CSC161 2011S *Imperative Problem Solving*

Class 44: Linked Lists (1)

Held: Friday, 22 April 2011

Summary: We consider linked lists, one of the more important data structures.

Related Pages:

- EBoard.
- Lab: Linked Lists..

Notes:

- Reading for Monday: Examples/LinkedLists. And yes, I do want questions! (Page to be created.)
- Assignment 8 is available.
- Warning! The lab will change between now and Monday. (The code won't, but the rest of the lab will.)
- Congratulations to RS!
- I was sorry to see so few of you at the Ginsburg talk.
- 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.
- Awesome job opportunity: Web Developer for S&B and B&S.
- EC for CS Table today. [academic]
- EC for Benji's films event Saturday at 4. [peer]
- EC for Spring Bling on Friday. [peer]
- EC for Grinnell Monologues on Friday/Saturday. [peer]
- EC for ISO Cultural evening on Saturday. When else can you watch AG dance? [peer]

Overview:

- Lists in Scheme: Implementation, Central Methods.
- Representing pairs in C.
- Detour: `malloc`.
- A simple list API.
- Lab.

Lists in Scheme

We'll start our exploration of linked lists in C by revisiting the Scheme list type.

- What is a list?
- How are lists implemented in Scheme?
- What are the central methods? (That is, what are the methods you can't do without; the ones that you can write everything else in terms of?)

Pairs in C

- At the heart of a Scheme list is the pair structure.
- In C lists, we tend to call these *nodes*.
- C lists are more likely to be homogenous.
- So, what do we need?
  - A pointer to the value.
  - A pointer to the next element.
- For a list of strings, we might write this as

```
struct node
  {
    char *element;
    struct node *next;
  }
```

Detour: Allocating Spaces

- 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 *));`

A Simple Node API

- See Examples/LinkedLists/node.h.

Lab!

- Start the lab.
- We will continue tomorrow.
