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

|  |  |
| --- | --- |
| 담당 교수 : | 문의현 |
| 학번 : | 20181256 |
| 이름 : | 김도현 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*, struct list\_elem \*) |
| **Parameter** | struct list\_elem \*element1, struct list\_elem \*element2 |
| **Return** | None |
| **Function** | Swap element1 and element2 in the list |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*) |
| **Parameter** | Struct list \* list |
| **Return** | None |
| **Function** | Random shuffle contents in the list |

|  |  |
| --- | --- |
| **Prototype** | Unsigned hash\_int\_2(int i) |
| **Parameter** | Int i |
| **Return** | Hash value at i |
| **Function** | Hashing to i. (data mapping) |

|  |  |
| --- | --- |
| **Prototype** | Struct bitmap \*bitmap\_expand(struct bitmap \*, int size) |
| **Parameter** | struct bitmap \*bitmap, int size |
| **Return** | Struct bitmap \*bitmap |
| **Function** | the bitmap size back expanded (+ int size) and return that bitmap if fail then return NULL. |

|  |  |
| --- | --- |
| **Prototype** | Void hash\_action\_destroy(struct hash\_elem \*, void \*aux) |
| **Parameter** | struct hash\_elem \*element, void \*aux |
| **Return** | None |
| **Function** | Destruct the element in hashtable and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square(struct hash\_elem \*, void \* aux) |
| **Parameter** | struct hash\_elem \*element, void \*aux |
| **Return** | None |
| **Function** | Square the element in hashtable and aux is auxiliary |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple(struct hash\_elem \*, void \* aux) |
| **Parameter** | struct hash\_elem \*element, void \*aux |
| **Return** | None |
| **Function** | triple the element in hashtable and aux is auxiliary |

|  |  |
| --- | --- |
| **Prototype** | Unsigned hash\_hash(const struct hash\_elem \*, void \*aux) |
| **Parameter** | struct hash\_elem \* element, void \*aux |
| **Return** | Hash value of hashtable element |
| **Function** | By hash\_int, Hashing data to hash value. (mapping data to certain!) |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*, const struct list\_elem \*, void \*aux) |
| **Parameter** | struct list\_elem \* element1, struct list\_elem \*element2, void \*aux |
| **Return** | True or false |
| **Function** | If element1 < element2 then true else false |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*, const struct hash\_elem \*, void \*aux) |
| **Parameter** | struct hash\_elem \* element1, struct hash\_elem \*element2, void \*aux |
| **Return** | True or false |
| **Function** | If element1 < element2 then true else false |

|  |  |
| --- | --- |
| **Prototype** | Void list(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main(), if we get some input that first word is ‘list’, then some certain actions performed for list input by input. |

|  |  |
| --- | --- |
| **Prototype** | Void hash(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main(), if we get some input that first word is ‘hash’, then some certain actions performed for hash input by input. |

|  |  |
| --- | --- |
| **Prototype** | Void bitmap(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main(), if we get some input that first word is ‘bitmap’, then some certain actions performed for bitmap input by input. |

|  |  |
| --- | --- |
| **Prototype** | Void bitmap(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main(), if we get some input that first word is ‘bitmap’, then some certain actions performed for bitmap input by input. |

|  |  |
| --- | --- |
| **Prototype** | int main() |
| **Parameter** | None |
| **Return** | If no error then return 0 |
| **Function** | If we get some input, then cut the first word, for performing certain actions by first word. |

|  |  |
| --- | --- |
| **Prototype** | Void create(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main, if we get some input that first word is ‘create’, then some data structure(list, hashtable, bitmap) created input by input. |

|  |  |
| --- | --- |
| **Prototype** | Void dumpdata(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main, if we get some input that first word is ‘dumpdata’, then some data structure(list, hashtable, bitmap) in data is printed input by input. |

|  |  |
| --- | --- |
| **Prototype** | Void delete(char \* input) |
| **Parameter** | Char \* input |
| **Return** | None |
| **Function** | In main, if we get some input that first word is ‘delete, then some data structure(list, hashtable, bitmap) deleted input by input. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | Struct List \* list( that you want to initialized) |
| **Return** | None |
| **Function** | Initializes list as an empty list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin(struct list \*) |
| **Parameter** | Struct list \* list |
| **Return** | Struct list\_elem \* element |
| **Function** | Return the beginning of lsit |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*) |
| **Parameter** | Struct list \*list |
| **Return** | None |
| **Function** | Reverse elements of order in list. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*) |
| **Parameter** | struct list\* list |
| **Return** | Bool (true or false) |
| **Function** | Return the list is empty |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*) |
| **Parameter** | Struct list\* list |
| **Return** | Struct list\_elem\* elemenet |
| **Function** | Remove the front element of list and return that. If List is empty then undefined behavior. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*) |
| **Parameter** | struct list \* list, struct list\_elem \*element |
| **Return** | None |
| **Function** | Insert the element at the end of the list |

