**Project #1 : MyLib**

|  |  |
| --- | --- |
| 담당 교수 : | 김영재 교수님 |
| 학번 : | 20221197 |
| 이름 : | 백서연 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | int extract\_index\_from\_name(const char \*obj\_name) |
| **Parameter** | obj\_name: 인덱스가 포함된 객체 이름(예: "list1", "hash2" 등) |
| **Return** | int: 추출한 정수 인덱스, 없으면 -1 |
| **Function** | 문자열에서 숫자(인덱스) 부분을 찾아 정수로 변환하여 반환합니다. 숫자가 없는 경우 -1을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | int split\_line(char \*inputBuffer, char \*\*token\_arr) |
| **Parameter** | inputBuffer: 사용자 입력 문자열  token\_arr: 토큰을 저장할 배열 |
| **Return** | int: 파싱된 토큰의 개수 |
| **Function** | 입력 문자열을 공백, 탭, 개행 문자 등을 기준으로 분리하여 token\_arr에 저장하고, 분리된 토큰의 개수를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | bool compare\_list\_elements(const struct list\_elem \*node\_a, const struct list\_elem \*node\_b, void \*aux\_data) |
| **Parameter** | node\_a, node\_b: 비교 대상 리스트 요소  aux\_data: 비교 시 필요한 추가 데이터(사용되지 않음) |
| **Return** | bool: node\_a->value < node\_b->value이면 true |
| **Function** | 두 리스트 노드의 value 값을 비교하여 정렬 등에 사용될 기준을 제공합니다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* get\_nth\_element(struct list \*lst, int index) |
| **Parameter** | lst: 조회할 리스트  index: 0부터 시작하는 n번째 요소 인덱스 |
| **Return** | struct list\_elem\*: n번째 리스트 요소의 포인터, 범위를 벗어나면 NULL |
| **Function** | 리스트에서 index번째 요소를 찾아 반환합니다. 범위를 벗어나면 NULL을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | void free\_list(struct list \*lst) |
| **Parameter** | lst: 해제할 리스트 |
| **Return** | void |
| **Function** | 리스트 내 모든 노드를 제거하고, 동적으로 할당된 메모리를 해제합니다. |

|  |  |
| --- | --- |
| **Prototype** | void randomize\_list(struct list \*lst) |
| **Parameter** | lst: 무작위로 섞을 대상 리스트 |
| **Return** | void |
| **Function** | Fisher-Yates 알고리즘을 사용하여 리스트 요소들의 순서를 무작위로 섞습니다. |

|  |  |
| --- | --- |
| **Prototype** | void swap\_list\_elements(struct list\_elem \*node1, struct list\_elem \*node2) |
| **Parameter** | node1, node2: 스왑할 리스트 요소 |
| **Return** | void |
| **Function** | 두 리스트 요소의 연결 상태는 유지하고, 내부에 저장된 value만 교환합니다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned compute\_hash(const struct hash\_elem \*hash\_elem\_ptr, void \*aux\_data) |
| **Parameter** | hash\_elem\_ptr: 해시 요소(내부 구조체의 포인터)  aux\_data: 해시 계산 시 필요한 추가 데이터(사용되지 않음) |
| **Return** | unsigned: 해싱 결과값 |
| **Function** | 해시 요소(struct hash\_node)의 num\_value를 hash\_int()로 해싱하여 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | bool compare\_hash\_elements(const struct hash\_elem \*hash\_elem1, const struct hash\_elem \*hash\_elem2, void \*aux\_data) |
| **Parameter** | hash\_elem1, hash\_elem2: 비교 대상 해시 요소  aux\_data: 비교 시 필요한 추가 데이터(사용되지 않음) |
| **Return** | bool: 첫 번째가 더 작으면 true |
| **Function** | 두 해시 노드의 num\_value 값을 비교하여 정렬 등에 사용될 기준을 제공합니다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned alternate\_hash\_int(int num) |
| **Parameter** | num: 해싱할 정수 |
| **Return** | unsigned: 해싱 결과값 |
| **Function** | 곱셈과 XOR 연산을 활용해 정수를 해싱한 결과값을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* expand\_bitmap(struct bitmap \*bmp, int new\_capacity) |
| **Parameter** | bmp: 확장할 기존 비트맵  new\_capacity: 새 비트맵의 용량(비트 수) |
| **Return** | struct bitmap\*: 확장된 비트맵의 포인터 |
| **Function** | 기존 비트맵의 데이터를 새 비트맵에 복사하고, 이전 비트맵을 해제한 뒤 새 비트맵을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | void init\_hash\_table(const char \*table\_name) |
| **Parameter** | table\_name: 생성할 해시 테이블의 이름(예: "hash0") |
| **Return** | void |
| **Function** | 전역 hash\_arr[]에서 이름에 해당하는 인덱스를 추출하여 해시 테이블을 동적 할당 후 hash\_init()으로 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | void init\_bitmap(const char \*bitmap\_name, size\_t bit\_count) |
| **Parameter** | bitmap\_name: 생성할 비트맵의 이름(예: "bitmap1")  bit\_count: 비트 수 |
| **Return** | void |
| **Function** | 전역 bmp\_arr[]에서 이름에 해당하는 인덱스를 추출하여 bit\_count 크기의 비트맵을 생성합니다. |

