

Funkcionális STL

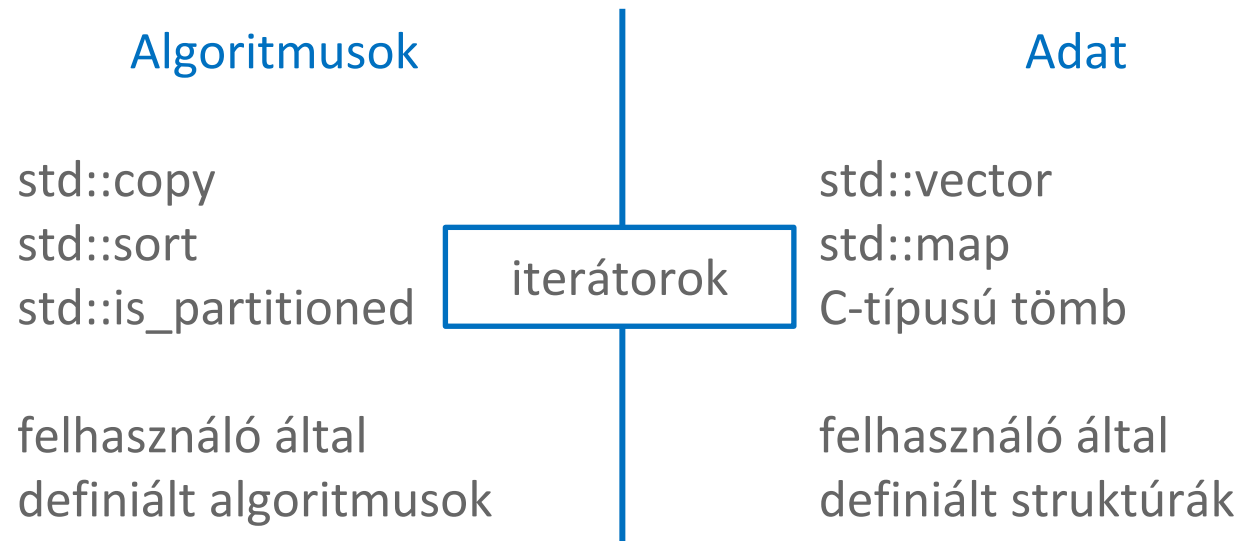
Mai témakör

- ▶ függékvobjektumok



Az STL

- ▶ Alexander Stepanov, 1993
- ▶ generikus könyvtár
- ▶ algoritmusok és adatstruktúrák szétválasztása



Iterátorok

- ▶ tagfüggvények
 - ▷ begin, end, cbegin, cend
 - ▷ rbegin, rend, crbegin, crend
- ▶ szabad függvények
 - ▷ std::begin, std::end, std::cbegin, std::cend
 - ▷ ...
 - ▷ std::next
 - ▷ std::distance



```
double data[100];  
for(auto it = std::cbegin(data); it!=std::cend(data); ++it) {  
    std::cout << *it;  
}
```

Algoritmusok vagy ciklusok?

```
std::copy(employees.begin(), employees.end(), std::back_inserter(eRegister));
```



```
for (const auto &emp : employees) {  
    eRegister.push_back(emp);  
}
```

std::accumulate I.

```
std::vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

const auto sum      = std::accumulate(v.cbegin(), v.cend(), 0);
const auto product = std::accumulate(v.cbegin(), v.cend(), 1,
                                     std::multiplies<int>());

std::cout << "Sum: "      << sum      << "\n";
std::cout << "Product: " << product << "\n";
```



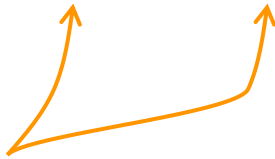
```
#include <numeric>    // std::accumulate
#include <functional> // std::multiplies
```

ADL

- ▶ argument-dependent lookup
 - ▶ régebben: Koenig lookup
 - ▶ a paraméterek névtérében is keres

```
std::vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
const auto sum = accumulate(v.cbegin(), v.cend(), 0);
```



OK, a paraméter a std névtérben van



Miért működik?
Hisz nincs névtér
megadva...

ex_0: std::accumulate II.

