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

|  |  |
| --- | --- |
| Professor : | 박성용 |
| ID : | 20181679 |
| Name : | 장서우 |
|  |  |

**Please write the document using the following forms.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | Two list elements that will be swapped |
| **Return** | None. |
| **Function** | Swap two list elements in parameters. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | List that will be shuffled |
| **Return** | None. |
| **Function** | Shuffle elements of LIST in the parameter. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i) |
| **Parameter** | Integer that will be hashed |
| **Return** | Hash value of interger I |
| **Function** | data mapping to certain hash value. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size) |
| **Parameter** | Bitmap that you want to expand and the size of it |
| **Return** | Expanded bitmap if succeed, NULL if fail |
| **Function** | Expand the given BITMAP to the SIZE (backward expansion) |

|  |  |
| --- | --- |
| **Prototype** | int find\_idx(char c, int cnt, char \*inst) |
| **Parameter** | c: list, hash, bitmap / cnt: size of c / inst: name of element |
| **Return** | index of element that has same name with inst |
| **Function** | find the index of element that has same name with inst |

|  |  |
| --- | --- |
| **Prototype** | bool list\_less(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux) |
| **Parameter** | Two list elements that you want to compare |
| **Return** | true if a data is less than b data, else false |
| **Function** | compare a data and b data |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux) |
| **Parameter** | Two hash elements that you want to compare |
| **Return** | true if a data is less than b data, else false |
| **Function** | compare a data and b data |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_hash(const struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash element that will be hashed |
| **Return** | Hash value of data of hash element e |
| **Function** | call hash\_int function to mapping data to certain hash value. |

|  |  |
| --- | --- |
| **Prototype** | void destructor(struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash element e that will be destructed |
| **Return** | None. |
| **Function** | destruct hash element e |

|  |  |
| --- | --- |
| **Prototype** | void square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash element e that data will be squared |
| **Return** | None. |
| **Function** | square the data of hash element e |

|  |  |
| --- | --- |
| **Prototype** | void triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | hash element e that data will be tripled |
| **Return** | None. |
| **Function** | triple the data of hash element e |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list\*) |
| **Parameter** | list that will be initialized |
| **Return** | None. |
| **Function** | initialize list as an empty list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin(struct list \*) |
| **Parameter** | list |
| **Return** | the first element of LIST |
| **Function** | return the beginning of LIST |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next(struct list\_elem \*elem) |
| **Parameter** | list element |
| **Return** | the element after the parameter in its list |
| **Function** | return the next element in its list |

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

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev(struct list\_elem \*) |
| **Parameter** | list element |
| **Return** | the element before the parameter in its list |
| **Function** | return the before element in its list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*list) |
| **Parameter** | list |
| **Return** | list’s head |
| **Function** | return list’s head |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*list) |
| **Parameter** | list |
| **Return** | list’s tail |
| **Function** | return list’s tail |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | new element that will be inserted and element(before) that show the insert index |
| **Return** | None. |
| **Function** | insert element just before parameter before |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | before show the insert index / first and last element will be inserted just before ‘before’ |
| **Return** | None. |
| **Function** | remove elements first to last from their current list, then inserts them just before the parameter before |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | element that will be inserted / list |
| **Return** | None. |
| **Function** | insert element at the beginning of list |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | element that wil be inserted / list |
| **Return** | None. |
| **Function** | insert element at the end of list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*elem) |
| **Parameter** | list element that will be removed |
| **Return** | removed element |
| **Function** | remove parameter from its list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*list) |
| **Parameter** | list |
| **Return** | the removed front element |
| **Function** | remove the front element from the parameter list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*list) |
| **Parameter** | list |
| **Return** | the removed back element |
| **Function** | remove the back element from the parameter list. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*list) |
| **Parameter** | list |
| **Return** | the front element |
| **Function** | return the front element in the parameter list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*list) |
| **Parameter** | list |
| **Return** | the back element |
| **Function** | return the back element in the parameter list |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*list) |
| **Parameter** | list |
| **Return** | the number of elements |
| **Function** | return the number of elements in list |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*list) |
| **Parameter** | list |
| **Return** | true if list is empty, else false |
| **Function** | check the parameter list is empty |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | list |
| **Return** | None. |
| **Function** | reverse the order of list |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | list, list\_less function |
| **Return** | None. |
| **Function** | sort list according to list less function |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux) |
| **Parameter** | list, element that will be inserted, list less function |
| **Return** | None. |
| **Function** | insert element in the proper index in list according to list less function. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \*aux) |
| **Parameter** | list and duplicates list that save duplicated value / list less function |
| **Return** | None. |
| **Function** | iterates through list and remove all but the first in each set of adjacent elements that are equal according to less function. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | list, less function |
| **Return** | the element in list with the largest value |
| **Function** | return the element in list with the largest value according to less function |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*list, list\_less\_func \*less, void \*aux) |
| **Parameter** | list, less function |
| **Return** | the element in list with the smallest value |
| **Function** | return the element in list with the smallest value according to less function |

