Santa Monica College CS 20A: Data Structures with C++ Assignment 3
Spring 2018

Name: Solutions **ID:**

Concepts:

Problem 1:
If you do not explicitly define a copy constructor or assignment operator for your objects, what happens when you attempt to make copies or assign one object to another? In what cases must you define a copy constructor and assignment operator?

If we do not define a copy constructor or overload an assignment operator, the compiler will provide implicit versions of these functions for our class, both of which will only do what is called a "shallow copy" which makes direct copies of the values in each of the member variables.

Shallow copy works fine in cases where we do not dynamically allocate memory within our class. If we do dynamically allocate, that means there is a pointer member variable that stores an address to that dynamically allocated object. A shallow copy of that pointer will copy that exact address into the object that is being copied/assigned to. This results in both classes pointing to the same memory location. This is not only logically unsound, it can result in catastrophic failure of programs using this class.

So in cases where our classes dynamically allocate memory, we must explicitly define a copy constructor and assignment operator for our class, accounting for the dynamically allocated memory.

Problem 2:
What are the primary advantages of inheritance, provide a small programing example with two classes illustrating each point.

Inheritance has three advantageous properties:

- Reuse: Base class defines functionality that is common in all derived classes, the derived classes need only inherit from the base class and automatically have all that functionality.
- Extension: Derived classes may have behavior that is unique to themselves and can define such behavior in addition to the base functionality.
- Specialization: Derived classes may do something slightly different than the base class for a given function, and may override that functionality.

Consider the following two classes:

```
Class Base{
public:
        void set(int x) {m_x=x;}
        int get() {return m_x;}
        virtual  void  print() { cout<<"Hello";}
private:
        int m_x;
};
```

```
Class Derived: public Base{

public:
        void flip() {Base::set(-1* Base::get());}
        virtual void print() {cout<<"Yo";}
};
```

The class Derived sets and gets the same way as Base and so there is no need to redefine those functions since Derived inherits those functions from Base and just reuses them. Derived has extends the Base class in that Derived can flip while the Base cannot. Derived prints like the Base class except is does it in a different way so, Derive specializes that behavior.

Name: Solutions **ID:**

Problem 3:
Suppose Child is a class derived from the class Parent, and the class Grandchild is a class derived from the class Child. This question is concerned with the constructors and destructors for the three classes Parent, Child, and Grandchild. When a constructor for the class Grandchild is invoked, what constructors are invoked and in what order? When the destructor for the class Grandchild is invoked, what destructors are invoked and in what order?

Suppose we have the following declaration:

```
Grandchild nancy;
```

This begins by calling the nancy's default constructor -> Child Constructor -> Parent Constructor -> Parent's member objects' Constructors -> Parent's Constructor Body -> Child's member objects' Constructors -> Child's Constructor Body -> Grandchild's member objects' Constructors ->Grandchild's Constructor Body,

Problem 4:
Suppose you were tasked with developing a program that incorporated the following three objects: Car, Prius, and Engine. How would you organize these objects' relationship with one another? Why?

A Prius **is a type** of Car, so I would have Prius inherit from Car. All Cars **have** Engines so I would have an Engine as a member variable of Car. And since Prius inherits from Car, Prius would automatically have an Engine without us having to include one.

Problem 5:
Why can't we assign a base class object to a derived class variable?

We know that we can do Base *ptr = new Derived because Derived has everything that Base does (and more). But the reverse is not true, the Base classes does not have everything the Derived class does. If we refer to the example from problem 2:

```
Derived *ptr  = new Base;
ptr->flip();  // ERROR!
```

This makes no sense since there is no Base::flip to call.

Problem 6:
Suppose the base class and the derived class each have a member function with the same signature. When you have a pointer to a base class object and call a function member through the pointer, discuss what determines which function is actually called—the base class member function or the derived-class function.

The concept being referred to here is that of Polymorphism which is using some Base pointer to call member functions from Derived classes, referring to the example in Problem 2

```
Base *ptr = new Derived
prt->print();
```

Will print "Yo" since that functions is virtual in the Base class which means the vtable to point to the function corresponding to the class to which ptr points (Derived::print). If print was not virtual that code would print "Hello" instead since there is no indication that it is being overridden.