- ▶ feladat
 - ▶ tárolóban tárolt számok elválasztása kötőjelekkel

1 2 3 4 5

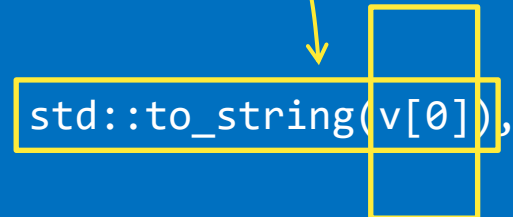
1 - 2 - 3 - 4 - 5



```
std::vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
const auto s = std::accumulate(std::next(std::cbegin(v)), std::cend(v),  
                                std::to_string(v[0]),  
                                [](std::string a, int b) {  
                                    return a + '-' + std::to_string(b);  
                                });
```

kezdőérték



a vector nem lehet üres

Függvényobjektumok

- ▶ `operator()` túlterhelve



```
class function_object {  
    ....  
public:  
    return_type operator()(arguments) const {  
        ...  
    }  
};
```

Generikus függvényobjektumok I.

```
template<typename T>
class older_than {
private:
    std::size_t limit;

public:
    older_than(std::size_t limit) : limit{ limit } {}

    bool operator()(const T &object) const {
        return object.age() > limit;
    }
};
```



```
std::count_if(people.cbegin(), people.cend(), older_than<person_t>{ 42 });
```

Generikus függvényobjektumok II.

```
class older_than {  
    private:  
        std::size_t limit;  
  
    public:  
        older_than(std::size_t limit) : limit{ limit } {}  
  
        template<typename T>  
        bool operator()(T &&object) const {  
            return std::forward<T>(object).age() > limit;  
        }  
};
```



```
std::count_if(people.cbegin(), people.cend(), older_than{ 42 });
```

#include <functional> I.

- ▶ aritmetikai és logikai műveletek függvényobjektum formában

- ▶ bit_and
- ▶ bit_or
- ▶ bit_xor
- ▶ logical_and
- ▶ logical_not
- ▶ logical_or

```
int flags[] { 1, 2, 4, 8, 16, 32, 64, 128 };
```

```
int acc = std::accumulate(std::cbegin(flags), std::cend(flags), 0, std::bit_or<>{});
```

```
std::cout << "accumulated: " << acc << '\n';
```

255



#include <functional> II.

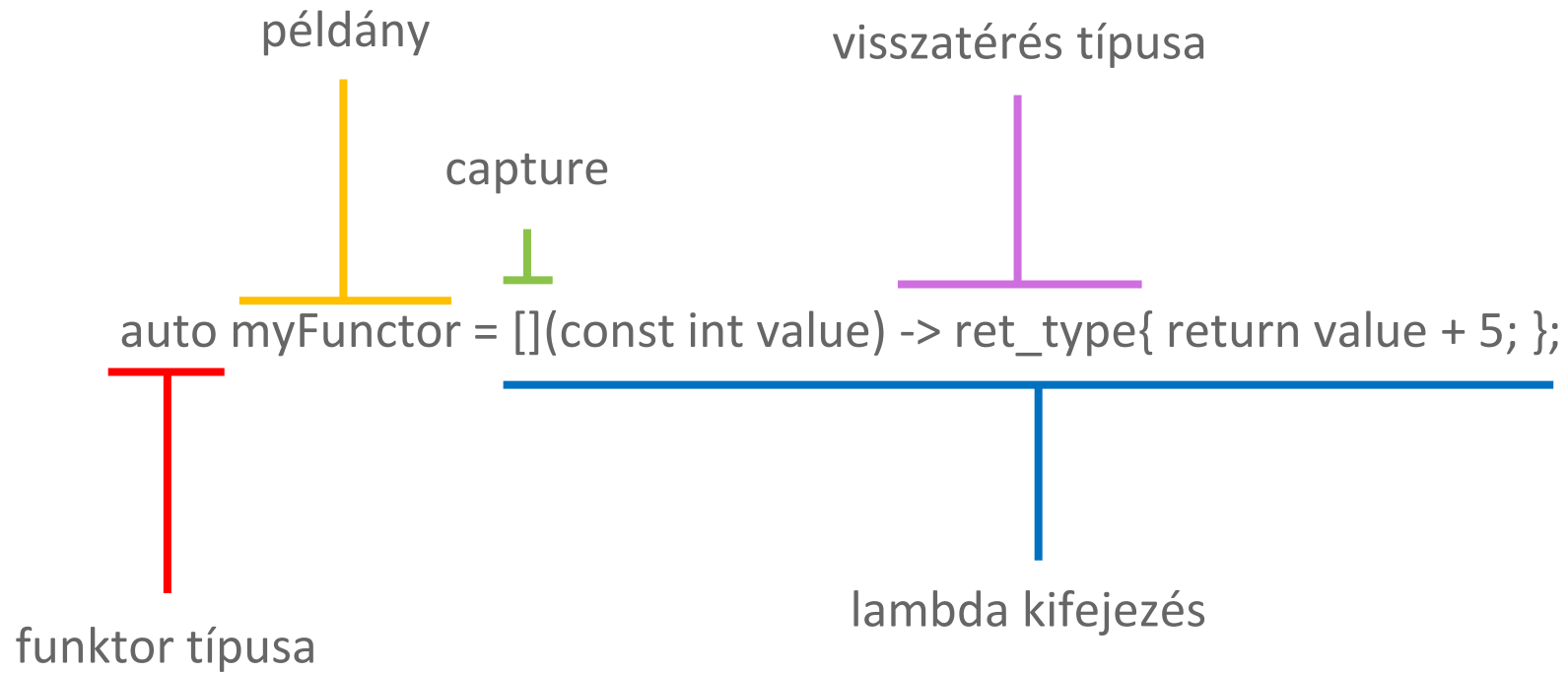
- ▶ aritmetikai és logikai műveletek függvényobjektum formában

