Spring 2003, CS 61C, Clancy, midterm 1

Instructions: open-book, you have ~50 minutes. The times provided next to each problem are only suggestions on how to use your time.

Problem 1 (7 points, 15 minutes)

Part a

Given the following definition,

```
struct node {
  char name[12];
  int value;
};
```

what is sizeof(struct node)? Assume that the sizes of chars and ints are the same as on the 271 Soda computers.

Part b

Translate the following code to assembly language in the space that follows. Your solution should adhere to conventions described in P&H section 3.6. Comments in your code will help us understand your solution approach, and may earn you partial credit for an incorrect solution.

```
void exam1 ( struct node **to ) {
  exam2 (*to);
  (*(to-1))--;
}
```

(Provide your solution by filling in the code outline below)

```
# prolog: save information on stack if necessary
exam1:



# call exam2


# compute (*(to-1))--




# epilog: restore necessary things and return
```

Problem 2 (3 points, 10 minutes)

The following program includes the buggy swap function encountered in a pre-lecture quiz. Some students observed that this function "worked" because of values accidentally in memory:

```
#include <stdio.h>

void swap (int *a, int *b) {
  int *temp;
  *temp = *a;
  *a = *b;
  *b = *temp;
}

int main ( ) {
  int x, y, z;
  x = 2;
  y = 3;
  f ( _____________ );   /* Supply the argument(s) to f. */
  swap (&x, &y);
  printf ("The values of x and y are now %d and %d.\n", x, y);
  return 0;
}
```

In the space below, supply the definition of a function f, and supply a call to f in the blank above, that will GUARANTEE that the program will NOT "work", that is, it will crash when the uninitialized temp pointer is dereferenced. Also explain why your call guarantees that swap will crash.

Problem 3 (6 points, 12 minutes)

Write a C function named copyStrArray that, given an integer "count" and an array "strArray" that contains "count" strings, returns a pointer to a complete ("deep") copy of the array. (In Java terminology, this would be a "clone".) For example, the program segment

```
int main (int argc, char **argv) {
  char **ptr;
  ptr = copyStrArray (argc, argv);
  ...
```

would place in ptr a pointer to a copy of argv, the command-line argument structure.

You may use brackets and other array notation in your solution. You may assume that there is sufficient free memory in which to build the copied structure. Make no assumptions about the size of a pointer or a char. Include all necessary casts, and allocate only as much memory as necessary. You may use any function in the stdio, stdlib, or string libraries.

Problem 4 (3 points, 12 minutes)

Consider the storage layout below, which represents five consecutive blocks of storage using the boundary tags method of storage allocation described in the Hilfinger notes. All values are in hexadecimal.

```
101C   00000032
1020   00001808
1024   00001B1C
1028   00000031
102C   00001054
1030   00001010
1034   00000032
1038   00000FF4
103C   0000200C
1040   00000031
1044   00001050 <---- ptr
1048   00001028
```

```
104C   00000030
1050   00001030
1054   0000102C
```

On the diagram, indicate which memory locations from addresses 101C through 1054 change as a result of the call free(ptr), and specify the new contents of each modified word.

END OF EXAM