Review and Goals

The goal of this handout isn't so much to assess what each of you knows, but more to get a vague idea of where I should be structuring the class. I don't want to be so basic as to bore most of you, and I want to make sure I won't be losing you. This is especially a problem since I think many of you are from different backgrounds. To that end, if you would give me some feedback on what you do and do not know—or what doesn't seem familiar—I would very much appreciate it, and I think it would make the class more interesting for you. I'm not looking for anything long, or even an answer to every question, just something to let me know where you are and where you'd like to be. Don't worry if you don't know some or many of these; I geared some of these questions to be rather tough.

Course Goals

1. What made you want to take this class?
2. What do you want to get out of it? (Examples: language lawyering; techniques to improve efficiency, design, and such; ultra-practical knowledge on how to deal with memory management, and the like; a cool new library to call your own)
3. I am required to give you two assignments. At the moment, one is a policy-based smart pointer, and the other will be the design (and partial implementation) of a library of your choosing. Does this sit well with you? (I understand that you may not know what a smart pointer is, or what a policy is. The former is a class that acts like a pointer but handles object lifetime and other things. We will discuss the latter at great length.)
4. Do you have any ideas for the second project already? (No pressure at all.)

Objects: Idioms and Best Practices

5. What is Resource-Acquisition-Is-Initialization (RAII)? Why should you use it?
6. Could you implement `operator=` for a container class?
7. When should a destructor be `virtual`?
8. What's wrong with this class? (Hint: What's missing?)

```
class log_file
{
private:
   FILE * file_;
public:
   log_file() // Ignore fopen failures...
   : file_(fopen("path/to/my/log","W"))
   {
        fprintf(file_,"Log opened.\n\r");
   }
   ~log_file()
```

```
    {
        fprintf(file_,"Log closed.\n\r");
        fclose(file_);
    }
    void print_message(const char * mesg)
    {
        fprintf(file_,mesg);
    }
};
```

Templates

9. Could you implement an STL-style iterator? (with, say, a standard library reference?)
10. What's `typename` for? Do you know when to use it?
11. What kinds of things can be template parameters?
12. Does Substitution-Failure-Is-Not-An-Error (SFINAE) mean anything to you?
13. What is a policy? What are traits?

Exceptions

(These are tentatively the subject of the first lecture.)

14. What are exceptions? Why are they better than error codes?
15. What kinds of things can be thrown? What's the best way to catch?
16. When might error codes be better than exceptions?

Efficiency

17. What are the first two rules of optimization?
18. What is the 80-20 rule?
19. What is the compiler allowed to optimize away? (What is a "trivial" destructor?)
20. Why are functions not `virtual` by default?
21. What's wrong with this code:

```
for(int i = 0; i < names.size(); ++i)
{
   std::string outstr = names[i] + " is taking CS93SI!";
   send_message(outstr);
}
```

Esoterica

22. Do you know what Argument-Dependent Lookup is? Two-Phase Name Lookup? Are you curious?