▶ divides	int first[] { 10, 40, 90, 40, 10 };
▶ equal_to	int second[] { 1, 2, 3, 4, 5 };
▶ greater	
▶ greater_equal	int results[5];
▶ less	
▶ less_equal	std::transform(first, first+5, second, results, std::divides<>{});
▶ minus	
▶ modulus	for (int i=0; i<5; i++)
▶ multiplies	std::cout << results[i] << ',';
▶ negate	
▶ not_equal_to	
▶ plus	



10,20,30,10,2

Lambda kifejezések



```
std::copy_if(people.cbegin(), people.cend(), std::back_inserter(females), [](const person_t &person){  
    return person.gender() == person_t::female  
});
```

IIFE, IILE

- ▶ IIFE = immediately-invoked function expression
- ▶ IILE = immediately Invoked Lambda Expression
- ▶ komplex inicializálás

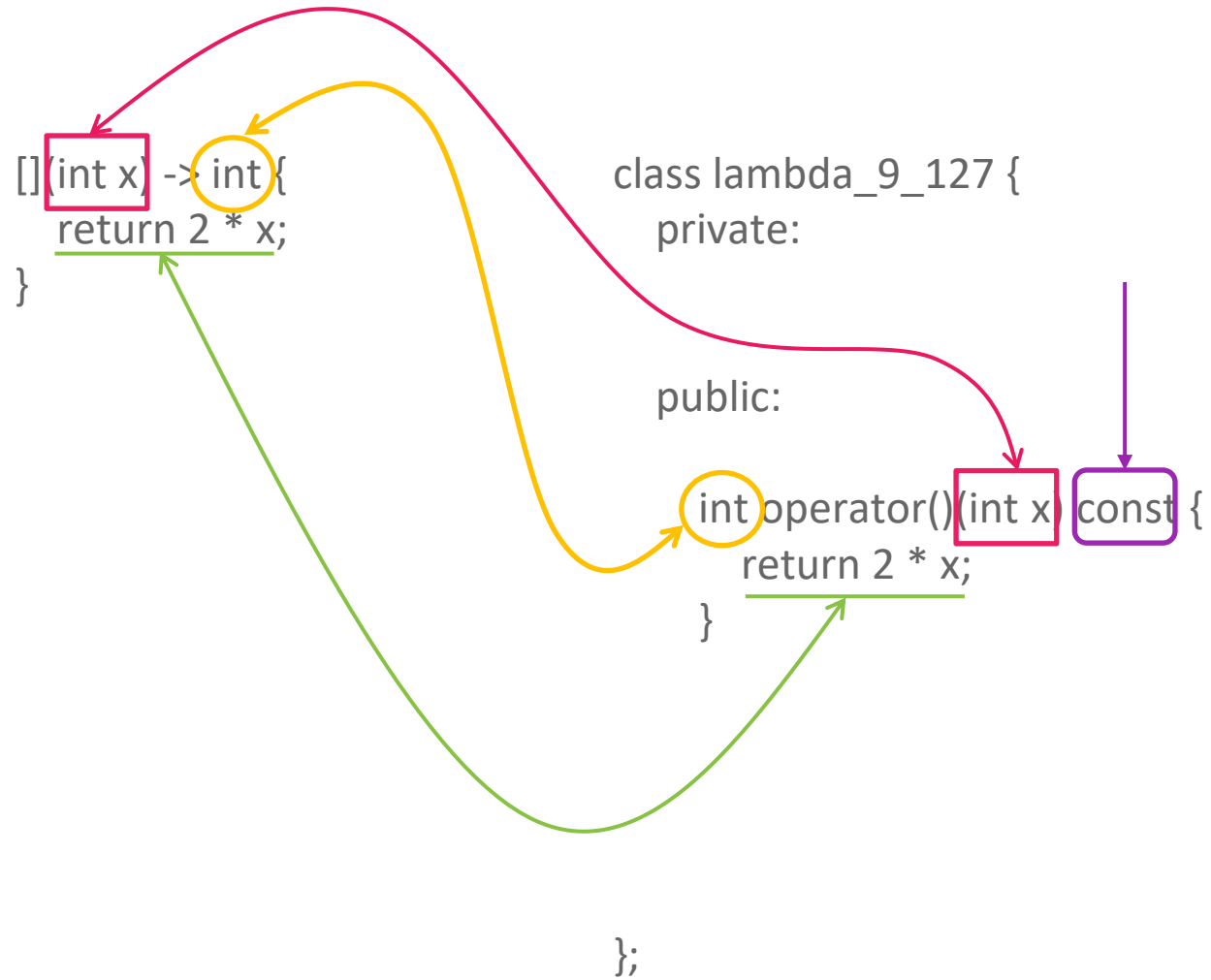
lehet const a változó

```
void BuildStringTestIIFE2(std::string link, std::string text) {  
    const std::string html = [&] {  
        const auto& inText = text.empty() ? link : text;  
        return "<a href=\"\" + link + \"\">\" + inText + "</a>\";  
    }();  
    std::cout << html << '\n';  
}
```

azonnali függvényhívás

```
buildString("https://isocpp.org", "ISO C++");  
buildString("https://isocpp.org", "");
```

