Generic Programming and the Standard Template Library

David R. Musser
Rensselaer Polytechnic Institute
Computer Science Department

Presentation Overview

- What is generic programming?
- The C++ Standard Template Library (STL)
- Principles of generic programming, with examples from STL
- Issues for the future

What is Generic Programming?

From the abstract of a talk given by Alexander Stepanov in the Schoffstall Distinquished Lecture Series, Rensselaer Polytechnic Institute, April, 1999:

"The purpose of this talk is to characterize Generic Programming as a scientific discipline. While Generic Programming is often compared with Object Oriented Programming or Functional Programming, we believe that such comparisons are not appropriate. While the goal of OOP or FP is to advocate a particular style of programming, the goal of GP is to provide a systematic classification of useful software components. For example, a computer scientist specializing in Object Oriented Programming might be interested in inheritance. A computer scientist specializing in Generic Programming might be interested in a comprehensive taxonomy of associative data structures. Generic Programming is fundamentally a study of algorithms, data structures and other software components with particular emphasis on their systematic organization. It is a mixture of theory and practice—it is a theoretical discipline in so far as it deals with classification of components based on their mathematical properties; it is a practical discipline in so far as it deals only with useful software artifacts. It is this tension between theory and practice that makes Generic Programming both a joy and a trial."

Software Productivity via Generic Programming

- Effective use of software libraries implies:
  - library components generally provide a significant portion of an application's solution
  - they admit to some degree of customization
  - time and space performance guarantees are provided
  - rigorous documentation is provided
- Nature and structuring of library components should also be in harmony with the designer's thought process
  - so that they provide useful and suggestive conceptual components for *designs*

***Standard* Template Library**

- The C++ Standard Template Library (STL) was adopted by the ANSI/ISO C++ Standards Committee in 1994
- Standardization of previously designed and implemented generic libraries in C++ (Stepanov, Lee, Musser)
- Standardization has many benefits, including the fact that it is a *useful constraint* on design and implementation choices

Standard *Template* Library

- A principal tool of generic programming is parameterization of container classes/packages and algorithms by type(s)
- The two basic approaches used are :
  - dynamic typing and inheritance (Smalltalk)
  - static typing and a facility for type parameters (*templates* in C++, *generics* in Ada)
- STL uses the second approach.

Standard Template *Library*

- STL is a general-purpose library of generic algorithms and data structures communicating through iterators
- Contains many small components that can be plugged together and used in an application
- Provides a framework into which different programming problems can be decomposed
- STL pillars: efficiency, orthogonality, and a solid theoretical foundation

C++ **Features**

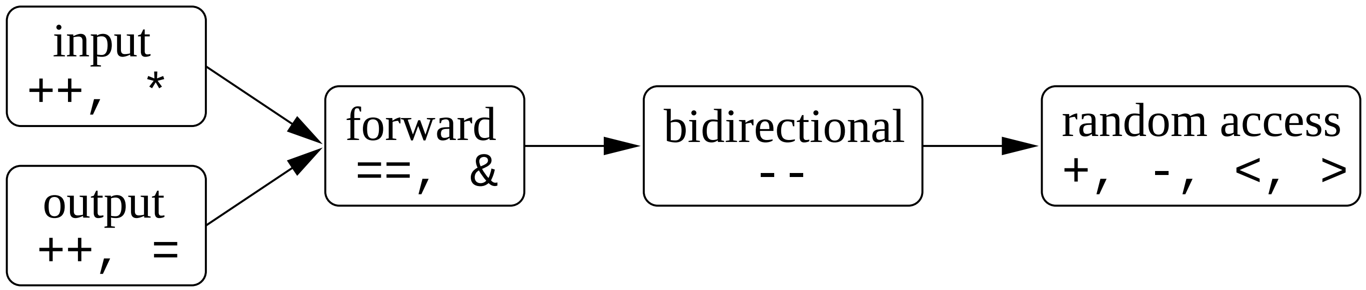
- (Almost) strongly typed: contractual model
- Supports modern software engineering methods:
  - Abstract Data Types (modular programming, information hiding)
  - Object-Oriented Programming (classes, inheritance, polymorphism, late binding)
  - Generic Programming

STL-Relevant C++ Features

- Generic Programming Support
  - Template classes and functions
  - Function inlining
- Systems Programming Support
  - Complete control of low-level details, when needed
- Object Oriented Programming Support
  - Automatically invoked constructors and destructors
  - Classes, inheritance
  - *Not used:* late binding with virtual functions

Overview of STL Components

- **Containers**
  - C++ array, `vector<T>`, `deque<T>`, `list<T>`
  - `set<Key>`, `multiset<Key>`, `map<Key,T>`, `multimap<Key,T>`
- **Algorithms**
  - *find*, *count*, *copy*, *remove*, *reverse*, *sort* …
- **Iterators**



- **Function Objects**
  - arithmetic, comparison, logical operations
- **Adaptors**
  - Function adaptors: binders, negators, adaptors for pointers to functions
  - Container adaptors: stack, queue, priority queue
  - Iterator adaptors: reverse, inserter
- **Allocators**
  - Encapsulate information about the memory model the program is using (pointers, references, sizes of objects, difference types, allocation/deallocation functions)