|  |  |
| --- | --- |
| **Prototype** | void init\_list(const char \*list\_name) |
| **Parameter** | list\_name: 생성할 리스트의 이름(예: "list0") |
| **Return** | void |
| **Function** | 전역 list\_arr[]에서 이름에 해당하는 인덱스를 추출하여 리스트를 동적 할당 후 list\_init()으로 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | void reset\_bitmap\_array() |
| **Parameter** | 없음 |
| **Return** | void |
| **Function** | 전역 bmp\_arr[] 배열의 모든 요소를 NULL로 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | void reset\_list\_array() |
| **Parameter** | 없음 |
| **Return** | void |
| **Function** | 전역 list\_arr[] 배열의 모든 요소를 NULL로 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | void print\_bitmap\_binary(const struct bitmap \*bmp) |
| **Parameter** | bmp: 출력할 비트맵 |
| **Return** | void |
| **Function** | 비트맵의 각 비트를 0 또는 1로 출력합니다. |

|  |  |
| --- | --- |
| **Prototype** | static void print\_hash\_element(struct hash\_elem \*hashElem, void \*aux\_data) |
| **Parameter** | hashElem: 해시 테이블의 요소  aux\_data: 추가 데이터(사용되지 않음) |
| **Return** | void |
| **Function** | 해시 테이블 요소(struct hash\_node)의 num\_value를 출력합니다. (내부용 static 함수) |

|  |  |
| --- | --- |
| **Prototype** | void print\_hash\_table(const struct hash \*hashTbl) |
| **Parameter** | hashTbl: 출력할 해시 테이블 |
| **Return** | void |
| **Function** | 해시 테이블의 모든 요소를 순회하며 값을 출력합니다. |

|  |  |
| --- | --- |
| **Prototype** | void square\_element(struct hash\_elem \*hashElem, void \*aux\_data) |
| **Parameter** | hashElem: 해시 테이블의 요소  aux\_data: 추가 데이터(사용되지 않음) |
| **Return** | void |
| **Function** | 해시 테이블 요소의 num\_value를 제곱으로 변경합니다. (hash\_apply 등에 사용) |

|  |  |
| --- | --- |
| **Prototype** | void cube\_element(struct hash\_elem \*hashElem, void \*aux\_data) |
| **Parameter** | hashElem: 해시 테이블의 요소  aux\_data: 추가 데이터(사용되지 않음) |
| **Return** | void |
| **Function** | 해시 테이블 요소의 num\_value를 세제곱으로 변경합니다. (hash\_apply 등에 사용) |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* find\_element\_by\_value(struct hash \*hashTbl, int search\_value) |
| **Parameter** | hashTbl: 검색할 해시 테이블  search\_value: 찾고자 하는 값 |
| **Return** | struct hash\_elem\*: 찾은 해시 요소의 포인터, 없으면 NULL |
| **Function** | 해시 테이블에서 search\_value와 동일한 값을 가진 요소를 찾아 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | void insert\_element\_at(struct list \*lst, int position, int value) |
| **Parameter** | lst: 요소를 삽입할 리스트  position: 삽입할 위치(0부터 시작)  value: 삽입할 정수 값 |
| **Return** | void |
| **Function** | 리스트의 position 위치에 새 노드를 동적 할당 후 삽입합니다. 인덱스 범위를 벗어나면 리스트 끝에 삽입합니다. |

|  |  |
| --- | --- |
| **Prototype** | void process\_create\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "create" 명령어를 처리하여 list, hashtable, bitmap 등을 생성합니다. (예: create list list0, create hashtable hash1, create bitmap bitmap2 16 등) |

|  |  |
| --- | --- |
| **Prototype** | void process\_delete\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "delete" 명령어를 처리하여 해당 인덱스의 리스트, 해시 테이블, 비트맵을 찾아 메모리를 해제하고 NULL로 만듭니다. |

|  |  |
| --- | --- |
| **Prototype** | void process\_dumpdata\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "dumpdata" 명령어를 처리하여 해당 인덱스 자료구조의 내용을 출력합니다. (리스트, 해시 테이블, 비트맵 등) |

