1. (10 points) Give the output for the following program.

```
#include <iostream>
class Game {
public:
  Game()             { std::cout << "default" << std::endl;     }
  Game(const char*) { std::cout << "convert" << std::endl;     }
  Game(const Game&) { std::cout << "copy" << std::endl;        }
  ~Game()            { std::cout << "destructor" << std::endl; }
  Game& operator=(const Game&) {
    std::cout << "copy assign" << std::endl;
    return *this;
  }
};
Game fun(Game g) {
  return g;
}

int main() {
  Game cat("cat");
  fun(cat);
}
```

```
convert
copy
copy
destructor
destructor
destructor
```

---

2. (10 points) Give the output for the following program.

```
#include <iostream>
class Game {
public:
  Game()             { std::cout << "default" << std::endl;     }
  Game(const char*) { std::cout << "convert" << std::endl;     }
  Game(const Game&) { std::cout << "copy" << std::endl;        }
  ~Game()            { std::cout << "destructor" << std::endl; }
  Game& operator=(const Game&) {
    std::cout << "copy assign" << std::endl;
    return *this;
  }
};
Game fun() {
  return Game( Game() );
}
int main() {
  Game cat("cat");
  fun();
}
```

```
convert
default
destructor
destructor
```

3. (10 points) Give the output for the following program.

```
#include <iostream>

void f(const int& x)  { std::cout << "l-value ref: " << x << std::endl; }
void f(int&& x)       { std::cout << "r-value ref: " << x << std::endl; }
int  f()              { return 19; }

int main() {
  f(20);
  int x = 7;
  f(x);
  f(std::move(x));
  f(x+1);
  f(f());
}
```

```
r-value ref: 20
l-value ref: 7
r-value ref: 7
r-value ref: 8
r-value ref: 19
```

---

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

```
#include <iostream>

class A{
public:
  A()           { std::cout << "default constructor" << std::endl; }
  A(int)        { std::cout << "conversion constructor" << std::endl; }
  A(const A&)   { std::cout << "copy constructor" << std::endl; }
  A(const A&&) { std::cout << "move constructor" << std::endl; }
  A& operator=(const A&) {
    std::cout << "copy assignment" << std::endl;
    return *this;
  }
  A& operator=(const A&&) {
    std::cout << "move assignment" << std::endl;
    return *this;
  }
};

int main() {
  A a, b = a;
  a = b;
  b = 99;
}
```

```
default constructor
copy constructor
copy assignment
conversion constructor
move assignment
```

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

```
#include <iostream>
#include <vector>
class Game {
public:
  Game()             { std::cout << "default" << std::endl;     }
  Game(const char*) { std::cout << "convert" << std::endl;     }
  Game(const Game&) { std::cout << "copy" << std::endl;        }
  ~Game()            { std::cout << "destructor" << std::endl; }
private:
  const char* name;
};
Game fun(Game g) {
  return g;
}

int main() {
  std::vector<Game> games;
  games.emplace_back("Monopoly");
  games.push_back("Magic the Gathering");
}
```

```
convert
convert
copy
copy
destructor
destructor
destructor
destructor
```

---

6. (10 points) Write a move constructor for class Game above.

```
Game(Game&& g) : name(std::move(g.name)) {
  g.name = nullptr;
}
```

or

```
Game(Game&& g) : name(std::exchange(g.name, nullptr)) {}
```

7. (15 points) Give the output for the following program.

```
#include <iostream>
#include <string>
class A {
public:
  A(int n) : number(n) {}
  ~A() { std::cout << "deleting A" << std::endl; }
  int getNumber() const { return number; }
  virtual std::string getName() const { return "I'm A"; }
private:
  int number;
};
class B : public A {
public:
  B(int m, int  n) : A(m), number(n) {}
  ~B() { std::cout << "deleting B" << std::endl; }
  int getNumber() const { return number; }
  virtual std::string getName() const { return "I'm B"; }
private:
  int number;
};
int main() {
  A* x = new B(12, 13);
  B* y = new B(66, 77);
  A* z = new B(1, 2);
  std::cout << x->getNumber() << std::endl;
  std::cout << x->getName() << std::endl;
  std::cout << y->getNumber() << std::endl;
  std::cout << y->getName() << std::endl;
  delete z;
}
```

```
12
I'm B
77
I'm B
deleting A
```

8. (10 points) Convert the following program so that Clock is a *GoF* singleton.

```
1  #include <iostream>
2
3  class Clock {
4  public:
5    static Clock* getInstance() {
6      if ( !instance ) instance = new Clock;
7      return instance;
8    }
9    int getTicks() const { return ticks; }
10   void update() { ++ticks; }
11 private:
12   int ticks;
13   static Clock* instance;
14   Clock() : ticks(0) {}
15   Clock(const Clock&);
16   Clock&  operator=(const Clock&);
17 };
18
19 Clock* Clock::instance = NULL;
20 int main( ) {
21   Clock* clock = Clock::getInstance();
22   clock->update();
23   std::cout << clock->getTicks() << std::endl;
24 }
```

9. (15 points) The following program draws a rectangle on the screen. Modify the program so that it draws a triangle on the screen.

```
#include <iostream>
#include <string>
#include <SDL2/SDL.h>

const int WIDTH = 640;
const int HEIGHT = 480;
const std::string TITLE = "Drawing a Rectangle";

int main (int , char*[]) {
  SDL_Window* window = SDL_CreateWindow(
      TITLE.c_str(),
      SDL_WINDOWPOS_CENTERED,
      SDL_WINDOWPOS_CENTERED,
      WIDTH,
      HEIGHT,
      SDL_WINDOW_SHOWN
  );

  SDL_Renderer* renderer = SDL_CreateRenderer(
    window, -1, SDL_RENDERER_ACCELERATED
  );

  SDL_SetRenderDrawColor( renderer, 208, 209, 210, 255 );
  SDL_RenderClear( renderer );

  SDL_Rect r = {150, 150, 250, 150};
  SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255 );
  SDL_RenderDrawRect( renderer, &r );
  SDL_RenderPresent(renderer);

  SDL_Event event;
  const Uint8* keystate;
  while ( true ) {
    keystate = SDL_GetKeyboardState(0);
    if (keystate[SDL_SCANCODE_ESCAPE]) { break; }
    if (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) {
        break;
      }
    }
  }
  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
  return EXIT_SUCCESS;
}
```