**Project #1 : MyLib**

**.**

|  |  |
| --- | --- |
| 담당 교수 : | 이영민 |
| 학번 : | 20231552 |
| 이름 : | 박지민 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | struct list\_item  {  struct list\_elem elem;  struct hash\_elem hash\_elem;  int data;  }; |
| **Function** | 리스트와 해시 테이블 모두에서 사용할 수 있도록 list\_elem, hash\_elem을 포함한 데이터 구조  실제 데이터는 int data 필드에 저장됨 |

|  |  |
| --- | --- |
| **Prototype** | #define MAX\_OBJECTS 10  #define MAX\_ARGC 10  #define CMD\_LEN 100 |
| **Function** | MAX\_OBJECTS: 생성 가능한 list/hash/bitmap 객체의 최대 수  MAX\_ARGC: 명령어의 최대 인자 개수  CMD\_LEN: 한 줄 명령어 입력의 최대 길이 |

|  |  |
| --- | --- |
| **Prototype** | struct list lists[MAX\_OBJECTS];  struct hash hashtables[MAX\_OBJECTS];  struct bitmap \*bitmaps[MAX\_OBJECTS];  bool list\_created[MAX\_OBJECTS] = { false };  bool hash\_created[MAX\_OBJECTS] = { false };  bool bm\_created[MAX\_OBJECTS] = { false }; |
| **Function** | 각각 리스트, 해시, 비트맵 객체를 저장하기 위한 전역 배열  각 인덱스에 해당하는 객체의 생성 여부를 bool 배열로 관리함  main() 함수에서 명령어 처리 시 중복 생성을 방지하고, 유효성 검증에 사용됨 |

|  |  |
| --- | --- |
| **Prototype** | enum obj\_type  {  OBJ\_LIST,  OBJ\_HASH,  OBJ\_BM  }; |
| **Function** | 객체 타입 구분용 |

|  |  |
| --- | --- |
| **Prototype** | bool less\_func(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux); |
| **Parameter** | a, b: 비교할 리스트 요소, aux: 사용하지 않음 |
| **Return** | a->data < b->data이면 true, 아니면 false |
| **Function** | list\_sort, list\_insert\_ordered, list\_max, list\_min 등에서 정렬 기준으로 사용됨 |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func(const struct hash\_elem \*e, void \*aux); |
| **Parameter** | e : 해시 요소, aux : 사용하지 않음 |
| **Return** | item->data 값에 대한 해시값 |
| **Function** | 정수를 해시 테이블의 인덱스로 매핑하기 위한 함수, hash\_init() 시 해시 함수로 전달됨 |

|  |  |
| --- | --- |
| **Prototype** | bool main\_hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux); |
| **Parameter** | a, b: 비교할 해시 요소, aux: 사용하지 않음 |
| **Return** | a->ldata < b->data이면 true |
| **Function** | 해시 테이블의 삽입/탐색 시 비교 기준으로 사용됨 |

|  |  |
| --- | --- |
| **Prototype** | void triple\_apply(struct hash\_elem \*e, void \*aux); |
| **Parameter** | e: 연산 대상 해시 요소, aux: 사용하지 않음 |
| **Return** | 없음 |
| **Function** | 해시 요소의 값을 세제곱으로 만든다 |

|  |  |
| --- | --- |
| **Prototype** | void square\_apply(struct hash\_elem \*e, void \*aux); |
| **Parameter** | e: 연산 대상 해시 요소, aux : 사용하지 않음 |
| **Return** | 없음 |
| **Function** | 해시 요소의 값을 제곱한다 |

|  |  |
| --- | --- |
| **Prototype** | bool is\_str\_true(const char \*s); |
| **Parameter** | s: 문자열 포인터 |
| **Return** | 문자열이 "true"일 경우 true, 그 외는 false |
| **Function** | 문자열을 Boolean 값으로 변환해준다 |

|  |  |
| --- | --- |
| **Prototype** | void elem\_destructor(struct hash\_elem \*e, void \*aux); |
| **Parameter** | e : 해제할 요소 포인터, aux : 사용하지 않음 |
| **Return** | 없음 |
| **Function** | 해시 요소 메모리 해제 함수( hash\_destory, hash\_clear 등에 사용) |

