**Project #1 : MyLib**

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 |
| 학번 : | 20210428 |
| 이름 : | 정현정 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size) |
| **Parameter** | struct bitmap \*bitmap, int size |
| **Return** | Parameter로 들어온 bitmap |
| **Function** | 해당 함수는 인자로 들어온 bitmap의 오른쪽에 인자로 들어온 size만큼의 false로 set된 비트를 덧붙이는 기능을 수행한다. 우선 인자로 받은 bitmap의 현재 bit\_cnt를 size\_t 타입의 새로운 변수 size\_before에 저장한다. 이후 bitmap의 현재 bit\_cnt에 size 크기 만큼 더해서 비트의 개수를 늘려준다. 그 다음 for문과 이전에 저장한 size\_before의 값, bitmap\_set 함수를 사용하여서 뒤에 덧붙여진 비트가 false로 set되도록 한다. 여기서 bitmap\_set 함수는 인자로 받은 인덱스의 값을 인자로 받은 value로 지정하는 역할을 수행하므로 다음과 같이 for문을 작성할 수 있다.  for(int i = 0; i < size; i++){      bitmap\_set(bitmap, size\_before + i, false);    } |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | int i |
| **Return** | unsigned type의 hash int |
| **Function** | hash\_int\_2 함수는 인자로 들어온 int i의 값에 대해 일정한 규칙을 적용시켜 해시 테이블의 인덱스로 변환하는 함수이다. 이러한 변환을 통해 해시 테이블에 접근이 가능하다. 현재는 해시 함수 반환값을 도출하기 위해 i % 29라는 규칙을 적용시키고 있다. 이는 반환값이 0~28 사이임을 의미한다. 해시 함수에서 modulo 연산을 적용시키는 방법의 경우, 소수로 연산을 수행하면 조금 더 고르게 해시 테이블에 분배할 수 있다. 본 함수의 경우 소수 중 하나인 29를 선택하였다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | struct list\_elem \*a, struct list\_elem \*b |
| **Return** | void |
| **Function** | list\_swap 함수는 인자로 받아온 두 list\_elem 포인터를 사용하여서 리스트 내 두 원소의 위치를 바꾸는 역할을 수행한다. 이때 a와 b가 인접하게 있는 경우와 그렇지 않은 경우로 나누어서 함수를 작성하였다. 이는 조작해야 하는 포인터의 수가 a, b가 인접하지 않을 때 더 많기 때문이다. swap으로 인한 list의 head와 tail의 업데이트는 main.c에서 수행하였다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | struct list\* list |
| **Return** | void |
| **Function** | 해당 함수는 인자로 들어오는 list 포인터를 이용하여서 리스트를 랜덤하게 섞는 기능을 수행한다. 우선 현재 list의 size를 list\_size 함수를 사용하여서 구한 후, 해당 size가 2보다 작다면 섞을 필요가 없으므로 return 하도록 하였다. 이후 list 내부의 list\_elem 각각을 가리키는 포인터들의 배열을 동적할당하여 모든 요소를 복사하였다. 그 다음 랜덤하게 두개의 숫자를 선택하고 복사한 배열 상에서 두 숫자가 가리키는 인덱스 위치의 요소를 swap하도록 구현하였다. 랜덤하게 두개의 인덱스를 뽑아 swap하는 동작을 30번 반복한다. 마지막으로 랜덤하게 섞인 배열을 다시 이중 연결리스트로 복구한 다음, 동적 할당한 배열을 메모리에서 해제하였다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \* aux |
| **Return** | void |
| **Function** | 해당 함수는 인자로 받은 리스트에 중복인 요소가 있는 경우, 중복된 요소들 중 첫번째 요소만 남기고 나머지 요소들을 제거한다. 이때 만약 duplicates 인자의 자리에 NULL이 아닌 다른 리스트 포인터가 주어져 있다면, 추가로 주어진 리스트에 제거할 요소들을 담는다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | 해당 함수는 비어있는 리스트를 만들어준다. 이때 list 구조체에 있는 head와 tail도 모두 세팅해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* |
| **Function** | 이 함수는 리스트의 첫번째 요소, 즉 head의 next에 저장되어있는 list\_elem의 포인터를 반환한다. 주로 반복문을 작성할 때 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* |
| **Function** | 리스트의 마지막 요소를 가리키는 포인터를 반환한다. 즉, list의 tail에 저장되어 있는 주소를 반환한다. 주로 반복문의 종료 조건에 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem\* |
| **Function** | 이중 연결리스트에서 현재 인자로 들어온 elem의 next에 위치해 있는 elem을 반환한다. 주로 반복문을 작성할 때 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | struct list \*list, list\_less\_fun \*less, void \*aux |
| **Return** | void |
| **Function** | 이 함수는 인자로 받은 list에 대해 less 함수를 적용시켜 정렬이 되어 있지 않은 원소를 찾는다. 이후 inplace\_marge 함수를 사용하여서 해당 위치에서 정렬된 list를 갖도록 조작한다. 이를 list의 첫 원소부터 마지막 원소까지 반복문으로 반복하면서 모든 원소들이 정렬될 수 있도록 한다. |

|  |  |
| --- | --- |
| **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** | void |
