```c
/*
 * Paul Heidelman
 * Case Western Reserve University
 * EECS 337 Fall 2009
 * Homework 1
 * Modified for 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;
 }
 if(value == NULL)
 {
  fprintf(stderr, "List is empty!\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, "List is empty!\n");
  return NULL;
 }

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

int main(){

 /*
 insert(1,2)
 insert(3,4)
 insert(5,6)
 search(1)
 search(3)
 search(5)
 insert(3,7)
 search(8)
 search(3)
 */

 printf("Creating double_list...\n");
 list_head_t *double_list;
 double_list = list_init(cmp_double);

 double a = 1;
 double b = 2;
 list_insert(double_list, &a, &b);

 double c = 3;
 double d = 4;
 list_insert(double_list, &c, &d);

 double e = 5;
 double f = 6;
 list_insert(double_list, &e, &f);

 double s = 1;
 double *result;
 result = list_search(double_list, &s);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
  printf("not found\n");

 s = 3;
 result = list_search(double_list, &s);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
```

```
 printf("not found\n");

s = 5;
result = list_search(double_list, &s);
if(result != NULL)
 printf("found: %lf\n", *result);

else
 printf("not found\n");

double g = 3;
double h = 7;
list_insert(double_list, &g, &h);

s = 8;
result = list_search(double_list, &s);
if(result != NULL)
 printf("found: %lf\n", *result);

else
 printf("not found\n");


s = 3;
result = list_search(double_list, &s);
if(result != NULL)
 printf("found: %lf\n", *result);

else
 printf("not found\n");

list_delete(double_list);



/*
 *insert("one",2)
 *insert(two",4)
 *insert(three,6)
 *search(one)
 *search(three)
 *search(five)
 *insert(three,7)
 *search(eight)
 *search(three)
 */

printf("\n\nCreating string_list...\n");
list_head_t *string_list;
string_list = list_init(cmp_string);

char str1[] = "one";
double dbl1 = 2;
list_insert(string_list, &str1, &dbl1);

char str2[] = "two";
double dbl2 = 4;
list_insert(string_list, &str2, &dbl2);

char str3[] = "three";
double dbl3 = 6;
list_insert(string_list, &str3, &dbl3);

char search_str[] = "one";
result = list_search(string_list, &search_str);
if(result != NULL)
 printf("found: %lf\n", *result);
```

```
 else
  printf("not found\n");

 strcpy(search_str, "three");
 result = list_search(string_list, &search_str);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
  printf("not found\n");

 strcpy(search_str, "five");
 result = list_search(string_list, &search_str);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
  printf("not found\n");

 char str4[] = "three";
 double dbl4 = 7;
 list_insert(string_list, &str4, &dbl4);

 strcpy(search_str, "eight");
 result = list_search(string_list, &search_str);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
  printf("not found\n");

 strcpy(search_str, "three");
 result = list_search(string_list, &search_str);
 if(result != NULL)
  printf("found: %lf\n", *result);

 else
  printf("not found\n");


 list_delete(string_list);
 return 0;
};
```