“…the masses of humanity through the ages, feeling inadequately educated just like we do now…have had little choice but to believe this guesser or that one.”
from *Man without a Country*, by Kurt Vonnegut

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):** In C++, one can wrap a header include in a namespace block as follows:

```cpp
namespace moduleOne {
    #include "graphics.h"
}
```

Explain how such a construct can be of use in integrating several existing code modules. In other words, what problem might this construct help to solve?

**B (5 points):** Hiding the implementation of a data structure in C code often implies defining the structure within a single source file. Explain how the lack of a structure definition can limit the performance of code in other source files when using the structure. In particular, give an example of a structure use or optimization that, while desirable, is not possible.

**C (5 points):** Adoption of new programming languages is hindered by the difficulty of rewriting existing code in a new language. One strategy suggested to overcome this problem in the context of parallel languages is to use library interfaces to bridge the gap. Library routines can be written in a new parallel language, then called from existing sequential code. Explain one performance challenge for this type of integration (as opposed to code written in a single language). Note that the new programming language need not be parallel—introducing a new sequential language in this way poses similar challenges.
D (5 points): As you may recall, **gprof** adds instrumentation to count the number of calls made to each function in your program. Explain one drawback of this instrumentation.

E (5 points): A friend of yours has been trying to use OProfile to understand the behavior of the function shown below. When the friend uses the branch misprediction counter, however, OProfile reports that the line marked by the comment has mispredictions, even though the line should clearly not generate any branch instructions. Explain the problem.

```c
void oddity (int* array, int len, int value)
{
    int min = value;

    for (int i = 0; len > i; i++) {
        int tmp = array[i];  // line has branch mispredictions!?
        if (min > tmp) {
            min = tmp;
        }
    }
    return min;
}
```
F: 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.

F.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`.

```c
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 (__________________________);
}
```

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

---

G (5 points): A friend of yours comes to you with a puzzle. If they add the constructor `ALPHA (const BETA&)` to a class that already has a copy constructor (`operator=`) and arithmetic operators defined—for example, `friend ALPHA& operator+ (const ALPHA&, const ALPHA&)`, the compiler then accepts the following code:

```c
// alpha is an ALPHA
alpha = alpha + '\0';
```

Explain how the compiler must be interpreting the code in order to accept it.
**H:** Consider the following code fragment illustrating a function with error-checking of its arguments.

```c++
int foonction (int arg1, char* arg2, const ALPHA& arg3)
{
    // Check arguments using ASSERT macros
    // or by throwing exceptions.

    // Do something really cool!
}
```

When checking the function’s arguments, you can use either ASSERT macros or C++ exceptions. ASSERT macros abort the program when in debugging mode but generate no code in the final software product (production mode). C++ exceptions are compiled in the same way in both versions, as discussed in class.

**H.1 (5 points):** Explain one advantage of using ASSERT macros rather than C++ exceptions.

**H.2 (5 points):** Explain one advantage of using C++ exceptions rather than ASSERT macros.

---

**I (10 points):** The following implementation of addition for complex numbers has at least two bugs. Point two out and explain why each one might cause problems, then suggest how to fix the code to remove the bugs.

```c++
complex& complex::operator+ (complex& c1, complex& c2)
{
    complex c;

    c.real = c1.real + c2.real;
    c.imag = c1.imag + c2.imag;

    return c;
}
```
J (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 : public ALPHA {
    private:
        ALPHA one;
        ALPHA two;
    public:
        BETA (const ALPHA& arg, int value) :
            one (42, arg), two (arg) {
            ALPHA tmp (value, "temporary");
            tmp = tmp + two;
            one = tmp * (*this);
        }
    }

Write the sequence of calls (using the numbers to the right) made from the code generated for class BETA’s constructor to the routines listed in class ALPHA.
K: The Container class defined below maps objects of class Key into integers. However, the array operator as defined is slightly broken.

    class Container;
    class Key;

    class ContainerRef {
        private:
            Container* c; // container being used
            const Key* k; // key being read/modified
        public:
            ContainerRef (Container* con, const Key* key) :
                c (con), k (key) {}
            void operator= (int val) const;
            operator int () const;
    }

    class Container {
        public:
            ContainerRef operator[] (const Key& k) {
                return ContainerRef (this, &k);
            }
            // remainder of class definition omitted...
    }

K.1 (5 points): Give an example of a reasonable C++ statement using array operations on a Container CON and keys k1, k2, etc. that will not compile given the definition above.

K.2 (5 points): Explain how to fix the definition in a way that enables your answer to the previous question to compile (without breaking other statements!).
L: Answer the questions below about this simple template:

```cpp
template<class T> class Teg {
    private:
        int iVal;
        T tVal;
    public:
        Teg (const T& t) : iVal (42), tVal (t) {}       
        void absorb (const T& t) {
            iVal += t.iVal;
            tVal *= t.tVal;
        }
        bool operator< (const T& t) const {
            return (tVal < t.tVal);
        }
};
```

L.1 (5 points): For what values of `class T` is code generated when a C++ compiler sees this template definition?

L.2 (5 points): When is code for the function `Teg<ALPHA>::absorb` generated by a C++ compiler?

L.3 (5 points): A class BETA defines functions `BETA operator+= (const BETA&)` and `bool operator< (const BETA&, const BETA&)`, but contains no `operator*=`. Under what conditions, if any, can objects of type `Teg<BETA>` be used?