lecture7

what:classes
what more:classes and inheritance

class

class why
C has no objects-->no way of encapsulating(封装) data
containers are classes defined in the STL

Why classes?

- C has no **objects**
- No way of **encapsulating** data and the functions that operate on that data
- No ability to have object-oriented programming (OOP) design patterns

what classes

What is object-oriented-programming?

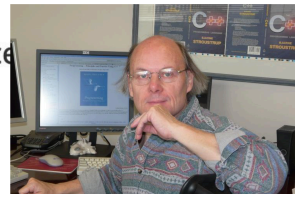
- Object-oriented-programming is centered around **objects**
- Focuses on design and implementation of classes!
- Classes are the **user-defined types** that can be declared as an object!

distinction：structs vs classes
structs:没有权限区别（structures which are classes without restrictions）

classes:有权限的区分

Comparing ‘struct’ and ‘class’

> classes *containing a sequence of objects of various types, a set of functions for manipulating these objects, and a set of restrictions on the access of these objects and function;*
>
> structures *which are classes without access restrictions;*
>
> Bjarne Stroustrup, The C++ Programming Language – Reference Manual, §4.4 Derived types



A good example

A backpack

Struct

Class

在类的定义和执行中，.h文件和.cpp文件用处不同

Header File (`.h`) vs Source Files (`.cpp`)

	Header File (.h)	**Source File (.cpp)**
Purpose	Defines the interface	Implements class functions
Contains	Function prototypes, class declarations, type definitions, macros, constants	Function implementations, executable code
Access	This is shared across source files	Is compiled into an object file
Example	`void someFunction();`	`void someFunction() {...};`

头文件主要是定义与声明接口和函数
源文件是具体写函数的实现

如何设计一个类：（design）
constructor
private/public functions/variable
destructor

constructor
.h文件

.h file

```
class StanfordID {
private:
    std::string name;
    std::string sunet;
    int idNumber;

public:
    // constructor for our StudentID
    StanfordID(std::string name, std::string sunet, int idNumber);
    // method to get name, sunet, and idNumber, respectively
    std::string getName();
    std::string getSunet();
    int getID();
}
```

.cpp文件

.cpp file (implementation)

```
#include "StanfordID.h"
#include <string>

StanfordID::StanfordID(std::string name, std::string sunet, int idNumber) {
    this->name = name;
    this->state = state;
    this->age = age;
}
```

Use this `this` keyword to disambiguate which 'name' you're referring to.

注意这个this，因为.h文件里有变量name，后续cpp文件里传入参数是name，就不确定这个name=name具体是哪个name，用this可以确定this->name就是之前.h文件里定义好的name

destructor

The destructor

.cpp file (implementation)

```
#include "StanfordID.h"
#include <string>

StanfordID::~StanfordID() {
    // free/deallocate any data here

    delete [] my_array; // for illustration
}
```

The destructor is not explicitly called, it is automatically called when an object goes out of scope

Inheritance（继承）

继承的优点:父类 子类

Dynamic Polymorphism(多态)

Extensibility

example：
一个设计形状的例子

Rectangle class definition

.h file

```
class Shape {
public:
    virtual double area() const = 0;
};
. . . .
class Rectangle: public Shape {
public:
    // constructor
    Rectangle(double height, double width): _height{height}, _width{width}
{};
    double area() const {
        return _width * _height;
    }
private:
    double _width, _height;
};
```

virtual就是虚函数，这个是多态里的内容，目的是为了子类重写这个函数，这个虚函数赋值为0，意味着是纯虚函数

继承的写法就是在子类上加：
class Rectangle:public Shape

继承的不同类型

Types of inheritance

Type	`public`	`protected`	**`private`**
Example	`class B: public A {...}`	`class B: protected A {...}`	**`class B: private A {...}`**
Public Members	Are public in the derived class	Protected in the derived class	Privated in the derived class
Protected Members	Protected in the derived class	Protected in the derived class	Private in the derived class
Private Members	Not accessible in derived class	Not accessible in derived class	Not accessible in derived class

关于protected的情况，关键是为了继承的类型，继承的时候有public继承和private继承

class Rectangle:public Shape 继承后，这种父类成员变量和函数随意访问
class Rectangle:private Shape 这种都不可以访问
（不写默认private）

