# Programming in C++

Tomáš Faltín (covering for Jiří Klepl)
tomas.faltin@matfyz.cuni.cz
https://fan1x.github.io/

# Basic information

- Email: klepl@d3s.mff.cuni.cz
- Labs web: https://teaching.mff.cuni.cz/nprg041-klepl-web/index.html
- Lecture web: https://www.ksi.mff.cuni.cz/teaching/nprg041-web/
- Mattermost
  - Invite link in SIS/Notice-board
  - Channel: `2526/nprg041-cpp-klepl`
  - PM: @jiriklepl
- Gitlab
  - https://gitlab.mff.cuni.cz/
  - https://gitlab.mff.cuni.cz/teaching/nprg041/2025-26/klepl
- Info about labs credit next week

# Labs Credit

- Next week

# 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

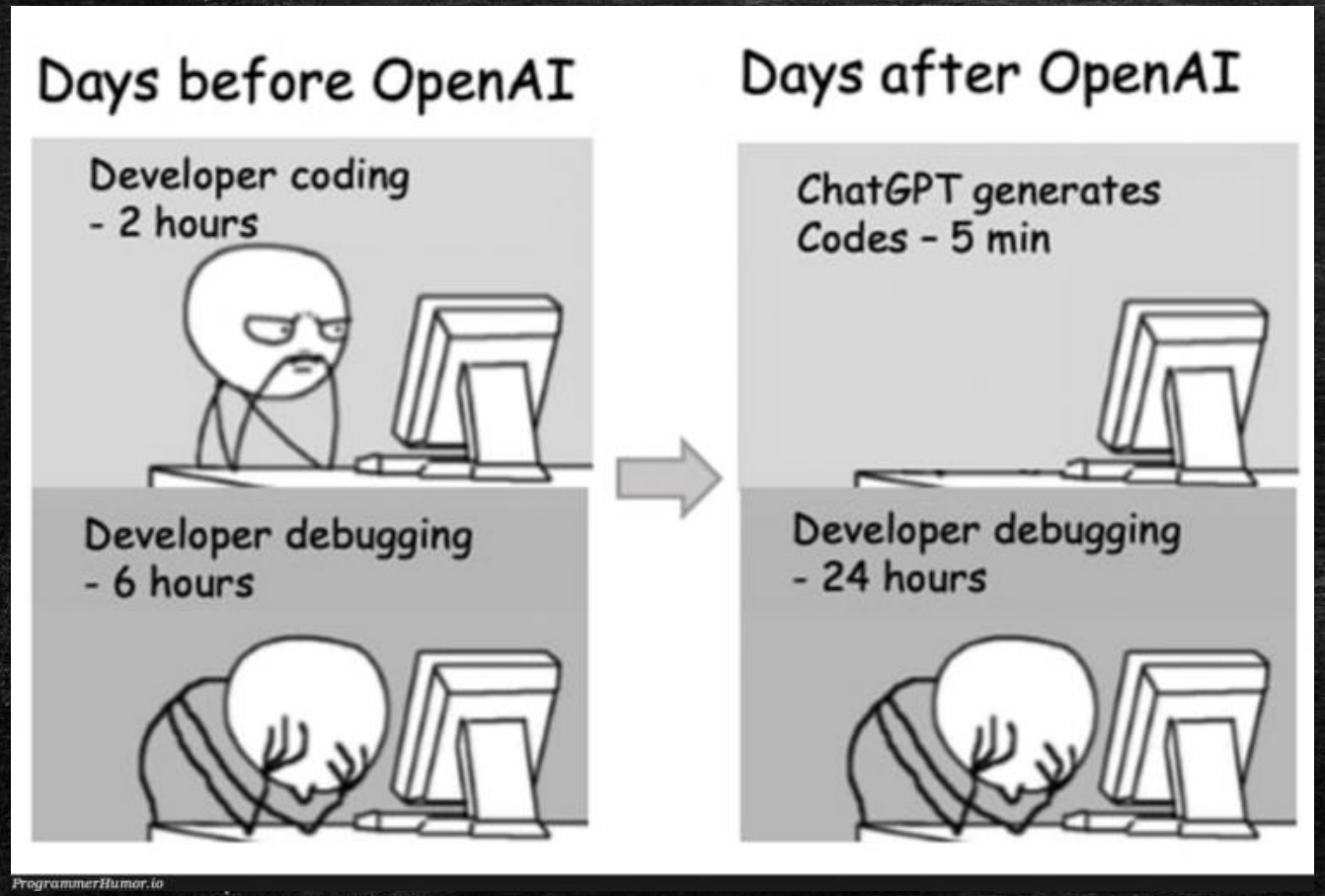
# Try things

- Research project at our University
  - Contact professor directly
- Internships in companies
- Erasmus
- Conferences
- …

