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

|  |  |
| --- | --- |
| 담당 교수 : | 문의현 |
| 학번 : | 20181210 |
| 이름 : | 유창호 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap \*target\_bitmap, size\_t size); |
| **Parameter** | struct bitmap \*target\_bitmap, size\_t size |
| **Return** | NULL if fail, struct bitmap\* if success |
| **Function** | Delete current bitmap and replace it with a larger one.  size\_t og\_size = bitmap\_size(target\_bitmap);  size = og\_size + size;  struct bitmap \*temp = target\_bitmap;  target\_bitmap = bitmap\_create (size);  if(target\_bitmap != NULL){  for(size\_t i = 0; i < og\_size; i++){  bitmap\_set(target\_bitmap, i, bitmap\_test(temp, i));  }  for(size\_t i = og\_size; i < size; i++){  bitmap\_set(target\_bitmap, i, false);    bitmap\_destroy(temp);  return target\_bitmap;  }  return NULL; |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | struct list\_elem \*a, struct list\_elem \*b |
| **Return** | void |
| **Function** | Using remove and insert, switch the postion of two list elements.  void list\_swap(struct list\_elem \*a, struct list\_elem \*b){  if(a != b){  struct list\_elem\* b\_right = b->next;  if(b\_right == a){  list\_remove(b);  list\_insert(a->next, b);  }  else{  list\_remove(b);  list\_insert(a->next, b);  list\_remove(a);  list\_insert(b\_right, a);  }  }  } |
| **Prototype** | void list\_shuffle(struct list\* target\_list); |
| **Parameter** | struct list \* |
| **Return** | void |
| **Function** | Use srand and time to set random seed.  Then iterate the list and swap the element itself between the iteration index and randomly generated index. Used user defined swap function to swap.  void list\_shuffle(struct list\* target\_list){  srand(time(NULL));  int si = list\_size(target\_list);  for(int i =0; i < si; i++){  int r = rand() % si;  struct list\_elem\* e;  struct list\_elem \*temp1;  struct list\_elem \*temp2;  int cnt = 0;  for (e = list\_begin (target\_list); cnt < i; e = list\_next (e)){  cnt++;  }  temp1 = e;  cnt = 0;  for (e = list\_begin (target\_list); cnt < r; e = list\_next (e)){  cnt++;  }  temp2 = e;  list\_swap(temp1, temp2);  }  } |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | struct list\_elem \* |
| **Function** | return beginning of list |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | void |
| **Function** | create new list only has head and tail |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | struct list\_elem \* |
| **Function** | return beginning of list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | struct list\_elem \* |
| **Function** | return next element of list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | struct list\_elem \* |
| **Function** | return prev element of list |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*); |
| **Parameter** | struct list \* |
| **Return** | struct list\_elem \* |
| **Function** | return the head of list |

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

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*before, struct list\_elem \*elem |
| **Return** | void |
| **Function** | insert element before the before parameter. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | void |
| **Function** | insert element at the very front |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_end (struct list \*list, struct list\_elem \*elem) |
| **Parameter** | struct list \*list, struct list\_elem \*elem |
| **Return** | void |
| **Function** | insert element at the very end |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_front (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | delete the front list element |

|  |  |
| --- | --- |
| **Prototype** | void list\_pop\_end (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | delete the end list element |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_remove (struct list\_elem \*elem) |
| **Parameter** | struct list\_elem \*elem |
| **Return** | struct list\_elem\* |
| **Function** | remove element in list |

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

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*list) |
| **Parameter** | struct list \*list |
| **Return** | void |
| **Function** | reverse the order of list |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux); |
| **Parameter** | struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux |
| **Return** | bool |
| **Function** | create hash with hash function received in parameter |

|  |  |
| --- | --- |
| **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 in hash |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | struct hash \*h, hash\_action\_func \*destructor |
| **Return** | void |
| **Function** | free memory allocated to hash |

|  |  |
| --- | --- |
| **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 hash element new to hash h. if there the bucket is already occupied, don’t insert it. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem\* hash\_replace (struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | struct hash \*h, struct hash\_elem \*new |
| **Return** | struct hash\_elem\* |
| **Function** | insert hash element new to hash h. if there the bucket is already occupied, replace with the new element. |

|  |  |
| --- | --- |
| **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 and return element in hash h |

|  |  |
| --- | --- |
| **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** | delete and return element in hash h |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*i, struct hash \*h) |
| **Parameter** | struct hash\_iterator \*i, struct hash \*h |
| **Return** | void |
| **Function** | iterate to the first element of hash |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*i) |
| **Parameter** | struct hash\_iterator \*i |
| **Return** | struct hash\_elem \* |
| **Function** | iterate to next element of hash |

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

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*h) |
| **Parameter** | struct hash \*h |
| **Return** | bool |
| **Function** | check if hash is empty |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt |
| **Return** | struct bitmap \* |
| **Function** | create bitmap |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*); |
| **Parameter** | const struct bitmap \* |
| **Return** | size\_t |
| **Function** | return bitmap size |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*); |
| **Parameter** | struct bitmap \*); |
| **Return** | void |
| **Function** | destory bitmap memory |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt |
| **Return** | size\_t |
| **Function** | return bitmap buffer size |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool); |
| **Parameter** | struct bitmap \*, size\_t idx, bool |
| **Return** | void |
| **Function** | set bitmap index to true or false according to parameter |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*, size\_t idx |
| **Return** | void |
| **Function** | mark bitmap index value as true |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*, size\_t idx |
| **Return** | void |
| **Function** | mark bitmap index value as true |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx); |
| **Parameter** | struct bitmap \*, size\_t idx |
| **Return** | void |
| **Function** | reverse bitmap index value |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx); |
| **Parameter** | const struct bitmap \*, size\_t idx |
| **Return** | bool |
| **Function** | reverse bitmap index bit |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt); |
| **Parameter** | size\_t bit\_cnt |
| **Return** | size\_t |
| **Function** | return bitmap buffer size |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool); |
| **Parameter** | (struct bitmap \*, bool |
| **Return** | void |
| **Function** | use to initalize all bitmap value to parameter |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (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** | return number of bits in bitmap |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | bool |
| **Function** | true if any bits in B between START and START + CNT,  exclusive, are set to VALUE, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | true if any bits in B between START and START + CNT,  exclusive, are set to VALUE, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | true if no bits in B between START and START + CNT,  exclusive, are set to VALUE, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt); |
| **Parameter** | const struct bitmap \*, size\_t start, size\_t cnt |
| **Return** | bool |
| **Function** | true if any bits in B between START and START + CNT,  exclusive, are set to VALUE, and false otherwise. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | const struct bitmap \*b, size\_t start, size\_t cnt, bool value |
| **Return** | size\_t |
| **Function** | return starting index of first gorup of cnt consecutive bits |