**Pintos Project 0-2: Pintos Data Structure**

|  |  |
| --- | --- |
| 담당 교수 : | 박성용 교수님 |
| 학번 : | 20161211 |
| 이름 : | 김동빈 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | char\* getName() |
| **Parameter** | None |
| **Return** | 입력 받은 문자열 |
| **Function** | getchar()를 이용하여 STDOUT으로부터 ‘ ‘, ‘\n’, EOF 단위로 쪼개진 문자열을 입력 받고 해당 문자열을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | COMMAND getCommad() |
| **Parameter** | None |
| **Return** | 입력 받은 명령어에 해당하는 enum 값 |
| **Function** | getName() 함수를 호출하여 STDOUT으로부터 명령어를 입력 받고 switch문을 이용하여 해당 명령어에 해당하는 enum 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | DATA\_STRUCTURE getDataStructure() |
| **Parameter** | None |
| **Return** | 입력 받은 자료구조에 해당하는 enum 값 |
| **Function** | getName() 함수를 호출하여 STDOUT으로부터 자료구조를 입력 받고 switch문을 이용하여 해당 자료구조에 해당하는 enum 값을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | int findEmptyIndex(DATA\_STRUCTURE data\_struct) |
| **Parameter** | 자료구조에 해당하는 enum 값 |
| **Return** | 해당 자료구조 배열에 비어 있는 공간의 index |
| **Function** | switch문을 이용하여 해당 자료구조에 해당하는 배열 중 사용가능한 index를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | int findTargetIndex(DATA\_STRUCTURE data\_struct, char\* targetName) |
| **Parameter** | 자료구조에 해당하는 enum 값과 자료구조 이름 |
| **Return** | 해당 이름을 가진 자료구조 배열의 index |
| **Function** | switch문을 이용하여 해당 자료구조에 해당하는 배열 중 자료구조 중 주어진 이름과 같은 자료구조의 index를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_item  {  struct list\_elem elem;  int data;  }; |
| **Parameter** | None |
| **Return** | None |
| **Function** | list\_elem과 정수형 data를 원소로 가지는 구조체 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_name {  struct list\* core;  char name[50];  bool allocated;  bool used;  }; |
| **Parameter** | None |
| **Return** | None |
| **Function** | listArray의 원소를 구현하기 위한 구조체로 list와 list의 이름, 메모리 할당 여부, list 사용 여부를 표현한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem\* a, const struct list\_elem\* b,  void\* aux) |
| **Parameter** | 대소를 비교할 원소 a, b |
| **Return** | 대소 비교 결과 |
| **Function** | a, b를 포함하고 있는 struct list\_item의 data를 비교하여 item\_a의 data가 item\_b의 data보다 작으면 true, 그 외의 경우는 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_item\_free(struct list\_elem\* e, void\* aux) |
| **Parameter** | 삭제할 elem |
| **Return** | None |
| **Function** | elem을 포함하고 있는 struct list\_item의 메모리를 free를 이용해 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list\* list) |
| **Parameter** | 원소의 순서를 바꿀 list |
| **Return** | None |
| **Function** | list의 원소의 순서를 random number와 list\_swap함수를 이용해 무작위로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem\* a, struct list\_elem\* b) |
| **Parameter** | 서로 위치를 교환할 a, b |
| **Return** | None |
| **Function** | list내에서 pointer를 바꾸어 a와 b의 위치를 교환한다. |

|  |  |
| --- | --- |
| **Prototype** | void createList() |
| **Parameter** | None |
| **Return** | None |
| **Function** | findEmptyIndex함수를 호출하여 listArray 배열에 비어 있는 index를 받고 해당 index에 메모리가 아직 할당되지 않은 경우 메모리를 할당하고 list\_init 함수를 호출해 빈 LIST를 만들어 준다. 메모리가 이미 할당되어 있는 경우에는 getName() 함수를 호출하여 LIST의 이름을 받아 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void deleteList(int deleteIndex) |
| **Parameter** | listArray에서 삭제할 LIST의 index |
| **Return** | None |
| **Function** | deleteIndex에 해당하는 LIST가 빌 때까지 list\_pop\_front함수를 이용하여 원소를 제거하고 해당 index의 LIST 이름을 ‘\0’으로 바꿔준다. 이때 해당 index에 새로 LIST를 create할 경우를 대비하여 메모리를 해제하지 않고 empty 상태로 유지한다. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdataList(int targetIndex) |
| **Parameter** | listArray에서 살펴볼 LIST의 index |
| **Return** | None |
