Object Oriented Scientific Programming in C++ (WI4771TU)

Matthias Möller

Numerical Analysis

Goal of todays lecture

- Understand container classes from standard C++ library
  - Extensive use of template metaprogramming
- Understand concept of iterators
  - Flexible alternative to for-loops
- Learn some of the standard algorithms (sort, merge, ...)
  - Example for use of lambda expressions

Containers

- **Aim:** provide a set of universal container classes that
  - can **store arbitrary types** (in general, only objects of the same type in each container; see std::tuple for multi-type containers)
  - provide a **uniform interface** to access, manipulate, insert, delete items and iterate over the items stored
  - provide optimal implementations of **standard data structures**, e.g., double-linked lists, balanced trees (red-black tree)

Containers, cont‘d

- Array: array with compile-time size (non-resizable)
- Vector: array with run-time size (resizeable)
- List: double-linked list
- Forward_list: single-linked list
- Stack: Last-In-First-Out stack
- Queue: First-In-First-Out queue
- Set/Multiset: Set of unique elements with a specific order
- Map/Multimap: Set of (key,value) elements

Containers, cont‘d

- Container classes support the following base functionality
  - size(): return the size of the container
  - empty(): return true of the container is empty
  - swap(container& a, container& b): swap contents of containers
- Many container classes provide so-called iterators
  - begin(), end(): editable iterator
  - cbegin(), cend(): constant, i.e., non-editable iterator

Algorithms

- Header file <algorithm> provides many standard algorithms
  - p = find(b, e, x)
  - p = find_if(b, e, f)
  - n = count(b, e, x)
  - n = count_if(b, e, f)
  - sort(b, e)
  - sort(b, e, f)
  - p = merge(b1, e1, b2, e2, out)