

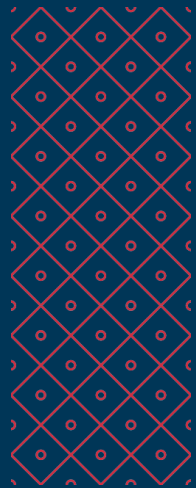


**Charles
University**

Programming in C++

Labs - winter 2025/26

Tomáš Faltín, <https://fan1x.github.io/>



Lab 4

24/10/2025

Homework Feedback

Will be put today into Gitlab

The **first occurrence** of mistakes **can be corrected** during one iteration of code review.

Repeated mistakes will be **penalized**.

Defining Your Own Types

Use `using` (or `typedef` in old C/C++)

Can be templated (later)

```
using my_int = int;  
using int_pair_t = std::pair<my_int, my_int>;  
using my_string = std::vector<char>;  
using int_vector_t = std::vector<int>;
```

```
my_int x = 3;  
int_pair_t p{10, 20};  
my_string str = {'a', 'b', 'c'};  
int_vector_t vi(10, 0);
```

Constant Values

Read only value that cannot be changed

Naming values in code

~ Every number in the code should be a named constant

```
constexpr double PI = 3.14;
```

```
constexpr size_t MAX_SIZE = 16 * 1024 * 1024;
```

const

A constant value that cannot be changed

constexpr

Constant value (potentially) evaluated at

compile time

Can be used as arguments to templates

Both can be used together with `static` (later)

Argument Passing - Recap

How would you design a (global) function that sums two matrixes and return the new one?

Return

- a) `Matrix sum(...)`
- b) `const Matrix sum(...)`
- c) `Matrix &sum(...)`
- d) `const Matrix &sum(...)`
- e) `Matrix &&sum(...)`
- f) `Matrix *sum(...)`
- g) `const Matrix *sum(...)`

Argument Passing - Recap

How would you design a (global) function that sums two matrixes and return the new one?

Arguments

- a) `Matrix sum(Matrix m1, Matrix m2)`
- b) `Matrix sum(const Matrix m1, const Matrix m2)`
- c) `Matrix sum(const Matrix &m1, const Matrix &m2)`
- d) `Matrix sum(Matrix &m1, Matrix &m2)`
- e) `Matrix sum(Matrix &&m1, Matrix &&m2)`

Transferring Ownership

R-value references - &&

Moves the object somewhere (into a function)

- Use `std::move` on the caller side

The object no longer lives outside (the function)

Typical usage

A single owner, e.g., `std::unique_ptr`

Processing large objects

```
vector<unique_ptr<int>>::push_back(  
    unique_ptr<int> &&new_obj);
```

```
vector<unique_ptr<int>> vector_of_ints;  
vector_of_ints.push_back(move(make_unique<int>(x)));
```

Transferring Ownership

R-value references - &&

Moves the object somewhere (into a function)

- Use `std::move` on the caller side

The object no longer lives outside (the function)

Typical usage

A single owner, e.g., `std::unique_ptr`

Processing large objects

```
vector<unique_ptr<int>>::push_back(  
    unique_ptr<int> &&new_obj);
```

```
vector<unique_ptr<int>> vector_of_ints;  
vector_of_ints.push_back(move(make_unique<int>(x)));
```

```
vector<int> vi1{1, 2, 3};  
vector<int> vi2 = std::move(vi1);  
print(vi1);
```



What is inside vi1?

Special Methods

```
class Verbose {
    int x;
public:
    Verbose() {
        cout << "default ctor\n";
        this->x = 1;
    }

    Verbose(const Verbose &v) {
        cout << "copy ctor\n";
        this->x = v.x;
    }

    Verbose(Verbose &&v) {
        cout << "move ctor\n";
        this->x = v.x;
        v.x = 0;
    }

    ~Verbose() {
        cout << "dtor\n";
    }

    Verbose(int x) {
        cout << "user ctor\n";
        this->x = x;
    }
}
```

```
Verbose &operator=(const Verbose &v) {
    cout << "copy assignment\n";
    this->x = v.x;
    return *this;
}

Verbose &operator=(Verbose &&v) {
    cout << "move assignment\n";
    this->x = v.x;
    return *this;
}

};

int main()
{
    Verbose v1; // default ctor
    Verbose v2(2); // user ctor
    Verbose v3{3}; // user ctor
    Verbose v4(v2); // copy ctor
    Verbose v5 = v3; // copy ctor
    Verbose v6(std::move(v1)); // move ctor
    Verbose v7 = std::move(v4); // move ctor
    v1 = v2; // copy assignment
    v2 = std::move(v3); // move assignment
} // Calls destructors
```

Static Members

Attribute/method belongs to a class (not an object/instance)

Need to **share** attribute/method **among** all the objects/**instances**

Accesses through ::

Most things belong to an object

```
class Verbose {  
    static int cnt;  
}; // class
```

Declaration

```
int Verbose::cnt = 0;
```

Initialization

```
int main()  
{  
    Verbose v1; // 1st object/instance  
    Verbose v2(2); // 2nd object/instance  
    std::cout << Verbose::cnt;  
}
```

