

原创

YoungYangD

2018-11-25 22:04:26


19186

收藏 247

版权

分类专栏：

C/C++

C/C++ 专栏收录该内容

11 订阅34 篇文章

订阅专栏

概述

我们在写代码时，总会遇到头文件多次包含的情况，刚开始时我们使用**宏定义**进行控制，之后发现有#pragma once这样简单的东西，当时是很兴奋，以为#pragma就这一种用法。唉~，现在想想当时还是年轻啊，不过还是年轻好啊。

1、什么是预处理

预处理是将源文件的文本作为翻译的第一阶段操作的文本处理步骤。预处理不会分析源文本，但会为了查找宏调用而将源文本细分为标记。主要包括了下面三个方面：

1. 预处理指令
2. 预处理运算符
3. 预定义宏，这个有很多了，比如__FILE__、__LINE__和__DATA__等。

其中预处理指令包括：

#define	#error	#import	#undef
#elif	#if	#include	#using
#else	#ifdef	#line	#endif
#ifndef	#pragma		

预处理运算符包括：

运算符	操作
字符串化运算符（#）	导致对应的实参括在双引号内
Charizing 运算符 (#@)	导致对应的参数括在单引号内且其被认为是字符（Microsoft 特定）
标记粘贴运算符 (##)	允许用作实参的标记串联以形成其他标记
定义的运算符	简化在特定宏指令的复合表达式的书写。

这里有很多预处理指令符，今天我只整理pragma了。

2、什么是pragma

#pragma指令的作用是：**用于指定计算机或操作系统特定的编译器功能**。C 和 C++ 的每个实现均支持某些对其主机或操作系统唯一的功能。例如，某些程序必须对将数据放入的内存区域进行准确的控制或控制某些函数接收参数的方式。在保留与 C 和 C++ 语言的总体兼容性的同时，#pragma 指令使每个编译器均能够提供特定于计算机和操作系统的功能。

根据定义，**#pragma指令是计算机或操作系统特定的，并且通常对于每个编译器而言都有所不同**。#pragma指令可用于条件语句以提供新的预处理器功能，或为编译器提供实现所定义的信息。

语法是：

```
#pragma token-string
__pragma( token-string )    //__pragma 关键字是特定于 Microsoft 编译器的
```

token-string 是一系列字符，这些字符提供了特定的编译器指令和参数（如果有）。数字符号"#" 必须是位于包含#pragma指令行上的第一个非空白字符；空白字符可以分隔数字符号和词“pragma”。在 #pragma 之后，编译转换器可分析为预处理标记的所有文本。#pragma 的参数受宏展开的约束，**如果编译器发现它无法识别的杂注，则它会发出警告并继续编译。**



alloc_text	auto_inline	bss_seg
check_stack	code_seg	comment
component	conform ¹	const_seg
data_seg	deprecated	detect_mismatch
fenv_access	float_control	fp_contract
function	hdrstop	include_alias
init_seg ¹	inline_depth	inline_recursion
intrinsic	loop ¹	make_public
managed	message	
omp	once	
optimize	pack	pointers_to_members ¹
pop_macro	push_macro	region、endregion
runtime_checks	section	setlocale
strict_gs_check	unmanaged	vtordisp ¹
warning	https://blog.csdn.net/waixin_39640298	

注：标注1的仅受C++编译器支持。

好，下面就对一些常用的指令进行说明，有一些不常用，或者我们上层不常用的就不介绍了，因为我也不懂（没有时间应用，看也看不懂）。

2.1 #pragma once

大家应该都知道：**指定该文件在编译源代码文件时仅由编译器包含（打开）一次。**

使用 #pragma once 可减少生成次数，和使用预处理宏定义来避免多次包含文件的内容的效果是一样的，但是需要键入的代码少，可减少错误率，例如：

```
使用#progma once
#pragma once
// Code placed here is included only once per translation unit
```

```
使用宏定义方式
#ifndef HEADER_H_
#define HEADER_H_
// Code placed here is included only once per translation unit
#endif // HEADER_H_
```

2.2 #pragma message (messageString)

不中断编译的情况下，发送一个字符串文字量到标准输出。message编译指示的典型运用是在编译时显示信息，例如：

```
#if _M_IX86 >= 500           //查看定义的宏有没有大于500，或者有时用作检查有没有定义宏
#pragma message("_M_IX86 >= 500")
#endif
```

messagestring 参数可以是扩展到字符串的宏，您可以通过任意组合将此类宏与字符串串联起来，例如：

```
#pragma message( "Compiling " __FILE__ )           //显示被编译的文件
#pragma message( "Last modified on " __TIMESTAMP__ ) //文件最后一次修改的日期和时间
```

