# Pointers & const

**Pointers have two values: an indirect and an explicit (or direct) value. Either (or both) may be `const`**. Consider this code.

```
string a{"A"}, b{"B"}, c{"C"};
const string *ps1 = &a;
string * const ps2 = &b;
const string * const ps3 = &c;
```

Note the word **`const`** in the declaration of **`p1`**, **`p2`** and **`p3`**.

- Prevent **writing to the pointer's indirect value**, by putting the const **before the type** (**`ps1`**). Thus `*ps1 = "x";` is illegal.
- When **`const`** comes **after the star**, (**`ps2`**), it means that the pointer itself cannot be changed; you **cannot make it point to a different location**. It would be illegal to write `ps2 = &a;`
- Prevent changing either the pointer, **or** what it points to, by using **both** versions of **`const`** (**`ps3`**).

When you write a function which **should not** change the item that it points to, make sure you define it as a "pointer to **`const`**. For instance, consider this template function which prints both the address and value of any variable:

```
template <typename T>
void printData(const T* p)
{
    cout << "Pointer: " << p << "->";
    if (p != nullptr) cout << *p << endl;
    else cout << "nullptr" << endl;
}
```

Because the parameter is a "pointer to **`const`**, you can pass the function **both `const`** and non-**`const`** variables. It works with everything because it **guarantees** that it won't inadvertently change the object that p points to.