fenghuan

C++中头文件（.h）和源文件（.cpp）都应该写些什么

头文件(.h)：

写类的声明（包括类里面的成员和方法的声明）、函数原型、**#define**常数等，但一般来说不写出具体的实现。

在写头文件时需要注意，在开头和结尾处必须按照如下样式加上预编译语句（如下）：

```
#ifndef CIRCLE_H
#define CIRCLE_H

//你的代码写在这里

#endif
```

这样做是为了防止重复编译，不这样做就有可能出错。

昵称：fenghuan
园龄：3年
粉丝：24
关注：0
+加关注

< 2018年9月 >

日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

至于CIRCLE_H这个名字实际上是无所谓的，你叫什么都行，只要符合规范都行。原则上来说，非常建议把它写成这种形式，因为比较容易和头文件的名字对应。

源文件（.cpp）：

源文件主要写实现头文件中已经声明的那些函数的具体代码。需要注意的是，开头必须#include一下实现的头文件，以及要用到的头文件。那么当你需要用到自己写的头文件中的类时，只需要#include进来就行了。

下面举个最简单的例子来描述一下，咱就求个圆面积。

第1步，建立一个空工程（以在VS2003环境下为例）。

第2步，在头文件的文件夹里新建一个名为Circle.h的头文件，它的内容如下：

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
private:
    double r;//半径
public:
    Circle();//构造函数
    Circle(double R);//构造函数
    double Area();//求面积函数
};

#endif
```

注意到开头结尾的预编译语句。在头文件里，并不写出函数的具体实现。

第3步，要给出Circle类的具体实现，因此，在源文件夹里新建一个Circle.cpp的文件，它的内容如下：

```
#include "Circle.h"

Circle::Circle()
{
    this->r=5.0;
}

Circle::Circle(double R)
{
    this->r=R;
}

double Circle:: Area()
{
    return 3.14*r*r;
}
```

需要注意的是：开头处包含了Circle.h，事实上，只要此cpp文件用到的文件，都要包含进来！这个文件的名字其实不一定要叫Circle.cpp，但非常建议cpp文件与头文件相对应。

最后，我们建一个main.cpp来测试我们写的Circle类，它的内容如下：

```
#include <iostream>
#include "Circle.h"
using namespace std;

