1. (10 points) The output for the program below is stuff, which results from the fact that function string::getBuf violates Meyers Item #28: “Avoid returning handles to object internals.”

   (a) Add code (but do not remove any code) to prevent this violation.
   To prevent returning a handle to buf, add const on line #8

   (b) Write function setBuf, which should be used to modify buf.

```
 1  #include <iostream>
 2  #include <cstring>
 3  class string {
 4  public:
 5    string(const char* b) : buf(new char[strlen(b)+1]) {
 6      strcpy(buf, b);
 7    }
 8    const char* getBuf() const { return buf; }
 9    void setBuf(const char* b) {
10      delete [] buf;
11      buf = new char[strlen[b]+1];
12      strcpy(buf, b);
13    }
14  private:
15    char* buf;
16  };
17  int main( ) {
18    string s(”Elysium”);
19    // The next line won’t compile if getBuf returns by const char*
20    //strcpy(s.getBuf(), ”stuff”);
21    std::cout << s.getBuf() << std::endl;
22  }
```

2. (10 points) Give the output for the following program.

```
#include <iostream>
#include <cstring>

class string {
public:
  string() { std::cout << "default" << std::endl; }
  string(const char *) { std::cout << "convert" << std::endl; }
  string(const string&) { std::cout << "copy" << std::endl; }
  string& operator=(const string&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
private:
  char * buf;
};
class Student {
public:
  Student(const char *n) {
    name = n;
  }
private:
  string name;
};
int main() {
  Student bill("bill");
}
```

```
default
convert
assign
```

3. (10 points) Figures 1 and 2 represent listings for two classes, A and B, that have mutual dependencies. As listed below, the files won't compile. Add whatever code is needed to be able to compile and execute the two classes. (10 points)

```


class A {
public:
  A();
   int f();
   int g() const;
private:
  B* b;
};
```

```
#include <iostream>




A::A() : b(new B) { }
 int A::f() { return b->g(this); }
 int A::g() const { return 17; }
```

Figure 1: **a.h and a.cpp**: Header and implementation files for class A

```



class B {
public:
  B();
   int g(const A*) const;
};
```

```
#include <iostream>



B::B() {}
 int B::g(const A* a) const { return a->g(); }
```

Figure 2: **b.h and b.cpp**: Header and implementation files for class B

4. (10 points) Give the output for the following program.

```
#include <iostream>
class A{
public:
   virtual void f(){}
};
class B : public A {};
bool f() { return 99; }

int main() {
  A* x = new B;
  B* y = dynamic_cast<B*>(x);
   if ( y && f() ) { std::cout << "true" << std::endl; }
   else { std::cout << "false" << std::endl; }
}
```

```
true
```

5. (15 points) Supply the line of code needed on line #26 so that vec is sorted. Then write function eraseMultiplesLess so that all the multiples of a given number less than a given number are erased from vec. For example, sample output for deleting all multiples of 5 less than 50 might be:

```
2, 3, 8, 11, 11, 12, 13, 13, 14, 19, 21, 21, 22, 23, 24, 24, 26, 26, 26,
26, 27, 27, 29, 29, 29, 29, 32, 34, 36, 36, 37, 39, 39, 42, 43, 46, 49,
50, 51, 54, 56, 56, 57, 58, 59, 60, 62, 62, 62, 63, 64, 67, 67, 67, 67,
68, 68, 69, 70, 70, 72, 73, 73, 76, 76, 77, 78, 80, 81, 82, 82, 83, 84,
84, 84, 86, 86, 86, 87, 88, 90, 91, 92, 93, 93, 94, 95, 96, 98, 99,
```

```
1  #include <iostream>
2  #include <list>
3  #include <cstdlib>
4  #include <algorithm>
5  const int MAX = 100;
6
7  void init(std::list<int>& vec) {
8    for (unsigned int i = 0; i < MAX; ++i) {
9      vec.push_back( rand() % MAX );
10   }
11 }
12
13 void print(const std::list<int>& vec) {
14   std::list<int>::const_iterator ptr = vec.begin();
15   while ( ptr != vec.end() ) {
16     std::cout << (*ptr)  << ", ";
17     ++ptr;
18   }
19   std::cout << std::endl;
20 }
21
22 void eraseMultiplesLess(std::list<int>& vec, int multiple, int bound) {
23   std::list<int>::iterator ptr = vec.begin();
24   while ( ptr != vec.end() && *ptr < bound) {
25     if ( (*ptr) % multiple == 0 ) {
26       ptr = vec.erase(ptr);
27     }
28     else ++ptr;
29   }
30 }
31
32 int main() {
33   std::list<int> vec;
34   init(vec);
35   vec.sort();
36   eraseMultiplesLess(vec, 5, 50);
37   print(vec);
38 }
```