Name: Solutions **ID:**

Program Problems

For the following programming problems make an attempt **without** using your computer this can be done on scratch paper if you prefer. After the initial attempt you may program the problems to check you work. Your final answer should be written on the assignment.

Problem 7:
This program is supposed to write 1 4 9 16 25 36 49 64 81 100 , but it probably does not. What is the problem with this program? (We're not asking you to propose a fix to the problem.)

```
int* computeSquares(int& n) {
      int arr[10];
      n = 10;
      for (int k = 0; k < n; k++)
            arr[k] = (k + 1) * (k + 1);
      return arr;
}

void f() {
      int junk[100];
      for (int k = 0; k < 100; k++)
            junk[k] = 123400000 + k;
}

int main() {
      int m;
      int* ptr = computeSquares(m);
      f();
      for (int i = 0; i < m; i++)
            cout << ptr[i] << ' ';
}
```

In computeSquares arr is being declared locally to computeSquare, and while the correct values are generated and placed into arr, the moment we leave computeSquare's scope the memory locations associated with arr go away. The return does return the address of the arr that was associated with arr, but it is no longer available for acces. So, when we return to main ptr has that address, and when we try to deference that address inside the for loop we experience a crash.

Name: Solutions **ID:**

Problem 8:
Write delete statements that correctly delete the following dynamically allocated entities.

`int *p1 = new int[10];`
 	
`int *p2[15];`
`for (int i = 0; i < 15; i++)`
`      p2[i] = new int[5];`
`int **p3 = new int*[5];`
`for (int i = 0; i < 5; i++)`
`      p3[i] = new int;`
`int *p4 = new int;`
`int *temp = p4;`
`p4 = p1;`
`p1 = temp;`
	delete p1; // "delete temp;" works too.
	
	for (int i = 0; i < 5; i++)
	      delete p3[i];
	delete[] p3; // This must happen AFTER
	                  // the above for loop.
	for (int i = 0; i < 15; i++)
	      delete[] p2[i];
	
	delete[] p4;

Problem 9:
Consider the code fragment below. It is supposed to construct a 3x4 (3 rows 4 columns) 2d array of integers and set each value to zero. However, as given it does not. Add the proper dereferences (*) or references (&) to make this code work properly:

```
int **rows,  *col1,  *col2,  *col3;

rows   = new  * int[3];       // Create 3 pointers to columns
col1   = new    int[4];       // Create first row with 4 elements
col2   = new    int[4];       // Create second row with 4 elements
col3   = new    int[4];       // Create third row with 4 elements

*(  rows + 0 ) =  &col1[0];   // Point to first row
*(  rows + 1 ) =  &col2[0];   // Point to second row
*(  rows + 2 ) =  &col3[0];   // Point to third row

for (int i = 0; i<3; i++)
      for (int j = 0; j<3; j++)
            *(   *(   rows + i) + j) = 0 // rows[i][j] = 0;
```

j should iterate until it is <4 to make the last column 0, but this won't cause the program to crash.

Name: Solutions **ID:**

Problem 10:
For this problem, you will be asked to write some code to accomplish a particular task given the code fragment below. Each task may depend on the tasks that came before it. Your code must be syntactically correct.

```
class S {
public:
      S(int init) :m_num(init) {}
      S() :m_num(0) {}
      void set(int num) {m_num = num);} // Dangling )
      int get() { return m_num; }
private:
      int num; // Should be m_num
};

S d1, d2(4), d3(-15);
S *sp1, **sp2, ***sp3;
```

Set sp1 to point to d1.

sp1 = &d1;

Using sp2 change the value of *num* in d1 to the value of *num* in d2 (you may not use d1).

sp2 = &sp1;

(**sp2).num = d2.num;

Using sp3 make sp1 point to d3 (you may not use sp1).

sp3 = &sp2;

**sp3 = &d3;

What does the following code output? `cout<< **&*sp3;` If it is a value, state the value, if it is an address state the name of the variable to which the address belongs.

The address of d3.