[] lambda kifejezések I.



[] lambda kifejezések II.

```
[](int x) {  
    return 2 * x;  
}
```

```
class lambda_9_127 {  
    private:  
        static inline int __invoke(int a) { return 2 * a; }
```

```
    public:
```

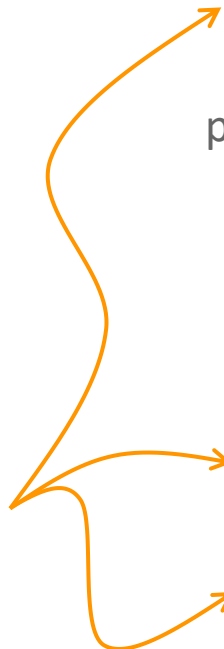
```
        int operator()(int x) const {  
            return 2 * x;  
        }
```

```
        using retType_15_16 = void (*)(int);
```

```
        inline operator retType_15_16 () const { return __invoke; };
```

```
};
```

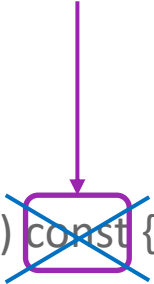
C típusú függvénymutató



[] lambda kifejezések III.

```
[](int x) mutable {  
    return 2 * x;  
}
```

```
class lambda_9_127 {  
    private:  
  
    public:  
  
    int operator()(int x) const {  
        return 2 * x;  
    }  
  
};
```




[...] lambda kifejezések I.

- ▶ `=`
 - ▶ minden változót érték szerint (ami nincs másképp megadva a capture-ben)
- ▶ `&`
 - ▶ minden hatókörben levő változót referencia szerint (ha nincs másképp megadva a capture-ben)
- ▶ `this`
 - ▶ pl. tagváltozók “elkapásához”
- ▶ `&x`
 - ▶ az x változót referencia szerint
- ▶ `up = std::move(p)`
 - ▶ C++14
 - ▶ elnevezés

```
auto lambda = [age = 18]() {  
    std::cout << age << std::endl;  
};
```

```
static int carWheels = 4;  
int main() {  
    auto howManyWheels = []() { return carWheels; };  
}
```



[...] lambda kifejezések II.

```
[x, &y](int a) {  
    y = 2 * a;  
    return 2 * a;  
}
```

```
class lambda_9_128 {
```

```
private:
```

```
    int x;  
    int &y;
```

```
public:
```

```
    lambda_9_128(int x, int &y) : x{ x }, y{ y } {}
```

```
    int operator()(int a) const {
```

```
        y = 2 * a;  
        return a * x;  
    }
```

```
};
```

nincs C típusú
függvénytípus

Generikus lambda kifejezések

- ▶ paraméter típusa **auto**

- ▶ sablon

```
[(auto x)]{  
    return 2 * x;  
}
```

```
class lambda_9_127 {  
public:
```

```
template<typename T>  
auto operator()(T x) const {  
    return 2 * x;  
}
```

```
};
```



C++20 lambda kifejezések

- ▶ `auto glambda = []<class T>(T a, auto&& b) { return a < b; };`
- ▶ `auto f = []<typename ...Ts>(Ts&& ...ts) { return foo(std::forward<Ts>(ts)...); };`

C++20 előtt:

```
[(auto first, decltype(first) second) {  
    ...  
}]
```



C++20-től kezdve:

```
[]<typename T>(T first, T second) {  
    ...  
}]
```

std::inner_product l.

- ▶ skaláris (belső) szorzat

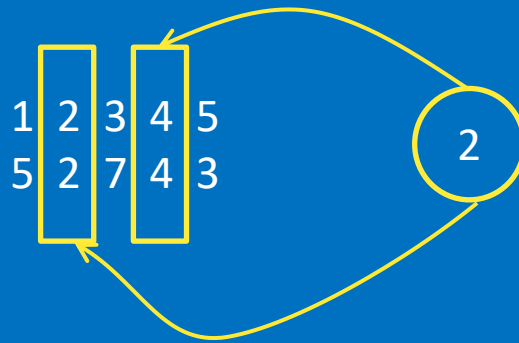


```
std::vector<int> a{ 0,1,2,3,4 };  
std::vector<int> b{ 5,4,2,3,1 };
```

```
const auto r1 = std::inner_product(a.cbegin(), a.cend(), b.cbegin(), 0);  
std::cout << "The inner product is: " << r1 << "\n";
```

ex_1: std::inner_product II.

