Homework 5

Problem 1

a)

The type of refCount should be std::shared_ptr<int> refCount so that the reference count can be shared across all my_shared_ptr instances pointing to the same object. It itself is a smart pointer and will allow for the reference counting to work across all

b)

```
my_shared_ptr(int *ptr){
  this->ptr = ptr
  refCount = std::make_shared<int>(1);
}
```

c)

```
my_shared_ptr(const my_shared_ptr &other) {
  ptr = other.ptr;
  refCount = other.refCount;
  *refCount++;
}
```

d)

```
~my_shared_ptr(){
  *refCount--;
  if (*refCount == 0) {
    delete[] ptr;
  }
}
```

e)

```
my_shared_ptr& operator=(const my_shared_ptr & obj){
  ~my_shared_ptr();
```

```
  return *(my_shared_ptr(obj));
}
```

Problem 2

a)

If decisions about collisions with incoming asteroids and meteors are made in between 100ms, then consistent performance and timeliness of the program run is perhaps the most important factor in determining the language.

Garbage control could add un-wanted pauses int the program, whereas C, C++, and Rust's languages have more control over memory management that is more reliable and better suited to fast response requirements.

b)

I think it would address some concerns, but it still doesn't offer the control that lower level programs have in managing memory that would increase reliability of the program. Additionally, it could add overhead in incrementing the reference counter which would hurt the timeliness of the program

c)

C# offers better memory utilization due to compacting, which would improve performance of garbage collection. The mark and compact GC in C# would also reduce memory fragmentation which would improve performance for frequent allocations and deallocations.

Go would still be functional, but in terms of consistent performance and reliability, C# may offer a better option.

d)

C++’s destructor is typically called immediately when an object goes out of scope or is explicitly deleted, which cleans up the socket file descriptor. Therefore, when RoomView isn’t needed anymore, the socket file descriptor is released immediately.

Whereas, Go’s garbage collection isn’t called immediately when the RoomView object goes out of scope. It’s called when the garbage collector decides to run, which can be problematic if too many socket file descriptors are created.

She can resolve this by calling the destructor when the RoomView is no longer needed. She’ll need to explicitly call it when needed to ensure proper garbage collection.

Problem 3

a)

If this code works as I would assume the developer wanted to, it would be call by reference because they would want the changes made to x and y within the function f to be reflected in the main function. So f would take the references of x and y.

b)

Since print(x) and print(y) both output 2, the language uses call by value. The changes to x and y do not affect the original variables, so the function f just copies and y and doesn’t have access to the member variables.

c)

If pass by value, print(x.x) should just bring 2 which is the initial value set in the X constructor

If pass by object reference, the function operates on the same object x so it would print 5, which is the new value set in the function call.

d)

Lines 1 and 2 use casts — the value in the code is just being treated as a different type given that only mov and call instructions are being used

Lines 3, 4, and 5 use conversion —

- line 3 uses mvswl to sign extend a short to a long value after int is converted to short, to make the short an int again
- line 4 uses compl, setne, and mvzbl to convert an integer to a bool value by comparing a with 0, setting a to true or false
- line 5 uses cvtsi2ssl that converts a signed int to a float