Programming in C++

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

Programming in C++ - lab 6

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

Homework feedback

- Plagiarism
- Passing large objects by (const-) reference
- Using const functions
- Warnings/cannot compile with another compiler
- Using named constants instead of any number
- Function decomposition
- "Too complex solution"
  - `vector<vector<string>>`
  - `map<string, tuple<int, int, int>>`

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 queue/array
  - push_back(), push_front(), back(), front(), …
- **std::list<Type>** - linked list
- **std::map<Key,Value>**, **std::unordered_map<Key, Value>** - map
  - my_map[key] = value, find(), insert(), …
- **std::set<Key>**, **std::unordered_set<Key>** - set
  - contains(), insert(), find(), …

Homeworks: 1. Dictionary

```
// An example of API
class Dictionary {
  // Insert a new language and returns its ID
  size_t add_language(const string &name);

  // Insert new words into a dictionary
  bool add_vocabulary(size_t words1_language_id, const string &word1,
    size_t words2_language_id, const string &word2);

  // Translate a given text with the given language into the output language
  string translate(size_t input_language_id, const string &text,
    size_t output_language_id) const;

  // Automatically translate a given text into a given language
  string translate(const string &text, size_t output_language_id) const;

  // Return all vocabularies for a given language
  const vector<string> &all_vocabulary(size_t language_id) const;
};
```

Homeworks: 2. Simple People Database

- A simple people database
  - In Recodex: https://recodex.mff.cuni.cz/

Programming in C++ - lab 5

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

Homework feedback

- Lack of responsibility ☹
  - Lots of missed deadlines
  - Not working homeworks
  - No communication
- Summing program
  - Used forbidden functions/constructs
- Integer matrix
  - `const` functions

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;
```

Homework: TicTacToe for 2 players

- For 2 players only
  - Set the names at the beginning
- Game ends when one of the player has 5 in a row
  - Write who is the winner
- Validate user inputs

Programming in C++ - lab 4

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

Homework feedback

const with Classes

```
class Person {
  const string name;
  uint8_t age;
public:
  Person(const string &name, uint8_t age) : name(name), age(age) {}

  void inc_age() {
    ++age;
  }

  uint8_t get_age() const {
    return age;
  }

  const string &get_name() const {
    return name;
  }
};
```

Homeworks

1. Create a verbose class C and show its usage
   - Prints identifier on each call to a class special method
2. Use the verbose class in another verbose class and show its usage
   - Usage:
     - a single C
     - a vector of Cs
       - Show usage – insert things into it
3. Finish the summing program
4. Finish/Fix Matrix for Integers

Summing Program

- Implement only special functions
  - ctors, dtor, operators
- You can add O(1) attributes into C
  - E.g., cannot add a vector
- Use `print()` for printing
  - Cannot use anything else for printing
- Example
  - Input: `5 7`
  - Output:

```
Numbers:
5
6
7
Preparing...
Sum of the numbers:
18
```

```
class C {
    /* CAN ADD MORE ATTRIBUTES */
    int value;

    void print() const {
        cout << value << "\n";
    }

public:
    /* IMPLEMENT SPECIAL FUNCTIONS */
};

class D {
    std::vector<C> cs;
    /* CANNOT ADD MORE ATTRIBUTES */
public:
    /* IMPLEMENT SPECIAL FUNCTIONS */
};

int main(int argc, char *argv[]) {
    int first, last;
    cin >> first >> last;
    cout << "Numbers:\n";
    D d(first, last); // prints number first, first+1, ..., last
    cout << "Preparing...\n";
    D d2 = d;
    cout << "Sum of the numbers:\n";
    d2 = d; // prints sum of numbers first..last
}
```

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

- Enable warnings

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;
    }
};

{
    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 with Classes

```
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(); }
};

size_t CountingClass::num_instances = 0; // initialization
```

```
assert(!CountingClass::has_instance() && "No instances");
{
    CountingClass cc1;
    assert(CountingClass::get_num_instances() == 1);
    {
        CountingClass cc2 = cc1;
        assert(CountingClass::get_num_instances() == 2);
    }
    assert(CountingClass::get_num_instances() == 1);
}
```

Homework1: Implement class C

- Can touch only class C
- Don't use `exit()`, `break`, `goto`, …
- Program writes: `1, 2, 3, …, 12`

```
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";
}
```

Voluntary Homework2: Finish Matrix for Integers

- Correct all issues in the previous HW
- Implement correctly all special methods
- Show usage/test all the methods with assertions
- **Not needed for the next week**

Programming in C++ - lab 2

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

Homework Feedback

- Deadlines
  - Communication!
- Use functions
  - Don't put everything into `main()`
  - Function should do a single thing
- Don't put binaries into a repo
  - Source/header files, configs, project files, …
  - Search what to put into repo on-line
- Use objects/structs
- Use (const) references for large objects

Argument Passing – By Value

- Creates a copy
- Use for small/elementary types (int, double)

```
int max(int x, int y);
```

Argument Passing – By Const-reference

- Doesn't create a copy
- Use for large objects (containers, e.g., `vector`, `string`, …)