Access

Static Members - Example

```
class CountingClass {
    static size_t num_instances;

    static void change_num_instances(int val) {
        num_instances += val;
    }

public:
    static bool has_instance() {
        return num_instances > 0;
    }
    static size_t get_num_instances() {
        return num_instances;
    }

    CountingClass() { ??? }
    ~CountingClass() { ??? }

    CountingClass(const CountingClass &) {
        ???
    }
    CountingClass(CountingClass &&) { ??? }
};

size_t CountingClass::num_instances = 0;
```

```
void f() {
    cout << CountingClass::get_num_instances() << endl; // 0
    CountingClass cc1;
    cout << CountingClass::get_num_instances() << endl; // 1
    CountingClass cc2 = cc1;
    cout << CountingClass::get_num_instances() << endl; // 2
    std::vector<CountingClass> ccs(10);
    cout << CountingClass::get_num_instances() << endl; // 12
}

int main() {
    cout << CountingClass::get_num_instances() << endl; // 0
    f();
    cout << CountingClass::get_num_instances() << endl; // 0
}
```

Static Members - Example

```
class CountingClass {
    static size_t num_instances;

    static void change_num_instances(int val) {
        num_instances += val;
    }

public:
    static bool has_instance() {
        return num_instances > 0;
    }
    static size_t get_num_instances() {
        return num_instances;
    }

    CountingClass() { change_num_instances(+1); }
    ~CountingClass() { ??? }

    CountingClass(const CountingClass &) {
        ???
    }
    CountingClass(CountingClass &&) { ??? }
};

size_t CountingClass::num_instances = 0;
```

```
void f() {
    cout << CountingClass::get_num_instances() << endl; // 0
    CountingClass cc1;
    cout << CountingClass::get_num_instances() << endl; // 1
    CountingClass cc2 = cc1;
    cout << CountingClass::get_num_instances() << endl; // 2
    std::vector<CountingClass> ccs(10);
    cout << CountingClass::get_num_instances() << endl; // 12
}

int main() {
    cout << CountingClass::get_num_instances() << endl; // 0
    f();
    cout << CountingClass::get_num_instances() << endl; // 0
}
```

Static Members - Example

```
class CountingClass {
    static size_t num_instances;

    static void change_num_instances(int val) {
        num_instances += val;
    }

public:
    static bool has_instance() {
        return num_instances > 0;
    }
    static size_t get_num_instances() {
        return num_instances;
    }

    CountingClass() { change_num_instances(+1); }
    ~CountingClass() { change_num_instances(-1); }

    CountingClass(const CountingClass &) {
        ???
    }
    CountingClass(CountingClass &&) { ??? }
};

size_t CountingClass::num_instances = 0;
```

```
void f() {
    cout << CountingClass::get_num_instances() << endl; // 0
    CountingClass cc1;
    cout << CountingClass::get_num_instances() << endl; // 1
    CountingClass cc2 = cc1;
    cout << CountingClass::get_num_instances() << endl; // 2
    std::vector<CountingClass> ccs(10);
    cout << CountingClass::get_num_instances() << endl; // 12
}

int main() {
    cout << CountingClass::get_num_instances() << endl; // 0
    f();
    cout << CountingClass::get_num_instances() << endl; // 0
}
```

Static Members - Example

```
class CountingClass {
    static size_t num_instances;

    static void change_num_instances(int val) {
        num_instances += val;
    }

public:
    static bool has_instance() {
        return num_instances > 0;
    }
    static size_t get_num_instances() {
        return num_instances;
    }

    CountingClass() { change_num_instances(+1); }
    ~CountingClass() { change_num_instances(-1); }

    CountingClass(const CountingClass &) {
        change_num_instances(+1);
    }
    CountingClass(CountingClass &&) { ??? }
};

size_t CountingClass::num_instances = 0;
```

```
void f() {
    cout << CountingClass::get_num_instances() << endl; // 0
    CountingClass cc1;
    cout << CountingClass::get_num_instances() << endl; // 1
    CountingClass cc2 = cc1;
    cout << CountingClass::get_num_instances() << endl; // 2
    std::vector<CountingClass> ccs(10);
    cout << CountingClass::get_num_instances() << endl; // 12
}

int main() {
    cout << CountingClass::get_num_instances() << endl; // 0
    f();
    cout << CountingClass::get_num_instances() << endl; // 0
}
```

Static Members - Example

```
class CountingClass {
    static size_t num_instances;

    static void change_num_instances(int val) {
        num_instances += val;
    }

public:
    static bool has_instance() {
        return num_instances > 0;
    }
    static size_t get_num_instances() {
        return num_instances;
    }

    CountingClass() { change_num_instances(+1); }
    ~CountingClass() { change_num_instances(-1); }

    CountingClass(const CountingClass &) {
        change_num_instances(+1);
    }
    CountingClass(CountingClass &&) { ??? }
};

size_t CountingClass::num_instances = 0;
```

```
void f() {
    cout << CountingClass::get_num_instances() << endl; // 0
    CountingClass cc1;
    cout << CountingClass::get_num_instances() << endl; // 1
    CountingClass cc2 = cc1;
    cout << CountingClass::get_num_instances() << endl; // 2
    std::vector<CountingClass> ccs(10);
    cout << CountingClass::get_num_instances() << endl; // 12
}

int main() {
    cout << CountingClass::get_num_instances() << endl; // 0
    f();
    cout << CountingClass::get_num_instances() << endl; // 0
}
```

Try yourself

Work 1: Implement Class C

Implement class C so the program outputs:

1,2,3,...,16

Touch **only** class C, nothing else

- Nothing can be added/changed main() or fn_XXX()

Don't use exit(), break, goto, ...