如果在 message 杂注中使用预定义的宏，则该宏应返回字符串，否则必须将该宏的输出转换为字符串，例如：

```
#define STRING2(x) #x
#define STRING(x) STRING2(x)

#pragma message ( __FILE__ "[" STRING(__LINE__) "]: test" ) //注意把行号转成了字符串
```



启用编译器警告消息的行为和选择性修改，语法为：

```
#pragma warning( warning-specifier : warning-number-list [,warning-specifier : warning-number-list...] )

#pragma warning( push[ , n ] )
#pragma warning( pop )
```

warning-specifier能够是下列值之一：

警告说明符	含义
1, 2, 3, 4	将给定级别应用于指定的警告。这也会启用默认情况下处于关闭状态的指定警告。
default	将警告行为重置为其默认值。这也会启用默认情况下处于关闭状态的指定警告。警告将在其默认存档级别生成。 有关详细信息，请参阅默认情况下处于关闭状态的编译器警告。
disable	不发出指定的警告消息。
error	将指定警告报告为错误。
once	只显示指定消息一次。
suppress	将杂注的当前状态推送到堆栈上，禁用下一行的指定警告，然后弹出警告堆栈，从而重置杂注状态。

估计看这个表会头晕，我来举几个例子：

```
#pragma warning( disable : 4507 34; once : 4385; error : 164 )    //这1行跟下面3行效果一样

#pragma warning( disable : 4507 34 )    //不发出4507和34警告，即有4507和34警告时不显示
#pragma warning( once : 4385 )          //4385警告信息只报告一次
#pragma warning( error : 164 )          //把164警告信息作为一个错误
```

注意：对于范围 4700-4999 内的警告编号（与代码生成相关联），在编译器遇到函数的左大括号时生效的警告状态将对函数的其余部分生效。在函数中使用 warning 杂注更改编号大于 4699 的警告的状态只会在函数末尾之后生效。以下示例演示如何正确放置 warning 杂注来禁用代码生成警告消息然后还原该消息

```
#pragma warning(disable:4700)    //不发生4700警告
void Test()                    //这里之前的状态管用，若把上面一行移动到函数里面，则照常发生警告
{
    int x;
    int y = x;                //没有C4700的警告
    #pragma warning(default:4700)    //4700的警告恢复到默认值
}

int main()
{
    int x;
    int y = x;    //发生C4700的警告
}
```


下面来看warning推送和弹出的用法，语法为：


```
#pragma warning( push [ , ``n ] )
#pragma warning( pop )
```

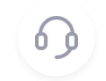
其中 n 表示警告等级（1 到 4）。

warning(push)指令存储每个警告的当前警告状态。
warning(push, n)指令存储每个警告的当前状态并将全局警告级别设置为 n。
warning(pop)指令 弹出推送到堆栈上的最后一个警告状态。
在 push 和 pop 之间对警告状态所做的任何更改都将被撤消。

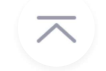
```
#pragma warning( push )
#pragma warning( disable : 4705 )
#pragma warning( disable : 4706 )
#pragma warning( disable : 4707 )
// Some code                                //代码的书写，这里不会发出4705、4706、4707的警告
#pragma warning( pop )                      //会将每个警告（包括4705、4706、4707）的状态还原为代码开始的状态
```

惊喜盲盒





举报



```
#pragma warning( push, 3 )
// Declarations/ definitions           //要书写的代码
#pragma warning( pop )
```

2.4、#pragma comment (comment-type [, “commentstring”])

该指令将一个注释记录放入一个对象文件或可执行文件中。

comment-type 是一个预定义的标识符（如下所述，一共5个），它指定了注释记录的类型。可选 commentstring 是一个字符串，它提供了某些注释类型的附加信息。由于 commentstring 是一个字符串，因此它遵循有关转义字符、嵌入的引号 (") 和串联的字符串的所有规则。

compiler

将编译器的名称和版本号置于对象文件中。此注释记录将被链接器忽略。如果为此记录类型提供 commentstring 参数，则编译器会生成警告。

exestr

将 commentstring 置于对象文件中。在链接时，会将该字符串置于可执行文件内。加载可执行文件时，不会将字符串加载到内存中；但是，可以使用在文件中查找可打印字符串的程序来找到它。此注释记录类型的一个用途是将版本号或类似信息嵌入可执行文件中。

lib（这个最常用了）

将库搜索记录置于对象文件中。此注释类型必须带有包含您希望链接器搜索的库的名称（和可能的路径）的 commentstring 参数。库名称遵循对象文件中的默认库搜索记录；链接器会搜索此库，这就像在命令行上对其命名一样，前提是未使用 /nodefaultlib 指定库。可以将多个库搜索记录置于同一个源文件中；各个记录将以其在源文件中显示的顺序出现在对象文件中。

