1 笔记

2 第二章

1. 'void*' 是一种特殊的指针类，可以存放任意对象的地址。

2. const 对象必须初始化。const int buf =10

3. const 指针，其值不能被改变，所有必须初始化。
4. 顶层 const 和底层 const，顶层 const 表示指针本身是一个常量，底层 const 表示指针所指对象是一个常量。更一般的，顶层 const 可以表示任意的对象是常量，这一点对任何数据类型都适用。底层 const 则与指针和引用等复合类型的基本类型部分有关。比较特殊的是，指针类型既可以是顶层 const 也可以是底层 const。

```
int i = 0;
int *const p1 = &i;     // 不能改变p1的值，p1是一个顶层const
const int ci = 42;      // 不能改变ci的值，ci是一个顶层const
const int *p2 = &ci;    // 允许改变p2的值，p2是一个底层const
const int *const p3 = p2; // 靠右的const是顶层const，靠左的是底层const
const int &r = ci;      // 用于声明引用的const都是底层const
```

C++ 新标准引入了第二种类型说明符 **decltype**，它的作用是选择并返回操作数的数据类型。在此过程中，编译器分析表达式并得到它的类型，却不实际计算表达式的值：

```
decltype(f()) sum = x; // sum的类型就是函数f的返回类型
```

编译器并不实际调用 f，而是使用当调用发生时 f 的返回值的类型作为 sum 的类型。

如果 decltype 使用的表达式是一个变量，则 decltype 返回该变量的类型（包括顶层 const 和引用在内）：

```
const int ci = 0, &cj = ci;
decltype(ci) x = 0; // x的类型是const int
decltype(cj) y = x; // y的类型是const int&，y绑定到x
```

decltype 和引用

如果 decltype 使用的表达式不是一个变量，则 decltype 返回表达式结果对应的类型。如果表达式向 decltype 返回一个引用类型，一般来说，意味着该表达式的结果对象能作为一条赋值语句的左值：

```
// decltype的结果可以是引用类型
int i = 42, *p = &i, &r = i;
decltype(r + 0) b; // 正确，加法的结果是int，因此b是一个int
decltype(*p) c; // 错误，c是int&，必须初始化
```

如果表达式的内容是解引用操作，则 decltype 将得到引用类型。

有一种情况需要特别注意：对于 decltype 所用的表达式来说，如果变量名加上了一对括号，编译器就会把它当成一个表达式。变量是一种可以作为赋值语句左值的特殊表达式，所以这样的 decltype 就会得到引用类型：

```
// decltype的表达式如果是加上了括号的变量，结果是引用
decltype((i)) d; // 错误，d是int&，必须初始化
decltype(i) e; // 正确，e是一个int。
```

auto 和 decltype 的区别主要有三个方面：第一，auto 类型说明符用编译器计算变量的初始值来推断其类型，而 decltype 虽然也让编译器分析表达式并得到它的类型，但是不实际计算表达式的值。第二，编译器推断出来的 auto 类型有时候和初始值的类型并不完全一样，编译器会适当地改变结果类型使其更符合初始化规则。例如，auto 一般会忽略掉顶层 const，而把底层 const 保留下来。与之相反，decltype 会保留变量的顶层 const。第三，与 auto 不同，decltype 的结果类型与表达式形式密切相关，如果变量名加上了一对括号，则得到的类型与不加括号时会有不同。如果 decltype 使用的是一个不加括号的变量，则得到的结果就是该变量的类型；如果给变量加上了一层或多层括号，则编译器将推断得到引用类型。