Charles University

Programming in C++

Labs - winter 2025/26

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

Lab 9

28/11/2025

Credit Project

Merge-request with detailed project description by **Friday December 12th**

- E.g., 1-2 pages **README.md**
- Describe used **libraries:** what they can/cannot do
- The programmed **functionality**
- The programmed **functionality** from user perspective

Homework Discussion

Templates

```
class complex {
  int real_; int im_;
public:
  int &re() { return real_; }
  int re() const { return real_; };
  int &im() { return im_; };
  int im() const { return im_; };
};

void print(int v) { cout << v; }
void print(double v) { cout << v; }
void print(const string &str) { cout << str; }

int main() {
  complex c;
  print(c.re());
}
```

```
template<typename T>
class complex {
  T real_; T im_;
public:
  T &re() { return real_; }
  T re() const { return real_; };
  T &im() { return im_; };
  T im() const { return im_; };
};

template<class T>
void print(const T &v) { cout << v; }

int main() {
  complex<int> c;
  complex<double> c;
  print(c.re());
}
```

Equivalent to using auto:
`void print(const auto &v)`

Static instantiation:
Create two different types

Templates

class = template

Can we somehow restrict allowed types? E.g., no complex<string>

```
template<class Cont>
class container_wrapper {
  Cont cont;

public:
  void push_back(
      const typename Cont::value_type& val) {
    cont.push_back(val);
  }

  void push_back(typename Cont::value_type &&rval) {
    cont.push_back(std::move(rval));
  }
};

template<class Cont>
auto make_wrapper() {
  return container_holder<Cont>();
}
```

A type or a static member? → Hint the compiler

```
auto vector_wrapper =
  make_wrapper<std::vector<int>>();
vector_wrapper.push_back(3);
```

Troubleshooting: Cannot move template definition into a source file (*.cpp)

Reason
ODR: One Definition rule
Template instantiation requires definition

Solution
Opt1: split into declaration + definition, but in the same header

Opt2: move definition into *.tpp + include this *.tpp file from the header

Concepts*

```
#include <concepts>

template<std::integral T>
class Complex {
    T real_; T im_;
public:
    T &re() { return real_; }
    T re() const { return real_; };
    T &im() { return im_; };
    T im() const { return im_; };
};
```

satisfied iff T is an integral type

Specify a new concept

Can be combined with basic bool operators

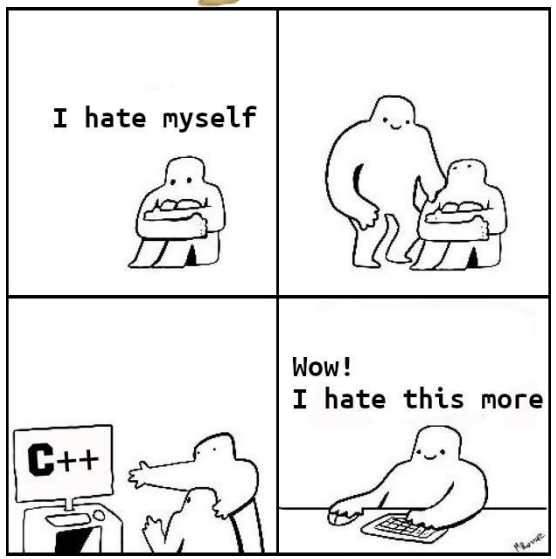
```
template<class T>
concept UnsignedIntegral =
  std::integral<T>
  && !std::signed_integral<T>;
```

Some other examples

```
// a type is derived from another type
std::derived_from<Derived, Base>
// an object of a type can be moved
std::movable<T>
...
```

Concepts** (Advanced C++)

```
template<class T>
struct IsIntegral : std::bool_constant<
    // T* parameter excludes reference types
    requires (T t, T* p, void (*f)(T))
{
  // Exclude class types
  reinterpret_cast<T>(t);
  // Exclude enumeration types
  f(0);
  // Exclude everything but integral types
  p + t;
}> {};

template<class T>
concept Integral = IsIntegral<T>::value;
```



Work: Change your containers to support any type

Polymorphic vector (lab 8)
vector<T> (lab 6)
linked_list<T> (lab 6)
Matrix3D (lab 4)

Add missing operations
`push_back, insert, pop_back, erase`

Hints
Use templates

Submit the final version to Gitlab

- **Each container → 1 point (4 in total)**
- **Create a *merge-request***
- **Deadline: Thursday December 4th 5:00**

Bonus: Improve performance of polymorphic vector (→ 2p)
Not all inserts require dynamic allocation
Add benchmark measurements (insert + iteration)

Hints
Buffer allocation of same type together

Lab 8

21/11/2025

Credit Project

Merge-request with detailed project description by **Friday December 12th**

- E.g., 1-2 pages **README.md**
- Describe used **libraries:** what they can/cannot do
- The programmed **functionality**
- The programmed **functionality** from user perspective
- I'll put an example and recommendation on my website

Survey Discussion

Big thank you!

1. **More specific assignments**
2. Labs on work with files
3. Be more in sync with the lectures
4. ReCodex tests
5. Less granular points, ping-pong the solutions
6. Too much time spent on the homeworks

Survey Discussion

Big thank you!

1. **More specific assignments**
2. Labs on work with files
3. Be more in sync with the lectures
4. ReCodex tests
5. Less granular points, ping-pong the solutions
6. Too much time spent on the homeworks

Survey Discussion

Big thank you!

More specific assignments

Motivation
real-world

I'll try to make it more specific
Email/DM me with the specific questions – I'm the stakeholder
Comment with how you understood it is fine

Survey Discussion

Big thank you!

Labs on work with files

There is going to be one in 2 weeks

Survey Discussion

Big thank you!

Be more in sync with the lectures

Motivation

Get quickly hands on *important* stuff

Here the things multiple times from different people

I'll try to be more in sync

Survey Discussion

Big thank you!

Use ReCodex

Motivation

Real-world (testing is part of the job)

I (personally) don’t like Recodex: hard to debug failures

MR approach better for students

Survey Discussion

Big thank you!

Less granular points, possibility to ping-pong the solutions

Motivation

We already can ping-pong using MR

Points are less granular on purpose to motivate fix issues

Survey Discussion

Big thank you!

Too much time spent on the assignments

Motivation

Prepare you for the practical exam

I'll try to reevaluate the complexity of the assignments

(The deadline is not always strictly enforced)

Use AI to help you with the search/code

- *Always understand the generated code, don't just copy*

Homework Discussion

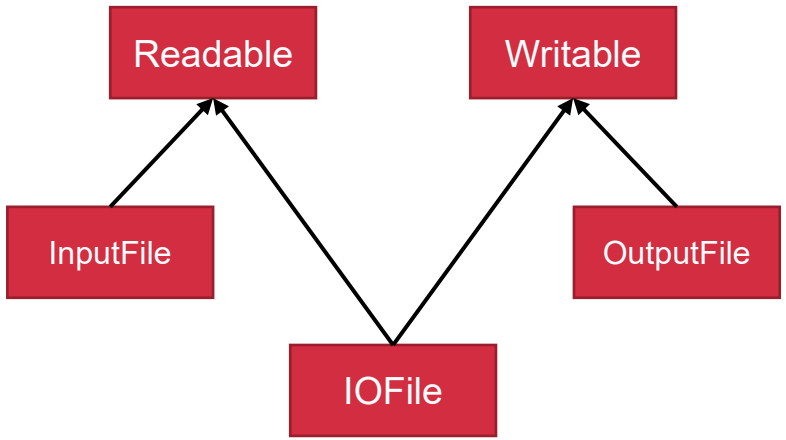
I’ll go through the merge-requests (new and old) by end of the weekend

Inheritance

To inherit (=have access to) attributes and methods of the ancestor class

Interface implementation
Typically use of abstract classes

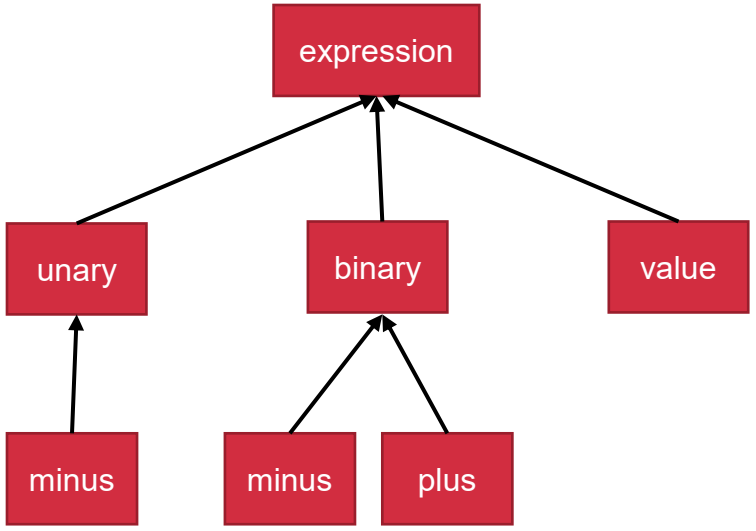
```
class Readable {};
class InputFile : public Readable {};

class Writable {};
class OutputFile : public Writable {};

class IOFile: public Writable, public Readable {};
```



Inheritance

To inherit (=have access to) attributes and methods of the ancestor class

IS-A hierarchy
Might inherit some common attributes



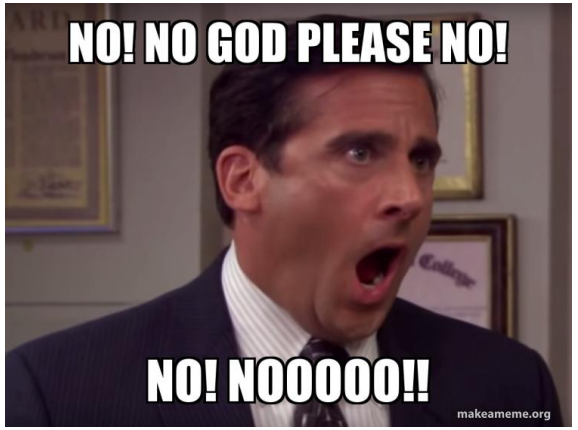

```
class expression {};
class binary : public expression {};
class plus : public binary {};

class car {};
class volvo : public car {};
class skoda : public car {};

class animal {};
class dog : public animal {};
class cat : public animal {};

class employee {};
class accountant : public employee {};

class object {};
class triangle : public object {};
class polygon : public object {};
```

