1. (8 points) Give the output for the following program.

```
#include <iostream>

class Empty { };

class TrulyEmpty {
public:
  virtual ~TrulyEmpty() {}
};

class Full {
  ~Full() {}
};

int main() {
  Empty * empty = new Empty;
  std::cout << sizeof(empty) << std::endl;
  std::cout << sizeof(*empty) << std::endl;
  std::cout << sizeof(TrulyEmpty) << std::endl;
  std::cout << sizeof(Full) << std::endl;
}
```

```
8
1
8
1
```

---

2. (6 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& operator= (const Game&) { std::cout << "assignment" << std::endl; }
  ~Game()           { std::cout << "destructor" << std::endl; }
private:
  const char* name;
};
Game play(Game g) {
  return g;
}
int main() {
  Game game = "Fallout 76";
  play( game );
}
```

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

3. (4 points) Give the output for the following program.

```
1  #include <iostream>
2  int main() {
3    int x = 99;
4    int y = 7;
5    int& r = x;
6    r = y;
7    std::cout << x << std::endl;
8  }
```

7

4. (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;
};

int main() {
  std::vector<Game> games;
  games.push_back("Cyberpunk 2077");
  games.push_back("Elder Scrolls 6");
}
```

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

---

5. (8 points) Give the output for the following program.

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

int main() {
  std::vector<string> vec;
  vec.reserve(2);
  vec.emplace_back( "cat" );
  vec.emplace_back( "antelope" );
}
```

```
convert
convert
destructor
destructor
```

6. (6 points) In Meyer's book, Effective Modern C++, Item #11 states "Prefer deleted functions to private underfined ones". Fix class string so that it conforms to Item #11.

```
#include <iostream>
#include <cstring>

class string {
public:
  ~string() { delete [] buf; }
private:
  char* buf;
  string(const string&);
  string& operator=(const string&);
};
```

```
#include <iostream>
#include <cstring>

class string {
public:
  ~string() { delete [] buf; }
  string(const string&) = delete;
  string& operator=(const string&) = delete;
private:
  char* buf;
};
```

7. (6 points) Give the output for the following program.

```
#include <iostream>
#include <vector>

int main() {
  std::vector<int> vec1;
  for (unsigned int i = 0; i < 1000; ++i) {
    vec1.push_back(i);
  }
  std::cout << vec1.size() << ", "<< vec1.capacity() << std::endl;

  std::vector<int> vec2(1000);
  vec2.push_back( 1000 );
  std::cout << vec2.size() << ", "<< vec2.capacity() << std::endl;

  std::vector<int> vec3;
  vec3.reserve(1000);
  vec3.push_back( 1000 );
  std::cout << vec3.size() << ", "<< vec3.capacity() << std::endl;
}
```

```
1000, 1024
1001, 2000
1, 1000
```

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

```
1  #include <iostream>
2  #include <string>
3
4  int display(const std::string& msg) {
5    std::cout << msg << std::endl;
6    return 1;
7  }
8  class Base {
9  public:
10   Base() : number( display("Base") ) {
11     std::cout << "Base Constructor" << std::endl;
12   }
13   ~Base() { std::cout << "Base Destructor" << std::endl; }
14 private:
15   int number;
16 };
17 class Derived : public Base {
18 public:
19   Derived()  : number( display("Derived") ) {
20     std::cout << "Derived Constructor" << std::endl;
21   }
22   ~Derived() { std::cout << "Derived Destructor" << std::endl;    }
23 private:
24   int number;
25 };
26 int main() {
27   Base* obj = new Derived;
28   delete obj;
29 }
```

```
Base
Base Constructor
Derived
Derived Constructor
Base Destructor
```

9. (12 points) Give the output for the following program.

```
1  #include <iostream>
2
3  class A {
4  public:
5            void f() const { std::cout << "A::f" << std::endl; }
6    virtual void g() const { std::cout << "A::g" << std::endl; }
7    virtual void h() const = 0;
8  };
9
10 class B : public A {
11 public:
12           void f() const { std::cout << "B::f" << std::endl; }
13           void g() const { std::cout << "B::g" << std::endl; }
14   virtual void h() const { std::cout << "B::h" << std::endl; }
15 };
16
17 int main() {
18   A* a = new B;
19   a->f();
20   a->g();
21   a->h();
22
23   B b;
24   b.f();
25 }
```

```
A::f
B::g
B::h
B::f
```

10. (6 points) Write function eraseEvens, which accepts vec and removes all even numbers.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>
const unsigned MAX = 20;
const unsigned MAX_NUMBER = 100;

void removeOddsFast(std::vector<int> & vec) {
  std::vector<int>::iterator front = vec.begin();
  std::vector<int>::iterator back = vec.end();
  --back;
  while ( front != back ) {
    if ( *front%2 ) {
      std::swap(*front, *back);
      --back;
    }
    else ++front;
  }
  if ( *back%2 == 0 ) ++back;
  vec.erase(back, vec.end());
}

void removeOdds(std::vector<int> & vec) {
  std::vector<int>::const_iterator it = vec.cbegin();
  while (it != vec.cend()) {
    if ( *it%2 ) {
      it = vec.erase(it);
    }
    else ++it;
  }
}

void init(std::vector<int> & vec) {
  for (unsigned i = 0; i < MAX; ++i) {
    vec.push_back( rand() % MAX_NUMBER );
  }
}

void print(const std::vector<int> & vec) {
  for ( const int x : vec ) {
    std::cout << x  << ", ";
  }
  std::cout << std::endl;
}

int main() {
  std::vector<int> vec;
  init(vec);
  print(vec);
  removeOdds(vec);
  removeOddsFast(vec);
  print(vec);
}
```

