Modern C++

Jan Šimeček (SIM0178)

Timeline

C++98

1998

- STL (containers, algorithms)
- Strings
- I/O Streams

C++11

2011

- Move semantic
- List initializers
- Lambda functions
- **`auto`** and **`decltype`**
- Multithreading
- Smart pointers
- Default/delete/copy constructors
- Tuples, variadic templates
- User-defined literals
- Regular expressions
- **`std::array`**

C++14

2014

- Generalized lambda functions
- Type deduction
- Reader-writer locks

C++17

2017

- Attributtes ([[deprecated]])
- Inline variables
- Template deduction of constructors
- File system TS
- **`std::any`**, **`std::optional`**, **`std::variant`**

Usability enhancements

- Automatic type deductions (`auto`, `decltype`), iterator based `foreach` loops

- Rvalues (`&&T`), move semantic (`std::move`), control of defaults (**default**, **copy**, **move** constructors), list initializers, delegating constructors

```
std::vector<int> indices = {1, 2, 3, 4, 5, 6};
```

- Tuples, user-defined literals

```
double x = 90.0_deg; // x contains value in radians
auto student = std::make_tuple(178, "VSB-FEI", "Jan Simecek");
```

Lambda functions

- `std::bind` and `std::function` are used together to handle functions and function arguments

- A lambda expression is mechanism for specifying a function object. The primary use is to specify simple action to be performed by some function. Lambdas can access local variables within the defined scope. It is defined as `[](int x) { return x * x; }`

```
std::sort(v.begin(), v.end(), [](int a, int b) {
    return std::abs(a) < std::abs(b);
});
```

Smart pointers

- Smart pointers provide automatic object deallocation (i.e. automatically calling delete), when the object is not used anymore

- `std::unique_ptr` - strict ownership, owns the object it holds a pointer to (relies critically on move semantic)

- `std::shared_ptr` - shared ownership, e.g. when two pieces of code needs access to some data but neither has exclusive ownership

- `std::weak_ptr` - holds (non-owning) reference to an object that is managed by `std::shared_ptr`

Multithreading

- C++11 includes built-in support for threads, mutual exclusions, condition variables and atomics

- Threads can be created using `std::thread` class provided in `<thread>` header file which allow multiple functions to be executed concurrently

- Atomic `std::atomic` operations library provides components allowing lockless concurrent programming

- A `std::mutex` is primitive object used for controlling access in concurrent programming (`lock`, `unlock`)

Time and Generators

- `std::chrono_library` defines types and utility functions to track time with varying degrees of precision

- The `<random>` library provides several classes that generate random and pseudo-random numbers. Ranging from many random number engines, generators to several random number distributions

```cpp
auto start = std::chrono::system_clock::now();
// ...
auto end = std::chrono::system_clock::now();

std::chrono::duration<double> elapsedSeconds = end - start;
std::cout << "elapsed time: " << elapsedSeconds.count();
```

Thank you for your attention