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.

```
#include <iostream>
#include <ctime>
// [capture clause] (parameters) -> return-type {body}

int main() {
  srand( time(0) );
  int x = rand() % 100;

  std::cout << ”x before ” << x << std::endl;
  auto maybe = [](int& x){ if (rand()%2) return x=-x; else return x; };
  std::cout << ”x during ” << maybe(x) << std::endl;
  std::cout << ”x after ” << x << std::endl;
}
```

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
```

```
#include <iostream>
#include <ctime>
// [capture clause] (parameters) -> return-type {body}

int main() {
  char ch;
  std::cin >> ch;
  std::cout << ch << std::endl;
  //bool flag = isalpha(ch);
  //if ( flag )
    //std::cout << ”letter” << std::endl;
  //else
    //std::cout << ”NOT letter” << std::endl;
  auto isletter = [](char ch) { return (ch >= ’A’ && ch <= ’Z’); };
  std::cout << ”is: ” << isletter(ch) << std::endl;

}
```

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>

void display( const std::map<std::string, int>& pokemon ) {
  for ( auto it : pokemon ) {
    std::cout << it.first << ”, ” << it.second << std::endl;
  }
}

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();
```

Illustrate two ways to fix the program so that line #16 prints “I can swim”.

```
1  #include <iostream>
2  #include <string>
3
4  class Bird {
5  public:
6    virtual ~Bird() {}
7    Bird(const std::string & s) : species(s) {}
8    virtual void swim() const { std::cout << ”I␣can't␣swim” << std::endl; }
9  private:
10   std::string species;
11 };
12
13 class Penguin : public Bird {
14 public:
15   Penguin(const std::string & species) : Bird(species) {}
16   void swim() const { std::cout << ”I␣can␣swim” << std::endl; }
17 };
18
19 int main() {
20   Bird * bird = new Penguin(”penguin”);
21   dynamic_cast<Penguin*>(bird)->swim();
22 }
```

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

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

```
#include <ctime>
#include <iostream>
#include <list>

void display(const std::list<int>& numbers) {
  for ( auto x : numbers ) {
    std::cout << x << ”  ”;
  }
  std::cout << std::endl;
}

void removeEvenAfterMult3( std::list<int>& numbers ) {
  if ( numbers.size() == 0 ) return;
  std::list<int>::iterator it = numbers.begin();
  int number = *it;
  ++it;
  while ( it != numbers.end() ) {
    if ( number%3==0 && *it%2==0 ) {
      it = numbers.erase(it);
    }
    else {
      number = *it;
      ++it;
    }
  }
}

int main() {
  srand( time(0) );
  std::list<int> numbers;
  for (int i = 0; i < 20; ++i) {
    numbers.push_back( rand() % 100 );
  }
  display(numbers);
  removeEvenAfterMult3(numbers);
  display(numbers);
}
```

6. (30 points) For the following program:

   (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.

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <list>

class Number {
public:
  Number() : n( rand()%100 ) {}
  int getNumber() const { return n; }
private:
  int n;
};
std::ostream& operator<<(std::ostream& out, const Number* n) {
  return out << n->getNumber();
}

struct CompareLess {
  bool operator()(const Number* lhs, const Number* rhs) const {
    return lhs->getNumber() < rhs->getNumber();
  }
};

void init(std::list<Number*>& numbers) {
  srand( time(0) );
  unsigned int max = rand()%10+5;
  for (unsigned int i = 0; i < max; ++i) {
    numbers.push_back( new Number() );
  }
}

void display(const std::list<Number*>& numbers) {
  for ( auto n : numbers ) {
    std::cout << n << ”,␣”;
  }
    std::cout << std::endl;
}

void removeMedian(std::list<Number*>& n) {
  std::list<Number*>::iterator slow = n.begin();
  std::list<Number*>::iterator fast = n.begin();
  while ( fast != n.end() && ++fast != n.end() ) {
    ++fast;
    ++slow;
  }
  std::cout << ”median␣is:␣” << (*slow) << std::endl;
  delete *slow;
  n.erase( slow );
}

int main() {
  std::list<Number*> numbers;
  init(numbers);
  display(numbers);

  numbers.sort(CompareLess());

  auto compare = [](const Number* l, const Number* r){
    return l->getNumber() < r->getNumber();
  };
  numbers.sort(compare);

  display(numbers);
  removeMedian( numbers );
  display(numbers);
  for ( auto n : numbers ) {
    delete n;
  }
}
```

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.

```
 1 #include <cstdlib>
 2 #include <ctime>
 3 #include <iostream>
 4 #include <list>
 5
 6 class Number {
 7 public:
 8   Number() : n( rand()%5+1 ) {}
 9   int getN() const { return n; }
10   void decrement() { --n; }
11   void reset() { n = rand()%2+1; }
12 private:
13   int n;
14 };
15
16 class NumberPool {
17 public:
18   NumberPool() : numberList(), freeList(), sum(0) {}
19   ~NumberPool();
20   void makeNumber();
21   void processNumbers();
22   void display() const {
23     std::cout << ”sum is ” << sum << std::endl;
24   }
25   void update();
26 private:
27   std::list<Number*> numberList;
28   std::list<Number*> freeList;
29   int sum;
30 };
31
32 NumberPool::~NumberPool() {
33   std::cout << ”Deleting:” << std::endl;
34   display();
35   for( Number* n : numberList ) {
36     delete n;
37   }
38   for( Number* n : freeList ) {
39     delete n;
40   }
41 }
```

```

void NumberPool::makeNumber() {
  Number* number;
  if ( freeList.empty() ) {
    number = new Number;
    numberList.push_back( number );
  }
  else {
    number = freeList.front();
    number->reset();
    numberList.push_back(number);
    freeList.erase( freeList.begin() );
  }
  sum += number->getN();
}

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