

行者的学习博客

博客园 :: 首页 :: 博问 :: 闪存 :: 新随笔 :: 联系 :: 订阅 [XML](#) :: 管理 ::

83 随笔 :: 0 文章 :: 28 评论 :: 0 引用

2017年1月						
日	一	二	三	四	五	六
25	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	31	1	2	3	4

搜索

 找找看

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)

我的标签

[VC\(26\)](#)
[Ansys\(22\)](#)
[Tcl/Tk\(7\)](#)
[Csharp\(7\)](#)
[Flex\(5\)](#)
[Java\(3\)](#)
[CAD\(3\)](#)
[杂项\(3\)](#)
[Matlab\(2\)](#)
[JavaScript\(1\)](#)
[更多](#)

随笔档案(83)

[2010年11月 \(1\)](#)
[2010年9月 \(1\)](#)
[2010年8月 \(1\)](#)
[2010年7月 \(3\)](#)
[2010年6月 \(2\)](#)
[2010年5月 \(1\)](#)
[2010年4月 \(3\)](#)
[2010年3月 \(4\)](#)
[2010年1月 \(7\)](#)
[2009年12月 \(8\)](#)
[2009年11月 \(4\)](#)
[2009年10月 \(8\)](#)
[2009年9月 \(3\)](#)
[2009年8月 \(1\)](#)
[2009年7月 \(11\)](#)
[2009年6月 \(6\)](#)
[2009年5月 \(5\)](#)
[2009年3月 \(7\)](#)
[2009年2月 \(7\)](#)

个人博客

[行者个人博客](#)

c++ 中__declspec 的用法

c++ 中__declspec 的用法

语法说明：

__declspec (extended-decl-modifier-seq)

扩展修饰符：

1 : align(#)

用__declspec(align(#))精确控制用户自定数据的对齐方式，#是对齐值。

e.g

```
__declspec(align(32))
struct Str1{
int a, b, c, d, e;
};
```

【转】它与#pragma pack()是一对兄弟，前者规定了对齐的最小值，后者规定了对齐的最大值。同时出现时，前者优先级高。__declspec(align())的一个特点是，它仅仅规定了数据对齐的位置，而没有规定数据实际占用的内存长度，当指定的数据被放置在确定的位置之后，其后的数据填充仍然是按照#pragma pack规定的方式填充的，这时候类/结构的实际大小和内存格局的规则是这样的：在__declspec(align())之前，数据按照#pragma pack规定的方式填充，如前所述。当遇到__declspec(align())的时候，首先寻找距离当前偏移向后最近的对齐点（满足对齐长度为max(数据自身长度,指定值)），然后把被指定的数据类型从这个点开始填充，其后的数据类型从它的后面开始，仍然按照#pragma pack填充，直到遇到下一个__declspec(align())。当所有数据填充完毕，把结构的整体对齐数值和__declspec(align())规定的值做比较，取其中较大的作为整个结构的对齐长度。特别的，当__declspec(align())指定的数值比对应类型长度小的时候，这个指定不起作用。

2: allocate("segname")

用__declspec(allocate("segname"))声明一个已经分配了数据段的一个数据项。它和#pragma的code_seg, const_seg, data_seg, section, init_seg配合使用，segname必须有这些东东声明。

e.g

```
#pragma data_seg("share_data")
int a = 0;
int b;
#pragma data_seg() __declspec(allocate("share_data")) int c = 1;
__declspec(allocate("share_data")) int d;
```

3. deprecated

用__declspec(deprecated)说明一个函数，类型，或别的标识符在新的版本或未来版本中不再支持，你不应该用这个函数或类型。它和#pragma deprecated作用一样。

最新评论**1. Re:发送命令给AutoCAD的方法,比如Cancel命令**

你好,我想在反应器中终止AutoCAD的保存操作,请问有什么方法吗? void
Reactor::commandWillStart(const ACHAR * cmdStr){ if
(wcscmp(c.....

--limboPitt

2. Re:卷积的物理意义

过了两年回来看,这个帖子依旧很是实用和搞笑

--Biran

3. Re:ANSYS使用技巧(二)螺栓预载荷定义

恩,感谢你的回答!
想请问有没有相关实例,我现在大多情况下使用的是gui,并非命令流,读你这段关于定义螺栓预紧载荷单元及施加载荷的命令流有些生涩,所以希望通过实例加深理解!谢谢!

--请叫我乳鸽

4. Re:ANSYS使用技巧(二)螺栓预载荷定义

@请叫我乳鸽账号是几年前上学的时候用的,现在不用了,基本没有回复。建模的意思就是创建螺母和螺栓组合模型,两个圆环就是螺帽和螺母。螺母和螺栓可用体粘接,此处感觉体粘接和面粘接效果应该是一样的。...