| **Function** | 한 list에서 인자로 받은 first~last 사이의 원소들을 모두 제거하고 이를 인자로 받은 before 앞에 붙이는 기능을 하는 함수이다. 포인터 조작을 통해 이루어지고 있다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*elem |
| **Return** | void |
| **Function** | 인자로 받은 before 앞에 인자로 받은 새로운 elem을 삽입한다. 포인터 조작을 통해 이루어지고 있으며 list\_push\_front, list\_push\_back, list\_insert\_ordered 함수에서 내부적으로 사용되고 있다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | bool |
| **Function** | 인자로 받은 list가 비어있다면 true를, 비어있지 않다면 false를 리턴하는 함수이다. while문의 조건문 혹은 dumpdata를 하는데에 사용된다. dumpdata의 경우, list가 비어있지 않다면 리스트에 저장되어 있는 모든 원소들을 print하도록 main.c에 구현되어 있다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | void |
| **Function** | list\_insert 함수를 사용하여서 리스트의 맨 뒤에 인자로 받은 새로운 elem을 삽입한다. 이때 내부적으로 list\_insert의 첫번째 인자로 list\_end 함수를 사용한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem\* |
| **Function** | 인자로 들어온 list의 맨 처음 원소를 제거한 뒤 이를 반환한다. 내부적으로 list\_remove를 사용하여 구현하였다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem \* |
| **Function** | 인자로 받은 원소를 제거하는 기능을 수행한다. 메모리를 해제시키는 것이 아니라 포인터 조작을 통해 해당 리스트에서 원소가 사라진 것처럼 보이도록 한다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux |
| **Return** | bool |
| **Function** | 이 함수는 인자로 받은 해시 테이블을 초기화 하는 기능을 수행한다. hash 구조체에 대해서 elem\_cnt를 0으로 설정하고 초기 bucket\_cnt를 4로 설정하며 버켓의 메모리를 동적으로 할당한다. 하지만 각각의 버켓 안에서 원소들을 동적 할당하는 코드는 존재하지 않는다. 만약 버켓의 메모리가 정상적으로 할당이 되었다면 내부적으로 hash\_clear 함수를 실행시킨다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | 각각의 버켓마다 빈 이중 연결리스트를 생성하는 코드이다. 버켓이 비어있지 않은 상태라면 각각의 버켓 안에 들어있는 원소들을 모두 비운다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | 내부적으로 hash\_clear 함수를 사용하여서 모든 버켓을 비운 다음 마지막으로 동적 할당된 공간까지 해제하여 해시 테이블을 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*find\_bucket (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | static struct list\* |
| **Function** | 인자로 받은 해시 테이블에서 인자로 받은 hash\_elem인 e가 들어있는 버켓을 반환한다. 여기서 버켓의 인덱스는 구조체에 저장되어 있는 hash 함수를 사용하여 해시 값을 계산한 후 실제 인덱스 범위인 bucket\_cnt-1과 and 연산을 수행하여 구할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct list \*bucket, struct hash\_elem \*e |
| **Return** | static struct hash\_elem\* |
| **Function** | 인자로 주어진 버켓에서 elem을 찾아서 반환한다. 해당 hash\_elem이 있다면 포인터를 리턴하고 없다면 null 포인터를 리턴한다. 즉, 인자로 복제품을 주고 복제품의 data를 사용하여서 data와 같은 값을 가진 원소가 해시 테이블 내에 존재하는지 확인하는 방식이라고 할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | struct hash\_elem\* |
| **Function** | 우선 내부적으로 find\_bucket과 find\_elem 함수를 통해 해당 원소가 이미 해시 테이블에 존재하는지 확인한다. 만약 존재한다면 rehash 함수를 바로 실행시키고 아니라면 새로운 원소이므로 해시 테이블에 새롭게 삽입한 후 rehash 함수를 실행시킨다. 이후 해당 elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct list \*bucket, struct hash\_elem \*e |
| **Return** | void |
| **Function** | 이 함수는 list\_push\_front 함수를 내부적으로 사용하여서 인자로 받은 hash\_elem의 list\_elem을 버켓에 추가한다. list\_push\_front 함수를 사용하기 때문에 해당 버켓의 맨 앞에 삽입된다. 또한 현재 해시 테이블의 elem\_cnt 개수를 하나 증가시키는 작업도 포함되어 있다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h) |
| **Parameter** | struct hash \*h |
| **Return** | void |
| **Function** | 이 함수는 hash\_insert 함수 내부에서 사용된다. hash\_insert에서 insert\_elem 함수를 사용하여서 새로운 원소를 현재의 해시 테이블에 삽입한다. 새로운 원소가 삽입되었으므로 원소들을 재배열하면 더 고르게 분포되고 더 좋은 해시 테이블이 만들어질 수 있다. rehash 함수는 이러한 재분배의 기능을 수행하는 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | hash\_elem\* |