|  |  |
| --- | --- |
| **Prototype** | void process\_hash\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "hash\_\*"로 시작하는 해시 테이블 관련 명령을 처리합니다. 예: hash\_insert, hash\_delete, hash\_apply, hash\_size, hash\_empty 등 |

|  |  |
| --- | --- |
| **Prototype** | void process\_list\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "list\_\*"로 시작하는 리스트 관련 명령을 처리합니다. 예: list\_push\_front, list\_pop\_back, list\_sort, list\_splice 등 |

|  |  |
| --- | --- |
| **Prototype** | void process\_bitmap\_command(char \*\*cmd\_tokens, int token\_count) |
| **Parameter** | cmd\_tokens: 파싱된 명령어 토큰 배열  token\_count: 토큰 개수 |
| **Return** | void |
| **Function** | "bitmap\_\*"로 시작하는 비트맵 관련 명령을 처리합니다. 예: bitmap\_set, bitmap\_scan, bitmap\_expand, bitmap\_test 등 |

|  |  |
| --- | --- |
| **Prototype** | int main(void) |
| **Parameter** | 없음 |
| **Return** | int: 프로그램 종료 시 반환값 |
| **Function** | 전역 배열 초기화 후, 입력된 명령어를 무한 루프에서 파싱하여 해당 명령어 처리 함수를 호출합니다. "quit" 명령어가 입력되면 루프를 빠져나와 종료합니다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | **void list\_init(struct list \*list)** |
| **Parameter** | **list: 초기화할 리스트 구조체의 포인터** |
| **Return** | 없음 |
| **Function** | 리스트를 빈 상태로 초기화합니다. 헤드의 next는 테일을, 테일의 prev는 헤드를 가리키도록 설정합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_insert(struct list\_elem \*before, struct list\_elem \*elem)** |
| **Parameter** | **before: 삽입할 위치로, 이 요소 앞에 새 요소가 들어갑니다. (내부 요소나 테일 가능)**  **elem: 삽입할 새 리스트 요소의 포인터** |
| **Return** | 없음 |
| **Function** | 새 요소를 지정한 위치 앞에 삽입하여 리스트의 연결을 재구성합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_splice(struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last)** |
| **Parameter** | **before: 삽입할 위치로, 이 요소 앞에 서브리스트가 들어갑니다.**  **first: 이동할 서브리스트의 시작 요소의 포인터**  **last: 이동할 서브리스트의 마지막 요소의 바로 다음 요소의 포인터 (exclusive)** |
| **Return** | 없음 |
| **Function** | 지정된 범위의 요소들(first부터 last 전까지)을 기존 리스트에서 제거한 후, 지정된 위치(before) 앞에 삽입합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_push\_front(struct list \*list, struct list\_elem \*elem)** |
| **Parameter** | **list: 대상 리스트의 포인터**  **elem: 삽입할 새 요소의 포인터** |
| **Return** | 없음 |
| **Function** | 리스트의 맨 앞(헤드 뒤)에 새 요소를 삽입합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_push\_back(struct list \*list, struct list\_elem \*elem)** |
| **Parameter** | **list: 대상 리스트의 포인터**  **elem: 삽입할 새 요소의 포인터** |
| **Return** | 없음 |
| **Function** | 리스트의 맨 뒤(테일 앞)에 새 요소를 삽입합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_remove(struct list\_elem \*elem)** |
| **Parameter** | **elem: 제거할 리스트 요소의 포인터 (내부 요소여야 함)** |
| **Return** | 제거된 요소의 다음 요소의 포인터 |
| **Function** | 지정된 요소를 리스트에서 제거하고, 제거된 요소의 뒤를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_pop\_front(struct list \*list)** |
| **Parameter** | **list: 대상 리스트의 포인터** |
| **Return** | 제거된 첫 번째 요소의 포인터 |
| **Function** | 리스트의 첫 번째 요소를 제거하고 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_pop\_back(struct list \*list)** |
| **Parameter** | list: 대상 리스트의 포인터 |
| **Return** | 제거된 마지막 요소의 포인터 |
| **Function** | 리스트의 마지막 요소를 제거하고 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_front(struct list \*list)** |
| **Parameter** | list: 대상 리스트의 포인터 |
| **Return** | 리스트의 첫 번째 실제 요소의 포인터 |
| **Function** | 리스트가 비어있지 않다면 첫 번째 요소를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_back(struct list \*list)** |
| **Parameter** | 대상 리스트의 포인터 |
| **Return** | 리스트의 마지막 실제 요소의 포인터 |
| **Function** | 리스트가 비어있지 않다면 마지막 요소를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_reverse(struct list \*list)** |
| **Parameter** | list: 대상 리스트의 포인터 |
| **Return** | 반환값: 없음 (**void**) |
| **Function** | 리스트의 요소 순서를 뒤집어, 역순으로 만듭니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_sort(struct list \*list, list\_less\_func \*less, void \*aux)** |
| **Parameter** | list: 정렬할 리스트의 포인터  less: 요소들을 비교할 함수 포인터  aux: 비교 함수에 전달될 추가 데이터 |
| **Return** | 없음 |
| **Function** | 자연 반복형 병합 정렬(merge sort) 알고리즘을 이용해 리스트를 O(n lg n) 시간 내에 정렬합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_insert\_ordered(struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux)** |