--ylclass

5. Re:ANSYS使用技巧(二)螺栓预载荷定义

"直接使用ANSYS中创建体的命令创建一个圆柱和两个圆环,组合成螺栓,注意要将螺栓粘接起来"——请问,将螺栓粘结起来,指的是将螺栓柱的表面与螺母内表面粘结起来么?还是将螺栓柱的体与螺母的体粘结起来?...

--请叫我乳鸽

e.g

#define MY_TEXT "function is deprecated"

void func1(void) {}

__declspec(deprecated) void func1(int) { printf("func1n");}

```
__declspec(deprecated("*** this is a deprecated function **")) void func2(int) {
printf("func2n");}
```

__declspec(deprecated(MY_TEXT)) void func3(int) { printf("func3");}

int main()

{

fun1();

fun2();

fun3();

}

4.dllimport 和dllexport

用__declspec(dllexport), __declspec(dllimport)显式的定义dll接口给调用它的exe或dll文件,用 dllexport定义的函数不再需要(.def)文件声明这些函数接口了。注意:若在dll中定义了模板类那它已经隐式的进行了这两种声明,我们只需在调用的时候实例化即可,呵呵。

e.g 常规方式dll中

class __declspec(dllexport)

testdll{

testdll(){};

~testdll(){};

};

调用客户端中声明

#import comment(lib, "***.lib)

class __declspec(dllimportt)

testdll{

testdll(){};

~testdll(){};

};

e.g 模板类:dll中

template<class t>

class test{

test(){};

~test(){};

}

调用客户端中声明

int main()

{

test< int > b;

return 0;

}

5. jitintrinsic

用__declspec(jitintrinsic)标记一个函数或元素为64位公共语言运行时。具体用法未见到。

6. __declspec(naked)

对于没有用naked声明的函数一般编译器都会产生保存现场(进入函数时编译器会产生代码

来保存ESI, EDI, EBX, EBP寄存器 ——prolog) 和清除现场 (退出函数时则产生代码恢复这些寄存器的内容——epilog) 代码, 而对于用naked声明的函数一般不会产生这些代码, 这个属性对于写设备驱动程序非常有用, 我们自己可以写这样一个过程, 它仅支持x86。naked只对函数有效, 而对类型定义无效。对于一个标志了naked的函数不能产生一个内联函数即时使用了__forceinline 关键字。

```
e.g __declspec ( naked ) func()
{
int i;
int j;
__asm    /* prolog */
{
push ebp
mov     ebp, esp
sub     esp, __LOCAL_SIZE
}
/* Function body */
__asm    /* epilog */
{
mov     esp, ebp
pop     ebp
ret
}
}
```

7. restrict 和 noalias

__declspec(restrict) 和 __declspec(noalias)用于提高程序性能, 优化程序。这两个关键字都仅用于函数, restrict针对于函数返回指针, restrict 说明函数返回值没有被别名化, 返回的指针是唯一的, 没有被别的函数指针别名化, 也就是说返回指针还没有被用过是唯一的。编译器一般会去检查指针是否可用和 是否被别名化, 是否已经在使用, 使用了这个关键字, 编译器就不在去检查这些信息了。noalias 意味着函数调用不能修改或引用可见的全局状态并且仅仅修改指针参数直接指向的内存。如果一个函数指定了noalias关键字, 优化器认为除参数自生之外, 仅仅参数指针第一级间接是被引用或修改在函数内部。可见全局状态是指没有定义或引用在编码范围外的全部数据集, 它们的直至不可以取得。编码范围是指所有源 文件或单个源文件。其实这两个关键字就是给编译器了一种保证, 编译器信任他就不在进行一些检查操作了。

```
e.g
#include <stdio.h>
#include <stdlib.h>
#define M 800#define N 600#define P 700float * mempool, * memptr;
__declspec(restrict) float * ma(int size)
{
float * retval;
    retval = memptr;
    memptr += size;
return retval;
}
__declspec(restrict) float * init(int m, int n)
{
float * a;
    int i, j;
int k=1;
a = ma(m * n);
    if (!a) exit(1);
```

```

for (i=0; i<m; i++)
    for (j=0; j<n; j++)
        a[i*n+j] = 0.1/k++;
return a;
}
__declspec(noalias) void multiply(float * a, float * b, float * c)
{
    int i, j, k;
    for (j=0; j<P; j++)
        for (i=0; i<M; i++)
            for (k=0; k<N; k++)
                c[i * P + j] = a[i * N + k] * b[k * P + j];
}

int main()
{
    float * a, * b, * c;
    mempool = (float *) malloc(sizeof(float) * (M*N + N*P + M*P));
    if (!mempool)
        puts("ERROR: Malloc returned null"); exit(1);
    memptr = mempool;
    a = init(M, N);
    b = init(N, P);
    c = init(M, P);
    multiply(a, b, c);
}

```

8. noinline __declspec(noinline)

告诉编译器不去内联一个具体函数。

9. noreturn __declspec(noreturn)

告诉编译器没有返回值.注意添加__declspec(noreturn)到一个不希望返回的函数会导致已没有定义错误.