Name: Solutions **ID:**

Problem 11:
Consider the following program:

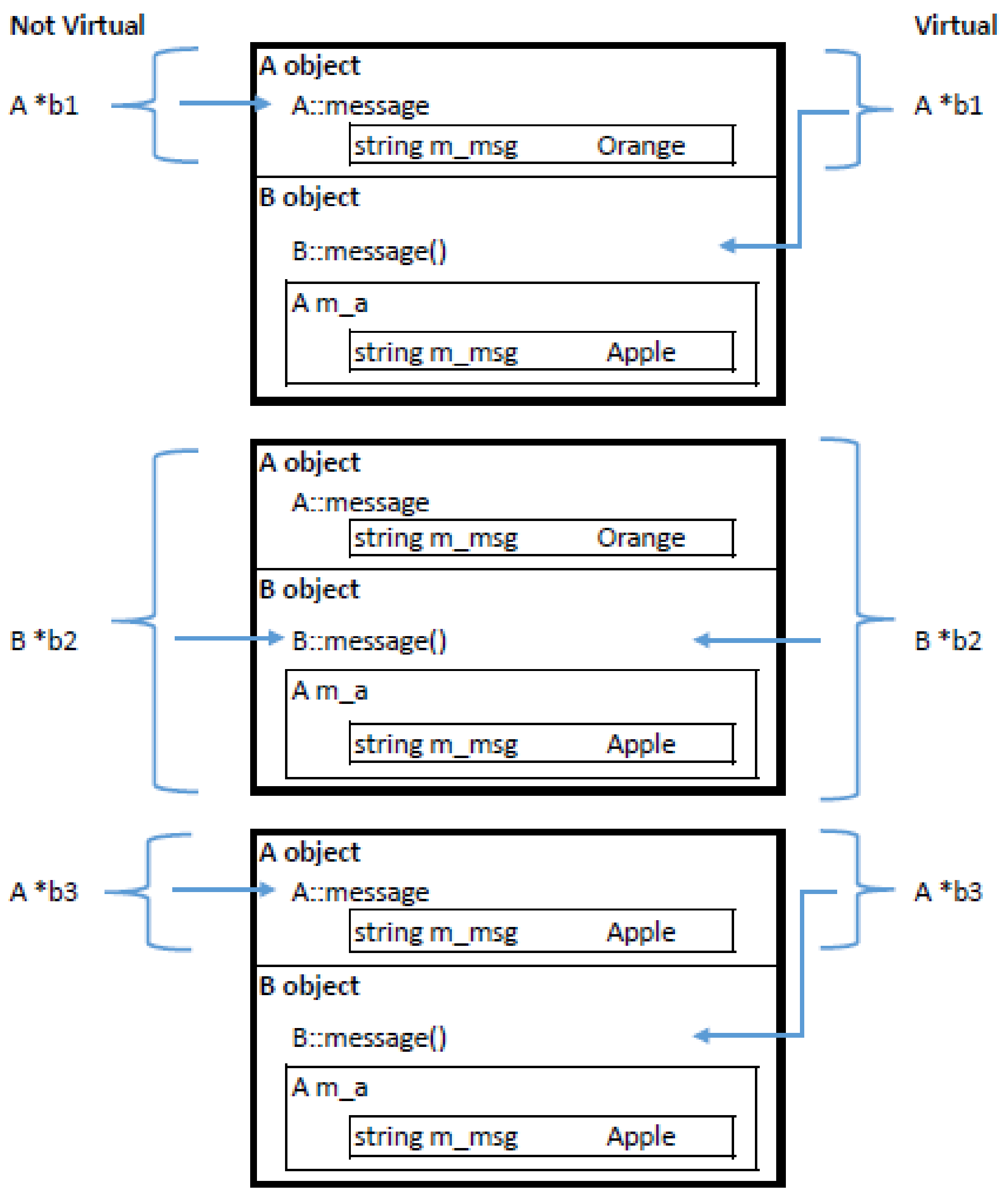
```
class A {
public:
      A() :m_msg("Apple") {}
      A(string msg) : m_msg(msg) {}
      virtual ~A() { message(); }
      void message() const {
            cout << m_msg << endl;
      }
private:
      string m_msg;
};
```

```
class B :public A {
public:
      B() :A("Orange") {}
      B(string msg) : A(msg), m_a(msg) {}
      void message() const {
            m_a.message();
      }
private:
      A m_a;
};
```

```
int main() {
      A *b1 = new B;
      B *b2 = new B;
      A *b3 = new B("Apple");
      b1->message();
      b2->message();
      (*b3).message();
      delete b1;
      delete b2;
      delete b3;
}
```