| **Parameter** | list: 정렬된 상태를 유지할 대상 리스트  elem: 삽입할 새 요소의 포인터  less: 비교 함수 포인터  aux: 비교 함수에 전달될 추가 데이터 |
| **Return** | 없음 |
| **Function** | 리스트가 정렬된 상태라고 가정하고, 적절한 위치에 새 요소를 삽입하여 정렬 순서를 유지합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void list\_unique(struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \*aux)** |
| **Parameter** | list: 중복 제거를 수행할 원본 리스트  duplicates: 중복 요소들을 저장할 리스트의 포인터 (NULL일 경우 저장하지 않음)  less: 요소들이 같은지 비교할 함수 포인터  aux: 비교 함수에 전달될 추가 데이터 |
| **Return** | 없음 |
| **Function** | 인접한 중복 요소들을 제거하고, 필요 시 제거된 요소들을 별도의 리스트에 저장합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_max(struct list \*list, list\_less\_func \*less, void \*aux)** |
| **Parameter** | list: 최대값을 찾을 리스트의 포인터  less: 비교 함수 포인터  aux: 비교 함수에 전달될 추가 데이터 |
| **Return** | 리스트 내 최대값을 가진 요소의 포인터 (여러 개라면 가장 먼저 나타난 요소, 비어있으면 테일 반환) |
| **Function** | 리스트의 요소들을 순회하며 최대값을 가진 요소를 찾아 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_min(struct list \*list, list\_less\_func \*less, void \*aux)** |
| **Parameter** | list: 최소값을 찾을 리스트의 포인터  less: 비교 함수 포인터  aux: 비교 함수에 전달될 추가 데이터 |
| **Return** | 리스트 내 최소값을 가진 요소의 포인터 (여러 개라면 가장 먼저 나타난 요소, 비어있으면 테일 반환) |
| **Function** | 리스트의 요소들을 순회하며 최소값을 가진 요소를 찾아 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_begin(struct list \*list)** |
| **Parameter** | - list: 대상 리스트 구조체의 포인터 |
| **Return** | - 리스트의 첫 번째 실제 요소의 포인터    (리스트가 비어있다면, 테일(&list->tail)의 포인터를 반환) |
| **Function** | 리스트의 헤드 다음에 위치한 첫 번째 실제 요소를 반환하여, 리스트 순회의 시작점을 제공합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_next(struct list\_elem \*elem)** |
| **Parameter** | - elem: 현재 리스트 요소의 포인터 (헤드 또는 내부 요소) |
| **Return** | - 주어진 요소의 다음 요소를 가리키는 포인터    (마지막 실제 요소의 경우, 테일(&list->tail)의 포인터를 반환) |
| **Function** | 주어진 리스트 요소의 다음 요소를 반환하여, 리스트를 앞에서 뒤로 순회할 수 있도록 합니다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end(struct list \*list) |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트의 테일 요소의 포인터 |
| **Function** | 리스트의 끝을 나타내는 테일 요소를 반환합니다. 보통 순회 종료 조건으로 사용됩니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_rbegin(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트의 마지막 실제 요소의 포인터    (테일의 바로 앞 요소) |
| **Function** | 리스트를 역방향으로 순회할 때, 마지막 실제 요소를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_prev(struct list\_elem \*elem)** |
| **Parameter** | - elem: 현재 리스트 요소의 포인터 (내부 요소 또는 테일) |
| **Return** | - 주어진 요소의 이전 요소를 가리키는 포인터    (첫 번째 실제 요소인 경우, 헤드(&list->head)를 반환) |
| **Function** | 주어진 요소의 바로 앞에 위치한 요소를 반환하여, 리스트를 역방향으로 순회할 수 있도록 합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_rend(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트의 헤드 요소의 포인터 |
| **Function** | 역방향 순회 시 종료 조건으로, 리스트의 헤드(데이터가 아닌 더미)를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_head(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트의 헤드 요소의 포인터    (헤드는 실제 데이터 요소가 아니며, 리스트 시작을 나타냄) |
| **Function** | 리스트의 헤드를 반환합니다. 헤드는 리스트의 시작을 나타내며, 내부적으로 순회나 검사에 사용됩니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct list\_elem \*list\_tail(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트의 테일 요소의 포인터    (테일은 실제 데이터 요소가 아니며, 리스트의 끝을 나타냄) |
| **Function** | 리스트의 테일을 반환합니다. 테일은 순회 종료 조건 및 리스트 끝의 표시로 사용됩니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t list\_size(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - 리스트에 포함된 실제 요소의 개수 |
| **Function** | 리스트의 첫 요소부터 테일 이전까지 순회하면서 실제 데이터 요소의 개수를 세어, 리스트의 크기를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool list\_empty(struct list \*list)** |
| **Parameter** | - list: 대상 리스트의 포인터 |
| **Return** | - true: 리스트에 실제 요소가 없을 경우   - false: 하나 이상의 실제 요소가 있을 경우 |
| **Function** | 리스트의 첫 요소가 테일과 동일한지 확인하여, 리스트가 비어 있는지를 판단합니다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | **bool hash\_init(struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux)** |
| **Parameter** | h: 초기화할 해시 테이블을 가리키는 포인터 hash: 해시 값을 계산하는 함수 포인터 less: 해시 요소를 비교하는 함수 포인터 aux: 보조 데이터 (비교나 해싱에 사용) |
| **Return** | true: 초기화 성공, false: 메모리 할당 실패 등으로 초기화 실패 |
| **Function** | 해시 테이블을 초기화하여 기본 버킷 수(4개)를 할당하고, 함수 포인터와 보조 데이터를 설정한 후 버킷들을 빈 상태로 만듭니다. |

