9 the `const` keyword and compound objects

9.1 `const` as type qualifier

这一节非常逆天，很绕.

```
void copy(int dst[], const int src[], int n) {
     for (int i = 0; i < n; ++i) {
         src[i] = dst[i];
         //error, 因为 scr[]被添加了const
     }
}
```

`const` 是一个 **type qualifier**.

const forbids assignment.
(仅支持 Initialization)

```
const int x = 3; // ok
x = 5; // won't compile
```

9.1.1 `const` pointer

带 `const` 的指针类型变量和其他变量一样，就是值在 initialization 之后就不能变了. 也就是**它装的 address 不能变.**

```
int x = 3; int y = 5;
int * const ptr = &x;
ptr = &y; // ERROR
```

但是它装的 address 下的变量是可以变的.

```
int x = 3; int y = 5;
int * const ptr = &x;
*ptr = 10 // ok, x = 10
```

9.1.2 pointer-to-`const`

`const int * ptr = &x;` 或 `int const * ptr = &x` 的意思是：添加一个 指针 `ptr` 指向 `x`，并且这个指向的行为是 `const` 的.

这并不代表 `x` 变成了 `const` 变量，而这个 `ptr` 也并不是 `const` 指针. **这里的意思是 `ptr` 不能用来改变 `x` 的值.**

因而你还是可以改变 `x` 的值，但是不能用 `ptr` 改变.

```
int x = 3;
int const * ptr = &x; //或者int const * ptr = &x;
*ptr = 10; // ERROR
x = 10; // ok, *ptr现在值为10
```

并且不仅如此，这个 pointer to `const` 和 `const` pointer 还有一个区别是，**pointer to `const` 可以改绑其他 object 的 address.**

```
int x = 3;
int y = 6;
const int * ptr = &x; //或者int const * ptr = &x;

*ptr = 10; // ERROR
x = 10; // ok, *ptr现在值为10

ptr = &y; //ok, ptr现在存的是y的地址
*ptr = 10; // ERROR
y = 10; // ok, *ptr现在值为10
```

写法的区别是 `const` pointer 的 `const` 在 `*` 后面，直接在变量名前面.

还有一点是：如果你要指向一个 const variable，那么必须使用 ptr-to-const!

总结：`ptr` 和 `x` 都不是 `const`，但是 `ptr` 对 `x` 的行为是 `const` 的：你既可以改变 `ptr` 的值又可以改变 `x` 的值，但是你不能通过解引用 `ptr` 来改变 `x` 的值.

9.1.3 reference to `const` variables

`const` reference variable 不能用来 change object 的值，但是仍然可以用这个 object 的非 `const` variable 改变原来的 object 的值.

```
int x = 3;
int const &ref = x; // ref是x的alias
ref = 10; // error
x = 10; // still legal
```

9.1.4 `const` conversions

```
int x = 3;
int const *cptr = &x; //ptr-to-const
const int *dptr = &x; //ptr-to-const, 一样
int * const eptr = &x; //const ptr
int *ptr1 = cptr; // error
int *ptr2 = dptr; // error
int *ptr2 = fptr; // error
// 一个都不行
```

这里我们尝试把一个 pointer-to-const 的值传给一个普通的 pointer，不行.

我们尝试把一个 const pointer 的值传给一个 普通的 pointer，也不行.

注意到，我们可以把一个 const int 的值传给一个普通 int，这是可以的，但是指针完全不行.

这是因为指针的类型不仅包含了它所指向的数据类型，还包含了对这些数据的访问和修改权限.

9.1.5 例子

```
int x = 3;
int const y = 3; // const变量
int const * ptr1 = &x; // ptr-to-const, 指向x
int * const ptr2 = &x; // const ptr, 指向x
int const &ref = x;

y = 5; //error
*ptr1 = 5; //error
ptr1 = &y; //ok, ptr-to-const可以改绑
*ptr2 = 5; //ok, const ptr可以调整指向对象值
ptr2 = &x; //error, const ptr不可以改绑
ref = 5; //error, ref-to-const不可以调整值
x = 5; //ok, x不是const
```

9.1.6 `const` in function

9.1.6.1 `const` parameter

当 function 中一个 parameter 被声明为 `const`，这意味着它在函数内部不能被修改. 这对于 ref 和 ptr parameters 尤其重要，因为它们可以用来修改传递给函数的原始数据。声明参数为 `const` 可以保证安全性.

```
void function(const int x) {
    // x 是 const, 所以不能被修改
}
```

```
void function(const int *ptr) {
    // ptr-to-const, 不能改变指向对象值
}
```

```
void function(int * const ptr) {
    //const ptr, 指向的地址不能改变, 可以改变指向对象值
}
```

9.1.6.2 return `const`

```
const int getNumber() {
    return 5;
}
```

```
const int& getNumber(const int& x) {
    return x;
}
```

表示返回的值或ref 不能被修改.

9.1.6.3 把普通 ptr 传给参数是 const ptr 或 ptr-to-const 的函数

1. **将普通指针传递给要求 pointer-to-const 的函数**:

- 将一个指向非 `const` 数据的指针传递给一个接受指向 `const` 数据的指针的参数时，在这个函数内部，不会通过这个指针修改数据. 但是原指针可以.

```
void func(const int *ptr) {
    // 在这里，*ptr 不能被修改
}

int main() {
    int x = 10;
    int *ptr = &x;
    func(ptr);  // 合法：将 int* 转换为 const int*
    return 0;
}
```

2. **将普通指针传递给要求 const pointer 的函数**:

   这意味着函数内部不能改变指针的值，但是原指针可以.

```
void func(int *const ptr) {
    // ptr 本身不能被改变，但可以通过 ptr 修改所指向的数据
}

int main() {
    int x = 10;
    int *ptr = &x;
    func(ptr);  // 合法
    return 0;
}
```

9.1.6.4 把 const variable 传给参数不是 const 的函数：error

如果一个变量是 const，那么

```
void teal(const int *a);
void red(int *a);
void purple(int a);

int main() {
  const int arr1[6] = { ... };
  int arr2[6] = { ... };

  teal(arr1); // ok, const to const
  red(arr1); // error, const 变量传给非 const 函数参数
  purple(arr1[0]); //ok. 因为只传array的一个值是值传递，如果传array自身就是引用传递第一个元素地址.

  teal(arr2); // ok
  red(arr2); //ok
  purple(arr2[0]); //ok
}
```