Generic Programming

Karl Lieberherr

What is it?

- Goal: to have libraries of generic or reusable software components
- *reusable* means *widely adaptable, but still efficient*
- adaptation is done by programming language mechanism rather than manual text editing.

What is it?

- Focus on high degree of adaptability and efficiency
- Essential idea:
  - Generic algorithms, not self-contained, use container access operations
  - Container classes with iterator classes

What is it?

- Expressing algorithms with minimal assumptions about data abstractions, and vice versa, thus making them as interoperable as possible
- Lifting of a concrete algorithm to as a general level as possible without losing efficiency

What is it?

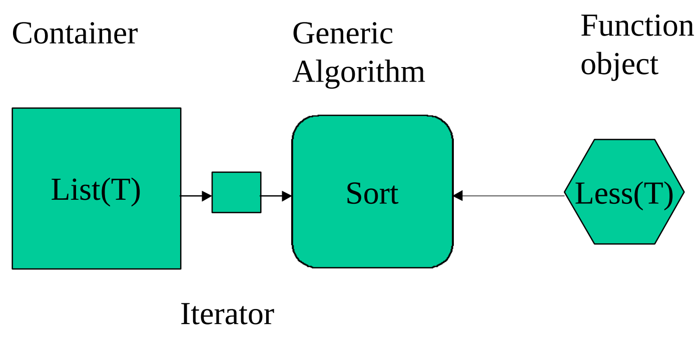
- Lifting of a concrete algorithm to as a general level as possible without losing efficiency i.e., the most abstract form such that when specialized back to the concrete case the result is just as efficient as the original algorithm.
- From Dagstuhl 98 conference on generic programming

What is it?

- Generic programming is about making programs more adaptable by making them more general
- Embody non-traditional kinds of polymorphism
- Parameters of a generic program are rich in structure (programs, types, graphs).
- From Workshop on Gen. Prog. Sweden 98

Overview of STL components

- Containers
  - Sequence containers
  - Sorted Associative Containers
- Generic Algorithms
  - find, merge, search, copy, count, sort, accumulate
- Iterators
- Function Objects
- Adaptors
- Allocators



Using STL is about plugging the right components together.

Iterators

- Input Iterator: reading
- Output Iterator: writing
- Forward Iterator: Input Iterator, Output Iterator, traversal in one direction
- Bidirectional Iterator: Forward Iterator plus bidirectional traversal
- Random Access Iterator: Bidirectional Iterator, constant time access

```
template <class InputIterator, class T>
InputIterator find(InputIterator first,
  InputIterator last,
  const T& value) {
  while (first != last && *first != value)
    ++ first;
  return first;
}
```

Works with any input iterator since (1) it only applies !=, * and ++ to its iterator parameters (2) it never tries to assign to objects it obtains using * and (3) it is a single pass algorithm.

Designing Generic Algorithms

Constant versus Mutable Iterators

- Forward, bidirectional and random-access iterators can be *mutable* or *constant*.

Iterator categories provided by containers

Container	*Iterator*	*Iterator category*
vector<T>	vector<T>::iterator	Mutable random access
vector<T>	vector<T>::const_iterator	Constant random access
list<T>	list<T>::iterator	Constant bidirectional

Generic Algorithms

- Basic algorithm organization
  - in-place: places result into same container
  - copying: copies result to a different container
- Nonmutating Sequence Algorithms
  - find, adjacent_find, count, for_each,…
- Mutating Sequence Algorithms
  - copy, copy_backward, fill, generate, ...

Generic Algorithms

- Sorting-Related Algorithms
  - sort, partial_sort, nth_element, binary_search, ...
- Generalized Numeric Algorithms
  - accumulate, partial_sum, adjacent_difference, ...

Containers

- Sequence containers
  - – vector, deque, list
- Sorted associative containers
  - – set, multiset, map and multimap

Container Adaptors

- Change interface of another component
  - – Stack Container Adaptor
  - – Queue Container Adaptor
  - – Dequeue Container Adaptor

Iterator Adaptors

- Change interface of an iterator
  - reverse_bidirectional_iterator
  - reverse_iterator (mutable)
  - const_iterator (mutable)
  - Insert Iterators
    - back_inserter
    - front_inserter
    - inserter

Function Adaptors

- Negators
- Binders (partial evaluation)
- Adaptors for Pointers to Functions

Summary

- STL does a good job at providing a library of reusable components.
- But generic programming can be much more generic than STL indicates.
- Parameterize with more complex structures than classes.