NAME:

Login name:

Precept number:

Computer Science 217
Midterm Exam
March 14, 2003
2-4PM

This test is 5 questions. Put your name on every page, and write out and sign the Honor Code pledge before turning in the test.

``I pledge my honor that I have not violated the Honor Code during this examination."

Question	Score
1	
2	
3	
4	
5	
Total	

QUESTION 1 *(20 POINTS)*

Provide the C code matching the following descriptions.

(a) Write the C definition of structure *employee* that has three components: *name (an array of 128 characters), access codes* (a pointer to an array of integers), and *dept* (a pointer to a not-yet-defined department structure).

(b) Write the C definition of structure *department* that has three components: *name* (an array of 128 characters), *employees* (a pointer to an array of pointers to employee structures, one for each employee in the department, where the number of employees is not known), and *size* (an integer that indicates how many employees are in the department).

(c) Write a header (.h) file declaring the interface to an ADT that keeps track of employees in a department. The ADT should be able to create and delete the data structure for a new department, add and remove an employee (in an existing *employee* structure) to a department, and apply a function provided by the client to each employee in the department. Be sure to use opaque pointers, macros for protecting against multiple file inclusion, and good programming style.

(d) Write an implementation for your function of part (c) that applies a function provided by the client to each employee in the department. You should assume that the *employees* field in the *department* structure points to an array (with *size* elements) of pointers to *employee* structures. Be sure to use good and robust programming style.

QUESTION 2 *(20 POINTS)*

For each variable *definition* in the following program, write a few words next to it indicating its scope, linkage, and duration. Write the output of the program in the box provided.

```
=== main.c ===

   int a = 3;
   int b = 4;

   extern void f(int b);

   int main()
   {
     int a;
     for (a = 0; a < 4; a++)
       f(a);
   }


=== f.c ===

   #include <stdio.h>

   extern int a;
   static int b;

   void f(int b)
   {
     int c = 0;
     static int d = 0;

     printf("%d %d %d %d\n", a, b, c, d);

     a++;
     b++;
     c++;
     d++;
   }
```

Program output:

```
3 0 0 0
4 1 0 1
5 2 0 2
6 3 0 3
```

QUESTION 3 *(20 POINTS)*

(a) Often programmers write their own memory allocation modules that allocate a large region of memory (e.g., several megabytes) with the standard library function *malloc*, and then provide their own *mallocX* and *freeX* functions that manage allocation of blocks of memory from that region.

Write an implementation for the fictitious library functions *malloc16* and *free16* that allocates and frees blocks of 16 bytes from a region you've allocated with *malloc* according to the following rules. Your code should call the standard library function *malloc* at most once during execution (do not worry about calling the standard library function *free*). All blocks returned by *malloc16* should contain exactly 16 bytes. The maximum number of blocks that will ever be allocated by calling *malloc16* (and not yet freed) at any time is 1000. If any previously freed block of 16 bytes is available, then *malloc16* should return one of them. Otherwise, it should return the lowest address following all previously allocated blocks. Be sure to use good and robust programming style.

Hint: you can use space in each freed block to store the address of the next available block on the list thereby creating a singly-linked list of available blocks (this is not required).

Extra credit: add a mechanism that detects when free16 has been called for a block of memory not allocated by malloc16 (this is not required).

```
/* Global variables go here */




void *malloc16(void)
{
  /* Code for malloc16 goes here */
```

```
}
```

```
void free16(void *ptr)
{
   /* Code for free16 goes here */




}
```

(b) Provide at least two reasons why *malloc16* and *free16* might be advantageous for a program that allocates/frees blocks of 16 bytes frequently and allocates/frees blocks of 1000 bytes several times less frequently.

QUESTION 4 *(20 POINTS)*

Imagine that you work for a major security firm, and you are asked by your boss to build a keypad controller that recognizes a code of digits pressed on a keypad. The keypad sensors provide the controller with two lines of input (i0 and i1), which are:

        (i1=0, i0=0) when no key is being pressed,
        (i1=0, i0=1) when the "1" key is pressed,
        (i1=1, i0=0) when the "2" key is pressed, and
        (i1=1, i0=1) when the "3" key is pressed.

Your circuit should provide one line of output (o0), which is 1 only when the most recently pressed sequence of digits matches the following code: 2 1 3. You can assume that the keypad prevents anybody from pressing the same key twice in a row.

a) Draw a state diagram for the keypad controller.

b) Draw a truth table for the keypad controller.

QUESTION 4 *(continued)*

c) Draw a digital circuit for the keypad controller.
   You may use any gates, flip flops, or components described in class.

QUESTION 5 *(20 POINTS)*

a) What are the five types of errors that can occur in a C program?
   (One short phrase per type)

b) The following function tries to print all lines from a file whose name is provided by user input. Yet, although it compiles without warnings, it does not work correctly and has several robustness problems. On the next page, rewrite the function to make it work robustly with complete error checking and efficient use of memory. For each piece of code you add or change, provide a brief comment to indicate what problem is being fixed and/or what type of error is being checked/handled. *Note: the manual page for* gets *appears on the last page of the exam.*

```c
#include <stdio.h>

void PrintFile(void)
{
   FILE *fp;
   char filename[16];
   char *buffer;

   gets(filename);
   fp = fopen(filename, "r");
   buffer = (char *) malloc(16);
   while (gets(buffer)) {
     printf(buffer);
     free(buffer);
   }
   fclose(fp);
}
```

```c
#include <stdio.h>

void PrintFile(void)
{
   /* Code for PrintFile goes here */




}
```

NAME
    gets, fgets - get a string from a stream

SYNOPSIS
    #include <stdio.h>

    char *gets (char *s);

    char *fgets (char *s, int n, FILE *stream);

DESCRIPTION
    gets reads characters from the standard input stream, stdin, into the
    array pointed to by s, until a new-line character is read or an end-of-
    file condition is encountered. The new-line character is discarded and
    the string is terminated with a null character.

    fgets reads characters from the stream into the array pointed to by s,
    until n-1 characters are read, or a new-line character is read and
    transferred to s, or an end-of-file condition is encountered. The string
    is then terminated with a null character.

SEE ALSO
    lseek(2), read(2), ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S),
    stdio(3S), ungetc(3S).

NOTES
    When using gets, if the length of an input line exceeds the size of s,
    indeterminate behavior may result.

DIAGNOSTICS
    If end-of-file is encountered and no characters have been read, no
    characters are transferred to s and a NULL pointer is returned. If a
    read error occurs, such as trying to use these functions on a file that
    has not been opened for reading, a NULL pointer is returned. Otherwise s
    is returned.