|  |  |
| --- | --- |
| **Prototype** | int parse\_object\_index(const char \*name, enum obj\_type \*type); |
| **Parameter** | Name : list0, hash1, bm2 형태의 문자열, type :해당 객체 타입을 반환받을 포인터 |
| **Return** | 객체 인덱스, 유효하지 않은 인덱스라면 -1 |
| **Function** | 문자열에서 객체 타입과 인덱스를 추출하는 유틸리티 함수 |

|  |  |
| --- | --- |
| **Prototype** | void dump\_list(int idx); |
| **Parameter** | Idx : 출력할 리스트 인덱스 |
| **Return** | 없음 |
| **Function** | 리스트 내부의 모든 요소를 순서대로 출력 |

|  |  |
| --- | --- |
| **Prototype** | void dump\_hash(int idx); |
| **Parameter** | Idx : 출력할 해시 테이블 인덱스 |
| **Return** | 없음 |
| **Function** | 해시 테이블의 모든 요소를 출력 |

|  |  |
| --- | --- |
| **Prototype** | void dump\_bm(int idx); |
| **Parameter** | Idx : 출력할 비트맵 인덱스 |
| **Return** | 없음 |
| **Function** | 비트맵의 모든 비트 출력 |

|  |  |
| --- | --- |
| **Prototype** | Main 함수 |
| **Description** | 1. 명령어 입력 받기 : stidin으로부터 fgets 함수를 사용해 한 줄 단위로 입력을 받는다. 이때, 입력된 문자열에서 ‘\n을 제거한다.  2. 입력 파싱 : strtok 함수를 사용해 공백 기준으로 명령어 분해, 분해된 각 명령어는 argv[] 배열에 저장되고 전체 명령어 개수는 argc에 저장  3. 명령어 분기 처리  argv[0]을 기준으로 명령어의 종류를 판별하여 분기 처리  4. 객체 이름 파싱 및 자료구조 식별 : list3과 같은 객체 이름에서 parse\_object\_index 함수를 호출하여 인덱스를 분리한다. 타입(enum)과 인덱스를 기준으로 각 자료구조 배열에 접근  6. 인자 해석 및 연산 수행 : 파싱된 인자들을 정수 또는 Boolean 값으로 atoi, is\_str\_true 등으로 해석, 해석한 명령에 맞는 함수를 호출한다.  7. 명령어가 quit일 때 루프를 빠져나와 프로그램 종료 |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list); |
| **Parameter** | list : 섞을 리스트 구조체 포인터 |
| **Return** | 없음 |
| **Function** | 리스트에 포함된 요소들을 무작위로 섞는다. 리스트의 모든 요소를 struct list\_elem\* 배열에 복사하면서 리스트에서 제거한다  Fisher-Yates 알고리즘을 적용하여 배열 내 요소들의 순서를 무작위로 섞음  셔플된 순서대로 리스트에 다시 삽입한다 |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b); |
| **Parameter** | a, b : 서로 자리를 바꿀 리스트 요소 |
| **Return** | 없음 |
| **Function** | 리스트 내의 두 요소의 위치를 서로 교환한다. 인접한 요소와 비인접한 요소의 경우를 구분하여 안전하게 링크를 재구성함. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i); |
| **Parameter** | i : 해시할 정수 |
| **Return** | 해시값 |
| **Function** | Hash\_int 에서 해시 분포가 더 고르게 되도록 보완한 함수  0x55555555과 xor 마스킹을 적용하여 낮은 비트들만을 사용하는 해시 충돌을 줄임. |

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*)((uint8\_t \*)(HASH\_ELEM) - offsetof(STRUCT, MEMBER))) |
| **Parameter** | * HASH\_ELEM: struct hash\_elem\* 포인터 * STRUCT: 이 포인터가 포함된 구조체의 타입 * MEMBER: 구조체 내 hash\_elem의 멤버 이름 |
| **Return** | struct hash\_elem이 포함된 원래 구조체의 포인터 |
| **Function** | struct hash\_elem 포인터를 원래 구조체로 되돌리기 위한 역변환 도구  list\_entry와 같은 방식으로 동작하며, offsetof 매크로를 이용해 구조체 기준 주소를 계산 |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size); |
| **Parameter** | Bitmap : 기존 비트맵 포인터, size : 확장할 비트 수 |
| **Return** | 확장된 새 비트맵 포인터 (실패 시 NULL) |
| **Function** | 기존 비트맵을 size만큼 확장하여 새로운 비트맵을 만들고 기존 값을 복사하여 새로운 공간을 초기화함  기존 메모리는 해제되며 새로운 포인터가 반환됨 |