Return	bool 형을 return하는데 만약 a의 data가 b의 것보다 작으면, true 아닌 경우 false를 return한다.
Function	hahs_elem 구조체의 data멤버를 확인하여 대소 비교를 실시한다.

Prototype	void hash_triple(struct hash_elem *e, void *aux);
Parameter	hash_elem pointer인 e와 보조데이터인 aux를 parameter로 받는다.
Return	void형으로 값을 return하지 않는다.
Function	hash_elem pointer의 e에서 data 멤버를 확인하고 이를 임시 변수인 temp에 넣는다. 이를 세제곱하여 new_result에 넣고. new_result는 다시 e의 data멤버에 넣는다. hash_apply 함수와 함께 사용하여 예를 들어 hash_apply(hash_ptr, hash_triple) 같은 방식으로 해시 테이블의 모든 요소에 적용할 수 있다

.

Prototype	void hash_square(struct hash_elem *e, void *aux);

Parameter	hash_elem pointer인 e와 보조데이터인 aux를 parameter로 받는다.
Return	void형으로 값을 return하지 않는다.
Function	hash_elem pointer의 e에서 data 멤버를 확인하고 이를 임시 변수인 temp에 넣는다. 이를 제곱하여 new_result에 넣고. new_result는 다시 e의 data멤버에 넣는다. hash_apply 함수와 함께 사용하여 예를 들어 hash_apply(hash_ptr, hash_square) 같은 방식으로 해시 테이블의 모든 요소에 적용할 수 있다

Prototype	void hash_free(struct hash_elem *e, void *aux);
Parameter	hash_elem pointer인 e와 보조데이터인 aux를 parameter로 받는다.
Return	void형으로 값을 return하지 않는다.
Function	요소 자체의 memory를 free해준다. hash_action_func 타입의 함수로, hash_clear와 hash_destroy의 함수에서 parameter로 입력받아 해시 요소를 파괴하거나 내용을 비울 때 사용된다.

Prototype	unsigned hash_int_2(int num);
Parameter	해시할 정수의 값을 parameter로 받는다.
Return	unsigned 형으로 계산된 해시 값을 return한다.
Function	DJB2 알고리즘을 사용하여 hash 값을 계산하여 return한다. 초기 값은 5381로 설정하고, int형 num의 주소를 unsigned char * 타입으로 캐스팅한다. 이렇게 num의 개별 바이트를 순회할 수 있다. (hash << 5) + hash에 bytes[j]를 더하여 입력 숫자의 각 바이트 값을 해시에 넣는다. 최종적으로 해시 값을 return한다.

IV. Bitmap

Prototype	struct bitmap *bitmap_expand (struct bitmap *bitmap, int size);

Parameter	구조체 bitmap pointer bitmap과 더 늘리고(추가하고) 싶은 int형 변수 size를 parameter로 받는다.
Return	size가 증가한 bitmap pointer를 return한다.
Function	우선, NULL 비트맵 ptr를 parameter로 받았거나 음수 크기로 확장하려는 시도를 막기 위해, 이런 경우 즉시 NULL을 return한다. 이후, 현재 비트맵의 크기를 bitmap_size 함수로 가져와서 이를 parameter로 받은 요청된 크기에 더해 새로운 사이즈 new_size를 구한다. Realloc 함수를 사용하여 메모리를 새 크기로 재할당한다. 그리고 실패 시 마찬가지로 NULL을 return한다. 성공시 bitmap 구조체의 멤버인 bits는 새 메모리 위치로, bit_cnt는 새 크기로 업데이트해준다. 그리고 bitmap_set_multiple 함수를 사용하여 새로 늘린 bit들은 모두 false, 즉 0으로 초기화해준다. 원래 크기부터 시작해 새로 추가된 만큼 설정해주는 것이다. 마지막으로 size를 늘린 bitmap pointer를 return한다.