Fall 2007 EE 380L Final: NAME:______________________________

1. (10 pts) Using the following code, write a template class BlackBox<T>. The base class for your template should be a Box<T, x> where the value of x is set based upon whether T has a virtual function table. If the type T has a virtual function table, then the base class for BlackBox<T> should be the class Box<T, VPTR>. If T does not have a virtual function table, then the base class should be Box<T, NO_VPTR. Be sure to include a contructor in your BlackBox template.

```
template <typename T> class HasVirtual : public T {
   virtual ~HasVirtual(void) {}
};

template <typename T>
bool has_vptr(const T& x) { return sizeof(T) < sizeof(HasVirtual<T>); }

enum Constants {
      NO_VPTR = 0
      VPTR = 1
      };

template <typename T, int BoxStyle> class Box {};

template <typename T> class Box<T, NO_VPTR> {
   const T& x;
public:
   Box(T& _x) : x(_x) {}
   template <typename Z> Z returnAs(void) { return Z(x); }
};

template <typename T> class Box<T, VPTR> {
   const T& x;
public:
   Box(T& _x) : x(_x) {}
   template <typename Z> const Z& returnAs(void) {
     return dynamic_cast<const Z&>(x);
   }
};
```

2. (10 pts) Write a template class Larger<typename X, typename Y> that has a nested type called “Answer”. Write your class such that Larger<X, Y>::Answer is always either X or Y. If sizeof(Y) is larger than sizeof(X) then Larger<X, Y>::Answer should be Y, otherwise it should be X.

3. (15 pts) Write an operator<< for ostream and class Foo such that the output is displayed in reverse order. So, for example, the following program should print CBA. Hint, use expression templates.

```
class Foo {
  char let;
public:
  Foo(char v) { let = v; }
};

int main(void) {
  Foo a('A'), b('B'), c('C');
  cout << a << b << c << endl;
}
```

4. (15 pts) Design and implement a class Number with the following behavior.
   - Any **int** or **double** can be promoted to a Number without using a type cast.
   - Two Numbers can be added with operator+(). You do not need to support any other arithmetic with Numbers.
   - When adding two Numbers, if both the values are known to be integers, then integer arithmetic should be used to add them. If one (or both) of the numbers is a double, then double-precision floating point should be used to add the numbers.

   Use an object-oriented approach (i.e., avoid if statements or comparable designs)

```
int main() {
  Number x = 3;
  Number y = 5;
  Number z = 2.5;
  x = x + y; // uses int arithmetic, x is assigned 8
  x = x + z; // uses double arithmetic, x is assigned 10.5
  cout << x + z << endl; // uses double arithmetic, prints 13.0
}
```

5. (10 pts) Arrays in C++ just don’t work like they should. For example

   a. What is wrong with the following program – i.e., what happens when it is compiled and run?

```
struct Base {
      virtual void doit(void) { cout << "Hello World\n"; }
};

struct Derived : public Base {
      int x;
      Derived(void) { x = 42; }
      virtual void doit(void) { cout << "x is " << x << endl; }
};

void doit(Base x[]) {
      for (int k = 0; k < 10; k += 1)
            x[k].doit();
}

int main(void) {
      Derived derived[10];
      doit(derived);
}
```

   b. What happens when I change doit to be a template? Will the program produce the correct behavior (ten lines of “x is 42”)?

```
template <typename T>
void doit(T x[]) {
      for (int k = 0; k < 10; k += 1) {
            Base& b = x[k];
            b.doit();
      }
}
```

6. (10 pts) I wrote a program that dynamically allocated an array of objects. The array could either be an array of Base objects or an array of Derived objects (Derived is a subtype of Base). Assume that class *Base* has no data members.

```
class Derived : public Base {
  int* p;
public:
  Derived(void) { p = new int; }
  ~Derived() { delete p; }
};
int main(void) {
  Base* b;
  if (random() % 2 == 1) { // coin toss
    b = new Base[5];
  } else {
    b = new Derived[5];
  }
  // how do I delete b?
}
```

The problem I have is trying to delete the array. I want to avoid any memory leaks (i.e., I need to make sure that every byte of memory allocated by **new** is deallocated by **delete**). I see two issues, each issue with two choices. What combination of the following two choices (if any) will solve my problem:

Choice 1: The *Base* destructor can be **virtual** or **not virtual**
Choice 2: I can deallocate b using "`delete b`" or "`delete[] b`"

Can the problem be solved using those options? If so, how?

7. (15 pts) Complete the following generic algorithm *reverse*. Recall that "reversing" a container means reversing the order that the elements appear inside the container (make the last element first, the first element last, *etc*). You may write as many "support functions" as you find necessary, as always a simple solution is better than a complex solution. You may assume that iterator_traits is available to you and provides the following types, iterator_traits<T>::value_type, iterator_traits<T>::iterator_category. Your answer must satisfy the following criteria.
   - *reverse* can be invoked on any type of container (i.e., iterator type).
   - If *reverse* is invoked with bi-directional iterators, then the elements are reversed "in place" (no additional storage is required).
   - If *reverse* is invoked with forward iterators, then the elements are reversed using a temporary *vector* object. You must declare/define the *vector* you use in your solution.

```
template <typename Iterator>
void reverse(Iterator b, Iterator e) { // you finish.
```

8. (15 pts) Short answer questions:

a. Assume we have an application that uses the following function:

```
void doit(T x) {
  x.foo();
}
```

What if we changed the function to pass the argument by constant reference:

```
void doit(const T& x) {
  x.foo();
}
```

If the program compiles with the new version of doit, but behaves differently than it had before, what can we conclude about type T and method foo?

b. Assume that reference counting is used as part of the implementation of a garbage collector.

ii. Is it possible that an object will have a reference count of zero and **not** be garbage?

iii. Is it possible that an object will be garbage and have a reference count larger than zero?

d. Assume an inheritance hierarchy with type Base that has *F* virtual functions, *D* different derived types, assume that each derived type overrides all of the virtual functions.

i. What is the time complexity to invoke one of the virtual functions for a pointer p of type Base*?

ii. What is the space complexity of the system if there are *O* objects total (equally divided among the types) and all objects have the same data members.