1. (10 points) Write an assignment operator and setBuf function (the whole functions) for class string

```
#include <iostream>
#include <cstring>
class string {
public:
  string(const char* s) : buf(new char[strlen(s)+1]) {
    strcpy(buf, s);
    std::cout << "convert" << std::endl;
  }
  char* getBuf() const { return buf; }
  void setBuf(const char* b) {
    if ( b == nullptr ) return;
    delete [] buf;
    buf = new char[strlen(b)+1];
    strcpy(buf, b);
  }

  string& operator=(const string& rhs) {
    if ( this == &rhs ) return *this;
    delete [] buf;
    buf = new char[strlen(rhs.buf)+1];
    strcpy(buf, rhs.buf);
    std::cout << "assign" << std::endl;
    return *this;
  }
private:
  char* buf;
};
int main() {
  string str("Cuphead");
  str.setBuf("Eevie");
  std::cout << str.getBuf() << std::endl;
}
```

---

2. (10 points) The following program terminates with a *double free* error. Write the string function (just one function) (the whole function) needed to remove this error.

   Need to write a copy constructor.

```
#include <iostream>
#include <cstring>
class string {
public:
  string(const char* s) : buf(new char[strlen(s)+1]) { strcpy(buf, s); }
  string(const string& s) : buf(new char[strlen(s.buf)+1]) { strcpy(buf, s.buf); }
  ~string() { delete [] buf; }
  const char* getBuf() const { return buf; }
private:
  char* buf;
};

int main() {
  string game1("Cuphead"), game2 = game1;
}
```

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

```
#include <vector>
#include <iostream>
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; }
  char* getBuf() const { return nullptr; }
  string& operator=(const string& ) {
    std::cout << ”assign” << std::endl;
    return *this;
  }
};

int main() {
  std::vector<string> games;
  games.push_back(”Skyrim”);
  games.push_back(”GTA 5”);
}
```

```
convert
copy
convert
copy
copy
```

---

4. (10 points) Give the output for the following program. Note the use of reserve on line 17.

```
1  #include <vector>
2  #include <iostream>
3  class string {
4  public:
5    string() { std::cout << ”default” << std::endl; }
6    string(const char*) { std::cout << ”convert” << std::endl; }
7    string(const string& ) { std::cout << ”copy” << std::endl; }
8    char* getBuf() const { return nullptr; }
9    string& operator=(const string& ) {
10     std::cout << ”assign” << std::endl;
11     return *this;
12   }
13 };
14
15 int main() {
16   std::vector<string> games;
17   games.reserve(2);
18   games.push_back(”Skyrim”);
19   games.push_back(”GTA 5”);
20 }
```

```
convert
copy
convert
copy
```

5. (10 points) Give the output for the following program. Note the use of *emplace_back* on lines 18–19.

```
 1 #include <vector>
 2 #include <iostream>
 3 class string {
 4 public:
 5   string() { std::cout << ”default” << std::endl; }
 6   string(const char*) { std::cout << ”convert” << std::endl; }
 7   string(const string& ) { std::cout << ”copy” << std::endl; }
 8   char* getBuf() const { return nullptr; }
 9   string& operator=(const string& ) {
10     std::cout << ”assign” << std::endl;
11     return *this;
12   }
13 };
14
15 int main() {
16   std::vector<string> games;
17   games.reserve(2);
18   games.emplace_back(”Skyrim”);
19   games.emplace_back(”GTA 5”);
20 }
```

```
convert
convert
```

---

6. (10 points) Convert class GameDB to a GoF singleton. Make all necessary changes in the whole program so that your singleton works. Use delete to "Explicitly disallow the use of compiler-generated."

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

class GameDB {
public:
  static GameDB* getInstance() {
    if (!instance) instance = new GameDB;
    return instance;
  }
  void addGame(const std::string& g) { games.emplace_back(g); }
  GameDB(const GameDB&) = delete;
  GameDB& operator=(const GameDB&) = delete;
private:
  static GameDB* instance;
  std::vector<std::string> games;
  GameDB() : games() {}
};
GameDB* GameDB::instance = nullptr;

int main() {
  GameDB* games = GameDB::getInstance();
  games->addGame(”Skyrim”);
}
```

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

