1. nullptr NULL 0

**C的NULL**

在C语言中，我们使用NULL表示空指针，也就是我们可以写如下代码：

int \*i = NULL;

foo\_t \*f = NULL;

实际上在C语言中，NULL通常被定义为如下：

#define NULL ((void \*)0)

也就是说NULL实际上是一个void \*的指针，然后吧void \*指针赋值给int \*和foo\_t \*的指针的时候，隐式转换成相应的类型。

而如果换做一个C++编译器来编译的话是要出错的，因为C++是强类型的，void \*是不能隐式转换成其他指针类型的，所以通常情况下，编译器提供的头文件会这样定义NULL：

#ifdef \_\_cplusplus ---简称：cpp c++ 文件

#define NULL 0

#else

#define NULL ((void \*)0)

#endif

C++的0

因为C++中不能将void \*类型的指针隐式转换成其他指针类型，而又为了解决空指针的问题，所以C++中引入0来表示空指针（注：0表示，还是有缺陷不完美），这样就有了类似上面的代码来定义NULL。实际上C++的书都会推荐说C++中更习惯使用0来表示空指针而不是NULL，尽管NULL在C++编译器下就是0。主要由于在C++中，使用0来做为空指针会比使用NULL来做空指针会让你更加警觉。（你会清楚可能0会被当作int或void\*，但NULL不会给你这种警觉）

C++的nullptr

如果使用 nullptr 初始化对象，就能避免 0 指针的二义性的问题（传参数的时候）。

1. constexpr

将变量声明为constexpr类型以便由编译器来验证变量的值是否是一个常量表达式；

声明为constexpr的变量一定是一个常量，而且必须用常量表达式来初始化

1. using typedef

using a=double;

typedef double a;

using func=int(int);

typedef int func(int);

// 函数指针，指向返回值类型为int，参数类型为int的函数

typedef int (\*pfunc)(int);

// 函数引用，指向返回值类型为int，参数类型为int的函数

typedef int (&tfunc)(int);

using pfunc2 = int(\*)(int);

1. auto让编译器通过初始值来推算变量的类型，所以，其定义的变量必须要有初始值；

auto让编译器通过初始值来推算变量的类型，所以，其定义的变量必须要有初始值；

使用auto也能在一条语句中声明多个变量；因为一条声明语句只能有一个基本数据类型，所以该语句中所有的变量的初始基本数据类型都必须是一样的；

Auto会忽略掉顶层const，保留底层const

顶层const：指针、变量本身是一个常量；

底层const：指针指向的对象是一个常量；

int main() {

int i = 0;

const int ci = i;

auto a=ci;//a是int

auto b = &i; // b是一个整形指针（整数的地址就是指向整数的指针）

auto c = &ci; // c是一个指向整数常量的指针（对常量对象取地址是一种底层const）

return 0;

}

1. decltype

const int ci = 0, &cj = ci;

decltype(ci) x = 0;//const int x=0

decltype(cj) y = x;//const int &y=x;

decltype(cj) z; //报错，因为cj是一个引用，因此作为引用的 z 必须要进行初始化

引用从来都是作为其所指对象的同义词出现，也就是根据其所指的对象决定，只有在decltype处是个例外