类

C结构体的缺陷

```
手表 表;
设置时间(&表,时间);
表.设置时间(时间);        // 更符合人类的思维
```

- 结构体没有访问控制，任何人都可以访问其成员
- 使用全局函数对其设置
- 利用函数指针模拟对象调用但也会显示传入结构体地址，容易造成如下情况

```
手表 手表1;
手表 手表2;

手表1.设置时间(&手表2, 时间);        // 通过实例1成员设置了实例2的数据
```

访问权限关键字

- `class` 定义了一个新的变量
- `class` 定义了新的作用域，称之为类作用域
- 关键字
    1. `public`：公有，允许所有人访问类的成员，类的作用域内外都可以
    2. `private`：私有，只允许类作用域内访问（默认权限）
    3. `protected`：保护，允许类作用域和子类访问
- 语法层面不可修改
- 每个关键字的影响范围从本关键字开始直到遇到下一个关键字

规范

类内的函数叫做成员函数（方法），数据叫做数据成员
类的声明放在头文件里面，成员函数放在cpp文件中

类的内存布局

- 现阶段，C++内存布局与C一致
- 不同变量的数据是单独的，成员函数是共享的
- 空类的大小为**1**（占位用）

this指针与__thiscall

- 成员函数通过`ecx`传入变量的地址，这个地址赋给了`this`指针，`this`指针的类型为`type * const`

- 使用`ecx`传入`this`指针的调用方式称为`__thiscall`（不是关键字）
- 成员函数的调用约定默认使用`__thiscall`，也可以使用其他调用约定，修改后使用栈传参

成员函数指针

```
class my_class;
typedef void (my_class::*foo_ptr)(int, int);

class my_class {
    ...
    void foo(int a, int b)
    {
        ...
    }
    ...
}


// 调用
my_class mc;
my_class *mc_ptr;
foo_ptr myfoo = &my_class::foo;
(mc.*myfoo)(1, 2);
(mc_ptr->*myfoo)(1, 2);
```

class与struct的区别

- `class`默认**私有**，`struct`默认**公有**

面向对象编程

面向对象思想中，对象包含**特征**和**行为——数据成员**和**成员函数**

```
class 笔记本电脑 {
    // 数据
    尺寸
    屏幕参数
    硬件型号
    电池类型

    // 行为（操作）
    鼠标操作
    键盘操作
    音频播放
    视频播放
};
笔记本电脑 程序员A的笔记本;
笔记本电脑 程序员B的笔记本;
```