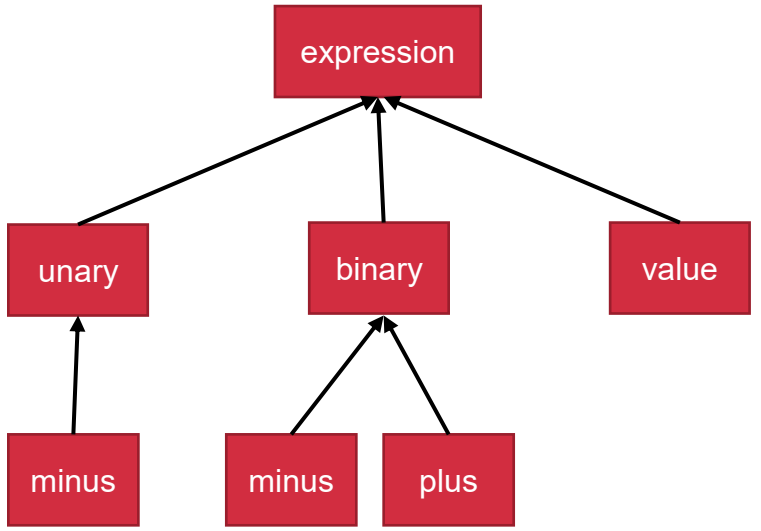
Inheritance: Rule-of-Thumbs

Think about the API

Other inheritance usage typically means bad design

`if-else` → bad OOP design





Don't forget to `break` in `switch`

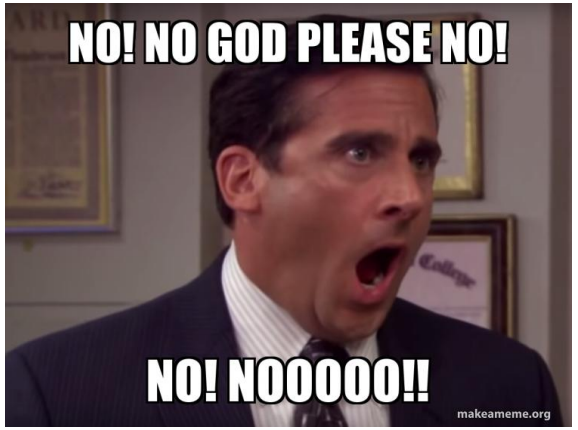
```
switch (type) {
  case ExprType::MINUS:
    // something
    break;
  case ExprType::PLUS:
    // something else
    break;
  ...
```

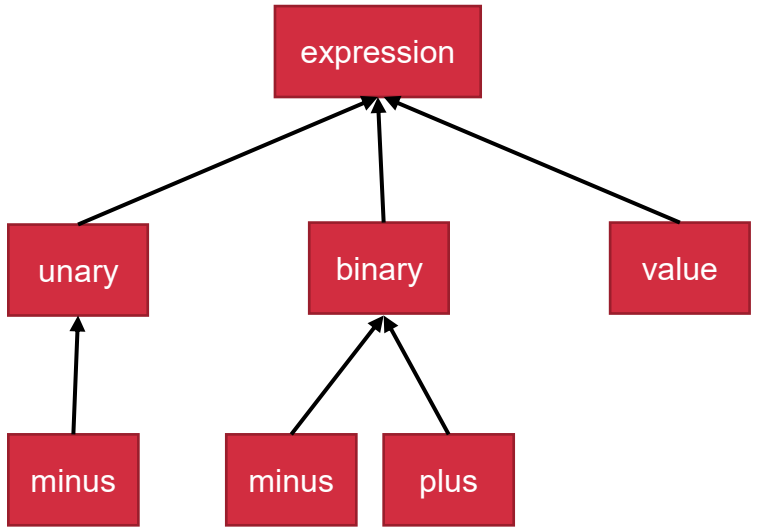
Inheritance: Rule-of-Thumbs

Think about the API

Other inheritance usage typically means bad design

`if-else` → bad OOP design






```
switch (type) {
  case ExprType::MINUS:
    // something
    break;
  case ExprType::PLUS:
    // something else
    break;
  ...
```

Don't forget to `break` in `switch`

What to do about it?

Polymorphism

Base class acts as the (internally stored) derived class
Function must be **virtual** and gets called via reference/pointer

```
int main() {
  derived d;
  d.print(); // derived: 0
  base &b = d;
  b.print(); // derived: 0
  base *b_ptr = &d;
  b_ptr->print(); // derived: 0
  base b2 = d;
  b2.print(); // base: 0
}
```

```
class base {
protected:
  int value = 0;
public:
  virtual ~base() = default;
  virtual void print() const {
    std::cout << "base: " << value;
  }
};

class derived : public base {
public:
  void print() const override {
    std::cout << "derived: " << value;
  }
};
```

Work 1: Polymorphic Vector

Create a vector which can store different types
(int, double, string)
Use inheritance (no union, variant, …)

Hints
Create a wrapper for inserted types
Use overloading

Submit the final version to Gitlab → 1 point

- **Create a *merge-request***
- **Deadline: Thursday November 27th 5:00**

```
// API
// insert a value
push_back(value);
// print the value of i-th element
print(size_type i);
// print all values inside the array
print_all();
```

```
int main() {
  my_vec v;
  v.push_back(1);
  v.push_back(2.3);
  v.push_back("four");
  v.print(0); // 1
  v.print(2); // "four"
  v.print_all(); // [1, 2.3, "four"]
}
```

Lab 7

14/11/2025

Feedback

https://forms.microsoft.com/e/bx4Tj8VW8J

Credit Project

Topic must be **approved** by Friday November 14th

- DM me Mattermost

Next

Approved Merge-request with detailed project description by **Friday December 12th**

Homework Discussion

```
vector(const vector &obj) :
    capacity(obj.capacity),
    size(obj.size)
{
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
}
```

```
vector & operator=(const vector &obj) {
    if (this == &obj) {
        return *this;
    }
    capacity = obj. capacity;
    size = obj. size;
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
    return *this;
}
vector & operator=(vector &&obj) {
    capacity = obj. capacity;
    size = obj.size;
    array_ptr = std::move(obj.array_ptr);
    obj.capacity = 0;
    obj.size = 0;
    return *this;
}
```

Homework Discussion

```
vector(const vector &obj) :
    capacity(obj.capacity),
    size(obj.size)
{
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
}
```

```
vector &operator=(const vector &obj) {
    if (this == &obj) {
        return *this;
    }
    capacity = obj. capacity;
    size = obj. size;
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
    return *this;
}
vector &operator=(vector &&obj) {
    capacity = obj. capacity;
    size = obj.size;
    array_ptr = std::move(obj.array_ptr);
    obj.capacity = 0;
    obj.size = 0;
    return *this;
}
```

DRY: Do not repeat yourself

Same code → function

Homework Discussion

```
vector(const vector &obj) :
    capacity(obj.capacity),
    size(obj.size)
{
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
}

friend void swap(vector &lv, vector &rh) noexcept {
  std::swap(lv.capacity, rh.capacity);
  std::swap(lv.size, rh.size);
  std::swap(lv.array_ptr, rh.array_ptr);
}
```

Copy-and-swap idiom

Strong exception guarantee

```
vector &operator=(const vector &obj) {
  // ???


}

vector(vector &&obj) : vector{} {
  // ???
}

vector &operator=(vector &&obj) {
   // ???


}
```

How to implement using swap?

Homework Discussion

```
vector(const vector &obj) :
    capacity(obj.capacity),
    size(obj.size)
{
    array_ptr = make_unique<int[]>(capacity);
    for (std::size_t i = 0; i < size; ++i) {
        array_ptr[i] = obj.array_ptr[i];
    }
}

friend void swap(vector &lv, vector &rh) noexcept {
  std::swap(lv.capacity, rh.capacity);
  std::swap(lv.size, rh.size);
  std::swap(lv.array_ptr, rh.array_ptr);
}
```

Copy-and-swap idiom

Useful in concurrent programming

```
vector &operator=(const vector &obj) {
  vector tmp(obj);
  swap(*this, tmp);
  return *this;
}

vector(vector &&obj) : vector{} {
  swap(*this, obj);
}

vector &operator=(vector &&obj) {
  vector tmp(std::move(obj));
  swap(*this, obj);
  return *this;
}
```

Homework Discussion

```
vector() : vector_capacity(8), vector_size(0) {
    array_ptr = std::make_unique<int[]>(vector_capacity);
}

vector(std::size_t size, int value) : vector_size(size) {
    std::size_t cap = 8;
    while (cap < size) {
        cap *= 2;
    }
    vector_capacity = cap;
    array_ptr = std::make_unique<int[]>(vector_capacity);
    for (std::size_t i = 0; i < vector_size; ++i) {
        array_ptr[i] = value;
    }
}
```

How about this?

INIT_CAPACITY

INIT_CAPACITY

CAPACITY_MULT

Function as Parameter

std::function
Functors
Lambdas
Templates
Function pointers/references (☹ C-style)

std::function<RET(PAR1, PAR2,…)>

```
void for_each(const std::vector<int> &vi,
    const std::function<void(int)> &fn) {
  for (int x : vi) {
    fn(x);
  }
}

void print(int x) { cout << x << endl; }
using myfn_t = std::function<void(int)>;

int main(int argc, char* argv[]) {
  vector<int> vi{1, 2, 3, 4, 5, 6};
  myfn_t f = print;
  ...
  for_each(f, print);
}
```

Function as Parameter

std::function

Functors

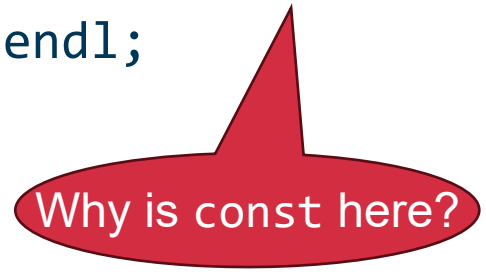
Lambdas

Templates

Function pointers/reference (☹ C-style)

A class with operator()


```
struct PrintFtor {
  void operator()(int x) const {
    cout << x << endl;
  }
};
```



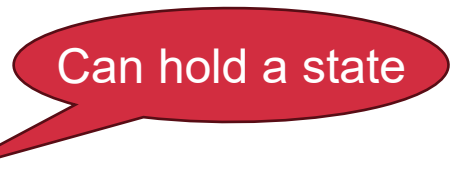

```
void for_each(const std::vector<int> &vi,
    const PrintFtor &fn) {
  for (int x : vi) {
    fn(x);
  }
}
```

```
int main(int argc, char* argv[]) {
  vector<int> vi{1, 2, 3, 4, 5, 6};
  PrintFtor print;
  ...
  for_each(f, print);
}
```

Function as Parameter

std::function
Functors
Lambdas
Templates
Function pointers/references (☹ C-style)

A class with operator()




```
struct NumOddFtor {
  size_t num = 0;
  void operator()(int x){
    num += x % 2;
  }
};
```

```
void for_each(const std::vector<int> &vi,
    NumOddFtor &fn) {
  for (int x : vi) {
    fn(x);
  }
}
```

```
int main(int argc, char* argv[]) {
  vector<int> vi{1, 2, 3, 4, 5, 6};
  NumOddFtor num_odd;
  ...
  for_each(f, num_odd);
  cout << num_odd.num << end;
}
```