If A:message is not virtual		If A:message is virtual
Orange	b1->message()	Apple
Apple	b2->message()	Apple
Apple	b3->message()	Apple
Apple	destruct m_a of b1	Apple
Orange	destruct b1	Orange
Apple	destruct m_a of b2	Apple
Orange	destruct b2	Orange
Apple	destruct m_a of b3	Apple
Apple	destruct b3	Apple
6 Apples and 3 Oranges		7 Apples and 2 Oranges

Name: Solutions		**ID:**



Name: Solutions **ID:**

Problem 12:
Consider the following program and generated output:

```cpp
class Security {
public:
   Security(int id)
       :m_id(id), m_badge(id % 10) {}

   ~Security() {
       cout << "Security::~Security: "
            << m_id << endl;
   }
   // Get badge reference
   Badge & badge() {
       cout << "Security::badge: Ret ref "
            << endl;
       return m_badge;
   }
   // Get badge value
   Badge  badgeV() {
       cout << "Security::badge: Ret val "
            << endl;
       return m_badge;
   }
private:
   int m_id;
   Badge m_badge;
};
```

```cpp
class Badge {
public:
    // Constructor
    Badge(int num) :m_num(num) {
       m_stuff = new int[6];
       for (int i = 0; i < 6; i++) {
             m_stuff[i] = num;
       }
    }
    // Destructor
    ~Badge() {
       cout << "Badge::~Badge: " << m_num
             << endl;
       delete[] m_stuff;
    }

    void setNum(int num) {
       m_num = num;
       for (int i = 0; i < 6; i++) {
             m_stuff[i] = num;
       }
    }

    void print() {
       cout << "Badge Num: ";
       for (int i = 0; i < 6; i++) {
             cout <<m_stuff[i];
       }
       cout << endl;
    }
private:
    int m_num;
    int *m_stuff;
};
```

```cpp
int main() {
   Security s(11);
   s.badge().print();
   if (true) {
       Badge b = s.badge();
       b.setNum(2);
       b.print();

       s.badge().print();

       cout << "Main::Leaving if:"
            << endl;
   }
   cout << "Main::Leaving main:" << endl;
}
```

```
Output:
 Security::badge: Ret ref
 Badge Num: 111111
 Security::badge: Ret ref
 Badge Num: 222222
 Security::badge: Ret ref
 Badge Num: 222222
 Main::Leaving if:
 Badge::~Badge: 2
 Main::Leaving main:
 Security::~Security: 11
 Badge::~Badge: 1
```

