

Const

- Basic Rules of `const` in C++
 - 1. Constant Variables
 - 2. Function Parameters (Const References)
 - 3. Const Member Functions
 - 4. Const Objects
 - 5. Const in Function Return Types
- 🎯 Quiz!

Below is a GERNERALIZED list of rules when it comes to `const`. From this point on in your labs you should be thinking about what needs to be `const` and why -- this rule set is here to help. For the most part in the first 2 weeks you will be adding `const` where needed in the parameters of your methods as a hint.

Basic Rules of `const` in C++

1. Constant Variables

A `const` variable is a variable that cannot be modified. Its value cannot change, and it is what we call in programming **immutable**. If you see the word immutable, we are usually referring to something being protected from change using the `const` keyword.

Example:

```
const int x = 5;
```

In this example, `x` cannot be modified after it is initialized. `x` must be initialized at the time of declaration due to it being `const`.

2. Function Parameters (Const References)

A `const &` (const reference) parameter means that the function receives a reference to the original data without copying it, and promises not to modify it. This technique reduces memory usage and improves performance while also ensuring data safety.

Example:

```
void print(const std::string& s);
```

In this example, we have a function declaration named `print` that takes in a `const &` of a `std::string`. This `string` is passed by reference (so it avoids copying), and marked as `const` because the function doesn't need to change it.

3. Const Member Functions

A `const` member function is a function that does not modify the internal state of the object. It is indicated by placing the `const` keyword after the function signature. Only `const` member functions can be called on `const` objects.

Example:

```
class MyClass {  
public:  
    int getValue() const; //cannot modify the "price" field of the class.  
private:  
    double price;  
};
```

In this example, `getValue` is a const member function, which means it cannot modify any member variables of the object (unless they are marked `mutable`).

4. Const Objects

A `const` object is one declared with the `const` keyword, meaning it cannot be modified after creation. Only `const` member functions can be called on such objects.

Example:

```
const MyClass obj;
```

In this example, we have declared a `const` object of type `MyClass`. This means that `obj` cannot be modified, and only member functions marked as `const` can be called on it (like `getValue()` in the previous example).

5. Const in Function Return Types

Returning `const` by **value** is rarely useful and often avoided, because the returned copy is still modifiable by the caller.

Example:

```
const int getVal(); // Usually discouraged
```

Returning `const` by **reference** is more useful. It avoids copying large objects and ensures the returned reference cannot be used to modify the original data.

Example:

```
const std::string& getName() const;
```

In this example, the function returns a `const` reference to a string, meaning the caller can access the string but not change it.

🎯 Quiz!

Here's a short quiz on the topic: [quiz](#)

Footer Separator

Markdown Viewer

How to view the markdown files in a browser...

- [Markdown Viewer](#)

Lecture Practices

Here are the lecture Practices...

- [Day 1](#)
- [Day 2](#)
- [Day 3](#)

Lecture Quizzes

Here are the lecture quizzes...

- [Day 1](#)
- [Day 2](#)
- [Day 3](#)

⚙️ Weekly Topics

Here are the topics for the week...

- [Calling Methods](#)
- [Calling Methods 2](#)
- [Creating Methods](#)
- [Iterators](#)
- [Vectors](#)
- [References](#)
- [Const](#)
- [Erasing in a Loop](#)
- [Default Parameters](#)