Programming Languages

Lisp Implementation Project

Due Wednesday, November 14th, 2001 5:00 p.m.

Implementing Lisp

For this assignment you are going to implement a piece of a lisp interpreter -- the memory management system. You will write code that maintains a free list, allocates memory when it is called for, and does mark and sweep garbage collection.

Representing Memory

The lisp interpreter is written in C, and the heap will be represented as a C array. Each memory location will be able to store type information, and either:

- The car and cdr, if the memory location stores a list
- a string and value, if the memory location stores an atom, or
- an integer if the memory location stores a number (this is a simplified lisp interpreter that does not handle rational numbers) .

The C declarations are as follows:

```
typedef int memptr;                          Pointers are just integers --
                                              indices into the Memory array

struct memcell {
  int used;                                  Used for garbage collection
  enum {Atom, List, Number} kind;            What kind of memory location is it?
  union {
    struct {
      memptr car;
      memptr cdr;
    } list;
    struct {
      char *name;                            Name of the atom (symbol)
      int value;                             Value of atom (used for function
    } atom;                                                  parameters)
    int number;                              Value of the number
  } u;
};

struct memcell Memory[MEMSIZE];              The Heap
```

What You Need to Write

You need to write the following C functions, as defined in the file memange.h (you also will probably need to come up with some of your own helper functions)

```
/*  Function:     InitMemorySystem                        */
/*  Input Params: None                                    */
/*  Return Value: None                                    */
/*  Purpose:      This function must be called before     */
/*                any other memory functions (New,        */
/*                PrintMemory, CollectGarbage) are        */
/*                called.  This function initializes the  */
/*                freelist, and does any other necessary  */
/*                initialization                          */
void InitMemorySystem(void);
```

The function InitMemorySystem initializes the memory system. It needs to

- set up the free list
- set the SP, FP, and functionStackPointer to 0

```
/* Function:      New                                    */
/* Input Params:  None                                   */
/* Return Value:  a pointer to a free block of memory    */
/* Purpose:       This function returns a pointer to a   */
/*                free memory block (possibly doing some */
/*                garbage collection to free up some     */
/*                memory first)                          */
memptr New(void);
```

The function New finds an unused memory cell on the free list, removes it from the free list, and returns a pointer to it. New may need to call CollectGarbage if the free list is empty. If there is no memory available (even after a garbage collection step), then New should return 0. (Thus, you will never use Memory[0])

```
/* Function:      CollectGarbage                         */
/* Input Params:  none                                   */
/* Return Value:  none                                   */
/* Purpose:       This function forces a garbage         */
/*                collection                             */
void CollectGarbage(void);
```

The function CollectGarbage does mark-and-sweep garbage collection

Programming Environment

For this project, you can use any ANSI C compiler that you wish, though it needs to compile and execute correctly using the lab PCs. Though you only need to write a small piece of the code, the interpreter is fairly large (a thousand or so lines of code), so it is broken up into several files to make it more manageable.

What You Are Given

On the website for the class, you can find the following files:

- makefile — A makefile for your interpreter. If all your files are in the same directory, then the command:
  % make
  will compile all the necessary files, and create the executable **lisp**.
  The command
  % make clean
  will remove all object files, which will force a complete recompilation the next time **make** is called.
- mainloop.c — This function handles the bulk of the work for the interpreter
- hashtable.h — Header file for a standard hash table
- hashtable.c — C file for a standard hash table
- samplelisp — An executable of a complete interpreter (with the memory management system included and working, so you can see how the project is supposed to work)
- memmanage.h — Header file for the memory manager (Take a close look at this file!)
- memmangage.c — A skeleton file for you to complete. This file also has the definition of PrintMemory, which you might find useful for debugging. The function call (printmem n) prints the first n elements of the heap.

Defined Lisp Functions

The following lisp functions are defined in the mini-interpreter:

- car, cons, cdr, null, not, and, or, number, equal cond defun — standard lisp functions
- gc — run garbage collection now
- +, -, *, /, % — in this interpreter, these functions take only 2 arguments
- load — in this interpreter, path names do not work. The file to load needs to be in the same directory as the interpreter
- trace, notrace — Turns tracing on and off
- printmem — Takes an optional argument, the number of heap elements to print

What to turn in

- A hardcopy of the file memmange.c. **Be sure to add comments to all of your functions**, telling what each of the input parameters are, and what the function returns.
- Also, you need to turn in an electronic copy by copying everything your program needs to run (makefile, all .c and .h files) to your submit folder. Be sure permissions are set correctly, and do not use subfolders!