ECE498SL: Engineering Software Systems

Midterm Examination

9 March 2010

- 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.

"…reason thinks all reasoning out of season."
from *Don Juan*, by Lord Byron

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 (5 points): On the first day of class, we examined a case study that showed that Java was only about a factor of two slower than C. Explain why such a conclusion might be misleading in regard to other sources of optimization potential.

B (10 points): Consider the following code.

```
class ALPHA {
public:
    ALPHA ();                                                   // #1
    ALPHA (const ALPHA&);                                       // #2
    ALPHA (short int, ...);                                     // #3
    ALPHA (double, const char*);                                // #4
    ALPHA& operator= (const ALPHA&);                            // #5
    ~ALPHA ();                                                  // #6
    friend ALPHA operator+ (const ALPHA&, const ALPHA&);        // #7
    friend ALPHA operator- (const ALPHA&, const ALPHA&);        // #8
};

class BETA {
    private:
        ALPHA one;
        ALPHA two;
    public:
        BETA (const ALPHA& arg, int value) : two (arg) {
            ALPHA tmp (value, "temporary");
            tmp = tmp + two;
            one = tmp - one;
        }
};
```

Write the sequence of calls (using the numbers to the right) made directly from the code generated for class **BETA**'s constructor to the routines listed in class **ALPHA**.

C (5 points): Modularity, extensibility, and information hiding are desirable high-level properties in a large software system. Choose one of these properties (say which) and give an example of how some C++ feature makes writing code with that property easier relative to C.

D: In C, derived types can be defined by including the structure for the base type as the first member of the derived type's structure, as shown below.

D.1 (5 points): Fill in the blanks so that (1) the function **`some_function`** can accept a pointer to any type derived from **`base_type`** (for example, a **`derived_t`**) and (2) to show how **`some_function`** is called from **`another_function`** on parameter **`d`**.

```
typedef struct {
    double base_field;
} base_t;

typedef struct {
    base_t base;  // must be first field in sub-type structure
    int    new_field;
} derived_t;

extern void some_function (________________________________);

void
another_function (derived_t* d)
{

    some_function (_________________________________________);
}
```

D.2 (5 points): Explain a potential drawback of your solution and how C++ enables you to solve this problem more elegantly.

E (5 points): In C++, **`friend`** functions must be declared in the class definition. Briefly explain an advantage of not allowing a function to declare itself a **`friend`** of a class outside of the class' definition.

F (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.

G (5 points): Recall the cleanup mechanism described in class (and used, for example, with Posix threads) for handling exceptional conditions. Cleanups are block structured: at the start of a block of code, a function pointer and single argument are pushed together onto a stack. At the end of the block, they are popped off and possibly executed. If an exception occurs, the entire stack is popped and executed one by one.

Explain how one might implement C++'s approach to exception handling using a cleanup mechanism. In particular, explain when and what type of cleanups should be pushed, whether they should be executed when they are popped, and how exceptions can be handled.

H (5 points): Let **`A`** and **`B`** be instances of a class **`ALPHA`**. Explain why the statement **`A += B`** might be executed more efficiently than the statement **`A = A + B`**.

I (5 points): Consider the two C++ expressions below. Are they equivalent? Explain what each does.

```
instancePointer->member
```

```
(*instancePointer).member
```

J (5 points): Explain one benefit of including support for operator overloading in a programming language.

K: Recall the sample-based approach to application profiling, in which a program is periodically stopped and its current program counter/instruction pointer recorded as a sample.

K.1 (5 points): A longer sampling time implies a larger absolute error in the number of samples observed in any given function. Does this fact imply that you should always profile your code for as little time as possible? Explain your answer.

K.2 (5 points): Recall the aliasing problem with sample-based profiling. As an example, if the number of events in a loop evenly divides the number of events between samples, the sample distribution will not be uniform (as desired) across the events in the loop. Instead, all samples will be for the same single event from the loop; other events will receive zero samples. The ProfileMe work added hardware to address problems such as aliasing. What approach can you use to reduce aliasing without adding hardware?

K (10 points): The code below defines a templated function **`mystery`**. Fill in the blank to invoke this code *as simply as possible* on variable **`alpha`** with class **`B`** set to **`BETA`**, then explain how the template might be rewritten to simplify the invocation.

```
template<class A, class B> A mystery (const A& arg) {
    B = A.getObject ();
    if (B.isGood ()) {
        return A;
    }
    return A (42);
};

void someOtherFunction ()
{
    ALPHA alpha;

    ALPHA result = ____________________________________________;
}
```

L.1 (5 points): Explain one advantage of explicitly instantiating a class template for a given set of types (relative to not doing so).

L.2 (5 points): Explain one disadvantage of explicitly instantiating a class template for a given set of types (relative to not doing so).

M (5 points): A certain program consists of the following sequence of operations:

1. read in data set of size N from disk
2. create auxiliary structure of size N^3
3. execute algorithm with $N \log_2(N)$ steps
4. execute algorithm with N^2 steps

What is the overall computational complexity of this algorithm (that is, what polynomial in N bounds the total run time within a constant factor for large values of N)?