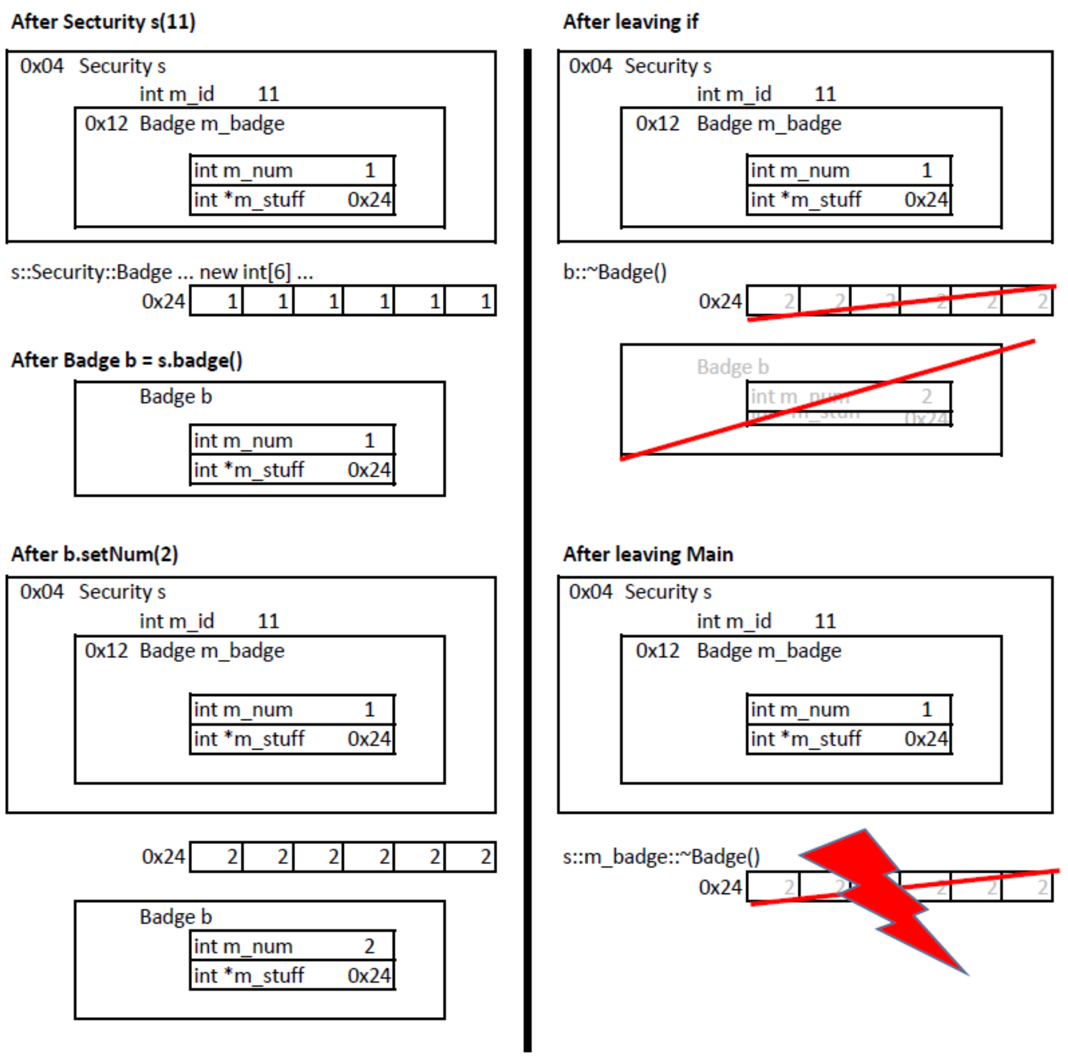
Question on the next page:

Name: Solutions **ID:**

This program runs fine until the very end where we experience a runtime crash. Using a debugger we discover that there is an exception when calling the destructor for Badges at the line `delete[] m_stuff;`. What is causing this crash how would you improve the Badge class to prevent this from occurring?

When we make the assignment in the if statement to b, we make a shallow copy of s::m_badge, which copies over the same address to the dynamically array that m_badge points to. Once we leave the scope of the if statement the destructor for b is called and that deletes the dynamically allocated memory b::m_stuff points too. But, s::m_badge::m_stuff pointed to that same memory location. So, when we leave mean and s's destructor is called and m_badges destructor is called there is no valid array to delete causing the crash.

We should overload the copy constructor for Badge since we are create a new Badge b and making a copy of s::m_badge. Note overloading this assignment operator won't work here since assignments only work for objects that have already been constructed, not newly constructed.


After Sectuity s(11)
0x04 Security s
int m_id 11
0x12 Badge m_badge
int m_num 1
int *m_stuff 0x24
s::Security::Badge ... new int[6] ...
0x24 1 1 1 1 1 1
After Badge b = s.badge()
Badge b
int m_num 1
int *m_stuff 0x24
After b.setNum(2)
0x04 Security s
int m_id 11
0x12 Badge m_badge
int m_num 1
int *m_stuff 0x24
0x24 2 2 2 2 2 2
Badge b
int m_num 2
int *m_stuff 0x24
After leaving if
0x04 Security s
int m_id 11
0x12 Badge m_badge
int m_num 1
int *m_stuff 0x24
b::~Badge()
0x24 2 2 2 2 2 2
Badge b
int m_num 2
int *m_stuff 0x24
After leaving Main
0x04 Security s
int m_id 11
0x12 Badge m_badge
int m_num 1
int *m_stuff 0x24
s::m_badge::~Badge()
0x24 2 2 2 2 2 2