int main()
{
    Circle c(3);
    cout<<"Area="<<c.Area()<<endl;
    return 1;
}
```

注意到开头时有#include "Circle.h"的声明，证明我们使用到了我们刚才写的Circle类。

2015年9月 (30)
2015年8月 (3)

最新评论

1. Re:C++中头文件（.h）和源文件（.cpp）都应该写些什么
@haojie13这就是对Circle 的对象c进行初始化，参数为3...
--jason*liu

2. Re:C/C++存储区划分
自由存储区域堆有什么实质的区别呢
--wenmingxing

3. Re:C++中虚函数功能的实现机制
你好，写的很好，借鉴不错
--maxiaozhu

4. Re:C++中头文件（.h）和源文件（.cpp）都应该写些什么
很感谢楼主的分享，之前学了C语言和Java，一直不知道类书写位置，一直用着Java那一套。看了觉得受益良多。赞一个。
--皓少

5. Re:C++中头文件（.h）和源文件（.cpp）都应该写些什么
main函数中的Circle c（3）这里怎么理解？
--haojie13

阅读排行榜

1. C++中头文件（.h）和源文件（.cpp）都应该写些什么(50792)
2. C++ 字符串指针与字符串数组(11917)
3. C++类包含问题(重复包含和相互包含)(5279)

至此，我们工程的结构为：

运行一下，输出结果为：

说明我们写的Circle类确实可以用了。

1..h叫做头文件，它是不能被编译的。**“#include”**叫做编译预处理指令，可以简单理解成，在**1.cpp**中的**#include"1.h"**指令把**1.h**中的代码在编译前添加到了**1.cpp**的头部。每个**.cpp**文件会被编译，生成一个**.obj**文件，然后所有的**.obj**文件链接起来你的可执行程序就算生成了。

发现了没有，你要在**.h**文件中严格区分声明语句和定义语句。好的习惯是，头文件中应只处理常量、变量、函数以及类等等等等的声明，变量的定义和函数的实现等等等等都应该在源文件**.cpp**中进行。

至于**.h**和**.cpp**具有同样的主文件名的情况呢，对编译器来讲是没有什么意义的，编译器不会去匹配二者的主文件名，相反它很傻，只认**#include**等语句。但是这样写是一种约定俗成的编程风格，一个类的名字作为其头文件和源文件的主文件名比如**Class1.h**和**Class1.cpp**，这个类的声明在**Class1.h**中，实现在**Class1.cpp**中，我们人类看起来比较整齐，读起来方便，也很有利于模块化和源代码的重用。

为什么这个风格会约定俗成？有一句著名的话，叫“程序是为程序员写的”。

2.h文件和**cpp**文件也就是说，在**h**文件中声明**Declare**，而在**cpp**文件中定义**Define**。“声明”向计算机介绍名字，它说，“这个名字是什么意思”。而“定义”为这个名字分配存储空间。无论涉及到变量时还是函数时含义都一样。无论在哪种情况下，编译器都在“定义”处分配存储空间。对于变量，编译器确定这个变量占多少存储单元，并在内存中产生存放它们的空间。对于函数，编译器产生代码，并为之分配存储空间。函数的存储空间中有一个由使用不带参数表或带地址操作符的函数名产生的指针。定义也可以是声明。如果该编译器还没有看到过名字**A**，程序员定义**int A**，则编译器马上为这个名字分配存储地址。声明常常使用于**extern**关键字。如果我们只是声明变量而不是定义它，则要求使用**extern**。对于函数声明， **extern**是可选的，不带函数体的函数名连同参数表或返回值，自动地作为一个声明。

另篇：

4. 在C++中如何使用C(4501)
5. 深入理解C/C++数组和指针(4036)

评论排行榜

1. C++中头文件（.h）和源文件（.cpp）都应该写些什么(4)
2. C/C++存储区划分(1)
3. C++中虚函数功能的实现机制(1)

推荐排行榜

1. C++中头文件（.h）和源文件（.cpp）都应该写些什么(13)
2. C++ 字符串指针与字符串数组(2)
3. C++ Struct(1)
4. C++指针(1)
5. 一道试题引发的血案 int *ptr2=(int *)((int)a+1);(1)

在C++编程过程中，随着项目的越来越大，代码也会越来越多，并且难以管理和分析。于是，在C++中就要分出了头(.h)文件和实现(.cpp)文件，并且也有了Package的概念。

对于以C起步，C#作为"母语"的我刚开始跟着导师学习C++对这方面还是感到很模糊。虽然我可以以C的知识面对C++的语法规范，用C#的思想领悟C++中类的使用。但是C#中定义和实现是都在一个文件中(其实都是在类里面)，而使用C的时候也只是编程的刚刚起步，所写的程序也只要一个文件就够了。因此对于C++的Package理解以及.h文件和.cpp文件的总是心存纠结。

幸好导师有详细的PPT让我了解，一次对于Package的认识就明白多了。简单讲，一个Package就是由同名的.h和.cpp文件组成。当然可以少其中任意一个文件：只有.h文件的Package可以是接口或模板(template)的定义；只有.cpp文件的Package可以是一个程序的入口。

当然更具体详细的讲解，欢迎下载导师的教学PPT-Package来了解更多。

不过我在这里想讲的还是关于.h文件和.cpp文件

知道Package只是相对比较宏观的理解：我们在项目中以Package为编辑对象来扩展和修正我们的程序。编写代码时具体到应该把什么放到.h文件，又该什么放在.cpp文件中，我又迷惑了。

虽然Google给了我很多的链接，但是大部分的解释都太笼统了：申明写在.h文件，定义实现写在.cpp文件。这个解释没有差错，但是真正下手起来，又会发现不知道该把代码往哪里打。

于是我又把这个问题抛给了导师，他很耐心地给我详详细细地表述了如何在C++中进行代码分离。很可惜，第一次我听下了，但是没有听太懂，而且本来对C++就了解不深，所以也没有深刻的印象。

经过几个项目的试炼和体验之后，我又拿出这个问题问导师，他又一次耐心地给我讲解了一遍（我发誓他绝对不是忘记了我曾经问过同样的问题），这次我把它记录了下来。

为了不再忘记，我将它们总结在这里。

概览

	非模板类型(none-template)	模板类型(template)

	非模板类型(none-template)	模板类型(template)
头文件(.h)	• 全局变量申明（带extern限定符） • 全局函数的申明 • 带inline限定符的全局函数的定义	• 带inline限定符的全局模板函数的申明和定义
	• 类的定义 • 类函数成员和数据成员的申明（在类内部） • 类定义内的函数定义（相当于inline） • 带static const限定符的数据成员在**类内部**的初始化 • 带inline限定符的类定义外的函数定义	• 模板类的定义 • 模板类成员的申明和定义（定义可以放在类内或者类外，类外不需要写inline）
实现文件(.cpp)	• 全局变量的定义（及初始化） • 全局函数的定义	(无)
	• 类函数成员的定义 • 类带static限定符的数据成员的初始化	

*申明：declaration
*定义：definition

头文件

头文件的所有内容，都必须包含在

