CS 32 Solutions Week 2

Concepts: Copy constructors, assignment operators

1. (5 minutes) What is the output of the following code?

```
#include <iostream>
using namespace std;

class A {
  public:
    A() { cout << "DC" << endl; }

    A(const A& other) { cout << "CC" << endl; }

    A& operator=(const A& other) {
        cout << "AO" << endl;
        return *this;
    }

    ~A() { cout << "Destructor!" << endl; }
};

int main() {
    A arr[3];                    // default constructs 3 A's
    arr[0] = arr[1];             // assignment operator
    A x = arr[0];                // copy constructor
    x = arr[1];                  // assignment operator
    A y(arr[2]);                 // copy constructor
    cout << "DONE" << endl;  // writes DONE
} // destroys y, x, and the 3 elements of arr
```

Solution:

```
DC
DC
DC
AO
CC
AO
CC
DONE
Destructor!
Destructor!
Destructor!
Destructor!
Destructor!
```

2. (10 minutes) Find the **4 errors** in the following class definitions so the main function runs correctly.

```
#include <iostream>
#include <string>
using namespace std;

class Account {
 public:
   Account(int x) { cash = x; }
   int cash;
};  // 1.

class Billionaire {
 public:
   Billionaire(string n) : account(10000) {  // 2.
       offshore = new Account(1000000000);   // 3.
       name = n;
   }

    ~Billionaire() { // 4.
         delete offshore;
     }

   Account account;
   Account* offshore;
   string name;
};

int main() {
   Billionaire jim = Billionaire("Jimmy");
   cout << jim.name << " has " << jim.account.cash + jim.offshore->cash
        << endl;
}
```

Output:

```
Jimmy has 1000010000
```

Solution:

1. Classes need a following semicolon in C++!

2. We need to initialize the account's cash to 10,000 to see the correct output. We could also move the initialization of `name` to the initialization list.

3. Since the offshore is a pointer to an Account object, we need to allocate it on the heap using the `new` keyword.

4. Now that we have a dynamically allocated object, we need to free it in the destructor or we'll have a memory leak.

3. (10 minutes) What is the output of the following code:

```cpp
#include <iostream>
using namespace std;

class B {
   int m_val;

 public:
   B(int x) : m_val(x) { cout << "Wow such " << x << endl; }
   B(const B& other) {
       cout << "There's another me???" << endl;
       m_val = other.m_val;
   }
   ~B() { cout << "Twas a good life" << endl; }
};

class A {
   int m_count;
   B* m_b;

 public:
   A() : m_count(9.5) {
       cout << "Construct me with " << m_count << endl;
       m_b = new B(m_count + 10);
   }
   A(const A& other) {
       cout << "Copy me" << endl;
       m_count = other.m_count;
       m_b = (other.m_b != nullptr) ? new B(*other.m_b) : nullptr;
   }
   ~A() {
       cout << "Goodbye cruel world" << endl;
       if (m_b) delete m_b;
   }
   int getCount() { return m_count; }
};

int main() {
   A a1, a2; // Construct me with 9 Wow such 19 Construct me with 9 Wow such 19
   A a3 = a2; // copy me, there's another me??
   B b1(a3.getCount()); // Wow such 9
   cout << "Where are we?" << endl;
}
```

Solution:

```
Construct me with 9
Wow such 19
Construct me with 9
Wow such 19
Copy me
There's another me???
Wow such 9
Where are we? // by the main function
Twas a good life // for b1 by B’s destructor
Goodbye cruel world // for a3 by A’s destructor
Twas a good life // for a3 by B's destructor called by A's dtor's delete
m_b
Goodbye cruel world // for a2
Twas a good life // for a2
Goodbye cruel world // for a1
Twas a good life // for a1
```

4. (15 minutes) Complete the copy constructor, assignment operator, and destructor of the following class. Be careful to avoid aliasing, memory leaks, and other pointer issues!

```cpp
#include <iostream>
using namespace std;

class A {
 public:
   A(int sz) {
       //...implement this!
   }

   A(const A& other) {
       //...implement this!
   }

   A& operator=(const A& other) {
       //...implement this!
   }

   //...other functions

   ~A() {
       //...implement this!
   }

 private:
   B* b;  // one dynamically allocated B object; assume B has a
       // default constructor, a copy constructor, and an
       // assignment operator

   int* arr; // dynamically allocated array

   int n; // size of arr (determined by a constructor)

   string str;
};
```

Solution:

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

class A {
 public:
   A(int sz) {
       b = new B;
       arr = new int[sz];
       n = sz;
   }

   A(const A& other) {
       b = new B(*other.b);
       n = other.n;
       arr = new int[n];
       for (int i = 0; i < n; i++) {
           arr[i] = other.arr[i];
       }
       str = other.str;
   }

   A& operator=(const A& other) {
       if (this != &other) {
           A temp(other);
           swap(b, temp.b);
           swap(arr, temp.arr);
           swap(n, temp.n);
           swap(str, temp.str);
       }
       return *this;
   }

   //...other functions

   ~A() {
       delete b;
       delete[] arr;
   }
 private:
   // one dynamically allocated B object; assume B has a
   // default constructor, a copy constructor, and an
   // assignment operator
   B* b;
   // dynamically allocated array
   int* arr;
   // size of arr (determined by a constructor)
   int n;
   string str;
};
```

5. (5 minutes) After being defined by the above code, Jim the Billionaire funded a cloning project and volunteered himself as the first human test subject. Sadly, all his money isn't cloned, so his clone has his name, but has $0. Add the needed function to the Billionaire class so the following main function produces the following output.

```
int main() {
   Billionaire jim = Billionaire("Jimmy");
   Billionaire jimClone = jim;
   cout << jimClone.name << " has "
        << jimClone.account.cash + jimClone.offshore->cash << endl;
   cout << jim.name << " has " << jim.account.cash + jim.offshore->cash
        << endl;
}
```

Output:

```
Jimmy has 0
Jimmy has 1000010000
```

Solution:

```
class Billionaire {
 public:
   Billionaire(string n) : account(10000) {
       offshore = new Account(1000000000);
       name = n;
   }

   // added
   Billionaire(const Billionaire& b) : account(0), name(b.name) {
       offshore = new Account(0);
   }

   ~Billionaire() {
       delete offshore;
   }

   Account account;
   Account* offshore;
   string name;
};
```