EECS 337

Compiler Design

2009 Fall Semester

Homework 1

Revision 2; August 28, 2009

Due in class, September 15, 2009

Objectives: The objectives of the first assignment are

- Learn the C programming language
- Code linked lists in C

In the next assignments, you will reuse the functions that you will develop in this first assignment.

Java-to-C tutorial: For your reference, you may want to consult the following tutorial: http://www.cs.brown.edu/courses/cs123/javatoc.htm

Pointers: The following function is an introduction to pointers and will be useful in the rest of the assignment.

Write a function that implements `int cmp_string(const void *a, const void *b);` where a and b are pointers to char (cast into pointers to void) and that returns 0 if *a=*b and a non-zero value otherwise (*hint:* the `cmp_string` function can call any function in `string.h`). Further, `cmp_string` should handle errors somewhat gracefully (*hint*: use the `assert` function in `assert.h`). Test the correctness of your function with a main program that invokes `cmp_string` with the following values of *a and *b and checks `cmp_string`'s return value:

*a	*b	Expected return value
"eecs"	"eecs"	0
"eecs"	"EECS"	Non-zero
"eecs"	"eec"	Non-zero
"eecs"	b=NULL	Error handling

Linked Lists: Write functions for linked lists that support insertions and search.

Define the types:

```
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;
```

The comparison function `cmp` is used to compare the searched key with the keys of the elements that are already present in the list.

Write the following functions:

- `list_head_t *list_init(int (*cmp)(const void *, const void *));` which allocates a list head, assigns the comparison function, and returns the list head. The new list should be empty (i.e., contain no element). If an error occurs, the `list_init` function returns NULL.
- `int list_insert(list_head_t *, void *key, void *value);` which adds a key and its corresponding value to the list. The insert function does *not* check whether the key is already present in the list. It returns 0 if all went well or -1 in case of error.
- `void *list_search(list_head_t *, void *key);` search the list for an element whose key matches the second argument. The function returns the value of the element in the list, or NULL if the key was not found.
- `void list_delete(list_head_t *);` de-allocates the list head as well as all of the list entries in it.

Now, test your code:

- Test your code on a list whose keys and values are doubles. Your test code should execute the following operations and print out the item returned by `list_search` (if `list_search` returns NULL, your code should print out a warning message): insert(1,2), insert(3,4), insert(5,6), search(1), search(3), search(5), insert(3,7), search(8), search(3).
- Test your code on a list whose keys are strings and values are doubles. Your test code should execute the following operations and print out the item returned by `list_search` (if `list_search` returns NULL, your code should print out a warning message): insert(“one”,2), insert(“two”, 4), insert(“three”,6), search(“one”), search(“three”), search(“five”), insert(“three”,7), search(“eight”), search(“three”).

Hash Table (extra credit) Design and implement a hash table with separate chaining. Use or modify the linked list implementation to hold elements in the hash table. Turn in your design, source code, instruction to compile and run the code, test cases (design of test cases, their implementation, and their output).

Hand in:

- Instructions to compile and run all of your code
- Source code for all the code that you develop
- Program output in the test cases