11. (8 points) Supply the missing functions to the class below so that it conforms to the Rule of 3.

```
#include <iostream>
#include <cstring>

class string {
public:
  ~string() { delete [] buf; }
private:
  char* buf;
};
```

```
string(const string& s) : buf(new char[strlen(s.buf)+1]) {
  strcpy(buf, s.buf);
}

string& operator=(const string& rhs) {
  if ( this == &rhs ) return *this;
  delete [] buf;
  buf = new char[strlen(rhs.buf)+1];
  strcpy(buf, rhs.buf);
  return *this;
}
```

12. (8 points) Modify class Clock so that it is a Gang of Four Singleton. Be sure to modify function main.

```
#include <iostream>

class Clock {
public:
  Clock() : ticks(0) {}
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
};


int main( ) {
  Clock clock;
  clock.update();
  std::cout << clock.getTicks() << std::endl;
}
```

```
#include <iostream>

class Clock {
public:
  static Clock* getInstance() {
    if ( !instance ) instance = new Clock;
    return instance;
  }
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
  static Clock* instance;
  Clock() : ticks(0) {}
  Clock(const Clock&);
  Clock&  operator=(const Clock&);
};

Clock* Clock::instance = NULL;
int main( ) {
  Clock* clock = Clock::getInstance();
  clock->update();
  std::cout << clock->getTicks() << std::endl;
}
```

13. (8 points) Draw the figure that is rendered to the screen by the following program.

```
#include <SDL.h>
const double PI = 3.1415926535897;
const double DEG2RAD = (PI/180);
const int WIDTH = 640;
const int HEIGHT = 480;
const int RADIUS = 50;
const SDL_Point CENTER = {WIDTH/2, HEIGHT/2};
const SDL_Color RED = {255, 0, 0, 255};

double DegToRad(double x) {
  return x*DEG2RAD;
}

int main(void) {
  SDL_Renderer *renderer;
  SDL_Window *window;

  SDL_Init(SDL_INIT_VIDEO);
  SDL_CreateWindowAndRenderer(
    WIDTH, HEIGHT, 0, &window, &renderer
  );

  SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255 );
  SDL_RenderClear(renderer);
  SDL_SetRenderDrawColor(renderer, RED.r, RED.g, RED.b, RED.a);


  for (int theta = 0; theta < 360; theta += 1) {
    SDL_RenderDrawPoint(renderer,
      CENTER.x+RADIUS*cos(DegToRad(theta)),
      CENTER.y+RADIUS*sin(DegToRad(theta))
    );
  }

  SDL_Rect r;
  r.x = CENTER.x-RADIUS;
  r.y = CENTER.y-RADIUS;
  r.w = 2*RADIUS;
  r.h = 2*RADIUS;


  SDL_RenderDrawRect( renderer, &r );

  SDL_RenderPresent(renderer);

  SDL_Event event;
  while ( true ) {
    if (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) {
        break;
      }
    }
  }
  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
}
```

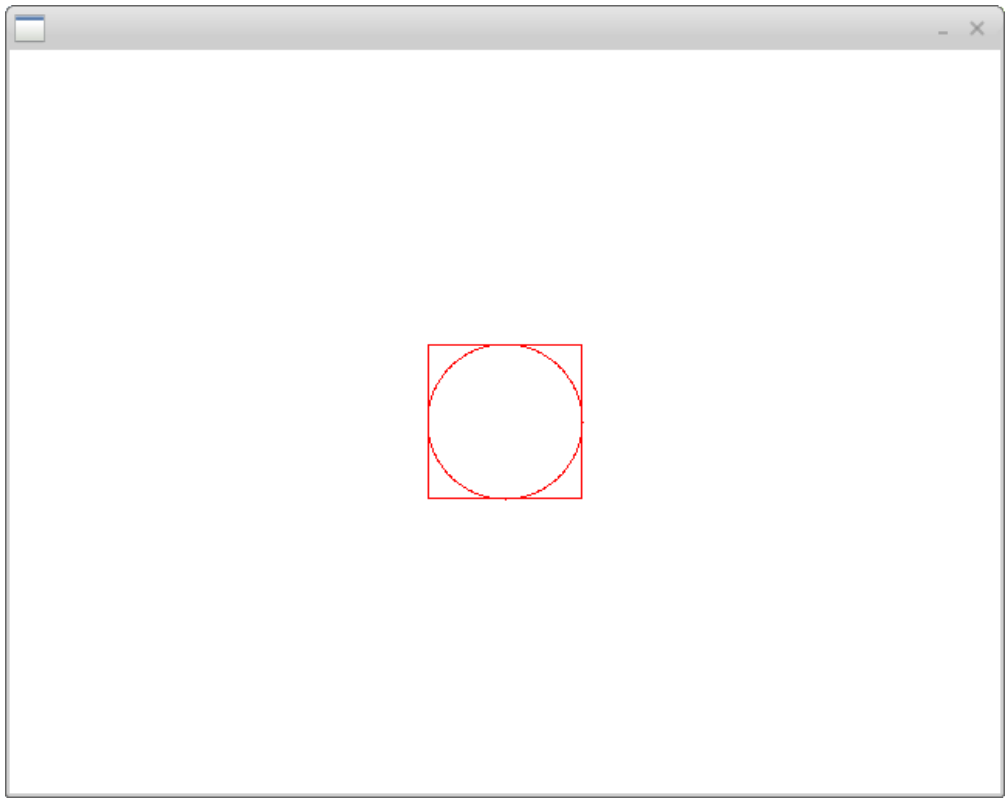

Figure 1: **Solution**.