## C & C++

### C 和 C++的差别

* C是面向过程的，重点在于算法和数据结构，C++多了面向对象，支持继承、封装、多态
* C++是C的超集，多了string，布尔类型，STL
* C++有重载，模板实现静态多态，inline、extern函数
* C中其他可以调用main(), C++不可以

### struct 和 class 差别

* C++中两者的区别，成员默认访问权限和默认继承权限public和private的差别。模板中不能用struct

### 宏定义 和 inline 的区别

Inline和普通函数相比可以加快程序运行的速度，因为编译的时候内联函数直接镶嵌到目标代码当中，不需要中断调用。内联函数会做类型检查，宏不是函数，只是一个简单的替换。

### extern “C” 的作用

C++支持重载，编译后库中的名字于C不同，如：int foo(int x, int y) （\_foo \_foo\_int\_int）。

作用是告诉编译器按C语言的方式编译

## C++11

### 左值、右值

* **左值**：能取地址，有名字。一般变量都是
* **右值**：不能取地址，没有名字，一般通过引用的方式找到他的存在。表达式的值函数返回值。
  + 纯右值：临时对象，常量
  + 将亡值：返回有值引用的函数返回值，std::move()的返回值
* **右值引用**：不具名变量的引用，都属于引用类型，必须初始化

只要临时对象的应用存在，临时对象就会一直存在，

不能绑定到左值对象。 Is\_rvalue\_reference()<type\_traits>

* **移动构造**：移动语义，有值引用作为参数，将原来指向堆的指针置空，新的指针指向原来的堆

避免了构造时的再次申请内存



### move 与forward

std::move：将一个左值强制转化成右值引用。只做转换不做移动。确保是生命期即将结束的对象，可以调用移动构造函数

std::forward：完全依照模板的参数类型，将产生传递给模板函数中调用的另一个函数，左值对象仍然是左值对象，右值对象仍然是右值对象；

通过引用折叠实现，将复杂未知的表达式折叠为已知简单的表达式。



模板类型推导，不管TR的定义类型， TR是左值就左值，TR是右值就右值

### emplace\_back 与push\_back

### 智能指针

### lambda

### async

## 指针& 引用

### 指针和引用的区别

## static

* 静态成员函数属于类而不是类对象，所以没有this指针，就是读写静态成员的，不能访问普通成员
* 内存中只有一份数据，所有的对象共享，必须初始化，不能再类的定义中初始化
* 想当于nomember 函数，有利于成为callback，方便作为线程函数

## 虚函数virtual

### virtual 介绍和使用规则

* 虚函数的作用主要是实现了多态的机制，也就是通过基类指针调用实际派生类的成员函数
* 非类的成员函数不能定义为虚函数，
* 类的成员函数中静态成员函数和构造函数也不能定义为虚函数，但可以将析构函数定义为虚函数

### 虚析构

Delete一个指向派生类对象的基类指针，子类和基类的虚构函数都会调用。非虚析构的话，只调用基类的析构函数

### 虚函数表

 为了保证正确取到虚函数的偏移量，编译器必需要保证虚函数表的指针存在于对象实例中最前面的位置

typedef void(\*Fun)(void);

Base b;

Fun pFun = NULL;

cout << "虚函数表地址：" << (int\*)(&b) <<endl; //虚表的地址

cout << "虚函数表 — 第一个函数地址：" << (int\*)\*(int\*)(&b) <<endl;

pFun = (Fun)\*((int\*)\*(int\*)(&b)); // 将地址转化成函数地址

pFun();

(Fun)\*((int\*)\*(int\*)(&b)+0); // Base::f()

(Fun)\*((int\*)\*(int\*)(&b)+1); // Base::g()

(Fun)\*((int\*)\*(int\*)(&b)+2); // Base::h()

## 

## 异常处理exception

## 虚函数virtual

## 虚函数virtual

## 虚函数virtual