Charles University

Programming in C++

Labs - winter 2025/26

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

Lab 2

10/10/2025

Homeworks

Comments in Gitlab

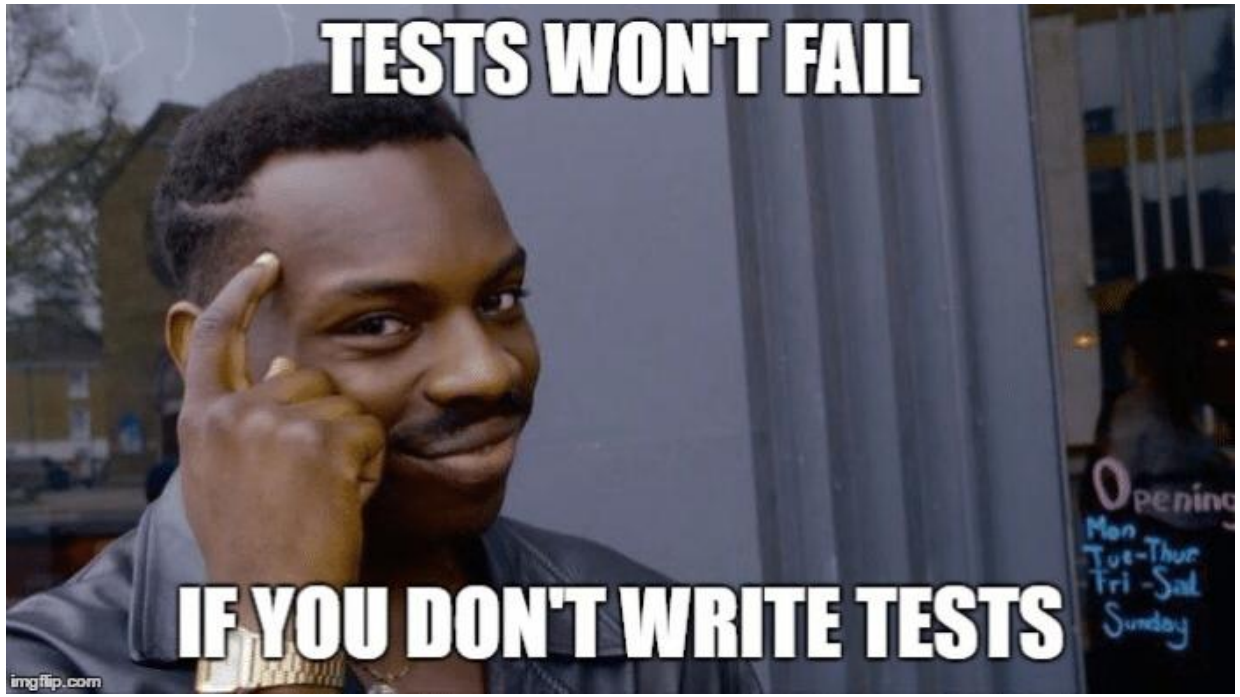
Points can be found in “Studijní mezivýsledky”

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:

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 &`**

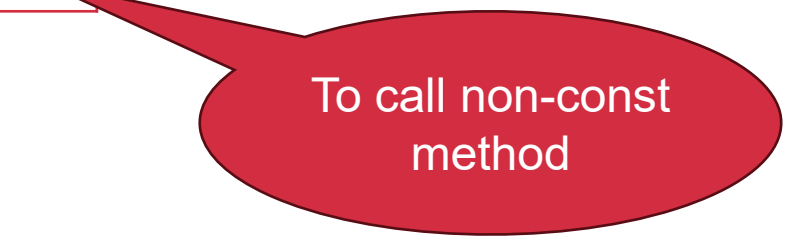
```
template<typename _Ch_type, class _Alloc, class _Rx_traits>
    inline bool
    regex_search(const _Ch_type* __s,
                 match_results<const _Ch_type*, _Alloc>& __m,
                 const basic_regex<_Ch_type, _Rx_traits>& __e
                 regex_constants::match_flag_type __f,
   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

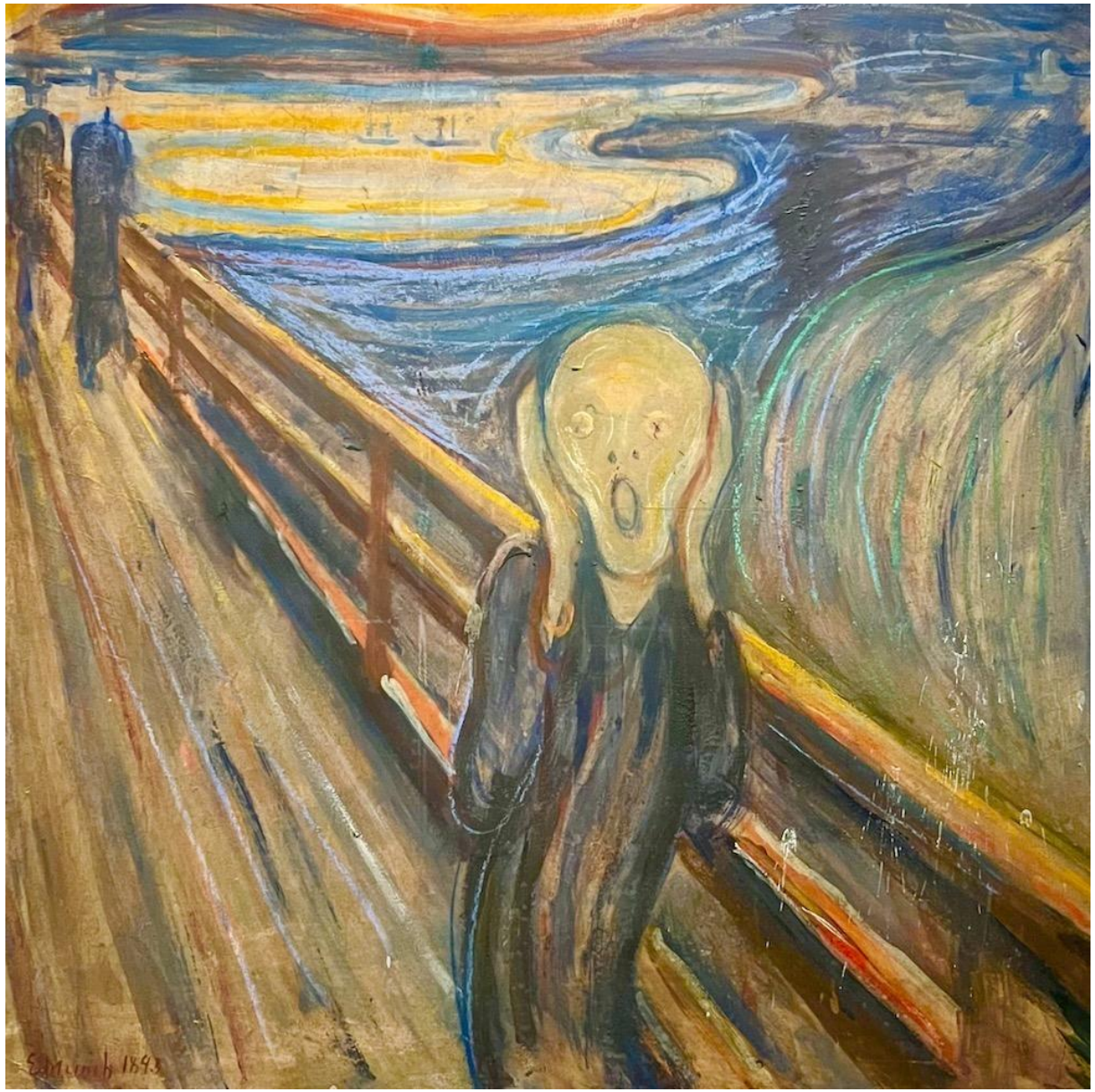