Hint: which methods are called?

Submit the final version to Gitlab → 1 point

- **Deadline: Thursday October 30th 5:00**

```
class C { /* implement me */ };
```

```
// Don't touch anything below!!!
void fn_copy(C) {}
void fn_cref(const C&) {}
void fn_rref(C&&) {}

int main(int argc, char* argv[])
{
    cout << "1\n";
    C c1;
    cout << "3\n";
    C c2(c1);
    cout << "5\n";
    C c3 = c2;
    cout << "7\n";
    fn_copy(c1);
    cout << "9\n";
    fn_cref(c1);
    fn_copy(std::move(c1));
    fn_rref(std::move(c2));
    cout << "11\n";
    c3 = c2;
    cout << "13\n";
    c2 = std::move(c1);
    cout << "15\n";
}
```

Work 2: 3D Matrix for Integers v2.0

New API

- `print()`
- `sort_vector(x, y) // check STL`

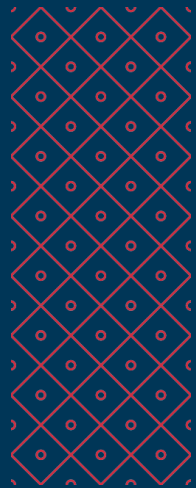
Try and benchmark performance of different containers: `std::vector`, `std::deque`, `std::list`, `std::array`

- Should be ~2 (un)commented lines only to change the container
- `std::chrono::system_clock::now();`
- Put your measured numbers into a comment
 - **Release** mode, **large enough tests**, ...

Correct all issues from the previous HW
Implement correctly all special methods
Show usage/test

Submit the final version to Gitlab → 2 points

- **Deadline: Thursday October 30th 5:00**



Lab 2

10/10/2025

Homework

Comments in Gitlab

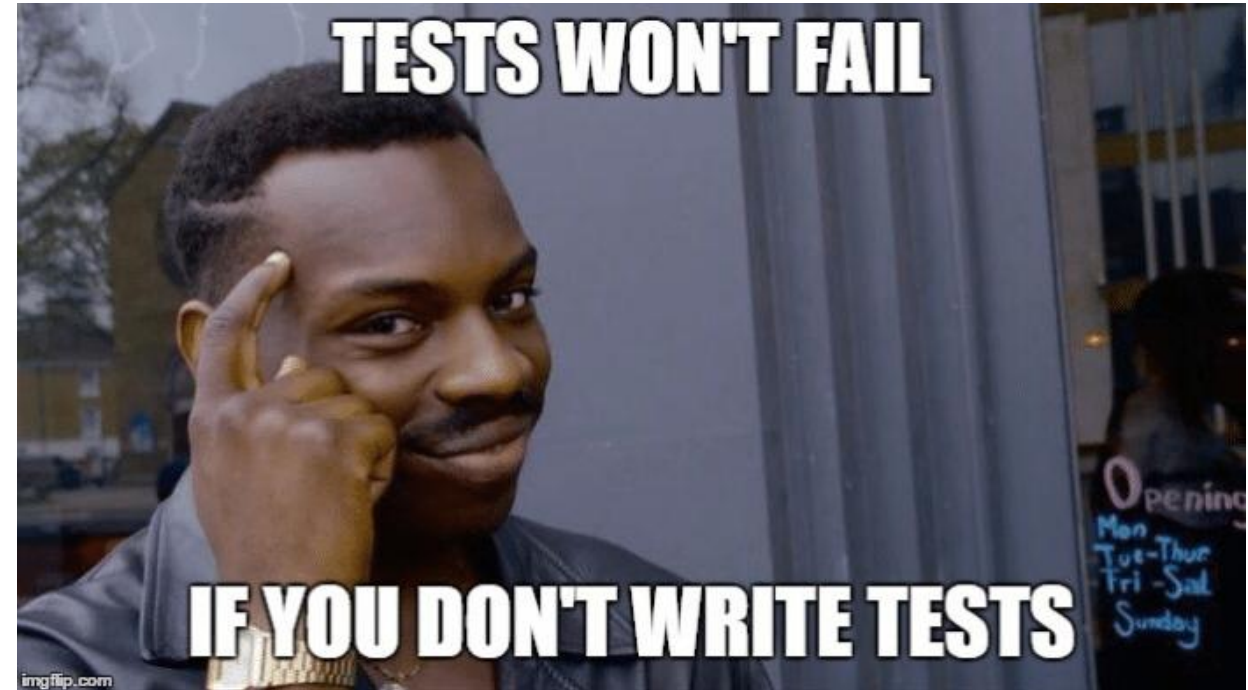
Points can be found in “Studijní mezivýsledky”

