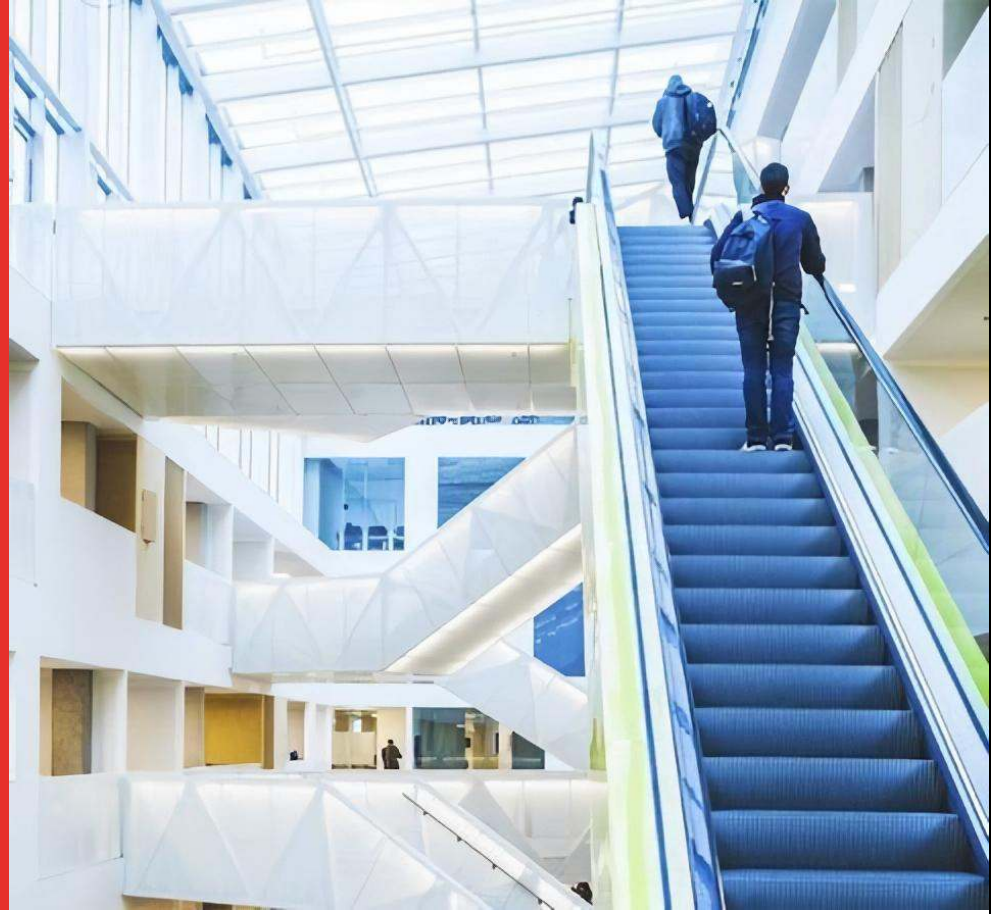
LAMBDA'S & STL



HOGESCHOOL UTRECHT

HIER KOMT ALLES SAMEN

PROGRAMMA

1. Lambda; wat?
2. Lambda; hoe?
3. Lambda; Capture Clause
4. Lambda; verdere syntax?
5. Lambda; waarom?
6. STL Algorithms
7. STL documentatie
8. Verdere info:
9. Huiswerk

LAMBDA; WAT?

'Anonieme' functies; closures

Pre C++11

```
int main () {
   std::vector<int> x = {2, 3, 5, 7, 11, 13};

   struct {
      int center;
      bool operator()(int x, int y) const {
         return abs(x - center) < abs(y - center);
      }
   } comp = {10};

   std::sort(x.begin(), x.end(), comp);

   std::for_each(x.begin(), x.end(), printer);

   return 0;
}
```

Post C++11

```
int main () {
   std::vector<int> x = {2, 3, 5, 7, 11, 13};

   int center = 10;
   std::sort(x.begin(), x.end(), [=](int x, int y){
      return abs(x - center) < abs(y - center);
   });

   std::for_each(x.begin(), x.end(), [](int v) {
      std::cout<<v<<std::endl;
   });

   return 0;
}
```

LAMBDA; WAT?

```cpp
int main () {
   std::vector<int> x = {2, 3, 5, 7, 11, 13};

   int center = 10;
   std::sort(x.begin(), x.end(), [=](int x, int y){ return abs(x - center) < abs(y - center); });

   std::for_each(x.begin(), x.end(), [](int v) { std::cout<<v<<std::endl; });

   return 0;
}
```

LAMBDA; HOE?