|  |  |
| --- | --- |
| **Prototype** | **void hash\_clear(struct hash \*h, hash\_action\_func \*destructor)** |
| **Parameter** | h: 해시 테이블을 가리키는 포인터 destructor: 각 해시 요소에 대해 호출할 함수 포인터 (필요시 메모리 해제 등), NULL일 경우 호출하지 않음 |
| **Return** | **없음 (void)** |
| **Function** | 해시 테이블의 모든 버킷을 순회하며 요소들을 제거하고, 필요 시 각 요소에 destructor를 호출한 후 버킷을 초기화합니다. 최종적으로 해시 테이블의 요소 개수를 0으로 만듭니다. |

|  |  |
| --- | --- |
| **Prototype** | **void hash\_destroy(struct hash \*h, hash\_action\_func \*destructor)** |
| **Parameter** | - h: 해시 테이블을 가리키는 포인터   - destructor: 각 요소에 대해 호출할 함수 포인터 (NULL이면 호출하지 않음) |
| **Return** | 없음 (void) |
| **Function** | 해시 테이블의 모든 요소를 제거(필요 시 destructor 호출)한 후, 할당된 버킷 메모리를 해제하여 테이블을 파괴합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_insert(struct hash \*h, struct hash\_elem \*new)** |
| **Parameter** | - h: 해시 테이블을 가리키는 포인터   - new: 삽입할 새로운 해시 요소를 가리키는 포인터 |
| **Return** | - NULL: 삽입 성공 (동일한 요소가 없었음)   - 기존 요소의 포인터: 이미 동일한 요소가 있어서 삽입하지 않고 반환 |
| **Function** | 해시 테이블에 새로운 요소를 삽입하며, 동일한 값의 요소가 이미 있으면 그 요소를 반환합니다. 삽입 후 자동으로 rehash()를 호출하여 버킷 개수를 조정합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void hash\_apply(struct hash \*h, hash\_action\_func \*action)** |
| **Parameter** | h: 해시 테이블을 가리키는 포인터 action: 각 해시 요소에 대해 실행할 함수 포인터 |
| **Return** | 없음 (void) |
| **Function** | 해시 테이블의 모든 버킷을 순회하면서 각 요소에 대해 action() 함수를 호출합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool hash\_empty(struct hash \*h)** |
| **Parameter** | h: 해시 테이블을 가리키는 포인터 |
| **Return** | true: 해시 테이블이 비어 있음, false: 하나 이상의 요소가 있음 |
| **Function** | 해시 테이블에 요소가 하나도 없으면 true를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **unsigned hash\_int(int i)** |
| **Parameter** | i: 해시 계산에 사용할 정수 값 |
| **Return** | 정수에 대한 해시 값 |
| **Function** | 정수 i의 메모리 바이트를 대상으로 hash\_bytes()를 호출하여 해시 값을 반환합니다 |