Parameter Passing

Rule of thumb

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

Why is there no Out?

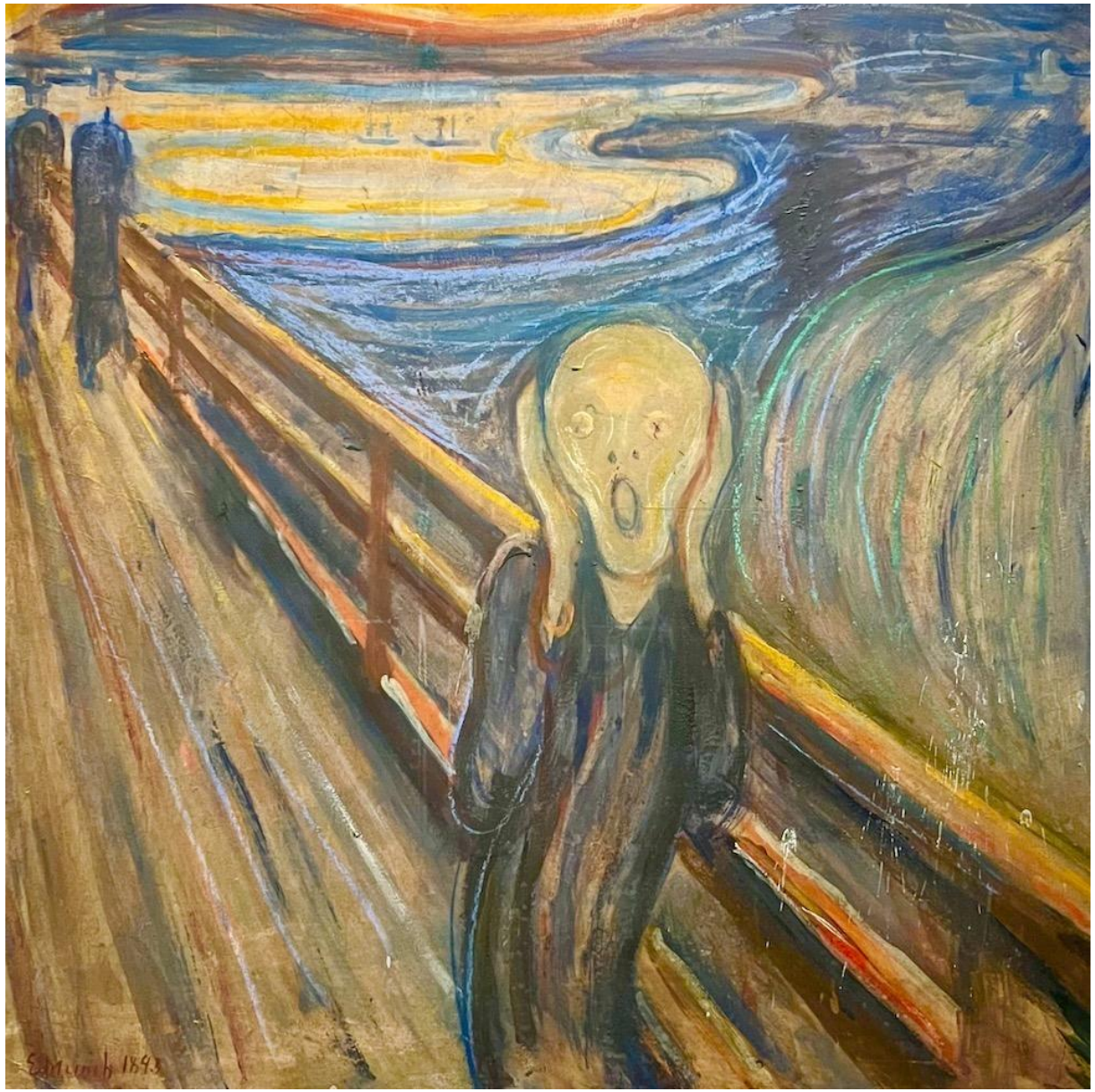


Parameter Passing

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

Returning multiple values

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

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

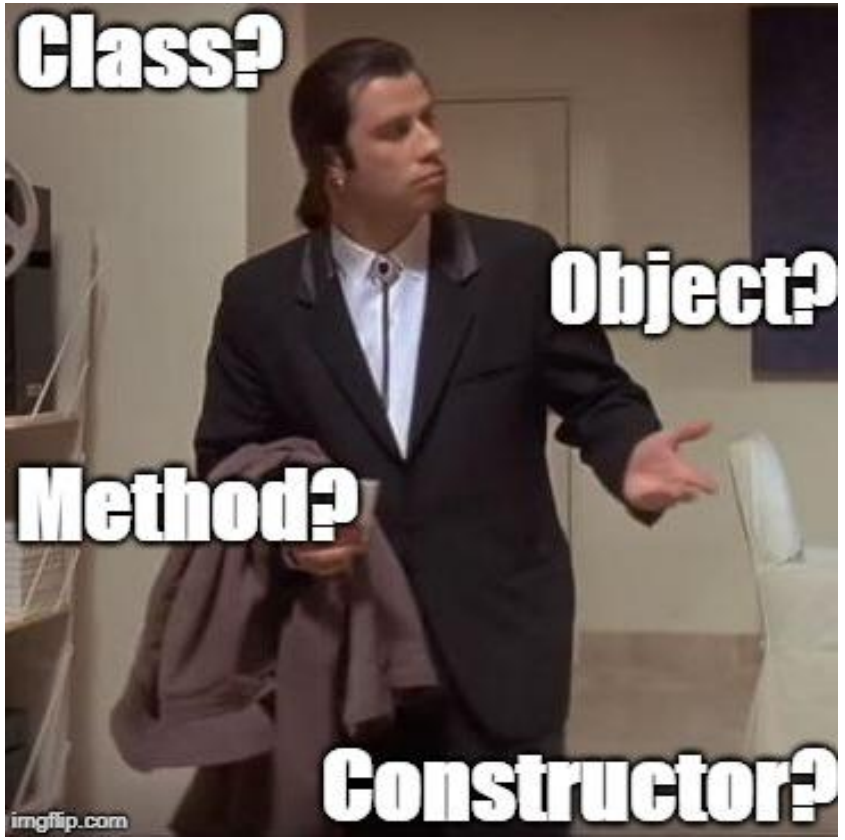
Internal things → private

- 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 substract();

