```c
/*
 * Paul Heidelman
 * Case Western Reserve University
 * EECS 337 Fall 2009
 * Homework 2
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>


typedef struct list_entry list_entry_t;

struct list_entry {
 void *key;
 void *value;
 list_entry_t *next;
};

typedef struct {
 list_entry_t *list;
 int (*cmp)(const void *, const void *); /* Comparison function */
} list_head_t;


list_head_t *list_init(int (*cmp)(const void *, const void *)){
 //assign cmp
 list_head_t *n;
 n = (list_head_t *) malloc(sizeof(list_head_t));
 if(n == NULL)
 {
  fprintf(stderr, "Error creating list_head_t\n");
  return NULL;
 }
 n->cmp = cmp;
 return n;
};

list_entry_t *list_insert(list_head_t *head, void *key, void *value){
 if(head == NULL)
 {
  fprintf(stderr, "Invalid list_head_t\n");
  return NULL;
 }
 if(key == NULL)
 {
  fprintf(stderr, "Invalid key\n");
  return NULL;
 }

 list_entry_t *new_entry;
 new_entry = (list_entry_t *) malloc(sizeof(list_entry_t));
 new_entry->key = key;
 new_entry->value = value;
 list_entry_t *n;
 n = (list_entry_t *) malloc(sizeof(list_entry_t));

 if(head->list == NULL)
 {//then the list is empty, so add a first entry
  //printf("new");
  head->list = new_entry;
 }
 else
 {
  //printf("adding");
```

```
  n = head->list;
  while(n->next != NULL)
   n = n->next;
  n->next = new_entry;
 }
 return new_entry;
};

void *list_search(list_head_t *head, void *key){
 if(head == NULL)
 {
  fprintf(stderr, "Invalid list_head_t\n");
  return NULL;
 }
 if(key == NULL)
 {
  fprintf(stderr, "Invalid key\n");
  return NULL;
 }
 if(head->list == NULL)
 {
  fprintf(stderr, "Search: list is empty!\n");
  return NULL;
 }

 list_entry_t *n;
 n = head->list;
 while(n != NULL)
 {
  if(head->cmp(n->key, key) == 0)
  {
   //printf("found!");
   return n->key;
  }
  else
  {
   //printf("key %s is not %s\n",key, n->key);
   n = n->next;
  }
 }
 return NULL;
};

void list_delete(list_head_t *head){
 assert(head != NULL);
 list_entry_t *temp1;
 temp1 = (list_entry_t *) malloc(sizeof(list_entry_t));
 temp1 = head->list;
 while (temp1 != NULL)
 {
  list_entry_t *temp2;
  temp2 = (list_entry_t *) malloc(sizeof(list_entry_t));
  temp2 = temp1->next;
  free(temp1);
  temp1 = temp2;
 }
/*
 list_entry_t *o;
 o = (list_entry_t *) malloc(sizeof(list_entry_t));
 list_entry_t *p;
 p = (list_entry_t *) malloc(sizeof(list_entry_t));
 o = head->list;
 free(head);
 while(o != NULL)
 {
  //list_entry_t *p = n->next;
  p = o->next;
```

```
  free(o);
  o = p;
 }
 free(p);
*/
 return;
};

int cmp_double(double *a, double *b){
 if(*a == *b)
  return 0;
 else
  return 1;
};

int cmp_string(const void *a, const void *b){
 assert(a != NULL);
 assert(b != NULL);
 return strcmp(a, b);
};
```