ECE498SL: Engineering Software Systems

Midterm Examination

10 March 2009

• You may use your brain and assorted writing implements.

• You may not use books, notes, calculators, other people’s brains, etc.

• You may have one 8.5×11-inch page (double-sided) of handwritten notes.

“The Dean glared. ‘That’s not magic!’ he snapped.
‘That’s just…engineering!’ ”
from *Interesting Times*, by Terry Pratchett

Most of the problems require a short response. If you choose to write a long response, say more than about 25 words, I’ll probably stop reading after the short part at the front. Avoid pronouns: it makes it hard to know what it means.

Name: _______________________________   (5 points!)
A (15 points): Describe three practical concerns when developing a programming language. Here “practical” means those concerns that would not be relevant if your language were fully integrated into college computing curricula and used by programmers trained using your language to write completely new software systems twenty years from now.

#1:

#2:

#3:

B (5 points): Explain how the concept of scope in C interacts with file organization and how C++ decouples these two aspects of software design.

C (5 points): Describe how a C++ class can be designed to guarantee that a class initialization routine is called before instances of the class are used by any given program.
D (5 points): Code for managing resource and permission failures, such as those that might occur with memory management and file system operations, poses one of the more difficult challenges for systematic software testing. The system does not normally fail, thus testing the failure-handling code might require execution under control of a debugger and changes to return values being checked. Explain how you can simplify the process for dynamic memory management using standard C++ mechanisms.

E: When using `setjmp/longjmp`, a programmer must be careful to provide methods through which partially completed work within routines between the `setjmp` ancestor and the `longjmp` descendant can be cleaned up.

E.1 (5 points). Describe one such type of work.

E.2 (5 points). How does the C++ exception handling mechanism help to reduce such cleanup efforts by a programmer.

E.3 (5 points). Does C++ exception handling deal with all such work? If not, describe one such type of work that remains when using exceptions in C++.
**F (5 points):** Explain why references are necessary to support operator overloading. Use an example to illustrate your explanation.

**G (5 points):** Describe one advantage of throwing C++ exceptions rather than inserting preprocessor-based ASSERT macros to check assumptions at module boundaries.

**H (10 points):** The following implementation of addition for complex numbers has at least two bugs. Point two out and explain why each one might cause problems, they suggest how to fix the code to remove the bugs.

```cpp
complex& complex::operator+ (complex& c1, complex& c2)
{
    complex c;

    c.real = c1.real + c2.real;
    c.imag = c1.imag + c2.imag;

    return c;
}
```
I. Let’s say that an $N \times N$ matrix has density $\varepsilon$ if at most a fraction $\varepsilon$ elements within any row (a total of $N\varepsilon$ elements) are non-zero. We call matrices with low density sparse and represent each row or column with a more efficient data structure than an array full of zeroes.

A friend is extending a sparse matrix code and finds the lookup time for individual matrix elements to be irritating. In particular, each lookup takes time proportional to $\log_2(N\varepsilon)$ using the code’s data structure. Your friend wants to create a full copy of the matrix (with all of the zeroes) as a two-dimensional array so as to allow random access to matrix elements in a fixed amount of time.

I.1 (5 points): In terms of $N$, how many matrix element lookups must your friend perform to make such a transformation worthwhile?

I.2 (5 points): What other practical issues must your friend consider if $N$ is large? Explain one such issue.

J.1 (5 points): How does function inlining complicate interpretation of results for sample-based profiling?

J.2 (5 points): If you find that function inlining is causing confusion in such a case, why not just turn it off and profile the code again without function inlining?
K (10 points): Consider the following code. Class BETA’s definition is not relevant to the question and is omitted, although it would be necessary for compilation.

class ALPHA {
public:
    ALPHA (); // #1
    ALPHA (const ALPHA&); // #2
    ALPHA& operator= (const ALPHA&); // #3
    friend ALPHA operator+ (const ALPHA&, const ALPHA&); // #4
    friend ALPHA operator- (const ALPHA&, const ALPHA&); // #5
    friend ALPHA operator* (const ALPHA&, const ALPHA&); // #6
};

BETA* crunch (const ALPHA& a1, const ALPHA& a2) {
    ALPHA sum;
    ALPHA diff = a1;
    sum = a1 + a2;
    diff = diff - a2;
    return new BETA (sum * diff);
}

Write the sequence of calls (using the numbers to the right) made directly from the crunch routine to the routines listed in class ALPHA.