<https://baike.baidu.com/item/typedef/9558154?fr=aladdin>

## 用法总结

[编辑](javascript:;)

如何创建平台无关的数据类型，隐藏笨拙且难以理解的语法：

使用typedef为现有类型创建别名，定义易于记忆的[类型名](https://baike.baidu.com/item/%E7%B1%BB%E5%9E%8B%E5%90%8D" \t "_blank)

|  |  |
| --- | --- |
| 1 | typedef　int　size; |
| 1  2  3  4  5  6  7  8  9  10 | void　measure(size\*psz);      size　array[4];      size　len=file.getlength();      std::vector<size>vs; |

typedef 还可以掩饰[复合](https://baike.baidu.com/item/%E5%A4%8D%E5%90%88/2437031)类型，如[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)和[数组](https://baike.baidu.com/item/%E6%95%B0%E7%BB%84)。

例如，你不用像下面这样重复定义有 81 个字符元素的数组：

|  |  |
| --- | --- |
| 1  2  3 | char　line[81];    char　text[81]; |

只需这样定义，Line类型即代表了具有81个元素的[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "_blank)数组，使用方法如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | typedef char Line[81];    Line　text,line;    getline(text); |

同样，可以像下面这样隐藏[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "_blank)语法：

|  |  |
| --- | --- |
| 1 | typedef　char\*　pstr; |
| 1 | int　mystrcmp(const　pstr　p1,const　pstr　p3); |

用GNU的gcc和g++[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)，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](https://baike.baidu.com/item/%E6%8C%87%E9%92%88%E5%B8%B8%E9%87%8F" \t "_blank)），而事实上，const char\*和char\* const表达的并非同一意思（详见C++ Primer 第四版 P112）。

char \* const p : 定义一个指向字符的指针常数，即const指针，常量指针。

const char\* p : 定义一个指向字符型常量的指针。

char const\* p : 等同于const char\* p。

## 语言用法

[编辑](javascript:;)

**基本解释**

typedef为C语言的[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97" \t "_blank)，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](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)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](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "_blank)吗？请你先猜想一下，然后看下文说明：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | typedef　struct　tagNode  {    char\*　pItem;    pNode\*　pNext;    }pNode; |

分析：

1、typedef的最简单使用

|  |  |
| --- | --- |
| 1 | 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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | 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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | 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](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97" \t "_blank)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](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "_blank)，我们可以在建立[链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8)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](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8" \t "_blank)根本不认识pNode。

解决这个问题的方法有多种：

1)、

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | typedef　struct　tagNode  {      char\*　pItem;      struct　tagNode\*　pNext;    }\*pNode; |

2)、

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | typedef　struct　tagNode\*　pNode;      struct　tagNode  {    char\*　pItem;    pNode　pNext;//这边不用pNode\* ，pNode 已经表示了struct tagNode\*  }; |

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](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8" \t "_blank)支持这种做法。

3)、规范做法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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？

|  |  |
| --- | --- |
| 1  2  3 | 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](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "_blank)的场合。请看例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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](https://baike.baidu.com/item/%E6%8C%87%E9%92%88%E5%8F%98%E9%87%8F" \t "_blank)，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; 这样才能正常执行。

#define用法例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <stdio.h>  #define f(x) x\*x  int main(void)  {      int a=6, b=2, c;      c = f(a) / f(b);      printf("%d\n", c);      return 0;  } |

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](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "_blank)，必须使用括号，则上述定义应该如下定义才对：

|  |  |
| --- | --- |
| 1 | #definef(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](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)会报一个错误，你知道是哪个语句错了吗？

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 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](https://baike.baidu.com/item/%E6%96%87%E6%9C%AC%E6%9B%BF%E6%8D%A2" \t "_blank)。上述代码中const pStr p2并不等于const char \* p2。const pStr p2和pStr const p2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](https://baike.baidu.com/item/%E5%AE%8F%E5%AE%9A%E4%B9%89)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

理解复杂声明可用的“右左法则”：  
　　从变量名看起，先往右，再往左，碰到一个圆括号就调转阅读的方向；括号内分析完就跳出括号，还是按先右后左的顺序，如此循环，直到整个声明分析完。举例：  
　　int (\*func)(int \*p);  
　　首 先找到变量名func，外面有一对圆括号，而且左边是一个\*号，这说明func是一个指针；然后跳出这个圆括号，先看右边，又遇到圆括号，这说明 (\*func)是一个函数，所以func是一个指向这类函数的指针，即函数指针，这类函数具有int\*类型的形参，返回值类型是int。  
　　int (\*func[5])(int \*);  
　　func 右边是一个[]运算符，说明func是具有5个元素的数组；func的左边有一个\*，说明func的元素是指针（注意这里的\*不是修饰func，而是修饰 func[5]的，原因是[]运算符优先级比\*高，func先跟[]结合）。跳出这个括号，看右边，又遇到圆括号，说明func数组的元素是函数类型的指 针，它指向的函数具有int\*类型的形参，返回值类型为int。