Function as Parameter

std::function
Functors
Lambdas
Templates
Function pointers/references (☹ C-style)

```
void for_each(const std::vector<int> &vi,
    auto fn) {
  for (int x : vi) {
    fn(x);
  }
}
```

Needs auto

An anonymous in-place functor

```
[capture](par1,...){ /* body */ };
```

Passes things from the outside

```
auto l = [&num_odd,TWO](int x){
  num_odd += x % TWO;
};
```

By reference

By copy

```
int main(int argc, char* argv[]) {
  vector<int> vi{1, 2, 3, 4, 5, 6};

  ...
  for_each(f, [](int x){
    cout << x << endl;
  });
}
```

Algorithms

`#include <algorithm>` (+ namespace `std::ranges`)

A library defining functions for variaty of purposes

- **Batch:** for_each, …
- **Search:** find, find_first, all, any, …
- **Fold:** fold_left, fold_right, …
- **Modifying sequence:** copy, copy_if, …
- **swap, minimum/maximum:** swap, max, min, minmax, …
- **Transformation and Generation:** transform, replace, fill, generate, …
- **Removing and Order-changing:** remove, reverse, rotate
- **Partitioning and Sorting:** partition, sort, is_sorted
- **Binary search, Set, Heap:** lower_bound, binary_search, set_union, includes, make_heap, push_heap, …
- **Numeric:** iota, accumulate, reduce, …
- **Lexicographical comparison, Permutation, on Unitialized memory**

Define range through iterators
Anything iterator-like (iterators, pointers, …)

Define action through function/functor/lambda
Anything function-like (functions, functors, …)

Work 1: People big data database

Store all information found on your **ID** (name, age, address, …)
Efficient **search** based on individual fields
Compute **statistics** for individual fields (min, max, avg, histogram)
Optimize for big data but fit into your main memory

Hints
Think about efficient data structures
Careful with the copies (owners vs. observers)
Do you need dynamic allocation?
Use algorithms

Submit the final version to Gitlab → 2 point

- **Create a *merge-request***
- **Deadline: Thursday November 20th 5:00**

https://forms.microsoft.com/e/bx4Tj8VW8J

Lab 6

07/11/2025

Credit Project

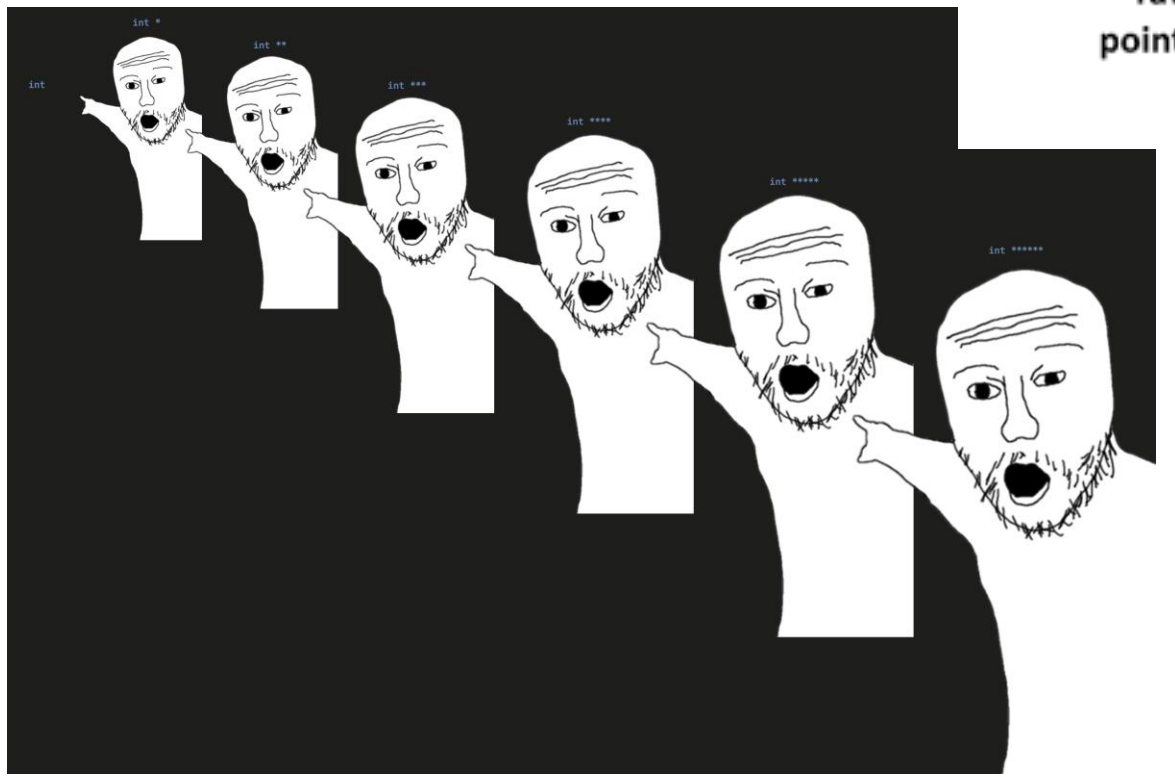
Topic must be **approved** by Friday November 14th

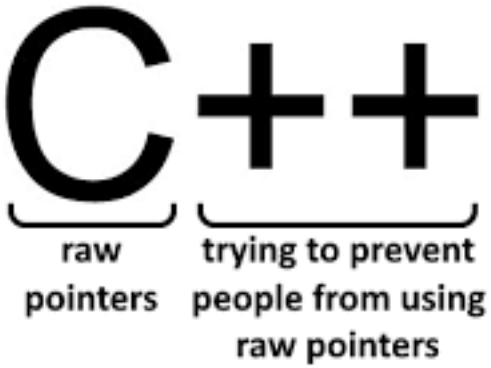
- DM me Mattermost

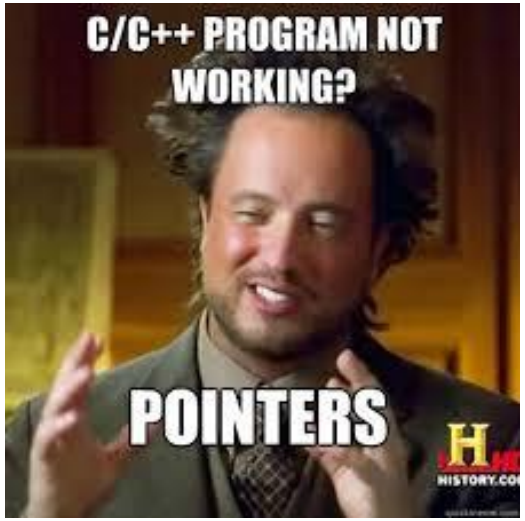
Homework Discussion

Pointers

random dev: I got stuck with this problem. Can anyone give me some pointers?

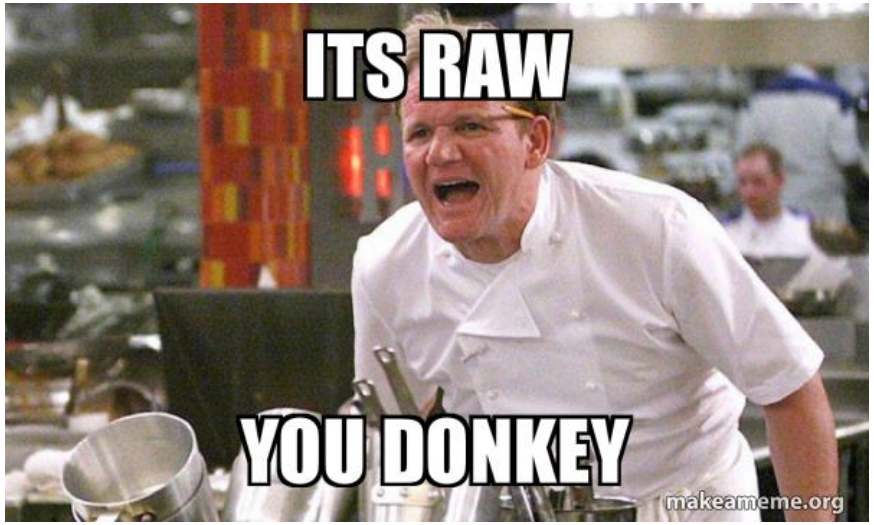






Dynamic Allocation

Use smart pointers (no raw pointers, i.e., **no new/new[]**)



Single owner

Most common case

Passing the ownership - move only, no copy

`unique_ptr<T>`

Shared ownership (multiple owners)

`shared_ptr<T>`

`weak_ptr<T>` // to break the cycle

Do you **really** need to own or just observe/work with the object?

Creation

`make_unique<T>, make_shared<T>`

```
make_unique<int[10]>() // Allocation of consecutive memory (~array)
```

Observers

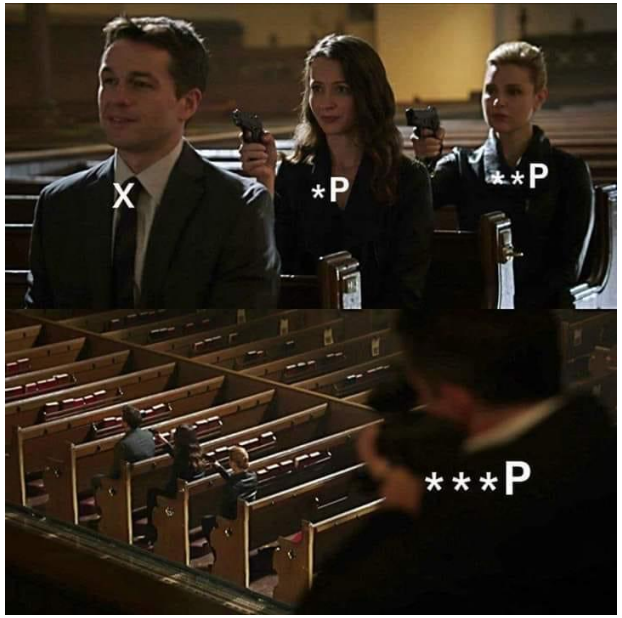
Working with the pointer with **no changes to ownership**

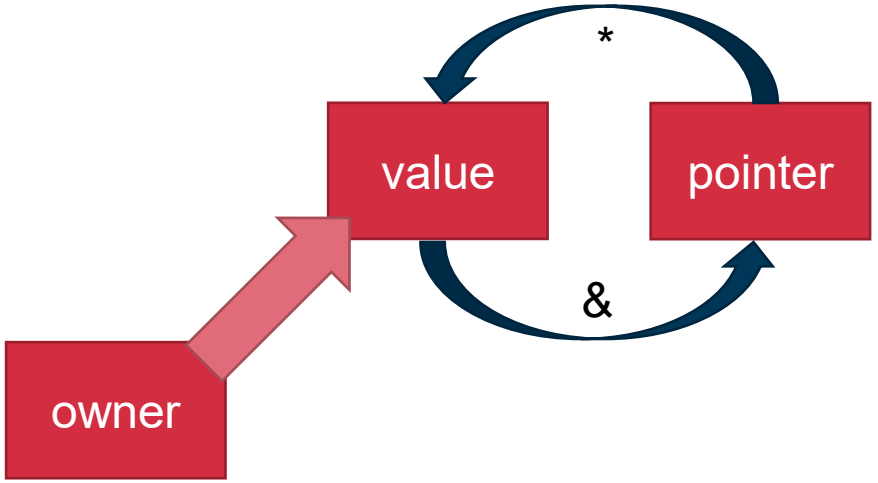
Return type (pointer)
Modifying observer → `T *` (pointer)
Read-only observer → `const T *`

Getting an observer
smart pointers → `get()`
otherwise → `&` (observer ~ address)

Accessing the value
access the value directly → `*`
access member attribute → `->` (same as doing (*var).