Homework Feedback

Testing is integral part of the assignment

I expected to see at least an attempt

Don't choose the easier way out



Homework Feedback

Testing is integral part of the assignment

Use STL wherever possible

- Someone **smart** already spent lots of time **programming** and **testing** it

`std::stod` instead of handmade parsing:

When I show my C code to

C Developers



ProgrammerHumor.io

C++ Developers



@12_Semitones

Core Guidelines

Set of guidelines for using C++ well

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#fcall-parameter-passing>



Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

```
_GLIBCXX_NODISCARD _GLIBCXX20_CONSTEXPR
reference
operator[](size_type __n) _GLIBCXX_NOEXCEPT
{
    __glibcxx_requires_subscript(__n);
    return *(this->_M_impl._M_start + __n);
}

/GLIBCXX_NODISCARD _GLIBCXX20_CONSTEXPR
const_reference
operator[](size_type __n) const _GLIBCXX_NOEXCEPT
{
    __glibcxx_requires_subscript(__n);
    return *(this->_M_impl._M_start + __n);
}
```

Link: https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/stl_vector.h

Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

```
template<typename _CharT, typename _Traits, typename _Alloc>
    _GLIBCXX20_CONSTEXPR
    void
    basic_string<_CharT, _Traits, _Alloc>::
    _M_assign(const basic_string& __str)
    {
        if (this != std::__addressof(__str))
        {
            const size_type __rsize = __str.length();
            const size_type __capacity = capacity();

            if (__rsize > __capacity)
            {
                size_type __new_capacity = __rsize;
                pointer __tmp = _M_create(__new_capacity, __capacity);
                _M_dispose();
                _M_data(__tmp);
                ...
            }
        }
    }
```

Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

```
template<typename _CharT, typename _Traits, typename _Alloc>
    _GLIBCXX20_CONSTEXPR
    typename basic_string<_CharT, _Traits, _Alloc>::pointer
    basic_string<_CharT, _Traits, _Alloc>::
        _M_create(size_type& __capacity, size_type __old_capacity)
    {
        // _GLIBCXX_RESOLVE_LIB_DEFECTS
        // 83. String::npos vs. string::max_size()
        if (__capacity > max_size())
            std::__throw_length_error(__N("basic_string::_M_create"));

        // The below implements an exponential growth policy, necessary to
        // meet amortized linear time requirements of the library: see
        // http://gcc.gnu.org/ml/libstdc++/2001-07/msg00085.html.
        if (__capacity > __old_capacity && __capacity < 2 * __old_capacity)
        {
            __capacity = 2 * __old_capacity;
            // Never allocate a string bigger than max_size.
            if (__capacity > max_size())
                __capacity = max_size();
            ...
        }
    }
```

Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

```
template<typename _Ch_type, class _Alloc, class _Rx_traits>
inline bool
regex_search(const _Ch_type* __s,
              match_results<const _Ch_type*, _Alloc>& __m,
              const basic_regex<_Ch_type, _Rx_traits>& __e
              regex_constants::match_flag_type __f,
              = regex_constants::match_default)
{ return regex_search(__s, __s + _Rx_traits::length(__s), __m,
__e, __f); }
```

Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

```
template<typename _Ch_type, class _Alloc, class _Rx_traits>
inline bool
regex_search(const _Ch_type* __s,
             match_results<const _Ch_type*, _Alloc>& __m,
             const basic_regex<_Ch_type, _Rx_traits>& __e
             regex_constants::match_flag_type __f,
             = regex_constants::match_default)
{ return regex_search(__s, __s + _Rx_traits::length(__s), __m,
__e, __f); }
```

To call non-const
method

Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

Why is there no Out?



Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

Use return



Parameter Passing

Rule of thumb

Size	In	In-Out
Small	int	int &
Large	const string &	string &

What if there are multiple values?



Returning from Functions

Rule of thumb

```
template< class T >  
void minmax(std::initializer_list<T> ilist,  
            T& min, T& max);
```

Returning multiple values

Violates SOLID?

- single responsibility principle

Returning from Functions

Rule of thumb

Returning multiple values

Violates SOLID?

- single responsibility principle

Refactoring

```
template< class T >  
void minmax(std::initializer_list<T> ilist,  
           T& min, T& max);
```

→

```
template< class T >  
T min(std::initializer_list<T> ilist);
```

```
template< class T >  
T max(std::initializer_list<T> ilist);
```

Returning from Functions

Rule of thumb

Returning multiple values

Does not violate SOLID

Use `std::pair`/`std::tuple`

```
template< class T >  
void minmax(std::initializer_list<T> ilist,  
           T& min, T& max);
```

→

```
template< class T >  
std::pair<T, T> minmax(  
    std::initializer_list<T> ilist );
```



Class/Struct

Put all related things (data, functions) together

- Represents objects in OOP
- almost everything should belong to a class

No real difference except for default visibility, inheritance, ...

