If the file Foo.h defines the class Foo, when does another file require  
you to say  
 #include "Foo.h"  
and when can you instead simply provide the incomplete type declaration  
 class Foo;

You have to #include the header file defining a class when  
 \* you declare a data member of that class type  
 \* you declare a container (e.g. an array or a vector) of objects of that class type  
 \* you create an object of that class type  
 \* you use a member of that class type

class Blah  
{  
 ...  
 void g(Foo f, Foo& fr, Foo\* fp); // just need to say class Foo;  
 ...  
 Foo\* m\_fp; // just need to say class Foo;  
 Foo\* m\_fpa[10]; // just need to say class Foo;  
 vector<Foo\*> m\_fpv; // just need to say class Foo;

Foo m\_f; // must #include Foo.h  
 Foo m\_fa[10]; // must #include Foo.h  
 vector<Foo> m\_fv; // must #include Foo.h  
};

void Blah::g(Foo f, Foo& fr, Foo\* fp)  
{  
 Foo f2(10, 20); // must #include Foo.h  
 f.gleep(); // must #include Foo.h  
 fr.gleep(); // must #include Foo.h  
 fp->gleep(); // must #include Foo.h  
}

Order of construction:

Member variables are constructed in order

The current class constructor is executed

Order of destruction:

The current class destructor is executed first

The member variables are destructed in the reverse order

Constructor: Inside -> Outside

Go as deep as you can inside nested layer of classes

Destructor: Outside -> Inside

Destruct yourself first, and then destruct your member variables

What happens when an object gets constructed?

Constructor goes through 3 steps:

* 1. Not relevant yet
  2. Construct the data members, using the **member initialization list**; if a member is not listed:
  3. If a data member is of a builtin type, it's left uninitialized
  4. If a data member is of a class type(Ex. String..), a constructor is called for it
  5. Execute the body of the constructor

class String

{

public:

String(const char\* value = ""); //arrays are passed into function as pointer to first element of array

String();

~String();

String(const String& other);

String& operator=(const String& rhs);

void swap(String &other);

private:

char\* m\_text;

int m\_len;

}

String::String(const char\* value)

{

//const is saying if you follow the value ptr and arrive at a character, you're not allowed to change that character

//But you're allow to change what the pointer points to!

if(value == nullptr)

value = ""; //a pointer is created for the ""

//"" is a one element array with a nullbyte, can be used as a ptr

m\_len = strlen(value);

m\_text = new char[m\_len+1];

strcpy(m\_text, value);

}

String::~String()

{

//C++ language rule: If you allocate something as a single object you call delete to the pointer to that object

//If you allocate something as an array of objects, then to get rid of it, you're required to use:

delete [] m\_text;

}

String::String(const String& other) //other is the string that is being used to construct t

{

m\_len = other.m\_len;

m\_text = new char[m\_len+1];

strcpy(m\_text, other.m\_text);

}

String& String::operator=(const String& rhs)

{

String temp(rhs);

swap(temp); //no way this can fail

return \*this;

}

class A {

public:

A(int sz) {

b = new B;

arr = new int[sz];

n = sz;

}

A(const A& other) {

**b = new B(\*other.b);**

n = other.n;

arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = other.arr[i];

}

str = other.str;

}

String& String::operator=(const String& rhs)

{

if(this != &rhs){ //check if they're pointing to the same address

delete [] m\_text;

m\_len = rhs.m\_len;

m\_test = new char[m\_len+1];

strcpy(m\_test, rhs.m\_text);

}

return \*this;

}

A& operator=(const A& other) {

if (this != &other) {

delete b;

delete [] arr;

**b = new B(\*other.b);**

n = other.n;

arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = other.arr[i];

}

str = other.str;

}

return \*this;

}

//...other functions

~A() {

delete b;

delete [] arr;

}

private:

//one dynamically allocated B object; assume B has a default

//constructor, a copy constructor, and an assignment operator

B\* b;

//dynamically allocated array

int\* arr;

//size of arr (determined by a constructor)

int n;

string str;

};