|  |  |
| --- | --- |
| **Prototype** | Void list\_push\_front (struct list \*, struct list\_elem \*) |
| **Parameter** | struct list \* list, struct list\_elem \*element |
| **Return** | None |
| **Function** | Insert the element at the start of the list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*) |
| **Parameter** | Struct list \* list |
| **Return** | Struct list\_elem \* element |
| **Function** | Remove the back element of list and return that. If list is empty then undefined behavior. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*) |
| **Parameter** | Struct list \*list |
| **Return** | Struct list\_elem\* element |
| **Function** | Return the front element of list. If list is empty then undefined behavior. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*) |
| **Parameter** | Struct list \*list |
| **Return** | Struct list\_elem\* element |
| **Function** | Return the front back of list. If list is empty then undefined behavior. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*, list\_less\_func \*, void \*aux) |
| **Parameter** | Struct list \*list, list\_less\_fun \*less, void \*aux |
| **Return** | None |
| **Function** | Sorting the list by list\_less\_fun and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*, struct list\_elem \*, list\_less\_func \*, void \*aux) |
| **Parameter** | Struct list \*list, struct list\_elem \*element, list\_less\_fun \*less, void \*aux |
| **Return** | None |
| **Function** | Inserting the element in proper position in list by list\_less\_fun and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*) |
| **Parameter** | Struct list \*list |
| **Return** | Size\_t listsize |
| **Function** | Return the size of list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux) |
| **Parameter** | Struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | Struct list\_elem \* element |
| **Function** | Return the element in list which is max value by list\_less\_fun and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*) |
| **Parameter** | Struct list \*list |
| **Return** | Struct list\_elem \* element |
| **Function** | Return the element which is list’s head. |

|  |  |
| --- | --- |
| **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** | None |
| **Function** | Removes elements (first~last) from current list, and inserts them before the element ‘before’. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates, list\_less\_func \*, void \*aux) |
| **Parameter** | Struct list \*list, struct list \* duplicates, list\_less\_func \*less, void \*aux |
| **Return** | None |
| **Function** | Iterate to list and remove all except the first in each set of adjacent elements that are same by list\_less\_fun and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux) |
| **Parameter** | Struct list \*list, list\_less\_func \*less, void \*aux |
| **Return** | Struct list\_elm \* element |
| **Function** | Return the element in list which is max value by list\_less\_fun and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*element |
| **Return** | None |
| **Function** | Insert the element before “before” |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*) |
| **Parameter** | struct list\_elem \*element1 |
| **Return** | Struct list\_elem \* element2 |
| **Function** | Return the element2 that is after(next) the element1. If element1 is last element then return list tail. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*) |
| **Parameter** | Struct list\_elem \*element |
| **Return** | Struct list\_elem \* element |
| **Function** | Remove the element at the list and return the element. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*) |
| **Parameter** | Struct list \* list |
| **Return** | Struct list\_elem \* element |
| **Function** | Return the tail of list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*) |
| **Parameter** | Struct list\_elem \* element1 |
| **Return** | Struct list\_elem \* element2 |
| **Function** | Return the element2 which is before the element1 in the list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*) |
| **Parameter** | struct list\* list |
| **Return** | Struct list\_elem \* element |
| **Function** | Return the tail of list |