- ▶ feladat
 - ▶ azon pozíciók számát, amelyben mindkét tároló azonos elemet tartalmaz



```
std::vector<int> a{ 0, 1, 2, 3, 4 };  
std::vector<int> b{ 5, 4, 2, 3, 1 };
```

```
const auto r2 = std::inner_product(a.begin(), a.end(), b.begin(), 0,  
                                   std::plus<>(), std::equal_to<>());
```

redukció

elemenkénti művelet

std::all_of, std::any_of, std::none_of

- ▶ C++17
- ▶ predikátum ellenőrzése az összes elemre

```
std::vector<int> v(10, 2);  
std::partial_sum(v.cbegin(), v.cend(), v.begin());
```

```
if (std::all_of(v.cbegin(), v.cend(), [](int i) { return i % 2 == 0; })) {  
    std::cout << "Mind paros."  
}
```

```
if (std::any_of(v.cbegin(), v.cend(), [](int i) { return i % 7 == 0; })) {  
    std::cout << "Legalabb egy osztható 7-tel."  
}
```

```
if (std::none_of(v.cbegin(), v.cend(), std::bind(std::modulus<>(), std::placeholders::_1, 2))) {  
    std::cout << "Egyik sem paratlan."  
}
```

ex_2: Makaronok

- ▶ feladat
 - ▶ asztal telis-tele macaronokkal
 - ▶ C_i sorrendben az i -edik alkalommal megevett makaron kalóriatartalma
 - ▶ $2^i C_i$ kilométert kell futni
 - ▶ minden makaront meg kell enni
- ▶ legkevesebb hány kilométer kell futni
 - ▶ milyen sorrendben kell ehhez megenni a makaronokat

C++11 óta nem kötelező
a típust megadni



Először azt, amelynek a legnagyobb a kalóriatartalma.

```
std::sort(std::begin(v), std::end(v), std::greater<>());
```

Makaronok



```
std::vector<int> v{ 1,2,3,5,1,5};
```

```
std::sort(v.begin(), v.end());
```

```
const auto km = std::accumulate(v.cbegin(), v.cend(), 0ll, [](long long res, int i) {  
    return (res << 1) + i;  
});
```

$$\begin{array}{ccc} & \xrightarrow{\hspace{10em}} & \\ C_2 & C_1 & C_0 & & 127 \\ 2^0 \cdot C_2 + 2^1 \cdot C_1 + 2^2 \cdot C_0 & & 10^0 \cdot 7 + 10^1 \cdot 2 + 10^2 \cdot 1 \\ 2 (2 (C_0) + C_1) + C_2 & & 10 (10 (1) + 2) + 7 \\ & \xleftarrow{\hspace{10em}} & \end{array}$$

ex_3: Camel case

- ▶ feladat
 - ▶ fájl feldolgozása soronként
 - ▶ a sorokban található camel case szavak felbontása, összegyűjtése

