# CSE 110 Final Exam

M–J. Dominus

5 August, 1992

Due: 5 PM, 7 August 1992

Welcome to the exam.

Make sure to make it clear which parts of the things you write are the ones I should grade. Be sure to label each answer with a number of the problem it's the answer to.

The exam is take-home, open-book, and open-notes. You may consult any printed or electronic reference medium, but you may not consult other people and in particular you may not consult your classmates. So the rules are the same as for an in-class open-book test.

## 1 50 Points

Enclosed is a program. The specification for this program went something like this: "Read an arbitrarily large number of lines of input from the standard input, and then print them out again in random order; each line must be printed exactly once."

Evaluate and criticize the program at length.

You might consider some of the following: Indentation, white space, variable and function names, argument passing, information hiding, comments (too many? too few?), the algorithm used, the conciseness of the code, the correctness of the program, the attractiveness of the input and output, the use of manifest constants, formatting, dependence of the code on things that are None of Our Business, length of functions, appropriateness of control structures (did the programmer use `if`...`if` when they should have used `if`...`else`? Did they use `while` when `do`...`while` would have been better?), appropriateness of variable types.

(Please note: I wrote this list before I saw the code, so I do not necessarily expect all of these subjects to be worthy of comment.)

The code uses a function you haven't seen before:

`index` is a function whose arguments are a string `s` and a char `c`. `index` tried to find an occurrence of the character `c` in the string `s`. If it finds one, it returns a pointer to it. Otherwise, it returns the `NULL` pointer. For example, `index("CSE110", 'E')` returns a pointer to the character `'E'`.

## 2 25 Points

Suppose we have a large, complicated program which stores information in a doubly-linked list whose nodes have the following form:

```
struct node {
  int province_code;
  int beheadment_count;
  int rioting_index;
  struct node *next; /* Pointer to next node in list */
  struct node *prev; /* Pointer to previous node in list */
} ;
```

The first node in the list has a `NULL` value stored in its `prev` member; the last node in the list has a `NULL` value stored in its `next` member. There are no bogus nodes.

Write a single function, whose argument is a pointer to a node in the list, and which deletes the node from the list and frees the memory that was used to hold that node.

Don't worry about communicating back to the caller if you happen to delete the head or the tail node.

# 3 Miscellany

## 3.1 15 Points

Many beginners are daunted by the complexity of `argc` and `argv` and the details of managing command-line arguments.

Try to explain the reasons for the existing design. Defend or criticize it. If you criticize the design, try to think of a better design.

## 3.2 5 Points

Abscissa Eckert-Mauchly, a beginning programmer who knows too many things that are none of her business, is writing a function to allocate a structure of the type discussed in question 2. She says, "The struct has three int members, of two bytes each, and two pointers, which are four bytes each, for a total of fourteen bytes." Then she writes

```
newnode = malloc(14);
```

What should she have written instead, and why?

## 3.3 5 Points

After seeing your solution to the previous question, Abscissa disagrees. "My way is faster that yours," she says, "because the program doesn't have to stop and figure out the value of your argument to `malloc` every time the function is called; it gets the `14` right away without having to compute anything extra."

Is this a legitimate complaint? If it is, whose way is better? If not, why not?