Name:_____

CSE 303, Spring 2007, Final Examination
5 June 2007

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 4:20.**
- You can rip apart the pages, but please staple them back together before you leave.
- There are **90 points** total, distributed **unevenly** among **7** questions.
- 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 total** points) Consider an application that has these pieces:

   - An executable `prog1` built from files `a.c`, `b.c`, and `shared.h` (which both C files include)
   - A Java program built from `X.java` (which defines one class `X` holding the whole program)
   - Two text files `start` and `end`
   - A Bash script `scr`, with these contents:

```
#!/bin/bash
tmpfile=‘mktemp‘
cat start >> $tmpfile
./prog1 >> $tmpfile
java X >> $tmpfile
cat end >> $tmpfile
mv $tmpfile output
```

   (a) (**6** points) Write a Makefile with a target `run` that creates the file `output` by running the Bash script as appropriate. This target should have several dependencies. The Makefile should do the minimal amount of recompiling. (Sample solution is 10 lines.)

   (b) (**2** points) After running `make run` what *thirteen* files are definitely in the current directory?

   (c) (**6** points) After running `make run`, what are all the files you could remove and still have `scr` run properly (without any recompiling)? (Assume you never wish to recompile again.)

   (d) (**6** points) If you had a version-control (e.g., cvs) repository for this project, which files should you put in it?

Solution:

(a)
```
run: prog1 X.class start end
        scr
prog1: a.o b.o
        gcc -o $@ $^
a.o: a.c shared.h
        gcc -c $<
b.o: b.c shared.h
        gcc -c $<
X.class: X.java
        javac $<
```

(b) Makefile, prog1, a.c, b.c, shared.h, X.java, X.class, a.o, b.o, start, end, scr, output

(c) Makefile, a.o, b.o, a.c, b.c shared.h, X.java, output

(d) Makefile, a.c, b.c, shared.h, X.java, start, end, scr

Name:_____

2. (**10** points) Suppose a group project with the following:

   - Alice is writing `foo.c` and Bob is writing `bar.c`. They are also using library `libbaz.a`.
   - They are using cvs, but neither modifies the other's file nor the library.
   - Bob links his code via: `gcc -o myprog foo.o -lbaz bar.o`

   Explain how it could be that Bob's linking command works before doing a `cvs update` but not afterwards. Suggest an improved command.

   Solution:
   Suppose `bar.c` uses a function `f` defined only in a `.o` file contained in `libbaz.a`, but originally `foo.c` also uses `f` (or some other function defined in the same `.o`). Then `foo.o` will cause the linker to include the function `f`. If `foo.c` is changed to omit such calls, then the linker will not include the `.o` because of where `-lbaz` is. An improved command is `gcc -o myprog foo.o bar.o -lbaz`

Name:_____

3. (**10** points) You sold some C code to a company including a header file with these lines:

```
// init MUST be called before any calls to f1, f2, or f3
void init();
int f1(int);
int f2(int);
int f3(int);
```

They want their money back because their application keeps crashing in your functions. They give you their source code, test inputs, and Makefile to investigate the problem. You suspect that they are not calling `init` *first* even though they have calls to `init` in their code. How would you confirm your suspicions *without* changing any of the source code? Be specific about what you modify, what tool(s) you would use, what features you would use, etc.

Solution:
Use a debugger like gdb. First change the Makefile to compile all the C files with flag `-g` (though actually it would be enough just to compile your code that way). In gdb, before running the program on some test input, set breakpoints at `init`, `f1`, `f2`, and `f3` (by, for example, typing `break init`). Then run the program (by typing `run some-input`). If the first breakpoint reached is not `init` your suspicions are confirmed.

Name:_____

4. (**18 total** points) Consider this C code:

```
int is_prefix(char * str1, char * str2) {
  int i = 0;
  while(str1[i] != '\0') {
    if(str1[i] != str2[i])
      return 0;
    ++i;
  }
  return 1;
}
int either_prefix(char * str1, char * str2) {
  if(!str1 || !str2)
    return 0;
  if(strlen(str1) < strlen(str2))
    return is_prefix(str1,str2);
  else if(strlen(str2) < strlen(str1))
    return is_prefix(str2,str1);
}
```

(a) (**3** points) What is wrong with `either_prefix`? Give a test-case that reveals the error.

(b) (**3** points) Describe an easy fix so that `either_prefix` no longer has the error.

(c) (**8** points) Provide a test-suite for (the original) `either_prefix` that has full branch-coverage.

(d) (**4** points) Suppose we decided the specification of `either_prefix` required that `str1` be a non-`NULL` string with length at least 1. Show to check for this at run-time with an assertion (where and what would you add).

Solution:

(a) If passed two strings of the same length, it has no explicit return, so behavior is completely undefined. Example: `either_prefix("a","b")`

(b) Replace the last "else if" with just an "else" – the helper function works fine for two strings of the same length.