| **Function** | targetIndex에 해당하는 LIST에 들어있는 모든 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_front\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_front함수를 이용해 LIST의 front에 해당하는 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_back\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_back함수를 이용해 LIST의 back에 해당하는 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 LIST내에 삽입할 position과 원소의 data를 입력 받아 list\_insert 함수를 이용해 LIST의 해당 position에 원소를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 LIST내에 삽입할 원소의 data를 입력 받아 list\_insert\_ordered 함수를 이용해 LIST의 해당 position에 원소를 삽입한다. 그 결과 LIST내 원소는 삽입 이후 정렬이 된 모습을 보인다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_max\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_max함수를 이용해 LIST의 data가 최대인 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_min\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_min함수를 이용해 LIST의 data가 최소인 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_size\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_size함수를 이용해 LIST의 크기를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_empty\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_empty함수를 이용해 LIST가 empty라면 true를 출력, 아니라면 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 push할 data를 입력받아 list\_push\_front함수를 이용해 LIST의 front에 원소를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 push할 data를 입력받아 list\_push\_back함수를 이용해 LIST의 back에 원소를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_front\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_pop\_front함수를 이용해 LIST의 front에서 원소를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_back\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_pop\_back함수를 이용해 LIST의 back에서 원소를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_remove\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 삭제할 원소의 position을 입력 받아 list\_remove함수를 이용해 LIST로부터 원소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_reverse함수를 이용해 LIST의 원소의 순서를 반대로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_shuffle함수를 이용해 LIST 내 원소의 순서를 무작위로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 list\_reverse함수를 이용해 LIST의 원소를 data의 오름차순으로 정렬한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | from과 to에 해당하는 LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 각각 구하고 from LIST에서 splice될 원소의 범위를 의미하는 start, end와 to LIST에서 삽입될 위치인 insertPosition을 입력 받아 list\_splice함수를 이용해 from LIST의 start와 end사이의 원소들을 to LIST의 insertposition에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 서로 교환할 두 index를 입력 받아 list\_swap함수를 이용해 LIST의 두 원소의 위치를 교환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | LIST 이름과 listArray내에서 해당 이름을 가진 LIST의 index를 구하고 LIST의 이름이 하나만 주어지는 경우는 해당 LIST에서 unique한 data를 가진 원소들만 남기고 나머지 원소들은 제거하며 LIST의 이름이 두개 주어지는 경우 첫번째 LIST에는 unique한 data를 가진 원소들만 남기고 나머지 원소들은 두번째 LIST에 추가해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_item  {  struct hash\_elem elem;  int data;  }; |
| **Parameter** | None |
| **Return** | None |
| **Function** | hash\_elem과 정수형 data를 원소로 가지는 구조체 |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_name {  struct hash\* core;  char name[50];  bool allocated;  bool used;  }; |
| **Parameter** | None |
| **Return** | None |
| **Function** | hashArray의 원소를 구현하기 위한 구조체로 hashtable과 hashtable의 이름, 메모리 할당 여부, hashtable 사용 여부를 표현한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_func(const struct hash\_elem\* e, void\* aux) |
| **Parameter** | key 값을 구하기 위한 hash 원소 e, auxiliary data aux |
| **Return** | e에 대한 key 값 |
| **Function** | e를 포함하는 hash\_item을 찾고 해당 hash\_item의 data를 hash\_int함수에 적용한 결과를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem\* a, const struct hash\_elem\* b, void\* aux) |
| **Parameter** | 대소 비교를 할 hash\_elem\* a, b, auxiliary data aux |
| **Return** | a, b가 정렬기준을 따르는지 여부 |
| **Function** | a와 b를 포함하는 hash\_item을 각각 item\_a, item\_b라 할 때 item\_a의 data가 item\_b의 data보다 작다면 true, 그 이외의 경우에는 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square(struct hash\_elem\* e, void\* aux) |
| **Parameter** | data를 바꿔줄 원소 e |
| **Return** | None |
| **Function** | e를 포함하는 hash\_item의 data를 제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple(struct hash\_elem\* e, void\* aux) |
| **Parameter** | data를 바꿔줄 원소 e |
| **Return** | None |
| **Function** | e를 포함하는 hash\_item의 data를 세제곱한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_item\_free(struct hash\_elem\* e, void\* aux) |
| **Parameter** | 메모리를 해제할 원소 e |
| **Return** | None |
| **Function** | e를 포함하는 hash\_item의 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | key 값을 구하기 위한 정수 i |
| **Return** | i에 대한 key값 |
