gcc是编译器名称，g++是实际运行的命令。

signed, unsigned, short, long都可以修饰int，有时int可忽略，只要不互斥。wchar\_t为一个宽字符，视机器情况2-4字节。unsigned和signed short长度不定。

typedefine A B; A以B为别名。

枚举：

enum enumName {

valueName1, valueName2 = 5,…valueNameN //可以赋值，自动加一。

};

enumName varName = valueName2;

函数：有声明就能调用，定义在任何地方进行。

变量：声明规定类型和名字，定义申请存储空间，初始化给初值，赋值为重写； 全局变量定义只能一次，声明可以多次。内置类型默认初始值：函数外初始化为0，函数内不初始化。

extern用于全局变量和函数，声明有这东西，具体用别人的。特殊情况：extern且初始化全局变量，则又是声明又是定义。

数字的后缀：u为unsigned，l为long，可以一起出现，不能重复。

1f不合法，1.0f才可以。1e5=100000。

字符常量前加L：宽字符常量，要用wchar\_t存储。

signed和unsigned互相赋值：拷贝各位。

auto：看网上，现在不深究。

register：建议放在寄存器。

static用于局部变量：在函数各次运行之间保持值；用于全局变量：作用域限制在文件中。

^：位异或运算符。~：位取反运算符。<<=，|==都有。sizeof运算符。

else if ()

引用做参数：传入时和一般参数一样，不像指针要明确传指针。接收时收到的是引用，所以更改会反映到参数处。

默认参数只能放右边。

数组可用大括号初始化，多维数组也一样（嵌套括号可选）。此时可以忽略数组大小。

一位数组的数组名是一个常量指针，指向&name[0]，可赋值给另一个指针。高维的不适用，不能直接赋值，要用&取地址给一维指针然后迭代。不能\*(name+1)这么访问，如int d[1][2][3] = {1,2,3,4,5,6}，\*\*\*d=1，\*(\*(\*d + 1)+1)=5 但不能int\*\*\* ss = d，作为常量d也不能被赋值。

数组参数：int\*，int param[10]，intparam[]。

C++ 不支持在函数外返回局部变量的地址，除非定义局部变量为 static 变量。

NULL

指针参数：可以修改参数指向的东西但不能修改参数本身。如：

void swap (int \* a, int \* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

int\* te = a;

a = b;

b = te;

}//前三句有效，交换成功。后三句无效，交换失败。

当然如果你用int\*\*a做参数则能修改到\*\*a，能修改到\*a但是修改不了a，以此类推。

引用vs指针：不能空，不能改，创建必须初始化。引用作为参数和返回值和指针一样，不过只能控制指向的东西而不能控制引用本身。作为参数好处是可以影响实参，作为返回值好处是可以修改被返回的值。

结构和enum都可以在定义之后紧接着放几个variable。

struct基本干啥都要写struct structName，麻烦。typedef常常和struct一起出现就是为了把它当成一个类型，以后不用老打struct关键字。此时可以在定义时省略structName，直接给出typedef后的name。

类定义中定义的成员函数自动内联，不管有没有inline。inline也可以用于类外方法。

默认private。protected：派生类可访问。

初始化列表：method():左值(右值), 左值(右值){}

构造函数：类名()。析构函数：~类名()。

拷贝构造函数：classname (const classname &obj) {

// 构造函数的主体

}//如果没有会自动定义一个

局部变量类实例定义：className variable(args)。如果没有args，则连括号也不要，否则编译出错！

new className(args)

等号会调用拷贝构造函数。直接使用普通参数的话，形参实参绑定时也会调用。形参销毁时会调用析构函数。

类声明中可以使用friend指出友元方法或友元类，他们可以访问一切protected、private。友元类的所有方法都是友元方法。

如果不存在其他的初始化语句，在创建第一个对象时，所有的静态数据都会被初始化为零。我们不能把静态成员放置在类的定义中，但是可以在类的外部通过使用范围解析运算符 :: 来重新声明静态变量从而对它进行初始化。也就是说，初始化要用int Box::objectCount = 0;如果没有则默认初始化为0。

静态成员函数的调用：类::成员函数名()

继承：class subClassName: access-modifier superClassName[, access-modifier superClassName2…]默认private

派生类不继承：构造，析构，拷贝，运算符，友元

我们几乎不使用 protected 或 private 继承，通常使用 public 继承。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 父类的public成员 | Protected | Private |
| Public继承 | 还是public | 还是protected | 不能用 |
| Protected | 变成protected | 还是protected | 不能用 |
| Private | 变成private | 变成private | 不能用 |