Pointing to nothing → `nullptr`





Dynamic Allocation: Final Notes

Prefer static/automatic storage

☺ complex c;

☹ auto c = make_unique<complex>();

Dynamic allocation **is slow!**

Prefer containers

☺ vector<int> vi(10);

☹ auto ia = make_unique<int[]>(10);

Prefer `unique_ptr`

use shared_ptr only for complex ownership, e.g., unknown ownership protocol (multithreading)

Use only when necessary

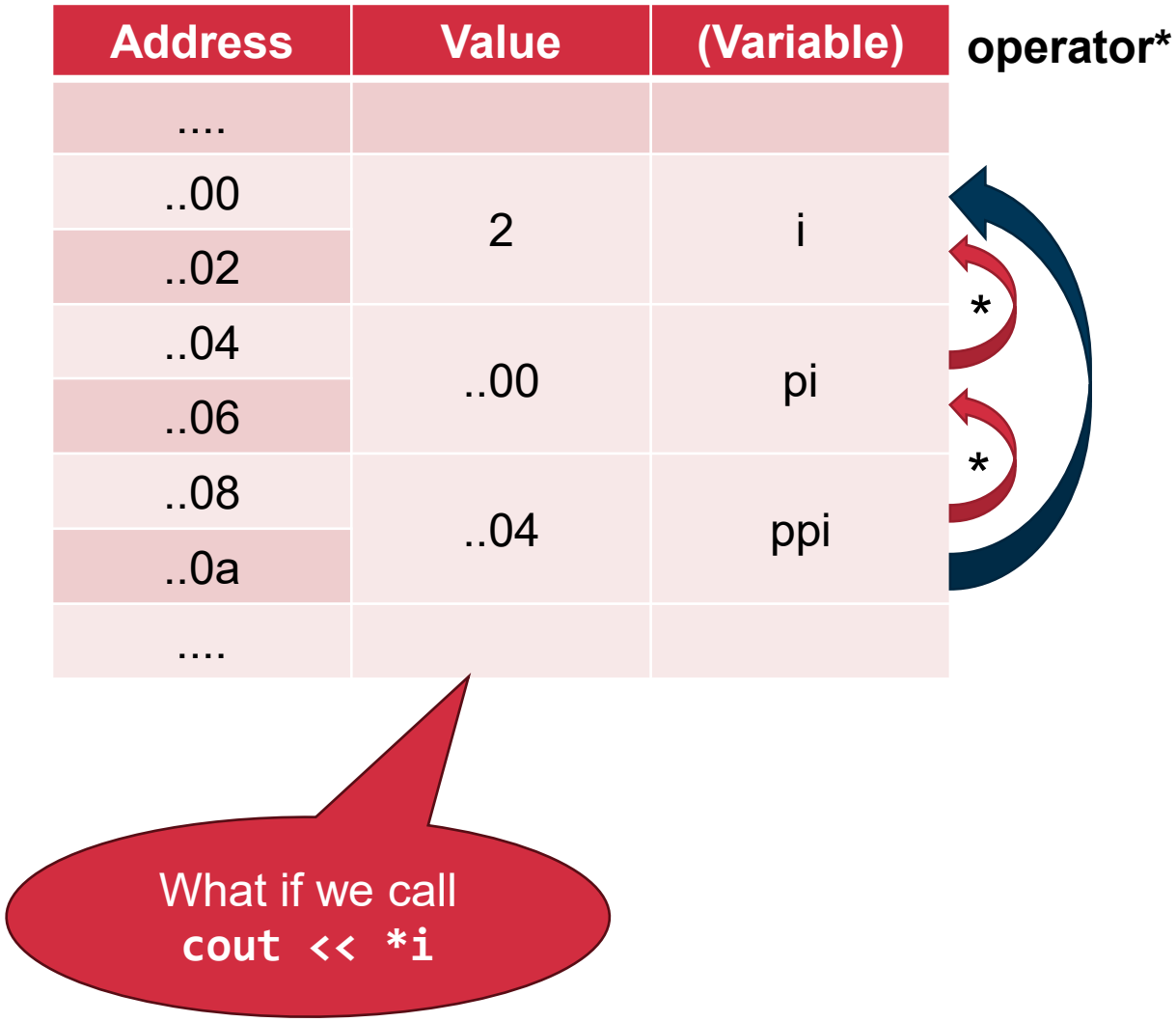
Object lifetime doesn't correspond to function invocations

Polymorphism

Is it really that **complex** or just **messy code?**

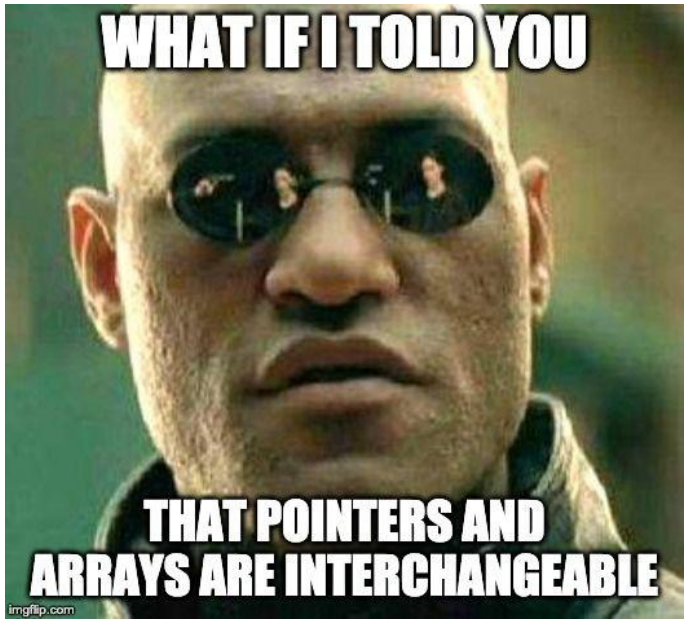
Pointers in Memory

```
int main() {
  int i = 2;
  int *pi = &i;
  int **ppi = &pi;
  cout << i;      // 2
  cout << pi;     // ..00
  cout << *pi;    // 2
  cout << ppi;    // ..04
  cout << *ppi;   // ..00
  cout << **ppi; // 2
}
```



Pointer Arithmetic

```
unique_ptr<int[]> int_arr = make_unique<int[]>(10);
for (size_t i = 0; i < 10; ++i) { int_arr[i] = 0; }
int *ptr = int_arr.get();
(*ptr) = 1;
*(ptr + 1) = 2;
cout << *ptr << ' ' << *(ptr + 1) << endl; // 1 2
ptr += 1;
cout << *ptr++ << endl; // 2
cout << *ptr << ' ' << *(ptr - 1) << endl; // 0 2
*ptr-- = 3;
cout << ptr[-1] << ' ' << ptr[0] << ' ' << ptr[1] << endl; // 1 2 3
```



Takes type into account

Linked List: Example

```
struct node {
  unique_ptr<node> next;
  int value;

  node(int value, unique_ptr<node> &&next) :
    value(value), next(std::move(next)) {}
};

class linked_list {
  unique_ptr<node> first_node;
public:
  node *front() {
    return first_node.get(); // Observer
  }

  const node *back() const {
    node *ptr = first_node.get(); // Observer
    if (ptr != nullptr) {
      while (ptr->next != nullptr) {
        ptr = (*ptr).next.get(); // Equivalent to ->
      }
    }
    return ptr;
  }
```

```
  void push_front(int value) {
    auto new_node = std::make_unique<node>(
        value,
        std::move(first_node));
    first_node = std::move(new_node);
  }

  void pop_front() {
    auto first = std::move(first_node);
    first_node = std::move(first->next);
  } // automatic deallocation of first
```

Work 1: Finish the LL

```
size(), print(), push_back(), pop_back()
ctor(), ctor(init_size, default_value), dtor
operator[]
```

Submit the final version to Gitlab → 1 point

- **Create a *merge-request***
- **Deadline: Thursday November 13th 5:00**

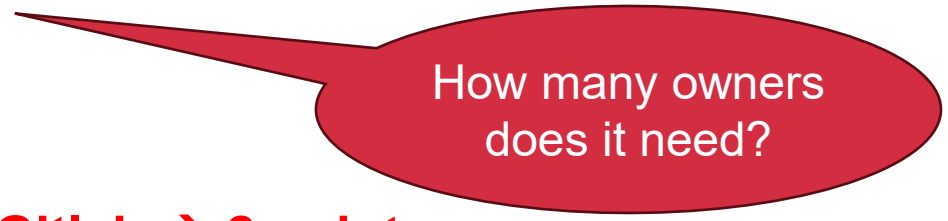
Work 2: `vector<int>`

Implement your own integer vector

Mandatory operations

default ctor, ctor(size_t, value_type), copy/move ctor/assignment
size(), capacity(), reserve(), push_back(), operator[]()

Use array allocation, no LL



Submit the final version to Gitlab → 2 point

- **Create a *merge-request***
- **Deadline: Thursday November 13th 5:00**

Lab 5

31/10/2025

Credit Project

Topic must be **approved** by Friday November 14th

- DM me Mattermost

Declaration/Definition

```
// file: my_class.hpp

#ifndef MY_CLASS_HPP
#define MY_CLASS_HPP

void fn(int x);

class my_class {
public:
  my_class();
  int exec(int x);

private:
  double d;
  static size_t i;
};

#endif // MY_CLASS_HPP
```

```
// file: my_class.cpp

#include “my_class.hpp”
#include <iostream>

void fn(int x) {
  cout << “fn()”;
}

my_class::my_class() : d(1.0) {
  cout << “ctor”;
}

int my_class::exec(int x) {
  for(int i=0; i < x; ++i) { … }
}

size_t my_class::i = 0;
```

