1. (10 points) Give the output for the following program.

```
#include <iostream>
#include <functional>
// [capture clause] (parameters) -> return-type {body}
int main() {
  std::function<int(int)> compute;
  compute = [&compute](int x) {
    if(x==1 || x==2) return 1; else return compute(x-1)+compute(x-2);
  };
  std::cout << ”compute(5) = ” << compute(5) << std::endl;
}
```

```
compute(5) = 5
```

---

2. (10 points) Give the output for the following program. Make note of parameter transmission modes, and note that number, in display, is a static local variable.

```
#include <iostream>
#include <functional>
void display(std::function<int(int&)> incr) {
  static int number = 77;
  std::cout << incr(number) << std::endl;
}
int main() {
  int x = 7, y = 7;
  auto incrX = [](int& x){ return ++x; };
  auto incrY = [](int y) { return ++y; };

  incrX(x);
  std::cout << ”x = ” << x << std::endl;

  incrY(y);
  std::cout << ”y = ” << y << std::endl;

  display(incrX);
  display(incrX);
}
```

```
x = 8
y = 7
78
79
```

---

3. (10 points) Give the output for the following program.

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

int main() {
  int x = 17;
  int y = 109;
  const int * q = &x;
  int& ref = x;
  ref = y;
  std::cout << x << std::endl;
  std::cout << ref << std::endl;
}
```

```
109
109
```

4. (15 points) Write functions printRadius and removeRectangles, used on lines #36 and #37. printRadius prints the radius of the circles and removeRectangles removes all the rectangles in list shapes.

```
 1 #include <iostream>
 2 #include <list>
 3 class Shape {
 4 public:
 5   virtual ~Shape() {}
 6   virtual void display() const = 0;
 7 };
 8 class Circle : public Shape {
 9 public:
10   Circle(float r) : Shape(), radius(r) {}
11   float getRadius() const { return radius; }
12   virtual void display() const { std::cout << ”circle” << std::endl; }
13 private:
14   float radius;
15 };
16 class Rectangle : public Shape {
17 public:
18   Rectangle(int w, int h) : Shape(), width(w), height(h) {}
19   virtual void display() const { std::cout << ”rectangle” << std::endl; }
20 private:
21   int width;
22   int height;
23 };
24
25 // Notice the parameter transmission mode: Never pass
26 // containers by value:
27 void printShapes(const std::list<Shape*>& shapes) {
28   for ( const Shape* const shape : shapes ) {
29     shape->display();
30   }
31 }
32
33
34 // Notice the parameter transmission mode: Never pass
35 // containers by value:
36 void cleanUp(std::list<Shape*>& shapes) {
37   for ( Shape* const shape : shapes ) {
38     delete shape;
39   }
40 }
41
42
43 // Notice the parameter transmission mode: Never pass
44 // containers by value:
45 void removeRectangles(std::list<Shape*>& shapes) {
46   auto it = shapes.begin();
47   while ( it != shapes.end() ) {
48     Rectangle* rect = dynamic_cast<Rectangle*>(*it);
49     if ( rect ) {
50       delete rect;
51       it = shapes.erase( it );
52     }
53     else ++it;
54   }
55 }
56
```

```
// Notice the parameter transmission mode: Never pass
// containers by value:
void printRadius1(const std::list<Shape*>& shapes) {
  for ( Shape* const shape : shapes ) {
    Circle* circle = dynamic_cast<Circle*>(shape);
    if ( circle ) {
      std::cout << circle->getRadius() << std::endl;
    }
  }
}


// Notice the parameter transmission mode: Never pass
// containers by value:
void printRadius2(const std::list<Shape*>& shapes) {
  for ( Shape* const shape : shapes ) {
    if ( dynamic_cast<Circle*>(shape) ) {
      std::cout << static_cast<Circle*>(shape)->getRadius() << std::endl;
    }
  }
}

int main() {
  std::list<Shape*> shapes;
  const int n = rand()%25 + 5;
  for ( int i = 0; i < n; ++i ) {
    if ( rand()%2 ) {
      shapes.push_back( new Circle(rand()%25+5) );
    }
    else {
    shapes.push_back( new Rectangle(rand()%100, rand()%100) );
    }
  }
  printRadius1(shapes);
  removeRectangles(shapes);
  printShapes(shapes);
  cleanUp(shapes);
}
```

5. (15 points) Write an overloaded assignment operator for class Derived.

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

class Base {
};

class Derived : Base {
public:
  Derived() : name(new char[1]) {
    name[0] = '\0';
  }
  Derived(const char* n) : name(new char[strlen(n)+1]) {
    strcpy(name,  n);
  }
  Derived& operator=(const Derived& rhs) {
    if ( this == &rhs ) return * this;
    Base::operator=(rhs);
    delete [] name;
    name = new char[strlen(rhs.name)+1];
    strcpy(name, rhs.name);
    return *this;
  }
  const char* getName() const { return name; }
private:
  char * name;
};

int main() {
  Derived d("bill"), e;
  e = d;
  std::cout << e.getName() << std::endl;
}
```

