LEARN C++

Skill up with our free tutorials

10.15 — Pointers and const

ALEX AUGUST 25, 2021

Pointing to const variables

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

```
int value{ 5 };
int* ptr{ &value };
*ptr = 6; // change value to 6
```

However, what happens if value is const?

```
const int value{ 5 }; // value is const
int* ptr{ &value }; // compile error: cannot convert const int* to int*
*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 perform indirection through 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:

```
const int value{ 5 };
const int* ptr{ &value }; // this is okay, ptr is a non-const pointer that is pointing to a "const int"
*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:

```cpp
int value{ 5 }; // value is not constant
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:

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

But the following is not:

```cpp
int value{ 5 };
const int* ptr{ &value }; // ptr points to a "const int"
*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:

```cpp
int value1{ 5 };
const int* ptr{ &value1 }; // ptr points to a const int

int value2{ 6 };
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:

```cpp
int value{ 5 };
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).

```cpp
int value1{ 5 };
int value2{ 6 };

int* const ptr{ &value1 }; // okay, the const pointer is initialized to the address of value1
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 indrectly through the const pointer:

```
int value{ 5 };
int* const ptr{ &value }; // ptr will always point to value
*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:

```
int value{ 5 };
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:

```cpp
int value{ 5 };
const int* ptr1{ &value }; // ptr1 points to a "const int", so this is a pointer to a const value.
int* const ptr2{ &value }; // ptr2 points to an "int", so this is a const pointer to a non-const value.
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.

Next lesson
10.16 Reference variables

Back to table of contents

Previous lesson
10.14 Dynamically allocating arrays

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ co

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: 🔔 POST COMMENT

DP N N FOUT

Newest ▾

©2021 Learn C++

X

X