| **Function** | 본 함수는 인자로 전달받은 해시 테이블 h에서 인자로 받은 new hash\_elem과 동일한 원소가 있다면 그것을 지우고 새롭게 다시 삽입하는 역할을 수행한다. 만약 대체가 아니라 새롭게 삽입한 것이라면 null을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | hash\_elem\* |
| **Function** | 인자로 주어진 해시 테이블에서 hash\_elem e를 찾아서 리턴한다. 내부적으로 find\_elem 함수를 사용한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | struct hash\_elem\* |
| **Function** | 인자로 받은 hash\_elem을 해시 테이블 h에서 삭제하고 삭제한 원소를 반환하는 함수이다. remove\_elem 함수로 원소를 삭제한 후 rehash 함수를 통해 해시 테이블의 원소들을 재분배한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | struct hash \*h, hash\_action\_func \*action |
| **Return** | void |
| **Function** | 해시 테이블에 있는 모든 원소들에 대해서 인자로 받은 action 함수를 수행한다. 이때 action 인자는 함수 포인터이다. 반복문을 사용하기 위해 list\_begin, list\_end, list\_next 함수를 사용하고 있다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx) |
| **Parameter** | size\_t bit\_idx) |
| **Return** | elem\_type |
| **Function** | 인자로 받은 인덱스 위치의 비트만 1로 설정하고 나머지는 다 0으로 설정한 unsigned long을 반환한다. 이 함수를 통해 생성된 unsigned long과 and 연산을 통해 bitmap\_test 함수를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | const struct bitmap \*b |
| **Return** | size\_t |
| **Function** | 현재 인자로 받은 bitmap 구조체의 bit\_cnt를 반환한다. main.c에서 dumpdata 기능을 수행할 때 반복문에 사용되었다. 구조체가 bitmap.c에 선언되어 있기 때문에 해당 함수를 통해 구조체의 bit\_cnt 필드에 접근해야 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | size\_t bit\_cnt |
| **Return** | struct bitmap\* |
| **Function** | 인자로 들어온 bit\_cnt만큼 bitmap의 bits 필드에 동적할당 한다. 그리고 모든 비트들은 0으로 설정하고 있다. 즉, bit\_cnt만큼의 0으로 기본적으로 설정된 비트맵을 생성하는 것이다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | struct bitmap \*b |
| **Return** | void |
| **Function** | 비트맵의 bits 필드에 동적할당 된 메모리를 해제하고 있다. 현재 코드에 따르면 bitmap\_create에서 동적할당이 이루어지고 있다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | struct bitmap \*b, size\_t idx, bool value |
| **Return** | void |
| **Function** | 인자로 들어온 인덱스의 비트를 value 값으로 설정한다. 이 함수는 bitmap\_mark와 bitmap\_reset 함수에서 내부적으로 사용되고 있다. value값이 1이면 bitmap\_mark가 실행되는 것과 같고 value값이 0이면 bitmap\_reset이 실행되는 것과 같다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | void |
| **Function** | 해당 인덱스의 비트를 토글하는 함수이다. 만약 해당 인덱스에 저장된 비트의 값이 false이면 true로, true이면 false로 바꿔서 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*b, size\_t idx) |
| **Parameter** | const struct bitmap \*b, size\_t idx |
| **Return** | bool |
| **Function** | 인자로 받은 인덱스의 비트가 true인지 false인지 리턴한다. bit\_mask 함수를 사용하여 해당 인덱스만 1로 표시된 비트맵을 가지고 and 연산을 통해 값을 도출한다. bitmap 구조체가 bitmap.c에 선언되어 있으므로 bits 필드에 접근하기 위해서 해당 함수를 사용한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | void |
| **Function** | 인자로 받은 bitmap에 대하여 인덱스가 start인 지점부터 시작하여서 cnt 개수만큼 value 값으로 설정하는 함수이다. bitmap\_set\_all과 bitmap\_scan\_and\_flip에서 내부적으로 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | 인자로 들어온 bitmap에 대해서 인덱스가 start인 것부터 시작하여서 cnt 개수만큼의 범위를 보았을 때, value값으로 설정되어 있는 비트의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | bool |
| **Function** | 인자로 들어온 bitmap에 대하여 인덱스가 start인 것부터 시작하여서 cnt 개수만큼의 범위를 보았을 때, 해당 범위 내에서 하나라도 value값으로 set되어 있는지를 반환하는 함수이다. bitmap\_any, bitmap\_none, bitmap\_scan에서 내부적으로 사용한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | 인자로 받은 bitmap에 대하여 인덱스가 start인 지점에서 시작하여 cnt개수 만큼의 범위를 보았을 때, 해당 범위 내에서 모두 value값으로 set되어 있다면 최초로 시작되는 인덱스를 반환한다. 만약 없다면 BITMAP\_ERROR를 반환한다. |