- **class** – by default everything **private**
- **struct** – by default everything **public**

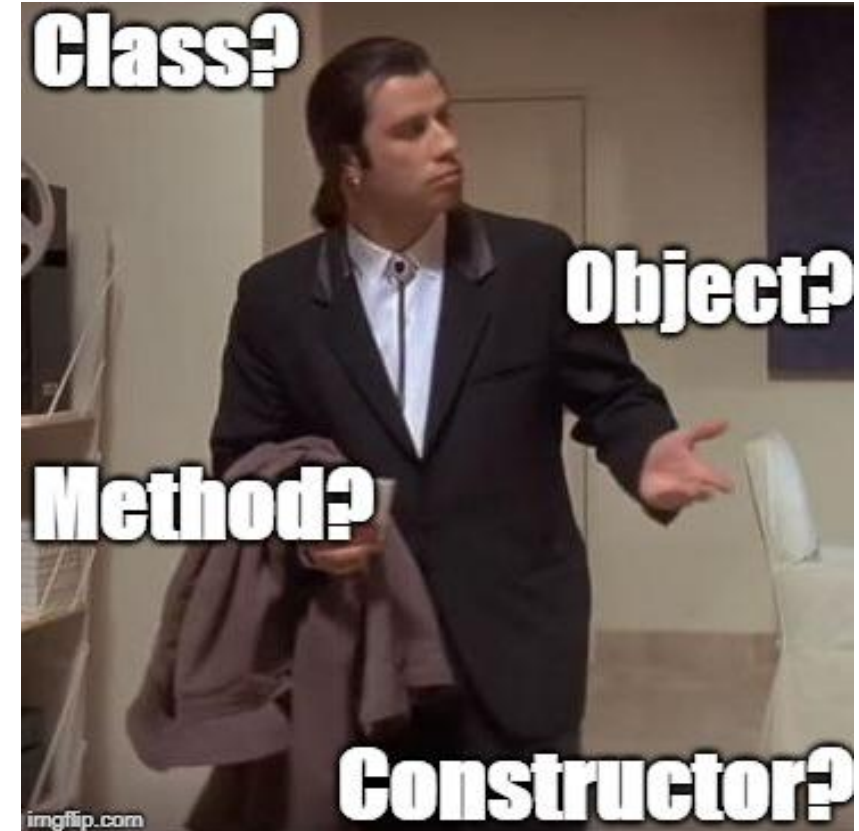
Internal things → **private**

- **protected** if need access from a child

Read-only functions → **const**

- const-correctness

Special methods (**constructor**, destructor, ...)



Class Example

```
class calculator {  
    // by default everything is private  
    void sum();  
    void subtract();  
  
public:  
    calculator() { /* default ctor */ }  
    calculator(const std::string &str) () {  
        /* ctor */  
    }  
    void calc(const std::string &str);  
    void print_result() const;  
  
private:  
    void multiply();  
  
protected:  
    void init();  
};
```

can be used
multiple times

semicolon
at the end!

```
calculator c; // no need for new  
c.calc("1+2-3");  
c.print_result();  
  
// calling non-default ctor  
calculator c2("1+2-3");  
c2.print_result();  
  
// creating a vector  
std::vector<calculator> calcs;
```

Class vs. Struct

Use `class` if the class has an invariant;

Use `struct` if the data members can vary independently

```
struct coordinate {  
    int x;  
    int y;  
    int z;  
  
    coordinate();  
    coordinate(int x);  
    coordinate(int x, int y);  
    coordinate(int x, int y, int z);  
  
    void set(int x, int y, int z);  
};
```

Dynamic Array - `std::vector<T>`

Beware of time complexity of individual methods

`vector<bool>` optimization

```
#include <vector>
int main() {
    std::vector<int> vi{1, 2, 3, 4, 5, 6};           // [1, 2, 3, 4, 5, 6]
    std::vector<float> vf(5, 0.0f);                 // [0.0, 0.0, 0.0, 0.0, 0.0]
    std::cout << vi[3] << " " << vf.at(3) << std::endl; // access the 4th! element
    std::cout << vi.size();
    vi[3] = 100; vi.at(6) = 600;                    // access the 4th and 7th element
    vf.push_back(100.0f); vf.emplace_back(200.0f); // insert at the end
    vf.emplace_back(200.0f);                        // create element at the end
    vf.insert(3, 300.0f); vf.emplace(3, 300.0f); // insert at the specific place
    vf.emplace(3, 300.0f);                          // create element at the specific place
    vi.pop_back();                                   // erase the last element
    vf.erase(2);                                    // erase the 3rd element
    vi.clear();                                     // clear whole container
    vi.reserve(10);                                // reserve space(=memory) for 10 elements
    vi.resize(10);                                 // actually create 10 elements using default ctor
}
```

3D Matrix for Integers – minimal API

```
ctor(), ctor(x, y, z)
set(x, y, z, value), get(x, y, z), print()
set_width(), set_length(), set_height(), get_width(), get_length(),
get_height()
get_matrix(x), get_matrix(y), get_matrix(z)
get_vector(x, y), get_vector(y, z), get_vector(x, z)
clear() // set all values to 0 (zero)
fill_with_value(value) // set all values to a given value
num_zeros(), num_negatives(), num_positives(); // number of zeros/negatives/...
```

rename/adjust functions if needed – but keep the semantic

Submit the final version to Gitlab → 2 points

- **Deadline: Thursday October 16th 5:00**

3D Matrix for Integers - Hints

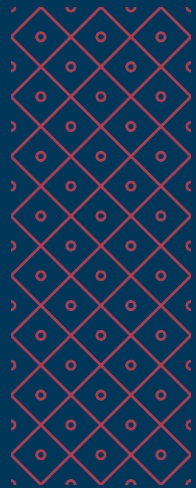
Think about the design

- Compose: array $\rightarrow$ matrix $\rightarrow$ 3-D matrix $\rightarrow$ 4D matrix $\rightarrow$... $\rightarrow$ XD matrix
- Design simple first, then continue to the next level

No need to focus too much on performance yet

Focus on:

- Passing arguments: const-references, references, ...
- const functions
- class design: decomposition into functions, function reusing, private/public, ...



