C++总结1：

C到C++

0： C++对C的扩展

- C++的输入/输出（cin/cout）
- C++字符串类型（string） —— string s1;
- C++布尔类型（bool） —— true/flase
- C++枚举类型（enum）
- C++命名空间（namespace）
  - 作用：解决命名冲突以及引入歧义
  - 注意： 在定义命名空间是，结束不要加分号
  - 命名空间成员使用方法总结
    - 方法1： 命名空间名：：成员
    - 方法2： using 命名空间名：：成员 —— *把指定的成员暴露给外部
    - 方法3： using namespace 命名空间名 —— *将其所有成员暴露给外部

1： 引用

- 1.1：引用的基本语法
  - 语法：数据类型 &别名 = 原名（int &a = b;）
  - 作用：给变量起别名
- 1.2：引用的注意事项
  - 1 引用必须初始化
  - 2 引用在初始化之后不可以改变
- 1.3：引用做函数参数
  - 作用： 函数传参时，可以利用引用的技术让形参修饰实参
  - 优点： 可以简化指针修改实参
  - 注意： 引用作为函数的形参，则该参数采用的是址传递，作为函数形参时，可以不初始化
- 1.4：引用做函数返回值
  - 作用： 引用可以作为函数的返回值存在
  - 注意： 不要返回局部变量引用
  - 用法： 函数调用作为左值
- 1.5：引用的本质
  - 本质： 引用的本质在C++内部实现是一个指针常量
  - 在C++中，引入引用是为了避免指针在使用过程中可能出的问题（空指针，不合法内存），提高程序的安全性
  - 引用是一种关系型声明的类型
- 1.6 深入引用
  - 1 引用本质
    - 引用它只能代表某个变量的别名，只能跟一个变量进行绑定
    - 指针常量
      - const type* p ---该指针变量指向的对象可以改变，而该内存地址上的内容是不可修改的，跟type const* p等价
      - type* const p ---该指针变量指向的对象不可改变，而内存地址上的内容是可以修改的

    引用的本质 ---type* const p
  - 2 const引用
    - 普通引用： type&
    - const引用： const type&
      - 1 const 引用可以使用常量给它初始化，而普通 引用不行
      - 2 const 引用可以使用普通变量或者普通引用来给它初始化
      - 3 const 引用在严格上讲，要是用const变量来给它初始化
      - 4 普通引用不能通过const变量，const引用 常量来给它初始化，只能接受普通引用或者普通变量

      const引用总结

2： 内联函数（inline）

- 2.1： 内联函数的基本语法
  - 语法：inline 数据类型 名称(参数，...)
  - 注意：inline关键字一个与函数体放在一起才能起作用，函数声明时，关键字可加可不加
- 2.2： 作用 —— 用来建议编译器对一些特殊函数进行内联扩展（在线扩展）；指建议编译器将制定的函;数体插入并取代每一处调用该函数的地方（上下文），可以更节省每次调用函数带来的额外时间开支
- 2.2： 注意事项 —— 在选择使用内联函数时，必须在程序占用空间和程序执行效率之间进行权衡，过多的比较复杂的函数进行内联扩展会带来更大的存储资源开支。
- 2.3： 总结 —— 宏和内联函数都是为了普通函数调用时增加的栈开销，从而提高程序的运行效率

3： 默认参数

- 3.1： 默认参数的定义 —— 在函数定义实现时，给形参的一个初始化值
- 3.2： 默认参数的语法
  - 语法： 返回值 函数名(数据类型 形参变量名 =默认值){ }
  - 应用场景引入： —— 通常情况下，函数在被调用时，形参从实参那取得值，该过程中发生了值拷贝操作，在对于多次调用一个函数同一个实参时，利用默认参数则可以不用从实参那拷贝值了
- 3.3： 默认函数的注意事项 —— 如果函数的声明和函数的定义分离实现的，那么默认参数只需要在函数的声明时进行赋值，定义时不再需要

4： 占位参数

- 语法 —— 返回值类型 函数名(数据类型){ }
- * —— 占位参数使用场景较少，只在运算符重载重可以应用

5： 函数重载（overload）

- 5.1： 函数重载的定义 —— 在函数名一致的基础上有其他差别的多组函数
- 5.2： 判断为函数重载的依据
  - 1 函数名相同
  - 2 函数的参数类型不||参数个数不同||参数类型的顺序不同，三选其一即可
  - 3 函数的返回值不能作为本函数重载判断的依据
- 5.3： C++如何实现函数重载 —— C++编译器编译源文件时通过底层倾轧技术来实现 —— 底层倾轧---将原有函数名+参数类型 --->在底层时，形成一个新的函数名，从底层，各个函数名还是不一样的
- 5.4： extern "C"的主要作用 —— 为了能够正确实现C++代码调用其他C语言代码。加上extern "C"后，会只是编译器这部分代码按C语言的进行编译，而不是C++