**typedef与#define的区别**

**1.typedef**

typedef故名思意就是类型定义的意思，但是它并不是定义一个新的类型而是给已有的类型起一个别名，在这一点上与引用的含义类似，引用是变量或对象的别名，而typedef定义的是类型的别名。typedef的作用主要有两点：

**1.1简化复杂的类型声明**

简化复杂的类型声明，或给已有类型起一含义明确的别名；如：

(1)typedef bool (\*FuncPointer)(int, double); //声明了一个返回 bool 类型并带有两个(int和double)形参的函数的指针类型FuncPointer;

(2)FuncPointer pFunc; //声明了一个FuncPointer类型的函数指针对象pFunc;

**1.2定义与平台无关的类型**

定义与平台无关的类型，屏蔽不同平台的类型差异化；如：

用typedef来定义与平台无关的类型。

比如定义一个叫 REAL 的浮点类型，在目标平台一上，让它表示最高精度的类型为：

typedef long double REAL;

在不支持 long double 的平台二上，改为：

typedef double REAL;

在连 double 都不支持的平台三上，改为：

typedef float REAL;

也就是说，当跨平台时，只要改下 typedef 本身就行，不用对其他源码做任何修改。

标准库就广泛使用了这个技巧，比如size\_t。另外，因为typedef是定义了一种类型的新别名，不是简单的字符串替换，所以它比宏来得稳健。

**1.3与struct的结合使用**

在C++中，struct与class的作用相同，就是默认的访问权限不同，struct默认为public，而class默认为private的。

【例1.3.1】：

1. struct Person
2. {
3. string name;
4. int age;
5. float height;
6. };
7. Person person;

定义一个Struct的类型Person，定义一个Person的对象person。

或者

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. struct Person
2. {
3. string name;
4. int age;
5. float height;
6. }person;

定义一个Struct的类型Person，在定义的同时还声明了一个Person的对象person。

但是在C语言中，struct的定义和声明要用typedef。

【例1.3.2】：

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. typedef struct \_\_Person
2. {
3. string name;
4. int age;
5. float height;
6. }Person;    //这是Person是结构体的一个别名
7. Person person;

如果没有typedef就必须用struct Person person;来声明，如：

【例1.3.3】

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. struct Person
2. {
3. string name;
4. int age;
5. float height;
6. };
7. struct Person person;

或

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. struct Person
2. {
3. string name;
4. int age;
5. float height;
6. }person;    //person是Person的对象

**2.typedef与#define的区别**

**2.1执行时间不同**

关键字typedef在编译阶段有效，由于是在编译阶段，因此typedef有类型检查的功能。

#define则是宏定义，发生在预处理阶段，也就是编译之前，它只进行简单而机械的字符串替换，而不进行任何检查。

【例2.1.1】typedef会做相应的类型检查：

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. typedef unsigned int UINT;
3. void func()
4. {
5. UINT value = "abc"; // error C2440: 'initializing' : cannot convert from 'const char [4]' to 'UINT'
6. cout << value << endl;
7. }

 【例2.1.2】#define不做类型检查：

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. //#define用法例子：
2. #define f(x) x\*x
3. int main()
4. {
5. int a=6, b=2, c;
6. c=f(a) / f(b);
7. printf("%d\n", c);
8. return 0;
9. }

程序的输出结果是: 36，根本原因就在于#define只是简单的字符串替换。

**2.2功能有差异**

typedef用来定义类型的别名，定义与平台无关的数据类型，与struct的结合使用等。

#define不只是可以为类型取别名，还可以定义常量、变量、编译开关等。

**2.3作用域不同**

#define没有作用域的限制，只要是之前预定义过的宏，在以后的程序中都可以使用。

而typedef有自己的作用域。

【例2.3.1】没有作用域的限制，只要是之前预定义过就可以

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. void func1()
2. {
3. #define HW "HelloWorld";
4. }
6. void func2()
7. {
8. string str = HW;
9. cout << str << endl;
10. }

【例2.3.2】而typedef有自己的作用域

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. void func1()
2. {
3. typedef unsigned int UINT;
4. }
6. void func2()
7. {
8. UINT uValue = 5;//error C2065: 'UINT' : undeclared identifier
9. }

【例2.3.3】

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. class A
2. {
3. typedef unsigned int UINT;
4. UINT valueA;
5. A() : valueA(0){}
6. };
8. class B
9. {
10. UINT valueB;
11. //error C2146: syntax error : missing ';' before identifier 'valueB'
12. //error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
13. };

上面例子在B类中使用UINT会出错，因为UINT只在类A的作用域中。此外，在类中用typedef定义的类型别名还具有相应的访问权限，【例2.3.4】：

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. class A
2. {
3. typedef unsigned int UINT;
4. UINT valueA;
5. A() : valueA(0){}
6. };
8. void func3()
9. {
10. A::UINT i = 1;
11. // error C2248: 'A::UINT' : cannot access private typedef declared in class 'A'
12. }

而给UINT加上public访问权限后，则可编译通过。

【例2.3.5】：

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. class A
2. {
3. public:
4. typedef unsigned int UINT;
5. UINT valueA;
6. A() : valueA(0){}
7. };
9. void func3()
10. {
11. A::UINT i = 1;
12. cout << i << endl;
13. }

**2.4． 对指针的操作**

二者修饰指针类型时，作用不同。

[cpp] [view plain](http://blog.csdn.net/luoweifu/article/details/41630195) [copy](http://blog.csdn.net/luoweifu/article/details/41630195)

1. typedef int \* pint;
2. #define PINT int \*
4. int i1 = 1, i2 = 2;
6. const pint p1 = &i1;    //p不可更改，p指向的内容可以更改，相当于 int \* const p;
7. const PINT p2 = &i2;    //p可以更改，p指向的内容不能更改，相当于 const int \*p；或 int const \*p；
9. pint s1, s2;    //s1和s2都是int型指针
10. PINT s3, s4;    //相当于int \* s3，s4；只有一个是指针。
12. void TestPointer()
13. {
14. cout << "p1:" << p1 << "  \*p1:" << \*p1 << endl;
15. //p1 = &i2; //error C3892: 'p1' : you cannot assign to a variable that is const
16. \*p1 = 5;
17. cout << "p1:" << p1 << "  \*p1:" << \*p1 << endl;
19. cout << "p2:" << p2 << "  \*p2:" << \*p2 << endl;
20. //\*p2 = 10; //error C3892: 'p2' : you cannot assign to a variable that is const
21. p2 = &i1;
22. cout << "p2:" << p2 << "  \*p2:" << \*p2 << endl;
23. }

结果：

p1:00EFD094  \*p1:1

p1:00EFD094  \*p1:5

p2:00EFD098  \*p2:2

p2:00EFD094  \*p2:5