Meest basale vorm

```
1  int main(){
2     [](){std::cout << "Hallo Wereld" << std::endl;};
3  }
```

Als je het ook nog uit wilt voeren

```
1  int main(){
2     [](){std::cout << "Hallo Wereld" << std::endl;}();
3  }
```

Of wilt bewaren voor later

```
1  int main(){
2     auto func = [](){std::cout << "Hallo Wereld" << std::endl;};
3     func();
4  }
```

Een lambda is dus een soort functie, bestaande uit (minimaal):

```
1 [...](){...}
2 // Capture clause en een body
```

KLEIN UITSTAPJE; TYPES

- In vorige slide; `auto func = ...` - waarom?
- Elke lambda heeft een uniek type

```cpp
#include <iostream>
#include <typeinfo>
using namespace std;

int main(){
   auto x = []{return true;};
   auto y = []{return true;};
   cout << boolalpha << is_same<decltype(x), decltype(y)>::value << endl;
   cout << typeid(x).name() << " - " << typeid(y).name() << endl;
}
```

```
false
Z4mainE3$_0 - Z4mainE3$_1
```

KLEIN UITSTAPJE; TYPES

- In vorige slide; `auto func = ...` - waarom?
- Elke lambda heeft een uniek type, maar niet bij kopieën

```cpp
#include <iostream>
#include <typeinfo>
using namespace std;

int main(){
   auto x = []{return true;};
   auto y = x;
   cout << boolalpha << is_same<decltype(x), decltype(y)>::value << endl;
   cout << typeid(x).name() << " - " << typeid(y).name() << endl;
}
```

```
true
Z4mainE3$_0 - Z4mainE3$_0
```

LAMBDA; CAPTURE CLAUSE

Voor toegang tot buiten de scope

```cpp
#include <iostream>
int main(){
   int x = 12;
   int y = 10;
   auto foo = [x,y]{std::cout <<"x+y by value: "<< x+y << std::endl;};
   auto bar = [&x,&y]{std::cout <<"x+y by reference: "<< x+y << std::endl;};
   x = 42;
   auto baz = [&x,y]{std::cout <<"x+y mixed: " << x+y <<std::endl;};
   y = 22;
   foo();
   bar();
   baz();
}
```

```
x+y by value: 22
x+y by reference: 64
x+y mixed: 52
```

Je kunt ook álles capturen: `[=]` == alles by value, `[&]` == alles by reference

LAMBDA; VERDERE SYNTAX?

- If it quacks like a duck, and walks like a duck, it must be a duck!
- Lambda al bijna een 'functie', maar mist nog e.e.a!

```cpp
// Return type is niet nodig, maar is vooral voor leesbaarheid
int main(){
    auto halve = [](int& x){return x/2.0;};
    int n = 21;
    std::cout << halve(n) << std::endl;
    // >> 10.5!
}
```

```cpp
[capture_clause](parameters) -> return-type {body};
```

Trailing return types mogen overigens ook bij normale functies;

```cpp
auto halve(int& x) -> float {
    return x/2.0;
}
```

LAMBDA SYNTAX; EXTRA'S

```
1 // C++20 -> Template parameters
2 int y = 2;
3 [y]<typename T>(std::vector<T> v){return v[0]%y == 0; };
```


Daarna:

LAMBDA; WAAROM?

LAMBDA; WAAROM?

Belangrijkste usecase:

- Kleine functies meegeven aan andere functies
- Hogere orde functies (komen we later(?) op terug)

STL ALGORITHMS

STL ALGORITHMS

- Implementeert veelgebruikte algoritmes op containers
- vector, array, string, …
- Voor zoeken, aanpassen, tellen, sorteren, etc.

STL ALGORITHMS

Bijvoorbeeld, 'in place' verdubbelen:

```
1 // En vanaf C++20 nóg makkelijker
2 #include <vector>
3 #include <algorithm>
4
5 int main(){
6    std::vector v = {1,2,3,4,5,6,7}
7    std::ranges::for_each(v, [](auto& x){x*=2;} );
8 }
```

STL ALGORITHMS SYNTAX

Definities:

- 'Predicate'
- 'Criteria'

```
1   std::any(v.begin(), v.end(), [](auto& x){x%2 == 0;} );
2
3   std::find_if(v.begin(), v.end(), [](auto& x){x == 'c';} );
```

C++ Standard Library

The C++ standard library provides a wide range of facilities that are usable in standard C++.

Category

The language support library provides components that are required by certain parts of the C++ language, such as memory allocation (new/delete) and exception processing.

The concepts library describes library components that C++ programs may use to perform compile-time validation of template arguments and perform function dispatch based on properties of types. (since C++20)

The diagnostics library provides a consistent framework for reporting errors in a C++ program, including predefined exception classes.

The memory management library provides components for memory management, including smart pointers and scoped allocator(since C++11).

The metaprogramming library describes facilities for use in templates and during constant evaluation, including type traits, integer sequence,(since C++14) and rational arithmetic. (since C++11)

The general utilities library includes components used by other library elements, such as a predefined storage allocator for dynamic storage management, and components used as infrastructure in C++ programs, such as tuples and(since C++11) function wrappers.

The containers, iterators, ranges(since C++20), and algorithms libraries provide a C++ program with access to a subset of the most widely used algorithms and data structures.

The strings library provides support for manipulating text represented as homogeneous sequences of following types: `char`, `char8_t`(since C++20), `char16_t`, `char32_t`(since C++11), `wchar_t`, and any other character-like types.

The text processing library provides regular expression matching and searching(since C++11), utilities for text

STL DOCUMENTATIE

Waar te vinden:

- https://en.cppreference.com
- Offline door een tool als Zeal (https://zealdocs.org/)

VERDERE INFO:

Youtube:

- 'Algorithm Intuition' van Connor Hoekstra
- 'Better Algorithm Intuition' van Connor Hoekstra
- '105 STL Algorithms in Less Than an Hour' van Jonathan Boccara

HUISWERK

STL opdracht: schrijf zo min mogelijk eigen code!

\0