Name:Osbaldo Gonzalez Jr. Group 2
Name: Zimin Yang Group 2

Checked with: Luwei Tan, Group3 Talal Al Shumais, Group 3

Checked with: Jolen Reid, Sasha Saeed, George Vong(group 4)

Rule of 3 Worksheet

1. What is the difference between a shallow copy and a deep copy?

A shallow copy is just an exact, base, no extra frills copy. If you have a pointer to an array, instead of the array being copied, with a shallow copy, only the address of the pointer is copied, therefore, two objects are now sharing the same array. A deep copy involves extra work, as we are no longer just copying a pointer address, but now we want to copy the data that is associated with that pointer.

2. What is the rule of 3?

If you have either a destructor, a copy constructor, or an overload of the "=" operator, then you should implement them all.

3. The copy constructor for a class X has the form `X(const X& b)`. Why is the parameter passed by reference? Explain why you can't define a constructor of the form `X(X b)`?

The parameter is passed by reference, since it is often faster to get the reference of an object (basically the address of) instead of copying the whole object. The reason we need to have a constructor of the form X(const X& b) instead of X(X b) is because we want to be able to copy from const objects or objects with just an r value, and because we don't want to do more copying than needed. Also, if it was X(X& b), we would not be assured that the passed in object will not change.

4. What would be the implication of declaring a copy constructor private?

We would not be able to make copies of objects by calling the constructor when making a new object. The copy constructor would be basically useless, unless you used it to make a new object and return that object inside a public method.

5. What is the difference between destruction and deletion of an object?

Destruction of an object usually relates to the act of cleaning up after all the assets the object held like allocated memory. The deletion of an object usually means just making that object no longer valid to use if either by getting out of scope, or using delete on an object pointer that was dynamically allocated.

6. What problems could a programmer encounter if they defined a destructor for a class but no assignment operator? Illustrate your description with an example class.

Since a shallow copy is done (due to not having an assignment operator implementation), only the address of pointers are copied, and therefore multiple objects can have attributes that point to the exact same address. This is a problem because one the destructor is called on one of the objects, that memory is now freed, and should not be freed again, but since the other objects still have to go through the destructor, we will see multiple objects trying to delete/free the same address, and therefore we will most likely get a segmentation fault. What can be even

scarier is a memory leak, since this will be silent. This is due to the fact that if we are assigning an already created object, with already allocated memory a new address without storing the previous address, all the memory associated with the previous address is now lost, and therefore we have a memory leak!

7. What problems could a programmer encounter if they defined a destructor for a class but no copy constructor? Illustrate your description with an example class.

The exact same reasoning as the previous answer, it just doesn't also include the possibility of leaking memory, since we are creating a completely new object with no previous allocated memory.

8. Which objects are destroyed when the following function exits? Which values are deleted?

```
void f(const Fraction& a)
{
    Fraction b = a;
    Fraction* c = new Fraction(3, 4);
    Fraction* d = &a;
    Fraction* e = new Fraction(7,8);
    Fraction* f = c;
    delete f;
}
```

object b, which is a copy of object a. The object c points to, which is indirectly destroyed by running delete f, since f has the same address as c. object e does NOT get destroyed (with the class destructor, it just leaves scope and is no longer accessible) and most likely will end up being a memory leak. f and d are NOT objects, but object pointers (I also believe that there is d = &a will throw an error, can change by just adding the const qualifier). The passed in "a" variable is not destructed in this function, since it is a reference to a Fraction object, rather than a copy (it would destruct if instead "Fraction a" the parameter, but it is to be noted that the "temp" variable a would be destructed, not the actual passed in object)

9. What error is being committed in the assignment operator for the following class?

```
class String
{
public:
    String(const char right[]);
    String& operator=(const String& right);
private:
    char* buffer;
```

```
};

String::String(const char right[])
{
    len = 0;
    while (right[len] != '\0')
        len++;
    buffer = new char[len + 1];
    for (int i = 0; i < len; i++)
        buffer[i] = right[i];
    buffer[len] = '\0';
}

String& String::operator=(const String& right)
{
   int n = right.length();
    for (int i = 0; i <= n; i++)
        buffer[i] = right.buffer[i];
    return *this;
}
```

The string class does not have a length method, but saying it did, it might also be that the buffer of the calling object is smaller than the of the object being passed in, which would mean we would be accessing memory we are not supposed to access. We could fix this by just stopping short of copying in the right object fully, and only copying up the length of the calling object, or we could just allocate a new buffer for the calling object that would be able to accommodate the size of the right object's buffer.

Use the following class for the following problem. The only purpose of the class is to display a message both when the constructor is invoked and when the destructor is executed.

```
class Trace
{
public:
     Trace(string n);
     ~Trace();
private:
     string name;
```

```
};

Trace::Trace(string n) : name(n)
{
      cout << "Entering " << name << "\n";
}

Trace::~Trace()
{
      cout << "Exiting " << name << "\n";
}
```

10. Extend the class Trace with a copy constructor and an assignment operator, printing a short message in each. Use this class to demonstrate
    a. the difference between initialization
       ```
       Trace t("abc");
       Trace u = t;
       ```
       and assignment.
       ```
       Trace t("abc");
       Trace u("xyz");
       u = t;
       ```
    b. the fact that all constructed objects are automatically destroyed.
    c. the fact that the copy constructor is invoked if an object is passed by value to a function.
    d. the fact that the copy constructor is not invoked when a parameter is passed by reference.
    e. the fact that the copy constructor is used to copy a return value to the caller.

```
class Trace
{
public:
      Trace(string n);
      ~Trace();
      Trace(const Trace&);
      const Trace& operator =(const Trace&);
private:
      string name;
};

Trace::Trace(string n) : name(n)
{
      cout << "Entering " << name << "\n";
```

```
}

Trace::Trace(const Trace& obj)
{
      cout << “Entering copy constructor\n”;
}

const Trace& Trace::operator =(const Trace& obj)
{
      cout << “Using the assignment operator overload!\n”;
      return *this;
}

Trace::~Trace()
{
      cout << "Exiting " << name << "\n";
}
```

a)

```
//Test:
      Trace t("abc");
      Trace u = t;
//Output:
      Entering abc
      Entering copy constructor
      Exiting//no name since didn’t copy the string in copy constructor
      Exiting abc

//Test:
      Trace t("abc");
      Trace u("xyz");
      u = t;
//Output:
      Entering abc
      Entering xyz
      Using the assignment operator overload!
      Exiting xyz
      Exiting abc
```

b) Based on the output, you can see whenever we get out of scope of main, the object automatically calls the destructor.

c) yes can be seen in the first test: Trace u = t;
d) yes can be seen in the second test: u = t;
e) yes it copies the strings (if implemented), used to copy values indeed.