```
#include <iostream>
#include <cstring>
class string {
public:
  string(const char* s) : buf(new char[strlen(s)+1]) {
    strcpy(buf, s);
  }
  char* getBuf() const { return buf; }
private:
  char* buf;
};
int main() {
  string game("Cuphead");
  char* buffer = game.getBuf();
  buffer[0] = 'b';
  buffer[2] = 't';
  std::cout << game.getBuf() << std::endl;
  int x = 99;
  int* ptr = &x;
  int& ref = x;
  ref = 7;
  std::cout << *ptr << std::endl;
}
```

buthead

7

---

8. (10 points) Add code to the main function below so that you use stringstream to enable you to use function writeText in class IOmod to write the following message:

```
The fps is 7.69
```

```
#include <iostream>
#include <string>
#include <sstream>
class Clock{
public:
  float getFps() const { return 345.9/45.7; }
};
class IOmod {
public:
  IOmod() {}
  void writeText(const std::string& msg) const {
    std::cout << msg << std::endl;
  }
};
int main() {
  IOmod io;
  Clock clock;
  float fps = clock.getFps();
  std::stringstream strm;
  strm << "The fps is " << fps;
  io.writeText(strm.str());
}
```

9. (10 points) Write function display, used in function main on line 48, so that the program below prints the output listed. (Yes, you have to use a loop but you can use any kind of loop you prefer)

```
Circle, 25
Rectangle, 15, 40
Circle, 5
Circle, 17
```

```
1  #include <iostream>
2  #include <vector>
3
4  class Shape {
5  public:
6    Shape(const std::string& n) : name(n) {}
7    const std::string& getName() const { return name; }
8    virtual void display() const {
9      std::cout << name << std::endl;
10   }
11 private:
12   std::string name;
13 };
14
15 class Circle : public Shape {
16 public:
17   Circle(int r) : Shape("Circle"), radius(r) {}
18   virtual void display() const {
19     std::cout << getName() << ", " << radius << std::endl;
20   }
21 private:
22   int radius;
23 };
24
25 class Rectangle : public Shape {
26 public:
27   Rectangle(int h, int w) : Shape("Rectangle"), height(h), width(w) {}
28   virtual void display() const {
29     std::cout << getName() << ", " << height << ", " << width << std::endl;
30   }
31 private:
32   int height, width;
33 };
34
35 void display(const std::vector<Shape*>& shapes) {
36   for ( auto s : shapes ) {
37     s->display();
38   }
39 }
40
41 int main() {
42   std::vector<Shape*> shapes;
43   shapes.push_back( new Circle(25) );
44   shapes.push_back( new Rectangle(15, 40) );
45   shapes.push_back( new Circle(5) );
46   shapes.push_back( new Circle(17) );
47   display(shapes);
48 }
```

10. (10 points) Write function eraseEvens, used on line 25, so that all of the even numbers in the vector are removed. Write the whole function.

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <cstdlib>
 4 #include <algorithm>
 5 const int MAX = 20;
 6 const int MAX_NUMBER = 10;
 7 void init(std::vector<int> & vec) {
 8   for (unsigned int i = 0; i < MAX; ++i) {
 9     vec.push_back( rand() % MAX_NUMBER );
10   }
11 }
12 void print(const std::vector<int> & vec) {
13   for ( int number : vec ) {
14     std::cout << number << ",␣";
15   }
16   std::cout << std::endl;
17 }
18
19 void eraseEvens(std::vector<int> & vec) {
20   std::vector<int>::iterator ptr = vec.begin();
21   while ( ptr != vec.end() ) {
22     if ( (*ptr) % 2 == 0 ) {
23       ptr = vec.erase(ptr);
24     }
25     else ++ptr;
26   }
27 }
28
29 int main() {
30   std::vector<int> vec;
31   init(vec);
32   eraseEvens(vec);
33   print(vec);
34 }
```

---

11. (+5 extra credit) Give the output for the following program:

```
#include <iostream>
class Shape {
public:
  Shape(const char* n) {}
  ~Shape() { std::cout << "Delete␣Shape" << std::endl; }
private:
  char* name;
};
class Circle : public Shape {
public:
  Circle(const char* n) : Shape("Circle"), radius(n) {}
  ~Circle() { std::cout << "Delete␣Circle" << std::endl; }
private:
  const char* radius;
};
int main() {
  Shape* shape = new Circle("MyCircle");
  delete shape;
}
```

Delete Shape