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");
c.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_heigth(), 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 sematic

Submit the final version to Gitlab → 2 points

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

3D Matrix for Integers - Hints

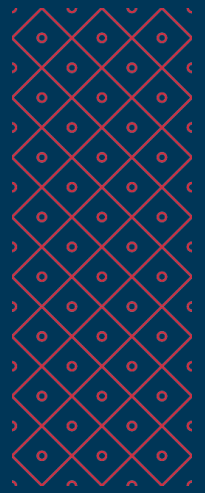
Think about the design

- Compose: array → matrix → 3-D matrix → 4D matrix → … → 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…

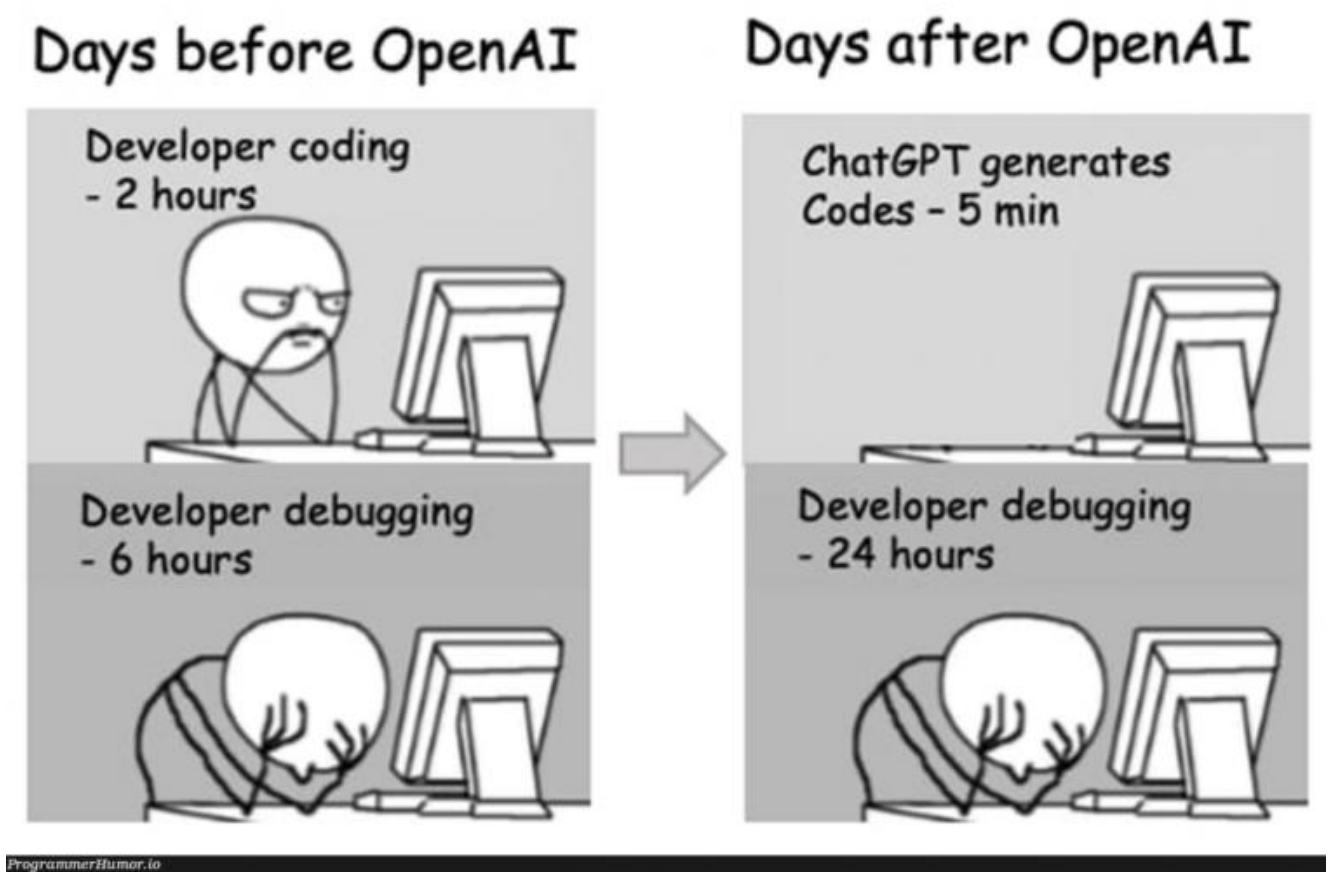
"There's a tiger in you"

Tiger in me :

Charles University

Let’s get it started

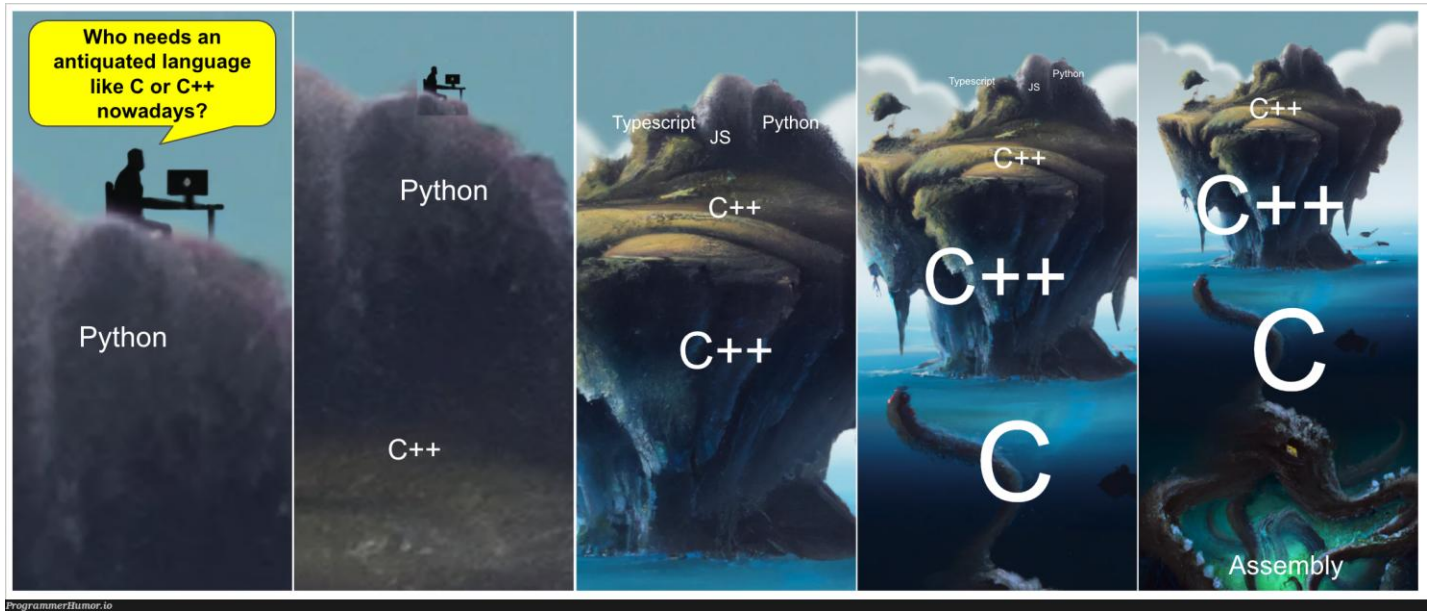
Motivation


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


WAITING FOR A RESPONSE FROM
MY SUPPOSEDLY SMART AI AGENT
imgflip.com

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



Charles University

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

Charles University

