8.10 Copy assignment operator

Default assignment operator behavior

Given two MyClass objects, classObj1 and classObj2, a programmer might write `classObj2 = classObj1;` to copy classObj1 to classObj2. The default behavior of the assignment operator (=) for classes or structs is to perform memberwise assignment. Ex:

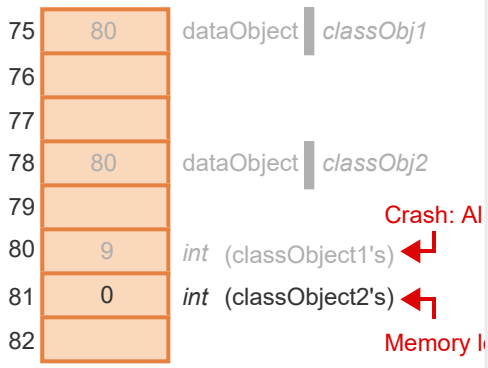
```
classObj2.memberVal1 = classObj1.memberVal1;
classObj2.memberVal2 = classObj1.memberVal2;
...
```

Such behavior may work fine for members with basic types like int or char, but typically is not the desired behavior for a pointer member. Memberwise assignment of pointers may lead to program crashes or memory leaks.

PARTICIPATION ACTIVITY 8.10.1: Basic assignment operation fails when pointer member involved.

Start 2x speed

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

class MyClass {
public:
   MyClass();
   ~MyClass();
   void SetDataObject(const int i) {*dataObject = i;}
   int GetDataObject() const {return *dataObject;}

private:
   int* dataObject;
};

MyClass::MyClass() {
   cout << "Constructor called." << endl;
   dataObject = new int; // Allocate data object
   *dataObject = 0;
}

MyClass::~MyClass() {
   cout << "Destructor called." << endl;
   delete dataObject;
}

int main() {
   MyClass classObject1;
   MyClass classObject2;

   classObject1.SetDataObject(9);
   classObject2 = classObject1;
   cout << "classObject2: ";
   cout << classObject2.GetDataObject() << endl;
```




```
Constructor called.
Constructor called.
classObj2: 9
Destructor called.
Destructor called.
```

```
    return 0;
}
```

Feedback?

PARTICIPATION ACTIVITY 8.10.2: Default assignment operator behavior.

1) The default assignment operator often works for objects without pointer members.
   - True
   - False

2) When used with objects with pointer members, the default assignment operator behavior may lead to crashes due to the same memory being freed more than once.
   - True
   - False

3) When used with objects with pointer members, the default assignment operator behavior may lead to memory leaks.
   - True
   - False

Feedback?

Overloading the assignment operator

The assignment operator (=) can be overloaded to eliminate problems caused by a memberwise assignment during an object copy. The implementation of the assignment operator iterates through each member of the source object. Each non-pointer member is copied directly from source member to destination member. For each pointer member, new memory is allocated, the source's referenced data is copied to the new memory, and a pointer to

the new member is assigned as the destination member. Allocating and copying data for pointer members is known as a ***deep copy***.

The following program solves the default assignment operator behavior problem by introducing an assignment operator that performs a deep copy.

Figure 8.10.1: Assignment operator performs a deep copy.

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

class MyClass {
public:
   MyClass();
   ~MyClass();
   MyClass& operator=(const MyClass& objToCopy);

   // Set member value dataObject
   void SetDataObject(const int setVal) {
      *dataObject = setVal;
   }

   // Return member value dataObject
   int GetDataObject() const {
      return *dataObject;
   }
private:
   int* dataObject;// Data member
};

// Default constructor
MyClass::MyClass() {
   cout << "Constructor called." << endl;
   dataObject = new int; // Allocate mem for data
   *dataObject = 0;
}

// Destructor
MyClass::~MyClass() {
   cout << "Destructor called." << endl;
   delete dataObject;
}

MyClass& MyClass::operator=(const MyClass& objToCopy) {
   cout << "Assignment op called." << endl;

   if (this != &objToCopy) {           // 1. Don't self-assign
      delete dataObject;                  // 2. Delete old dataObject
      dataObject = new int;               // 3. Allocate new dataObject
      *dataObject = *(objToCopy.dataObject); // 4. Copy dataObject
   }

   return *this;
}

int main() {
   MyClass classObj1; // Create object of type MyClass
   MyClass classObj2; // Create object of type MyClass

   // Set and print object 1 data member value
   classObj1.SetDataObject(9);

   // Copy class object using copy assignment operator
   classObj2 = classObj1;

   // Set object 1 data member value
   classObj1.SetDataObject(1);

   // Print data values for each object
   cout << "classObj1:" << classObj1.GetDataObject() << endl;
   cout << "classObj2:" << classObj2.GetDataObject() << endl;

   return 0;
}
```

```
Constructor called.
Constructor called.
Assignment op called.
obj1:1
obj2:9
Destructor called.
Destructor called.
```

Feedback?

PARTICIPATION ACTIVITY 8.10.3: Assignment operator.

1) Declare a copy assignment operator for a class named `EngineMap` using `inVal` as the input parameter name.

```
EngineMap& operator=(
                    );
```

Check Show answer

2) Provide the return statement for the copy assignment operator for the EngineMap class.

Check Show answer

Feedback?

CHALLENGE ACTIVITY 8.10.1: Write a copy assignment.

Write a copy assignment operator for CarCounter that assigns objToCopy.carCount to the new objects's carCount, then returns *this. Sample output for the given program:

```
Cars counted: 12
```

```
1  #include <iostream>
2  using namespace std;
3
4  class CarCounter {
5     public:
6        CarCounter();
7        CarCounter& operator=(const CarCounter& objToCopy);
8        void SetCarCount(const int setVal) {
9           carCount = setVal;
10       }
11       int GetCarCount() const {
12          return carCount;
13       }
14    private:
```

```
15      int carCount;
16  };
17
18  CarCounter::CarCounter() {
19     carCount = 0;
20  }
21
```

Run

View your last submission

Feedback?