Programming in C++

https://fan1x.github.io/cpp21.html
tomas.faltin@matfyz.cuni.cz

Programming in C++ - lab 4

https://fan1x.github.io/cpp21.html
tomas.faltin@matfyz.cuni.cz

Homework Feedback

- Use const functions for read-only functions
  - `print() const, get_matrix() const, get_vector() const`
- Use `class` or `using` to create new types
  - Decomposition!
  - `using` can be anywhere (inside the class as well)

Argument Passing - Recap

- By copy/value
  - `int max(int x, int y);`
- By const-reference:
  - `Matrix sum(const Matrix &m1, const Matrix &m2);`
- By reference
  - `void find_zero_matrix(const vector<Matrix> &ms,`
    `Matrix &zero_matrix);`

Argument Passing – By R-value Reference (&&)

- To transfer an ownership
- Moves the object into a function
  - the object no longer lives outside the function
- Typical usage
  - a single owner (`std::unique_ptr`)
  - moving large objects
- Use `std::move()` on the caller side

```
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));
```

Static With Classes

- Attribute/method belongs to a class (not an object/instance)
- Need to share attribute/method among the objects/instances
- Most things belong to an object

```
class Verbose {}; // class

int main()
{
  Verbose v1; // object/instance
  Verbose v2(2); // object/instance
}
```

Static With Class

```
class CountingClass {
  static size_t num_instances;

  static void inc_num_instances() {
    ++num_instances;
  }

  static void dec_num_instances() {
    --num_instances;
  }

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

  static size_t get_num_instances() {
    return num_instances;
  }

  CountingClass() { inc_num_instances(); }

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

  ~CountingClass() { dec_num_instances(); }
};
```

```
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
}
```

Special Methods In Classes

```
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
```

Homework1: Implement class C

- Finish program so it writes: `1,2,3,…,16`
- Touch **only** class C, nothing else
  - Nothing can be into `main()` or `fn_XXX()`
- Don't use `exit()`, `break`, `goto`, …
- Hint: which methods are called?

```
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";
}
```

Homework2: Finish Matrix for Integers

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

Programming in C++ - lab 3

https://fan1x.github.io/cpp21.html
tomas.faltin@matfyz.cuni.cz

Down to operator

```
void op_downto(int x) {
  while (x --> 0) {
    cout << x;
  }
}

op_downto(10); // prints 9,8,7,…,1,0
```

Homework Feedback

- Use `const &` for large objects
- Only source codes and project/config files to GIT
  - No binaries (they can be compiled from the source codes)
- Use STL functions
  - `isdigit(), stoi(), …`
- Prefer C++ strings to C-style strings
  - `std::string, std::string_view`

`Class/Struct - Recap`

- Put all related things (data, functions) together
- 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, …)

Defining your own types - using

- Use `using` (or `typedef` in old C/C++)
- Can be used together with templates (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 – constexpr/const

- Read only value that cannot be changed
- Naming values in code
  - ~ Every number in the code should be a named constant
- `constexpr` – constant value (potentially) evaluated in the compile time
  - Can be used as arguments to templates
- `const` – constant value
- Both can be used together with `static` (later)

```
constexpr double PI = 3.14;
constexpr size_t MAX_SIZE = 16 * 1024 * 1024;
```

Coding: 3D Matrix for Integers - API

- ctor(), ctor(width, length, heigth)
- 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();

Coding: 3D Matrix for Integers - Hints

- Think about the desing
  - array → matrix → 3D matrix → 4D matrix → … → XD matrix
  - Design simple first, then continue to the next level
- No need to focus too much on performance yet
- Focus:
  - Passing arguments: const-references, references, …
  - const functions
  - class design
    - Decomposition into functions
    - Function reusing
    - private/public

Coding: 3D Matrix - Improvements

- `print()`
- `sort_vector(x, y)`
  - Use `std::sort()`
- change underlying matrix container - `std::deque, std::list`
  - the change to different container must be only few lines of change
    - Hint: use `using`
- change underlying matrix container - `std::array`
  - Use large enough array
    - ! Use constants
  - Report error in case of overflow

Programming in C++ - lab 2

https://fan1x.github.io/cpp21.html
tomas.faltin@matfyz.cuni.cz

Recap

Homework Example

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();