```
#ifndef {Filename}
#define {Filename}

//{Content of head file}

#endif
```

这样才能保证头文件被多个其他文件引用(include)时，内部的数据不会被多次定义而造成错误

inline限定符

在头文件中，可以对函数用inline限定符来告知编译器，这段函数非常的简单，可以直接嵌入到调用定义之处。

当然inline的函数并不一定会被编译器作为inline来实现，如果函数过于复杂，编译器也会拒绝inline。

因此简单说来，代码最好短到只有3-5行的才作为inline。有循环，分支，递归的函数都不要用做inline。

对于在类定义内定义实现的函数，编译器自动当做有inline请求（也是不一定inline的）。因此在下边，我把带有inline限定符的函数成员和写在类定义体内的函数成员统称为"要inline的函数成员"

非模板类型

全局类型

就像前面笼统的话讲的：申明写在.h文件。

对于函数来讲，没有实现体的函数，就相当于是申明；而对于数据类型（包括基本类型和自定义类型）来说，其申明就需要用extern来修饰。

然后在.cpp文件里定义、实现或初始化这些全局函数和全局变量。

不过导师一直反复强调：不许使用全局函数和全局变量。用了之后造成的后果，目前就是交上去的作业项目会扣分。当然不能用自有不能用的理由以及解决方案，不过不在目前的讨论范围内。

自定义类型

对于自定义类型，包括类（class）和结构体（struct），它们的定义都是放在.h文件中。其成员的申明和定义就比较复杂了，不过看上边的表格，还是比较清晰的。

函数成员

函数成员无论是否带有static限定符，其申明都放在.h文件的类定义内部。

对于要inline的函数成员其定义放在.h文件；其他函数的实现都放在.cpp文件中。

数据成员

数据成员的申明与定义都是放在.h文件的类定义内部。对于数据类型，关键问题是其初始化要放在什么地方进行。

对于只含有static限定符的数据成员，它的初始化要放在.cpp文件中。因为它是所有类对象共有的，因此必须对它做合适的初始化。

对于只含有const限定符的数据成员，它的初始化只能在构造函数的初始化列表中完成。因为它是一经初始化就不能重新赋值，因此它也必须进行合适的初始化。

对于既含有static限定符，又含有const限定符的数据成员，它的初始化和定义同时进行。它也是必须进行合适的初始化

对于既没有static限定符，又没有const限定符的数据成员，它的值只针对本对象可以随意修改，因此我们并不在意它的初始化什么时候进行。

模板类型

C++中，模板是一把开发利器，它与C#，Java的泛型很相似，却又不尽相同。以前，我一直只觉得像泛型，模板这种东西我可能一辈子也不可能需要使用到。但是在导师的强制逼迫使用下，我才真正体会到模板的强大，也真正知道要如何去使用模板，更进一步是如何去设计模板。不过这不是三言两语可以讲完的，就不多说了。

对于模板，最重要的一点，就是在定义它的时候，编译器并不会对它进行编译，因为它没有一个实体可用。

只有模板被具体化（specialization）之后（用在特定的类型上），编译器才会根据具体的类型对模板进行编译。

所以才定义模板的时候，会发现编译器基本不会报错（我当时还很开心的：我写代码尽然会没有错误，一气呵成），也做不出智能提示。但是当它被具体用在一个类上之后，错误就会大片大片的出现，却往往无法准确定位。

因此设计模板就有设计模板的一套思路和方式，但是这跟本文的主题也有偏。

因为模板的这种特殊性，它并没有自己的准确定义，因此我们不能把它放在.cpp文件中，而要把他们全部放在.h文件中进行书写。这也是为了在模板具体化的时候，能够让编译器可以找到模板的所有定义在哪里，以便真正的定义方法。

至于模板类函数成员的定义放在哪里，导师的意见是放在类定义之外，因为这样当你看类的时候，一目了然地知道有那些方法和数据；我在用Visual Studio的时候查看到其标准库的实现，都是放在类内部的。

可能是我习惯了C#的风格，我比较喜欢把它们都写在类内部，也因为在开发过程中，所使用的编辑器都有一个强大的功能：代码折叠。

当然还有其他原因就是写在类外部，对于每一个函数成员的实现都需要把模板类型作为限定符写一遍，把类名限定符也要写一遍。

分类: C++

fenghuan
关注 - 0
粉丝 - 24
+加关注

« 上一篇：C++模板

» 下一篇：C++类

posted @ 2015-09-09 13:49 fenghuan 阅读(50792) 评论(4) 编辑 收藏

发表评论

#1楼 2016-09-05 21:29 | 若渴9 回复 引用

写得非常好，通俗易懂，支持一下！

支持(1) 反对(0)

#2楼 2017-04-18 15:03 | haojie13 回复 引用

main函数中的Circle c（3）这里怎么理解？

支持(0) 反对(0)

#3楼 2017-05-18 21:03 | 皓少 回复 引用

很感谢楼主的分享，之前学了C语言和Java，一直不知道类书写位置，一直用着Java那一套。看了觉得受益良多。赞一个。

支持(0) 反对(0)

#4楼 2018-07-24 15:48 | jason*liu 回复 引用

@ haojie13
这就是对Circle 的对象c进行初始化，参数为3

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

发表评论

昵称： Justry2015

评论内容：