Lab 1

3/10/2025



Basic information

Email: tomas.faltin@matfyz.cuni.cz

Labs web: <https://fan1x.github.io/teaching/2025-cpp>

Lecture web: <https://www.ksi.mff.cuni.cz/teaching/nprg041-web/>

Mattermost

- Invite link in [SIS/Notice-board](#)
- Channel: `2526/nprg041-cpp-faltin`
- DM: @faltin.tomas

Gitlab

- <https://gitlab.mff.cuni.cz/>
- <https://gitlab.mff.cuni.cz/teaching/nprg041/2025-26/klepl>



Labs Credit

Small (home) works assignments

- Submitted either to ReCodex or Gitlab
- At least 20 out of 30 points required
- Points contributed to the final exam grade

Credit project



Communication is the key

Don't be afraid to ask

Be proactive

- via email
- on Mattermost (instant)
 - DM if related to you only
 - Into a channel if others can benefit from it

If you struggle with something

If you feel like you might miss a deadline

Try things

Research project at our University

- Contact professor directly

Internships in companies

Erasmus

Conferences

...



Or not...

"There's a tiger in you"

Tiger in me :



MemeZilla.com



Let's get it started

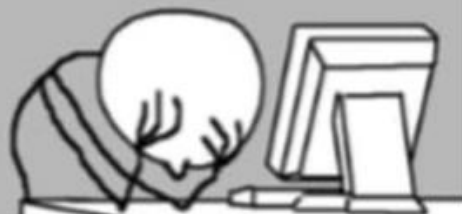
Motivation

Days before OpenAI

Developer coding
- 2 hours



Developer debugging
- 6 hours

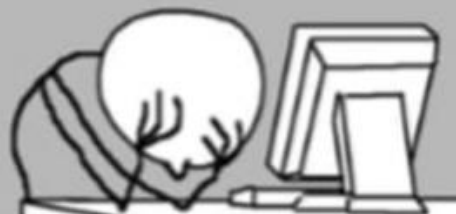


Days after OpenAI

ChatGPT generates
Codes - 5 min



Developer debugging
- 24 hours



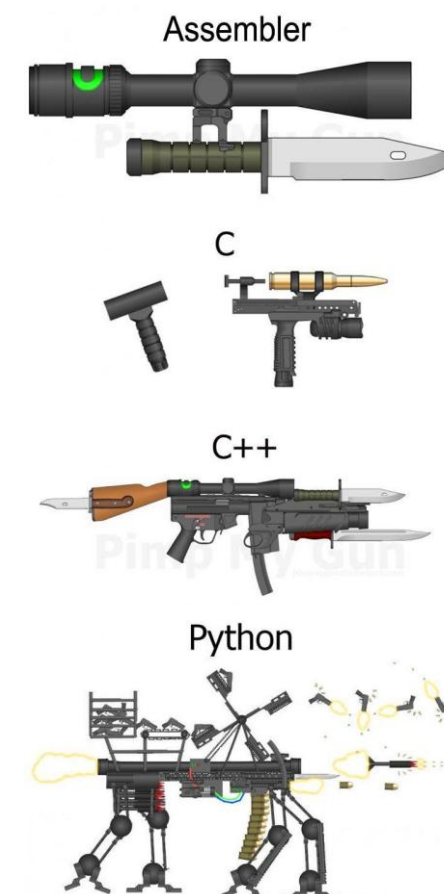
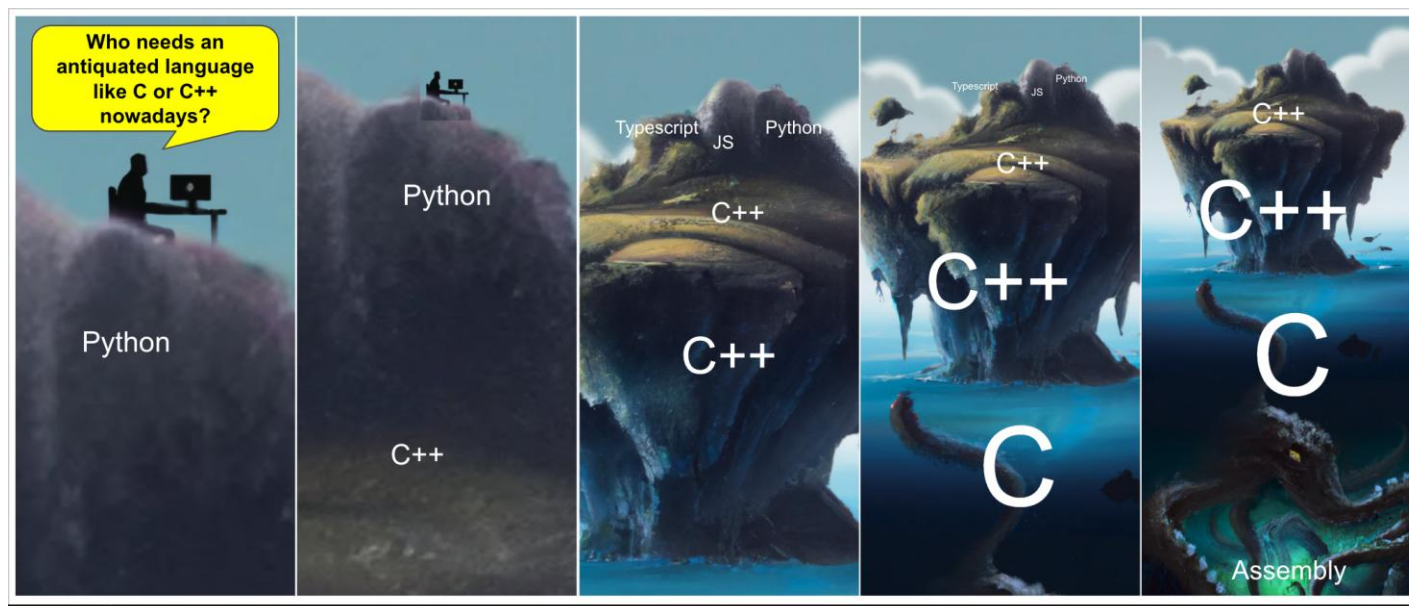
ProgrammerHumor.io



