Fall 2006 EE380L Quiz1: NAME: _____

RULES: No internet searches of any kind. You may work on this examination for 2 hours (no longer). The time can be distributed over at most two contiguous intervals where each interval is at least 15 minutes long. When you elect to work on the examination, you must be in a private work environment with no outside assistance of any kind. Text books are permitted, as are all EE380L materials (including the EE380L web sites). You are not permitted to consult any "research papers" or published articles of any kind (only text books). You may use a compiler to write and test possible solutions. However, it is not necessary that your solution compile or run in order to receive a perfect score (be wary, as the compiler can be a time sink).

Addendum: You are permitted a grand total of four hours on this assignment. You can spend at most 2 hours actively working on your solution. In addition to the two hours of active work, you are permitted to "daydream" about the problem for at most an additional two hours. That means, if you are thinking about this problem on your ride home today, then that time spent in the car/bus will not count against the time you have available to spend sitting at your desk working on the exam. You can think about the problem while you're in the shower, lay awake at night thinking about it, etc. However, you must not abuse this license. After two hours of stewing over the problem, any additional time you spend thinking on the problem counts against the time allotted. Time spent in class today does not count against your four hours.

By attaching your name to this paper, you affirm the following statement:
I have fully complied with both the letter and the spirit of the academic honesty policies for this examination. I recognize that this examination is intended to be an evaluation of my ability to develop my own designs. Therefore, I recognize that searching for and/or reproducing designs, approaches or solutions developed by others (whether published or not) is in violation of the academic honesty rules. I attest that the following log accurately reflects the time I spent working on this examination:

Date time started time completed

_____

_____

Imagine you are chief software architect at a small company. You are about to undertake a project where the application that you are building may need to run for hours (or days) at a time. One of your design goals (and the focus of this particular quiz) is enabling efficient and convenient checkpointing.

Let's assume your first decision is that there must be a common interface to checkpoint something, and that you have determined that there should be a pair of functions as follows:

```
void save(ostream& output_device, const T& object);
T restore<T>(istream& input_device);
```

The save function is used to create a checkpoint. In typical usage, the template argument for T will be deduced by the compiler (i.e., implicit). The restore function is used to recover an object previously saved in a checkpoint. The template argument for T cannot be deduced in this function, and so would be specified explicitly. Your task is to design and implement these functions. Obviously, templates will be involved. You may mix templates with overloaded functions and you may also specialize the function templates. Here are a few additional details.

- Many objects (including all the C++ base types) can be saved (and restored) by writing (reading) the binary image of the object directly to the stream. In other words, given an object, *x,* of type *T*, we can save the object by writing sizeof(T) bytes into the stream starting with the first byte at &x. The ostream::write method (and istream::read) method can be used for this purpose.
- Pointers cannot be handled by the default approach. If *x* is an object of type T*, then we must not write *x* directly to the checkpoint file. Instead, we must save the object (*x) to the file. In that way, when we recover x later, we can construct the object (*x) on the heap and restore x so that it points to the recovered object. Your save and restore functions must work in this manner when operating on pointers.
- (You should assume that your first decision as chief software architect was to forbid the use of arrays in the software (your programming staff will have to use vector<T> whenever they think they need an array). Hence, you can assume that any T* object points to a single object and not an array). This is not an additional requirement, but it simplifies (justifies?) your approach for the previous bullet.
- Class objects with a non-trivial copy constructor will require a non-trivial mechanism to save and restore. Assume that you have instructed the programming staff to write a method T::writeMe(ostream&) for any class T that has a non-trivial copy constructor. Simiarly, assume that any class that has a T::writeMe method also has a constructor T::T(istream&) that can be used to restore the object. Getting the save and restore functions to work with these classes may require your company to follow additional specific guidelines when writing the classes, as well as careful implementation of the *save* and *restore* functions. Fortunately, you're the chief software architect, so you get to dictate any code conventions or guidelines that you think are necessary. Write save and

restore that will work on arbitrary class types. Be sure to clearly state what additional requirements (if any) you feel need to be placed on type T (be specific, and be sure to relate these requirements to your implementation of save/restore). As a small hint, I do not think it is possible to write *save* such that it checks to see if T::writeMe exists and then uses the default method when it does not exist, so I'd suggest trying a different approach. You may want to insist upon the existance of T::*something_else*. If that doesn't work, maybe you can insist upon the existance of *something_else*<T>

- Finally… you are not permitted to use or to require the use of virtual functions (i.e., neither your *save* or *restore* nor your coding conventions can include any virtual functions). If you simulate virtual functions (by having, for example, a data member which is **true** when writeMe exists and false when writeMe does not exist), then your solution will be graded as if you had used virtual functions. Your best bet is to attempt, as much as possible, to solve this problem with no **if** statements and no virtual functions (if you do that, then you're probably doing what I want you to do).

I am expecting a reasonable concise collection of templates and other C++ functions implementing *save* and *restore*. I also insist that you clearly document any and all coding conventions that you would impose upon the team (limit your coding conventions to things relevant to your templates – I really don't care whether spaces or tabs are used for indentation).