Algorithms in STL

Algorithms in STL

- Overview of algorithms that act on containers
- Categories of algorithms
- Complexity Analysis issues to determine algorithm performance
- Using algorithms in applications

2

Header Files

- `<algorithm>` - the STL algorithms
- `<numeric>` - some numeric algorithms
- `<functional>` - for function objects and function adapters
- And auxiliary functions, e.g. `max()`, `min()`, `swap()`

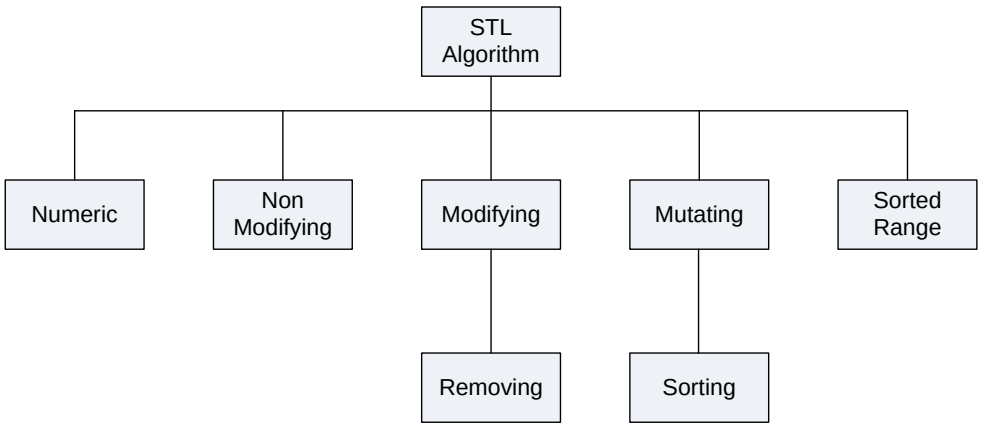
3

Prerequisites

- Template classes and template functions
- Function objects, input arguments and return types
- Iterators
- General data structures; complexity analysis, searching, sorting
- Knowing how to define sorting criteria

4

Algorithm Categories



5

Non-modifying Algorithms

- Change neither order nor values of elements in containers
- Used with input (read access) and forward iterators
- Can be used for all standard containers
- Another sub category: non-modifying algorithms for sorted input ranges (later)

6

Examples
Non-modifying Algorithms

- `for_each()`: perform an operation on each element (read-only)
- Counting number of elements satisfying criteria
- Finding 1st element satisfying criteria
- Search for occurrences of a sub range
- Equality and mismatches of ranges
- Min and max elements

7

Modifying Algorithms

- Changes the value of elements
- Can also change the elements of a range while being copied into another range
- Impact of performance issues with some algorithms
- Major ones: `for_each()` (read-write) and `transform()`

8

Examples
Modifying Algorithms

- Copy ranges
- Merge ranges
- Replace elements in ranges
- Replace elements with the result of an operation
- More advanced replace algorithms

9

Removing Algorithms

- Special kind of modifying algorithms
- Remove elements in a range or while copying into another range
- Cannot use associative containers as destination (elements are constant)
- !! Remove logically and remove physically (`remove()` versus `erase()`)

10

Examples

Removing Algorithms

- Remove elements with given value or match a criterion
- Remove adjacent duplicates
- Copy elements while removing adjacent duplicates
- Copy elements that do not match a given criterion

11

Mutating Algorithms

- Change order of elements (not values)
- Assign and swap values
- Cannot use associative as destination (elements are not constant)
- Useful for certain kinds of applications

12

Examples

Mutating Algorithms

- Reverse the order of elements
- Rotate the order of the elements
- Element permutations
- Random shuffling
- Change order of elements based on some criterion

13

Sorting Algorithms

- Special kind of mutating algorithms (change order of elements)
- More complicated than simple mutating algorithms
- Worse than linear complexity
- These algorithms require random access iterators

14

Examples
Sorting Algorithms

- Sort all elements
- Sort and preserve order of equal elements
- Partial sort
- Convert a range to a heap (also add element and remove an element from a heap)
- Sort a heap

15

Sorted Range Algorithms

- Apply to containers that are sorted according to their sorting criteria
- These algorithms have better complexity
- Algorithms can be non-modifying or modifying (output result is created)
- In general, algorithm results are also sorted

16

Numeric Algorithms

- These algorithms combine numeric elements in different ways
- More powerful and flexible that they seem at first sight (!)
- Good use of operator overloading possibilities

17

Examples
Numeric Algorithms

- Combine all element values (sum, products etc.)
- Inner products (operator overloading)
- Partial sums
- Adjacent difference (combine elements with their predecessor)

18

Function Objects

- Part of STL/C++
- Used with STL algorithms
- Can be used as sorting criteria
- Can have internal state
- Special case: predicates (return bool)
- Predefined STL functions objects
- C++0X Lambda functions

19

String Algorithms

- Not much functionality
- Better to use Boost String Algo library
- Converting strings upper/lower case
- Removing, trimming
- Finding substrings
- Substituting a string by another string
- Splitting and joining strings

20

Summary

- Overview of algorithm categories in STL
- Useful to know/use but a long term project in general
- Mathematical algorithms (accumulate, inner product, partial sum, adjacent_difference)
- For some apps, sorting algorithms can be useful

21

Copy Algorithm

- Mutating sequence algorithm
- Simple way to produce one sequence from another
- Possible to copy elements between different container types
- Use back_inserter class to wrap iterator of destination container
- Use ostream_iterator class to wrap an output stream like cout

1

Example
Copy Algorithm

```
vector<double> src;    // Vector source
list<double> des;      // List destination

// Fill source.
src.push_back(1);
src.push_back(2);
src.push_back(3);

// Copy source to destination.
copy(src.begin(), src.end(), back_inserter(des));

// Copy destination to cout.
copy(des.begin(),des.end(), ostream_iterator<double>(cout, "\n"));
```

2

Find Algorithm

- Nonmutating sequence algorithm
- Finds a value in a sequence
- Value type must support operator ==
- If value is not found, find returns iterator to end of sequence (this is not NULL!)

3

Example
Find Algorithm

```
vector<double> v(3);

// Initialize vector.
v[0] = 1;
v[1] = 2;
v[2] = 3;

// Find number 2.
vector<double>::iterator result = find(v.begin(), v.end(), 2);

if (result != v.end())
{
  // Found it.
}
```

4