1. (10 points) Give the output, if any, for the program below. Explain why the program crashes.

```
#include <cstring>
#include <iostream>
class string {
public:
   string(const char* s) : buf(new char[strlen(s)+1]) { strcpy(buf, s); }
   ~string() { delete [] buf; }
   const char* getBuf() const { return buf; }
   void setBuf(const char* s) {
     delete [] buf;
     buf = new char[strlen(s)+1];
     strcpy(buf, s);
   }
private:
   char * buf;
};

int main() {
  string a("Bat"), b = a;
  b.setBuf("Antelope");
  std::cout << a.getBuf() << std::endl;
}
```

---

2. (10 points) Write function removeEvens.

```
#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);
  removeEvens(sprites);
}
```

3. (15 points) In the program below, sprites contains a list of Sprite objects that need to be sorted from low to high.

   (a) Add code (an overloaded operator) so that the sort algorithm works;

   (b) overload an output operator so that the print function works.

```
1  #include <iostream>
2  #include <list>
3  #include <algorithm>
4  #include <limits>
5  const int MAX = 20;
6  float getRandInRange(int min, int max) {
7    return min + (rand()/(std::numeric_limits<int>::max()+1.0f))*(max-min);
8  }
9
10 class Sprite {
11 public:
12   Sprite() : scale(0) { }
13   Sprite(float n) : scale(n) { }
14   Sprite(const Sprite& a) : scale(a.scale) { }
15   float getScale() const { return scale; }
16 private:
17   float scale;
18 };
19
20 void init(std::list<Sprite> & sprites) {
21   for (unsigned int i = 0; i < MAX; ++i) {
22     sprites.push_back( getRandInRange(0, 1) );
23   }
24 }
25
26 void print(const std::list<Sprite> & sprites) {
27   std::list<Sprite>::const_iterator ptr = sprites.begin();
28   while ( ptr != sprites.end() ) {
29     std::cout << (*ptr)  << ", ";
30     ++ptr;
31   }
32   std::cout << std::endl;
33 }
34
35 int main() {
36   std::list<Sprite> sprites;
37   init(sprites);
38   print(sprites);
39   sprites.sort();
40   print(sprites);
41 }
```

4. (20 points) In the program below, sprites contains a list of pointers to Sprite objects that need to be sorted from low to high.

   (a) (15 points) Add code so that the sort algorithm works;

   (b) (5 points) overload output so that the print function, lines #26 to #33, works.

```
1  #include <iostream>
2  #include <list>
3  #include <algorithm>
4  #include <limits>
5  const int MAX = 20;
6  float getRandInRange(int min, int max) {
7    return min + (rand()/(std::numeric_limits<int>::max()+1.0f))*(max-min);
8  }
9
10 class Sprite {
11 public:
12   Sprite() : scale(0) { }
13   Sprite(float n) : scale(n) { }
14   Sprite(const Sprite& a) : scale(a.scale) { }
15   float getScale() const { return scale; }
16 private:
17   float scale;
18 };
19
20 void init(std::list<Sprite*> & sprites) {
21   for (unsigned int i = 0; i < MAX; ++i) {
22     sprites.push_back( new Sprite(getRandInRange(0, 1)) );
23   }
24 }
25
26 void print(const std::list<Sprite*> & sprites) {
27   std::list<Sprite*>::const_iterator ptr = sprites.begin();
28   while ( ptr != sprites.end() ) {
29     std::cout << (*ptr) << ", ";
30     ++ptr;
31   }
32   std::cout << std::endl;
33 }
34
35 int main() {
36   std::list<Sprite*> sprites;
37   init(sprites);
38   print(sprites);
39
40   //------------------------------------------------
41   print(sprites);
42 }
```

5. (15 points) An STL map is a dictionary of (*key*, *value*) pairs, with quick lookup of a *value* for a *key*. The program below contains an inheritance hierarchy with base class A, derived class B, and an STL map<string, A*>.

   (a) Add code in main to access the map to print the *value* for *key* "starlord".

   (b) Write lines of code that use two different approaches to *casting down the inheritance hierarchy* to print the power of "spiderman" by accessing a *key* in the *map*.