Declaration/Definition

```
// file: my_class.hpp

#ifndef MY_CLASS_HPP
#define MY_CLASS_HPP

void fn(int x);

class my_class {
public:
  my_class();
  int exec(int x);

private:
  double d;
  static size_t i;
};

#endif // MY_CLASS_HPP
```

Declaration

```
// file: my_class.cpp

#include “my_class.hpp”
#include <iostream>

void fn(int x) {
  cout << “fn()”;
}

my_class::my_class() : d(1.0) {
  cout << “ctor”;
}

int my_class::exec(int x) {
  for(int i=0; i < x; ++i) { … }
}

size_t my_class::i = 0;
```

Include header

Definition

Declaration/Definition

```
// file: my_class.hpp

#ifndef MY_CLASS_HPP
#define MY_CLASS_HPP

void fn(int x);

class my_class {
public:
  my_class();
  int exec(int x);

private:
  double d;
  static size_t i;
};

#endif // MY_CLASS_HPP
```

Include guards

```
// file: my_class.cpp

#include “my_class.hpp”
#include <iostream>

void fn(int x) {
  cout << “fn()”;
}

my_class::my_class() : d(1.0) {
  cout << “ctor”;
}

int my_class::exec(int x) {
  for(int i=0; i < x; ++i) { … }
}

size_t my_class::i = 0;
```

Declaration/Definition

```
// file: my_class.hpp

#ifndef MY_CLASS_HPP
#define MY_CLASS_HPP

void fn(int x);

class my_class {
public:
  my_class();
  int exec(int x);

private:
  double d;
  static size_t i;
};

#endif // MY_CLASS_HPP
```

#pragma once

Include guards

```
// file: my_class.cpp

#include “my_class.hpp”
#include <iostream>

void fn(int x) {
  cout << “fn()”;
}

my_class::my_class() : d(1.0) {
  cout << “ctor”;
}

int my_class::exec(int x) {
  for(int i=0; i < x; ++i) { … }
}

size_t my_class::i = 0;
```

Sequence Containers

std::vector<Type> - dynamic array

- my_vec[idx] = value, push_back(), …

std::array<Type, Size> - fixed size array

- my_array[idx] = value, …

std::deque<Type> - double ended dynamic array

- push_back(), push_front(), back(), front(), …

std::list<Type> - linked list

Adaptors
Stack, queue, priority_queue
flat_set, flat_map, flat_multiset, flat_multimap

Iterators

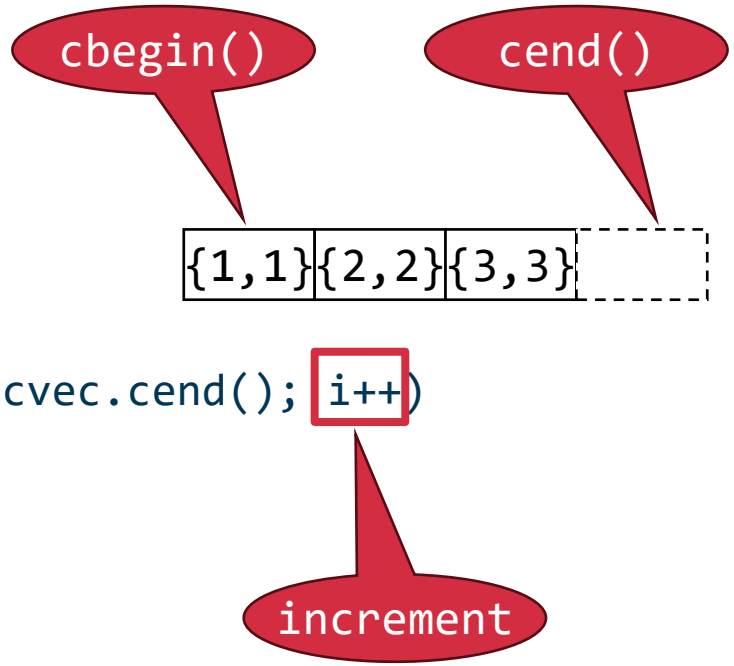
*An object representing **references to elements** inside a container*

```
struct Complex { double re; double im; };
using ComplexVec = std::vector<ComplexVec>;
ComplexVec cvec{{1., 1.}, {2., 2.}, {3., 3.}};

for (ComplexVec::const_iterator i = cvec.cbegin(); i != cvec.cend(); i++)
  cout << “[“ << i->re << “,” << i->im << “]” << endl;

std::vector<int> vi{1, 2, 3};
cout << “begin: “ << *vi.begin() << “, end: “ << *vi.end(); << endl;
```

Iterators

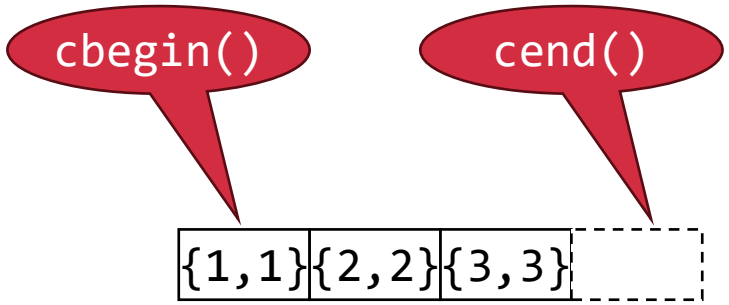
*An object representing **references to elements** inside a container*

```
struct Complex { double re; double im; };
using ComplexVec = std::vector<ComplexVec>;
ComplexVec cvec{{1., 1.}, {2., 2.}, {3., 3.}};

for (ComplexVec::const_iterator i = cvec.cbegin(); i != cvec.cend(); i++)
  cout << "[" << i->re << "," << i->im << "]" << endl;

std::vector<int> vi{1, 2, 3};
cout << "begin: " << *vi.begin() << endl;
```



Iterators

*An object representing **references to elements** inside a container*



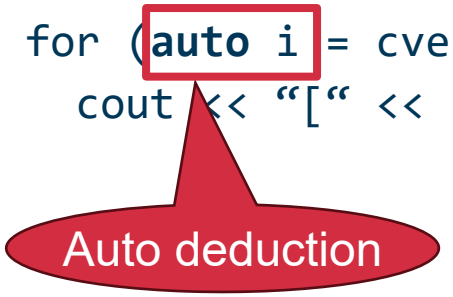

```
struct Complex { double re; double im; };
using ComplexVec = std::vector<ComplexVec>;
ComplexVec cvec{{1., 1.}, {2., 2.}, {3., 3.}};

for (ComplexVec::const_iterator i = cvec.cbegin(); i != cvec.cend(); i++)
  cout << "[" << i->re << "," << i->im << "]" << endl;

std::vector<int> vi{1, 2, 3};
cout << "begin: " << *vi.begin() << ", end: " << *vi.end(); << endl;
```

What happens?

Iterators

*An object representing **references to elements** inside a container*

```
struct Complex { double re; double im; };
using ComplexVec = std::vector<ComplexVec>;
ComplexVec cvec{{1., 1.}, {2., 2.}, {3., 3.}};

for (auto i = cvec.cbegin(); i != cvec.cend(); i++)
  cout << “[“ << i->re << “,” << i->im << “]” << endl;
```



Iterators

*An object representing **references to elements** inside a container*

```
struct Complex { double re; double im; };
using ComplexVec = std::vector<ComplexVec>;
auto cvec{{1., 1.}, {2., 2.}, {3., 3.}};

for (aut i = cvec.cbegin(); i != cvec.cend(); i++)
  cout << "[" << i->re << "," << i->im << "]" << endl;
```

What is the type here?

Work 1: Summing Program

Program that prints a sum of all numbers

Implement **special methods only in C/D**

- Ctors, dtors, operators, …

Can add O(1) **attributes into C** (e.g., no `vector`)

`C::print()` **only** used for printing

Goal: Understand **when** special methods are called

Submit the final version to Gitlab → 1 point

- **Deadline: Thursday November 6th 5:00**

```
class C {
  /* CAN ADD MORE ATTRIBUTES */
  const int value;
  /* USE THIS FUNCTION FOR PRINTING */
  void print() const {
    cout << value << "\n";
  }

public: /* IMPLEMENT SPECIAL METHODS ONLY */
};

class D {
  std::vector<C> cs; /* NO MORE ATTRIBUTES */
public: /* IMPLEMENT SPECIAL METHODS ONLY */
};

int main(int argc, char *argv[]) {
  int first, last;
  cin >> first >> last;
  cout << ”Summing numbers:\n";
  D d(first, last); // prints: first, ..., last
  cout << "Preparing...\n";
  D d2 = d;
  cout << "Sum of the numbers:\n";
  d2 = d; // prints sum of numbers first...last
}
```

Work 2: N-in-row for P players

A generic N-in-row (“Piškvorky”) for P players

Set the **number of players** and **how many symbols in row** is needed
Set the **names** and **symbols** for individual players to **visualize** the game
Players take **turns**
Game ends when the **first** player has **N-in-row**
Validate player **inputs**

Submit the final version to Gitlab → 3 point

- **Deadline: Thursday November 6th 5:00**

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);
```

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

int Verbose::cnt = 0;

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

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
}
```

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

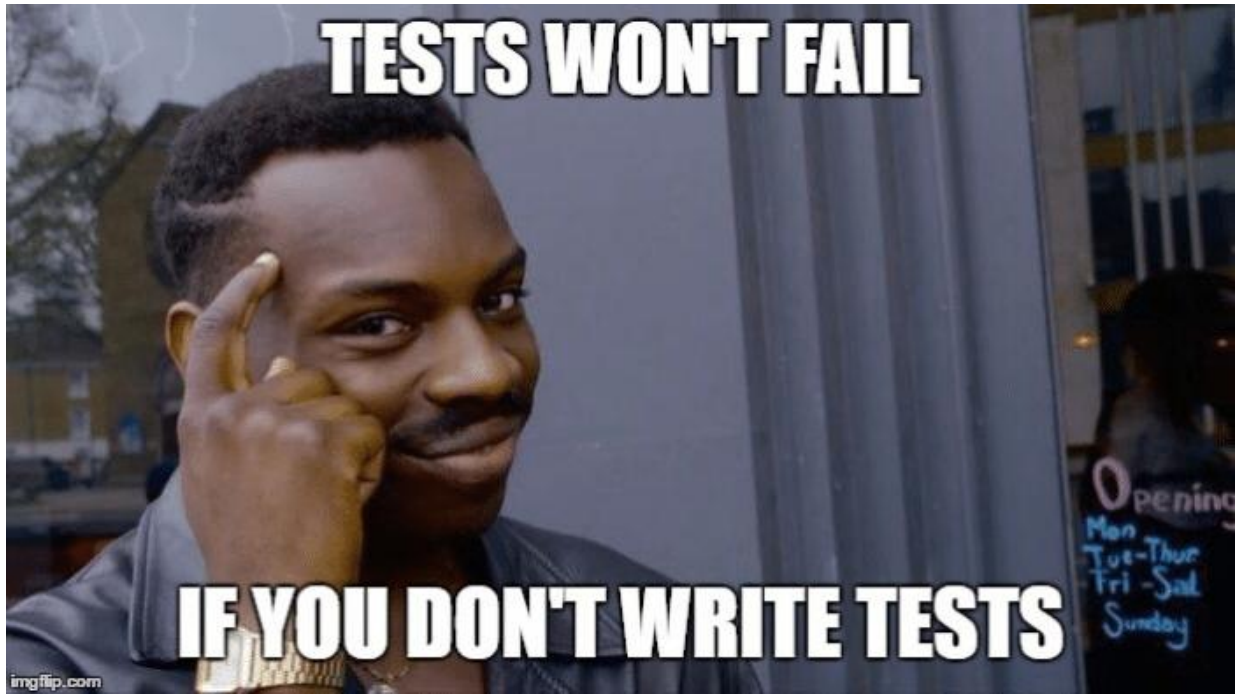
Points can be found in “Studijní mezivýsledky”