(c) (It's unclear for such illegal code if path-coverage includes a test of two strings of equal length, so full credit whether or not that is included. You do need a test where a string is `NULL`, a test where `str1` is longer and a test where `str2` is longer.) Example:

```
either_prefix(NULL,NULL);
either_prefix("a","");
either_prefix("","a");
```

Note this (rather minimal) example does not have much coverage over `is_prefix` but that wasn't the question.

(d) Put `assert(str1 != NULL && str1[0] != '\0');` (or something equivalent to that; calling `strlen` is okay) as the first statement in the function body. You also need to have `#include <assert.h>` but you did not have to say that for full credit. Note you *do* have to check for `NULL` *first*.

Name:_____

5. (**12 total** points) Suppose the following pthreads code is run on a computer with two processors. (Assume `f` is called with an argument appropriately initialized.)

```
#include <pthread.h>
int large_computation1(int); // defined elsewhere
int large_computation2(int); // defined elsewhere

struct AllData {
  int array_lengths;
  int * input_data;
  int * result_of_1;
  int * result_of_2;
};
void* do1(void * a) {
  struct AllData * d = a;
  int i;
  for(i=0; i < d->array_lengths; ++i)
    d->result_of_1[i] = large_computation1(d->input_data[i]);
  return NULL;
}
void* do2(void * a) {
  struct AllData * d = a;
  int i;
  for(i=0; i < d->array_lengths; ++i)
    d->result_of_2[i] = large_computation2(d->input_data[i]);
  return NULL;
}
void f(struct AllData * data) {
  pthread_t t1, t2;
  pthread_create(&t1,NULL,do1,data);
  pthread_join(t1,NULL);
  pthread_create(&t2,NULL,do2,data);
  pthread_join(t2,NULL);
}
```

(a) (**6** points) Why will this program run no faster than a similar one without threads? How would you change the program to try to improve performance? (Be specific about your code change.)

(b) (**6** points) Suppose your fix is some help, but the program runs about, say, 20% faster instead of twice as fast. How could this be? How could you use `gprof` to determine which computation is taking more time and what could you do to further improve performance? (An English sketch of what you would do to the code is fine.)

Solution:

(a) The second thread (for `do2`) does not start until the first thread (for `do1`) is done, so there is no parallelism. The fix is to move `pthread_join(t1,NULL)` to after the second call to `pthread_create`.

(b) `do1` and `do2` might not take close to the same amount of time because, for example, one of the large computations is larger than the other. By compiling with `-pg` and running `gprof` we can see which takes more time. We could then move some of the computation from one to the other (for example, one thread could do all of the `large_computation1` and a third of the `large_computation2` if `large_computation2` takes more time).

Name:_____

6. (**10** points) Give C++ code that could be put in place of the `/* YOUR CODE HERE */` below such that the program compiles without warning and prints out the 13 characters `Hello, World!` (and nothing else) when run.

   - You must provide 1 class definition and 2 method *definitions*. (You can *declare* more methods.)
   - Sample solution is 8 lines (but like the provided code, some methods are defined in 1 line).

```
#include <iostream>
using namespace std;

/* YOUR CODE HERE */

class D : public C {
public:
  void m1();
  void m2();
};
void D::m1() { cout << "summer "; }
void D::m2() { cout << ", World"; }

int main(int argc, char** argv) {
  C* c = new D(); // upcast
  char x = 'F';
  c->m1();
  c->m2();
  c->m3(x);
  cout << x;
  return 0;
}
```

Solution:

```
class C {
public:
  void m1();
  virtual void m2() = 0;
  void m3(char& a);
};
void C::m1() { cout << "Hello"; }
void C::m3(char& x) { x = '!'; }
```

Name:_____

7. (**10 total** points)

   (a) (**5** points) What is wrong with this C++ program? Be specific.

```
class C {
public:
  int * x;
  C(int * _x) : x(_x) {}
  ~C() {}
};
int main() {
  for(int i=0; i < 10000000; ++i) {
     C *c1 = new C(new int(42));
     C *c2 = new C(new int(42));
     delete c1;
     delete c2;
  }
}
```

   (b) (**5** points) What is wrong with this C++ program? Be specific.

```
class C {
public:
  int * x;
  C(int * _x) : x(_x) {}
  ~C() { delete x; }
};
int main() {
  for(int i=0; i < 10000000; ++i) {
     int * p = new int(42);
     C *c1 = new C(p);
     C *c2 = new C(p);
     delete c1;
     delete c2;
  }
}
```

Solution:

(a) It has space leaks; each created C object in the for-loop points to a new heap-allocated int, but the space for the ints is never reclaimed. This space is unreachable after each loop iteration.

(b) It has double-deletes; on each loop iteration, the two new C objects point to the same heap-allocated int. So the second object's destructor calls delete on a dangling pointer.