

1. (15 points) The following program prints “cat” and then crashes with a *double free* error. Write a function that will eliminate this error.

```
1 #include <cstring>
2 #include <iostream>
3 using std::cout; using std::endl;
4 class string {
5 public:
6     string(const char* s) : buf(new char[strlen(s)+1]) { strcpy(buf, s);}
7     ~string() { delete [] buf; }
8     const char* getBuf() const { return buf; }
9 private:
10    char * buf;
11 };
12
13 int main() {
14     string a("cat"), b = a;
15     cout << a.getBuf() << endl;
16 }
```

***** Solution *****

On line #14, an instance of string, a, is made using the conversion construction, and an instance of string, b, is made using the compiler generated copy constructor. However, the compiler generated copy constructor makes a shallow copy of b, so that a::buf and b::buf point to the same location. Thus, when the program reaches line #16, first a is deallocated (no problem), and then b is deallocated, which causes the double free. The fix is for the programmer to supply a copy constructor that makes a deep copy.

2. (15 points) The following program prints “cat” and then crashes with a *double free* error. Write a function that will eliminate this error.

```
1 #include <cstring>
2 #include <iostream>
3 using std::cout; using std::endl;
4 class string {
5 public:
6     string() : buf(new char[1]) { strcpy(buf, "dog"); }
7     string(const char* s) : buf(new char[strlen(s)+1]) { strcpy(buf, s);}
8     ~string() { delete [] buf; }
9     const char* getBuf() const { return buf; }
10 private:
11    char * buf;
12 };
13
14 int main() {
15     string a("cat"), b;
16     b = a;
17     cout << b.getBuf() << endl;
18 }
```

***** Solution *****

Same problem as above, except that this time the compiler generated assignment operator is used on line # 16. The fix is for the programmer to supply an assignment operator that makes a deep copy (see the extra credit at the end of the exam).

3. (20 points) Give the output for the following program.

```
1 #include <iostream>
2 class Binary {
3 public:
4     Binary() : number(0), myCount(count) {
5         ++count;
6         std::cout << "default: " << myCount << std::endl;
7     }
8     Binary(int n) : number(n), myCount(count) {
9         ++count;
10        std::cout << "convert: " << myCount << std::endl;
11    }
12    Binary(const Binary& bin) : number(bin.number), myCount(count) {
13        ++count;
14        std::cout << "copy: " << myCount << std::endl;
15    }
16    ~Binary() { std::cout << "destructor: " << myCount << std::endl; }
17    Binary& operator=(const Binary&) {
18        std::cout << "assignment" << std::endl;
19        return *this;
20    }
21    int getNumber() const { return number; }
22    void increment() { ++number; }
23 private:
24     int number;
25     int myCount;
26     static int count;
27 };
28
29 Binary increment(Binary bin) {
30     bin.increment();
31     return bin;
32 }
33 int Binary::count = 0;
34 int main() {
35     Binary a(17), b = a;
36     b = increment(a);
37 }
```

***** Solution *****

```
convert: 0
copy: 1
copy: 2
copy: 3
assignment
destructor: 3
destructor: 2
destructor: 1
destructor: 0
```

4. The following program compiles and executes.

- (a) Give the output for the program. (5 points)
- (b) What public inlined functions does C++ silently write? (10 points)

```
1 #include <iostream>
2 class Binary {
3 public:
4     int getNumber() const { return number; }
5     void setNumber(int n) { number = n; }
6 private:
7     int number;
8 };
9
10 int main() {
11     Binary a, b = a;
12     a.setNumber(19);
13     std::cout << b.getNumber() << std::endl;
14 }
```

***** Solution *****

0

C++ compiler generates: default, copy, assignment, and destructor.

5. (5 points) The following program does **not** compile, and the compiler issues an error message indicating that the compiler did not write a default constructor. Why didn't the compiler write a default constructor?

```
1 #include <iostream>
2 class Binary {
3 public:
4     Binary(int n) : number(n) {}
5     int getNumber() const { return number; }
6     void setNumber(int n) { number = n; }
7 private:
8     int number;
9 };
10
11 int main() {
12     Binary a, b(19);
13     std::cout << b.getNumber() << std::endl;
14 }
```

main.cpp:12:10: error: no matching constructor for initialization of 'Binary'

Binary a, b(19);

***** Solution *****

If the programmer supplies a conversion constructor, then the compiler will not generate a default constructor.

6. (20 points) Make class Manager a Singleton.

```
1 #include <iostream>
2 #include <vector>
3 class Sprite {};
4
5 class Manager {
6 public:
7     Manager() : sprites() { }
8     void manage() {
9         // By doing nothing
10    }
11 private:
12     std::vector<Sprite> sprites;
13     Manager(const Manager&);
14     Manager& operator=(const Manager&);
15 };
16
17 int main() {
18     Manager manager;
19     manager.manage();
20 }
```

***** Solution *****

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

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

```
1 #include <cstdio>
2 #include <iostream>
3 using std::cout; using std::endl;
4
5 class Student {
6 public:
7     Student() { cout << "default" << endl; }
8     Student(const char *b) { cout << "convert" << endl; }
9     Student& operator=(const Student&) {
10         cout << "assign" << endl;
11         return *this;
12     }
13 private:
14     char *buf;
15 };
16
17 class TestStudent {
18     Student str;
19 public:
20     TestStudent(const char* s) {
21         str = s;
22     }
23 };
24
25 int main() {
26     TestStudent t1("cat");
27 }
```

***** Solution *****

default
convert
assign

8. (5 points extra) Consider the following implementation of an assignment operator for string. Does this implementation correctly handle the problem of assignment to self? Explain.

```
1 string& string::operator=(const string& rhs) {
2     string temp(rhs);
3     delete [] buf;
4     buf = new char[strlen(temp.buf)+1];
5     strcpy(buf, temp.buf);
6     return *this;
7 }
```

***** Solution *****

The need for an **if** statement to check for assignment to self is obviated by first making a copy of rhs in temp.