**Project #1 : MyLib**

|  |  |
| --- | --- |
| 담당 교수 : | 이영민 |
| 학번 : | 20231508 |
| 이름 : | 강유영 |
|  |  |

1. **Additional Implementation** (구현을 위해 main.c에 추가한 함수들)

**<구조체 및 list, hash, bitmap을 모두 처리하는 함수들>**

* DataType : 리스트, 해시 테이블, 비트맵의 종류를 구분하는 열거형이다. (LIST, HASH\_TABLE, BITMAP).
* DataStruct : 이름(name), 자료형(type), 자료 포인터(data)를 가지는 공통 구조체이다. 하나의 배열에서 다양한 타입의 구조체를 다루기 위해 사용된다.
* list\_struct, hash\_struct, bitmap\_struct : 각각 리스트, 해시 테이블, 비트맵을 저장하는 전역 배열이다. 각 구조체는 최대 MAX\_STRUCTS개까지 저장 가능하며, l\_cnt, h\_cnt, b\_cnt로 개수를 관리한다.
* insertion\_order: 리스트에 삽입된 항목들의 순서를 저장하는 포인터 배열이다. insertion\_count는 현재 저장된 항목 수를 나타낸다.

|  |  |
| --- | --- |
| **Prototype** | void to\_lowercase(char \*str) |
| **Parameter** | str : 변환할 문자열 (char 배열) |
| **Return** | x |
| **Function** | 주어진 문자열의 모든 문자를 소문자로 변환한다. |

|  |  |
| --- | --- |
| **Prototype** | void delete\_struct(const char \*name) |
| **Parameter** | name : 삭제할 데이터 구조체 이름 |
| **Return** | x |
| **Function** | 이름에 해당하는 리스트, 해시 테이블, 또는 비트맵을 찾아 메모리를 해제하고 배열에서 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdata(const char \*name) |
| **Parameter** | name : 출력할 데이터 구조체 이름 |
| **Return** | x |
| **Function** | 이름에 해당하는 리스트, 해시 테이블 또는 비트맵의 현재 상태를 표준 출력으로 출력한다. |

**<list 관련 추가 함수들>**

|  |  |
| --- | --- |
| **Prototype** | void list\_dump(const struct list \*list) |
| **Parameter** | list : 출력할 리스트 포인터 |
| **Return** | x |
| **Function** | 리스트의 모든 요소를 순서대로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | bool compare\_items\_list(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux) |
| **Parameter** | a : 첫 번째 리스트 요소 포인터, b : 두 번째 리스트 요소 포인터, aux : 추가 인자 |
| **Return** | true (a < b), false (a ≥ b) |
| **Function** | 리스트 정렬을 위한 비교 함수로, 두 요소의 값을 비교하여 정렬 기준을 제공한다. |

|  |  |
| --- | --- |
| **Prototype** | void create\_list(const char \*name) |
| **Parameter** | name : 생성할 리스트의 이름 |
| **Return** | x |
| **Function** | 지정한 이름으로 새로운 리스트를 생성하고 전역 리스트 배열에 추가한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\* get\_list\_by\_name(const char \*name) |
| **Parameter** | name : 찾고자 하는 리스트의 이름 |
| **Return** | 리스트 포인터 또는 NULL |
| **Function** | 이름에 해당하는 리스트를 리스트 배열에서 찾아 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 리스트 이름, value : 리스트에 추가할 정수 값 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 앞에 새로운 값을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 리스트 이름, value : 리스트에 추가할 정수 값 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 뒤에 새로운 값을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_front\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 앞 요소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_back\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 뒤 요소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_front\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 앞 값을 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_back\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 맨 뒤 값을 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 리스트 이름, value : 삽입할 정수 값 |
| **Return** | x |
| **Function** | 정렬된 리스트에 정렬 순서를 유지하며 값을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_empty\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트가 비어 있는지 여부를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_size\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 요소 개수를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_at(struct list \*lst, int idx, int value) |
| **Parameter** | lst : 대상 리스트 포인터, idx : 삽입할 위치 인덱스, value : 삽입할 정수 값 |
| **Return** | x |
| **Function** | 리스트의 지정된 위치에 새 요소를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_remove\_by\_idx(const char \*name, int idx) |
| **Parameter** | name : 대상 리스트 이름, idx : 제거할 요소의 인덱스 |
| **Return** | x |
| **Function** | 지정한 리스트에서 해당 인덱스에 있는 요소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 순서를 반대로 뒤집는다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트의 요소들을 무작위로 섞는다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort\_by\_name(const char \*name) |
| **Parameter** | name : 대상 리스트 이름 |
| **Return** | x |
| **Function** | 지정한 리스트를 오름차순으로 정렬한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice\_by\_name(const char \*name, int idx, const char \*name2, int start\_idx, int end\_idx) |
| **Parameter** | name : 삽입 대상 리스트 이름, idx : 삽입 위치 인덱스, name2 : 잘라낼 리스트 이름, start\_idx : 잘라낼 시작 인덱스, end\_idx : 잘라낼 종료 인덱스 |
| **Return** | x |
| **Function** | 다른 리스트에서 지정한 구간의 요소들을 잘라 현재 리스트에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap\_by\_idx(const char \*name, int idx1, int idx2) |
| **Parameter** | name : 대상 리스트 이름, idx1 : 교환할 첫 번째 인덱스, idx2 : 교환할 두 번째 인덱스 |
| **Return** |  |
| **Function** | 지정한 리스트의 두 인덱스에 위치한 요소를 서로 교환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique\_by\_name(const char \*name, const char \*duplicate\_name) |
| **Parameter** | name : 대상 리스트 이름, duplicate\_name : 비교할 리스트 이름 (NULL일 수 있음) |
| **Return** | x |
| **Function** | 리스트에서 중복 요소를 제거한다. 다른 리스트와 비교하여 중복도 제거 가능하다. |

