CSC161 2011S *Imperative Problem Solving*

Class 45: Linked Lists (2)

Held: Monday, 25 April 2011

Summary: We attempt some hands-on work with the linked list implementation.

Related Pages:

- EBoard.
- Lab: Linked Lists..
- Reading: Linked List Code (`Examples/LinkedLists`.

Notes:

- Answers to your questions on my confusing code have been posted.
- Are there questions on Assignment 8?
- Due to prereg and College committee work, I need folks to sign up for appointments. Times at http://www.cs.grinnell.edu/~rebelsky/Schedule/signup.2011S.txt. Signup via email.
- CS Picnic, Friday, 6 May 2011.
- EC for Bites to Bits Thursday. [academic]
- EC for CS Table Friday. [academic]
- *Email me any other EC opportunities.*

Overview:

- Allocating Memory with `malloc`.
- Program Structure and Strange Macros.
- Lab.

Detour: Allocating Space with `malloc`

- If we're building lists "on the fly", we need to create new nodes (pairs) (cons cells) (whatever you want to call them).
- Lists are not the only time we want to do this.
- Hence, C implementations include a standard library method to allocate memory. This method is typically called `malloc` (for "memory allocate").
- Signature: `void *malloc(int amt);`
- Meaning: Allocate this much memory.
- Usage: `struct node *pair = (struct node *) malloc (sizeof (struct node *));`

Program Structure

I've broken our sample linked list code up into parts.

- `node.h` - defines the structure and declares primary operations
- `node.c` - implements the operations
- `expt.h` - tools for experimenting with code
- `utest.h` and `utest.c` - tools for unit testing

I've chosen an "interesting" strategy for the design of `node.c`

- If you compile with the `EXPERIMENTS` flag, it builds an executable that runs a series of experiments.
- If you compile with the `UNIT_TESTS` flag and link with `utest.c` (or `utest.o`), it builds an executable that runs unit tests.
- If you compile with neither flag, it simply builds the utility code that you can then link.

To make it easier to write multi-statement macros, I've written a `BLOCK` macro. The idea is that I should be able to write

```
while (x < 10)
  DO_WHATEVER (x);
```

and have it work whether `DO_WHATEVER` is a statement, a macro that expands to a statement, or a macro that expands to a sequence of statements. To make that idea work properly, we need to make sure that the macro that expands to a sequence of statements appropriately surrounds those statements. I use `BLOCK` for clarity.

Lab

- Do the lab.
