Standard library overview

- content is overall fairly traditional, e.g.
  - containers (data structures)
  - algorithms (standard functions)
  - input and output (streams)
- new features in C++11
  - threads
  - regular expressions
  - the whole standard library is adapted to support move semantics and perfect forwarding
- may seem complicated, but a closer look shows it to be
  - flexible
  - general
  - extendible
  - efficient
- the implementation uses several advanced core language features, often in combination, e.g.
  - templates, including the new feature variadic templates
  - derivation (single, multiple, repeated, virtual)
  - operator overloading
  - rvalue references (move semantics, perfect forwarding,…)
  - type traits, template meta programming

© 2015 *Tommy Olsson, IDA, Linköpings universitet*

Standard library content

- **Language support** – program start/termination, limits, dynamic memory handling, initializer lists, runtime support
- **Diagnostics** – components used to detect and report error conditions – exceptions and assertions
- **General utilities** – components used by other elements of the standard library and may be used by programs
  - pair, tuples, type traits, function objects, function call wrappers, memory allocation, smart pointers, date and time, functions for move semantics and forwarding, type traits
- **Strings** – components for manipulating sequences of characters, numeric conversions
- **Localization** – components programs may use to encapsulate cultural differences.
  - character classification, string collation, numerics, monetary and date/time formatting and parsing
- **Containers** – components used for organizing collections of information – sequence containers, adaptors, associative containers
- **Iterators** – components used to perform iterations over containers and stream buffers
- **Algorithms** – components used to perform algorithmic operations on containers and other sequences
- **Numerics** – components used to perform numerical operations, random number generators, conformance with C99 functions
- **Input / output** – components for input/output operations
- **Regular expressions** – components to perform operations making regular expressions matching an searching
- **Atomic operations** – components for fine-grained atomic access
- **Thread support** – components to create and manage threads, perform mutual exclusion, and communicate conditions between threads

© 2015 *Tommy Olsson, IDA, Linköpings universitet*

C++1y Standard Library

ISO/TS Technical Specifications – to be part of the standard, or withdrawn:

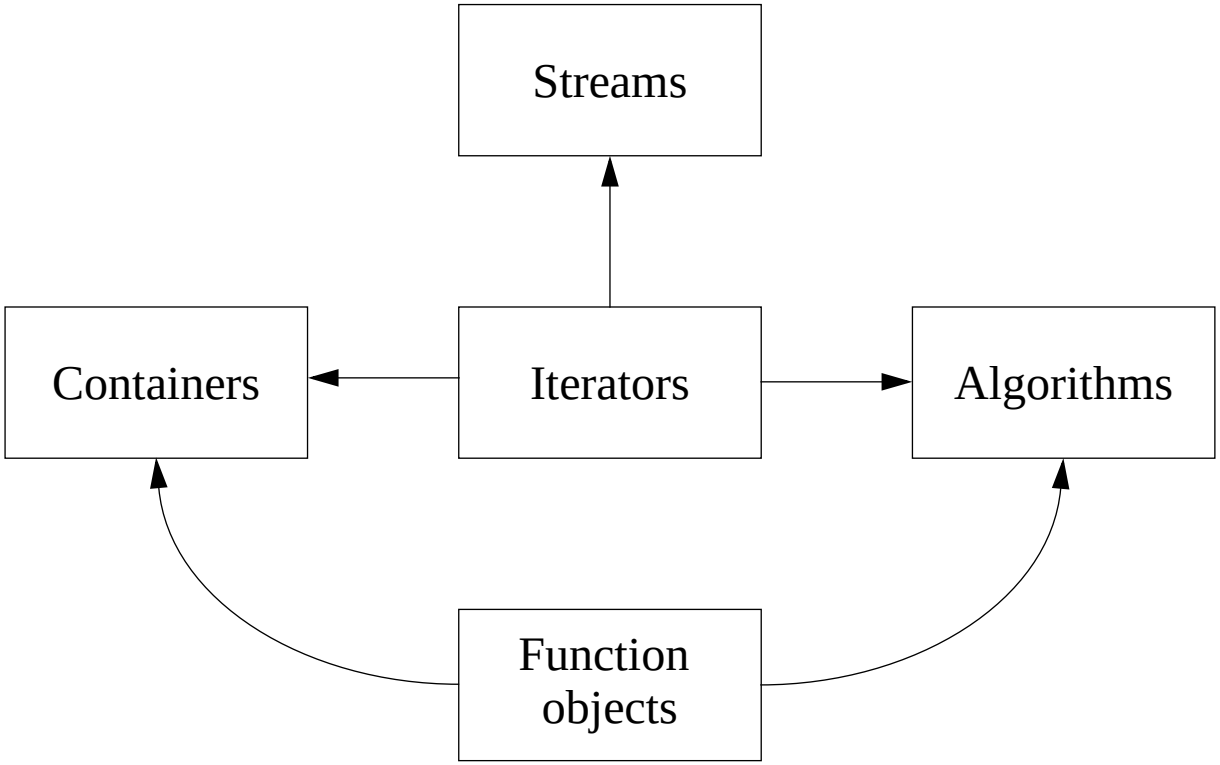
- Library fundamentals – e.g. classes and functions likely to be used widely within a program
- File system – standardized file access (paths, regular files, directories)
- Parallelism – vector and multi-core support
- Concurrency – tasks, thread pools
- Concepts – extensions to enable specification and checking of constraints on template arguments
- Arrays – dynamic arrays
- Transactional memory – STM (Software Transactional Memory)
- Ranges – support for range-based versions of the standard algorithms
- Networking

© 2015 *Tommy Olsson, IDA, Linköpings universitet*

Standard library in this course

Main focus in this course:

- things related to data structures and algorithms
  - containers
  - algorithms
  - iterators
  - function objects
  - lambda expressions (core language)
  - related utilities
- streams and strings
  - general streams
  - file streams
  - string streams
  - std::string is a kind of container
- smart pointers



© 2015 *Tommy Olsson, IDA, Linköpings universitet*

Example: containers – iterators – algorithms – function objects – streams

```
#include <algorithm>    // copy, sort, unique, for_each
#include <functional>   // greater, unary_function
#include <iostream>     // ostream, cin, cout
#include <iomanip>      // setw
#include <iterator>     // istream_iterator, ostream_iterator
#include <numeric>      // accumulate
#include <vector>       // vector
```

// Function object to be used in algorithms for printing (suitable) objects of type T to an ostream

```
template<typename T>
class printw
{
public:
   printw(std::ostream& os, int width = 6) : os_{ &os }, width_{ width } {}

   void operator()(const T& x) const
   {
      *os_ << std::setw(width_) << x << ((++n_ % 10 == 0) ? "\n" : "");
   }

private:
   std::ostream* os_;
   const int     width_;
   mutable int   n_{ 0 };
};
```


© 2015 *Tommy Olsson, IDA, Linköpings universitet*

```
int main()
{
    using namespace std;

    vector<int> v{ istream_iterator<int>{cin}, istream_iterator<int>{} };

    copy(begin(v), end(v), ostream_iterator<int>{cout, " "});

    sort(begin(v), end(v));

    v.erase(unique(begin(v), end(v)), end(v));

    sort(begin(v), end(v), greater<int>{});

    for_each(begin(v), end(v), printw<int>{cout, 4});

    cout << "\nSum = " << accumulate(begin(v), end(v), 0) << '\n';
}
```


© 2015 *Tommy Olsson, IDA, Linköpings universitet*