1. (10 points) The following program generates a positive random number and then uses the C++ ternary operator to possibly make the number negative. Refactor the program so that it uses a *lambda* function rather than the ternary operator to possibly **make** the number negative.

```
1  #include <iostream>
2  #include <ctime>
3  // [capture clause] (parameters) -> return-type {body}
4
5  int main() {
6    srand( time(0) );
7    int x = rand() % 100;
8
9    x = x*(rand()%2?-1:1);
10   std::cout << "x " << x << std::endl;
11 }
```

---

2. (10 points) The following program reads and prints a character. Extend the program so that it uses a lambda function to determine if the character is an upper case letter and then prints an appropriate message. (You may not use a built-in function). Possible output:

```
A
is: 1
malloy@aramis:~/pubgit/4160-2017/quiz/3/code/isLetter/soln$ r
8
is: 0
```

```
1  #include <iostream>
2  #include <ctime>
3  // [capture clause] (parameters) -> return-type {body}
4
5  int main() {
6    char ch;
7    std::cin >> ch;
8    std::cout << ch << std::endl;
9
10 }
```

3. (10 points) Write function display, which prints the key and value for each item in a map (use while or ranged for). Then, assuming your display works, give the output for the following program.

```
#include <iostream>
#include <string>
#include <map>

int main() {
  std::map<std::string, int> pokemon;
  pokemon["Noctis"] = 2750;
  pokemon["Pronto"] = 1725;
  pokemon["Noctis"] = 1750;
  std::cout << pokemon.size() << std::endl;
  display( pokemon );
}
```

---

4. (10 points) The program below gives the following error:

```
main.cpp:16:9: error: class Bird has no member named swim
   bird->swim();
```

Write code to illustrate two ways to fix the program so that line #16 prints "I can swim"; You may not change Bird or Penguin.

```
#include <iostream>
#include <string>
class Bird {
public:
  Bird(const std::string & s) : species(s) {}
private:
  std::string species;
};
class Penguin : public Bird {
public:
  Penguin(const std::string & species) : Bird(species) {}
  void swim() const { std::cout << "I can swim" << std::endl; }
};
int main() {
  Bird * bird = new Penguin("penguin");
  bird->swim();
}
```

5. (10 points) The program below builds a list of integers in numbers. Remove from the list all even numbers whose preceeding number in the list is a multiple of 3. To remove the numbers you may only use erase and you may not use any functions in algorithm. Possible output:

```
Before: 25  65  65  90  38  11  30  28  89  11  34  39  97  4  35  34  99  24  90  1
After:  25  65  65  90  11  30  89  11  34  39  97  4  35  34  99  1
```

```
1  #include <iostream>
2  #include <list>
3
4  void display(const std::list<int>& numbers) {
5    for ( auto x : numbers ) {
6      std::cout << x << "  ";
7    }
8    std::cout << std::endl;
9  }
10
11 int main() {
12   std::list<int> numbers;
13   for (int i = 0; i < 20; ++i) {
14     numbers.push_back( rand() % 100 );
15   }
16   display(numbers);
17 }
```

6. (30 points) For the following program:

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <list>
4
5  class Number {
6  public:
7    Number() : n( rand()%100 ) {}
8    int getNumber() const { return n; }
9  private:
10   int n;
11 };
12
13 int main() {
14   std::list<Number*> numbers;
15   for (unsigned int i = 0; i < 100; ++i) {
16     numbers.push_back( new Number() );
17   }
18
19   // ------------------------------------------------------------
20
21   // ------------------------------------------------------------
22   removeMedian( numbers );
23 }
```

(a) Write a functor that can sort numbers, low to high. Use the functor to sort on Line #19.

(b) Write a lambda function that can sort numbers, low to high. Use lambda to sort on Line #21.

(c) Write function removeMedian, which removes the median number using the following algorithm, attributed to Dr. Dean, on a sorted list: use two iterators, a fast iterator and a slow iterator. Move the slow iterator by one element at a time, but move the fast iterator by two elements at a time. When the fast iterator points to the end of the list, the slow iterator points to the median number in the list. Remove this median number.

7. (20 points) The *object pool* pattern improves memory management by reusing objects from a list or pool instead of allocating and deallocating them individually. The program on the following page simulates an *object pool* of Numbers.

   (a) Write a destructor (declared on line #16) for NumberPool, which deallocates memory in NumberPool, and

   (b) Write NumberPool::makeNumber (declared on line #17), which returns a number from the pool, either by returning one from the free list or by making a new Number. If you reuse a number from the free list, don't forget to reset its value using Number::reset.

   Function NumberPool::processNumbers helps manage the pool by checking the value stored in each Number; if that value is zero the Number is moved from NumberPool::numberList to NumberPool::freeList.

```
#include <cstdlib>
#include <iostream>
#include <list>
class Number {
public:
  Number() : n( rand()%5+1 ) {}
  int getN() const { return n; }
  void decrement() { --n; }
  void reset() { n = rand()%2+1; }
private:
  int n;
};
class NumberPool {
public:
  NumberPool() : numberList(), freeList(), sum(0) {}
  ~NumberPool();
  Number* makeNumber();
  void processNumbers();
  void display() const {
    std::cout << "sum is " << sum << std::endl;
  }
  void update();
private:
  std::list<Number*> numberList;
  std::list<Number*> freeList;
  int sum;
};

void NumberPool::update() {
  for ( auto n : numberList ) {
    n->decrement();
  }
}

void NumberPool::processNumbers() {
  std::list<Number*>::iterator ptr = numberList.begin();
  while ( ptr != numberList.end() ) {
    if ( (*ptr)->getN() == 0 ) {
      freeList.push_back(*ptr);
      ptr = numberList.erase(ptr);
    }
    else ++ptr;
  }
}

int main() {
  int sum = 0;
  NumberPool pool;
  int duration = rand() % 10 + 5;
  for (int i = 0; i < 3; ++i) {
    pool.makeNumber();
  }
  while ( duration ) {
    --duration;
    pool.update();
    pool.processNumbers();
  }
}
```