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

|  |  |
| --- | --- |
| 담당 교수 : | 문의현 교수님 |
| 학번 : | 20161385 |
| 이름 : | 고예성 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b); |
| **Parameter** | struct list\_elem \*a, struct list\_elem \*b |
| **Return** | void |
| **Function** | list의 node에 저장되어 있는 data를 swap하는 함수이다. list elem의 pointer type의 두 원소를 parameter로 받는다. 받은 두 포인터를 list\_entry 매크로를 통해 data를 얻고 그 두 데이터를 swap한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | 특정 list를 받아서 node의 데이터를 무작위로 셔플한다. 무작위 셔플을 위해 stdlib.h에 정의되어 있는 rand함수를 사용하여 list에 무작위 index를 부여했다. 또 srand함수에 time.h에 정의된 time함수를 argument로 넣어 시간에 따라 seed가 바뀌게 하였다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i); |
| **Parameter** | int i |
| **Return** | unsigned |
| **Function** | hashTable에 저장할 data를 가지고 저장할 위치를 알고리즘을 통해 만들어 낸다. 이 함수에서는 doubleTmp에 실수 값을 i와 곱하고 그 값을 int로 변환 한 후 여러 처리를 거쳐서 hash\_bytes에 넣어서 해싱 값을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size); |
| **Parameter** | struct bitmap \*bitmap, int size |
| **Return** | struct bitmap \* |
| **Function** | 이미 생성되어 있는 bitmap자료구조를 확장할 때 사용한다. 기존의 bitmap이 가지고 있던 size와 확장하려는 size 더한 후에 새로운 bitmap자료구조를 만들고 기존의 자료구조에 있던 데이터를 복사한다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size); |
| **Parameter** | struct bitmap \*bitmap, int size |
| **Return** | struct bitmap \* |
| **Function** | 이미 생성되어 있는 bitmap자료구조를 확장할 때 사용한다. 기존의 bitmap이 가지고 있던 size와 확장하려는 size 더한 후에 새로운 bitmap자료구조를 만들고 기존의 자료구조에 있던 데이터를 복사한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | list 타입의 구조체를 초기화한다.  이 구조체는 list의 head와 tail을 담고 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list의 head.next에 접근해서 제일 처음 원소를 list\_elem \*로 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list \*list를 parameter로 받는다. list의 tail에 접근한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list의 tail.prev에 접근해서 제일 마지막 원소를 list\_elem \*로 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev(struct list\_elem \*elem); |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem \* |
| **Function** | struct list\_elem \*elem을 parameter로 갖고 elem의 이전 node에 접근할 수 있도록 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list의 head를 list\_elem \*로 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list의 head를 list\_elem \*로 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | struct list의 tail을 list\_elem \*로 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert(struct list\_elem \*before, struct list\_elem \*elem); |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*elem |
| **Return** | void |
| **Function** | 이전 노드와 새로 삽입할 노드를 parameter로 받는다. 그 다음 두 노드를 서로 연결한다. |

|  |  |
| --- | --- |
| **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** | 특정 list에 다른 list의 원소를 삽입한다. 이때 first의 위치부터 last 이전의 우치까지 잘라서 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void  list\_push\_front(struct list \*list, struct list\_elem \*elem); |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | void |
| **Function** | list\_insert함수의 wrapper함수이다. list의 제일 앞에 node를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | void  list\_push\_back(struct list \*list, struct list\_elem \*elem); |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | void |
| **Function** | list\_insert함수의 wrapper함수이다. list의 맨 뒤에 node를 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_remove(struct list\_elem \*elem); |
| **Parameter** | struct list\_elem \* |
| **Return** | struct list\_elem \* |
| **Function** | struct list\_elem \*를 parameter로 받는다. 즉 list의 특정 node를 받아서  그 node를 list에서 삭제하고 이전 node와 이후 node를 서로 연결한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_pop\_front(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | list의 맨 앞에 있는 node를 제거한다. list\_remove의 wrapper함수이다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_pop\_back(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | list의 맨 뒤에 있는 node를 제거한다. list\_remove의 wrapper함수이다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_front(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | list의 맨 앞에 있는 node를 가리킨다. list의 head.next에 접근한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_back(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | struct list\_elem \* |
| **Function** | list의 맨 뒤에 있는 node를 가리킨다. list의 tail.prev에 접근한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t  list\_size(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | size\_t |
| **Function** | list의 노드의 개수를 센다. 노드를 순차적으로 접근하는데 이때 list\_begin과 list\_end, list\_next함수를 통해 노드에 접근한다. |

|  |  |
| --- | --- |
| **Prototype** | bool  list\_empty(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | bool |
| **Function** | list\_begin과 list\_end의 값이 같은지 다른지를 통해 list가 비어있는지를 체크한다. 만일 둘이 같다면 list가 비어있는 것이다. |

|  |  |
| --- | --- |
| **Prototype** | void  list\_reverse(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | list의 순서를 역으로 만든다. tail에 있던 것을 head에서 시작하게 된다. 내부의 swap함수를 통해 구현된다. |