| **Function** | i를 13으로 나눈 나머지를 key값으로 반환한다. 이때 13은 hash의 성능을 높이기 위해 소수를 사용함을 이용해 선정한 값이다. |

|  |  |
| --- | --- |
| **Prototype** | void createHashtable() |
| **Parameter** | None |
| **Return** | None |
| **Function** | findEmptyIndex함수를 호출하여 hashArray 배열에 비어 있는 index를 받고 해당 index에 메모리가 아직 할당되지 않은 경우 메모리를 할당하고 hash\_init 함수를 호출해 빈 HASH TABLE을 만들어 준다. 메모리가 이미 할당되어 있는 경우에는 getName 함수를 호출하여 HASH TABLE의 이름을 받아 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void deleteHashtable(int deleteIndex) |
| **Parameter** | hashArray에서 삭제할 HASHTABLE의 index |
| **Return** | None |
| **Function** | hash\_clear함수를 이용해 deleteIndex에 HASH TABLE로부터 원소를 모두 제거하고 해당 index의 HASH TABLE 이름을 ‘\0’으로 바꿔준다. 이때 해당 index에 새로 HASH TABLE을 create할 경우를 대비하여 메모리를 해제하지 않고 empty 상태로 유지한다. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdataHashtable(int targetIndex) |
| **Parameter** | hashArray에서 살펴볼 LIST의 index |
| **Return** | None |
| **Function** | hash\_iterator를 이용해 targetIndex에 해당하는 HASH TABLE에 들어있는 모든 원소의 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_insert\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 삽입할 data를 입력 받아 hash\_insert함수를 이용해 HASH TABLE에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_delete\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 삭제할 data를 입력 받아 hash\_delete함수를 이용해 HASH TABLE에서 해당 원소를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 option을 입력 받아 hash\_apply 함수를 이용해 HASH TABLE의 모든 원소에 해당 option을 적용한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_empty\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 hash\_empty함수를 이용해 HASH TABLE이 empty이면 true, 아니라면 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_size\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 hash\_size함수를 이용해 HASH TABLE의 크기를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 hash\_clear함수를 이용해 HASH TABLE내의 모든 원소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_find\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 data를 입력 받아 hash\_find함수를 이용해 HASH TABLE내에 해당 data를 가진 원소가 있다면 data를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_replace\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | HASH TABLE의 이름과 hashArray에서 해당 이름을 가진 HASH TABLE의 index를 구하고 data를 입력 받아 hash\_replace함수를 이용해 HASH TABLE내에 해당 data를 가진 원소가 있다면 원소를 대체해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\_name {  struct bitmap \* core;  char name[50];  bool used;  }; |
| **Parameter** | None |
| **Return** | None |
| **Function** | bitmapArray의 원소를 구현하기 위한 구조체로 bitmap과 bitmap의 이름, bitmap 사용 여부를 표현한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap\* bitmap, int size) |
| **Parameter** | target bitmap bitmap, 추가할 size |
| **Return** | bitmap에 size만큼의 bit을 추가한 bitmap |
| **Function** | Bitmap에 size만큼의 bit을 추가하고 추가된 bit는 false로 set한 상태로 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void createBitmap() |
| **Parameter** | None |
| **Return** | None |
| **Function** | findEmptyIndex함수를 호출하여 bitmapArray 배열에 비어 있는 index를 받고 getName 함수를 호출하여 BITMAP의 이름을 받아 저장한 후 bit\_cnt를 입력 받아 bitmap\_create함수를 이용해 BITMAP을 생성한다. |

|  |  |
| --- | --- |
| **Prototype** | void deleteBitmap(int deleteIndex) |
| **Parameter** | bitmapArray에서 삭제할 BITMAP의 index |
| **Return** | None |
| **Function** | hash\_clear함수를 이용해 deleteIndex의 BITMAP을 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdataHashtable(int targetIndex) |
| **Parameter** | hashArray에서 살펴볼 LIST의 index |
| **Return** | None |
| **Function** | bitmap\_test 함수를 이용해 targetIndex에 해당하는 BITMAP을 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 mark할 idx를 입력 받아 bitmap\_mark함수를 이용해 해당 idx의 bit을 1로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_all\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt를 입력 받아 bitmap\_all함수를 이용해 start부터 cnt개의 bit가 모두 true인 경우 true를 출력, 아닌 경우 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_any\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt를 입력 받아 bitmap\_any함수를 이용해 start부터 cnt개의 bit내에 true가 하나라도 존재하는 경우 true를 출력, 아닌 경우 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_contains\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt, 특정 truth