1. (10 points) Give the output for the following program. Notice the output statements in the constructors and the transmission mode for function parameters and return values.

```
 1 #include <iostream>
 2 using std::cout;  using std::endl;
 3 class string {
 4 public:
 5    string() { cout << "default" << endl; }
 6    string(const char * n) { cout << "convert" << endl; }
 7    string(const string&) { cout << "copy" << endl; }
 8   const string& operator=(const string&) {
 9     cout << "assignment" << endl;
10   }
11 private:
12    char * buf;
13 };
14
15 const string doit(const string x) {
16   return x;
17 }
18 int main() {
19   string a("antelope");
20   a = doit(a);
21 }
```

2. (10 points) The following program prints "dog" and then crashes with a *double free* error. What causes this problem? Fix the program so that it doesn't crash; you may add code, but not delete code.

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

3. (10 points) Write an output and assignment operator for class string in the previous problem.

4. (10 points) Give the output for the following program.

```
#include <iostream>
#include <vector>
#include <cstdlib>
const int MAX = 3;

void print(std::vector<int>& vec) {
  std::cout << "size: " << vec.size() << '\t' << "cap: " << vec.capacity() << std::endl;
}

int main() {
  std::vector<int> vec1;
  std::vector<int> vec2(MAX);
  std::vector<int> vec3;
  vec3.reserve(MAX);
  vec1.push_back(rand() % 100);
  vec2.push_back(rand() % 100);
  vec3.push_back(rand() % 100);
  print(vec1);
  print(vec2);
  print(vec3);
}
```

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

```
 1 #include <iostream>
 2 #include <vector>
 3 using std::string; using std::cout; using std::endl;
 4 class A {
 5 public:
 6   A(const string& n) : name(n) {}
 7   const string getName() const { return name; }
 8   int getAge() const { return 111; }
 9 private:
10   string name;
11 };
12 class B : public A {
13 public:
14   B(const string& n, int a) : A(n), age(a) {}
15   const string getName() const { return "Torvalds"; }
16   int getAge() const { return age; }
17 private:
18   int age;
19 };
20
21 int main() {
22   std::vector<A*> people;
23   people.push_back(new B("Abe", 21));
24   people.push_back(new A("Bill"));
25   cout << people[0]->getAge() << endl;
26   cout << people[1]->getAge() << endl;
27 }
```

6. (10 points) The program in the previous problem leaks 24 bytes of memory. Write code to eliminate the memory leak(s); You may add code, but not delete code.

7. (10 points) Give the output for the following program.

```
#include <iostream>
#include <vector>
const unsigned int MAX = 3;
class A {
public:
  virtual void print() const { std::cout << "I'm an A" << std::endl; }
  void display() const { std::cout << "I'm an A" << std::endl; }
};
class B : public A {
public:
  virtual void print() const { std::cout << "I'm an B" << std::endl; }
  void display() const { std::cout << "I'm an B" << std::endl; }
};

void printVecOfA(const std::vector<A> & vec) {
  std::cout << "Printing A: " << std::endl;
  for (unsigned i = 0; i < vec.size(); ++i) {
    vec[i].print();
    vec[i].display();
  }
}

void printVecOfAstar(const std::vector<A*> & vec) {
  std::cout << "Printing A star: " << std::endl;
  for (unsigned i = 0; i < vec.size(); ++i) {
    vec[i]->print();
    vec[i]->display();
  }
}

int main() {
  std::vector<A> vecOfA;
  vecOfA.push_back( B() );
  printVecOfA(vecOfA);
  std::vector<A*> vecOfAstar;
  vecOfAstar.push_back( new A() );
  vecOfAstar.push_back( new B() );
  printVecOfAstar(vecOfAstar);
}
```

8. The program below has two classes: Manager, and Clock (see the next page).

   (a) (10 points) Give the output for the program.
   (b) (10 points) Use the technique described in the Design Patterns book to convert Clock into a singleton; i.e., use a static pointer variable local to Clock to point to the singleton instance.
   (c) (10 points) change Manager so that it uses the Clock singleton correctly.

```
#include <SDL.h>
#include "clock.h"
const unsigned int MAX = 3;

class Manager {
public:
  Manager() :
    initVar(SDL_Init(SDL_INIT_VIDEO)),
    clock() {
    atexit(SDL_Quit);
  }
  void play() {
    for (unsigned i = 0; i < MAX; ++i) {
      ++clock;
    }
    std::cout << "Frames: " << clock.getFrames() << std::endl;
  }
private:
  int initVar;
  Clock clock;
};

int main(int, char*[]) {
  Manager manager;
  manager.play();
}
```

```
********************** clock.h ***********************************
class Manager;
class Clock {
public:
  Clock();
  unsigned getTicks() const;
private:
  unsigned frames;
  unsigned currTicks;
  unsigned prevTicks;
  unsigned ticks;
  friend class Manager;
  unsigned getElapsedTicks();
  Clock& operator++();
  Clock  operator++(int);
  unsigned getFrames() const  { return frames;  }
  unsigned getSeconds() const { return getTicks()/1000;  }
  int getFps() const;
  Clock(const Clock&);
  Clock&operator=(const Clock&);
};

******************** clock.cpp *********************************
#include <iostream>
#include <string>
#include <SDL.h>
#include "clock.h"

Clock::Clock() : frames(0), currTicks(0), prevTicks(0), ticks(0) { }

unsigned Clock::getTicks() const {
  return SDL_GetTicks();
}

unsigned Clock::getElapsedTicks() {
  currTicks = getTicks();
  ticks = currTicks-prevTicks;
  prevTicks = currTicks;
  return ticks;
}

int Clock::getFps() const {
  if ( getSeconds() > 0 ) return frames/getSeconds();
  else if ( getTicks() > 1000 and getFrames() == 0 ) {
    throw std::string("Can't getFps if you don't increment the frames");
  }
  else return 0;
}

Clock& Clock::operator++() {
  ++frames;
  return *this;
}
```