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, …

Charles University

Quick Recap

Charles University

Code Requirements

Absence of Unforgivable Curses

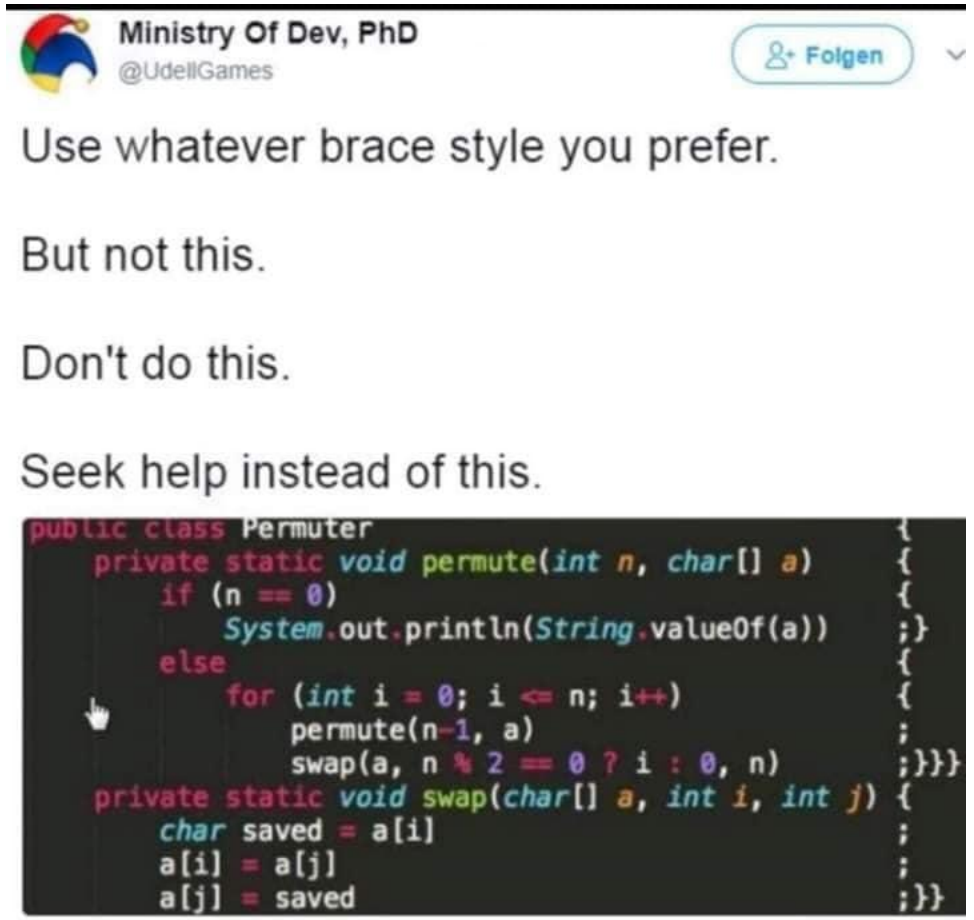
- https://teaching.ms.mff.cuni.cz/nswi170-web/pages/labs/coding/

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

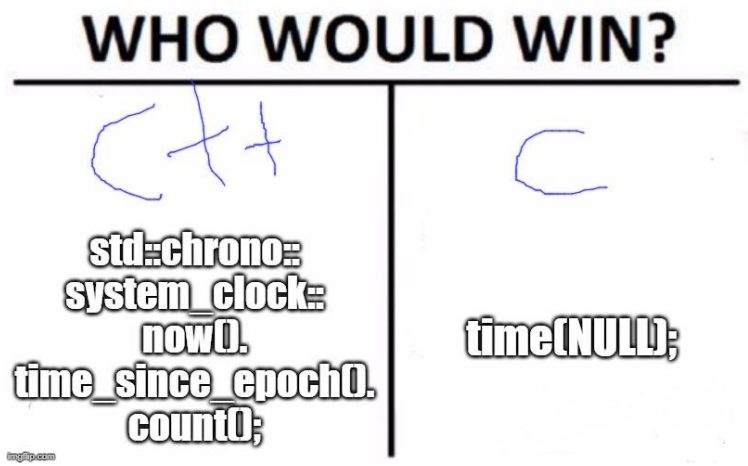


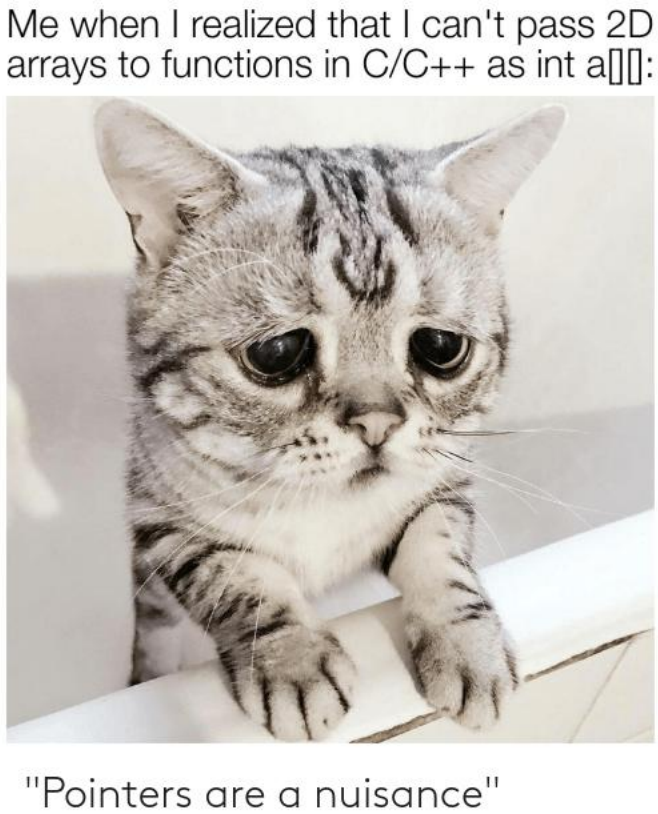
Code Requirements – Readability

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[]`

