# C++ cvičení

8.10.2018

faltin@ksi.mff.cuni.cz

# "Go to operator"

```cpp
// Working
int i = 100;
while (i --> 0) {
  std::cout << i;
}


// Not working (can compile)
int i = 0;
while (i --> 100) {
  std::cout << I;
}
```

# "Go to operator"
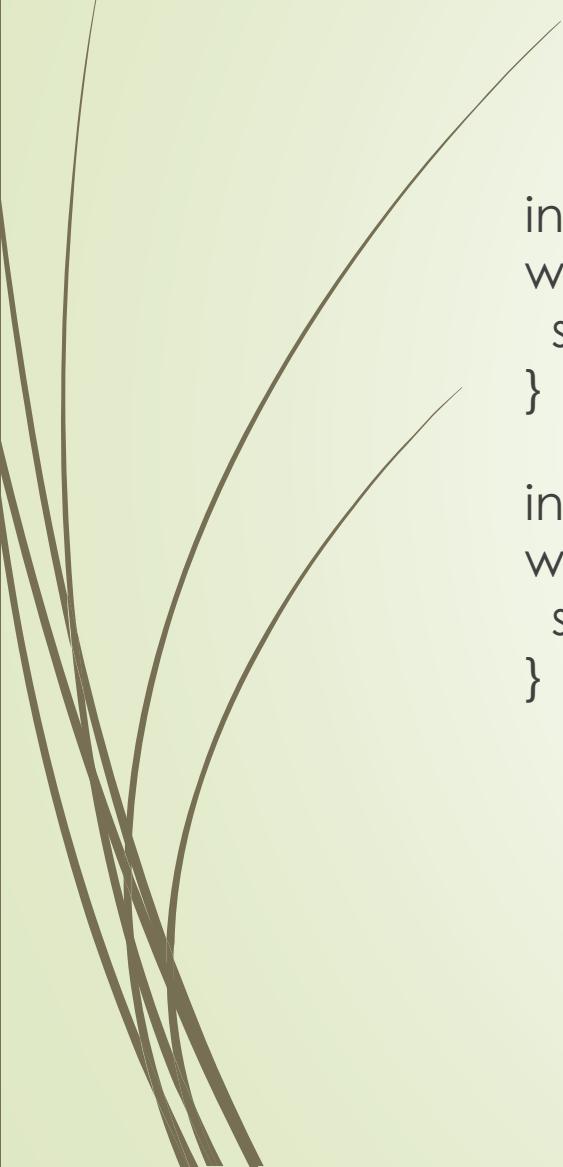
```cpp
int i = 100;
while (i --> 0) {
  std::cout << i;
}


int i = 100;
while ((i--) > 0) {
  std::cout << i;
}
```

# Trigraphs

```
int main() {
  // Why is this not working :( ?????/
  std::cout << "Hello World" << std::endl;
}
```

# Alternative tokens, trigraphs

- **Trigraphs removed in C++17 ☺**

- [https://en.cppreference.com/w/cpp/language/operator_alternative](https://en.cppreference.com/w/cpp/language/operator_alternative)

```
and -> &&
not -> !
or -> ||
xor -> ^
…

??/ -> \
??< -> {
??> -> }
```

# class/struct

```
class Complex {
 int r_; // private by default

public:
  Complex(int r, int i) : r_{r} { i_ = i; }
private:
  int i_;
};

struct S {
  int x;  // public by default
  int y;
};
```

**Use class if the class has an invariant; use struct if the data members can vary independently**

http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c-classes-and-class-hierarchies

http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#c2-use-class-if-the-class-has-an-invariant-use-struct-if-the-data-members-can-vary-independently

# class/struct

```
struct C {
  C() { std::cout << "default ctor" }
  C(const C &c) { std::cout << "copy ctor"; }
  C &operator=(C &c) { std::cout << "copy op"; return *this; }
  ~C() { std::cout << "dtor"; }

  void fn_mutable() { … }
  void fn_immutable() const { … }

  C(C &&c) { std::cout << "move ctor"; }
  C &operator=(C &&c) { std::cout << "move op"; return *this; }
};

void fn_par_by_copy(C c) { … }
void fn_par_by_cref(const C &c) { … }
void fn_par_by_ref(C &c) { … }
void fn_par_by_rref(C &&c) { … }
C fn_ret_by_copy() { … }
const C &fn_ret_by_cref() { … }
C &fn_ret_by_ref() { … }
C && fn_ret_by_rref() { … }
```

# std::vector<T>

```cpp
#include <vector>
int main() {
  std::vector<int> vi{1, 2, 3, 4, 5, 6};
  std::vector<float> vf(10, 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);
}
```

➡ vector<bool>

# Úkoly
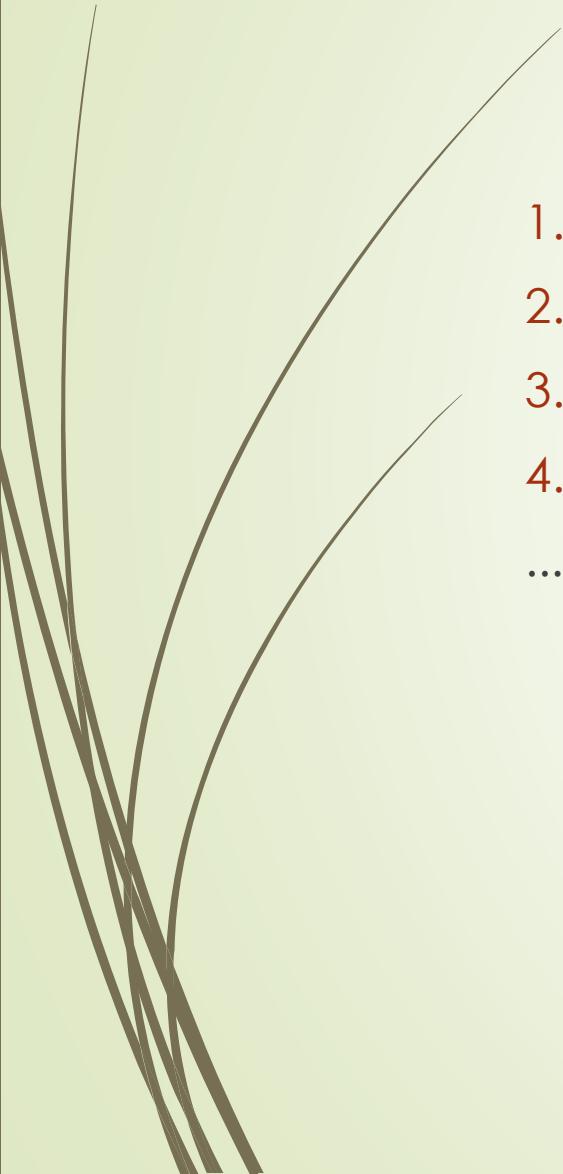
1. Zkusit si, jak fungují konstruktory
2. Zkusit si, jak fungují konstrukty + vector
3. Naimplementuj třídu C
4. Vytvořit třídu Complex

...

# Úkol 3

```cpp
void fn_copy(C c) {}
void fn_ref(const C &c) {}

// Print 1, 2, …. 20
int main() {
  std::cout << "1";
  C c;
  std::cout << "5";
  fn_copy(c);
  std::cout << "10";
}
```