---

6. (15 points)

   (a) The program below fails to compile. Fix the program by modifying class Player, without changing function display, so that the program compiles and display works. The error message is:

```
main.cpp:12:27: error: passing const Player as this argument discards
qualifiers [-fpermissive]
   return out << p.getName();
                           ^
main.cpp:7:16: note:   in call to std::__cxx11::string& Player::getName()
   std::string& getName() { return name; }
                ^
```

   (b) Write an output operator for class Player so that line #19 prints the player's name.

```
1  #include <iostream>
2  #include <string>
3
4  class Player {
```

```
public:
  Player(const std::string n) : name(n) {}
  std::string& getName() { return name; }
private:
  std::string name;
};
std::ostream& operator<<(std::ostream& out, const Player& p) {
  return out << p.getName();
}

void display(const Player& player) {
  std::cout << ”Name: ” << player.getName() << std::endl;
}

int main() {
  Player b(”Babe Ruth”);
  display(b);
  std::cout << b << std::endl;
}
```

7. (25 points) Write either a functor or a lambda function for each of the following:

   (a) (10 pts) So that spriteList is sorted; write code on line #35 to use the functor/lambda to sort spriteList (low to high);

   (b) (15 pts) So that spriteList is searched, using find or find_if, for an integer; write code on line #44 to use the functor/lambda to search for number and print an appropriate message.

   Sample output might be:

```
14, 10, 24, 6, 19, 11, 22, 9, 16, 12, 4, 18, 13, 14, 3, 17, 14, 16, 23, 19,
3, 4, 6, 9, 10, 11, 12, 13, 14, 14, 14, 16, 16, 17, 18, 19, 19, 22, 23, 24,
16 is in list
```

```
1  #include <iostream>
2  #include <list>
3  #include <cstdlib>
4  #include <ctime>
5  #include <algorithm>
6
7  class Sprite {
8  public:
9    Sprite() : scale(0) { }
10   explicit Sprite(int n) : scale(n) { }
11   int getScale() const { return scale; }
12 private:
13   float scale;
14 };
15 std::ostream& operator<<(std::ostream& out, const Sprite* sprite) {
16   return out << sprite->getScale();
17 }
18
19 class SpriteLess{
20 public:
21   bool operator()(const Sprite* lhs, const Sprite* rhs) const {
22     return lhs->getScale() < rhs->getScale();
23   }
24 };
25
26 class Target{
27 public:
28   Target( int n ) : scale(n) {}
29   bool operator()(const Sprite* rhs) const {
30     return scale == rhs->getScale();
31   }
32 private:
33   int  scale;
34 };
35
36 void init(std::list<Sprite*> & spriteList) {
37   for (unsigned int i = 0; i < 20; ++i) {
38     spriteList.push_back( new Sprite(rand()%25) );
39   }
40 }
41
42 void print(const std::list<Sprite*> & spriteList) {
43   for ( const Sprite* n : spriteList ) {
44     std::cout << n  << ”, ”;
45   }
```

```
  std::cout << std::endl;
}

int main() {
  srand( time(0) );
  std::list<Sprite*> spriteList;
  init(spriteList);
  print(spriteList);
  spriteList.sort( SpriteLess() );
  print(spriteList);

  int number = rand()%25;

  // The next two lines use the function Target, defined above
  //std::list<Sprite*>::iterator it =
    //find_if( spriteList.begin(), spriteList.end(), Target(number) );

  auto fun = [number](const Sprite* s) {
    return s->getScale() == number;
  };
  std::list<Sprite*>::iterator it =
    find_if( spriteList.begin(), spriteList.end(), fun );

  if (  it == spriteList.end() ) {
    std::cout << number << ” not in list” << std::endl;
  }
  else {
    std::cout << number << ” is in list” << std::endl;
  }

}
```