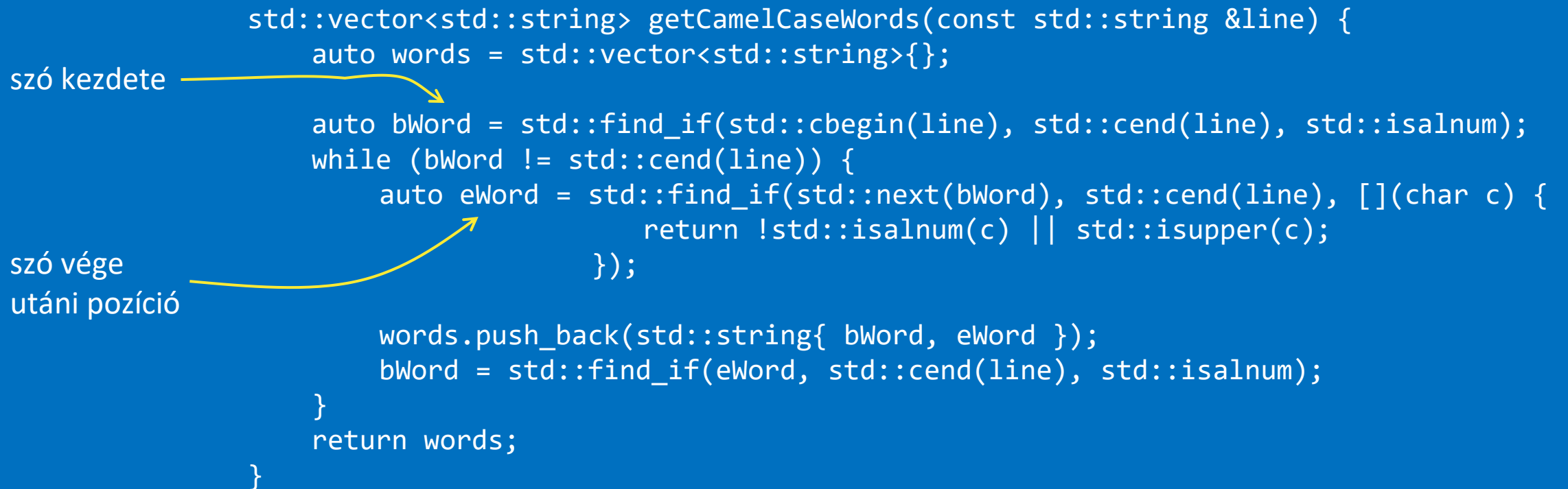


Camel case - megoldás

```
std::vector<std::string> getCamelCaseWords(const std::string &line) {  
    auto words = std::vector<std::string>{};  
  
    auto bWord = std::find_if(std::cbegin(line), std::cend(line), std::isalnum);  
    while (bWord != std::cend(line)) {  
        auto eWord = std::find_if(std::next(bWord), std::cend(line), [](char c) {  
            return !std::isalnum(c) || std::isupper(c);  
        });  
  
        words.push_back(std::string{ bWord, eWord });  
        bWord = std::find_if(eWord, std::cend(line), std::isalnum);  
    }  
    return words;  
}
```

szó kezdete

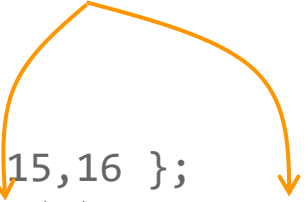
szó vége
utáni pozíció



Párhuzamos algoritmusok

- ▶ C++17
- ▶ `#include <execution>`
 - ▶ `std::execution::par`
 - ▶ `std::execution::seq`
 - ▶ `std::execution::par_unseq`
 - ▶ `std::execution::unseq` (C++20)
- ▶ “első előtti parameter”
- ▶ az algoritmus neve általában nem változik megváltoztatjuk az elemeket
 - ▶ `std::accumulate` -> `std::reduce`

```
int a[] { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };  
std::for_each(std::execution::par, std::begin(a), std::end(a), [](int &i)  
{  
    i++;  
});
```



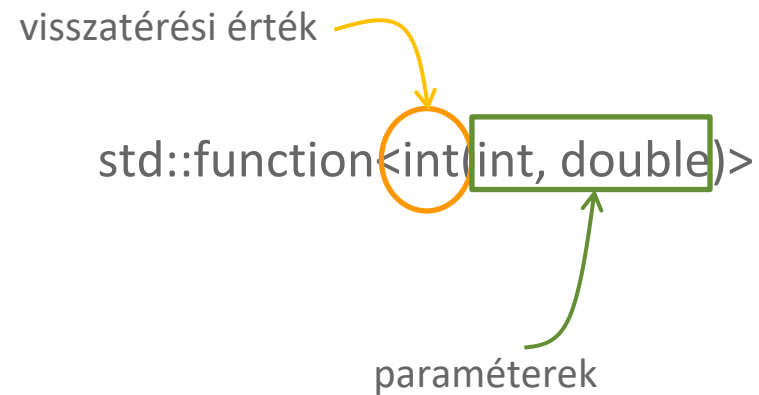
std::function

- ▶ `#include <functional>`
 - a tárolt függvényobjektum szignatúrája határozza meg a típusát
- ▶ függvényobjektum tárolása olyan osztályon belül, amely maga nem lehet sablon
- ▶ virtuális függvényhívással ekvivalens többletköltség
- ▶ small-function-object optimalizálás
 - pl. függvény mutató
 - `std::function`-on belül van tárolva
 - nem kell dinamikusan memóriát allokálni

