Name _____

CpSc 416/616: 2D Game Development with C++

Brian Malloy, PhD
School of Computing
Clemson University

Exam # 1

January 14, 2014

page 1. (20 pts) _____

page 2. (30 pts) _____

page 3. (20 pts) _____

page 4. (15 pts) _____

page 5. (15 pts) _____

Total: _____

1. (10 points) Give the output for the program below. Note the output statements in the constructors, and the push_back statement on line 18..

```
 1 #include <iostream>
 2 #include <list>
 3 #include <cstdlib>
 4 const unsigned MAX_NUMBERS = 3;
 5 class Number {
 6 public:
 7   Number(int n) : number(n) {
 8     std::cout << "convert" << std::endl;
 9   }
10   Number(const Number& n) : number(n.number) {
11     std::cout << "copy" << std::endl;
12   }
13 private:
14   int number;
15 };
16 void init(std::list<Number>& numbers) {
17   for (unsigned i = 0; i < MAX_NUMBERS; ++i) {
18     numbers.push_back(rand() % 100);
19   }
20 }
21 int main() {
22   std::list<Number> numbers;
23   init(numbers);
24 }
```

2. (10 points) Overload comparison and output operators so sort and print work (lines 26 & 27).

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <algorithm>
 4 #include <cstdlib>
 5 const unsigned MAX_NUMBERS = 3;
 6 class Number {
 7 public:
 8   Number(int n) : number(n) { }
 9   int getNumber() const { return number; }
10 private:
11   int number;
12 };
13 void print(const std::vector<Number>& numbers) {
14   for (unsigned i = 0; i < numbers.size(); ++i) {
15     std::cout << numbers[i] << std::endl;
16   }
17 }
18 void init(std::vector<Number>& numbers) {
19   for (unsigned i = 0; i < MAX_NUMBERS; ++i) {
20     numbers.push_back(rand() % 100);
21   }
22 }
23 int main() {
24   std::vector<Number> numbers;
25   init(numbers);
26   sort(numbers.begin(), numbers.end());
27   print(numbers);
28 }
```

3. (15 points) Overload an output operator for Student and write a function object so that the sort and printPtrs functions on lines 24 & 25 work properly.

```
 1 #include <iostream>
 2 #include <string>
 3 #include <vector>
 4 #include <algorithm>
 5
 6 class Student {
 7 public:
 8    Student(const std::string& n) : name(n) { }
 9    const std::string& getName() const { return name; }
10 private:
11    std::string name;
12 };
13 void printPtrs(const std::vector<Student*>& students) {
14   for (unsigned i = 0; i < students.size(); ++i) {
15     std::cout << students[i] << '\t';
16   }
17   std::cout << std::endl;
18 }
19 int main() {
20   std::vector<Student*> students;
21   students.push_back( new Student("Howard") );
22   students.push_back( new Student("Sheldon") );
23   students.push_back( new Student("Penny") );
24   sort(students.begin(), students.end(), CompareStudent());
25   printPtrs(students);
26 }
```

4. (15 points) Write function removeEvens so that it removes even numbers from numbers.

```
 1 #include <list>
 2 #include <cstdlib>
 3 const unsigned MAX_NUMBERS = 10;
 4
 5 class Number {
 6 public:
 7   Number(int n) : number(n) { }
 8   int getNumber() const { return number; }
 9 private:
10   int number;
11 };
12
13 void init(std::list<Number*>& numbers) {
14   for (unsigned i = 0; i < MAX_NUMBERS; ++i) {
15     numbers.push_back( new Number(rand() % 100) );
16   }
17 }
18 int main() {
19   std::list<Number*> numbers;
20   init(numbers);
21   removeEvens(numbers);
22 }
```

Figure 1: **A Sprite sheet containing five frames for animating a coin**.

5. (10 points) Figure 1 illustrates a sprite sheet that might be used in Painter's algorithm and the three statements listed below show how one might scale the surface on which the sprite sheet is loaded. Explain the problem that might result when animating the scaled multi-frame sprite; give a worst case scenario. Will this problem only happen with the coins illustrated in Figure 1 or can the problem occur with other sprites used in Painter's algorithm? What types of sprites might exemplify this behavior?