10. nothrow __declspec(nothrow)

用于函数声明,它告诉编译器函数不会抛出异常。

```

e.g
#define WINAPI __declspec(nothrow) __stdcall
void WINAPI f1();
void __declspec(nothrow) __stdcall f2();
void __stdcall f3() throw();

```

11. novtable __declspec(novtable)

用在任意类的声明,但是只用在纯虚接口类,因此这样的不能够被自己实例化.它阻止编译器初始化虚表指针在构造和析构类的时候,这将移除对关联到类的虚表的引用.如果你尝试这实例化一个有novtable关键字的类,它将发生AV(access violation)错误.C++里virtual的缺陷就是vtable会增大代码的尺寸,在不需要实例化的类或者纯虚接口的时候,用这个关键字可以减小代码的大小.

```

e.g
#if _MSC_VER >= 1100 && !defined(_DEBUG)
#define AFX_NOVTABLE __declspec(novtable)
#else

```

```

#define AFX_NOVTABLE
#endif
....
class AFX_NOVTABLE CObject
{
...
};

```

这是vc里面的一段代码,我们可以看出编译Release版本时,在CObject前是__declspec(novtable),在debug版本没有这个限制。

```

e.g
#include <stdio.h>
struct __declspec(novtable) X
{
virtual void mf();
};
struct Y : public X
{
void mf()
{
printf_s("In Yn");
}
};

```

12.selectany的作用 (转)

__declspec(selectany)可以让我们在.h文件中初始化一个全局变量而不是只能放在.cpp中。比如有一个类,其中有一个静态变量,那么我们可以在.h中通过类似"**__declspec(selectany) type class::variable = value;**"这样的代码来初始化这个全局变量。既是该.h被多次include,链接器也会为我们剔除多重定义的错误。这个有什么好处呢,我觉得对于 template的编程会有很多便利。

```

e.g
class test
{
public:
static int t;
};
__declspec(selectany) int test::t = 0;

```

13.thread

thread 用于声明一个线程本地变量。**__declspec(thread)**的前缀是Microsoft添加给Visual C++编译器的一个修改符。它告诉编译器,对应的变量应该放入可执行文件或DLL文件中它的自己的节中。**__declspec(thread)**后面的变量 必须声明为函数中(或函数外)的一个全局变量或静态变量。不能声明一个类型为**__declspec(thread)**的局部变量。

```

e.g
__declspec(thread)
class X{
public:
int I;
} x; // x is a thread object
X y; // y is not a thread object

```

14.uuid__declspec(uuid)

用于编译器关联一个GUID到一个有uuid属性的类或结构的声明或者定义.

```
e.g struct __declspec(uuid("00000000-0000-0000-c000-000000000046"))
IUnknown; struct __declspec(uuid("{00020400-0000-0000-c000-
000000000046}")) IDispatch; 我们可以在MFC中查看源码.:)
```

[转]<http://hi.baidu.com/zibbio/blog/item/49ad7654ce1c37c3b745aecf.html>

标签: VC



ylclass

关注 - 2

粉丝 - 27

+加关注

2

推荐

0

反对

« 上一篇: 给单文档创建button

» 下一篇: vc中调用Com组件的方法详解

posted on 2010-07-10 10:59 ylclass 阅读(29423) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

最新IT新闻:

- 尴尬：腾讯游戏《智龙迷城》上线6个月就宣告死亡
 - 现在可以在iPhone和iPad上加入Office Insider项目了
 - 2017年Google Cloud Next开放注册 1月17号前注册优惠\$500
 - 计算机科学家迈克尔·乔丹：霍金是外行 几百年内AI不会觉醒
 - 荷兰火车全部使用风能 提前一年完成目标
- » [更多新闻...](#)

最新知识库文章:

- 「代码家」的学习过程和学习经验分享
 - 写给未来的程序媛
 - 高质量的工程代码为什么难写
 - 循序渐进地代码重构
 - 技术的正宗与野路子
- » [更多知识库文章...](#)

Powered by:

[博客园](#)

Copyright © ylclass