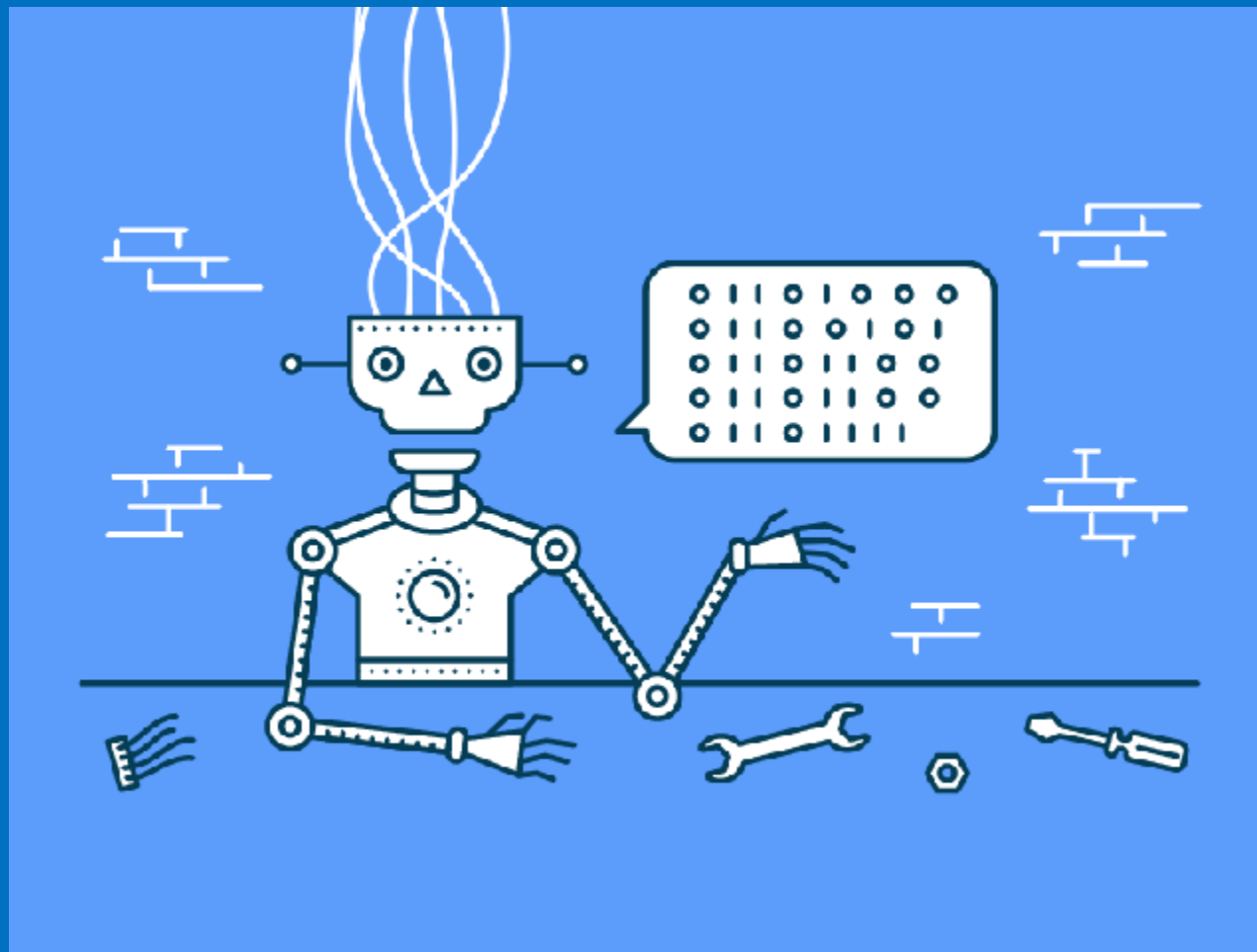
visszatérési érték

`std::function<int(int, double)>`

paraméterek

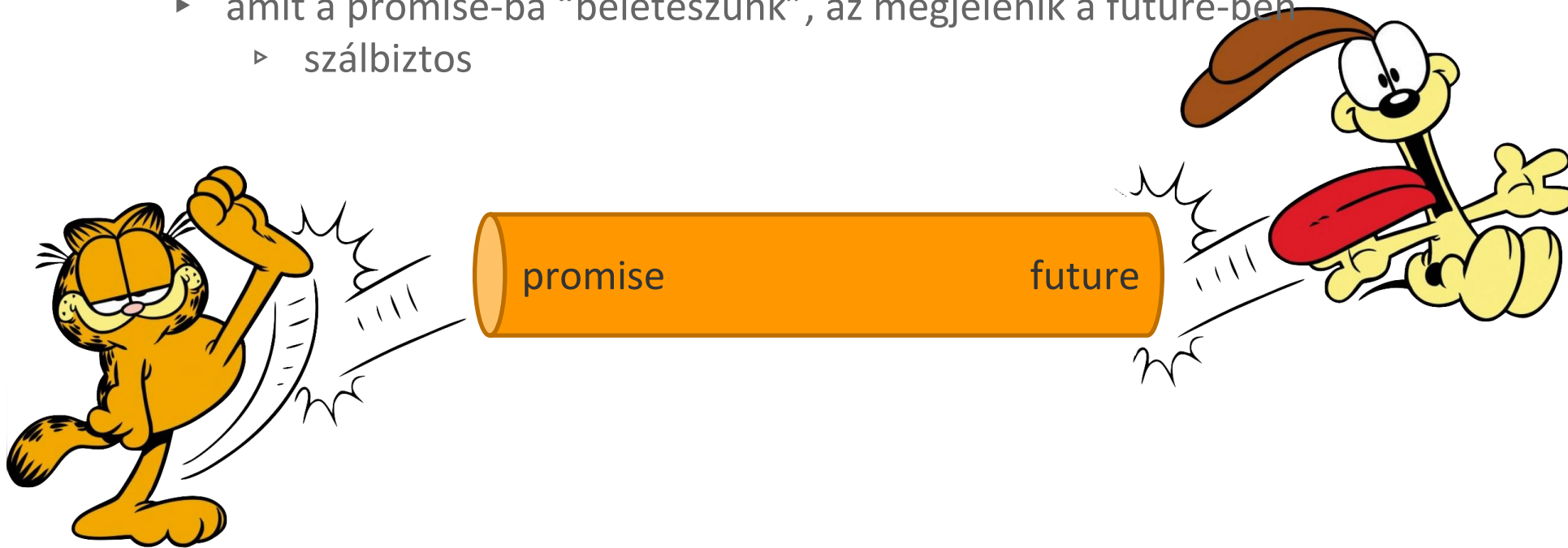
A diagram illustrating the components of the `std::function` template signature. The text `std::function<int(int, double)>` is shown. An orange circle highlights the `int` inside the angle brackets, with a yellow arrow pointing to it from the label 'visszatérési érték' (return value). A green rectangle highlights the `(int, double)` part of the signature, with a green arrow pointing to it from the label 'paraméterek' (parameters).

