

CH-231-A

Algorithms and Data Structures

ADS

Lecture 5

Dr. Kinga Lipskoch

Spring 2024

C++ Evolution

- ▶ Until 1989 Annotated C++ Reference Manual (ARM C++)
- ▶ 1990 - 1998 C++98 with addition of STL in 1995
- ▶ C++0x development started in 2002
 - ▶ C99
 - ▶ Boost Library
 - ▶ Library Extension TR1

C++11 (C++0x)

C++ is a general-purpose programming language with a bias towards systems' programming that

- ▶ Is a better C
- ▶ Supports data abstraction
- ▶ Supports object-oriented programming
- ▶ Supports generic programming

Compile with the option `-std=c++11` or `-std=c++0x`

Example: `g++ -std=c++11 -Wall -o test test.cpp`

B. Stroustrup: Goals of C++11

- ▶ Make C++ a better language for systems' programming and library building
 - ▶ Build on C++'s contributions to programming
 - ▶ Not providing specialized facilities for a particular sub-community (e.g., numeric computation or Windows-style application development)
- ▶ Make C++ easier to teach and learn
 - ▶ Increased uniformity
 - ▶ Stronger guarantees
 - ▶ Facilities supportive of novices: there will always be more novices than experts

C++11 Aims

- ▶ Maintain stability and compatibility
- ▶ Prefer libraries to language extensions
- ▶ Prefer generality to specialization
- ▶ Support both experts and novices
- ▶ Increase type safety
- ▶ Improve performance and ability to work directly with hardware
- ▶ Fit into the real world

Maintain Stability and Compatibility

- ▶ Billions of lines of existing code, which should not be broken
- ▶ But new keywords such as:
 - ▶ `auto` – example later
 - ▶ `decltype` – `decltype.cpp`
 - ▶ `constexpr` – example later
 - ▶ `nullptr` – `nullptr.cpp`are included as needed
- ▶ But many new features via libraries

auto vs. decltype

```
1 int& foo() {  
2     ...  
3 }  
4  
5 decltype(foo()) a = foo();    // int&  
6 auto b = foo();               // int  
7 auto& c = foo();              // int&
```

- ▶ `auto` determines value types
- ▶ `decltype` needs expression

Support both Experts and Novices

- ▶ Nested containers are allowed
 - ▶ `vector_list.cpp`
- ▶ New keyword `auto` creates easier to read code
 - ▶ `list_old.cpp`
 - ▶ `list_auto.cpp`
 - ▶ `list_range_for.cpp`

Improvements in the Standard Library

- ▶ New initializers – `initializer.cpp`
- ▶ Lambda-functions – `auto-lambda.cpp`
 - ▶ Anonymous functions
 - ▶ Allows to specify comparison function where it is needed
 - ▶ `[] () ->`
 - ▶ capture, parameter list, return type, function body
 - ▶ `lambda.cpp`

Variadic Functions

- ▶ To access the variadic arguments from the function body, library facilities are provided (`<stdarg.h>`):
 - ▶ `va_start` – enables access to variadic function arguments
 - ▶ `va_arg` – accesses the next variadic function argument
 - ▶ `va_copy` – (C++11) makes a copy of the variadic function arguments
 - ▶ `va_end` – ends traversal of the variadic function arguments
 - ▶ `va_list` – holds the information needed by `va_start`, `va_arg`, `va_end`, and `va_copy`
- ▶ `variadic_function.cpp`

Variadic Templates

Allow to handle arbitrary number of template parameters

- ▶ `variadic_templates.cpp`
- ▶ `f()` takes arbitrary number of parameters and returns its number
- ▶ `printCommaSeparatedList()` expects one or more parameters and returns them in a comma separated list
- ▶ new operator `sizeof...`
- ▶ recursive call to `printCommaSeparatedList()`

Tuples

- ▶ `pair` can be expanded to `tuple` now
- ▶ It is more general
- ▶ `tuple.cpp`

Constant Expressions

- ▶ Sometimes compiler needs constant to e.g., create an array
 - ▶ `int vals[4];`
 - ▶ `Array<SZ> arr;`
- ▶ But not
 - ▶ `int val[getsize()];`
 - ▶ `Array<std::max(3, 4)>`
- ▶ New keyword
 - ▶ `constexpr`

constexpr

- ▶ Determine expression's value at **compile time**
- ▶ Otherwise throw error
- ▶ May be declared as `constexpr`:
 - ▶ variables
 - ▶ functions
 - ▶ constructors
 - ▶ static methods
- ▶ `const_expr.cpp`

static_assert

- ▶ Allows to use assertions at compile time
 - ▶ possible before by using the Boost library or preprocessor
- ▶ `static_assert.cpp`