

# C++ 1 Les 4

Const

# Serius? Presentatie over *const*?

- C++ is een krachtige taal waarmee je rechtstreeks in de faciliteiten van het onderliggende OS kunt wroeten
- *"with great power comes great responsibility"*
- **const** maakt veiliger en correcter programmeren mogelijk
- Scott Meyer: *"use const wherever you can!"*

# Wat is const?

- Alles wat links van het woord “const” staat
- Tenzij er niets links van “const” staat, dan wat rechts van “const” staat.


# Wat kan const zijn? (1)

- variabele
  - `const int i = 42; // kan na initialisatie niet meer wijzigen, is dus een constante geworden`
  - beter (C++11): `const int i {42};`
- datgene waar pointer naar wijst
  - `const char* p {"hallo"};`
- pointer
  - `char* const p {"hallo"};`
- samen
  - `const char* const p {"hallo"};`


# Wat kan const

- parameter van een functie
- member function

wordt meegegeven by  
reference, maar kan toch  
niet worden gewijzigd  
door deze functie



```
class Book {  
public:  
    Book(const std::string& title);  
    std::string get_title() const;  
private:  
    std::string title;  
};
```



door deze functie aan  
te roepen wijzigt het  
object niet

# Wat kan const zijn? (3)

- zelfs van iterators bestaan const-versies

```
vector<int> v { 123, 456, 2000 };  
auto it = find(v.cbegin(), v.cend(), 456);  
v.insert(it, 1998);  
// 123, 1998, 456, 2000
```

# Gevolgen

- een functie met een const object als argument kan alleen de const functies van diens class aanroepen

```
void fun(const string& text) {  
    cout << text.length() << '\n';  
}
```

- het grijpt allemaal in elkaar
- frustratie van beginners: als je het fout toepast loop je steeds tegen een "muur van const"

# Conventie

- als je "getters" en "setters" maakt:
  - definieer de getters dan als const
  - geef setters een const & parameter

```
class Book {  
public:  
    Book();  
  
    void set_title(const std::string& title);  
    std::string get_title() const;  
  
private:  
    std::string title;  
};
```



# Complicatie

```
class A {  
public:  
    A() : b_initialized {false} {}  
    B get_b() const {  
        if (!b_initialized) {  
            b.init ();  
            b_initialized = true; // OEPS!  
        }  
        return b;  
    }  
private:  
    B b;  
    bool b_initialized;  
};
```

# Oplossing

```
class A {  
public:  
    A() : b_initialized {false} {}  
    B get_b() const {  
        if (!b_initialized) {  
            b.init ();  
            b_initialized = true;  
        }  
        return b;  
    }  
private:  
    B b;  
    mutable bool b_initialized;  
};
```

# Conclusie

- "use const wherever you can"
- als een member function de toestand van een object niet wijzigt, declareer hem dan const
- door const argumenten en const functies te gebruiken communiceer je duidelijk jouw intentie naar andere programmeurs
  - "ik blijf van jouw argumenten af"
  - "dit object verandert niet door deze functie"