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry (list\_elem, struct, member) |
| **Parameter** | list element and struct that list\_elem is embedded inside and member name of list\_elem |
| **Return** | struct address |
| **Function** | converts the pointer to list\_elem into a pointer to struct that list\_elem is embedded inside. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | void hash\_init (struct hash \*h, hash\_hash\_func \*hash,  hash\_less\_func \*less, void \*aux) |
| **Parameter** | hash table h, hash function(hash\_int), hash less function |
| **Return** | None. |
| **Function** | initialize hash table h and set hash function hash\_hash and comparison function hash\_less. In this project I use hash\_int function for hash function. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*h, hash\_action\_func \*action) |
| **Parameter** | hash table h, hash action function (square, triple) |
| **Return** | None. |
| **Function** | call action function for each element in hash table h in arbitrary order. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | hash table h and destructor function |
| **Return** | None. |
| **Function** | remove all the elements from h |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | hash table h and destructor function |
| **Return** | None. |
| **Function** | destroy hash table h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | hash table h and new hash element that will be inserted |
| **Return** | if equal element is already in the table return that element, else null |
| **Function** | insert new hash element into hash table h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | hash table h and new hash element that will be replaced |
| **Return** | if equal element is already in the table return that element, else null |
| **Function** | insert new hash element into hash table h, replacing any equal element already in the table, which is returned. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | hash table h and hash element e that will be found |
| **Return** | if equal element is already in the table return that element, else null |
| **Function** | find an element equal to e in hash table h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | hash table h and hash element e that will be deleteted |
| **Return** | if equal element is already in the table return that element, else null |
| **Function** | find and delete an element equal to e in hash table h |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | iterator i and hash table h |
| **Return** | None. |
| **Function** | initialize i for iterating hash table h |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*i) |
| **Parameter** | iterator i |
| **Return** | return the next element, or null pointer if no elements are left |
| **Function** | advance I to the next element in the hash table |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*i) |
| **Parameter** | iterator i |
| **Return** | the current element, or a null pointer at the end of the table |
| **Function** | return the current element in the table iteration |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*h) |
| **Parameter** | hash table h |
| **Return** | the number of elements in h |
| **Function** | return the number of elements in h |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | hash table h |
| **Return** | true if h contains no elements, false otherwise |
| **Function** | check if hash is empty |

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

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*s\_) |
| **Parameter** | string s |
| **Return** | a hash of string S |
| **Function** | return a hash of string S |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int i) |
| **Parameter** | integer I |
| **Return** | a hash of integer I |
| **Function** | return a hash of integer I |

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(hash\_elem, struct, member) |
| **Parameter** | hash element and struct that hash\_elem is embedded inside and member name of hash\_elem |
| **Return** | struct address |
| **Function** | converts pointer to hash element into a pointer to struct that hash element is embedded inside. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | bit\_cnt: bitmap bits |
| **Return** | created bitmap b |
| **Function** | initialize b to be a bitmap of bit\_cnt bits and set all of its bits to false |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*b) |
| **Parameter** | bitmap b |
| **Return** | None. |
| **Function** | destroy bitmap b |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*b) |
| **Parameter** | bitmap b |
| **Return** | the number of bits in b |
| **Function** | return the number of bits in b |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | bitmap b, index number, Boolean value (t or f) |
| **Return** | None. |
| **Function** | set the bit numbered index in b to value |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, bit index number |
| **Return** | None. |
| **Function** | set the bit numbered index in b to true |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, bit index number |
| **Return** | None. |
| **Function** | set the bit numbered index in b to false |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx) |
| **Parameter** | bitmap b, bit index number |
| **Return** | None. |
| **Function** | toggle the bit numbered index in b |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test(const struct bitmap \*b, size\_t idx) |
| **Parameter** | bitmap b, bit index number |
| **Return** | the value of the bit numbered index in b |
| **Function** | return the value of the bit numbered index in b |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*b, bool value) |
| **Parameter** | bitmap b, Boolean value (t or f) |
| **Return** | None. |
| **Function** | set all bits in b to value |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, start index, count, Boolean value (t or f) |
| **Return** | None. |
| **Function** | set the bits(size of count) starting at start in b to value |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, start index, count, Boolean value (t or f) |
| **Return** | the number of bits in b between start and start + cnt |
| **Function** | return the number of bits in b between start and start + cnt |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, start index, count, Boolean value (t or f) |
| **Return** | true if any bits in b between start and start + cnt are set to value, and false otherwise |
| **Function** | check if bitmap contains the parameter value in b between start and start+cnt |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, start index, count |
| **Return** | true if any bits in b between start and start + cnt are set to true, and false otherwise |
| **Function** | check if bitmap contains ‘true’ in b between start and start+cnt |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, start index, count |
| **Return** | true if no bits in b between start and start + cnt are set to true, and false otherwise |
| **Function** | check if no bits in b between start and start+cnt are set to true. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt) |
| **Parameter** | bitmap b, start index, count |
| **Return** | true if every bit in b between start and start + cnt are set to true, and false otherwise |
| **Function** | check if bitmap is all ‘true’ in b between start and start+cnt |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, start index, count, Boolean value (t or f) |
| **Return** | the starting index of the first group of count consecutive bits in b |
| **Function** | find and return the starting index of the first group of count consecutive bits in b or after start that are all set to value |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | bitmap b, start index, count, Boolean value (t or f) |
| **Return** | the index of the first bit in the group |
| **Function** | find and return the starting index of the first group of count consecutive bits in b or after start that are all set to value, flip them all to !value. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*b) |
| **Parameter** | bitmap b |
| **Return** | None. |
| **Function** | dump the content of b |