|  |  |
| --- | --- |
| **Prototype** | void  list\_sort(struct list \*list, list\_less\_func \*less, void \*aux); |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | void |
| **Function** | list의 순서를 정렬한다. 이때 list\_less\_func에 등록한 함수에 의해 대소비교가 진행된다. |

|  |  |
| --- | --- |
| **Prototype** | void  list\_insert\_ordered(struct list \*list, struct list\_elem \*elem,  list\_less\_func \*less, void \*aux); |
| **Parameter** | struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux |
| **Return** | void |
| **Function** | list에 노드가 대소비교를 통해 삽입된다. 비교는 less함수에 의해 진행된다.  만일 list에 이 함수만 쓰게 되면 오름차순/내림차순으로 정렬되어 있다. |

|  |  |
| --- | --- |
| **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** | list에서 중복되는 노드를 삭제하여 다른 노드에 삽입하거나 그냥 삭제만 한다. duplicates가 있냐 없냐에 따라 결과가 구분된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_max(struct list \*list, list\_less\_func \*less, void \*aux); |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | struct list\_elem \* |
| **Function** | list에서 노드 중에 가장 큰 값을 가진 노드를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_min(struct list \*list, list\_less\_func \*less, void \*aux); |
| **Parameter** | struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | struct list\_elem \* |
| **Function** | list에서 노드 중에 가장 작은 값을 가진 노드를 리턴한다. |

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** | hashtable 자료구조를 초기화한다. hash\_hash\_func \*hash는 해싱을 수행하는 hash\_function을 등록하는 것이고 hash\_less\_func은 대소비교함수를 등록하는 것이다. 그외의 bucket의 수, 총 노드의 개수 bucket의 동적할당 등을 수행한다. |

|  |  |
| --- | --- |
| **Prototype** | void  hash\_clear(struct hash \*h, hash\_action\_func \*destructor); |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | hash\_clean은 hashtable에 있는 모든 노드를 제거한다.  hash\_action\_func에는 hash에 있는 원소를 제거하는 함수를 등록한다.  단, 원소만 제거할 뿐 동적할당을 해제하지는 않는다.(bucket 삭제X) |

|  |  |
| --- | --- |
| **Prototype** | void  hash\_destroy (struct hash \*h, hash\_action\_func \*destructor); |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | hash\_destroy는 hashtable에 있는 모든 노드를 제거하고 bucket의 동적할당 마저 해제한다. 즉, 아예 hashtable자체를 지워버리는 것이다.  hash\_action\_func에는 hash에 있는 원소를 제거하는 함수를 등록한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*  hash\_replace (struct hash \*h, hash\_elem \*new); |
| **Parameter** | struct hash \*h, hash\_elem \*new |
| **Return** | struct hash\_elem |
| **Function** | 기존에 노드가 중복된 데이터를 가지고 있다면 새로 삽입되는 것으로 교체하고 만일 새로운 데이터라면 그냥 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*  hash\_find (struct hash \*h, struct hash\_elem \*e); |
| **Parameter** | struct hash \*h, struct hash\_elem \*e |
| **Return** | struct hash\_elem \* |
| **Function** | hashtable의 노드 중에 찾고자하는 데이터를 가진 노드가 있는지 확인한다. |

|  |  |
| --- | --- |
| **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** | hashtable의 노드 중에 삭제하려는 데이터를 가진 노드가 있는지 확인하고  확인이 된 경우 노드를 삭제한다. |

|  |  |
| --- | --- |
| **Prototype** | void  hash\_apply (struct hash \*h, hash\_action\_func \*action); |
| **Parameter** | struct hash \*h, hash\_action\_func \*action |
| **Return** | void |
| **Function** | hash\_apply는 hashtable이 가진 모든 노드의 데이터 값에 hash\_action\_func을 적용하는 함수이다. 예를 들어 hash\_action\_func에 데이터의 값을 제곱하는 함수를 집어넣으면 hashtable 안의 모든 노드의 데이터가 제곱된다. |

|  |  |
| --- | --- |
| **Prototype** | void  hash\_first (struct hash\_iterator \*i, struct hash \*h); |
| **Parameter** | struct hash\_iterator \*i, struct hash \*h |
| **Return** | void |
| **Function** | hashtable의 가장 첫 번째 노드에 iterator가 접근하게 한다. |

|  |  |
| --- | --- |
| **Prototype** | void  hash\_next (struct hash\_iterator \*i); |
| **Parameter** | struct hash\_iterator \*i, struct hash \*h |
| **Return** | void |
| **Function** | hashtable의 기존 iterator가 가리키던 노드에서  그 다음 노드로 iterator가 접근하게 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*  hash\_cur (struct hash\_iterator \*i); |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | struct hash\_elem \* |
| **Function** | hashtable의 iterator가 가리키는 노드에서  elem 맴버에 접근하도록 리턴해준다. |