# Or not…

# Let’s get it started

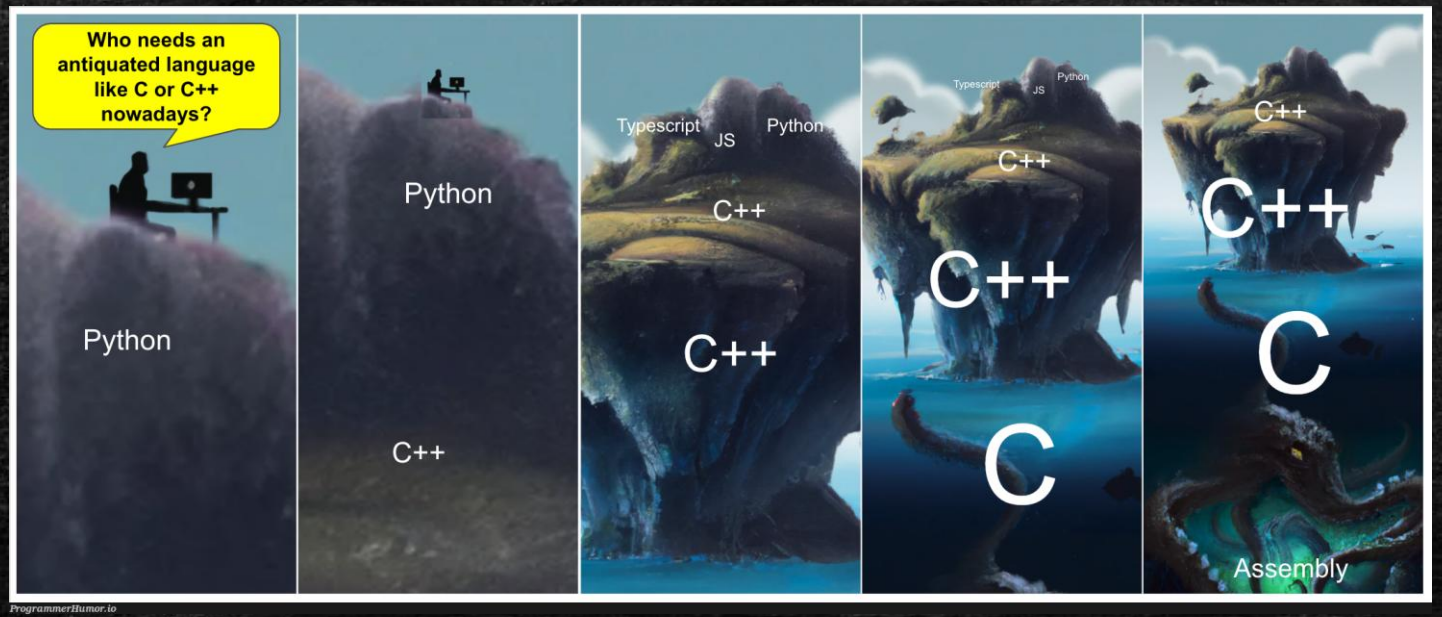
# Motivation


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




# 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

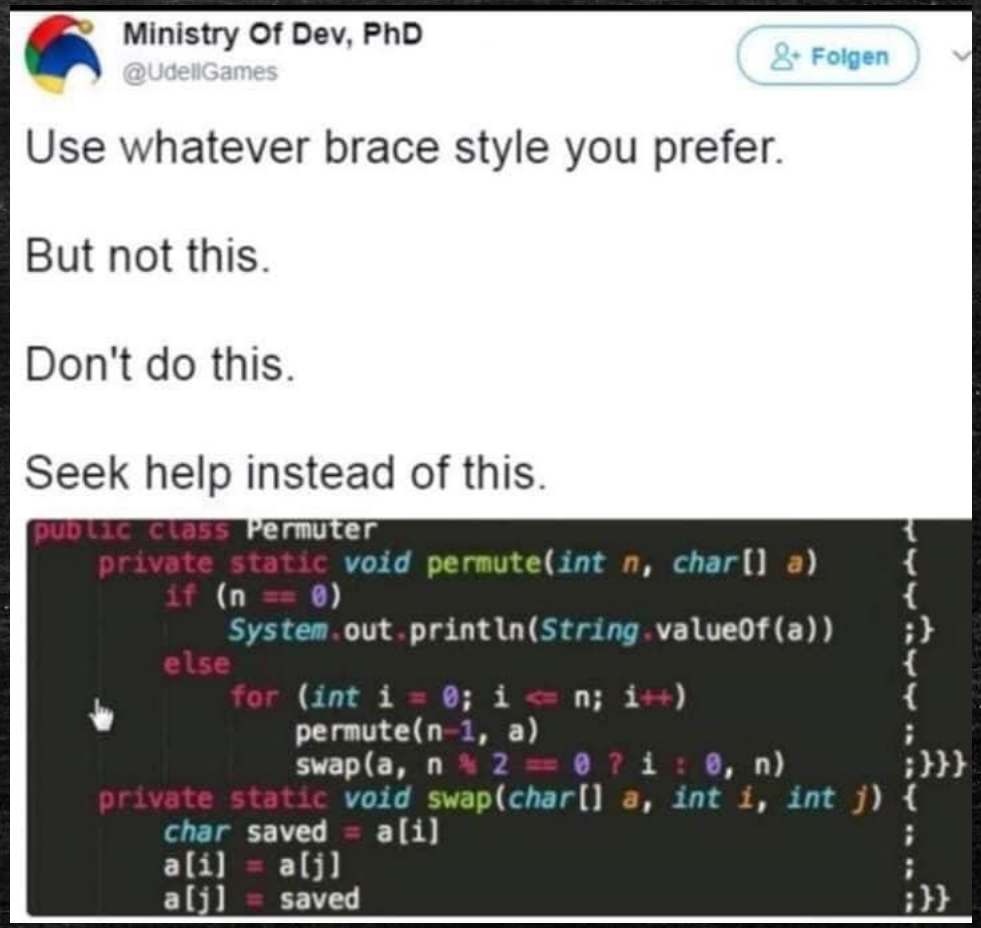
Absence of Unforgivable Curses

- 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

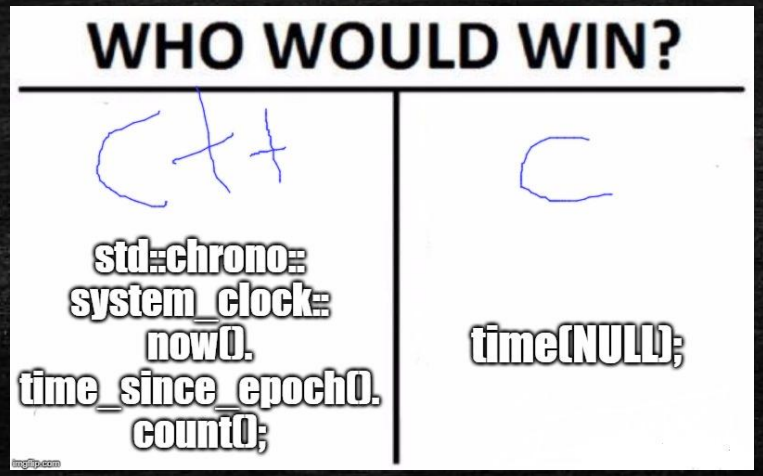


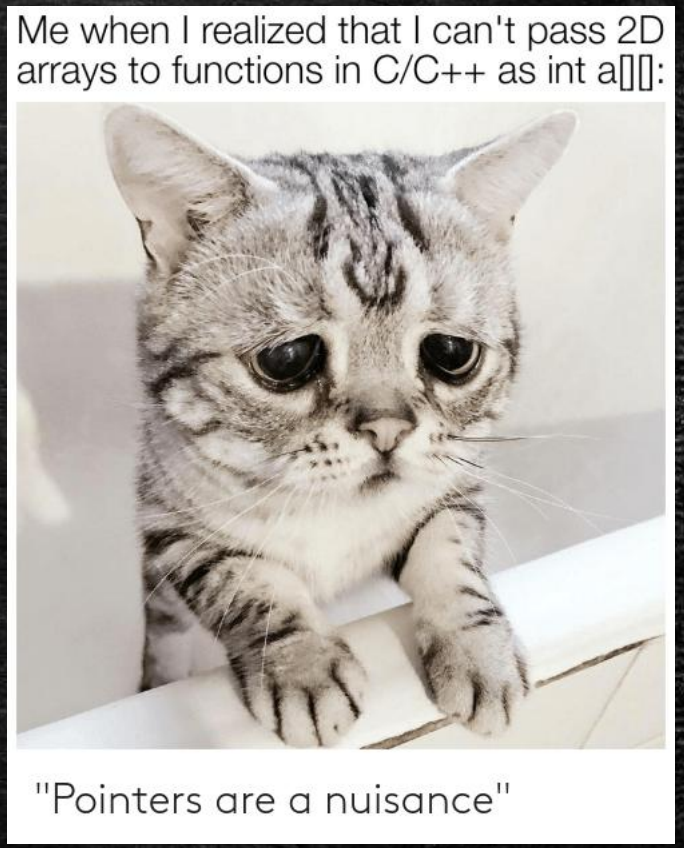
# Code Requirements – Readability

- 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;
}
```

# 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

- `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>

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 arguments

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

- 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`
   - to Gitlab
   - The previous programs are not needed, they should give you a lead