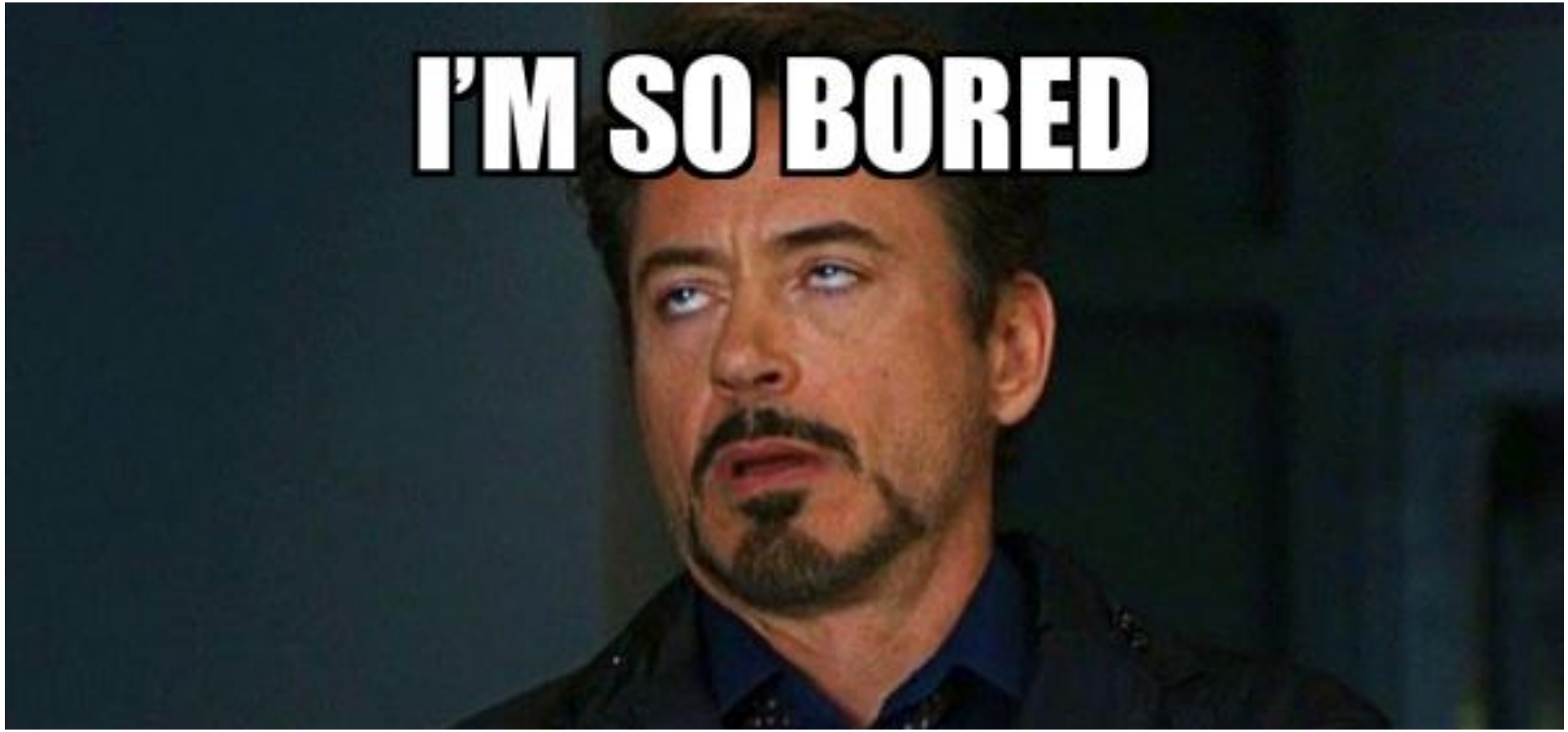




Code Requirements – Working

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;
}
```

Charles University

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

Read from standard input (keyboard)

Write to standard output (screen)

All the STL constructs live inside `std` namespace

Compilation

Charles University

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

Charles University

```
#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>

using namespace std;

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

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

Include the whole std namespace

Passing the argument by (const) reference

Number of argument

Arguments of the program on the command line

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

Charles University

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 **s**tring **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**