Core C++ 2025

19 Oct. 2025 :: Tel-Aviv

`const` correctly in C++

Alex Cohn

Priority

About myself: @sasha_cohn

- Software Craftsman with C++ as mother tongue
- Part of the **Core C++** community since the early days

- Enhancing core performance at ***Priority Software*** and migrating C++ of 1980s to the cloud

Agenda

- Fundamentals of **`const`**
- Making **`const`** More Flexible
- **`const`** in API Design & Member Functions
- Shallow vs. Deep **`const`**
- Best Practices
- Questions

const correctly 3

1. What is **const** in C ? values and pointers

```
int i = 42;
i = 12;
const int ci = 42;
ci = 12;
int *pi = &i;
*pi = 42;
int *pp = &ci;
const int *pci = &ci;
int k = *pci;
*pci = 12;
```

const correctly 4

1. Top-level *vs* low-level **`const`**

Variable: cannot change after initialization

```
char *const pc = s[0];
```

Variable type: cannot touch the memory it points to

```
const char *cp = s[0];
```

Both:

```
const char *const cpc = s[0];
```

1. Top-level *vs* low-level **`const`**

```
int a[] {4,2,3};
const int *p1 = a;
p1++;
*p1 = 5;
int const *p2 = a;
p2++;
*p2 = 5;
```

https://godbolt.org/z/3qxqx8zd6

```
int *const p3 = a;
p3++;
*p3 = 5;
int const *const p4 = a;
p4++;
*p4 = 5;

a++;  // behaves similar to p3
*a = 5;
```

1. What is **const** in C++? object

```
struct S {
  int i = 12;
};

const S s;
s.i = 10;
```

https://godbolt.org/z/Eer88oWa3

1. What is **const** in C++? member var

```
struct S {
   const int i = 12;
} s;

s.i = 10;
```

https://godbolt.org/z/br39axd3f

1. What is **const** in C++? initialize

```
struct S {
  const int i = 12; // similar to readonly in C#
};

S s {.i = 10}; // OK
```

1. What is **const** in C++? initialize

```
struct S {
  int i = 12;
};

const S s {.i = 10}; // OK
```

https://godbolt.org/z/9T9K9Mf96

1. What is **const** in C++? method

```
struct S {
  int i = 12;
  int get() { return i; } // must be explicit
};

const S s;
s.get();
```

https://godbolt.org/z/zc646185z

1. What is **const** in C++? method

```
struct S {
  int i = 12;
  int get() const { return i; }
};

const S s;
s.get(); // OK
```

https://godbolt.org/z/33z3jreeq

1. What is **const** in C++? parameters

```
int foo_bad(const int n) {
  return n++;
}

int foo_good(const int n) {
  return n+1;
}
```

https://godbolt.org/z/bacz7ajGK

2. What is **const_cast** ?

```
int i = 42;
const int *cp = &i;
*cp = 10;
int *p = const_cast<int *>(cp);
int *bad_p = (int *)cp; // -Wold-style-cast
*p = 10;
```

2. Be careful with **const_cast**

```
int i = 42;
const int *cp = &i;
int *p = const_cast<int *>(cp); // OK
*p = 10;
cout << i << endl; // Output: 10
```

https://godbolt.org/z/EW1zbKaca

2. Be careful with **const_cast**

```
const int i = 42;
const int *cp = &i;
int *p = const_cast<int *>(cp); // UB
*p = 10;
cout << i << endl; // Output: 42
```

https://godbolt.org/z/65Yr9vfWK

2. Be careful with **const_cast**

const_cast makes it possible to form a reference or pointer to non-const type that is actually referring to a const object or a reference or pointer to non-volatile type that is actually referring to a volatile object. Modifying a **const** object through a non-**const** access path and referring to a volatile object through a non-volatile glvalue results in **undefined behavior**.

const_cast conversion - cppreference.com

2. **mutable** member

```
 struct S {
   mutable mutex m;
   int i;
   int get() const {
     lock_guard g(m);
     return i;
  }
}; // https://godbolt.org/z/T1Movhhe5
```

```
const S s { .i = 4; }
cout << s.get() << endl;
```

