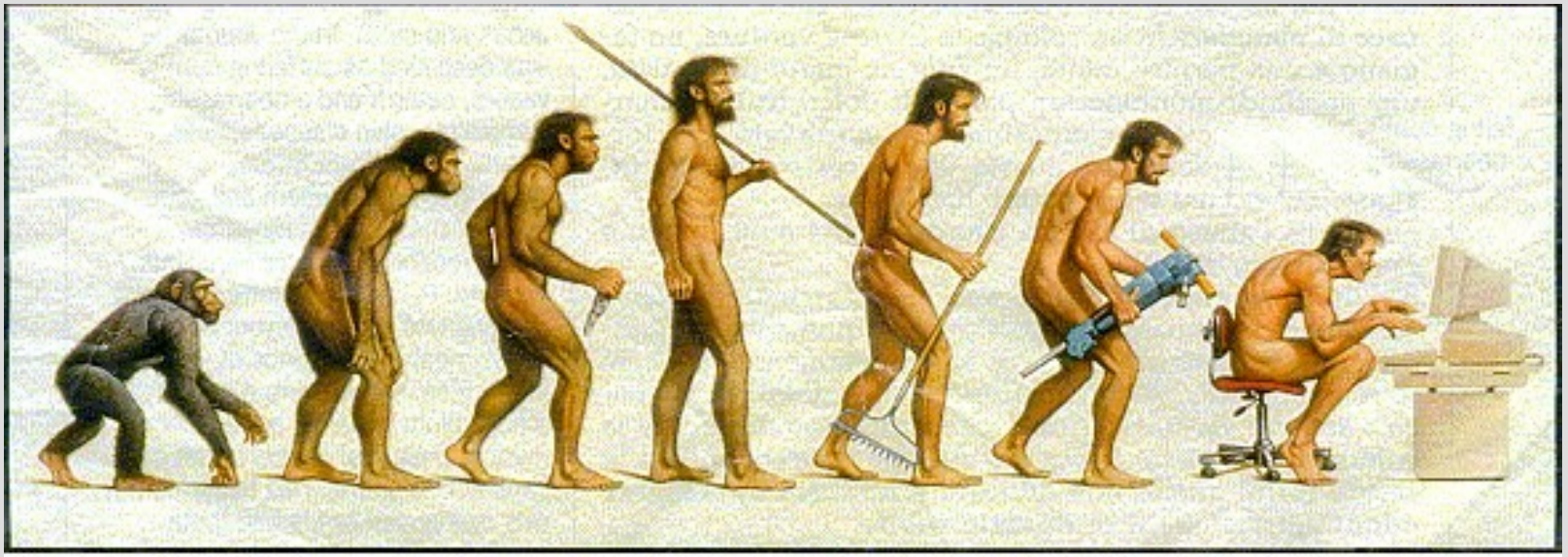
Inheritance

Ch 15.1-15.2



Somewhere, something went terribly wrong

Highlights

- protected

```
class Parent{
protected:
    int x;
};
```

- reuse constructors

```
Child::Child() : Parent()
{
    // runs parent default constructor before itself
}
```

Derived classes

Let's make this story into code!

To create create a child class from a parent class, use a : in the (child) class declaration

child class

parent class

```
class Dunecat : public ArrakianSandworm {
public:
    Dunecat();
};
```

(See last time: dunecat.cpp)

Constructors + inheritance

Constructors need to be run every time you make an object...

Now that objects have multiple types what constructors are being run?

Both actually (again)

(See: computerConstructor.cpp)

Constructors + inheritance

If you do not specify what constructor to use, it will use the default constructor
(or give an error if this does not exist)

You can also specify a non-default constructor by using a ":" after the child's constructor

```
Laptop::Laptop(string p, string r, double l) : Computer(p, r)
{
    //cpu = p; // done in Computer constructor
    //memory = r; // done in Computer constructor
    batteryLife = l;
}
```

(See: computerConstructorV2.cpp)

protected

We know about two scopes for variables:

1. public (anyone, anywhere can use)
2. private (only my class can use)

But there is a third:

