Problem 4 (10 points)

Consider the following code:

```
void func1(int& x) {
    int y = computeY();
    x += y;
    long_running_func_to_do_other_stuff();
}
void func2(int& x) {
    int z = computeZ();
    x = 2*x – z;
    another_long_running_func();
}
...
int x = 0;
...
std::thread t1 = new std::thread(func1, ref(x));
usleep(100000);          /* Delays for 100 milliseconds */
std::thread t2 = new std::thread(func2, ref(x));
t1.join();
t2.join();
std::cout << "The value of x is " << x << std::endl;
```

Is the value that will be printed for x deterministic? If so, explain why; if not, explain why and modify the code to fix the problem. You may assume that computeY and computeZ are deterministic.

Answer: there is a race condition in the code. Delaying for 100 ms will not guarantee that t1 **modifies** x **before** t2**: it's always possible that the scheduler could decide not to execute** t1 **for a long time, so that** t2 **ends up executing first, or even concurrently.**

The solution is to add explicit synchronization, such as this:

```
void func1(int& x, int& mod_done, std::mutex& m,
           std::condition_variable_any& t1_modified_x) {
    int y = computeY();
    x += y;
    m.lock();
    mod_done = 1;
    t1_modified_x.notify_one();
    m.unlock();
    long_running_func_to_do_other_stuff();
}
```

```
void func2(int& x) {
    int z = computeZ();
    x = 2*x – z;
    another_long_running_func();
}

...

int x = 0;
std::mutex m;
std::condition_variable_any t1_modified_x;
int mod_done = 0;

...

std::thread t1 = new std::thread(func1, ref(x), ref(mod_done), ref(m),
                                 ref(t1_modified_x));
m.lock();
while (!mod_done) {
     t1_modified_x.wait();
}
m.unlock();

std::thread t2 = new std::thread(func2, ref(x));

t1.join();
t2.join();
std::cout << “The value of x is “ << x << std::endl;
```