private:
};
```

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
- 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();

3D Matrix for Integers - Hints

- Think about the desing
  - array → matrix → 3D matrix → 4D matrix → … → XD matrix
  - Design simple first, then continue to the next level
- No need to focus too much on performance yet
- Focus:
  - Passing arguments: const-references, references, …
  - const functions
  - class design
    - Decomposition into functions
    - Function reusing
    - private/public

Programming in C++ - lab 1

https://fan1x.github.io/cpp21.html
tomas.faltin@matfyz.cuni.cz

Basic information

- Email: tomas.faltin@matfyz.cuni.cz
- Lab's web: https://fan1x.github.io/cpp21.html
- ZOOM for distance learning
  - https://cuni-cz.zoom.us/j/94350923737
  - Credentials in SIS/mail
- Mattermost
  - Invite link: https://ulita.ms.mff.cuni.cz/mattermost/signup_user_complete/?id=z1knw5ag6p8nipop1i7icig a6a
    - Use ASAP, might expire eventually
  - Channel: `nprg041-cpp-english`
- Gitlab
  - https://gitlab.mff.cuni.cz/
  - https://gitlab.mff.cuni.cz/teaching/nprg041/2021-22/eng

Communication is the key

- Don’t be afraid to ask
  - 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
- Be proactive

Labs credit

- Submitted homeworks before Monday midnight (to Gitlab)
  - Even if not attending!
  - Won't be graded, for a feedback
- Two large homeworks in ReCodex (40 points)
  - Points are included in the final score from the course
  - Smaller HW – 15 points, ~November
  - Larger HW – 25 points, ~December
- Software project
  - Topic must be approved by 28/11/2021
  - First submission: 24/4/2022
  - Final submission: 22/5/2022
  - **All the steps typically mean multiple iterations within multiple days. If you wait for the last minute, there is a chance you won't make it**

Code Requirements

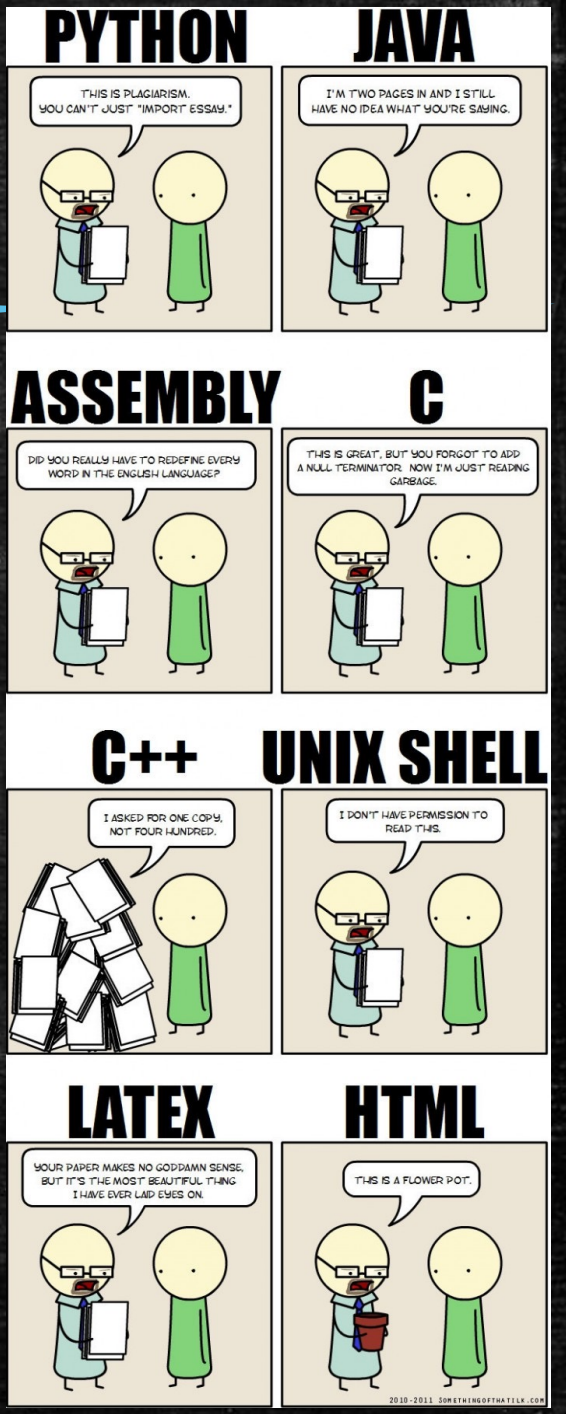
- Consistency
  - Be consistent within the code – keep a single code style
- Cleanness, readability
  - Code doesn't contain commented/dead parts
  - Code should be readable on its own
- Safe, modern
  - E.g., prefer `` `std::vector<int>` `` to `` `new int[]` ``
- Working
  - OFC, if the code is not working, all the above points are not that important, but they will help you with debugging at least ☺

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

"It was only supposed to be a joke, I never thought people would take the book seriously. Anyone with half a brain can see that object-oriented programming is counter-intuitive, illogical and inefficient."
-- Stroustrup C++ 'interview' (https://www-users.cs.york.ac.uk/susan/joke/cpp.htm)

C++ != speed, C++ ~ control



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, …

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://gcc.godbolt.org/
- ...

Hello World

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

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

Hello Wor

```
#include <ios
#include <str

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

Read from standard input (keyboard)

Write to standard output (screen)

All the STL constructs live inside `std` namespace

More Complex Program

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

using namespace std;

int length(const string& s) { ... }

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

int main(int argc, char** argv) {
  vector<string> arg(argv, argv+argc);
  if (arg.size() > 1 && arg[1] == "--help")  {
    cout << "Usage: myprg [OPT]... [FILE]..." << endl;
    return 8;
  }
  pretty_print(arg);
  return 0;
}
```

More Complex Program

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

using namespace std;

int length(const string& s) { ... }

void pretty_print(const vector<string>& a) { ... }

int main(int argc, char** argv) {
  vector<string> arg(argv, argv+argc);
  if (arg.size() > 1 && arg[1] == "--help") {
    cout << "Usage: myprg [OPT] .. [FILE]..." << endl;
    return 8;
  }
  pretty_print(arg);
  return 0;
}
```

Include the whole std namespace

Passing the argument by (const) reference

Arguments of the program on the command line

Transform the arguments into C++ array of strings

Homeworks

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