Name:____

CSE 303, Winter 2006, Final Examination
16 March 2006

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 10:20.**
- You can rip apart the pages, but please write your name on each page.
- There are **100 points** total, distributed **unevenly** among **7** questions (which have multiple parts).
- When writing code, style matters, but don't worry about indentation.

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit.**
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. (**20** points) This problem has 4 parts.

   Suppose you have a large C program named `app` that includes a function `f`. You want to know if running `app` on the input `17` causes `f` to be called. For each question below, be *very specific* about how you would modify files and/or run commands and programs, and in what order.

   (a) Describe how to solve this problem by adding code to a C file.

   (b) Describe how to solve this problem without changing code but using the debugger `gdb`.

   (c) Describe how to solve this problem without changing code but using the profile `gprof`.

   Now suppose `app` also has a function `g` and you want to know if running `app 17` causes `f` to be called after `g` is called but before `g` returns. (That is, does `g` ever cause a sequence of calls that includes `f`?) You may assume `g` is called a small number of times.

   (d) Describe how to extend the *first two* approaches above to solve this modified problem.

Name:____

2. (**20** points) Consider these two C files:

a.c:

```
void f(int p);

int main(int argc, char**argv) {
  f(17);
  return 0;
}
```

b.c:

```
void f(char * p) {
  *p = ’x’;
}
```

(a) Why is the program made from `a.c` and `b.c` incorrect? What would you expect running it to do?

(b) Will `gcc -Wall -c a.c` or `gcc -Wall -c b.c` give an error or produce `a.o` and `b.o`?

(c) Will `gcc -Wall a.c b.c` give an error or produce `a.out`?

(d) How would you use a standard C coding practice (using an extra file) to avoid the problem above? Write this extra file and modified versions of `a.c` and `b.c` to explain.

Name:____

3. (**15** points) Write a `Makefile` for this scenario:

   - An application `myprog` is written in C, with all the code in `myprog.c`.
   - You wrote two test-inputs, in files `input1` and `input2`.
   - You want to run `myprog` *with profiling* on each test-input and then use `gprof`, saving the result to file `prof1` (for input `input1`) or `prof2` (for input `prof2`).
   - You have a bash script `compare` that takes as arguments two files created by `gprof` and produces an interesting summary. You want a phony `run` target that runs `compare` on `prof1` and `prof2`.

   Your `Makefile` should re-compile or re-run programs only as necessary (except the `run` target should always execute `compare`), but it should never use out-dated programs.

   Hints: You should have 4 targets. Some will need multiple commands. Some will need multiple sources.

Name:____

4. (**10** points) Suppose you are using `cvs` for a group project. You decide to move some of the code in `foo.c` to a new file `bar.c`. You update the `Makefile` appropriately.

   (a) What `cvs` command should you use before your next commit?

   (b) If you forget to do your answer to part (a), who will discover your forgetfulness and when?

Name:____

5. (**8** points)

   Suppose you want to make a library `blah` (also known as an archive, i.e., a `libblah.a` file) containing functions `f1`, `f2`, ..., `fn` that may call each other but do not call any other functions.

   (a) If you want to make sure library users never have to write `-lblah` more than once when linking, how should you organize your n functions into files? Explain.

   (b) If you want to make sure library users never have any more code in their executable than absolutely necessary, how should you organize your n functions into files? Explain.

Name:____

6. (**12** points) Here is a C program for testing a function `f` to see if it always returns `0`:

```
int f(int x, int y);
int main(int argc, char** argv) {
  if(f(0,0)!=0)
    return 1; // failure
  if(f(1,1)!=0)
    return 1; // failure
  return 0; // success
}
```

Give an example of a function `f` such that:

- The test above achieves full statement and branch coverage.
- The function does *not* always return 0.

Explain your answer.

Name:____

7. (**15** points) This problem has 3 parts.

Assume we are using reference-counting to manage the memory pointed to by `p` and `q`. Recall that with reference-counting, when we assign `q` to `p` we should write:

```
  decr_count(p); // line 1
  p = q;
  incr_count(p); // line 3
```

(a) What error could occur (later) if you forget line 1? Explain.

(b) What error could occur (later) if you forget line 3? Explain.

Suppose the definition of `incr_count` looks like this:

```
void incr_count(struct Foo * x) {
  int c = x->count;
  x->count = c + 1;
}
```

(c) If two *threads* call `incr_count` with the same pointer at the same time, what could go wrong? What would happen to the count and what error could occur later as a result?