```
int find_max(const vector<int> &numbers);
```

Argument Passing – By Reference

- Doesn't create a copy
- Allows to change the passed object
- For output parameters
  - ! Is the function really doing a single things?
  - Could you use `std::pair`?
    - `pair<iterator, bool> map::emplace(…)`

```
void transform(Matrix &matrix);

pair<size_t, int> find_max(const vector<int> &numbers);
```

Argument Passing – By R-value Reference

- To transfer a ownership
- Moves the object into a function
  - the object no longer lives outside the function
- 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));
```

Class/Struct

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

```
class calculator {
    void sum(); // private by default

  public:
    // change internal attributes, cannot be const
    void calc(const std::string &expression);

    // doesn't change internals, should be const
    void print() const;

  private:
        // ...
    };
};

calculator c; // no need for `new`
c.calc("1+2*3/4");
```

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

std::vector<T>

```
#include <vector>
int main() {
  std::vector<int> vi{1, 2, 3, 4, 5, 6};
  std::vector<float> vf(5, 0.0f);
  std::cout << vi[3] << " " << vf.at(3) << std::endl;
  std::cout << vi.size();
  vi[3] = 100; vi.at(6) = 600;
  vf.push_back(100.0f); vf.emplace_back(200.0f);
  vf.insert(3, 300.0f); vf.emplace(3, 300.0f);
  vi.pop_back();
  vf.erase(3);
  vi.clear();
  vi.resize(10); vi.reserve(100);
}
```

- Beware of time complexity
- vector<bool>

Homework: Matrix for Integers

- set(x, y, value), get(x, y), print()
- set_width(), set_height(), get_width(), get_height()
- get_row(x), get_column(x)
- get_rows(), get_columns()
- clear() – set all values to 0 (zero)
- fill_with_value(value) – set all values to a given value
- reverse() – reverse values from [x, y] to [y, x]
- is_negative() – are all numbers in the matrix negative?
- zero_count()

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

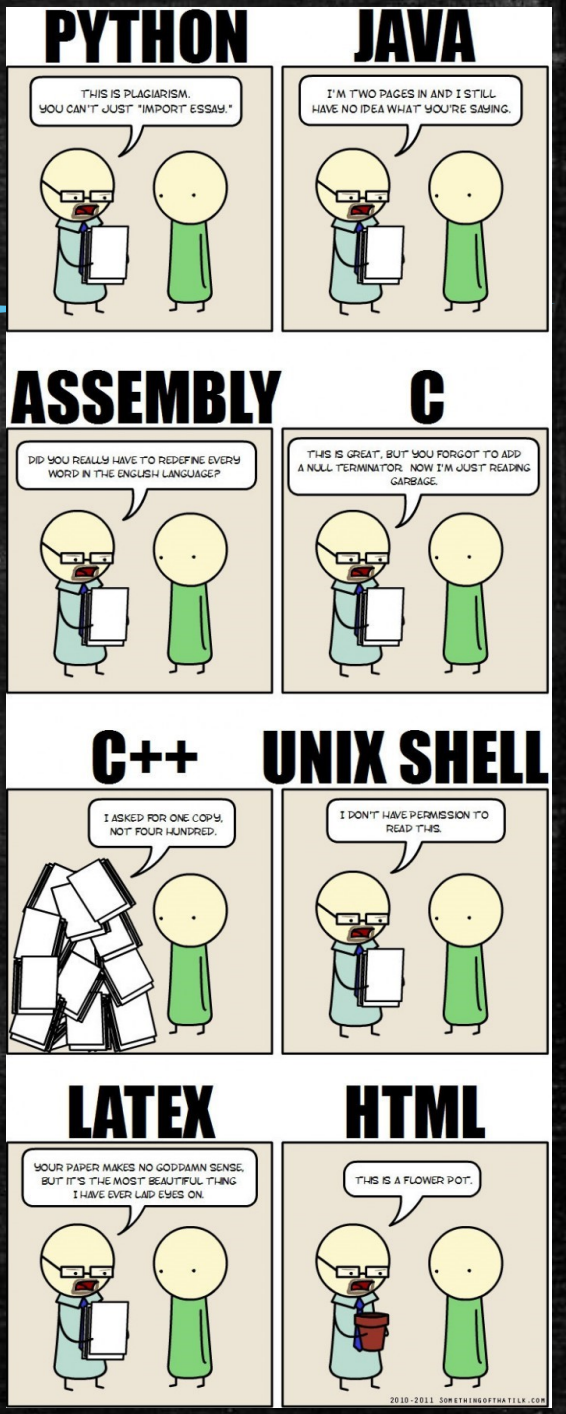
- Consistency
  - Be consistent within the code – keep a single code style
- Cleanness, readability
  - Code doesn't contain commented/dead parts
  - Code should be readable on its own
- Safe, modern
  - E.g., prefer `` `std::vector<int>` `` to `` `new int[]` ``
- Working
  - OFC, if the code is not 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 <iost
#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