Why C++

“C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg.”

-- Bjarne Stroustrup





C++ (interesting) links

Reddit, Slack, ...

<https://en.cppreference.com/w/>

<http://www.cplusplus.com/>

<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>

<https://www.youtube.com/user/CppCon>

<https://isocpp.org/>

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/>

<https://godbolt.org/>

...



Working Environment

Use anything you like
IDEs

- Visual Studio
 - License for students at <https://portal.azure.com/...>
- VS Code
- Clion
- Code::Blocks
- Eclipse
- ...

Compilers

- MSVC, GCC, Clang+LLVM, ICC, ...



Quick Recap



Code Requirements

Absence of Unforgivable Curses

- <https://teaching.ms.mff.cuni.cz/nswi170-web/pages/labs/coding/>

1. Code decomposition
2. Using constants
3. DRY
4. No global variables
5. Encapsulation

Code Requirements - Consistency

Consistency

- Be consistent within the code
- keep a single code style



Ministry Of Dev, PhD

@UdellGames

Folgen



Use whatever brace style you prefer.

But not this.

Don't do this.

Seek help instead of this.

```
public class Permuter
{
    private static void permute(int n, char[] a)
    {
        if (n == 0)
        {
            System.out.println(String.valueOf(a));
        }
        else
        {
            for (int i = 0; i <= n; i++)
            {
                permute(n-1, a);
                swap(a, n % 2 == 0 ? i : 0, n);
            }
        }
    }
    private static void swap(char[] a, int i, int j) {
        char saved = a[i];
        a[i] = a[j];
        a[j] = saved;
    }
}
```

Code Requirements – Readability

Code doesn't contain commented/dead parts
Code should be readable on its own
Comment complicated code



