<https://blog.csdn.net/ghevinn/article/details/24792209>

---------------------<typedef巧用和注意事项>-----------------------------  
<基本概念>  
typedef声明，简称typedef，为现有类型创建一个新的名字，或称为类型别名，在结构体定义，还有一些数组等地方都大量的用到。  
它有助于创建平台无关类型，甚至能隐藏复杂和难以理解的语法 。使用typedef可编写出更加美观和可读的代码。所谓美观，意指typedef能隐藏笨拙的语法构造以及平台相关的数据类型，从而增强可移植性以及未来的可维护性。  
  
  
<简单用法>  
使用typedef为现有类型创建同义字，定义易于记忆的类型名  
typedef使用最多的地方是创建易于记忆的类型名，用它来归档程序员的意图。类型出现在所声明的变量名字中，位于“typedef”关键字右边。例如：  
typedef int size;  
此声明定义了一个int的同义字，名字为size。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。你可以在任何需要int的上下文中使用size：  
void measure(size \* psz);  
size array[4];  
size len = file.getlength();  
std::vector<size> vs;  
typedef 还可以掩饰复合类型，如指针和数组。  
例如，你不用像下面这样重复定义有 81 个字符元素的数组：  
char line[81];  
char text[81];  
定义一个 typedef，每当要用到相同类型和大小的数组时，可以这样：  
typedef char Line[81];  
此时Line类型即代表了具有81个元素的字符数组，使用方法如下：  
Line text, secondline;  
getline(text);  
同样，可以像下面这样隐藏指针语法：  
typedef char \* pstr;  
int mystrcmp(pstr p1, pstr p2);  
这里将带我们到达第一个 typedef 陷阱。标准函数 strcmp()有两个‘ const char \*'类型的参数。因此，它可能会误导人们像下面这样声明 mystrcmp()：  
int mystrcmp(const pstr p1, const pstr p3);  
用GNU的gcc和g++编译器，是会出现警告的，按照顺序，‘const pstr'被解释为‘char\* const‘（一个指向char的指针常量），两者表达的并非同一意思（详见C++ Primer 第四版 P112）。  
char \* const cp : 定义一个指向字符的指针常数，即const指针，常指针。  
const char\* p : 定义一个指向字符常数的指针，即常量指针。  
char const\* p : 等同于const char\* p[2]。  
为了得到正确的类型，应当如下声明：  
typedef const char\* pstr;  
  
  
<深入探究>  
基本解释  
typedef为C语言的关键字，作用是为一种数据类型定义一个新名字。这里的数据类型包括内部数据类型（int,char等）和自定义的数据类型（struct等）。  
在编程中使用typedef目的一般有两个，一个是给变量一个易记且意义明确的新名字，另一个是简化一些比较复杂的类型声明。  
至于typedef有什么微妙之处，请你接着看下面对几个问题的具体阐述。  
2. typedef & 结构的问题  
当用下面的代码定义一个结构时，编译器报了一个错误，为什么呢？莫非C语言不允许在结构中包含指向它自己的指针吗？请你先猜想一下，然后看下文说明：  
typedef struct tagNode  
{  
char \*pItem;  
pNode pNext;  
} \*pNode;  
分析：  
1、typedef的最简单使用  
typedef long byte\_4;  
给已知数据类型long起个新名字，叫byte\_4。  
2、 typedef与结构结合使用  
typedef struct tagMyStruct  
{  
int iNum;  
long lLength;  
} MyStruct;  
这语句实际上完成两个操作：  
1) 定义一个新的结构类型  
struct tagMyStruct  
{  
int iNum;  
long lLength;  
};  
分析：tagMyStruct称为“tag”，即“标签”，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 varName来定义变量，但要注意，使用tagMyStruct varName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。  
答案与分析  
C语言当然允许在结构中包含指向它自己的指针，我们可以在建立链表等数据结构的实现上看到无数这样的例子，上述代码的根本问题在于typedef的应用。  
根据我们上面的阐述可以知道：新结构建立的过程中遇到了pNext域的声明，类型是pNode，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：  
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;  
3. typedef & #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？  
typedef char\* pStr;  
#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：  
typedef char\* pStr1;  
#define pStr2 char \*  
pStr1 s1, s2;  
pStr2 s3, s4;  
在上述的变量定义中，s1、s2、s3都被定义为char \*，而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。  
#define用法例子：  
#define f(x) x\*x  
main( )  
{  
int a=6，b=2，c；  
c=f(a) / f(b)；  
printf("%d \\n"，c)；  
}  
以下程序的输出结果是: 36。  
因为如此原因，在许多C语言编程规范中提到使用#define定义时，如果定义中包含表达式，必须使用括号，则上述定义应该如下定义才对：  
#define f(x) ((x)\*(x))  
当然，如果你使用typedef就没有这样的问题。  
4. typedef & #define的另一例  
下面的代码中编译器会报一个错误，你知道是哪个语句错了吗？  
typedef char \* pStr;  
char string[4] = "abc";  
const char \*p1 = string;  
const pStr p2 = string;  
p1++;  
p2++;  
答案与分析：  
是p2++出错了。这个问题再一次提醒我们：typedef和#define不同，它不是简单的文本替换。上述代码中const pStr p2并不等于const char \* p2。const pStr p2和const long x本质上没有区别，都是对变量进行只读限制，只不过此处变量p2的数据类型是我们自己定义的而不是系统固有类型而已。因此，const pStr p2的含义是：限定数据类型为char \*的变量p2为只读，因此p2++错误。  
#define与typedef引申谈  
1) #define宏定义有一个特别的长处：可以使用 #ifdef ,#ifndef等来进行逻辑判断，还可以使用#undef来取消定义。  
2) typedef也有一个特别的长处：它符合范围规则，使用typedef定义的变量类型其作用范围限制在所定义的函数或者文件内（取决于此变量定义的位置），而宏定义则没有这种特性。  
5. typedef & 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？  
>1：int \*(\*a[5])(int, char\*);  
>2：void (\*b[10]) (void (\*)());  
>3. doube(\*)() (\*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。  
>1：int \*(\*a[5])(int, char\*);  
//pFun是我们建的一个类型别名  
typedef int \*(\*pFun)(int, char\*);  
//使用定义的新类型来声明对象，等价于int\* (\*a[5])(int, char\*);  
pFun a[5];  
>2：void (\*b[10]) (void (\*)());  
//首先为上面表达式蓝色部分声明一个新类型  
typedef void (\*pFunParam)();  
//整体声明一个新类型  
typedef void (\*pFun)(pFunParam);  
//使用定义的新类型来声明对象，等价于void (\*b[10]) (void (\*)());  
pFun b[10];  
>3. doube(\*)() (\*pa)[9];  
//首先为上面表达式蓝色部分声明一个新类型  
typedef double(\*pFun)();  
//整体声明一个新类型  
typedef pFun (\*pFunParam)[9];  
//使用定义的新类型来声明对象，等价于doube(\*)() (\*pa)[9];  
pFunParam pa;  
=============================<typedef巧用和注意事项>====================================