value를 입력 받아 bitmap\_contains함수를 이용해 start부터 cnt개의 bit내에 value와 같은 truth value가 하나라도 존재하는 경우 true를 출력, 아닌 경우 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_count\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt, 특정 truth value를 입력 받아 bitmap\_count함수를 이용해 start부터 cnt개의 bit내에 value와 같은 truth value를 가진 bit의 개수를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 bitmap\_dump함수를 이용해 해당 BITMAP을 hexadecimal로 표현한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_expand\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 size를 입력 받아 bitmap\_expand 함수를 이용해 BITMAP을 size만큼 크기를 키운다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 flip할 idx를 입력 받아 bitmap\_flip함수를 이용해 해당 idx의 bit을 flip한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_none\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt를 입력 받아 bitmap\_none함수를 이용해 start부터 cnt개의 bit가 모두 false인 경우 true를 출력, 아닌 경우 false를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 reset할 idx를 입력 받아 bitmap\_reset함수를 이용해 해당 idx의 bit을 0으로 reset한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_scan\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt, 특정 truth value를 입력 받아 bitmap\_scan함수를 이용해 start부터 cnt개의 bit내에 value와 같은 truth value를 가진 bit의 첫번째 index를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_scan\_and\_flip\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt, 특정 truth value를 입력 받아 bitmap\_scan\_and\_flip함수를 이용해 start부터 cnt개의 bit내에 value와 같은 truth value를 가진 bit의 첫번째 index를 출력한다. 또한 start부터 cnt개의 bit을 모두 flip한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 idx와 특정 truth value를 입력 받아 bitmap\_set함수를 이용해 idx에 해당하는 bit을 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 특정 truth value를 입력 받아 bitmap\_set\_all함수를 이용해 BITMAP의 모든 bit을 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 start와 cnt, 특정 truth value를 입력 받아 bitmap\_set\_multiple함수를 이용해 start부터 cnt개의 bit을 value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_size\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 bitmap\_size함수를 이용하여 BITMAP의 크기를 출력한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_size\_command() |
| **Parameter** | None |
| **Return** | None |
| **Function** | BITMAP의 이름과 bitmapArray에서 해당 이름을 가진 BITMAP의 index를 구하고 idx를 입력 받아 bitmap\_test함수를 이용하여 BITMAP에서 해당 idx의 bit이 1이면 true, 0이면 false를 출력한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry(LIST\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next \  - offsetof (STRUCT, MEMBER.next))) |
| **Parameter** | LIST\_ELEM주소, 외부 struct 이름, 외부 struct의 변수 이름 |
| **Return** | 외부 struct의 주소 |
| **Function** | LIST\_ELEM주소의 주소를 이용해 해당 LIST ELEM을 변수이름으로 하는 외부 struct의 주소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static bool is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux); |
| **Parameter** | 정렬되었는지 확인할 list\_elem\* a, b, 정렬 함수 less, auxiliary data aux |
| **Return** | 정렬 여부 |
| **Function** | list\_elem\* a, b가 less에서 정의한 순서로 정렬되어 있는지 여부를 true또는 false로 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_head (struct list\_elem \*elem) |
| **Parameter** | head인지 알아볼 list\_elem \* elem |
| **Return** | elem이 head인지 여부 |
| **Function** | elem이 LIST이 head라면 true, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_interior (struct list\_elem \*elem) |
| **Parameter** | interior인지 알아볼 list\_elem \* elem |
| **Return** | elem이 interior인지 여부 |
| **Function** | elem이 LIST이 head나 tail이 아니라면 true, 그 이외의 경우는 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool is\_tail (struct list\_elem \*elem) |
| **Parameter** | tail인지 알아볼 list\_elem \* elem |
| **Return** | elem이 tail인지 여부 |
| **Function** | elem이 LIST이 tail이라면 true, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*list) |
| **Parameter** | 초기화할 list\* list |
| **Return** | None |
| **Function** | list의 head와 tail을 세팅함으로써 list를 초기화한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_begin (struct list \*list) |
| **Parameter** | 첫번째 element를 확인할 list |
| **Return** | list의 첫번째 element |
| **Function** | list의 첫번째 element를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_next (struct list\_elem \*elem) |