Code Requirements – Safe, Modern

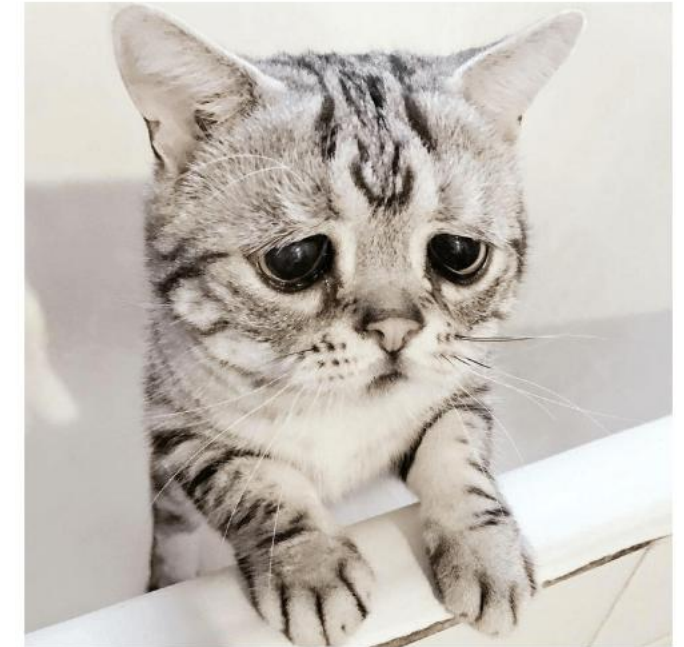
Prefer using modern constructs

Additional safety

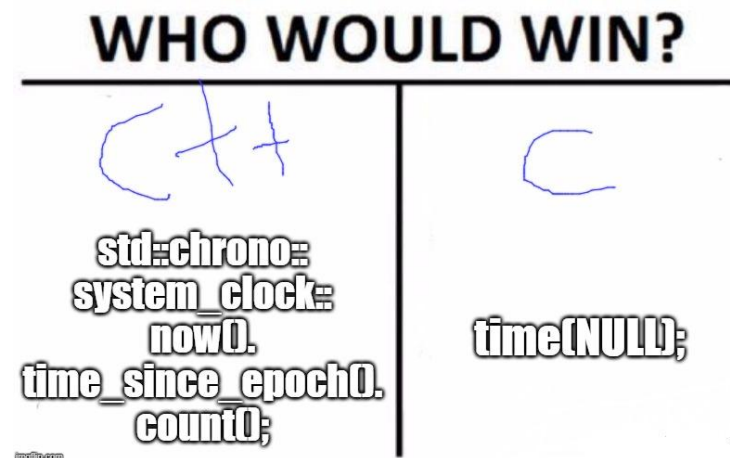
Maybe performance

E.g., prefer `std::vector<int>` to `new int[]`

Me when I realized that I can't pass 2D arrays to functions in C/C++ as `int a[]`:



"Pointers are a nuisance"



Code Requirements – Working

OFC, if the code is not working, all the above points are not that important they will help you with debugging at least 😊



Enough talking





Hello World

```
#include <iostream>
```

```
#include <string>
```

```
int main() {
```

```
    std::string name;
```

```
    std::cin >> name;
```

```
    std::cout << "Greetings from " << name << std::endl;
```

```
    return 0;
```

```
}
```



Hello World

```
#include <iostream>
#include <string>
```

```
int main() {
    std::string name;
    std::cin >> name;
    std::cout << "Greetings from " << name << std::endl;
    return 0;
}
```

Include the libraries
which implements the
used STL constructs
(string, cin, cout)

The main entry
point/function for all
programs. The
execution starts here

Declare a
variable of type
string

All the STL
constructs live
inside `std`
namespace

Write to
standard output
(screen)

Read from
standard input
(keyboard)



Compilation

c++ --version

- c++ is a compiler, here GCC

c++ hello_world.cpp -o hello_world

- Compile program into `hello\_world` executable (using default settings)

c++ -Wall -Wextra -Werror -O3 -std=c++2b hello_world.cpp -o hello_world

- Wall: Show all warnings
- Wextra: Show additional extra warnings
- Werror: Thread all warnings as errors
- O3: level of optimizations
- std=c++2b: Used C++ standard

Or use IDE ☺



More Complex Program

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

void pretty_print(const vector<string>& args) {
    // ... args[i]
}

int main(int argc, char** argv) {
    vector<string> args(argv, argv+argc);
    pretty_print(args);
    return 0;
}
```

More Complex Program

```
#include <iostream>
#include <string>
#include <vector>
```

Include the
whole std
namespace

```
using namespace std;
```

Passing the
argument by
(const)
reference

```
void pretty_print(const vector<string>& args) {
    // ... args[i]
}
```

Number
of
argument

Arguments of
the program on
the command
line

```
int main(int argc, char** argv) {
    vector<string> args(argv, argv+argc); // Wrap arguments
    pretty_print(args);
    return 0;
}
```

Transform
“magically” the
arguments into
C++ array of
strings

Functions And Parameters

```
int get_max(int v1, int v2) {  
    return v1 > v2 ? v1 : v2;  
}
```

```
int get_max1(const vector<int> &ints) {  
    int max = std::numeric_limits<int>::min();  
    for (int x : ints) {  
        max = get_max(x, max);  
    }  
    return max;  
}
```

```
bool get_max2(const vector<int> &ints, int &max) {  
    max = std::numeric_limits<int>::min();  
    for (int x : ints) {  
        max = get_max(x, max);  
    }  
    return !ints.empty();  
}
```

```
std::tuple<bool, int> get_max3(const vector<int> &ints) {  
    int max = std::numeric_limits<int>::min();  
    for (int x : ints) {  
        max = get_max(x, max);  
    }  
    return { !ints.empty(), max };  
}
```



Functions And Parameters

read-only input parameter

- Most of the types (string, vector, ...) → use const-reference - **const &**
 - `int get_max(const vector<int> &ints)`
- For small numeric types (int, float, double, ...) → use **direct parameter**
 - `int get_max(int v1, int v2)`

output parameters

- Single output parameter → use **return** value
 - `int get_max(const vector<int> &ints)`
- Few output parameters → use **tuple/pair/structure**
 - `std::tuple<bool, int> get_max(const vector<int> &ints)`
- Many output parameters → use reference - **&**
 - `bool get_max(const vector<int> &ints, int &max)`



Work

1. Hello World
2. A greeting program (use names from arguments)
 - ``hello.exe Adam Eve`` → ``Hello to Adam and Eve``
 - What is inside `args[0]`?
3. Summation of numbers from arguments
 - ``sum.exe 1 2 3 4 5`` → ``15``
 - ``stoi()`, ``stod()`, ``stoX()`
 - Functions for transformation from **string** to **<something>**
4. **A simple calculator (only for operations +-)**
 - ``calc.exe 1+2+3-4`` → ``2``
 - The previous programs are not needed, but they should give you a lead
 - **Submit the final version to Gitlab → 3 points**
 - **Deadline: Thursday October 9th 5:00**