December 16, 1991 Name: ______________________

TA: ______________________

CS/CE 218 — Unix and C programming

Final Exam

Before you begin, please take a moment to look over the entire test so that you can budget your time.

On this exam, the phrase "C" refers to ANSI C. The character ' is a normal quote (single quote).

In C programs you may freely use the functions defined in the ANSI C library (see appendix B of the textbook for a summary). However, **you are limited to these functions**; if you wish to call some other function from a textbook, homework, or the course notes, you will have to copy it into your answer.

English answers will be judged on their completeness and clarity. However, you do *not* need to fill up all the empty space on the page. Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

problem	subject	points	score
1	PRIVATE macro	5	
2	Abstract data types	15	
3	Debugging	40	
4	Write a C function	40	
5	Pointers, Arrays, and Structures	45	
6	Uses of `static`	20	
7	RCS	20	
8	make	15	
Total		200	

This exam (only) counts for 20% of your final grade.

1. (5 points) A book on portable C programming recommends defining a macro named PRIVATE to be used in definitions of variables and functions that are not supposed to be visible outside of the file in which the declaration appears. For example, given the definition

```
PRIVATE double s[100];
```

the variable `s` would not be visible outside of the file in which it was declared. Give the definition of the macro `PRIVATE`.

2. (15 points) Why is it a good idea to treat the stack in the calculator that you saw in homework 6 as an abstract data type?

3. (40 points) Consider the following code.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct string { char * line; int length; } LSTRING;

LSTRING make_lstring(char *s)
/* requires: s != NULL */
{
   LSTRING temp = { NULL, 0 };

   while (*(temp.line)++ = *s++) {
        temp.length++;
   }
   return temp;
}
```

The function make_lstring is supposed to return an LSTRING containing a copy of the argument s. That is, for all strings t, after executing

```
LSTRING result = make_lstring(t);
```

the following expressions should both be true:

```
result.length == strlen(t)
strcmp(result.line, t) == 0
```

Is there anything wrong with the function? If so, write a version that fixes the errors, and describe all the errors; if not, write "no errors". (Hint: there are *no* syntax errors in the program.)

You may assume that the argument s to make_lstring is not NULL. You may assume that there is enough memory available, should you need it.

4. (40 points) Write a C function (not a complete program)

```
void fgrep_n(FILE * fp, char * s)
```

that prints on standard output a newline separated list of the line numbers of each line in the file argument that exactly matches `s`. For example, if the contents of the stream `fp` are:

```
abcd
line 2
abc
abcdX
abcd
Xabcd
```

then the call `fgrep_n(fp, "abcd")` should print on standard output:

```
1
5
```

In your solution, to read lines from `fp` you must use `fgets`.

You may assume that the string `s` is null-terminated, and that `s` does not contain a newline character. You may assume that the stream `fp` is open for reading and that no errors will occur on reading from `fp`. You may assume that the maximum length of a line in the stream `fp` is `BUFSIZ`-1 characters, including the newline character.

Besides working correctly, your function should be clear and should compile without errors or warnings.

5. This question is about structures, arrays, and pointers; it continues onto the next page. Consider the following program fragment.

```
#define NULL 0

struct node { int data[5];
              int * best;
              struct node * next;   };

struct node a = { {0, 1, 2, 3, 4}, NULL, NULL};
struct node b = { {20, 21, 22, 23, 24}, NULL, NULL };
struct node * list = &b;
b.next = &a;
list->best = &(list->data[3]);
list->next->best = &(list->next->data[0]);
```

(a) (10 points) Draw a picture, like those drawn in class, of the state of the program *after* the execution of the above code. Write the names of variables to the left of their space with a colon (e.g., `a:`). Label the fields of the structs `.data`, `.best`, and `.next`, by writing these labels underneath the space. Write in the values of the integers, and show pointers as arrows. Draw a null pointer as a big "×".

(b) (5 points) Is the expression `list->data++` legal? If not, why not? If it is legal, what is its value?

(c) (5 points) What is the value of the expression `(*(*list).next).data[2]` ?

(d) (5 points) What is the value of the expression `*(*(list->next)).best` ?

(e) (5 points) Write an assignment statment that will make the expression

```
a.next == &b
```

true, but that does *not* mention either `a` or `b`. (That is, `a.next = b` is incorrect.)

(f) Consider the expression

```
*(list->next->best++) = *++(*list).best
```

i. (5 points) What is the value of this expression?

ii. (5 points) After the execution of the above expression, what is the value of the expression `*(list->best)` ?

iii. (5 points) After the execution of the above expression, imagine executing the following code.

```
int i;
for (i = 0; i < 5; i++) {
  printf("a.data[%d] == %d; b.data[%d] == %d\n",
                i, a.data[i], i, b.data[i]);
}
```

Fill in the underlined spaces (`___`) in the following to show what would be printed on standard output by the above.

```
a.data[0] == ___; b.data[0] == ___
a.data[1] == ___; b.data[1] == ___
a.data[2] == ___; b.data[2] == ___
a.data[3] == ___; b.data[3] == ___
a.data[4] == ___; b.data[4] == ___
```

6. (20 points) The keyword `static` is used in two (2) different ways in C programs; that is, its use in some contexts has a different effect on the program than its use in other contexts. Describe the two contexts and the different effects of `static` in each.

7. (20 points) Describe two (2) advantages to using the RCS system to maintain C program files. (That is, give two reasons for using RCS instead of just relying on the Unix file system.)

8. (15 points) The following is a question about `make`. The following transcript gives the contents of the file `Makefile` and a long listing of the current working directory.

```
$ cat Makefile
CC=gcc
CFLAGS = -Wall
OBJS = main.o module.o foo.o

prog: $(OBJS)
        $(CC) -o prog $(OBJS) -lm

main.o: main.c module.h
        $(CC) -c $(CFLAGS) main.c

partb:
        make prog
        rm -f prog
        make prog
$ ls -l
total 70
-rw-r--r--   1 leavens  staff       171 Dec 12 12:56 Makefile
-rw-r--r--   1 leavens  staff        29 Dec 12 13:00 foo.c
-rw-r--r--   1 leavens  staff       111 Dec 12 12:56 foo.o
-rw-r--r--   1 leavens  staff       125 Dec 12 12:54 main.c
-rw-r--r--   1 leavens  staff       190 Dec 12 12:56 main.o
-rw-r--r--   1 leavens  staff        32 Dec 12 12:55 module.c
-rw-r--r--   1 leavens  staff        13 Dec 12 13:00 module.h
-rw-r--r--   1 leavens  staff       114 Dec 12 12:56 module.o
-rwxr-xr-x   1 leavens  staff     26772 Dec 12 12:57 prog
```

To answer the following question, use the above information and assume that no compile or link errors occur. (You may also assume that your user id is `leavens`.)

List the commands that the `make` program executes when you type to the shell the command

```
make partb
```

You should list all the output of `make`, including output from any recursive calls to `make`. The output should be in the order that `make` prints it.