1. (30 points)

   (a) Figures 1 and 2 represent listings for two classes, A and B, that have mutual dependencies. As listed below, the files won't compile. Add whatever code is needed to be able to compile and execute the two classes. (20 points)

   (b) Assuming that your solution is correct and that the classes will compile with function main, give the output of the program. (10 points)

```
class A {
public:
  A();
  int f();
  int g() const;
private:
  B* b;
};
```

```
#include <iostream>

A::A() : b(new B) { }
int A::f() { return b->g(this); }
int A::g() const { return 17; }
```

Figure 1: **a.h and a.cpp**: Header and implementation files for class A

```
class B {
public:
  B();
  int g(const A*) const;
};
```

```
#include <iostream>

B::B() {}
int B::g(const A* a) const { return a->g(); }
```

Figure 2: **b.h and b.cpp**: Header and implementation files for class B

```
1 #include <iostream>
2 #include "a.h"
3
4 int main() {
5   A a;
6   std::cout << a.f() << std::endl;
7   return 0;
8 }
```

2. (10 points) Write an overloaded bracket operator for class Sprite so that the output statement in function printLetter (line # 14) prints the letter r.

```
1  #include <iostream>
2  #include <cstring>
3  class Sprite {
4  public:
5    Sprite(const char* n) : name(new char[strlen(n)+1]) {
6      strcpy(name, n);
7    }
8    const char* getName() const { return name; }
9  private:
10   char* name;
11 };
12 void printLetter(const Sprite& sprite, int n) {
13   std::cout << "letter is " << letter[n] << std::endl;
14 }
15 int main() {
16   Sprite sprite("redorb");
17   printLetter(sprite, 0);
18 }
```

3. (20 points) Write an overloaded assignment operator for class ShootingSprite

```
1  class ShootingSprite : public Drawable {
2  public:
3    ShootingSprite(const char* name, const char* b) :
4      Drawable(name),
5      bulletName(new char[strlen(b)+1]) {
6      strcpy(bulletName, b);
7    }
8    virtual ~ShootingSprite() { delete [] bulletName; }
9    virtual void shoot() const { std::cout << bulletName << std::endl; }
10 private:
11   char* bulletName;
12 };
```

4. (10 points) The following program fails to compile with the following error message:

```
g++ question.cpp
question.cpp: In function std::ostream& operator<<(std::ostream&, const A&):
question.cpp:13: error: passing const A as this argument of int A::getNumber() discards qualifiers [-fpermissive]
```

Fix the code so that it compiles; you may add code but you cannot remove any code.

```
1  #include <iostream>
2  class A {
3  public:
4    A() : number(0) {}                    // default constructor
5    A(int n) : number(n) {}               // conversion constructor
6    A(const A& a) : number(a.number) {}   // copy constructor
7    int getNumber() { return number; }
8    void setNumber(int n) { number = n; }
9  private:
10   int number;
11 };
12 std::ostream& operator<<(std::ostream& out, const A& a) {
13   return out << a.getNumber();
14 }
15
16 int main() {
17   A a(17);
18   std::cout << a << std::endl;
19 }
```

5. (10 points) Give the output for the following program.

```
1  #include <iostream>
2  #include <cstring>
3
4  class string {
5  public:
6    string(const char* b) : buf(new char[strlen(b)+1]) {
7      strcpy(buf, b);
8    }
9    char* getBuf() const { return buf; }
10 private:
11   char* buf;
12 };
13
14 int main( ) {
15   string s("Elysium");
16   strcpy(s.getBuf(), "stuff");
17   std::cout << s.getBuf() << std::endl;
18   return 0;
19 }
```

6. (20 points) Class Manager, listed below, animates a sprite called orb, which can explode and then reappear.

   (a) (5 points) Explain the logic and reasoning behind lines #33, and #34; how does dynamic_cast work, and what is *short-circuit evaluation*.

   (b) (15 points) **Add any variables and code** that you might need to reuse the orb as either a Sprite or an ExplodingSprite; i.e., **write code** so that you avoid repeatedly using the new on lines #37 and #51, and the delete on lines #36 and #52, without introducing memory leaks.

```
1  #include <SDL.h>
2  #include "gamedata.h"
3  #include "ioManager.h"
4  #include "sprite.h"
5  #include "clock.h"
6  class Manager {
7  public:
8    Manager ();
9    ~Manager ();
10   void play();
11 private:
12   bool env;
13   Gamedata& gdata;
14   IOManager& io;
15   Clock& clock;
16   SDL_Surface *screen;
17   SDL_Surface *orbSurface;
18   Frame* orbFrame;
19   Sprite* orb;
20   void drawBackground() const;
21   Manager(const Manager&);
22   Manager& operator=(const Manager&);
23 };
```

```
1  #include <cmath>
2  #include "manager.h"
3  #include "explodingSprite.h"
4  Manager::~Manager() {
5    delete orbFrame; delete orb;
6    SDL_FreeSurface(orbSurface);
7  }
8  Manager::Manager() :
9    env( "SDL_VIDEO_WINDOW_POS=0,0"),
10   gdata( Gamedata::getInstance() ),
11   io( IOManager::getInstance() ),
12   clock( Clock::getInstance() ),
13   screen( io.getScreen() ),
14   orbSurface( io.loadAndSet(gdata.getXmlStr("orb/file"), true) ),
15   orbFrame( new Frame("orb", orbSurface) ),
16   orb( new Sprite("orb", orbFrame) ) {
17   atexit(SDL_Quit);
18 }
```

```
19 void Manager::drawBackground() const {
20   SDL_FillRect( screen, NULL, SDL_MapRGB(screen->format, 255, 255, 255) );
21   SDL_Rect dest = {0, 0, 0, 0};
22   SDL_BlitSurface( screen, NULL, screen, &dest );
23 }
24 void Manager::play() {
25   SDL_Event event;
26   bool done = false;
27   while ( not done ) {
28     drawBackground();
29     orb->draw();
30     io.printMessageCenteredAt("Press <e> to explode the red orb", 20);
31     orb->update(clock.getElapsedTicks());
32     SDL_Flip(screen);
33     ExplodingSprite* sprite = dynamic_cast<ExplodingSprite*>(orb);
34     if ( sprite && sprite->chunkCount() == 0 ) {
35       Vector2f position(orb->getPosition());
36       delete orb;
37       orb = new Sprite("orb",
38         position,
39         Vector2f(gdata.getXmlInt("orb/speed/x"),
40                  gdata.getXmlInt("orb/speed/y")),
41         orbFrame);
42     }
43     SDL_PollEvent(&event);
44     if (event.type ==  SDL_QUIT) { break; }
45     if(event.type == SDL_KEYDOWN) {
46       switch ( event.key.keysym.sym ) {
47         case SDLK_ESCAPE : done = true; break;
48         case SDLK_e      : {
49           if (dynamic_cast<Sprite*>(orb)) {
50             Sprite *temp = orb;
51             orb = new ExplodingSprite(*orb);
52             delete temp;
53           }
54           break;
55         }
56         default          : break;
57       }
58     }
59   }
60 }
```