**1. typedef 最简单使用**

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef long byte\_4; // 给已知数据类型long起个新名字，叫byte\_4



你可以在任何需要 long 的上下文中使用 byte\_4。注意 typedef 并不创建新的类型。它仅仅为现有类型添加一个同义字。

**2. typedef 修饰数组**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef **char** mySizes[100];
2. mySizes xxx;
3. typedef **char** [100] mySizes; //error

typedef char mySizes[100];

mySizes xxx;

typedef char [100] mySizes; //error

这里 mySize 就是一个大小为100的 char 数组，sizeof(mySize) 为 100。

**3. typedef 修饰指针**

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef char \* pstr;
2. int mystrcmp(pstr, pstr);
3. ......
4. int mystrcmp(const pstr, const pstr); //error



这里有一个 typedef 陷阱。标准函数 strcmp()有两个‘const char \*’类型的参数。因此，它可能会误导人们象上面这样声明。

按照顺序，‘const pstr’被解释为‘char \* const’（一个指向 char 的常量指针），而不是‘const char \*’（指向常量 char 的指针）。

其实这点很好理解，const 就是**修饰** pstr 指针的，并不是简单替换。

这个问题很容易解决：

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef const char \* cpstr;
2. int mystrcmp(cpstr, cpstr); // 现在是正确的



**4. typedef 修饰结构体**

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef struct tagMyStruct
2. {
3. int iNum;
4. long lLength;
5. } MyStruct;



这语句实际上完成两个操作：  
(1).定义一个新的结构类型

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. struct tagMyStruct
2. {
3. int iNum;
4. long lLength;
5. };



分析：

tagMyStruct ，实际上是一个临时名字，struct 关键字和 tagMyStruct一起，构成了这个结构类型，不论是否有typedef，这个结构都存在。

我们可以用 struct tagMyStruct xxName 来定义变量，但要注意，使用tagMyStruct xxxrName 来定义变量是不对的，因为struct 和tagMyStruct合在一起才能表示一个结构类型。

(2). typedef 为这个新的结构起了一个名字，叫 MyStruct。

typedef struct tagMyStruct MyStruct；因此，MyStruct实际上相当于struct tagMyStruct，我们可以使用MyStruct varName来定义变量。

**5. typedef & 结构的问题**

在结构中包含指向它自己的指针

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef struct tagNode
2. {
3. char \*pItem;
4. pNode pNext; // error
5. } \*pNode;

  
答案与分析：   
根据我们上面的阐述可以知道：要知道pNode表示的是类型的新名字，那么在类型本身还没有建立完成的时候，这个类型的新名字 pNoed 也还不存在，也就是说这个时候编译器根本不认识pNode。

解决这个问题的方法有多种：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. // 1)
2. typedef struct tagNode
3. {
4. **char** \*pItem;
5. struct tagNode \*pNext;
6. } \*pNode;
7. // 2)
8. typedef struct tagNode\* pNode;
9. struct tagNode
10. {
11. **char** \*pItem;
12. pNode pNext;
13. };
14. //注意：在这个例子中，你用 typedef 给一个还未完全声明的类型起新名字。C语言编译器支持这种做法。
15. // 3)规范做法：
16. struct tagNode
17. {
18. **char** \*pItem;
19. struct tagNode \*pNext;
20. };
21. typedef struct tagNode \*pNode;

// 1)

typedef struct tagNode

{

char \*pItem;

struct tagNode \*pNext;

} \*pNode;

// 2)

typedef struct tagNode\* pNode;

struct tagNode

{

char \*pItem;

pNode pNext;

};

//注意：在这个例子中，你用 typedef 给一个还未完全声明的类型起新名字。C语言编译器支持这种做法。

// 3)规范做法：

struct tagNode

{

char \*pItem;

struct tagNode \*pNext;

};

typedef struct tagNode \*pNode;

**6. typedef 与 #define的问题**  
有下面两种定义pStr数据类型的方法，两者有什么不同？哪一种更好一点？

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef char\* pStr;
2. #define pStr char\*;



答案与分析：

通常讲，typedef要比#define要好，特别是在有指针的场合。  
请看例子：

**[c-sharp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. typedef char\* pStr1;
2. #define pStr2 char \*
3. pStr1 s1, s2; // char\* s1; char\* s2;
4. pStr2 s3, s4; // char\* s3, s4;即 char s4;

  
在上述的变量定义中，s4则定义成了char，不是我们所预期的指针变量，根本原因就在于#define只是简单的字符串替换而typedef则是为一个类型起新名字。

上例中define语句必须写成 pStr2 s3, \*s4; 这这样才能正常执行。

**7. typedef 与 复杂的变量声明**

在编程实践中，尤其是看别人代码的时候，常常会遇到比较复杂的变量声明,使用typedef作简化自有其价值，比如： 　　  
下面是三个变量的声明，我想使用typdef分别给它们定义一个别名，请问该如何做？

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. **int** \*(\*a[5])(**int**, **char**\*);
2. void (\*b[10]) (void (\*)());
3. **double**(\*)() (\*pa)[9];

int \*(\*a[5])(int, char\*);

void (\*b[10]) (void (\*)());

double(\*)() (\*pa)[9];

答案与分析：

对复杂变量建立一个类型别名的方法很简单，你只要在传统的变量声明表达式里用类型名替代变量名，然后把关键字typedef加在该语句的开头就行了。

**8. typedef 修饰函数指针**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6191942)

1. //1. 定义一个函数指针
2. typedef **int**(\_cdecl \*FUN\_Start)(**UCHAR**);
3. /\* typedef的功能是定义新的类型。定义这种类型为指向某种函数的指针，这函数以一个UCHAO为参数并返回int类型。\*/
4. //2. 定义这个函数指针的一个变量
5. FUN\_Start fun\_Start;
6. //3. 把函数的地址赋给此函数指针
7. fun\_Start = (FUN\_Start)GetProcAddress(m\_hLibrary,"Rec\_SetDevice");
8. //4. 调用
9. if (fun\_Start('a') == 1) //直接通过函数指针调用
10. {......}
11. //当然也可以这样
12. if ( (\*)fun\_Start('a') == 1) //先用\*fun\_start取出它所指的函数类型，再调用
13. {......}

//1. 定义一个函数指针

typedef int(\_cdecl \*FUN\_Start)(UCHAR);

/\* typedef的功能是定义新的类型。定义这种类型为指向某种函数的指针，这函数以一个UCHAO为参数并返回int类型。\*/

//2. 定义这个函数指针的一个变量

FUN\_Start fun\_Start;

//3. 把函数的地址赋给此函数指针

fun\_Start = (FUN\_Start)GetProcAddress(m\_hLibrary,"Rec\_SetDevice");

//4. 调用

if (fun\_Start('a') == 1) //直接通过函数指针调用

{......}

//当然也可以这样

if ( (\*)fun\_Start('a') == 1) //先用\*fun\_start取出它所指的函数类型，再调用

{......}

因为函数名就是一个地址，该函数名所代表的函数的入口地址。

注：

总之一点，不要把typedef看成简单的替换，要看成一种新的命名，要与default相区别！！