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:

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 &`**

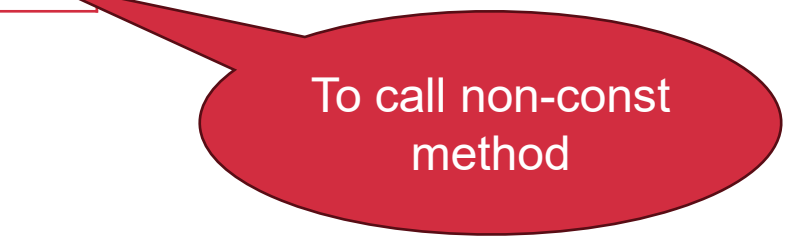
```
template<typename _Ch_type, class _Alloc, class _Rx_traits>
    inline bool
    regex_search(const _Ch_type* __s,
                 match_results<const _Ch_type*, _Alloc>& __m,
                 const basic_regex<_Ch_type, _Rx_traits>& __e
                 regex_constants::match_flag_type __f,
   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

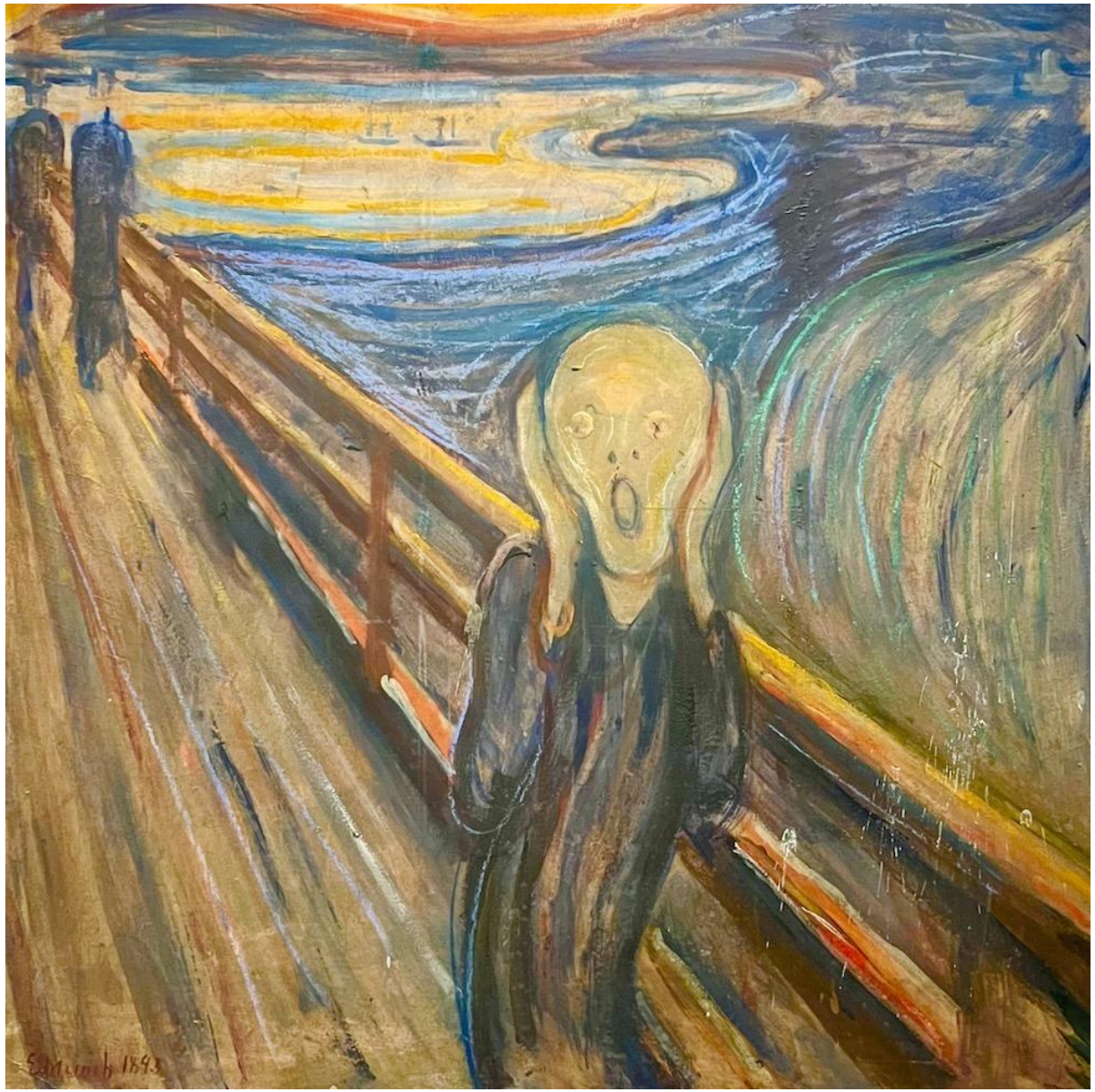


Parameter Passing

Rule of thumb