ex_4: Task



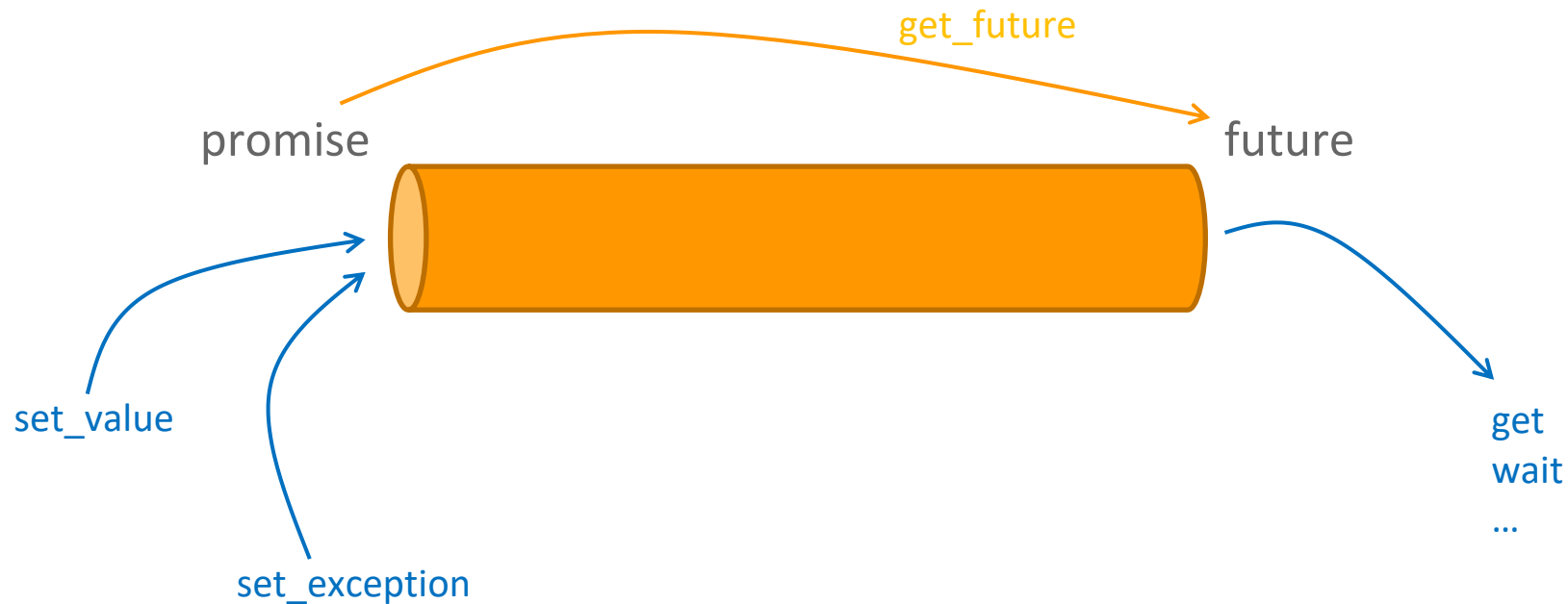
std::promise / std::future I.

- ▶ egy (egyszer használható) csatorna két vége
- ▶ amit a promise-ba “beleteszünk”, az megjelenik a future-ben
 - ▶ szálbiztos



std::promise / std::future II.

- ▶ egy (egyszer használható) csatorna két vége
- ▶ amit a promise-ba “beleteszünk”, az megjelenik a future-ben
 - ▶ szálbiztos



#include <future>

Köszönöm a figyelmet!

Folytatjuk...