**<hash 관련 추가 함수들>**

|  |  |
| --- | --- |
| **Prototype** | void get\_hash\_by\_name(const char \*name) |
| **Parameter** | name : 찾고자 하는 해시 테이블 이름 |
| **Return** | 해시 테이블 포인터 또는 NULL |
| **Function** | 이름에 해당하는 해시 테이블을 전역 배열에서 찾아 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_dump\_by\_name(const char \*name) |
| **Parameter** | name : 찾고자 하는 해시 테이블 이름 |
| **Return** | x |
| **Function** | 해시 테이블의 모든 요소를 순회하며 값을 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned my\_hash\_func(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | e : 해시 요소 포인터, aux : 추가 인자 |
| **Return** | 계산된 해시값 (unsigned) |
| **Function** | 해시 테이블에서 사용할 사용자 정의 해시 함수. 요소의 정수 값을 기반으로 해시값을 계산한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_equal\_check(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux) |
| **Parameter** | a : 첫 번째 해시 요소, b : 두 번째 해시 요소, aux : 추가 인자 |
| **Return** | true (데이터가 다름), false (데이터가 같음) |
| **Function** | 해시 테이블 요소 비교 함수로, 중복 여부를 판단하는 데 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | void free\_hash\_elem(struct hash\_elem \*e, void \*aux) |
| **Parameter** | e : 해시 요소 포인터, aux : 추가 인자 |
| **Return** | x |
| **Function** | 할당된 해시 요소의 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void create\_hashtable(const char \*name) |
| **Parameter** | name : 생성할 해시 테이블 이름 |
| **Return** | x |
| **Function** | 새로운 해시 테이블을 생성하고 전역 배열에 등록한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_insert\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 해시 테이블 이름, value : 삽입할 정수 값 |
| **Return** | x |
| **Function** | 해시 테이블에 값을 삽입한다. 중복된 값은 삽입되지 않는다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_dump(struct hash \*h) |
| **Parameter** | h : 해시 테이블 포인터 |
| **Return** | x |
| **Function** | 해시 테이블의 요소들을 정렬하여 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | e : 해시 요소 포인터, aux : 추가 인자 |
| **Return** | x |
| **Function** | 해시 요소의 값을 제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | e : 해시 요소 포인터, aux : 추가 인자 |
| **Return** | x |
| **Function** | 해시 요소의 값을 세제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply\_by\_name(const char \*name, const char \*operation) |
| **Parameter** | name : 대상 해시 테이블 이름, operation : "square" 또는 "triple |
| **Return** | x |
| **Function** | 지정한 연산(square 또는 triple)을 해시 테이블의 모든 요소에 적용한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_delete\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 해시 테이블 이름, value : 삭제할 값 |
| **Return** | x |
| **Function** | 해시 테이블에서 해당 값을 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_empty\_by\_name(const char \*name) |
| **Parameter** | name : 대상 해시 테이블 이름 |
| **Return** | x |
| **Function** | 해시 테이블이 비어 있는지를 출력한다 ("true" 또는 "false") |

