

6.10 — Pointers and const

BY ALEX ON JULY 16TH, 2007 | LAST MODIFIED BY ALEX ON AUGUST 28TH, 2017

Pointing to const variables

So far, all of the pointers you've seen are non-const pointers to non-const values:

```
1 int value = 5;
2 int *ptr = &value;
3 *ptr = 6; // change value to 6
```

However, what happens if value is const?

```
1 const int value = 5; // value is const
2 int *ptr = &value; // compile error: cannot convert const int* to int*
3 *ptr = 6; // change value to 6
```

The above snippet won't compile -- we can't set a non-const pointer to a const variable. This makes sense: a const variable is one whose value can not be changed. Hypothetically, if we could set a non-const pointer to a const value, then we would be able to dereference the non-const pointer and change the value. That would violate the intention of const.

Pointer to const value

A **pointer to a const value** is a (non-const) pointer that points to a constant value.

To declare a pointer to a const value, use the *const* keyword before the data type:

```
1 const int value = 5;
2 const int *ptr = &value; // this is okay, ptr is a non-const pointer that is pointing to a "const int"
3 *ptr = 6; // not allowed, we can't change a const value
```

In the above example, ptr points to a const int.

So far, so good, right? Now consider the following example:

```
1 int value = 5; // value is not constant
2 const int *ptr = &value; // this is still okay
```

A pointer to a constant variable can point to a non-constant variable (such as variable value in the example above). Think of it this way: a pointer to a constant variable treats the variable as constant when it is accessed through the pointer, regardless of whether the variable was initially defined as const or not.

Thus, the following is okay:

```
1 int value = 5;
2 const int *ptr = &value; // ptr points to a "const int"
3 value = 6; // the value is non-const when accessed through a non-const identifier
```

But the following is not:

```
1 int value = 5;
2 const int *ptr = &value; // ptr points to a "const int"
3 *ptr = 6; // ptr treats its value as const, so changing the value through ptr is not legal
```

Because a pointer to a const value is not const itself (it just points to a const value), the pointer can be redirected to point at other values:

```
1 int value1 = 5;
2 const int *ptr = &value1; // ptr points to a const int
3
4 int value2 = 6;
5 ptr = &value2; // okay, ptr now points at some other const int
```

Const pointers

We can also make a pointer itself constant. A **const pointer** is a pointer whose value can not be changed after initialization

To declare a const pointer, use the `const` keyword between the asterisk and the pointer name:

```
1 int value = 5;
2 int *const ptr = &value;
```

Just like a normal const variable, a const pointer must be initialized to a value upon declaration. This means a const pointer will always point to the same address. In the above case, `ptr` will always point to the address of `value` (until `ptr` goes out of scope and is destroyed).

```
1 int value1 = 5;
2 int value2 = 6;
3
4 int * const ptr = &value1; // okay, the const pointer is initialized to the address of value1
5 ptr = &value2; // not okay, once initialized, a const pointer can not be changed.
```

However, because the *value* being pointed to is still non-const, it is possible to change the value being pointed to via dereferencing the const pointer:

```
1 int value = 5;
2 int *const ptr = &value; // ptr will always point to value
3 *ptr = 6; // allowed, since ptr points to a non-const int
```

Const pointer to a const value

Finally, it is possible to declare a const pointer to a const value by using the `const` keyword both before the type and before the variable name:

```
1 int value = 5;
2 const int *const ptr = &value;
```

A const pointer to a const value can not be set to point to another address, nor can the value it is pointing to be changed through the pointer.

Recapping

To summarize, you only need to remember 4 rules, and they are pretty logical:

- A non-const pointer can be redirected to point to other addresses.
- A const pointer always points to the same address, and this address can not be changed.
- A pointer to a non-const value can change the value it is pointing to. These can not point to a const value.
- A pointer to a const value treats the value as const (even if it is not), and thus can not change the value it is pointing to.

Keeping the declaration syntax straight can be challenging. Just remember that the type of value the pointer points to is always on the far left:

```
1 int value = 5;
2 const int *ptr1 = &value; // ptr1 points to a "const int", so this is a pointer to a const value.
3 int *const ptr2 = &value; // ptr2 points to an "int", so this is a const pointer to a non-const value.
4 const int *const ptr3 = &value; // ptr3 points to a "const int", so this is a const pointer to a const value.
```

Conclusion

Pointers to const values are primarily used in function parameters (for example, when passing an array to a function) to help ensure the function doesn't inadvertently change the passed in argument. We will discuss this further in the section on functions.



6.11 -- Reference variables



Index