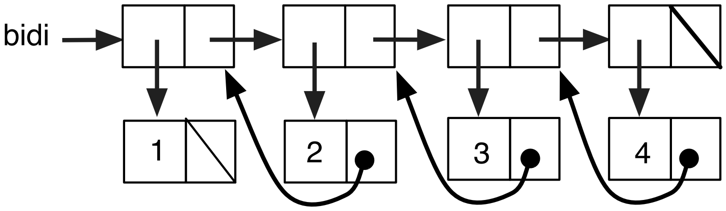
CS 136 Fall 2008
Mark Giesbrecht

Assignment 2
Due Tuesday, October 7, 2008 at 4pm.

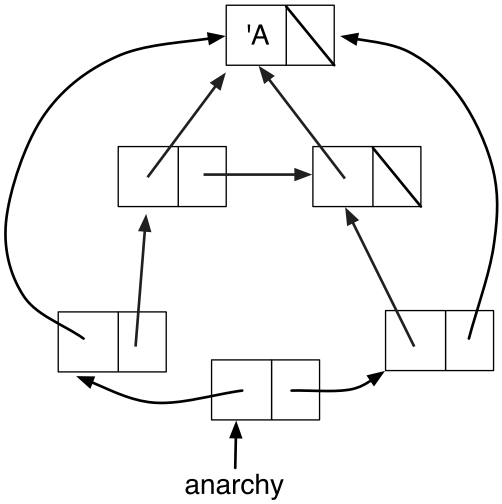
For the Scheme questions you must use the module language. Please follow the CS 135 guide for coding and commenting.

1. Write code to bind the identifiers *bidi* and *anarchy* to the *cons* structures shown in the following box-and-pointer diagrams. Create your structures using *mcons* cells.

(a)



(b)



Put your code in a file named `a2q1-bidianarchy.ss` for submission.

If you attempt to print your structures in the REPL, they will print using "shared" notation, which resembles *local* without the keyword **define**, and is described in Section 44 of the MzLib manual, available in Help Desk.

2. The first of the examples in the question above (*bidi*) is an example of a doubly-linked list. From any cell on the top layer, you can get to the previous cell and to the next cell on the top layer, as well as accessing a data element (in this case a number 1, 2, 3, or 4). In this question you are to write some functions to support doubly-linked lists.

   (a) The function *list->dlist* consumes a usual scheme list and produces a doubly linked list. For example, (*list->dlist* '(1 2 3 4)) would produce a structure equal to *bidi* above.

(b) The function *dlist->list* consumes a doubly linked list and produces a normal scheme list. For example, (*dlist->list bidi*) would produce (1 2 3 4).

(c) The function *ddata* consumes a doubly-linked list and produces the data element associated with the head of the list. For example, (*ddata bidi*) produces 1.

(d) The function *dnext* consumes a doubly linked list and produces the "next" cell in the list, or empty if there is no next cell. For example, (*ddata* (*dnext bidi*)) produces 2; (*ddata* (*dnext* (*dnext bidi*))) produces 3, etc.

(e) The function *dprev* consumes a doubly linked list and produces the "previous" cell in the list, or empty if there is no previous cell. For example, (*dprev bidi*) produces *empty*; (*ddata* (*dprev* (*dnext bidi*))) produces 1, etc.

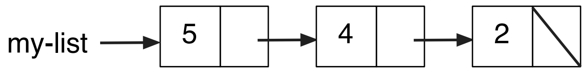
(f) The function *dlength* consumes a doubly linked list and produces the *length* of the list. This is the number of cells both previous to and after the consumed cell. For example,

(**define** *bidi3* (*dnext* (*dnext bidi*)))
(*dlength bidi*); ⇒ 4
(*dlength bidi3*); ⇒ 4

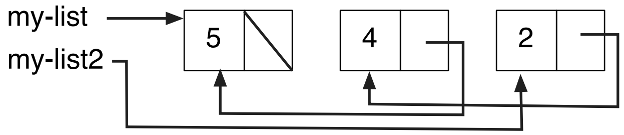
Put all your code in a file named `a2q2-dlist.ss` for submission.

3. Write the function *invert!*, which consumes an a mutable list (of mutable *mcons* cells) and reverses it using *set-mcdr!* to mutate pointers so that no new space is used (that is, the code does not use *mcons*). Here is an example of its use.

(define my-list (mlist 5 4 2))



(define my-list2 (invert! my-list))



Put your code in a file named `a2q3-invert.ss` for submission.

4. Write a function *remove-dupes!* which consumes a mutable list *ml*, and produces a list in which all but one of any duplicate elements have been removed. The original list is mutated to contain no duplicate elements. Your function must not create any new *mcons* (or *cons*) cells. For example,

(**define** *mlst* (*mlist* 'a 'b 'c 'a 'b 'd 'c))
mlst; ⇒ {a b c a b d c}
(*remove-dupes! mlst*)); ⇒ {a b c d}
mlst; ⇒ {a b c d}

Put your code in a file named `a2q4-remdupes.ss` for submission.

5. A mutable list is *cycle-free* if, for some natural number n, the result of applying *mcdr* to the list n times is *empty*. The following code, when evaluated, produces a list that is not cycle-free.

   ```
   (define mlst (mlist 3 5 4))
   (set-mcdr! (mcdr (mcdr mlst)))
   ```

   If you draw the box-and-pointer diagram of the result, it has a "cycle" in it, as the *cdr* of the last *mcons* box points to the first one. This is not the only way a list can fail to be cycle-free.

   Write a function *cycle-free?* that consumes an mutable list and produces #t if it is cycle-free, and #f otherwise. For full credit, your function must terminate on all lists and be reasonably efficient. For extra credit, it must use constant space (regardless of the size of the list) and run in time proportional to the number of unique *mcons* cells on the list.

   Put your code in a file named `a2q5-cyclefree.ss` for submission.

6. On the Assignments part of the course Web page, you will find the program `a2q6-trivial.c`. Download it, compile it using the command `gcc a2q6-trivial.c -o a2q6-trivial` and submit the resulting executable called `a2q6-trivial` (or `a2q6-trivial.exe` on Windows). Also prepare a text file `a2q6-survey.txt` in which you describe the computer, environment, and text editor you used (for example, "MacBookPro, Terminal, Vim" or "Windows, Cygwin, Nano"), and submit that as well .