|  |  |
| --- | --- |
| **Prototype** | void hash\_size\_by\_name(const char \*name) |
| **Parameter** | name : 대상 해시 테이블 이름 |
| **Return** | x |
| **Function** | 해시 테이블의 현재 요소 개수를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear\_by\_name(const char \*name) |
| **Parameter** | name : 대상 해시 테이블 이름 |
| **Return** | x |
| **Function** | 해시 테이블의 모든 요소를 제거하고 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_find\_by\_name(const char \*name, int value |
| **Parameter** | name : 대상 해시 테이블 이름, value : 검색할 값 |
| **Return** | x |
| **Function** | 해시 테이블에서 값을 찾아 출력한다 (없으면 출력 없음). |

|  |  |
| --- | --- |
| **Prototype** | void hash\_replace\_by\_name(const char \*name, int value) |
| **Parameter** | name : 대상 해시 테이블 이름, value : 교체할 값 |
| **Return** | x |
| **Function** | 해시 테이블에서 기존 값을 삭제하고 새로운 값을 삽입한다. |

**<bitmap 관련 추가 함수들>**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* get\_bitmap\_by\_name(const char \*name) |
| **Parameter** | name : 비트맵 이름 |
| **Return** | 비트맵 포인터 또는 NULL |
| **Function** | 전역 배열에서 이름에 해당하는 비트맵을 찾아 포인터를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump\_by\_name(const char \*name) |
| **Parameter** | name : 출력할 비트맵 이름 |
| **Return** | x |
| **Function** | 비트맵의 각 비트를 0 또는 1로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void create\_bitmap(const char \*name, int bit\_cnt) |
| **Parameter** | name : 생성할 비트맵 이름, bit\_cnt : 비트맵의 비트 개수 |
| **Return** | x |
| **Function** | 지정한 이름과 크기로 비트맵을 생성하여 배열에 추가한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark\_by\_name(const char \*name, size\_t idx) |
| **Parameter** | name : 대상 비트맵 이름, idx : 마킹할 비트 인덱스 |
| **Return** | x |
| **Function** | 지정한 위치의 비트를 true로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset\_by\_name(const char \*name, size\_t idx) |
| **Parameter** | name : 대상 비트맵 이름, idx : 리셋할 비트 인덱스 |
| **Return** | x |
| **Function** | 지정한 위치의 비트를 false로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_all\_by\_name(const char \*name, int start, int count) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 확인할 비트 개수 |
| **Return** | x |
| **Function** | 지정한 범위의 비트가 모두 true인지 여부를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_any\_by\_name(const char \*name, int start, int count) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 확인할 비트 개수 |
| **Return** | x |
| **Function** | 지정한 범위의 비트 중 하나라도 true인지 여부를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_contains\_by\_name(const char \*name, int start, int count, bool value) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 확인할 비트 개수, value : 검사할 값 (true 또는 false) |
| **Return** | x |
| **Function** | 지정한 범위의 비트가 모두 특정 값으로 채워져 있는지 여부를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_count\_by\_name(const char \*name, int start, int count, bool value) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 검사할 비트 개수, value : 기준 값 (true 또는 false) |
| **Return** | x |
| **Function** | 지정한 구간에서 value 값과 같은 비트 개수를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_hex\_dump\_by\_name(const char \*name) |
| **Parameter** | name : 출력할 비트맵 이름 |
| **Return** | x |
| **Function** | 비트맵의 내부 비트를 16진수 형식으로 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all\_by\_name(const char \*name, bool value) |
| **Parameter** | name : 대상 비트맵 이름, value : 설정할 값 (true 또는 false) |
| **Return** | x |
| **Function** | 비트맵의 모든 비트를 지정한 값으로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip\_by\_name(const char \*name, size\_t idx) |
| **Parameter** | name : 대상 비트맵 이름, idx : 반전할 비트 위치 |
| **Return** | x |
| **Function** | 지정한 위치의 비트를 반전시킨다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_none\_by\_name(const char \*name, int start, int count) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 검사할 비트 개수 |
| **Return** | x |
| **Function** | 지정한 범위의 비트가 모두 false인지 여부를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_scan\_and\_flip\_by\_name(const char \*name, int start, int count, bool value) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 검사할 비트 개수, value : 찾고자 하는 값 (true 또는 false) |
| **Return** | x |
| **Function** | 지정된 값이 연속으로 있는 구간을 찾아 반전시키고 시작 인덱스를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_scan\_by\_name(const char \*name, int start, int count, bool value) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 검사할 비트 개수, value : 찾고자 하는 값 (true 또는 false) |
| **Return** | x |
| **Function** | 조건을 만족하는 비트 구간의 시작 인덱스를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_by\_name(const char \*name, size\_t idx, bool value) |
| **Parameter** | name : 대상 비트맵 이름, idx : 설정할 비트 인덱스, value : 설정할 값 (true 또는 false) |
| **Return** | x |
| **Function** | 지정한 위치의 비트를 특정 값으로 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple\_by\_name(const char \*name, int start, int count, bool value) |
| **Parameter** | name : 대상 비트맵 이름, start : 시작 인덱스, count : 설정할 비트 개수, value : 설정할 값 (true 또는 false) |
| **Return** | x |
| **Function** | 지정한 범위의 여러 비트를 일괄 설정한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_size\_by\_name(const char \*name) |
| **Parameter** | name : 대상 비트맵 이름 |
| **Return** | x |
| **Function** | 지정한 비트맵의 총 비트 개수를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_test\_by\_name(const char \*name, size\_t idx) |
| **Parameter** | name : 대상 비트맵 이름, idx : 검사할 비트 위치 |
| **Return** | x |
| **Function** | 해당 위치의 비트가 true인지 여부를 출력한다. |

