1. (25 points) For the following program,

   (a) Write a function object that can be used to sort numbers from low to high.
   (b) Use the function object to sort list numbers on line #20.
   (c) Write function removeMedian, which removes the median number using the following algorithm 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.

```
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   removeMedian( numbers );
22 }
```

2. (5 points) What is the reference count for label at the end of the following program segment:

```
cocos2d::Label * label =
  cocos2d::Label::createWithTTF("First Animation",
                                "fonts/Marker Felt.ttf", 24);
label->setPosition(cocos2d::Vec2(origin.x + visibleSize.width/2,
  origin.y + visibleSize.height - label->getContentSize().height));
label->retain();
```

3. (10 points) Write function removeMultiples, which accepts 2 parameters, a list and an integer; the function should remove all multiples of the integer parameter.

```
#include <iostream>
#include <list>
#include <cstdlib>
const int MAX = 100;

void init(std::list<int> & sprites) {
  for (unsigned int i = 0; i < MAX; ++i) {
    sprites.push_back( rand()%100 );
  }
}
int main() {
  std::list<int> sprites;
  init(sprites);
  removeMultiples(sprites, rand()%5+1);
}
```

4. (10 points) The following program initializes a vector, names, with names of movies. It also initializes a map, movies, with the names of movies as the key, and the number of stars that the movie received as the value. For a map, first points to the key, and second points to the value stored for key. Give the output of the program.

```cpp
#include <iostream>
#include <map>
#include <vector>
const int MAX = 5;

void initNames( std::vector<std::string>& names ) {
  names.push_back( "Spider-man" );
  names.push_back( "Bloodpool" );
  names.push_back( "Thor" );
}

void printMovies( std::map<std::string, int> & movies ) {

  std::cout << "The size of the map is: " << movies.size() << std::endl;

  std::map<std::string, int>::const_iterator ptr = movies.begin();
  while ( ptr != movies.end() ) {
    std::cout << ptr->first << ", " << ptr->second << std::endl;
    ++ptr;
  }
}

void initMovies( std::map<std::string, int> & movies,
                 const std::vector<std::string>& names ) {
  int count = 0;
  for ( unsigned int i = 0; i < MAX; ++i ) {
    movies[names[count]] = i;
    count = (count + 1) % names.size();
  }
}

int main() {
  std::vector<std::string> names;
  std::map<std::string, int>  movies;

  initNames(names);
  initMovies(movies, names);
  printMovies(movies);
}
```

5. (10 points) The program below gives the following error:

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

   Rewrite line #16 so that the program compiles and prints: “I can swim”

```
1  #include <iostream>
2  #include <string>
3  class Bird {
4  public:
5    Bird(const std::string & s) : species(s) {}
6  private:
7    std::string species;
8  };
9  class Penguin : public Bird {
10 public:
11   Penguin(const std::string & species) : Bird(species) {}
12   void swim() const { std::cout << "I␣can␣swim" << std::endl; }
13 };
14 int main() {
15   Bird * bird = new Penguin("penguin");
16   bird->swim();
17 }
```

6. (20 points) The following program illustrates a polymorphic list of shapes.

   (a) Write findAverageArea, line #38, which returns the average area of all shapes in list.

   (b) Write function findAverageAreaSquares, line #39, which finds the average area of the squares in the list. (Hint: use dynamic_cast).

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <list>
4  const float PI = 3.14;
5
6  class Shape {
7  public:
8    virtual float getArea() const = 0;
9    virtual ~Shape() {}
10 };
11 class Circle : public Shape {
12 public:
13   Circle(int r) : radius( r ) {}
14   float getArea() const { return PI*radius*radius; }
15 private:
16   float radius;
17 };
18 class Square : public Shape {
19 public:
20   Square(int s) : side( s ) {}
21   float getArea() const { return side*side; }
22 private:
23   float side;
24 };
25 void init( std::list<Shape*> & shapes ) {
26   for (unsigned int i = 0; i < 100; ++i) {
27     if ( rand()%2 ) {
28       shapes.push_back( new Circle(rand()% 25) );
29     }
30     else {
31       shapes.push_back( new Square(rand()% 25) );
32     }
33   }
34 }
35 int main() {
36   std::list<Shape*> shapes;
37   init( shapes );
38   std::cout << ”avg:␣” << findAverageArea( shapes ) << std::endl;
39   std::cout << ”avg␣squares:␣” << findAverageAreaSquares( shapes ) << std::endl;
40 }
```

7. (10 points) Use stringstream to write function init, which initializes vector numbers to integers between 0 and 99 represented as strings.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
const int MAX = 100;

void print( const std::vector<std::string> & numbers ) {
  for (unsigned int i = 0; i < numbers.size(); ++i) {
    std::cout << numbers[i] << std::endl;
  }
}

int main() {
  std::vector<std::string> numbers;
  init( numbers );
  print( numbers );
}
```

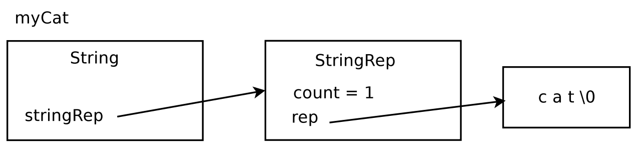

Figure 1: **Illustration of Reference Counting**.

8. (10 points) The following main program uses reference counted strings, listed below and on the following page. Figure 1 illustrates the state of the main program after execution of line #5. Modify the figure so that it illustrates the program after execution of line #6.

```
1 #include <iostream>
2 #include ”./string.h”
3
4 main() {
5   String mycat(”cat”);
6   String copycat(mycat);
7 }
```

```
#include ”stringRep.h”
#include <iostream>

class String {
public:
  String();
  String(const char *s);
  String(const String& s);
  ~String();
  const char* getRep() const { return stringRep->getRep(); }
  int getCount() const { return stringRep->getCount(); }
private:
  StringRep *stringRep;
};
String::String() : stringRep(new StringRep) {
  stringRep->count=1;
}
String::String(const String& s) : stringRep(s.stringRep) {
  ++(*stringRep);
}
String::String(const char *s) : stringRep(new StringRep(s)) {
  stringRep->count = 1;
}
String::~String() {
  if (--(*stringRep) <= 0) {
    delete stringRep;
  }
}
```

```
#include <string.h>
#include <iostream>

class String;
class StringRep {
friend class String;
private:
  StringRep();
  StringRep(const StringRep& s);
  ~StringRep();
  StringRep(const char *s);
  const char* getRep() const { return rep; }
  int getCount() const { return count; }
private:
  char *rep;
  int count;
};
StringRep::StringRep() : rep(new char[1]) {
  rep[0] = '\0';
}
StringRep::StringRep(const char *s) {
  ::strcpy(rep=new char[::strlen(s)+1], s);
}
StringRep::StringRep(const StringRep& s)  {
  ::strcpy(rep=new char[::strlen(s.rep)+1], s.rep);
}
StringRep::~StringRep() {
  delete[] rep;
}
```