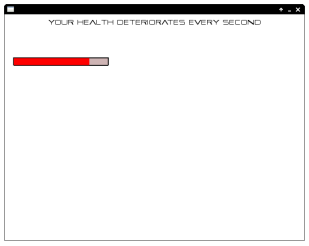
```
#include <iostream>
#include <string>
#include <map>

class A {
public:
  A(const std::string& n) : name(n) {}
  virtual ~A() {}
  std::string getName() const { return name; }
private:
  std::string name;
};

class B : public A {
public:
  B(const std::string& n, std::string x) : A(n), power(x) {}
  std::string getPower() const { return power; }
private:
  std::string power;
};

int main( ) {
  std::map<std::string, A*> heros;
  heros["superman"] = new B("Clark Kent", "fly");
  heros["starlord"] = new A("Peter Quill");
  heros["spiderman"] = new B("Peter Parker", "climb");
  heros["batman"] = new A("Bruce Wayne");
  // Print spiderman's super power using static_cast and dynamic_cast:
}
```

6. (15 points) The following program contains class Sprite, and class SpriteDB, which contains a list of Sprite*. The program has memory leaks.

   (a) Write a function to remove the leaks.

   (b) Write a function object so that the sort on line #26 would sort sprites

```
1  #include <iostream>
2  #include <list>
3  #include <cstdlib>
4  #include <algorithm>
5  #include <limits>
6  const int MAX = 20;
7
8  class Sprite {
9  public:
10   Sprite() : id(0) { }
11   Sprite(float n) : id(n) { }
12   Sprite(const Sprite& a) : id(a.id) { }
13   float getId() const { return id; }
14 private:
15   float id;
16 };
17 class SpriteDB {
18 public:
19   SpriteDB() {
20     for (unsigned int i = 0; i < MAX; ++i) {
21       sprites.push_back( new Sprite( rand() % 100 ) );
22     }
23   }
24   void print() const;
25
26   void sort() { //___code to sort the list of sprites_______; }
27 private:
28   std::list<Sprite*> sprites;
29 };
30
31 int main() {
32   SpriteDB sprites;
33   sprites.sort();
34   sprites.print();
35 }
```



(a) Basic Health Bar

```
#include "vector2f.h"
#include "ioManager.h"
#include "aaline.h"

class HealthBar {
public:
  HealthBar( ... ) :
  RED(SDL_MapRGB(screen->format, 0xff, 0x00, 0x00)),
  GRAY(SDL_MapRGB(screen->format, 0xce, 0xb4, 0xb4)),
  BLACK(SDL_MapRGB(screen->format, 0x00, 0x00, 0x00),
  color(RED) { }

  void draw() const;
  void update(Uint32);
private:
  const Uint32 RED;
  const Uint32 GRAY;
  const Uint32 BLACK;
  const Uint32 color;
};
```

(b) Start of Health Bar Class

Figure 1: The design and implementation of a Health Bar Class

---

7. (15 points) Figure 1a illustrates a basic health bar, and Figure 1b lists some code for the beginning of the design and implementation of a reconfigurable HealthBar class by passing parameters to the constructor to specify position, width and height of the HUD. Demonstrate your ability to design and implement a reconfigurable class in C^{++} by writing (1) the constructor, (2) draw function, and (3) an update function for the HealthBar class.

   The prototypes for the functions in aaline and IOManager are:

```
void Draw_Pixel(SDL_Surface* s, int x, int y, uint8_t r, uint8_t g, uint8_t b, uint8_t a);
void Draw_AALine(SDL_Surface* screen, float x0, float y0, float x1, float y1, float thick,
                 uint8_t r, uint8_t g, uint8_t b, uint8_t a);
void Draw_AALine(SDL_Surface* screen, float x0, float y0, float x1, float y1, float thick, uint32_t color);
void Draw_AALine(SDL_Surface* screen, float x0, float y0, float x1, float y1, uint32_t color);

class IOManager {
public:
  static IOManager* getInstance();
  SDL_Surface * getScreen() const { return screen; }
  SDL_Surface* loadAndSet(const char* filename, bool setcolorkey) const;
  void printMessageAt(const std::string& msg, Uint32 x, Uint32 y) const;
  void printMessageCenteredAt(const std::string& msg, Uint32 y) const;
  void printMessageValueAt(const std::string& msg, float value,
         Uint32 x, Uint32 y) const;
private:
  IOManager();
  SDL_Surface * screen;
  static IOManager* instance;
  TTF_Font *font;
};
```