1. **List (list.c, list.h에서 수정한 내용)**

* list.h에 list\_item 구조체 선언하였다. (elem : 리스트 연결을 위한 포인터 구조체, data : 리스트에 저장될 실제 정수 값)
* list.c에 #include <time.h> 추가 : list\_shuffle 구현을 위해 필요하다.
* list.c에 #include <stdlib.h> 추가 : list\_shuffle 구현을 위해 필요하다.

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | a : 위치를 바꿀 첫 번째 리스트 요소 포인터  b : 위치를 바꿀 두 번째 리스트 요소 포인터 |
| **Return** | x |
| **Function** | 두 리스트 요소의 위치를 서로 교환한다. 인접 또는 비인접한 경우 모두 처리 가능하다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | list : 섞을 대상 리스트 포인터 |
| **Return** | x |
| **Function** | Fisher–Yates 알고리즘을 사용하여 리스트 요소들의 순서를 무작위로 섞는다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_nth(struct list \*lst, size\_t idx) |
| **Parameter** | lst : 리스트 포인터  idx : 찾을 요소의 인덱스 (0부터 시작) |
| **Return** | 해당 인덱스의 리스트 요소 포인터 또는 NULL |
| **Function** | 지정한 인덱스에 위치한 리스트 요소를 반환한다. |

1. **Hash Table (hash.c, hash.h에서 수정한 내용)**

|  |  |
| --- | --- |
| **Prototype** | unsigned int hash\_int\_2(int i) |
| **Parameter** | i : 해시할 정수 값 |
| **Return** | Knuth의 곱셈 해시 결과 (unsigned int) |
| **Function** | Knuth의 곱셈 해시 알고리즘을 이용해 정수 값을 해시값으로 변환한다. 충돌을 줄이기 위해 2654435761을 곱한다. |

* Knuth 알고리즘

이 방식은 정수 값에 고정된 소수 상수 265443571을 곱해 해쉬 값을 생성한다. 황금비율의 역수인 2의 32제곱을 확장한 값을 기반으로 균등한 해쉬 분포를 만든다.

1. **Bitmap (bitmap.c, bitmap.h에서 수정한 내용)**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap \*bm, size\_t additional\_bits) |
| **Parameter** | bm : 기존 비트맵 포인터  additional\_bits : 확장할 비트 수 |
| **Return** | 새로 확장된 비트맵 포인터 또는 NULL |
| **Function** | 기존 비트맵을 지정한 비트 수만큼 확장한 새로운 비트맵을 생성하고, 기존 내용을 복사한 뒤 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void\* bitmap\_get\_raw(struct bitmap \*bm) |
| **Parameter** | bm : 비트맵 포인터 |
| **Return** | bm 내부의 raw 비트 배열 포인터 (void\*) |
| **Function** | bitmap의 내부 비트 배열에 직접 접근할 수 있도록 포인터를 반환한다. bitmap\_hex\_dump\_by\_name 함수에서 사용된다. |