COMP2012H

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”.

```cpp
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

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

class Clock {
public:
    Clock() { cout << "Constructor Clock" << endl; }
    ~Clock() { cout << "Destructor Clock" << endl; }
};

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

int main()
{
    cout << "Beginning of main" << endl;
    Postoffice x;
    cout << "End of main" << endl;
    return 0;
}
```

Here's the output:

- 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.

• The order of destruction is the exact opposite of the order of construction: the Clock constructor is called before the Postoffice constructor; but the Clock destructor is called after the Postoffice destructor.

• As always, construction of data member objects is done by calling the appropriate constructors.
  – If you do not do this explicitly, then the compiler will assume the default constructors should be used. Make sure they exist! That is,
    Postoffice::Postoffice() {}
    is equivalent to
    Postoffice::Postoffice() : clock() {}
  – Or, you may control construction of data member objects by calling their appropriate constructors using the member initialization list syntax.
Order of Constructions with Owned Objects

class Clock {
    public:
        Clock() { cout << "Constructor Clock" << endl; }
        ~Clock() { cout << "Destructor Clock" << endl; }
};

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

Here is the output:
Beginning of main
Constructor Clock
Constructor Postoffice
End of main
Destructor Postoffice
Order of Construction with Owned Objects: Remarks

What happened…?

- 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!

- The lesson: When object A owns object B, A must be responsible for B's destruction.
Order of Constructions with Owned Objects: Fix

class Clock {
    public:
        Clock() { cout << "Constructor Clock" << endl; }
        ~Clock() { cout << "Destructor Clock" << endl; }
    };

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

Here is the new output:
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" << endl; }
    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" << endl; }
    ~Room() { cout << "Destructor Room" << endl; }
};

class Postoffice {
    Clock clock;
    Room room;

public:
    Postoffice() { cout << "Constructor Postoffice" << endl; }
    ~Postoffice() { cout << "Destructor Postoffice" << endl; }
};

Here is the output:

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" << endl; }
    ~Clock() { cout << "Destructor Clock" << endl; }
};

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

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

Here is the output:
Beginning of main
Constructor Clock
Constructor Room
Constructor Postoffice
End of main
Destructor Postoffice
Destructor Room
Destructor Clock
Order of Constructions with Temporary Objects

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

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

class Postoffice {
    Clock clock;
public:
    Postoffice() {
        clock = Clock(1800); // creates and destroys a temporary object
        cout << "Constructor Postoffice" << endl;
    }
    ~Postoffice() { cout << "Destructor Postoffice" << endl; }
};
```

Here’s the output:

- Beginning of main
- Constructor Clock
- Constructor Clock at 1800
- Destructor Clock
- Constructor Postoffice
- End of main
- Destructor Postoffice
- Destructor Clock

- Here a temporary clock object is created by `Clock(1800)`.
- Like a ghost, it is created and destroyed behind the scenes.
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 (via pointers), remember to **explicitly** destruct them in A's destructor.

• By default, the default constructor is used for the data members.