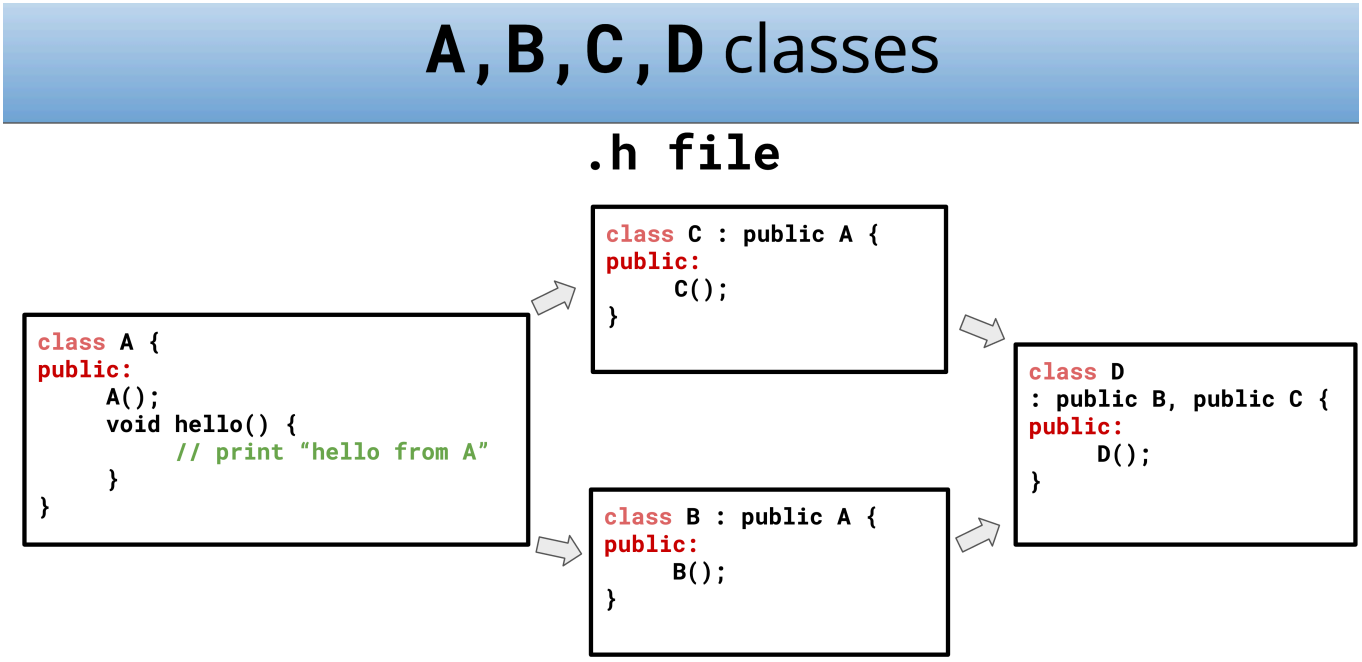
关于protected

```
class Shape {
protected:
    int width; // 只有家族成员能碰
public:
    void setWidth(int w) { width = w; }
};

class Rectangle : public Shape {
public:
    int getArea() {
        return width * 10; // 成功！子类可以直接访问父类的 protected 成员
    }
};

int main() {
    Rectangle rect;
    // rect.width = 20; // 报错！外部不可见，保证了封装性
    rect.setWidth(20);  // 成功！通过 public 接口操作
}
```

继承中出现一个问题（菱形继承）
简单来说就是A是父类，BC都继承自A，D继承了BC，那么D调用BC里继承A的函数，那么出现报错，到底输出谁




```
D obj {};

obj.B::hello()  // call B's hello method

obj.C::hello() // call C's hello method

obj.hello() // whose method do I call ???
```

引入虚拟继承可以解决这个问题（virtual） public private

Solution? Inherit Virtually!

.h file

```
class C : virtual public A {
public:
     C();
}
```

```
class B : virtual public A {
public:
     B();
}
```

This creates a shared instance of **A** between **B** and **C**!

这下只输出一个hello了

Fixed!

```
D obj {};

obj.B::hello()  // call B's hello method

obj.C::hello() // call C's hello method

obj.hello() // no longer ambiguous :)
```

Recap

1. Classes allow you to encapsulate functionality and data with access protections

2. Inheritance allows us to design powerful and versatile abstractions that can help us model complex relationships in code.

3. These concepts are tricky – this lecture *really* highlights the power of C++

对比java与c++

1.Java没有指针，垃圾回收自动

2.Java默认多态，所有非静态方法默认是虚函数（这种方便多态但是浪费空间）；c++默认是实函数

封装：Encapsulation

继承：Inheritance

多态：polymorphism