|  |  |
| --- | --- |
| **Prototype** | **static struct hash\_elem \*find\_elem(struct hash \*h, struct list \*bucket, struct hash\_elem \*e)** |
| **Parameter** | h: 해시 테이블을 가리키는 포인터 bucket: 검색할 버킷(리스트)의 포인터 e: 검색할 해시 요소를 가리키는 포인터 |
| **Return** | 버킷 내에서 e와 동일한 해시 요소의 포인터, 없으면 NULL |
| **Function** | 지정된 버킷을 순회하며, 비교 함수(less)를 사용해 e와 동일한 해시 요소를 찾아 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_delete(struct hash \*h, struct hash\_elem \*e)** |
| **Parameter** | h: 해시 테이블을 가리키는 포인터 e: 삭제할 해시 요소를 가리키는 포인터 |
| **Return** | 삭제된 요소의 포인터 (존재하면), 없으면 NULL |
| **Function** | 해시 테이블에서 e와 동일한 요소를 찾아 삭제한 후, 삭제된 요소를 반환하며, 삭제 후 rehash()를 호출합니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t hash\_size(struct hash \*h)** |
| **Parameter** | **h**: 해시 테이블의 포인터 |
| **Return** | 해시 테이블에 저장된 요소의 총 개수. |
| **Function** | 해시 테이블에 저장된 요소 수를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_replace(struct hash \*h, struct hash\_elem \*new)** |
| **Parameter** | **h: 해시 테이블을 가리키는 포인터   - new: 삽입할 새로운 해시 요소를 가리키는 포인터** |
| **Return** | 기존에 동일한 요소가 있었으면 그 요소의 포인터, 없으면 NULL |
| **Function** | 해시 테이블에 새로운 요소를 삽입하되, 동일한 요소가 이미 있으면 제거하고 대신 새 요소를 넣습니다. 삽입 후 rehash()를 호출합니다.. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_find(struct hash \*h, struct hash\_elem \*e)** |
| **Parameter** | - h: 해시 테이블을 가리키는 포인터   - e: 찾고자 하는 해시 요소를 가리키는 포인터 |
| **Return** | - 동일한 요소가 존재하면 그 해시 요소의 포인터, 없으면 NULL |
| **Function** | 해시 테이블 H에서 e와 동일한 요소를 검색하여 반환합니다.   동일한 요소가 없으면 NULL을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void hash\_first(struct hash\_iterator \*i, struct hash \*h)** |
| **Parameter** | - i: 해시 테이블 순회를 위한 이터레이터를 가리키는 포인터   - h: 순회할 해시 테이블을 가리키는 포인터 |
| **Return** | 없음 (void) |
| **Function** | 해시 테이블 H의 순회를 시작하기 위해 이터레이터 i를 초기화합니다.   i는 해시 테이블의 첫 번째 버킷의 첫 번째 요소를 가리키도록 설정됩니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_next(struct hash\_iterator \*i)** |
| **Parameter** | - i: 해시 테이블 순회를 위한 이터레이터를 가리키는 포인터 |
| **Return** | - 다음 요소를 가리키는 포인터, 더 이상 요소가 없으면 NULL |
| **Function** | 해시 테이블 H의 이터레이터 i를 다음 요소로 진행시키고, 해당 요소를 반환합니다.   순회가 끝나면 NULL을 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct hash\_elem \*hash\_cur(struct hash\_iterator \*i)** |
| **Parameter** | - i: 해시 테이블 순회를 위한 이터레이터를 가리키는 포인터 |
| **Return** | - 현재 이터레이터 i가 가리키는 해시 요소의 포인터, 순회가 끝난 경우 NULL |
| **Function** | 해시 테이블 순회 중 현재 이터레이터 i가 가리키는 요소를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **unsigned hash\_bytes(const void \*buf, size\_t size)** |
| **Parameter** | - buf: 해시 계산에 사용할 데이터 버퍼를 가리키는 포인터   - size: 버퍼의 바이트 크기 |
| **Return** | - 계산된 해시 값 (unsigned 정수) |
| **Function** | Fowler-Noll-Vo(FNV) 32비트 해시 알고리즘을 사용하여,   주어진 버퍼 buf의 size 바이트에 대한 해시 값을 계산하여 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **unsigned hash\_string(const char \*s)** |
| **Parameter** | - s: 해시 계산에 사용할 문자열을 가리키는 포인터 |
| **Return** | - 계산된 해시 값 (unsigned 정수) |
| **Function** | Fowler-Noll-Vo(FNV) 32비트 해시 알고리즘을 사용하여,   문자열 s의 각 문자를 대상으로 해시 값을 계산하고, 최종적으로 반환합니다.   문자열의 널 종료 문자까지 계산됩니다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_set(struct bitmap \*b, size\_t idx, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - idx: 설정할 비트의 인덱스 (0 ≤ idx < b->bit\_cnt)   - value: 설정할 값 (true: 1, false: 0) |
| **Return** | - 없음 (void) |
| **Function** | 비트맵 b의 인덱스 idx에 해당하는 비트를 value 값으로 설정합니다. 내부적으로 bitmap\_mark() 또는 bitmap\_reset()를 호출합니다. |