多继承：不深究的话就那样

运算符重载：Box operator+(const Box&, const Box&);实现同一件事，作为成员函数则需要一个参数，否则两个。常用：friend ostream &operator<<( ostream &output, const Distance &D )

子类的构造函数要调用父类的构造函数Rectangle( int a=0, int b=0):Shape(a, b) { }

动态绑定：virtual标记的虚函数被父类指针调用时会按照指针指向的内容的类型判断调用版本而非按照指针类型判断。纯虚函数：没有函数体，=0。有纯虚函数的都是虚基类，无法实例化，别的和普通类一样。不过子类实例化之后可以用父类的指针把它当父类实例用。

异常：throw一个实例、字符串……随你。try catch没有finally。catch可以用…表示所有异常，可以catch(const char \* msg)

new可能不成功，看返回是否null可以判断。

malloc给内存，new还创建了对象。

多维数组分配与释放内存：

double \*\*pvalue = new double\* [ROW];

for(int i = 0; i < COL; i++) {

pvalue[i] = new double[COL];

}

for(int i = 0; i < COL; i++) {

delete[] pvalue[i];

}

delete [] pvalue;

namespace：不深究的话不难。namespace namespaceName {}。有名字的namespace不对本文件开放，无名namespace对本文件开放。using的生效范围是从这句话开始到using所在块结束，包括其内部含有的块。注意无名namespace不算是块，namespace里面的东西和namespace上下的东西是平级的。有名namespace是块。命名空间可以分散在多个文件中。可嵌套。using上层对下层无效，只是打开了上层所以可以用“下层::内容”语法了。

模板：在头上加上template <class T, class T2…>

类模板的实现比较麻烦：

template <class T>

void Stack<T>::push (T const& elem) {}

如果const位于\*的左侧，即指针指向内容为常量；

如果const位于\*的右侧， 即指针本身是常量。

const成员变量只能在初始化列表中赋值。普通常变量一般在声明时候就赋值。

void func() const不改变对象的成员变量，不调用非const成员函数。

const对象/指针/引用的任何成员从构造结束后就都不能修改，只能调用const成员函数。

const\_cast <type> (name)用于转换本身为常量的变量而非转换指向常量的变量。

任何不会修改数据成员的函数都应该声明为const 类型。

const可以用来重载函数，非const版本只有非const对象可以调用，const版本谁都可以调用。一个const版本一般就够了，两个都有时非const对象调用非const函数，const对象调用const函数。

const修饰值传递函数或char\* const var参数都没有意义。

static const int成员变量需要在类外const int class::var = xxx;赋值。

const函数内部的this是const class\*类型的，保证了不能调用非const函数。

赋值表格：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 右值普通 | \*const | const\* | Const\*const |
| 左值普通 | Y | Y | N(体现了保护) | N(体现了保护) |
| \*const | Y(仅在初始化时) | Y(仅在初始化时) | N(体现了保护) | N(体现了保护) |
| const\* | Y | Y | Y | Y |
| Const\*const | Y(仅在初始化时) | Y(仅在初始化时) | Y(仅在初始化时) | Y(仅在初始化时) |

注意，指向常量这个保护并不可靠！因为总可以通过简单的类型声明破坏掉（对上面所说的this指针也一样），如：

const int\* i= new int; int\* j = (int\*) i;

const int& I = 4; int& j = (int&) I;

对常变量、引用、数组的const和类型都可以交换。

类型& const不合法。

对于常量来说，系统没有划定专门的区域来保护其中的数据不能被更改。也就是说，使用常量的方式对数据进行保护是通过编译器作语法限制来实现的。

mutable数据成员：可以再const函数中被修改。

const常量优于宏常量：类型，字符替换的隐患。

函数的返回值为某个const对象多用于操作符的重载，因为运算结果不能作为左值。

const全局变量默认局限于文件，如果要关系到全局则用extern修饰，大家都要用extern声明，其中一个同时赋值定义。

#ifndef NULL

#define NULL 0

#endif

也有#ifdef，#if

预定义宏：\_\_LINE\_\_, \_\_FILE\_\_, \_\_DATE\_\_, \_\_TIME\_\_

可用链接：<http://www.runoob.com/cplusplus/cpp-useful-resources.html>

static在函数中：不重复初始化。对全局变量：局限于本文件。

问题： const字符串？

自动类型转换？语法？

没有默认构造函数的话没法用new一次构造一列？

临时对象？只有常量引用能抓到他？