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

Why is there no Out?

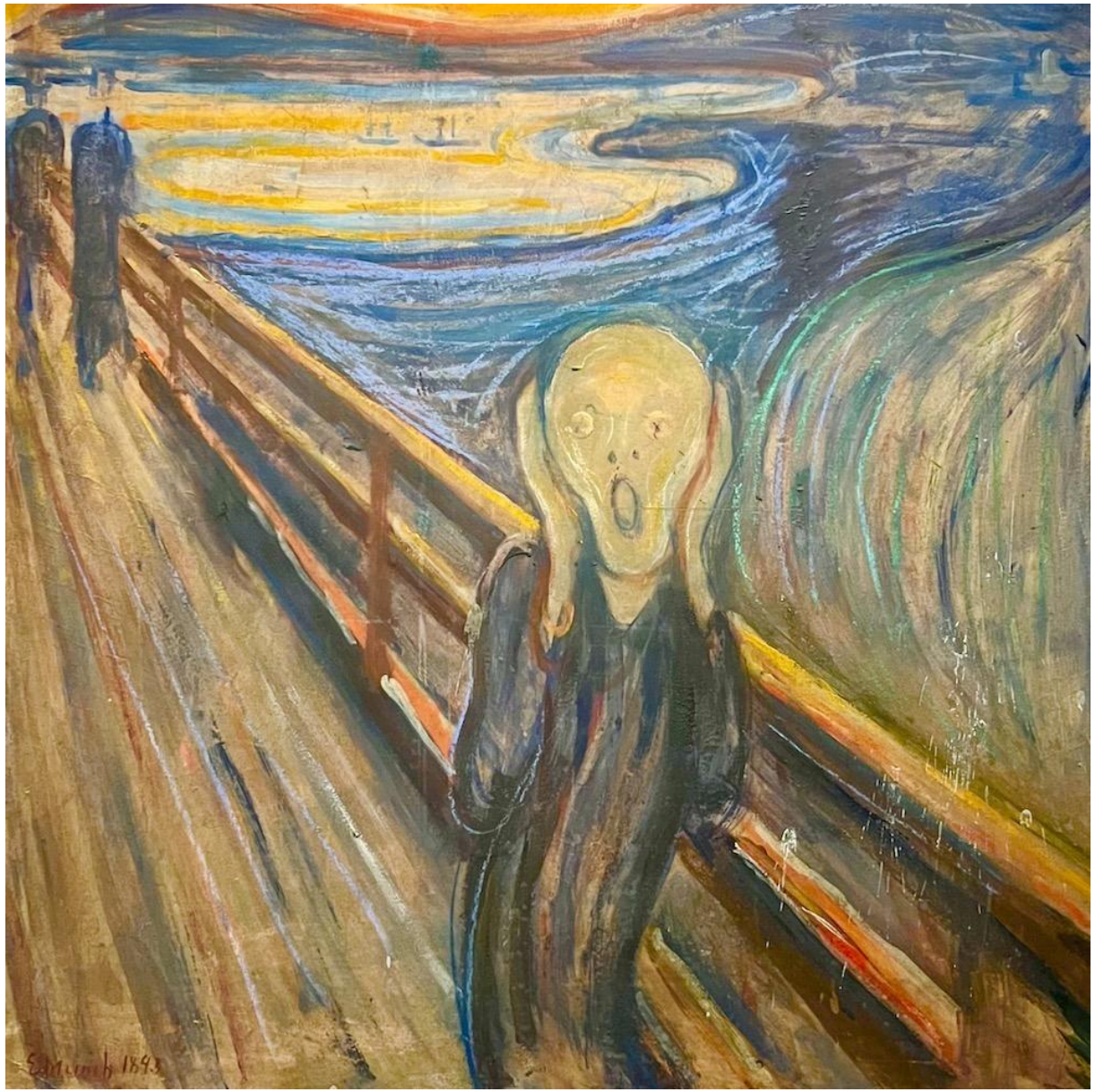


Parameter Passing

Rule of thumb

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

Use return

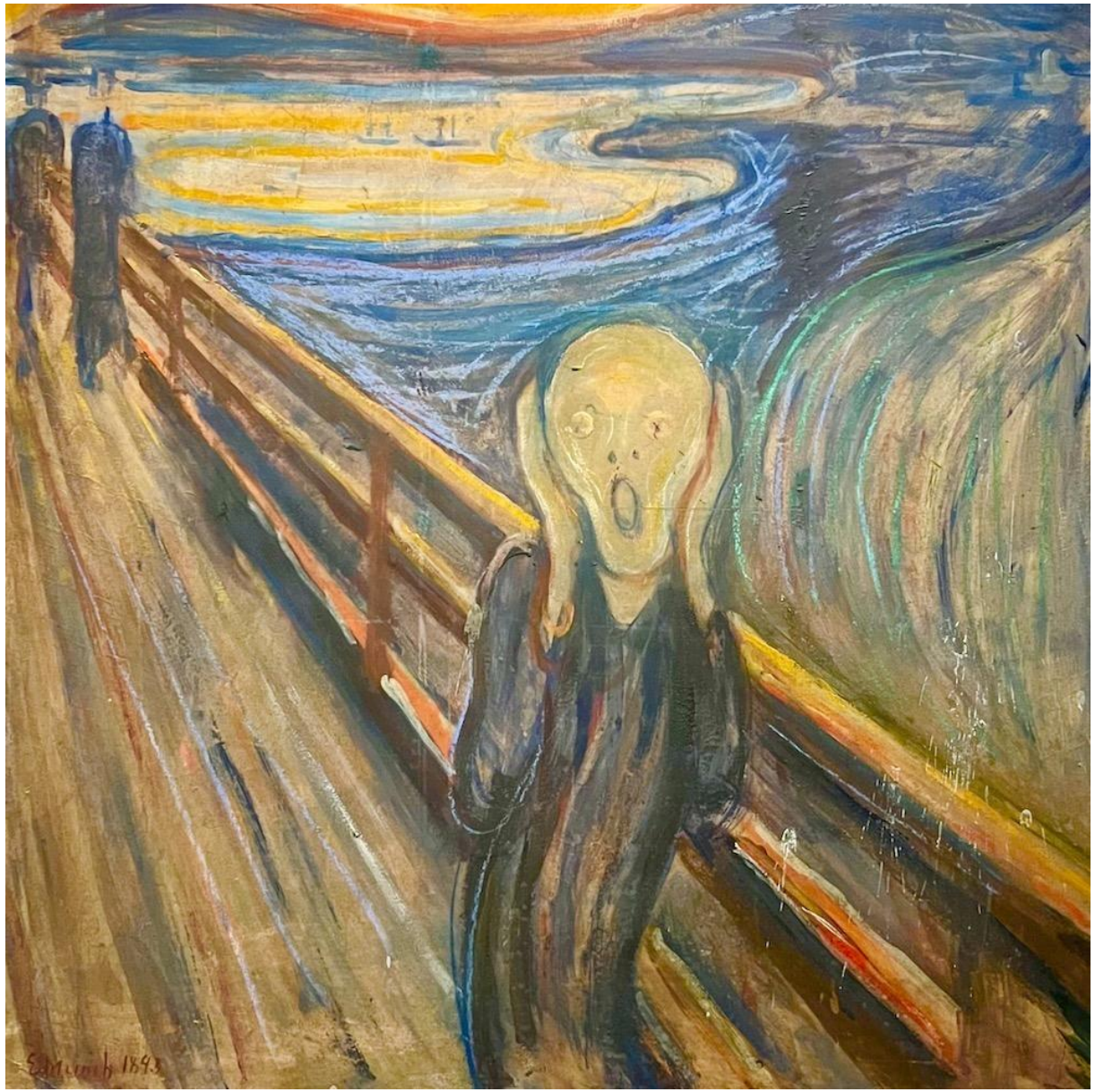


Parameter Passing

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

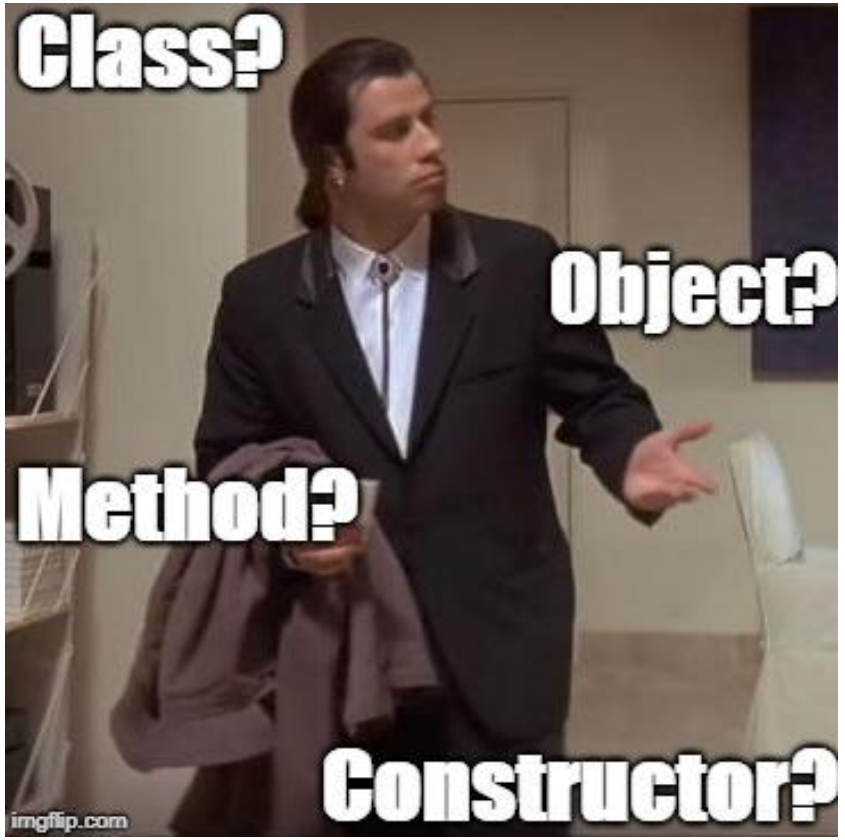
Internal things → private

- 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…

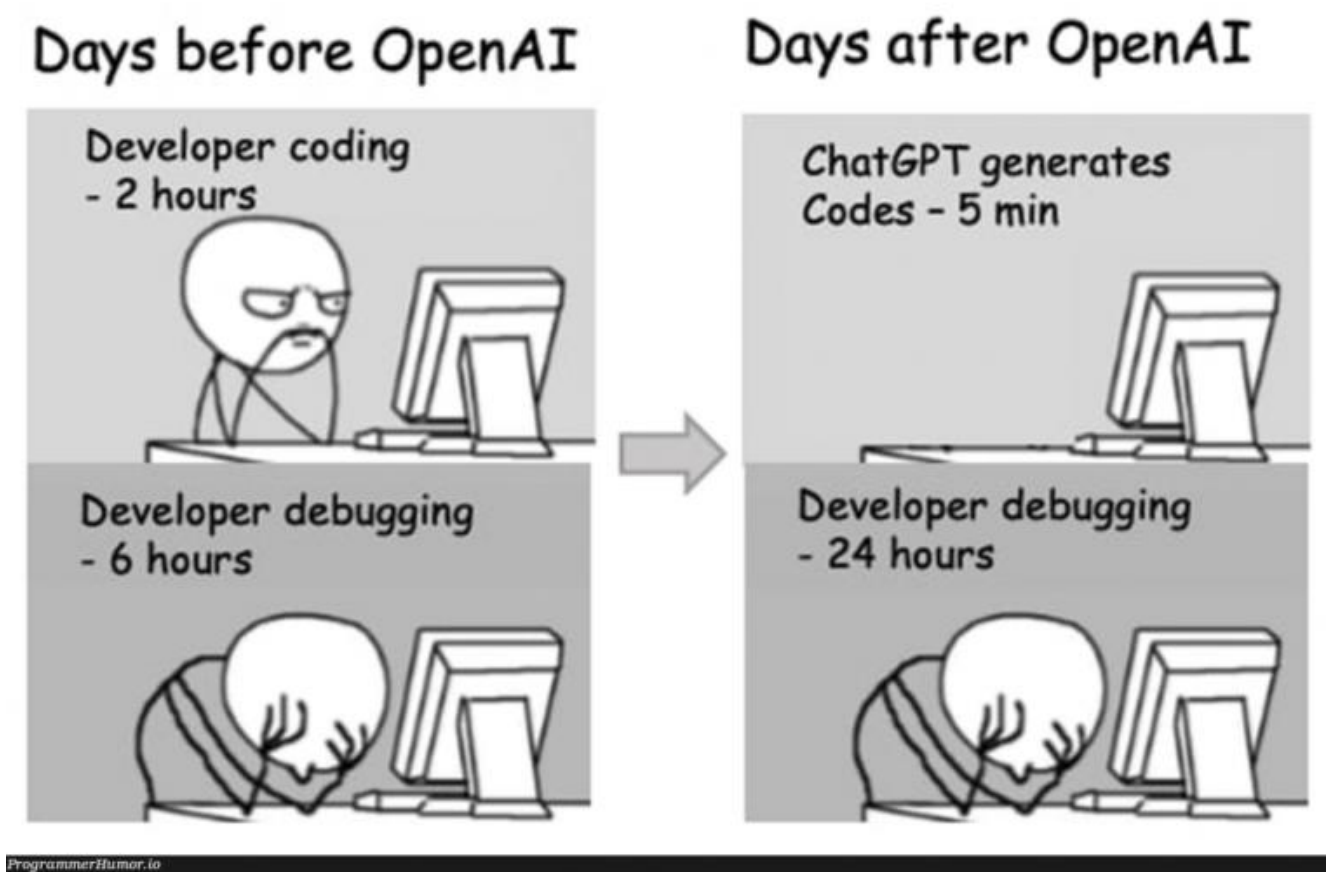
"There's a tiger in you"

Tiger in me :

Charles University

Let’s get it started