|  |  |
| --- | --- |
| **Prototype** | **struct bitmap \*bitmap\_create(size\_t bit\_cnt)** |
| **Parameter** | - bit\_cnt: 생성할 비트맵의 총 비트 수 |
| **Return** | - 성공 시 비트맵을 가리키는 포인터, 실패 시 NULL |
| **Function** | 주어진 비트 수(bit\_cnt)를 가진 비트맵을 동적으로 생성하며, 모든 비트를 false(꺼짐)으로 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_destroy(struct bitmap \*b)** |
| **Parameter** | b: 해제할 비트맵을 가리키는 포인터 |
| **Return** | 없음 (void) |
| **Function** | 비트맵 b의 비트 배열과 비트맵 구조체 자체의 메모리를 해제합니다.   (단, bitmap\_create\_in\_buf()로 생성된 경우에는 사용하지 않습니다.) |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_mark(struct bitmap \*b, size\_t bit\_idx)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - bit\_idx: true로 설정할 비트의 인덱스 |
| **Return** | 없음 (void) |
| **Function** | 비트맵 b의 bit\_idx에 해당하는 비트를 true(켜짐)으로 설정합니다. (어셈블리 명령을 사용하여 원자적 연산 보장) |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_reset(struct bitmap \*b, size\_t bit\_idx)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - bit\_idx: false로 설정할 비트의 인덱스 |
| **Return** | - 없음 (void) |
| **Function** | 비트맵 b의 bit\_idx에 해당하는 비트를 false(꺼짐)으로 설정합니다. (어셈블리 명령을 사용하여 원자적 연산 보장) |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_flip(struct bitmap \*b, size\_t bit\_idx)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - bit\_idx: 값을 반전시킬 비트의 인덱스 |
| **Return** | 없음 |
| **Function** | 비트맵 b의 bit\_idx에 해당하는 비트의 값을 반전(토글)시킵니다. (어셈블리 명령을 사용하여 원자적 연산 보장) |

|  |  |
| --- | --- |
| **Prototype** | **bool bitmap\_test(const struct bitmap \*b, size\_t idx)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - idx: 조회할 비트의 인덱스 |
| **Return** | - true: 비트가 1로 설정되어 있음, false: 0으로 설정되어 있음 |
| **Function** | 비트맵 b의 인덱스 idx에 해당하는 비트의 값을 검사하여 true 또는 false를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_set\_all(struct bitmap \*b, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - value: 모든 비트에 설정할 값 (true 또는 false) |
| **Return** | 없음 (void) |
| **Function** | 비트맵 b의 모든 비트를 value 값으로 설정합니다. 내부적으로 bitmap\_set\_multiple()를 호출합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_set\_multiple(struct bitmap \*b, size\_t start, size\_t cnt, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 설정 시작 인덱스   - cnt: 설정할 비트의 개수   - value: 설정할 값 (true 또는 false) |
| **Return** | 없음 (void) |
| **Function** | 비트맵 b의 start부터 start+cnt 전까지의 비트를 value 값으로 설정합니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t bitmap\_count(const struct bitmap \*b, size\_t start, size\_t cnt, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검사 시작 인덱스   - cnt: 검사할 비트의 개수   - value: 검사할 값 (true 또는 false) |
| **Return** | - 지정된 범위 내에서 value 값과 일치하는 비트의 개수 |
| **Function** | 비트맵 b의 start부터 start+cnt 전까지의 비트 중 value와 일치하는 비트의 개수를 세어 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool bitmap\_contains(const struct bitmap \*b, size\_t start, size\_t cnt, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검사 시작 인덱스   - cnt: 검사할 비트의 개수   - value: 검사할 값 (true 또는 false) |
| **Return** | - true: 검사 범위 내에 value와 일치하는 비트가 하나라도 존재하면, false: 없으면 |
| **Function** | 비트맵 b의 start부터 start+cnt 전까지의 비트 중 value와 일치하는 비트가 있는지 확인합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool bitmap\_any(const struct bitmap \*b, size\_t start, size\_t cnt)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검사 시작 인덱스   - cnt: 검사할 비트의 개수 |
| **Return** | - true: 검사 범위 내에 true로 설정된 비트가 하나라도 존재하면, false: 없으면 |
| **Function** | bitmap\_contains()를 이용하여, 지정 범위 내에 하나라도 true인 비트가 있는지 확인합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool bitmap\_none(const struct bitmap \*b, size\_t start, size\_t cnt)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검사 시작 인덱스   - cnt: 검사할 비트의 개수 |
| **Return** | - true: 검사 범위 내에 true로 설정된 비트가 하나도 없으면, false: 하나 이상 있으면 |
| **Function** | bitmap\_contains()를 반대로 사용하여, 지정 범위 내에 true인 비트가 전혀 없는지 확인합니다. |

