【C++】Day six(2)

Class	C++
Date	@November 23, 2021
Material	
Series Number	
Summary	

【Ch2】Const Continue

2.4.2 Pointers and const

A pointer to `const` may not be used to change the object to which the pointer points.
We store the address of a `const` object only in a pointer to `const`:

```
const double pi = 3.14;
double *ptr = &pi; //error: ptr is a plain pointer
const double *cptr = &pi; //ok: cptr may point to a double that is const
*cptr = 42; //error: cannot assign to *cptr
```

We can use a pointer to `const` to point to a `non-const` object:

```
double dval = 3.14;
const double *cptr = &dval;
```

A pointer to `const` says nothing about whether the object to which the pointer points is const.

Defining a pointer as pointer to `const` affects only what we can do with the pointer.

const Pointers

Pointers are objects. Hence, we can have a pointer that is itself const.

A `const` pointer must be initialized, and once initialized its value may not be changed.

```
int num = 0;
int *const ptr = &num; //ptr will always point to num
const double pi = 3.14;
const double *const cptr = &pi; //cptr is a const pointer to a const object
```

2.4.3 Top-Level const

- Top-level const: An object is itself a const
- Low-level const: Appear in the base type of compound types such as pointers or references.

The distinction between top-level and low-level matters when we copy an object. When we copy an object, top-level consts are ignored.

```
int i = 0;
const int ci = 42;
i = ci; //ok: copying the value of ci; top-level const is ignored.
```

On the other hand, low-level const is never ignored. When we copy an object, both objects must have the same low-level const qualification or there must be a conversion between the type of the two objects.

```
const int *const p3 = &i;
int *p = p3; //error: p3 has a low-level const but p doesn't
const int *p2;
p2 = p3; // ok: p2 has the same low-lvel const qualification as p3
```