Fall 2006 EE380L Quiz4: NAME: _____

RULES: No internet searches. 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).

Your solution to this exam must be received, by email, before 5:00 PM, CST, Friday November 17th. Please email your solution to chase@ece.utexas.edu with the subject QUIZ4 submission. Solutions should be no more than 500 words. Solutions can be sent in plain text, in Microsoft Word or in PDF format.

Addendum: NOTE: you are permitted to work on this quiz for only two 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

_____

_____

One of the nice properties of Expression templates is to combine the work from what would otherwise have been multiple loops into a single loop. Let me give an example:

Given Valarrays *x*, *y* and *z*, and a scalar *a* the statement "z = a*x + y;" will compile down to a single loop (assuming you have good expression templates). This particular expression is a rather important one, in fact, it's often referred to as the SAXPY expression (where "S" stands for single-precision, AX (meaning "A times X"), P (for plus) Y. SAXPY was the name for the original FORTRAN subroutine implementing this expression in the BLAS library (Basic Linear Algebra Subroutines) that ultimately became the basis for Matlab. There is (of course) also a DAXPY for Double-precision AX-Plus-Y in the BLAS library.

Anyway, the nice thing about using Expression Templates, is that we have no need of coding up SAXPY's or DAXPY's or even dot-products. The compiler will ultimately create a single loop, inline all the function calls, and optimize the code down to (pretty much) exactly what we'd have gotten from the FORTRAN library.

So, I've been thinking. Is there anyway to get the C++ language to "combine loops" from multiple assignments? Consider a case where we need to assign to two different arrays (V, W, X and Y are all Valarrays in the example), such as:

```
W = X + Y
V = X – Y
```

Even using Expression Templates, this code presents a problem. If the compiler creates two loops (one loop that assigns all the elements of W, and a second loop that assigns all the elements of V), then the computer ends up loading each element of X (and Y) twice. Instead, we'd prefer to have the compiler essentially generate the following code in which any optimizing compiler will load each element of X (and Y) only once.

```
for (k = 0; k < N; k += 1) {
  W[k] = X[k] + Y[k];
  V[k] = X[k] – Y[k];
}
```

I've thought about this a bit, and was wondering if maybe we could combine Expression Templates and some object-oriented programming to get this done. Consider a solution as follows

```
{ Atomic Block
  W = X + Y;
  V = X – Y; }
```

The idea is to create some type *Atomic* (the name "Block" is a variable here, and actually doesn't matter, we could have called it Stanley, I just picked "Block" for artistic purposes). The Expression templates inside the { } will add their work to the Atomic

object. Then when the destructor for the Atomic object is run (i.e., Block.~Atomic() in this example), the work from the Expression Templates is finally applied.

My objective for doing this is to end up the compiler generating a single loop which contains all the assignments from the block.

Sketch how this could be done. Identify what base class you'd need, what sub-classes and what virtual functions you'd rely upon. Explain what you think the difficulties are in meeting the goal. Please keep in mind that if the machine code that is ultimately run ends up being more than one loop, then we have failed (and the whole effort is pointless). So focus your analysis on trying to solve this problem.