STL Algorithms

and then some

Overview

- Design Rationale
- Basic algorithm scheme
- My top 5
- Hottest algorithms
- (I loved this: Sean Parent on Channel 9 09:00)

Design rationale

- Reusable
  - ‘orthogonal’ design
    - data type is abstracted (function templates)
    - iteration is abstracted
- Optimized (by pro’s)
- Building blocks
  - greatest common divisor
  - Compose programs by using algorithms
- Declare Your Intent
  - reverse-engineering a for loop is error prone and hard
- (extensible: make your own!)

Design rationale: abstract data type

C++ function templates!

Design rationale: abstract iteration

- Iterator concept
  - forward (++it)
  - bidirectional (--it)
  - input/output (value=*it/*it=value)
  - random-access (it += 10)
- Range concept: begin/end pairs
  - std::copy(begin(xs), end(xs), …)
  - ! end is *exclusive*
- :(no range library
  - but: boost::algorithm works on boost::range!
    - auto five = std::find(begin(xs), end(xs), 5);
    - auto five = boost::find(xs, 5);

Design rationale: abstract iteration

- Iterator concept
  - forward (++it)
  - bidirectional (--it)
  - input/output (value=*it/*it=value)
  - random-access (it += 10)
- Range concept: begin/end pairs
  - std::copy(begin(xs), end(xs), …)
  - ! end is ***exclusive***
- :(no range library
  - but: boost::algorithm works on boost::range!
    - auto five = std::find(begin(xs), end(xs), 5);
    - auto five = boost::find(xs, 5);

Basic Algorithm Scheme

- source range(s) with input iterator pairs
- target range(s) with output iterator
- extra: predicates, lambda, compare-to values, ...
- output: mostly an iterator
- preconditions!

```
transform(
    begin(xs), end(xs),
    ostream_iterator<string>(cout, ", "),
    [](const Address &a) { return a.zipcode; }
);
```

Basic Algorithm Scheme

- Some algo’s have preconditions!
  - sorted input
    (check with std::is_sorted(...))
    - merge, equal_range, lower_bound, …
    - set_intersection
    - ...
  - no side-effects
    - std::accumulate
- Read the reference

Basic Algorithm Scheme

- non-modifying:
  - Single-range:
    <it> = algorithm(<begin>, <end>, …)
    - ```
      auto it = std::find(begin(xs), end(xs), “abcd”);
      if (it == end(xs)) throw std::invalid_argument(“”)
      auto &value = *it;
      ```
  - Dual-range:
    <it1, it2> = algorithm(<begin1>, <end1>, <begin2>, …)
    - ```
      auto its = mismatch(begin(xs), end(xs), begin(ys) );
      if (its.first != end(xs))
          cout << “mismatch at “
               << distance(begin(xs), its.first) << endl;
      ```

Basic Algorithm Scheme

- modifying:
  - Self-modifying
    <out_it> = rotate(<begin>, <newbegin>, <end>)
  - General modifying
    <out_it> = algorithm(<begin>, <end>, <out>)
    - `copy(begin(xs), end(xs), back_inserter(output));`

My top 5

Yes, I cheated

- transform
  - incoming `vector<A>`, outgoing `vector<B>`
- fill, fill_n, copy, copy_n
- find, find_if
- any_of/all_of/none_off
- mismatch

Hottest algorithms

I admire them but I don't often get a chance to use them

- nth_element
  - which are the fastest 5 in this vector?
  - also: minmax and minmax_element
- adjacent_find
  - where's the border?
  - find zero-crossings
- adjacent_difference
  - discrete differentiation
- accumulate
  - basic: summing
  - advanced: the first step towards folding
- set_difference/union/intersection

Hottest algorithms

I admire them but I don't often get a chance to use them

- nth_element

```
int xs[] = {1, 30, 400, 2, 3, 100, 40, 33, 13};

const auto nth = std::next(std::begin(xs), 3);
std::nth_element(
    std::begin(xs),
    nth,
    std::end(xs),
    std::greater<int>());

std::copy_n(
    std::begin(xs), 3,
    std::ostream_iterator<int>(std::cout, ", "));
```

Hottest algorithms

I admire them but I don't often get a chance to use them

- nth_element
  - which are the fastest 5 in this vector?
  - also: minmax and minmax_element
- adjacent_find
  - [illegible]
  - [illegible]
- adj
  - discrete differentiation
- accumulate
  - basic: summing
  - advanced: the first step towards folding
- set_difference/union/intersection

```
std::adjacent_find( std::begin(xs), std::end(xs),
    [](int i1, int i2){ return i1 < 10 && i2 > 50; });
```

Hottest algorithms

I admire them but I don't often get a chance to use them

- nth_element
  - which are the fastest 5 in this vector?
  - also: minmax and minmax_element
- adjacent_find
  - where's the border?
  - find zero-crossings
- adjacent_difference
  - discrete diffe
- accumulate
  - basic: summ
  - advanced: th
- set_difference

```
std::string ss[] = {"abc", "def", "ghi"};
auto total_length =
    std::accumulate(std::begin(ss), end(ss),
        0,
        [](int total, const std::string &s){
            return total + s.size();
        });
```

Conclusion

Read up!

http://en.cppreference.com/w/cpp/algorithm