| **Parameter** | 다음 element를 확인할 elem |
| **Return** | elem의 다음 element |
| **Function** | elem의 next로 연결된 다음 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_end (struct list \*list) |
| **Parameter** | tail을 확인할 list |
| **Return** | list의 tail node |
| **Function** | list의 tail을 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_rbegin (struct list \*list) |
| **Parameter** | 뒤에서 첫번째 element를 확인할 list |
| **Return** | list의 뒤에서 첫번째 element |
| **Function** | list의 뒤에서 첫번째 element를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_prev (struct list\_elem \*elem) |
| **Parameter** | 이전 element를 확인할 elem |
| **Return** | elem의 이전 element |
| **Function** | elem의 prev로 연결된 이전 element를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_rend (struct list \*list) |
| **Parameter** | head을 확인할 list |
| **Return** | list의 head node |
| **Function** | list의 head를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_head (struct list \*list) |
| **Parameter** | head을 확인할 list |
| **Return** | list의 head node |
| **Function** | list의 head를 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_tail (struct list \*list) |
| **Parameter** | tail을 확인할 list |
| **Return** | list의 tail node |
| **Function** | list의 tail을 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | insert 할 위치를 의미하는 before, insert의 대상인 elem |
| **Return** | None |
| **Function** | before 직전의 위치에 elem을 insert해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | insert 할 위치를 의미하는 before, insert의 범위인 first, last |
| **Return** | None |
| **Function** | before 직전의 위치에 first부터 last까지의 element를 insert해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | push\_front할 list, 대상이 되는 원소인 elem |
| **Return** | None |
| **Function** | list\_insert 함수를 이용해 list의 front에 elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | push\_back할 list, 대상이 되는 원소인 elem |
| **Return** | None |
| **Function** | list\_insert 함수를 이용해 list의 back에 elem을 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_remove (struct list\_elem \*elem) |
| **Parameter** | 삭제할 원소인 elem |
| **Return** | elem의 다음 원소 |
| **Function** | elem을 LIST로부터 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_pop\_front (struct list \*list) |
| **Parameter** | pop\_front할 list |
| **Return** | 제거된 element |
| **Function** | list\_remove 함수를 이용해 list의 front에서 하나의 원소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_pop\_back (struct list \*list) |
| **Parameter** | pop\_back할 list |
| **Return** | 제거된 element |
| **Function** | list\_remove 함수를 이용해 list의 back에서 하나의 원소를 제거한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_front (struct list \*list) |
| **Parameter** | front를 확인할 list |
| **Return** | list의 front에 해당하는 원소 |
| **Function** | list의 front 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_back (struct list \*list) |
| **Parameter** | back을 확인할 list |
| **Return** | list의 back에 해당하는 원소 |
| **Function** | list의 back 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*list) |
| **Parameter** | size를 확인할 list |
| **Return** | list의 size |
| **Function** | list의 size를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | empty 여부를 확인할 list |
| **Return** | list가 empty인지 아닌지 여부 |
| **Function** | list의 empty라면 true, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | 교환할 list\_elem\*의 pointer |
| **Return** | None |
| **Function** | a와 b의 list\_elem\*의 pointer를 서로 교환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | 원소의 순서를 반전시킬 list |
| **Return** | None |
| **Function** | list의 원소의 순서를 반대로 만든다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \* find\_end\_of\_run (struct list\_elem \*a,  struct list\_elem \*b, list\_less\_func \*less, void \*aux) |
| **Parameter** | list\_elem\* a, b, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | less의 정의상 nondecreasing하는 마지막 원소 |
| **Function** | a부터 시작하여 b까지 less의 정의상 nondecreasing order를 유지하는 마지막 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void inplace\_merge (struct list\_elem \*a0, struct list\_elem \*a1b0,  struct list\_elem \*b1, list\_less\_func \*less, void \*aux) |