```
SDL_Surface *surface = IMG_Load("images/coin.png");
double scale = getRandFloat(0.5, 1.5);
SDL_Surface* tmp = rotozoomSurface(surface, 0, scale, SMOOTHING_ON);
```

6. (10 points) Figure 2 illustrates three sprites that might collide in an animation.

   - What collision strategy would be best for detecting collisions between the two orbs?

   - What collision strategy would be best for detecting collisions between the orbs and the U-shaped hollow rectangle?

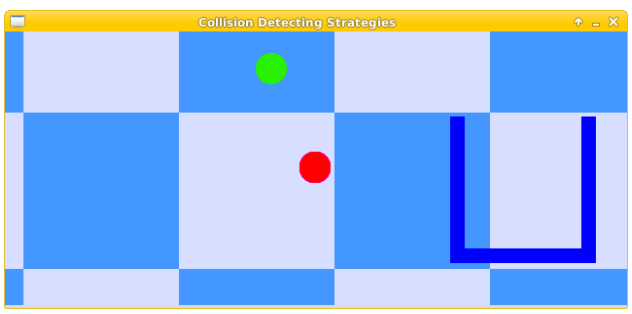

Figure 2: **This figure illustrates two orbs and a U-shaped hollow rectangle**.

7. (5 points) Give the output for the program below.

```
 1 #include <iostream>
 2 #include <string>
 3 #include <map>
 4 void print(const std::map<std::string, int> &m) {
 5   std::map<std::string, int>::const_iterator ptr;
 6   for (ptr = m.begin(); ptr != m.end(); ++ptr) {
 7     std::cout << ptr->first << '\t' << ptr->second << std::endl;
 8   }
 9 }
10 int main () {
11   std::map<std::string,int> mymap;
12   mymap["cat"]=150;
13   mymap["aardvark"]=50;
14   mymap["cat"]=200;
15   mymap["aardvark"]=100;
16   print(mymap);
17 }
```

8. (10 points) Give the output for the following program. Why is the static_cast needed on line 25?

```
 1 #include <iostream>
 2 #include <list>
 3 #include <cstring>
 4 class A {
 5 public:
 6   virtual ~A() {}
 7   A(const char* n) : name(new char[strlen(n)+1]) {
 8     strcpy(name, n);
 9   }
10   const char* getName() const { return name; }
11 private:
12   char* name;
13 };
14 class B : public A {
15 public:
16   B(const char* n, int a) : A(n), age(a) {}
17   int getAge() const { return age; }
18 private:
19   int age;
20 };
21 void print( const std::list<A*>& theList) {
22   std::list<A*>::const_iterator ptr = theList.begin();
23   while ( ptr != theList.end() ) {
24     if ( dynamic_cast<B*>(*ptr) ) {
25       std::cout << static_cast<B*>(*ptr)->getAge() << std::endl;
26     }
27     ++ptr;
28   }
29 }
30 int main() {
31   std::list<A*> theList;
32   theList.push_back( new B("Sheldon", 35) );
33   theList.push_back( new A("Howard") );
34   theList.push_back( new B("Penny", 22) );
35   print( theList );
36 }
```

9. (5 points) What is the intent of:

   - The Strategy Pattern

   - The Composite Pattern

10. (10 points) A Clock class and a Manager class are listed on the following pages. Modify the Clock class so that it computes the average frames per second computed over the last 100 events. An event is defined as one iteration of the main event loop in Manager::play().

```
class Manager;
class Clock {
public:
  static Clock& getInstance();
  unsigned getTicks() const;
private:
  friend class Manager;
  unsigned getElapsedTicks();
  Clock& operator++();
  Clock  operator++(int);
  bool isStarted() const { return started; }
  bool isPaused() const  { return paused;  }
  unsigned getFrames() const  { return frames;  }
  unsigned getSeconds() const { return getTicks()/1000;  }
  int getFps() const;
  void start();
  void pause();
  void unpause();
  bool started;
  bool paused;
  unsigned frames;
  unsigned timeAtStart;
  unsigned timeAtPause;
  unsigned currTicks;
  unsigned prevTicks;
  unsigned ticks;
  Clock();
  Clock(const Clock&);
  Clock&operator=(const Clock&);
};
```

```
#include <string> #include <SDL.h> #include "clock.h"

