# 结构体

由一系列具有相同类型或是不同类型的数据构成的数据集合。是一种数据格式。

为什么有结构体：扩展了数据类型，用来更加清晰的描述事物的特性。（c语言为了解决无法整合数据而诞生）

# 声明和初始化结构体

### 声明：

struct Box

{

float Height;

float Width;

float Length;

};

### 初始化

1. 构建后为成员变量进行初始化（比较笨）

Box box\_01;

box\_01.Width = 0;

1. 借助｛｝进行初始化

Box box\_01{};

1. 借助｛｝给成员变量进行赋值初始化

Box box\_01{0, 15.0f};//注意第一个成员变量值为0，第二个成员变量值为15，第三个成员变量值为0

1. 直接在构建时为成员变量初始化

struct Box

{

float Height = 60.5f;

float Width = 11.2f;

float Length = 3.0f;

};

直接在构建结构体时为成员变量进行初始化

**注意：当使用此种方法进行初始化后，不能再使用花括号｛｝（2和3都不行了）进行初始化**

# 点和双冒号的区别

. 和：：

一个点，是用来帮助对象进行成员属性或是函数引用操作时使用。对象只有是栈对象才会使用点。（成员操作运算符）

双冒号，是用来做空间对象或是空间内容引用操作。也可以用来引用静态函数或是静态成员变量，或是枚举选项操作。

## 结构体的大小

数据对齐，结构体中计算数据大小时，将向最大数据类型看齐，其他数据则采用对齐方案。即，小数据之间进行拼接，如果无法完整填充一个最大数据大小，则拼接。如果超过，则重新开启一个新的扩充尺寸（最大数据类型大小），再次填充

# 共用体

是一种数据格式，能够存储不同的数据类型，将他们放在同一块内存中，但是只能同时使用其中的一种类型。共用体只能存储基本数据类型，例如**整形，浮点型，布尔类型。**

应用场景：解决硬件平台内存吃紧问题，共用体可以灵活的应对不同的数据类型要求，根据需求进行调整

### 构建共用体:

union U1

{

char A;

short B;

int C;

bool D;

};

### 声明：

U1 a;

注意：如果将共用体的变量进行赋值，后面再向其他成员变量赋值时，将会覆盖之前成员变量的值。

### 共用体的特点

1. 共用体的大小永远等于最大数据成员的大小
2. 共用体同一时刻只能有效存储一种数据（也就是只有一个成员变量数据有意义）
3. 共用体一般用于节省数据内容，同时又标明了多种数据名称。
4. 使用时注意填充数据覆盖问题。（后存储数据可能覆盖之前存储的数据）

# 类

用来抽象描述对象属性行为的模版。我们可以借助类来创建具体的操作对象，丰富我们编程过程中的数据类型。

### 访问修饰符，修饰成员

类中创建的对象或是函数默认是不可以访问（私有的），如果我们想要在其他地方进行操作和访问必须添加访问修饰符，这体现了OOP编程的封装特性。

1. Public：公有访问，子类可以访问父类的成员，类的实例可以在外部访问类的成员。
2. Protected：受保护，只允许子类及本类进行成员访问，类的实例无法进行访问。
3. Private：私有，只允许本类进行成员访问，子类和类的实例均无法访问类的成员。

### 访问修饰符，修饰继承关系

1. Public继承：基类中的public成员，protected成员，private成员的访问属性在派生类中变成了：public，protected，private
2. Protected继承：基类中的public成员，protected成员，private成员的访问属性在派生类中变成了：protected，protected，private
3. Private继承：基类中的public成员，protected成员，private成员的访问属性在派生类中变成了：private，private，private

|  |  |  |  |
| --- | --- | --- | --- |
| 纵继承关系，横成员修饰 | Public | Private | Protected |
| Public | Public | Private | Protected |
| Private | Private | Private | Private |
| Protected | Protected | Private | Protected |

A： 修饰符B

只会影响继承自A类子类中访问B类的成员权限

类继承会承接父类中的所有成员属性和方法

# 构造函数

一种特殊的类的成员函数，没有返回类型，构造函数的名称必须和类名相同。对象被创建时会调用构造函数，我们可以显式定义构造函数，并且重载更多的构造函数。

# 析构函数

没有返回类，函数名称与类名相同，在函数前加了~符号，当对象被释放时调用的函数。析构函数无需主动调用，由系统调用，析构函数无法被重载，如果没有显式定义析构函数，编译器总会帮我们构建析构函数。