| **Parameter** | list\_elem\* a0, a1b0, b1, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | None |
| **Function** | a0와 b1사이에서 a0, a1b0, b1의 순서로 less의 정의상 nondecreasing order를 유지할 수 있는 위치에 list\_splice함수를 이용해 a1b0를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 정렬할 list, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | None |
| **Function** | list를 less의 정의에 의해 정렬한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 원소를 삽입할 list, 삽입의 대상인 elem, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | None |
| **Function** | elem을 list에 삽입할 때마다 list를 less의 정의대로 정렬한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates,  list\_less\_func \*less, void \*aux) |
| **Parameter** | unique한 원소를 저장할 list, 중복되는 원소를 저장할 duplicates, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | None |
| **Function** | list에서 중복되는 원소들은 list에서 제거하여 duplicate에 삽입하여 list에는 모두 unique한 원소들만 남기도록 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_max (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 최댓값을 찾을 list, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | list에서 최댓값을 가지는 원소 |
| **Function** | list에서 less의 정의상 최댓값을 가지는 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \* list\_min (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | 최솟값을 찾을 list, 대소관계를 정의한 함수 less, auxiliary data aux |
| **Return** | list에서 최솟값을 가지는 원소 |
| **Function** | list에서 less의 정의상 최솟값을 가지는 원소를 반환한다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem \  - offsetof (STRUCT, MEMBER.list\_elem))) |
| **Parameter** | HASH\_ELEM주소, 외부 struct 이름, 외부 struct의 변수 이름 |
| **Return** | 외부 struct의 주소 |
| **Function** | HASH\_ELEM주소의 주소를 이용해 해당 HASH ELEM을 변수이름으로 하는 외부 struct의 주소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_elem\_to\_hash\_elem(LIST\_ELEM) \  list\_entry(LIST\_ELEM, struct hash\_elem, list\_elem) |
| **Parameter** | LIST\_ELEM 주소 |
| **Return** | LIST\_ELEM외부의 hash\_elem 주소 |
| **Function** | 해당 LIST\_ELEM의 외부 struct hash\_elem 주소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*h,  hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | 초기화할 hashtable h, 초기화에 사용할 hash function hash, 대소관계를 정의한 less, auxiliary data aux |
| **Return** | 올바르게 초기화 되었는지 여부 |
| **Function** | h의 값들을 초기화 하고 bucket에 메모리를 할당한다. 정상적으로 hash table의 bucket에 메모리가 할당되었다면 true를, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | clear할 hashtable h, 제거된 원소들이 어떤 action을 취할지를 나타내는 destructor 함수 |
| **Return** | None |
| **Function** | h의 모든 원소를 제거하고 제거된 원소는 destructor의 parameter가 되어 destructor함수를 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | clear할 hashtable h, 제거된 원소들이 어떤 action을 취할지를 나타내는 destructor 함수 |
| **Return** | None |
| **Function** | destructor가 NULL이 아닌 경우 hash\_clear함수를 호출하고 h의 buckets에 할당된 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | 원소를 삽입할 hashtable h, 삽입할 원소 new |
| **Return** | new와 같은 값을 가진 element |
| **Function** | find\_bucket함수를 이용해 new가 들어갈 bucket을 찾고 해당 bucket에서 new와 같은 data를 가진 old가 존재하는지 확인하고 old가 NULL이면 바로 insert\_elem 함수를 통해 원소를 삽입하고 old가 NULL이 아닌 경우 중복된 data이므로 삽입하지 않고 old를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | 원소를 대체할 hashtable h, 대체할 원소 new |
| **Return** | new로 인해 대체된 element |
| **Function** | find\_bucket함수를 이용해 new가 들어갈 bucket을 찾고 해당 bucket에서 new와 같은 data를 가진 old가 존재하는지 확인하고 old가 NULL인 경우는 그냥 삽입하고 old가 NULL이 아닌 경우 old를 제거하고 new를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 원소를 찾을 hashtable h, 찾을 원소 e |
| **Return** | e와 같은 data를 가진 h 내부의 원소 |
| **Function** | find\_bucket함수와 find\_element함수를 e와 같은 data를 가진 원소가 존재하는지 확인하고 존재한다면 그 원소를, 존재하지 않는다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 원소를 제거할 hashtable h, 제거할 원소 e |
| **Return** | 제거된 원소 |
| **Function** | find\_bucket함수와 find\_element함수를 e와 같은 data를 가진 원소가 존재하는지 확인하고 존재한다면 그 원소를 제거하고 그 원소를 반환하며, 존재하지 않는다면 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | 함수를 적용할 hashtable h, hashtable의 원소에 적용할 함수 action |
| **Return** | None |
| **Function** | h의 모든 원소에 action함수를 적용한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | hash iterator i, 탐색할 hashtable h |
| **Return** | None |