如果默认库和添加的库的顺序很重要，则使用 /ZI 开关进行编译会阻止将默认库名称置于对象模块中。然后，可使用另一个注释指令在添加的库的后面插入默认库的名称。与这些指令一起列出的库将以其在源代码中的发现顺序出现在对象模块中。

```
#pragma comment(lib, "comctl32.lib")
#pragma comment( lib, "mysql.lib" )
```

linker

将链接器选项置于对象文件中。可以使用注释类型来指定链接器选项，而不是将其传递到命令行或在开发环境中指定它。例如，可以指定 /include 选项来强制包含符号：

```
#pragma comment(linker, "/include:__mySymbol")
```

user

将一般注释置于对象文件中。commentstring 参数包含注释文本。此注释记录将被链接器忽略。

2.5、#pragma region ... /endregion ...

#pragma region是Visual C++中特有的预处理指令。它可以让你折叠特定的代码块，从而使界面更加清洁，便于编辑其他代码。折叠后的代码块不会影响编译。你也可以随时展开代码块以进行编辑等操作

```
int main()
{
    ...
#pragma region demo_region           //这里前面会有折叠符，折叠时提示 demo_region所写内容
    int a = 10;
    int b = 20;
    int c = ADD( a + b );
#pragma endregion demo_region        //需要跟#pragma region配套使用
    ....
}
```

折叠代码块的方法：如同Visual C++中折叠函数、类、命名空间，当代码被包含在如上所述的指令之间后，#pragma region这一行的左边会出现一个“-”号，单击以折叠内容，同时“-”号会变成“+”号，再次单击可以展开代码块

2.6、#pragma alias (...)

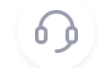
指定 short_filename 将用作 long_filename 的别名，语法为：

```
#pragma include_alias( "long_filename ", "short_filename" )
#pragma include_alias( <long_filename>, <short_filename> )
```

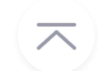
某些文件系统允许长度超过 8.3 FAT 文件系统限制的头文件名。因为较长的头文件名的前八个字符可能不是唯一的，因此编译器无法简单地将较长的名称截断为 8.3。每当编译器遇到 long_filename 字符串时，都将替换 short_filename，并改为查找头文件 short_filename。此杂注必须在相应的 #include 指令之前出现。

要搜索的别名**必须完全符合规范，无论是大小写、拼写还是双引号或尖括号的使用**。include_alias 杂注对文件名执行简单的字符串匹配；将不执行任何其他文件名验证。例如：

```
#pragma include_alias("mymath.h", "math.h")           //要在头文件引用之前定义
```



举报



您可以使用 `include_alias` 杂注将任何头文件名映射到另一个头文件名。 例如

```
#pragma include_alias( "api.h", "c:\version1.0\api.h" )
#pragma include_alias( <stdio.h>, <newstdio.h> )
#include "api.h"
#include <stdio.h>
```

不要将用双引号括起来的文件名与用尖括号括起的文件名组合在一起。 例如，给定上述两个 `#pragma include_alias` 指令，编译器将不对下列 `#include` 指令执行任何替换：

```
#include <api.h>
#include "stdio.h"
```

2.7#pragma pack ([show] | [push | pop] [, identifier] , n)