|  |  |
| --- | --- |
| **Prototype** | **bool bitmap\_all(const struct bitmap \*b, size\_t start, size\_t cnt)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검사 시작 인덱스   - cnt: 검사할 비트의 개수 |
| **Return** | - true: 검사 범위 내의 모든 비트가 true로 설정되어 있으면, false: 하나라도 false가 있으면 |
| **Function** | bitmap\_contains()를 이용하여, 지정 범위 내의 모든 비트가 true인지 확인합니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t bitmap\_scan(const struct bitmap \*b, size\_t start, size\_t cnt, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검색 시작 인덱스   - cnt: 연속적으로 검사할 비트의 개수   - value: 찾고자 하는 비트 값 (true 또는 false) |
| **Return** | 연속된 cnt개의 비트가 value로 설정된 첫 시작 인덱스, 없으면 BITMAP\_ERROR |
| **Function** | 비트맵 b에서 start 이후 연속된 cnt개의 비트가 모두 value와 일치하는 첫 번째 위치를 찾아 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t bitmap\_scan\_and\_flip(struct bitmap \*b, size\_t start, size\_t cnt, bool value)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터   - start: 검색 시작 인덱스   - cnt: 연속적으로 검사할 비트의 개수   - value: 찾고자 하는 비트 값 (true 또는 false) |
| **Return** | - 연속된 cnt개의 비트가 value로 설정된 첫 시작 인덱스, 없으면 BITMAP\_ERROR |
| **Function** | 비트맵 b에서 start 이후 연속된 cnt개의 비트가 모두 value와 일치하는 구간을 찾고, 해당 구간의 모든 비트를 반전(!value)시킨 후, 그 시작 인덱스를 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **size\_t bitmap\_file\_size(const struct bitmap \*b)** |
| **Parameter** | - b: 대상 비트맵을 가리키는 포인터 |
| **Return** | - 비트맵 b를 파일에 저장하기 위해 필요한 바이트 수 |
| **Function** | 비트맵 b에 저장된 전체 비트를 파일에 저장할 때 필요한 총 바이트 수를 계산하여 반환합니다. |

|  |  |
| --- | --- |
| **Prototype** | **void bitmap\_dump(const struct bitmap \*b)** |
| **Parameter** | **- b: 덤프할 비트맵을 가리키는 포인터** |
| **Return** | **- 없음 (void)** |
| **Function** | **비트맵 b의 내부 비트 배열을 hex\_dump()를 통해 헥사값 형태로 콘솔에 출력하여 디버깅에 도움을 줍니다.** |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf(size\_t bit\_cnt, void \*block, size\_t block\_size) |
| **Parameter** | - bit\_cnt: 생성할 비트맵의 총 비트 수   - block: 비트맵을 저장할 미리 할당된 메모리 블록을 가리키는 포인터   - block\_size: 메모리 블록의 크기(바이트 단위). 반드시 bitmap\_buf\_size(bit\_cnt) 이상이어야 함 |
| **Return** | - 성공 시: 생성된 비트맵을 가리키는 포인터   - 실패 시: (주로 assert()에 의해 중단되므로 반환값 정의 없음) |
| **Function** | 미리 할당된 메모리 블록(block) 내에 bit\_cnt 비트를 가지는 비트맵을 생성하고, 모든 비트를 false로 초기화합니다. 블록 크기가 충분한지 확인한 후, 비트맵 헤더와 내부 비트 배열을 초기화합니다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size(size\_t bit\_cnt) |
| **Parameter** | - bit\_cnt: 비트맵에 포함될 총 비트 수 |
| **Return** | - 비트맵을 저장하는 데 필요한 총 바이트 수    (비트맵 구조체 헤더와 비트를 저장하는 배열이 포함됨) |
| **Function** | 비트맵을 미리 할당된 메모리 블록에 저장할 때 필요한 버퍼의 크기를 계산합니다. 이 값은 bitmap\_create\_in\_buf() 호출 전에 적절한 블록 크기를 결정하는 데 사용됩니다 |