| **Function** | iterator i를 h의 첫번째 원소를 가리키도록 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_next (struct hash\_iterator \*i) |
| **Parameter** | hash iterator i |
| **Return** | i가 가리키는 원소의 다음 원소 |
| **Function** | iterator i가 hashtable에서 가리키는 원소의 다음 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \* hash\_cur (struct hash\_iterator \*i) |
| **Parameter** | hash iterator i |
| **Return** | i가 가리키는 현재 원소 |
| **Function** | iterator i가 hashtable에서 가리키는 현재 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*h) |
| **Parameter** | 크기를 확인할 hashtable h |
| **Return** | h의 크기 |
| **Function** | h의 원소의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | empty인지 여부를 확인할 hashtable h |
| **Return** | h가 empty인지 여부 |
| **Function** | h가 empty라면 true를, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*buf\_, size\_t size) |
| **Parameter** | key 값을 구하기 위한 size byte |
| **Return** | size byte에 대한 hash function의 key |
| **Function** | size byte에 대해 hash key를 구해 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*s\_) |
| **Parameter** | key 값을 구하기 위한 string |
| **Return** | string에 대한 hash function의 key |
| **Function** | string에 대해 hash key를 구해 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int i) |
| **Parameter** | key 값을 구하기 위한 정수 i |
| **Return** | i에 대한 hash function의 key |
| **Function** | i에 대해 hash key를 구해 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \* find\_bucket (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | bucket을 찾을 hashtable h, bucket을 찾을 elem |
| **Return** | h에서 elem이 속한 bucket |
| **Function** | h에서 elem을 포함하고 있는 bucket을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \* find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | elem를 찾을 hashtable h, elem을 찾을 bucket, 찾을 elem |
| **Return** | h에서 찾은 elem |
| **Function** | 주어진 h와 bucket에서 elem을 찾아 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t turn\_off\_least\_1bit (size\_t x) |
| **Parameter** | 숫자 x |
| **Return** | x & (x-1) |
| **Function** | x가 주어질 때 x & (x-1)을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t is\_power\_of\_2 (size\_t x) |
| **Parameter** | 숫자 x |
| **Return** | x가 2의 제곱수인지 여부 |
| **Function** | x가 주어질 때 x가 2의 제곱수라면 1을 반환 아니라면 0을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (struct hash \*h) |
| **Parameter** | rehash할 hashtable h |
| **Return** | None |
| **Function** | h의 bucket size를 변경하여 다시 hashtable로 정리한다. |

|  |  |
| --- | --- |
| **Prototype** | static void insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | e를 삽입할 hashtable h, e를 삽입할 bucket, 삽입할 element e |
| **Return** | None |
| **Function** | h의 bucket에 e를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | static void remove\_elem (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | e를 제거할 hashtable h, 제거할 element e |
| **Return** | None |
| **Function** | h로부터 e를 제거한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_idx (size\_t bit\_idx) |
| **Parameter** | bitmap의 index bit\_idx |
| **Return** | bit\_idx bit을 포함한 element의 index |
| **Function** | 주어진 bit\_idx에 해당하는 bit을 포함하는 element의 index를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type bit\_mask (size\_t bit\_idx) |
| **Parameter** | bitmap의 index bit\_idx |
| **Return** | bit\_idx bit이 true일 때 elem\_type |
| **Function** | 주어진 bit\_idx에 해당하는 bit이 true인 경우에 elem\_type으로 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t elem\_cnt (size\_t bit\_cnt) |
| **Parameter** | bitmap을 구성할 bit\_cnt |
| **Return** | bit\_cnt만큼의 bit를 위해 필요한 element의 수 |
| **Function** | 주어진 bit\_cnt만큼의 bit를 위해 필요한 element의 수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t byte\_cnt (size\_t bit\_cnt) |
| **Parameter** | bitmap을 구성할 bit\_cnt |
| **Return** | bit\_cnt만큼의 bit를 위해 필요한 byte 수 |
| **Function** | 주어진 bit\_cnt만큼의 bit를 위해 필요한 byte 수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type last\_mask (const struct bitmap \*b) |
| **Parameter** | last\_mask를 확인할 bitmap b |
| **Return** | b에서 사용된 last bit |
| **Function** | b에서 사용된 last bit을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \* bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | Bitmap의 size가 될 bit\_cnt |