6. (15 points) The following program maintains a list of students. Write a function object to enable studentList, declared on line #35, to be sorted. Then, supply the line of code (line #38) to sort the studentList.

```
 1 #include <iostream>
 2 #include <list>
 3 #include <cstdlib>
 4 const int MAX = 100;
 5
 6 class Student {
 7 public:
 8   Student() : number(0) { }
 9   Student(int n) : number(n) {
10   }
11   Student(const Student& a) : number(a.number) { }
12   int getStudent() const { return number; }
13   bool operator<(const Student& rhs) const { return number < rhs.number; }
14 private:
15   int number;
16 };
17 std::ostream& operator<<(std::ostream& out, const Student* number) {
18   return out << number->getStudent();
19 }
20
21 class StudentLess{
22 public:
23   bool operator()(const Student* lhs, const Student* rhs) const {
24     //return lhs->getStudent() < rhs->getStudent();
25     return *lhs < *rhs;
26   }
27 };
28
29
30 void init(std::list<Student*> & numberList) {
31   for (unsigned int i = 0; i < MAX; ++i) {
32     numberList.push_back( new Student(rand() % MAX) );
33   }
34 }
35
36 void print(const std::list<Student*> & numberList) {
37   std::list<Student*>::const_iterator ptr = numberList.begin();
38   while ( ptr != numberList.end() ) {
39     std::cout << (*ptr)  << ", ";
40     ++ptr;
41   }
42   std::cout << std::endl;
43 }
44
45
46 int main() {
47   std::list<Student*> studentList;
48   init(studentList);
49   studentList.sort(StudentLess());
50   print(studentList);
}
```

7. (30 points) The program segment below illustrates object pooling, where a pool of resources is maintained in a list. When the user requires a resource, the pool is checked and if one is available the resource is reset and returned to the user; if the pool is empty a new resource is created and returned to the user. Usually, in object pooling, the resources are not deleted until the end of the program. Write function getResource, whose prototype is listed on line #20. Also, write a destructor so that the resources are deleted at the end of the program. Add any code you need to make sure that the resources are deleted at the end of the program.

```
 1  // This example was originally written by Radek Pazdera
 2  // https://gist.github.com/pazdera/1124832
 3  // Then modified by Brian Malloy on 4/19/2014
 4
 5  #include <iostream>
 6  #include <string>
 7  #include <list>
 8
 9  class Resource {
10  friend class ObjectPool;
11  public:
12    int getValue() const { return value; }
13  private:
14    Resource(int n) : value(n) { }
15    void reset(int n) { value = n; }
16    int value;
17  };
18
19  class ObjectPool {
20  public:
21    static ObjectPool& getInstance() {
22      static ObjectPool instance;
23      return instance;
24    }
25    ~ObjectPool() {
26      std::list<Resource*>::iterator ptr = freeList.begin();
27      while ( ptr != freeList.end() ) {
28        delete *ptr;
29        ++ptr;
30      }
31    }
32
33    Resource* getResource(int n) {
34      if (freeList.empty()) {
35        std::cout << ”Making a new resource” << std::endl;
36        return new Resource(n);
37      }
38      else {
39        std::cout << ”Reusing resource” << std::endl;
40        Resource* resource = freeList.front();
41        resource->reset(n);
42        freeList.pop_front();
43        return resource;
44      }
```

```
  }

  void reallocateResource(Resource* object) {
    freeList.push_back(object);
  }

private:
  // This approach doesn't maintain a resource list!
  std::list<Resource*> freeList;
  ObjectPool() {}
};

int main() {
  ObjectPool& pool = ObjectPool::getInstance();
  Resource* one;
  Resource* two;
  one = pool.getResource(10);
  std::cout << "one = " << one->getValue() << std::endl;
  two = pool.getResource(20);
  std::cout << "two = " << two->getValue() << std::endl;
  pool.reallocateResource(one);
  pool.reallocateResource(two);

  one = pool.getResource(99);
  std::cout << "one = " << one->getValue() << std::endl;
  two = pool.getResource(100);
  std::cout << "two = " << two->getValue() << std::endl;
  pool.reallocateResource(one);
  pool.reallocateResource(two);
}
```