# Const-Guideline

Before you declare a variable, think about whether its value will be changed later or not!

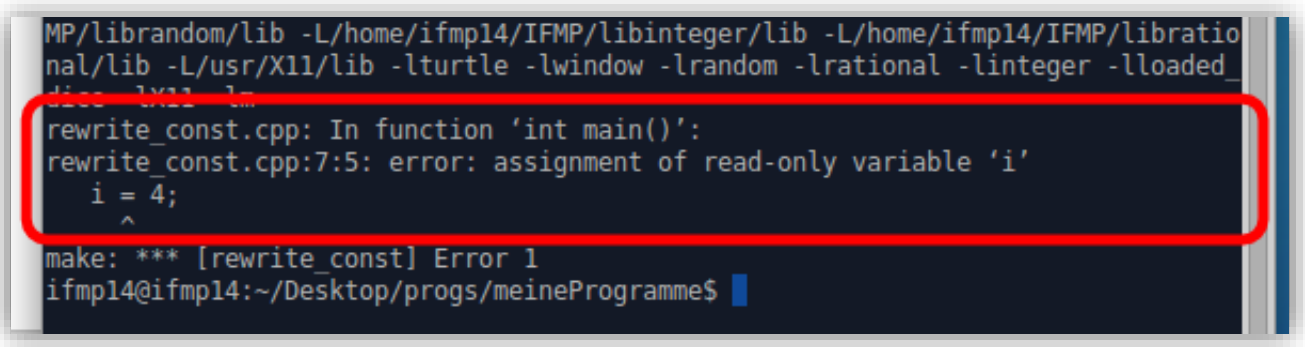
If not, use the keyword `const` to declare the variable as constant.

# `const` – Necessary?

- Protects against unintended changes

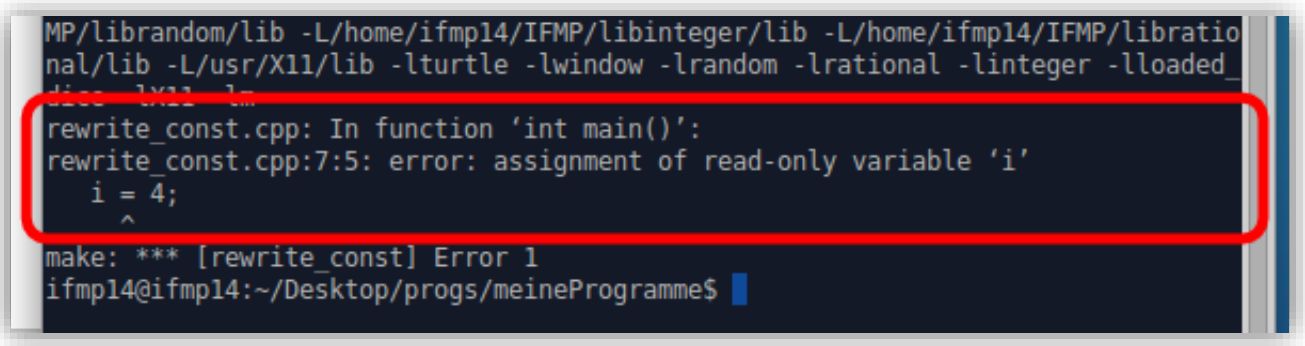
# `const` – Necessary?

- Protects against unintended changes
  - Compiler error message



# `const` – Necessary?

- Protects against unintended changes
  - Compiler error message




- Communicate to reader
  - Reader knows: value will not change

# Example

Make this `const`-correct.

1. Program:

```
#include <iostream>
int main ()
{
    const int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```

(From: Script Exercise 5)

# Example

**Problem:**

input operator >> changes constant variable

## 1. Program:

```
#include <iostream>
int main ()
{
    const int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```

## Solution:

```
#include <iostream>
int main ()
{
    int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```

(From: Script Exercise 5)

# Example

Make this `const`-correct.

2. Program:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    int c = 2*b;
    b = b*b;

    return 0;
}
```

(From: Script Exercise 5)

# Example

**Problem:**

- `c` should be `const`.
- `c` is initialized without a later use.

## 2. Program:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    int c = 2*b;
    b = b*b;

    return 0;
}
```

## Solution:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    const int c = 2*b;
    b = b*b;

    return 0;
}
```

(From: Script Exercise 5)

# Example

Make this `const`-correct.

3. Program:

```
int main ()
{
    const int a = 5;
    a = 5;

    return 0;
}
```

(From: Script Exercise 5)

# Example

**Problem:**

`a = 5;` overwrites `a` with **same** value.

But `a` is `const`; `const` prevails.

3. Program:

```
int main ()
{
    const int a = 5;
    a = 5;

    return 0;
}
```

Solution:

Remove `const` or

`a = 5;`