Laboratory: Linked Lists

Summary:

Prerequisites: Familiarity with structs and pointers. Familiarity with lists in Scheme.

- Preparation
- Exercises
  - Exercise 1: Utility Code
  - Exercise 2: Unit Tests
  - Exercise 3: Experiments
  - Exercise 4: Membership
  - Exercise 5: Some Strange Lists
- For Those With Extra Time
  - Extra 1: Deleting Nodes
  - Extra 2: Detecting Cycles
  - Extra 3: Deletion, Revisited

Preparation

a. Create a new directory for this lab. I’d suggest Labs/LinkedLists, but you can choose whatever you’d like.

b. Copy the tarball from Examples/LinkedLists.

c. Unpack that tarball with

```
tar xvf linked-lists.tar
```

d. Review the code to ensure that you understand what’s happening.

Exercises

Exercise 1: Utility Code

a. utest.c and utest.h contain some macros and functions for unit testing. Explain the purpose of those macros and functions to a partner. (Take turns: One person explains the first function or macro, the next person explains the next, and so on and so forth.)

b. expt.c and expt.h contain some macros and functions for conducting experiments. Explain the purpose of those macros and functions.
**Exercise 2: Unit Tests**

a. Review the body of `node.c` and determine what unit tests are conducted. (Look for the section titled "Unit Tests").

b. Check to confirm that the tests succeed.

c. Design your own set of five or so unit tests.

**Exercise 3: Experiments**

a. Review the body of `node.c` and determine what experiments are conducted. Predict the output of `node.expt`.

b. Check your answer experimentally.

c. Make notes of any strange things you observed.

**Exercise 4: Membership**

Consider the following function and stub (incorrect implementation).

```c
/**
 * Procedure:
 *   np_contains
 * Parameters:
 *   lst, a pointer to a node (possibly a null pointer)
 *   str, a string
 * Purpose:
 *   Determine if lst appears in string.
 * Produces:
 *   in_list, a Boolean value
 * Preconditions:
 *   [No additional]
 * Postconditions:
 *   If lst is NULL, then in_list is false (0).
 *   If list is not NULL and str is the car of lst,
 *   then in_list is true (non-zero)
 *   If list is not NULL and str is not the car of lst,
 *   then in_list holds exactly when str is in the cdr of lst.
 */
int np_contains (struct node *lst, char *str)
{
  return 0;
} // np_contains
```

a. Write four basic unit tests for `np_contains`. You might check the empty list, a list in which the string is equal to the car of the list, a longer list that contains the string as the 4th element, and a long list that does not contain the string.
b. Write an implementation of np.Contains that passes your unit tests.

**Exercise 5: Some Strange Lists**

a. Consider the following steps.

```c
struct node *lst3;
lst3 = cons ("hello", NULL);
set_cdr (lst3, lst3);
```

Draw a picture of `lst3`.

b. What do you expect to happen if you print out `lst3`?

c. Check your answer experimentally.

d. What do you expect the result of the following to be?

```c
fprintf (stderr, "Starting search ...
");
int in_list = np_contains (lst3, "Greetings");
fprintf (stderr, "Result: %d\n", in_list);
```

e. Consider the following steps, which attempt to replicate the previous example in a slightly different form.

```c
struct node *lst4;
lst4 = cons ("goodbye", lst4);
```

Draw a picture of `lst4`.

f. What do you expect to happen if you print out `lst4`?

g. Check your answer experimentally.

**For Those With Extra Time**

**Extra 1: Deleting Nodes**

Sketch an algorithm that takes a list and a string as input and deletes from that list all nodes that contain the given string.

**Extra 2: Detecting Cycles**

As you may have noted from an exercise above, our current design allows cyclic lists, which are somewhat dangerous. Sketch a way that we might be able to determine whether or not a list has a cycle.
Extra 3: Deletion, Revisited

Consider the following header for np_delete.

```c
/**
* Procedure:
*   np_delete
* Parameters:
*   lst, a pointer to a node (possibly NULL)
*   str, a string
* Purpose:
*   Delete all instances of str from lst.
* Produces:
*   newlst, a pointer to a node
* Preconditions:
*   [No additional]
* Postconditions:
*   All copies of str have been deleted. That is,
*   ! np_contains (newlst, str)
*   No other nodes are affected. That is,
*   For all strings, s, s != str
*       np_contains (newlst, s) == np_contains (lst, s)
*/
struct node *
np_delete (struct node *lst, char *str)
{
    return lst;
} // np_delete
```

a. Write unit tests.

b. Implement np_delete using the following strategy:

- np_delete (NULL, str) => NULL
- When strcmp (car (lst), str) == 0, np_delete (lst, str) =>> np_delete (cdr (lst), str)
- When strcmp (car (lst), str) != 0, we set the cdr of lst to the result of deleting all copies of str from cdr (lst).

Copyright © 2010 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.