1. **Hash Table**

|  |  |
| --- | --- |
| **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** | True or false |
| **Function** | Initialize hashtable h and set hash by hash\_hash\_func and compare elements using hash\_less\_func and aux is auxiliary. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int ) |
| **Parameter** | Int I |
| **Return** | Hash at int I |
| **Function** | Return Hash at int I |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*) |
| **Parameter** | struct hash \*h, hash\_action\_func \*action\_func |
| **Return** | None |
| **Function** | To some actions from action\_func at elements in hashtable h. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | None |
| **Function** | Destroy the hashtable h |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | None |
| **Function** | Remove all elements in hashtable h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | Struct hash\_elem\* element |
| **Function** | Insert the new to hashtable h or replace equal element already in h, and that will be returned. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*) |
| **Parameter** | const char \*s |
| **Return** | Hash at char \* s |
| **Function** | Return Hash at char \* s |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*, size\_t) |
| **Parameter** | const void \*buf\_, size\_t size |
| **Return** | Hash of the size(bytes) in buf |
| **Function** | Return Hash of the size(bytes) in buf |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*) |
| **Parameter** | struct hash \*h, struct hash\_elem \*element |
| **Return** | Struct hash\_elem \* element |
| **Function** | Delete the element in hashtable h and that will be returned. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | Struct hash\_elem \*element |
| **Function** | Insert the new to hashtable h and return the element |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*) |
| **Parameter** | struct hash\_iterator \*i, struct hash \*h |
| **Return** | None |
| **Function** | Initialize i to iterate hashtable h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*) |
| **Parameter** | struct hash \*h, struct hash\_elem \*element1 |
| **Return** | struct hash\_elem \* element2 |
| **Function** | Find element1 that is equal to element2 in hashtable h and it will be returned. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*) |
| **Parameter** | Hashtable \*h |
| **Return** | Size\_t size |
| **Function** | Return the size of hashtable h. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*) |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | Struct hash\_elem \*element |
| **Function** | Return the cur\_element in that hashtable iteration. Or return a null pointer if at the end of the hashtbale. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*) |
| **Parameter** | struct hash \*h |
| **Return** | True or false |
| **Function** | If hashtable h is empty then true else false. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*) |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | Struct hash\_elem \* element |
| **Function** | Advance I to next element in hash table and return that. or return null pointer if there is no element at left |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | size\_t bit\_cnt |
| **Return** | Struct bitmap\* bitmap |
| **Function** | Initialize bitmap that count is bit\_cnt and set all of bits are false and return bitmap |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*bitmap, size\_t bit\_idx |
| **Return** | None |
| **Function** | That is toggle action. If that idx in the bitmap is true then change false else true. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool ) |
| **Parameter** | struct bitmap \*bitmap, bool value |
| **Return** | None |
| **Function** | Set all bits in the bitmap to value |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*) |
| **Parameter** | struct bitmap \*bitmap |
| **Return** | Size\_t size |
| **Function** | Return the size of the bitmap |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*bitmap, size\_t bit\_idx |
| **Return** | None |
| **Function** | Set the the bit at idx that is true in the bitmap |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*) |
| **Parameter** | struct bitmap \*bitmap |
| **Return** | None |
| **Function** | Destroy the bitmap |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt |
| **Return** | True or false |
| **Function** | If ‘true’ are not contains in the bitmap between start~start+cnt then true else false. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t bit\_idx) |
| **Parameter** | struct bitmap \*bitmap, size\_t bit\_idx |
| **Return** | None |
| **Function** | Set the the bit at idx that is false in the bitmap |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool ) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt, bool value |
| **Return** | Size\_t size |
| **Function** | Return the index that is starting to the first group of cnt consecutive bits at bitmap or ‘start’ that are all set to value, toggle them reverse value. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool ) |
| **Parameter** | struct bitmap \*bitmap, size\_t idx, bool value |
| **Return** | None |
| **Function** | Set the value at idx in the bitmap |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test(const struct bitmap \*, size\_t idx) |
| **Parameter** | struct bitmap \*bitmap, size\_t idx |
| **Return** | 0 or 1(true or false) |
| **Function** | Return the value(0 or 1) at idx in the bitmap |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool ) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt, bool value |
| **Return** | True or false |
| **Function** | If value is contains in the bitmap between start~start+cnt then true else false. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool ) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt, bool value |
| **Return** | Size\_t size |
| **Function** | Return the size of bits in the bitmap between start ~ start+cnt |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*) |
| **Parameter** | const struct bitmap \*bitmap |
| **Return** | None |
| **Function** | Dump the element of bitmap |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt |
| **Return** | True or false |
| **Function** | If ‘true’ contains in the bitmap between start~start+cnt then true else false. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt, bool value |
| **Return** | Size\_t size |
| **Function** | Return the index that is starting to the first group of cnt consecutive bits at bitmap or ‘start’ that are all set to value. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool ) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt, bool value |
| **Return** | None |
| **Function** | Set bits in the bitmap starting to start from start+cnt-1 change to value. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | struct bitmap \*bitmap, size\_t start, size\_t cnt |
| **Return** | True or false |
| **Function** | If all bits are ‘true’ in the bitmap between start~start+cnt then true else false. |