Clock& Clock::getInstance() {
  if ( SDL_WasInit(SDL_INIT_VIDEO) == 0) {
    throw std::string("Must init SDL before Clock");
  }
  static Clock clock;
  return clock;
}
Clock::Clock() : started(false), paused(false), frames(0),
  timeAtStart(0), timeAtPause(0), currTicks(0), prevTicks(0), ticks(0)
{ start(); }
Clock::Clock(const Clock& c) :
  started(c.started), paused(c.paused), frames(c.frames),
  timeAtStart(c.timeAtStart), timeAtPause(c.timeAtPause),
  currTicks(c.currTicks), prevTicks(c.prevTicks), ticks(c.ticks)
{ start(); }
unsigned Clock::getTicks() const {
  if (paused) return timeAtPause;
  else return SDL_GetTicks() - timeAtStart;
}
unsigned Clock::getElapsedTicks() {
  if (paused) return 0;
  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++() {
  if ( !paused ) ++frames;
  return *this;
}
Clock  Clock::operator++(int) {
  if ( !paused ) frames++;
  return *this;
}
void Clock::start() {
  started = true; paused = false; frames = 0;
  timeAtPause = timeAtStart = SDL_GetTicks();
}
void Clock::pause() {
  if( started && !paused ) {
    timeAtPause = SDL_GetTicks() - timeAtStart;
    paused = true;
  }
}
void Clock::unpause() {
  if( started && paused ) {
    timeAtStart = SDL_GetTicks() - timeAtPause;
    paused = false;
  }
}
```

```
#include <SDL.h>
#include "ioManager.h"
#include "clock.h"
#include "gamedata.h"
#include "sprite.h"

class Manager {
public:
  Manager ();
  ~Manager ();
  void play();

private:
  const bool env;
  const Gamedata* gdata;
  const IOManager& io;
  Clock& clock;

  SDL_Surface *screen;
  int backRed;
  int backGreen;
  int backBlue;

  SDL_Surface * const orbSurface;
  Frame * const orbFrame;
  Sprite orb;

  void drawBackground() const;
  Manager(const Manager&);
  Manager& operator=(const Manager&);
};
```

```
#include "manager.h"
Manager::~Manager() {
  SDL_FreeSurface(orbSurface); delete orbFrame; delete Gamedata::getInstance();
}

Manager::Manager() :
  env( SDL_putenv(const_cast<char*>("SDL_VIDEO_CENTERED=center")) ),
  gdata( Gamedata::getInstance() ),
  io( IOManager::getInstance() ),
  clock( Clock::getInstance() ),
  screen( io.getScreen() ),
  backRed( gdata->getXmlInt("backRed") ),
  backGreen( gdata->getXmlInt("backGreen") ),
  backBlue( gdata->getXmlInt("backBlue") ),
  orbSurface( io.loadAndSet(gdata->getXmlStr("redorbFile"),
                gdata->getXmlBool("redorbTransparency")) ),
  orbFrame(new Frame(orbSurface,
                gdata->getXmlInt("redorbWidth"),
                gdata->getXmlInt("redorbHeight"),
                gdata->getXmlInt("redorbSrcX"),
                gdata->getXmlInt("redorbSrcY"))
  ),
  orb("redorb", orbFrame)
{
  if (SDL_Init(SDL_INIT_VIDEO) != 0) {
    throw string("Unable to initialize SDL: ");
  }
  atexit(SDL_Quit);
}
void Manager::drawBackground() const {
  SDL_FillRect( screen, NULL,
    SDL_MapRGB(screen->format, backRed, backGreen, backBlue) );
  SDL_Rect dest = {0, 0, 0, 0};
  SDL_BlitSurface( screen, NULL, screen, &dest );
}

void Manager::play() {
  SDL_Event event;

  bool done = false;
  while ( not done ) {
    drawBackground();
    orb.draw();
    SDL_Flip(screen);

    Uint32 ticks = clock.getElapsedTicks();
    orb.update(ticks);

    SDL_PollEvent(&event);
    if (event.type ==  SDL_QUIT) { break; }
    if(event.type == SDL_KEYDOWN) {
      switch ( event.key.keysym.sym ) {
        case SDLK_ESCAPE : done = true; break;
        case SDLK_q      : done = true; break;
        default          : break;
      }
    }
  }
}
```