Comp151

Order of Construction & Destruction

"Has" relationship

When an object A has an object B as a data member, we say that

"A has a B."

```
class B { ... };

class A
{
    B my_b;

  public:
    // some public members or functions
};
```

- It is easy to see which objects *have* other objects. All you need to do is to look at the class definition.

Example: Order of Constructions


```
class Clock {
  public:
    Clock() { cout << "Constructor Clock\n"; }
    ~Clock() { cout << "Destructor Clock\n"; }
};

class Postoffice {
    Clock clock;
  public:
    Postoffice() { cout << "Constructor Postoffice\n"; }
    ~Postoffice() { cout << "Destructor Postoffice\n"; }
};
```

```
#include <iostream.h>
#include "postoffice.h"
int main() {
    cout << "Beginning of main\n";
    Postoffice x;
    cout << "End of main\n";
    return 0;
}
```

```
Beginning of main
Constructor Clock
Constructor Postoffice
End of main
Destructor Postoffice
Destructor Clock
```

Order of Constructions: Remarks

- When an object is constructed, all its data members are constructed first.
- Note that the `Clock` constructor is called *before* the `Postoffice` constructor; but, the `Clock` destructor is called *after* the `Postoffice` destructor.
- The order of destruction is the exact *opposite* of the order of construction.

Order of Constructions with Owned Objects


```
class Clock
{
  public:
    Clock() { cout << "Constructor Clock\n"; }
    ~Clock() { cout << "Destructor Clock\n"; }
};

class Postoffice
{
    Clock *clock;
  public:
    Postoffice() { clock = new Clock; cout << "Constructor Postoffice\n"; }
    ~Postoffice(){ cout << "Destructor Postoffice\n"; }
};
```

```
Beginning of main
Constructor Clock
Constructor Postoffice
End of main
Destructor Postoffice
```

Order of Constr. with Owned Objects: Remarks

- Now the `Postoffice` *owns* the `Clock` (since it creates it dynamically)
- The `Clock` object is constructed in the `Postoffice` constructor, but it is never destructed, since we have not explicitly called `delete`.
- Remember that objects on the heap are never destructed automatically, so we have just created a memory leak.
- When object A owns object B, A is responsible for B's destruction.

Order of Constr. with Owned Objects: Fix


```
class Clock
{
  public:
    Clock() { cout << "Constructor Clock\n"; }
    ~Clock() { cout << "Destructor Clock\n"; }
};

class Postoffice
{
    Clock *clock;
  public:
    Postoffice() { clock = new Clock; cout << "Constructor Postoffice\n"; }
    ~Postoffice() { cout << "Destructor Postoffice\n"; delete clock; }
};
```

```
Beginning of main
Constructor Clock
Constructor Postoffice
End of main
Destructor Postoffice
Destructor Clock
```

Order of Constructions w/ Multiple Objects


```
class Clock
{
    int HHMM;
  public:
    Clock() : HHMM(0) { cout << "Constructor Clock\n"; }
    Clock(int hhmm) : HHMM(hhmm)
        { cout<<"Constructor Clock at "<< HHMM <<endl; }
    ~Clock() { cout << "Destructor Clock at " << HHMM << endl; }
};

class Room
{
  public:
    Room() { cout << "Constructor Room\n"; }
    ~Room() { cout << "Destructor Room\n"; }
};

class Postoffice
{
    Clock clock;
    Room room;
  public:
    Postoffice() { cout << "Constructor Postoffice\n"; }
    ~Postoffice() { cout << "Destructor Postoffice\n"; }
};
```

```
Beginning of main
Constructor Clock
Constructor Room
Constructor Postoffice
End of main
Destructor Postoffice
Destructor Room
Destructor Clock at 0
```

†† Note that the 2 data members, `Clock` and `Room` are constructed first, in the order in which they appear in the `Postoffice` class.

Order of Construction w/ Nested Objects

Let's move the clock to the room.

```
class Clock
{
  public:
    Clock() { cout << "Constructor Clock\n"; }
    ~Clock() { cout << "Destructor Clock\n"; }
};

class Room
{
    Clock clock;
  public:
    Room() { cout << "Constructor Room\n"; }
    ~Room() { cout << "Destructor Room\n"; }
};

class Postoffice
{
    Room room;
  public:
    Postoffice(){cout << "Constructor Postoffice\n"; }
    ~Postoffice(){cout << "Destructor Postoffice\n"; }
};
```

```
Beginning of main
Constructor Clock
Constructor Room
Constructor Postoffice
End of main
Destructor Postoffice
Destructor Room
Destructor Clock
```

Summary

- When an object is constructed, its data members are constructed first.
- When the object is destructed, the data members are destructed *after* the destructor for the object has been executed.
- When object A owns other objects, remember to destruct them as well in A's destructor.
- By default, the default constructor is used for the data members.