# 带参数构造函数

构造函数的主要目的用来进行对象构建时初始化数据。我们可以将构造函数进行重载，**为对象构建时带入数据，进行初始化操作**。

强制进行数据初始化。

# 重载

C++允许我们声明函数时，名称相同，但是函数的形参（类型，个数，顺序）必须不同（返回类型不同不可以），也就是说同一个名称的函数表现不同的运算功能，我们称这种操作为函数重载。

我们就不看返回类型，就看参数！看个数，看顺序，看类型

**函数重载的目的：**通过同一个函数名称，针对不同的数据，进行相似的操作。减少额外编写函数名称。提升了多个同目的，不同参数的函数之间整合。降低了程序的阅读困难。

**重载场景举例：**战斗中的角色受伤函数，如果你给予的是炮弹伤害，我的受伤函数是Hurt，如果是子弹伤害，我的受伤函数也是Hurt，差异就在于给予伤害的源。而伤害的本质是相同的。

# 继承关系中的构造函数显示重载

1. 父类没有显式编写构造函数，子类可以根据需求编写构造函数，或是不编写构造函数。
2. 父类显式编写了构造函数，并且构造函数没有参数。子类可以根据需求编写构造函数，或是不编写构造函数。
3. 父类显式编写了构造函数，构造函数带有参数，并且只有一个构造函数。则子类必须编写构造函数，并且需要明确给予父类构造函数调用。
4. 父类显式编写了多个构造函数，改造函数均带有参数。则子类必须编写构造函数，并且需要明确给予父类构造函数调用。
5. 父类显式编写了多个构造函数，并且构造函数中有一个是无参构造函数。则子类可以根据需求编写构造函数，或是不编写构造函数。

# C++中的数据存储空间

1. 堆 heap ：  
   由new分配的内存块，其释放编译器不去管，由我们程序自己控制（一个new对应一个delete）。如果程序员没有释放掉，在程序结束时OS会自动回收。涉及的问题：“缓冲区溢出”、“内存泄露”
2. 栈 stack ：  
   是那些编译器在需要时分配，在不需要时自动清除的存储区。存放局部变量、函数参数。  
   存放在栈中的数据只在当前函数及下一层函数中有效，一旦函数返回了，这些数据也就自动释放了。
3. 全局/静态存储区 （.bss段和.data段） ：  
   全局和静态变量被分配到同一块内存中。在C语言中，未初始化的放在.bss段中，初始化的放在.data段中；在C++里则不区分了。
4. 常量存储区 （.rodata段） ：  
   存放常量，不允许修改（通过非正当手段也可以修改）
5. 代码区 （.text段） ：  
   存放代码（如函数），不允许修改（类似常量存储区），但可以执行（不同于常量存储区）

# 内存泄漏

在程序运行期间，程序员主动进行堆中的内存申请。这些内存程序不进行管理的，由申请的人进行管理，而我们在使用期间，对于不使用的内存忽略了释放，那么这些内存将会无法被回收，重复利用。导致了内存越来越少，出现了泄漏。

# 指针

一种复合数据类型，可以用来指向堆中的数据空间，根据**构建指针的数据类型**确认空间中的数据大小，指针被分配在栈中。**指针本身的释放不需要我们管理，但是指针指向的空间，需要我们管理！**

指针主要用来存储地址！这是指针的主要用途。而地址中存放的就是数据（可以是基本数据类型（整形，浮点型），也可以是自定义数据类型（结构体，类），也可以是函数）

**枚举可以构建成为指针（这本质是合理的），但是指针无法操控枚举的枚举项。**

**构建指针时的数据类型，决定了指针指向的空间内存的解释方式。**

# New关键字

C++中的关键字，用来将数据构建到堆中。通知程序，使用给定的数据类型去申请堆内存空间，并将地址信息返回。

**在VS编译器下指针的大小是4个字节**

# 基本数据类型的指针

**声明指针（并没有给予初始化，这是无效指针）**

int \*pI;

说明：构建了一个int类型的指针，但无法操作

**赋值构建指针**

int \*pI = new int(5665);

说明：构建了一个int类型的指针，并在堆中开辟了空间，空间中装填了数据5665

## 取地址符&

帮助我们获取到变量的地址，可以将地址存放到指针中，方便操作

Int p=10；

Int\* a = &p；

分析

借助取地址符&，取得了p的地址，并存到指针a中

## 星号 \*

当\*号跟随在数据类型后面，用途是构建指针

当\*号跟随在变量前，而不是在声明语句中，则是解地址用途。可以把指针指向的内容转换为操作内容，即实际数据

# 指针作为参数传递

当指针作为参数传递时，指针本身被进行了拷贝，但是传递的地址值不会出现变化，这将使得我们可以在函数内部操作**形参指向空间的值**从而影响**实参指向空间的值**。

# 自定义数据类型指针（类的指针）

### 操作成员方法（2种）

1. 使用解地址符\*，将构建的对象地址进行解地址操作，然后借助成员引用符点来操作对象成员函数或是属性。
2. 直接使用指针成员引用符->，来操作对象成员函数或是属性。

## This指针（对象自身内部指针）

This不是对象的属性，不会影响对象的大小，this的作用域在类内，外部无法访问操作，只能在成员函数内使用，代表当前操作对象的指针，由编译器构造的一个隐形的操作变量

# Delete关键字

只能放在指针前面，将指针指向的内存空间标记为可回收再利用。

切记对于不再使用的对象一定要释放掉空间

我们使用delete关键字进行指针的空间释放，并将指针指向NULL（0），防止误操作

**我们可以使用if关键字来检查指针是否有效，为了保证检查的顺利必须将指针释放后指向NULL**

# 空指针NULL

空指针宏，用来帮助引导指针指向无效值，NULL的值为0

空指针的操作是非法的。将造成程序致命错误。

# 野指针

**指针本身存储了一个内存地址值，但是地址指向的空间已经被标记为删除回收或是访问受限，我们称这种指针为野指针。**

**野指针的产生原因**

1. 构建指针时没有给予指针进行初始化。
2. 指针释放时，没有将指针指向NULL空。

**野指针的最致命问题在于无法通过if判断是否有效，这也是C++中最令人头痛的问题！所以我们在释放内存时必须将指针指向NULL。**

# 指针的意义

面向对象编程与面向过程的编程区别在于，面向对象编程是在运行阶段进行决策，这样可以更加灵活的操控逻辑与数据进行组合，在运行阶段使用new关键字进行数据内存创建，使用delete进行内存释放，使得程序更加的灵活