指定结构、联合和类成员的封装对齐。其实就是改变编译器的内存对齐方式。这个功能对于集合数据体使用，默认的数据的对齐方式占用内存比较大，可进行修改。

在没有参数的情况下调用pack会将n设置为编译器选项/zp中设置的值。**如果未设置编译器选项，windows默认为8，linux默认为4。**

具体的使用方法为，其中n的取值必须是2的幂次方，即1、2、4、8、16等：

- | | |
|---------------------------------------|--|
| 1. #pragma pack(show) | 以警告信息的形式显示当前字节对齐的值。 |
| 2. #pragma pack(n) | 将当前字节对齐值设为 n 。 |
| 3. #pragma pack() | 将当前字节对齐值设为默认值(通常是8) 。 |
| 4. #pragma pack(push) | 将当前字节对齐值压入编译栈栈顶。 |
| 5. #pragma pack(pop) | 将编译栈栈顶的字节对齐值弹出并设为当前值。 |
| 6. #pragma pack(push, n) | 先将当前字节对齐值压入编译栈栈顶，然后再将 n 设为当前值。 |
| 7. #pragma pack(pop, n) | 将编译栈栈顶的字节对齐值弹出，然后丢弃，再将 n 设为当前值。 |
| 8. #pragma pack(push, identifier) | 将当前字节对齐值压入编译栈栈顶，然后将栈中保存该值的位置标识为 identifier 。 |
| 10. #pragma pack(pop, identifier) | 将编译栈栈中标识为 identifier 位置的值弹出，并将其设为当前值。注意，如果栈中所标识的位置之上还有值，那会5 |
| 11. #pragma pack(push, identifier, n) | 将当前字节对齐值压入编译栈栈顶，然后将栈中保存该值的位置标识为 identifier，再将 n 设为当前值。 |
| 12. #pragma pack(pop, identifier, n) | 将编译栈栈中标识为 identifier 位置的值弹出，然后丢弃，再将 n 设为当前值。注意，如果栈中所标识的位置之上还 |

注意：如果在栈中没有找到 pop 中的标识符，则编译器忽略该指令，而且不会弹出任何值。

使用时最好是成对出现的，要不容易引起错误，设置之后记得用完给恢复，如：

```
#pragma pack(n)      //设置以n个字节为对齐长度
struct
{
    int  ia;
    char cb;
}
#pragma pack ()      //弹出n个字节对齐长度，设置默认值为对齐长度
```

如果想给单独一个结构体设置对齐长度还可以使用C++ 11 标准中的alignas。

但是不是设置#pragma pack(n)就按照设置的字节进行对齐呢？其实并不是这样的，实际的对齐字节数要符合以下规则：

- 1、若设置了对齐长度n，实际对齐长度=Min（设置字节长度，结构体成员的最宽字节数）。若没有设置n，实际对齐长度 = Min（结构体成员的最宽字节数，默认对齐长度）。**
- 2、每个成员相对于首地址的偏移量（offset）都是实际对齐长度的整数倍，若不满足编译器进行填充。**
- 3、数据集合的总大小为实际对齐长度的整数倍，若不是编译器进行填充。**

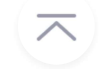
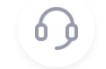
假设默认对齐为8时，看几个例子：

```
#pragma pack(n)
struct Stu1
{
    short   sa; //2个字节
    char    cb; //1个字节
    int     ic; //4个字节
    char    cd; //1个字节
}
#pragma pack()

cout << sizeof(Stu1) << endl;
```

若n为1：实际对齐长度为1 = Min(1,8)。这个就不用解释了，相当于各个元素相加，总长度为8。

若n为2：实际对齐长度为2 = Min(2,8)。sa占两个字节，不需要补齐。cb首地址偏移为2个字节，满足规则二。ic的首地址偏移3（2+1）个字节，不能满足规则二，填充一个字节到4。cd的首地址偏移为8个字节，满足规则。现在相加的8+1=9个字节，不满足规则三，填充一个字节，总长度为10；



2.8 其他


上面的有用过的，也有整理的实现现学的，我感觉有用处的。当然还有一些别的，但是我也没有看懂这里就不整理了，有需要的可以查看我下面的参考资料第2条，我列几个别的大概是干什么的。






```
#pragma hdrstop 停止编译头文件的，给我的感觉有点像“使用预编译头文件”
#pragma inline_depth( [0...255]) 设置内联调用深度的
...
...
```

感谢大家，我是假装很努力的YoungYangD（小羊）。

参考资料：


https://blog.csdn.net/jx_kingwei/article/details/367312
<https://msdn.microsoft.com/zh-cn/library/d9x1s805.aspx>

 **YoungYangD** [关注](#)


 59  2  247   [专栏目录](#)

#pragma的用法 04-24
#pragma语句在嵌入式系统程序中不可小觑啊。

#pragma预处理指令用法详解 02-01
描述了**#pragma** 预处理指令的含义及用法。



优质评论可以帮助作者获得更高权重



评论

#pragma用法_汇总.doc 08-18
#pragma用法 汇总 doc 最近总有人问**#pragma** CODE SEG NEAR SEG NON BANKED 还有**#pragma** LINK INFO DERIVATIVE "mc9s12xs128"这些函数是什么意思 我在...