| **Return** | bit\_cnt의 크기로 생성된 bitmap |
| **Function** | bit\_cnt의 크기로 bitmap의 bit에 메모리를 할당하고 모두 false로 초기화한 후 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*  bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*block, size\_t block\_size ) |
| **Parameter** | Bitmap의 size가 될 bit\_cnt, block, block의 크기 |
| **Return** | bit\_cnt의 크기로 생성된 bitmap |
| **Function** | bit\_cnt의 크기로 메모리가 할당된 block\_size크기의 block을 이용해 bitmap을 생성하고 모두 false로 초기화한 후 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt) |
| **Parameter** | Bitmap의 size가 될 bit\_cnt |
| **Return** | bit\_cnt의 크기로 bitmap을 생성하기 위해 필요한 byte수 |
| **Function** | bit\_cnt의 크기로 bitmap을 생성하기 위해 필요한 byte수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | 제거할 bitmap b |
| **Return** | None |
| **Function** | b의 모든 bit의 메모리를 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | size를 확인할 bitmap b |
| **Return** | b의 크기 |
| **Function** | b의 bit\_cnt를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | target bitmap b, target index idx, set해줄 truth value |
| **Return** | None |
| **Function** | b의 idx번째 bit를 value와 같은 truth value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | target bitmap b, target index bit\_idx |
| **Return** | None |
| **Function** | b의 bit\_idx번째 bit를 true로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | target bitmap b, target index bit\_idx |
| **Return** | None |
| **Function** | b의 bit\_idx번째 bit를 false로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | target bitmap b, target index bit\_idx |
| **Return** | None |
| **Function** | b의 bit\_idx번째 bit가 true면 false로 false면 true로 flip한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*b, size\_t idx) |
| **Parameter** | target bitmap b, target index idx |
| **Return** | b의 idx번째 truth value |
| **Function** | b의 idx번째 bit의 truth value를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*b, bool value) |
| **Parameter** | target bitmap b, set할 truth value |
| **Return** | None |
| **Function** | b의 모든 bit을 value와 같은 truth value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt, set할 truth value |
| **Return** | None |
| **Function** | b의 start번째 bit부터 cnt개의 bit를 value와 같은 truth value로 set한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt, 확인할 truth value |
| **Return** | b의 start번째 bit부터 cnt개의 bit 중 value와 truth value가 같은 bit의 개수 |
| **Function** | b의 start번째 bit부터 cnt개의 bit 중 value와 truth value가 같은 bit의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt, 확인할 truth value |
| **Return** | b의 start번째 bit부터 cnt개의 bit 중 value와 truth value가 같은 bit이 존재하는지 여부 |
| **Function** | b의 start번째 bit부터 cnt개의 bit 중 value와 truth value가 같은 bit이 하나라도 존재한다면 true, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt |
| **Return** | b의 start번째 bit부터 cnt개의 bit의 truth value가 true인 bit이 존재하는지 여부 |
| **Function** | b의 start번째 bit부터 cnt개의 bit의 truth value가 true인 bit이 하나라도 존재한다면 true, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt |
| **Return** | b의 start번째 bit부터 cnt개의 bit의 truth value가 모두 false인지 여부 |
| **Function** | b의 start번째 bit부터 cnt개의 bit의 truth value가 모두 false라면 true를, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt |
| **Return** | b의 start번째 bit부터 cnt개의 bit의 truth value가 모두 true인지 여부 |
| **Function** | b의 start번째 bit부터 cnt개의 bit의 truth value가 모두 true라면 true를, 아니라면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt, 확인할 truth value |
| **Return** | b의 start번째 bit부터 cnt개의 bit 중 truth value가 value가 같은 경우 그 첫번째 bit의 index |
| **Function** | b의 start번째 bit부터 cnt개의 bit 중 truth value가 value와 같은 bit이 있는 경우 그 첫번째 bit의 index를 반환하고, 없는 경우 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | target bitmap b, 범위를 나타낼 start, cnt, 확인할 truth value |
| **Return** | b의 start번째 bit부터 cnt개의 bit 중 truth value가 value가 같은 경우 그 첫번째 bit의 index |
| **Function** | b의 start번째 bit부터 cnt개의 bit 중 truth value가 value와 같은 bit이 있는 경우 그 첫번째 bit의 index를 반환하고, 없는 경우 BITMAP\_ERROR를 반환한다. 또한 b의 start번째 bit부터 cnt개의 bit를 모두 flip한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*b) |
| **Parameter** | target bitmap b |
| **Return** | None |
| **Function** | b의 bit을 hexadecimal로 표현한 값을 출력한다. |