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

|  |  |
| --- | --- |
| 담당 교수 : | 문의현 |
| 학번 : |  |
| 이름 : |  |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**
2. **list\_access**

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_access (struct list \*list, int index); |
| **Parameter** | struct list \*list, int index |
| **Return** | struct list\_elem \* |
| **Function** | Use index to access the list\_element and return the element, access to max\_index + 1, (or, when the list is empty,) returns list's tail (fixed for list\_insert) |

1. **list\_swap**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap (struct list\_elem \*a, struct list\_elem \*b); |
| **Parameter** | struct list\_elem \*a, struct list\_elem \*b |
| **Return** | void |
| **Function** | Swap the data of list\_item of the list\_elem, using list\_entry() |

1. **list\_shuffle**

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle (struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | Shuffle randomly using Knuth algorithm. Iterate from the first element to the (last-1) element, while randomly select an element to swap (one from next i~n elements) |

1. **list\_dumpdata**

|  |  |
| --- | --- |
| **Prototype** | void list\_dumpdata (struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | When the list is not empty, print all data of the list\_item which is in the list. Iterate from the first elem to the last. |

1. **list\_delete**

|  |  |
| --- | --- |
| **Prototype** | void list\_delete(struct list \*list); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | Delete list and free() all allocated list\_item. (pop\_front() and free() for each element) |

1. **less\_func (for comparing the elements of the list)**

|  |  |
| --- | --- |
| **Prototype** | bool less\_func (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux); |
| **Parameter** | const struct list\_elem \*a, const struct list\_elem \*b, void \*aux |
| **Return** | bool |
| **Function** | Compare the data of a with b, and return true if the data of a is less than that of b, otherwise return false. |

1. **hash\_action\_free**

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_free (struct hash\_elem \*e, void \*aux); |
| **Parameter** | struct hash\_elem \*e, void \*aux |
| **Return** | void |
| **Function** | Just free() allocated memory of the hash\_item – used as destructor in hash\_clear() and hash\_destroy(). |

1. **hash\_action\_square & hash\_action\_triple**

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_square (struct hash\_elem \*e, void \*aux);  void hash\_action\_triple (struct hash\_elem \*e, void \*aux); |
| **Parameter** | struct hash\_elem \*e, void \*aux |
| **Return** | void |
| **Function** | Just do square or triple ‘data’ of the hash\_item converted from the struct hash\_elem \*e. |

1. **hash\_less**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*A, const struct hash\_elem \*B, void \*aux) |
| **Parameter** | const struct hash\_elem \*A, const struct hash\_elem \*B, void \*aux |
| **Return** | bool |
| **Function** | Compare the value (the data of list\_item) of two hash elements A and B. Return true if A is less than B, otherwise return false. |

1. **hash\_hash**

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_hash(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | const struct hash\_elem \*e, void \*aux |
| **Return** | unsigned |
| **Function** | Return the hash value of data of hash\_item, using hash\_int() hashing function. |

1. **hash\_dumpdata**

|  |  |
| --- | --- |
| **Prototype** | void hash\_dumpdata (struct hash \*h); |
| **Parameter** | struct hash \*h |
| **Return** | void |
| **Function** | Print all the data in the hash table h. Iterate through hash table using struct hash\_iterator. |

1. **hash\_int\_2**

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2 (int); |
| **Parameter** | int |
| **Return** | unsigned |
| **Function** | Returns a hash of integer I. (manual version) Multiply i by the big prime number(FNV\_32\_PRIME) and return the result. |

1. **bitmap\_expand**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \* bitmap\_expand (struct bitmap\* b, int size); |
| **Parameter** | struct bitmap\* b, int size |
| **Return** | struct bitmap \* |
| **Function** | Resize B to be size of BIT\_CNT bits and sets newer bits to false.  Returns struct bitmap\* if success, NULL if memory allocation failed. If B != NULL and size == 0, then B will be freed, NULL be returned. (But it’s not usual in test case) |

1. **bitmap\_dumpdata**

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dumpdata (const struct bitmap \*b); |
| **Parameter** | const struct bitmap \*b |
| **Return** | void |
| **Function** | Print all data of B from start bit to end bit. Lower bit comes first. |

1. **List**
2. **list\_insert**

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*, struct list\_elem \*); |
| **Parameter** | struct list\_elem \* before, struct list\_elem \*elem |
| **Return** | void |
| **Function** | Inserts ELEM just before BEFORE. Implement the insert of doubly linked list. (ELEM should be either an interior element or a tail element.) |

1. list\_remove

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*  list\_remove (struct list\_elem \*elem); |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem \* |
| **Function** | Remove interior element from its list. Implement the delete of doubly linked list. No free() here. |

1. list\_splice

|  |  |
| --- | --- |
| **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** | Remove elements from the ‘first’ elem to the ‘last’ elem from their list, and insert them just before ‘before’ elem. (which is either an interior or a tail.) |

1. list\_init

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | Initialize the list as an empty list. It should be called before any other list function call. Link head and tail of the list. |

1. **Hash Table**
2. **hash\_init**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux |
| **Return** | bool |
| **Function** | Initialize hash table h to use hash to compute hash value, to compare hash elements using less. It should be called before any other hash function call, while elem\_cnt and bucket\_cnt are also initialized and malloc() for the bucket is called. |

1. hash\_clear

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor); |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | Remove all elements from the hash table h, and if there is a destructor (not null), do the destructor action for each popped-up element. (usually free()) |

1. hash\_find

|  |  |
| --- | --- |
| **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** | Find a hash\_element that has an equal data as the data of hash\_elem \*e (using hash\_less function), and return a pointer to the element if found, return NULL if not found. |

1. hash\_insert

|  |  |
| --- | --- |
| **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** | Insert new hash\_element to the hash table h, return NULL when there is no element such as e and e is inserted, or return a pointer to the old element, if there is already an element such as e. |

1. **Bitmap**
2. **bitmap\_create**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt |
| **Return** | struct bitmap \* |
| **Function** | Allocate and initialize new bitmap by computing allocate\_size. |

1. bitmap\_flip

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*b, size\_t bit\_idx |
| **Return** | void |
| **Function** | Toggle the bit of the index idx. true-> false, false -> true |

1. bitmap\_set\_multiple

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | struct bitmap \*b, size\_t start, size\_t cnt, bool |
| **Return** | void |
| **Function** | Set multiple bits to bool value. |

1. bitmap\_scan

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool); |
| **Parameter** | const struct bitmap \*, size\_t start, size\_t cnt, bool |
| **Return** | size\_t |
| **Function** | scan from the start by cnt, and return idx of the first occurring of bool pattern. |