|  |  |
| --- | --- |
| **Prototype** | bool  hash\_empty (struct hash \*h); |
| **Parameter** | struct hash \*h |
| **Return** | bool |
| **Function** | hashtable의 노드가 있는지 없는지 확인한다. elem\_cnt에 접근해서 확인한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned  hash\_int (int i); |
| **Parameter** | int i |
| **Return** | unsigned |
| **Function** | int 값이 데이터로 주어졌을 때 해싱을 하는 함수로 내부에 hash\_bytes함수를 가지고 있어 hash\_bytes 내부 알고리즘에 의해 해싱된 값을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned  hash\_bytes (const coid \*buf\_, size\_t size); |
| **Parameter** | const coid \*buf\_, size\_t size |
| **Return** | unsigned |
| **Function** | 해싱을 진행하는 알고리즘을 가진 함수로 parameter로 해싱을 위한 데이터의 주소값과 데이터변수의 바이트 값을 받는다. 또 특별한 FNV\_32\_PRIME이라는 값을 가지고 계산을 해서 리턴한다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*  bitmap\_create(size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt |
| **Return** | struct bitmap \* |
| **Function** | bitmap을 초기화한다. bit\_cnt만큼의 비트를 초기화한다.  초기화할 때 기본 값은 0(false)로 설정된다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*  bitmap\_size(const struct bitmap \*b); |
| **Parameter** | const struct bitmap \*b |
| **Return** | struct bitmap \* |
| **Function** | bitmap의 크기를 리턴한다. bit\_cnt의 값을 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | void  bitmap\_set(struct bitmap \*b, size\_t idx, bool value); |
| **Parameter** | struct bitmap \*b, size\_t idx, bool value |
| **Return** | void |
| **Function** | bitmap의 특정 인덱스에 해당하는 비트를 value로 세팅한다.  이때 함수 내부에는 bitmap\_mark와 bitmap\_reset이 value가  true/false에 따라 작동하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void  bitmap\_mark(struct bitmap \*b, size\_t bit\_idx); |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | void |
| **Function** | bitmap의 특정 인덱스에 해당하는 비트를 true로 세팅한다. |

|  |  |
| --- | --- |
| **Prototype** | void  bitmap\_reset(struct bitmap \*b, size\_t bit\_idx); |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | void |
| **Function** | bitmap의 인덱스에 해당하는 비트를 false로 세팅한다. |

|  |  |
| --- | --- |
| **Prototype** | void  bitmap\_flip(struct bitmap \*b, size\_t bit\_idx); |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | void |
| **Function** | bitmap의 인덱스에 해당하는 비트를 true는 false로 false는 true로 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | bool  bitmap\_test(struct bitmap \*b, size\_t bit\_idx); |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | bool |
| **Function** | bitmap의 인덱스에 해당하는 비트가 true이면 true, false이면 false를 리턴 |

|  |  |
| --- | --- |
| **Prototype** | bool  bitmap\_set\_all(struct bitmap \*b, bool value); |
| **Parameter** | struct bitmap \*b, bool value |
| **Return** | bool |
| **Function** | bitmap의 모든 비트값을 value값으로 세팅한다. |

|  |  |
| --- | --- |
| **Prototype** | bool  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** | bool |
| **Function** | bitmap에서 start부터 시작되는 인덱스에서 cnt만큼의 개수를  value값으로 세팅한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t  bitmap\_count(const struct bitmap \*b, size\_t start, size\_t cnt, bool value); |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | bitmap에서 start부터 시작되는 인덱스에서 cnt만큼의 개수를 체크해서  value값이 몇개인지 체크해서 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t  bitmap\_contains(const struct bitmap \*b, size\_t start, size\_t cnt, bool value); |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | bitmap에서 start부터 시작되는 인덱스에서 cnt만큼의 개수를 체크해서  value값과 같은 것이 하나라도 있으면 true를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | bool  bitmap\_any(const struct bitmap \*b, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | bitmap에서 범위 내에 true값이 하나라도 있으면 true를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | bool  bitmap\_none(const struct bitmap \*b, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | bitmap에서 범위내에 true값이 하나도 없다면 true를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | bool  bitmap\_all(const struct bitmap \*b, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | bitmap에서 범위 내에 값이 모두 true라면 true를 리턴한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t  bitmap\_scan(const struct bitmap \*b, size\_t start, size\_t cnt, bool value); |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | start index이후 연속적으로 두개의 연속된 true or false  (전달된 bool type 변수) 값이 있으면  시작 index를 반환, 없을 시 BITMAP\_ERROR값 반환 |

|  |  |
| --- | --- |
| **Prototype** | size\_t  bitmap\_scan\_and\_flip(const struct bitmap \*b, size\_t start, size\_t cnt, bool value); |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | start index 이후 연속된 true or false(전달된 bool type 변수)가 있으면  그 bit들을 뒤집고 나서 시작 index를 반환 |

|  |  |
| --- | --- |
| **Prototype** | void  bitmap\_dump(const struct bitmap \*b); |
| **Parameter** | const struct bitmap \*b |
| **Return** | void |
| **Function** | bitmap에 있는 비트를 16진수형태로 출력한다. |