13. Const correctness

Ján Dugáček

February 10, 2019

Table of Contents

1. Motivation
2. Const variables
   - Const variables
   - Const objects
   - Violating constness
   - Remarks
3. Exercise
4. Homework

Motivation

- If a variable isn't intended to be changed, it's better to mark it so in order to prevent someone else who edits the code from changing it

```
int doStuff(const int x) {
//...
x = x % 12; // Error, x is const
//...
return x + result;
      // We can be certain this is the original value
}
```

- Large objects can be passed as reference (`std::string&`) to prevent unnecessary copies and a const reference (`const std::string&`) ensures it's not edited inside the function

Const variables

```
const float a = 42;
const unsigned int b = 9;
std::pair<const int, float> c{ 3, 5};
c.first = 2; // Error, the first is const
c.second = 2.0; // The second is not const
const std::pair<int, float> d{ 3, 5};
d.second = 15.17; // Error: The whole object is const
std::pair<int, float> e = d;
        // Const can be copied into non-const
std::pair<int, float>& d = e;
        // Error: Cannot make non-const reference to const
```

- There is no way to accidentally change something that is const
- Anything can be converted into const

Const objects

```
const std::string text("Free beer");
int found = text.find(" "); // Okay, string not modified
std::string beer = text.substr(found + 1);
text[found] = '_'; // Error, string is modified
```

- How does the compiler know which methods modify the object and which ones don't?

Const objects #2

```
struct importantData {
        int precious;
        int& getPrecious() const {
                precious = 2;
                        // Error, we are in a const object
                std::cout << precious << std::endl; // Okay
                return precious;
                 // Error, turning into a nonconst reference
        }
};
```

- Methods can be declared `const`, which allows them to be called if the object is const, but all member variables are const inside them

Const objects #3

```
class importantData {
        int precious;
public:
        int getPrecious() const {
                return precious;
        } // Returned value is a copy
        int& getPrecious() {
                return precious;
        } // Editing the return value edits the member
        void setPrecious(int set) {
                precious = set;
        }
};
```

- There can be const and nonconst versions of functions chosen accordingly to the constness of the object

Violating constness

```
class importantData {
        int precious;
        mutable int imprecious;
public:
        int getPrecious() const {
                imprecious = precious - 1; // Okay, mutable
                return precious;
        }
};
```

- Some objects hide some internal attributes that don't affect their external behaviour but serve for some other purposes, like caching, synchronising and other stuff, these attributes have to be marked with keyword `mutable` (use it only if this is the case)

Utterly violating constness

```
class importantData {
        int precious;
        mutable int imprecious;
public:
        int getPrecious() const {
                imprecious = precious - 1;
                return precious;
        }
        void specialDebugFunction(int set) const {
                const_cast<int>(precious) = set;
        }
};
```

- You can modify a const variable by removing its constness using `const_cast`, but this is needed very rarely

Remarks

- Constness doesn't serve to limit you, its purpose is to protect yourself from making mistakes
- You should use it to make sure you don't break your code when you return to it a year later and protect others from using your code incorrectly
- In very rare cases, compiler uses the knowledge that something is const to make some optimisations
- It's rather important when methods accepts references to objects, as it's the only way to ensure they don't modify the variables in the function that called it
- It's crucial when using more threads (usually to leverage more processor cores), because more cores can read a variable simultaneously, but cannot write into it simultaneously

Exercise

1. Create a class that gives access to lines in a file, but reads the (possibly gigabytes long) file only up to the line that's requested
2. Create a `SortedVector` class that encapsulates a vector of some numbers and has a method to sort it
3. Create a `Collection` class that allows reading numbers, adding them and changing them only when non-const and doesn't alow removing any
4. Create a `Factoriser` class that can factor integers and remember all integers it has already factorised to make it factor faster

- Use `const` wherever there's any reason to use it

Advanced exercise

1. Create a class that reads a file by chunks of 16 bytes and holds them in a vector, allowing a const access and a non-const access that saves the changes to disk when done (returns some kind of wrapper whose destructor makes the class update the object)
2. Create a `Decimator` class that holds numbers representing some data, each with time and value, and has a method to remove the one that will produce data with the smallest possible mean square difference

- Use `const` wherever there's any reason to use it

Homework

- Write a class that represents some measured data as a vector of pairs of time and value, has a method to return an interpolated value at some time between known values and caches recently interpolated values so that it would not have to compute them again if requested again shortly later
- You have two weeks to do it
- You must keep const correctness (and use `mutable` where it makes sense)