3. **const** method: promise to the caller

```
struct S {
  int i = 1;
  int get() { return ++i; }
  int get() const { return ++i; }
};
```

https://godbolt.org/z/hjT7brMG4

3. **const** polymorphism

```
struct S {
  int i = 12;
  int get() const { return i; }
  int get() { return ++i; }
};
S s;
const S &sr = s;
s.get(); s.get(); s.get(); // ⇒ 13; 14; 15
sr.get(); // ⇒ 15
```

https://godbolt.org/z/fhG17TEc3

3. polymorphism: no overload on value

```
struct S {
  int get(int k) { return k; }
  int get(const int k) { return k; }
};
```

https://godbolt.org/z/ax99zMKKE

3. polymorphism: overload on reference

```
struct S {
  int get(int &k) { return k; }
  double get(const int &k) { return 1.0*k; }
};
S s;
int k = 2;
s.get(k);
const int n = 1;
s.get(n);
```

https://godbolt.org/z/KfrM9911M

3. **const** char* in execv()

```
int execv(const char *path, char *const argv[]);
```

Can I pass an array of const char pointers as the second argument?

stackoverflow.com: can-i-pass-a-const-char-array-to-execv

Yes, you can! But not for win32 CreateProcess().

3. surprise of std::map and std::string

`const map<int, int> m {{1,3},{2,1},{5,4}};`

`cout << m[2] << endl;` ✗ doesn't compile

`cout << m.at(3) << endl;` ✓ throws

But

`const string s {"abcd"};`

`cout << s[4] << endl;` ✗ works

`cout << s.at(4) << endl;` ✓ throws

https://godbolt.org/z/c5TEGdqY3

3. std:: iterators and containers

`vector::begin()`	*may be mutable*
`vector::cbegin()`	**const**
`vector::operator[]()`	*may be mutable*
`string::c_str()`	**const**
`string::data()`	*may be mutable*
`string_view::data()`	**const**

3. copy optimization: const ref parameter

Pass by value (copy is created)

```
void byValue(std::string str) {
    cout << "By value: " << str << endl;
}
```

Pass by const reference (no copy)

```
void byConstReference(const std::string& str) {
    cout << "By const reference: " << str << endl;
}
```

https://godbolt.org/z/PnP5feYa8

4. Composition challenge

```
struct Object {     int val = 0;    };
struct Container {
    Container() : m_Object(new Object()) { }
    void set(int i) const { m_Object->val = i; }
    int get() const { return m_Object->val; }
private:
    Object *m_Object;
}; // https://godbolt.org/z/ojPj9qzfd or https://godbolt.org/z/MMcYeGbb1
```

4. Composition challenge

```
struct Object {     int val = 0;   };
struct Container {
    Container() : m_Object(new Object()) { }
    void set(int i) const { m_Object->val = i; }
    int get() const { return m_Object->val; }
private:
    const Object *m_Object;
}; // https://godbolt.org/z/ojPj9qzfd or https://godbolt.org/z/MMcYeGbb1
```

4. Composition challenge, solution

The answer: const propagation
std::experimental::propagate_const - cppreference.com

Not available on MSVC yet.

```
std::experimental::propagate_const<Object *> m_Object;
```

https://godbolt.org/z/49KzeY64b

4. Composition challenge, how ->()

Minimal demo implementation:

```
template <class T> struct propagate_const {
    T* operator->() { return m_ptr; }
    const T* operator->() const { return m_ptr; }
  private:
    T *m_ptr;
};
```

https://godbolt.org/z/7jqqEaThr

5. Summary

- Don’t use C-cast, especially not instead of **const_cast**
- Don’t use **const_cast** when it’s undefined behavior
- Use **const** reference in your API to avoid copy
- Mark relevant methods **const**
- Use **mutable** for class members that are not exposed
- Remember the pitfalls of containers

5. What we *did not* talk about?

- Loop as `for(const auto &[_, value]: some_map)`
- C and C++ have immutable `enum` and `#define`
- `constexpr` functions
- `constexpr` vs `consteval`
- Other languages have `final` or `let` or `readonly`
- Other languages have `const` for compile-time constants
- Rust with `mut`

THANK YOU

Alex Cohn
@sasha_cohn

Priority