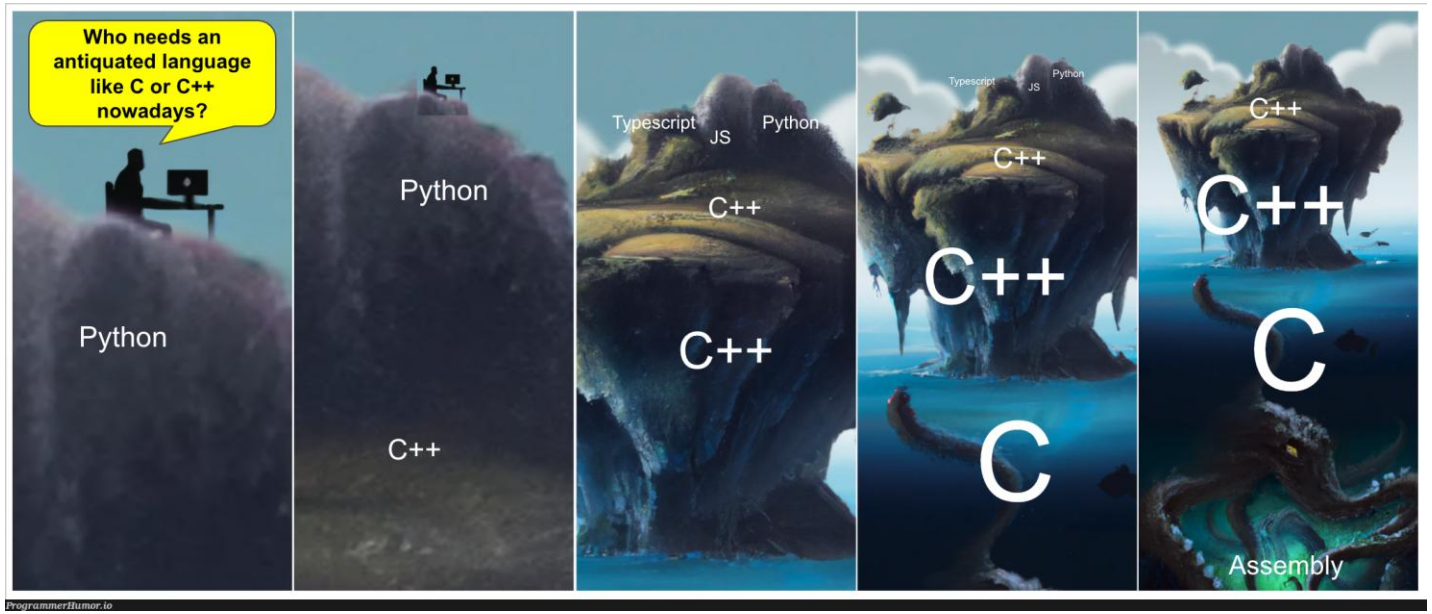
Motivation


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

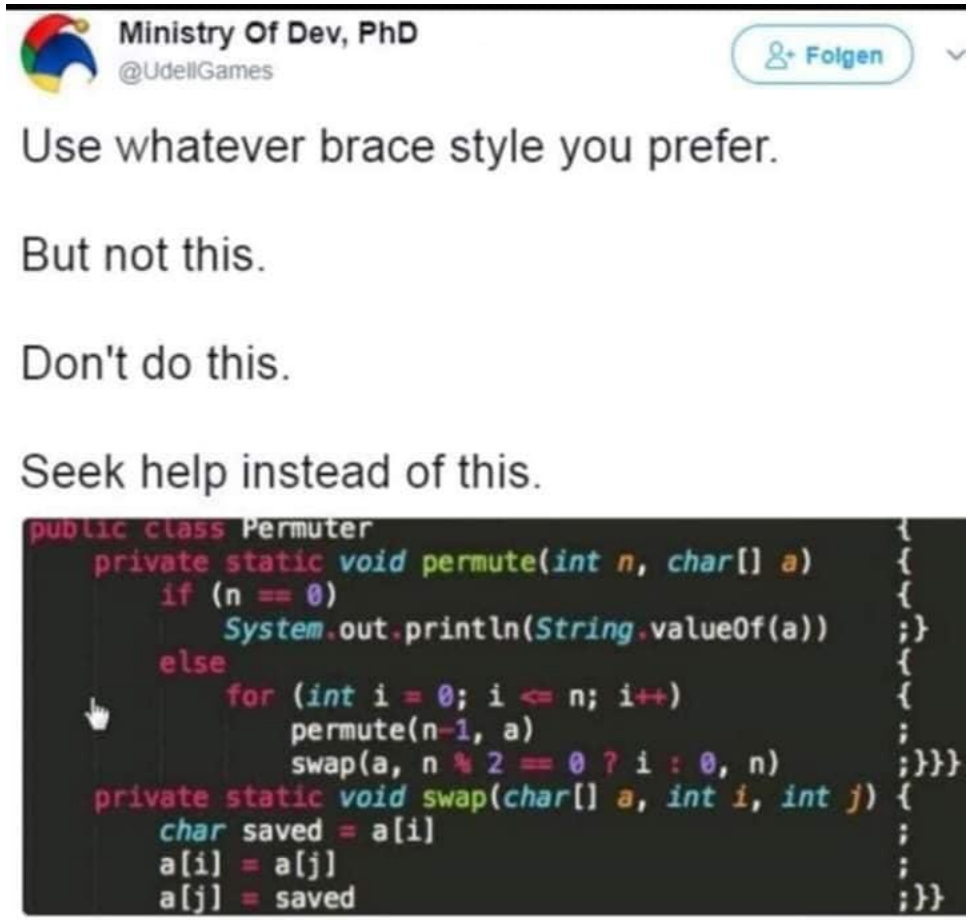
- https://teaching.ms.mff.cuni.cz/nswi170-web/pages/labs/coding/

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

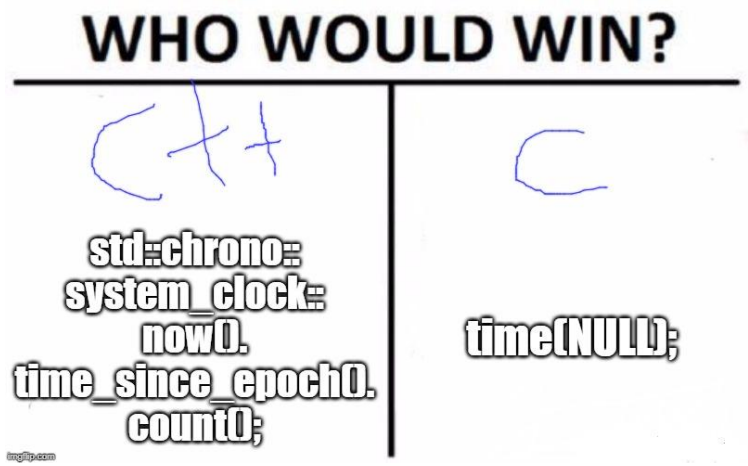


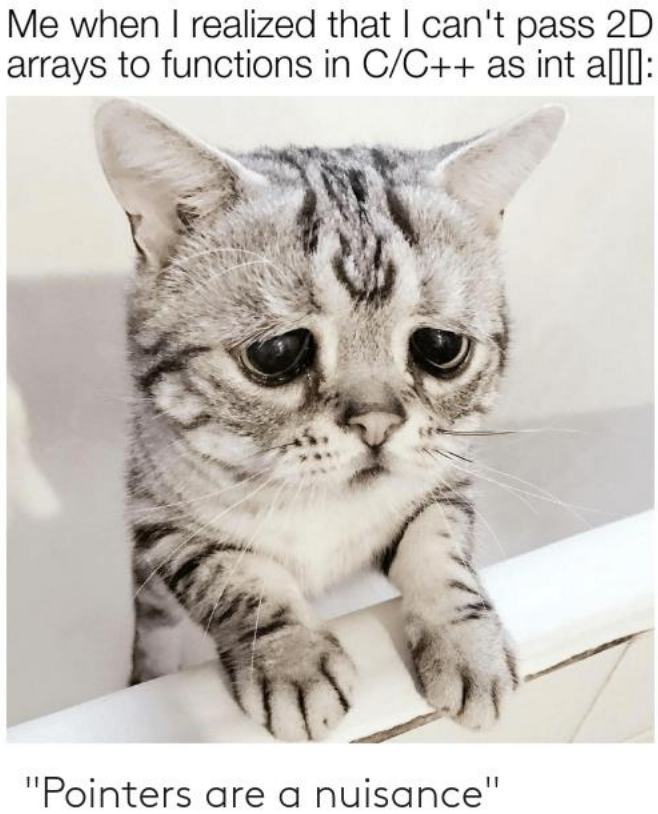
Code Requirements – Readability

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[]`

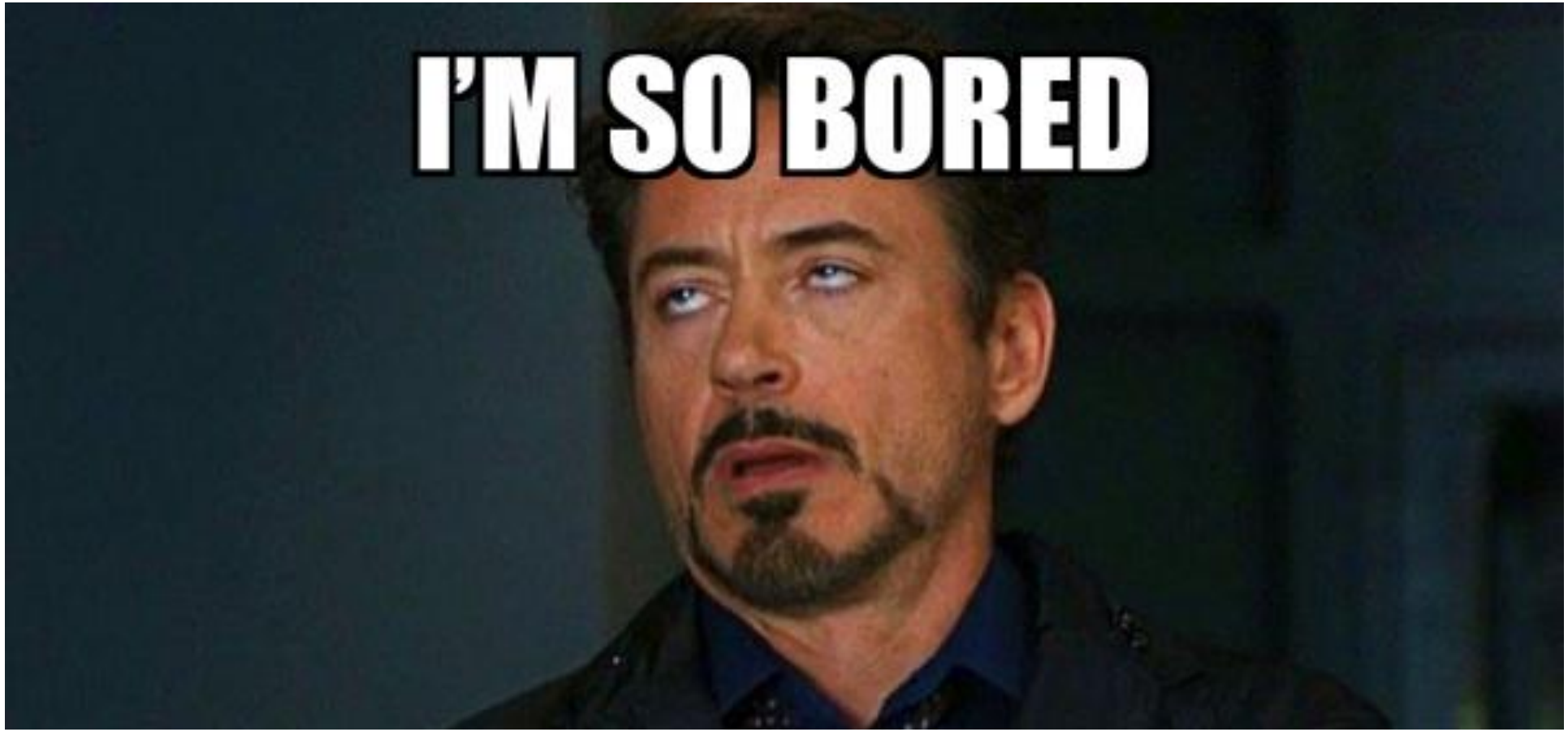




Code Requirements – Working

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**