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

```
#include <iostream>
#include <cstring>
class string {
public:
  string() { std::cout << "default" << std::endl; }
  string(const char*) { std::cout << "convert" << std::endl; }
  string(const string&) { std::cout << "copy" << std::endl; }
  ~string() { std::cout << "destructor" << std::endl; }

  string& operator=(const string&) {
    std::cout << "copy assign" << std::endl;
    return *this;
  }
};

void fun(string) { }

int main() {
  string circle("circle"), square = circle;
  fun(circle);
  int x = 19;
  int& r = x;
  r = 747;
  std::cout << x << std::endl;
}
```

```
convert
copy
copy
destructor
747
destructor
destructor
```

---

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

```
#include <iostream>
class VideoGame {
public:
  VideoGame()              { std::cout << "default" << std::endl;     }
  VideoGame(const char*) { std::cout << "convert" << std::endl;     }
  VideoGame(const VideoGame&) { std::cout << "copy" << std::endl;        }
  ~VideoGame()             { std::cout << "destructor" << std::endl; }
};
VideoGame fun() {
  return VideoGame( VideoGame( VideoGame() ) );
}
int main() {
  VideoGame paladins("Koga");
  fun();
}
```

```
convert
```

```
default
destructor
destructor
```

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

```
#include <iostream>

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

int main() {
  fun(747);
  int number = 42;
  fun(number);
  fun(number+98);
  fun(std::move(number));
  fun(fun());
}
```

```
r-value ref: 747
l-value ref: 42
r-value ref: 140
r-value ref: 42
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(17), b(a);
  b = 99;
  a = b;
}
```

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

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

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

int main() {
  std::vector<Console> games;
  games.push_back("Switch");
  games.emplace_back("PS4");
}
```

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

---

6. (10 points) Give the output for the following program.

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

int main() {
  std::vector<Console> games;
  games.reserve(2);
  games.emplace_back("Switch");
  games.emplace_back("PS4");
}
```

```
convert
convert
destructor
destructor
```

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

```
1  #include <iostream>
2  class Shape {
3  public:
4    Shape(const char*) { std::cout << "convert Shape" << std::endl;     }
5    ~Shape()           { std::cout << "destroy Shape" << std::endl; }
6    Shape(const Shape&) = delete;
7  private:
8    const char* name;
9  };
10 class Circle : public Shape {
11 public:
12   Circle(const char* c, int r) : Shape(c), radius(r) {
13     std::cout << "convert Circle" << std::endl;
14   }
15   ~Circle()  { std::cout << "destroy Circle" << std::endl; }
16   Circle(const Circle&) = delete;
17 private:
18   float radius;
19 };
20 int main() {
21   Shape* x = new Circle("circle", 5);
22   delete x;
23 }
```

```
convert Shape
convert Circle
destroy Shape
```

8. (10 points) Give the output for the following program.

```
#include <iostream>
#include <string>
class Shape {
public:
  Shape(const std::string& n) : name(n) {}
  virtual ~Shape() = default;
  Shape(const Shape&) = delete;
  std::string getName() const { return name; }
  virtual float getArea() const { return 77; }
private:
  std::string name;
};
class Circle : public Shape {
public:
  Circle(const char* c, int r) : Shape(c), radius(r) { }
  ~Circle() = default;
  Circle(const Circle&) = delete;
  std::string getName() const { return "Connor"; }
  virtual float getArea() const { return 3.14*radius*radius; }
private:
  float radius;
};
int main() {
  Shape* x = new Circle("Hank", 2);
  std::cout << x->getName() << std::endl;
  std::cout << x->getArea() << std::endl;
  delete x;
}
```

```
Hank
12.56
```

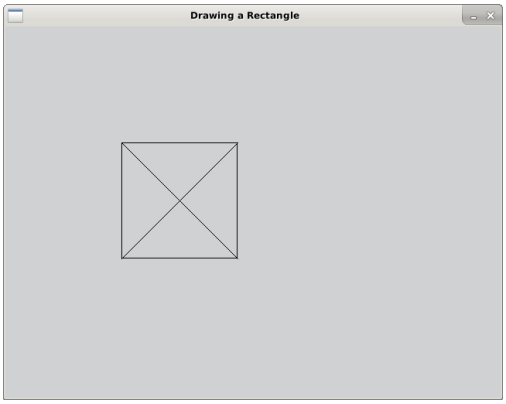
9. (10 points) Write a function, eraseLessThan, that removes all numbers from list, mylist, that are less than max, where max is a parameter to eraseLessThan. A sample execution might be:

```
32, 32, 54, 12, 52, 56, 8, 30, 44, 94, 44, 39, 65, 19, 51, 91, 1, 5, 89, 34,
removing all less than 53
54, 56, 94, 65, 91, 89,
```

```
1  #include <iostream>
2  #include <list>
3  #include <cstdlib>
4  const int MAX = 20;
5  const int MAX_NUMBER = 101;
6
7  void eraseLessThan(std::list<int> & mylist, int max) {
8    std::cout << "removing all less than " << max << std::endl;
9    auto it = mylist.begin();
10   while ( it != mylist.end() ) {
11     if (*it < max ) {
12       it = mylist.erase(it);
13     }
14     else {
15       ++it;
16     }
17   }
18 }
19
20 void init(std::list<int> & mylist) {
21   for (unsigned int i = 0; i < MAX; ++i) {
22     mylist.push_back( rand() % MAX_NUMBER );
23   }
24 }
25
26 void print(const std::list<int> & mylist) {
27   for ( int number : mylist ) {
28     std::cout << number  << ", ";
29   }
30   std::cout << std::endl;
31 }
32
33 int main() {
34   std::list<int> mylist;
35   init(mylist);
36   print(mylist);
37   eraseLessThan(mylist, rand()%50+50 - rand()%30);
38   print(mylist);
39 }
```

10. (10 points) Draw the figure that is rendered to the screen by the following program.

```
#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_SetRenderDrawColor( renderer, 0, 0, 0, 255 );
  SDL_RenderDrawLine(renderer, 150, 150, 300, 300);
  SDL_RenderDrawLine(renderer, 150, 300, 300, 150);

  SDL_Rect r = {150, 150, 150, 150};
  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;
}
```


Drawing a Rectangle