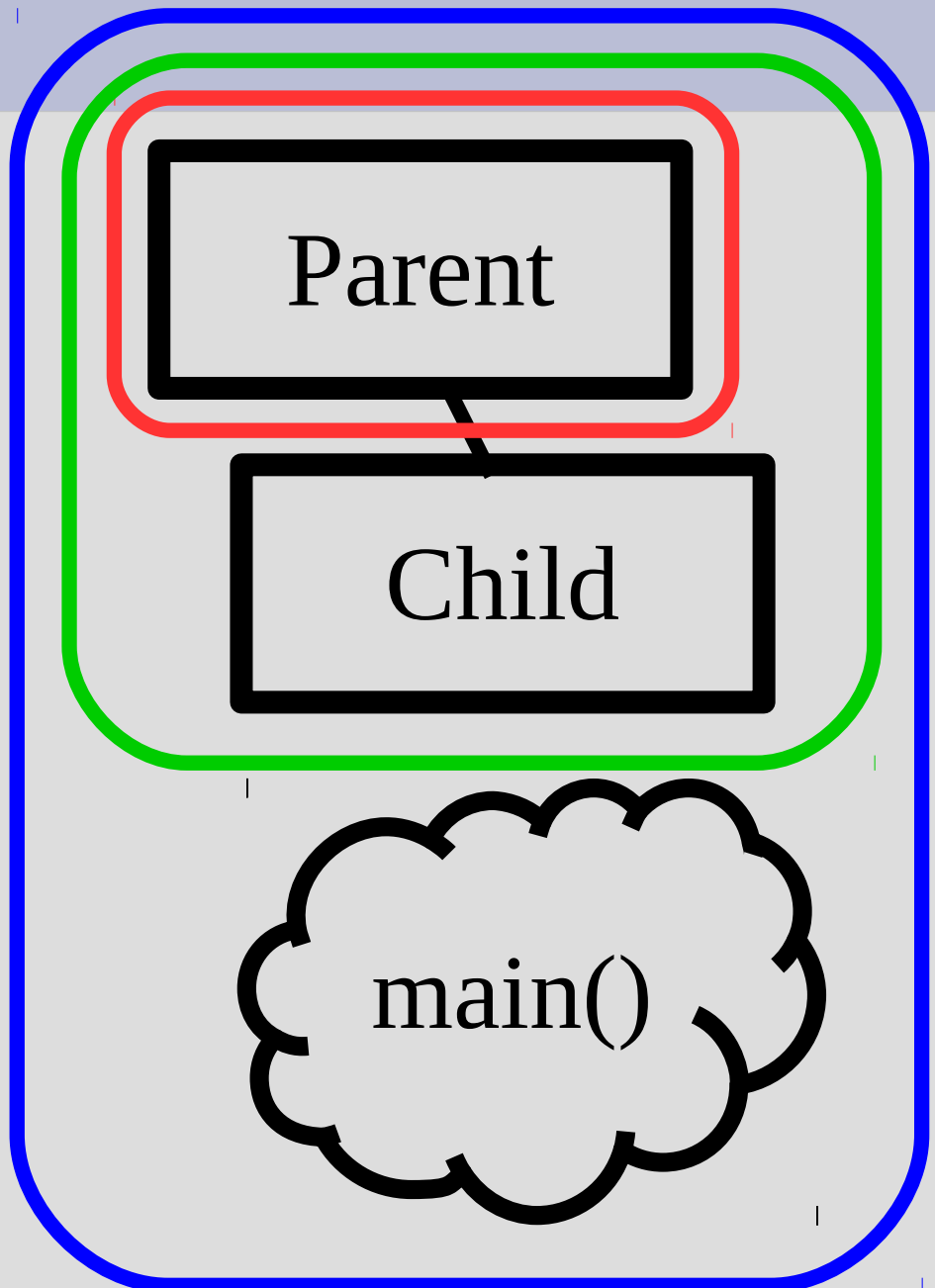
3. protected (me or my children can use)

If you think your children will modify/use a variable, make it protected
(See: classScopes.cpp)

protected

Picture:
Red = private
Green = protected
Blue = public

Variables should be either private or protected



protected

While children technically inherit the private variables/functions, they cannot use them

So effectively, they do not inherit these

It is not considered bad practice to make variables protected (unlike public)

Does access matter?
Yes, because computer viruses

Redefine functions

As children add functionality to a parent class, they may want to redefine some functions

This is different than overloading, where you create multiple versions with the same name

When you redefine, you are basically replacing an old function with a new version

(See: computerRedefine.cpp)

Redefine functions

After you have redefined a function,
the default name will go to the child's version

However, you can still access the parent's version by using “::” (class affiliation)

```
Laptop rightHere = Laptop("2.7 GHz i5", "8 GB DDR3", 3);
rightHere.displaySpecs();
// runs Laptop's version of displaySpecs
rightHere.Computer::displaySpecs();
// runs Computer's version of displaySpecs
```

Not inherited

As we saw before, constructors are not really inherited (though they are called)

overloading operators will also not be inherited (as computer cannot convert parent into child class)

Destructors are also not inherited, but the parent's version of the destructor will always run (See: childDestructor.cpp)