也可以记住2个模式：

type (\*)(....)函数指针

type (\*)[]数组指针

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. 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](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F)里用[类型名](https://baike.baidu.com/item/%E7%B1%BB%E5%9E%8B%E5%90%8D)替代变量名，然后把[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97)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](https://baike.baidu.com/item/%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "_blank)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. double(\*[1]  (\*pa)[9])()[2]  ;

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

pFunParam pa;

pa是一个指针，指针指向一个数组，这个数组有9个元素，每一个元素都是“doube(\*)()”--也即一个指针，指向一个函数，函数参数为空，返回值是“double”。

## 代码简化

[编辑](javascript:;)

上面讨论的 typedef 行为有点像 #define 宏，用其实际类型替代同义字。不同点是 typedef 在编译时被解释，因此让[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8" \t "_blank)来应付超越[预处理器](https://baike.baidu.com/item/%E9%A2%84%E5%A4%84%E7%90%86%E5%99%A8)能力的[文本替换](https://baike.baidu.com/item/%E6%96%87%E6%9C%AC%E6%9B%BF%E6%8D%A2)。例如：

typedef int (\*PF) (const char \*, const char \*);

这个声明引入了 PF 类型作为[函数指针](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E6%8C%87%E9%92%88" \t "_blank)的同义字，该函数有两个 const char \* 类型的参数以及一个 int 类型的返回值。如果要使用下列形式的函数声明，那么上述这个 typedef 是不可或缺的：

PF Register(PF pf);

Register() 的参数是一个 PF 类型的[回调函数](https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0" \t "_blank)，返回某个函数的地址，其署名与先前注册的名字相同。做一次深呼吸。下面我展示一下如果不用 typedef，我们是如何实现这个声明的：

int (\*Register (int (\*pf)(const char \*, const char \*)))

(const char \*, const char \*);

很少有程序员理解它是什么意思，更不用说这种费解的代码所带来的出错风险了。显然，这里使用 typedef 不是一种特权，而是一种必需。持怀疑态度的人可能会问："OK，有人还会写这样的代码吗？"，快速浏览一下揭示 signal()函数的头文件 ，一个有同样接口的函数。注意这里Register被定义为一个函数而不是函数指针，如果要定义为函数指针应该这样写：int (\*(\*Register) (int (\*pf)(const char \*, const char \*))) (const char \*, const char \*);

typedef 和存储类[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97" \t "_blank)（storage class specifier）

这种说法是不是有点令人惊讶，typedef 就像 auto，extern，mutable，static，和 register 一样，是一个存储类关键字。这并不是说 typedef 会真正影响对象的存储特性；它只是说在语句构成上，typedef 声明看起来象 static，extern 等类型的变量声明。下面将带到第二个陷阱：

typedef register int FAST\_COUNTER; // 错误

编译通不过。问题出在你不能在声明中有多个存储类[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97)。因为符号 typedef 已经占据了存储类关键字的位置，在 typedef 声明中不能用 register（或任何其它存储类关键字）。

## 平台开发

[编辑](javascript:;)

typedef 有另外一个重要的用途，那就是定义机器无关的类型，例如，你可以定义一个叫 REAL 的浮点类型，在目标机器上它可以获得最高的精度：

typedef long double REAL;

在不支持 long double 的机器上，该 typedef 看起来会是下面这样：

typedef double REAL;

并且，在连 double 都不支持的机器上，该 typedef 看起来会是这样：

typedef float REAL;

你不用对[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81" \t "_blank)做任何修改，便可以在每一种平台上编译这个使用 REAL 类型的应用程序。唯一要改的是 typedef 本身。在大多数情况下，甚至这个微小的变动完全都可以通过奇妙的[条件编译](https://baike.baidu.com/item/%E6%9D%A1%E4%BB%B6%E7%BC%96%E8%AF%91" \t "_blank)来自动实现。不是吗? 标准库广泛地使用 typedef 来创建这样的平台无关类型：[size\_t](https://baike.baidu.com/item/size_t" \t "_blank)，ptrdiff 和 fpos\_t 就是其中的例子。此外，象 std::string 和 std::ofstream 这样的 typedef 还隐藏了长长的，难以理解的模板特化语法，例如：basic\_string，allocator> 和 basic\_ofstream>。

参考资料

* 1.谭浩强．C程序设计（第三版）：清华大学出版社，2005：260-262
* 2.Stanley B.lippman 等，李师贤等译．C++ Primer 中文版（第四版）：人民邮电出版社，2011年：106-124