Cvičení z C++

7.11.2019

faltin@ksi.mff.cuni.cz

Úkol 1

- std::getline

Úkol 2

- ReCodex slouží pouze k odevzdávání
- Deadline je těsně před termínem první zkoušky

Politiky (1/2)

- “compile-time strategy pattern”
- Policy based design

Politiky (2/2)

```
template<typename T, class Allocator = DefaultAllocator>
class my_vector {
T *data_;
size_t size_;

public:
  my_vector(size_t size) : data_(Allocator::alloc<T>(size)), size_(size) {}
  ~my_vector() { Allocator::dealloc<T>(data_); }
};

struct DefaultAllocator {
  template<typename T>
  static T *alloc(size_t size) { return new T[size]; }

  template<typename T>
  static void dealloc(T *ptr) { delete[] ptr; }
};

struct Mallocator {
  template<typename T>
  static T *alloc(size_t size) { return std::malloc(sizeof(T) * size); }

  template<typename T>
  static void dealloc(T *ptr) { std::free(ptr); }
};
```

- Hledej: policy based design, policy

Traits (1/2)

Think of a trait as a small object whose main purpose is to carry information used by another object or algorithm to determine "policy" or "implementation details". - Bjarne Stroustrup

```
template<typename T>
struct is_integral;

template<>
struct is_integral<uint8_t> {
  static constexpr bool value = true;
};

template<>
struct is_integral<uint16_t> {
  static constexpr bool value = true;
};

…

template<typename T>
struct is_integral {
  static constexpr bool value = false;
};
```

Traits (2/2)

```
template<class T>
struct numeric_limits;

template<>
struct numeric_limits<char> {
  static constexpr char min() { return 0; }
  static constexpr char max() { return 255; }
};

…
```

- type_traits, numeric_limits, …

static_assert()

```
template<typename T, size_t Size>
struct my_array {
  static_assert(std::is_default_constructible_v<T>,
      "Type T must be default constructible");

  T data[Size];
};
```

SFINAE + enable_if

- SFINAE = substitution failure is not an error

```
template<typename T>
struct MyClass {
    void f(T const& x){}

    template<typename T_ = T>
    void f(T&& x,
           typename std::enable_if<!std::is_reference<T_>::value,
           std::nullptr_t>::type = nullptr){}
};
```

Concepts

A concept is a named set of requirements. The definition of a concept must appear at namespace scope.

```
// concept
template <class T, class U>
concept Derived = std::is_base_of<U, T>::value;
```

Constraints

A constraint is a sequence of logical operations and operands that specifies requirements on template arguments.

```
template<Incrementable T>
void f(T) requires Decrementable<T>;

template <class T = void>
requires EqualityComparable<T> || Same<T, void>
struct equal_to;
```

???

- https://stackoverflow.com/questions/11227809/why-is-it-faster-to-process-a-sorted-array-than-an-unsorted-array

CRTP

- https://eli.thegreenplace.net/2011/05/17/the-curiously-recurring-template-pattern-in-c

Sources

- https://en.cppreference.com
- https://en.cppreference.com/w/cpp/language/constraints