解析**#pragma** 简约而不简单 2688
在所有的预处理指令中，**#pragma** 指令可能是最复杂的了，它的作用是设定编译器的状态或者是指示编译器完成一些特定的动作。**#pragma**指令对每个编译器给出了一个...

Visual C++ 6.0编译指示 一叶舟轻 2323
Document Source:**Pragma** Directives, **Pre**processor Reference, Visual C++ **Pro**grammer Guide. 每种C和C++的实现支持对其宿主机或操作系统唯一的功能。例如，一些...

C语言**#pragma**使用**方法** **热门推荐** liuchunjie11的博客 3万+
C语言**#pragma** pack使用**方法**

#pragma详细解释(一) weixin_34176694的博客 385
在**#Pragma**是预处理指令它的作用是设定编译器的状态或者是指示编译器完成一些特定的动作。**#pragma**指令对每个编译器给出了一个**方法**,在保持与C和C ++语言完全兼...

#pragma预处理命令 weixin_34216107的博客 472
#pragma可以说是C++中最复杂的预处理指令了，下面是**最常用**的几个**#pragma**指令：**#pragma** comment(lib,"XXX.lib") 表示链接XXX.lib这个库，和在工程设置里写上XX...

#pragma用法详解 weixin_30532837的博客 161
Author :Jeffrey My Blog：http://blog.csdn.net/gueter/ 目录：（0）前言（1）**#pragma** mess**age**能够在编译信息输出窗口中输出相应的信息（2）**#pragma** code_seg能够...

Window下启动java程序，包含第三方jar包 04-17
Window下启动java程序，包含第三方jar包，详情查看：http://blog.csdn.net/jptiancai/article/details/23770713

#pragma 在嵌入式中的**讲解**和理解 **最新发布** 08-10
#pragma //提供额外信息的标准**方法**，可用于指定平台。这个标记其实是很复杂的，它是什么特点呢，它是根据你的编译平台，就是根据你所用的不同的编译器然后你再...

#pragma大全 mail_cm的专栏 812
C和C++的每个实现对它的主机或操作系统都支持一些独有的特征。 例如,某些程序须对存放数据的存储器区域进行精确的控制,或必须控制特定函数接受参量的方式。**#pra**...

#pragma详细解释 weixin_30945039的博客 124
#pragma详细解释(一) 2010-04-18 14:21:00| 分类：默认分类 | 标签：|字号大中小订阅 在**#Pragma**是预处理指令它的作用是设定编译器的状态或者是指示编译器完成一...





#pragma comment 的使用**方法** haiross的专栏 9659
[cpp] view plaincopy **#pragma** comment (lib,"wpcap.lib") 表示链接wpcap.lib这个库。 和在工程设置里写上链入wpcap.lib的效果一样（两种方式等价，或说一个隐式一个...





c++中**#pragma**用法详解 Poo_Chai的博客 3881
在所有的预处理指令中，**#Pragma** 指令可能是最复杂的了，它的作用是设定编译器的状态或者是指示编译器完成一些特定的动作。**#pragma**指令对每个编译器给出了一个...





#pragma详解 海纳百川 3747
#pragma 求助编辑百科名片 在所有的预处理指令中，**#Pragma** 指令可能是最复杂的了，它的作用是设定编译器的状态或者是指示编译器完成一些特定的动作。**#pragma**指...





#pragma指令的使用 麦田里的码农 5897
前言 **#pragma**指令为我们提供了让编译器执行某些特殊操作提供了一种**方法**。这条指令对非常大的程序或需要使用特定编译器的特殊功能的程序非常有用。**#pragma**...




#pragma的详细用法 lanlxb的博客 125
每种C和C++的实现支持对其宿主机或操作系统唯一的功能。例如，一些程序需要精确控制超出数据所在的储存 空间，或着控制特定函数接受参数的方式。**#pragma**指示使...

[关于我们](#) [招贤纳士](#) [广告服务](#) [开发助手](#)  400-660-0108  kefu@csdn.net  [在线客服](#) 工作时间 8:30-22:00

[公安备案号11010502030143](#) [京ICP备19004658号](#) [京网文〔2020〕1039-165号](#) [经营性网站备案信息](#) [北京互联网违法和不良信息举报中心](#)
[网络110报警服务](#) [中国互联网举报中心](#) [家长监护](#) [Chrome商店下载](#) ©1999-2021北京创新乐知网络技术有限公司 [版权与免责声明](#) [版权申诉](#)
[出版物许可证](#) [营业执照](#)