The Ada Standard Generic Library
(SGL)

Alexander V. Konstantinou

Computer Science Graduate Seminar
4/24/96
Presentation Overview

- Introduction (S{T|G}L)
- The C++ Standard Template Library (STL)
- Ada 95 Features
- The Ada Standard Generic Library (SGL)
- Conclusion
Standard Template Library

- The C++ Standard Template Library (STL) has been adopted by the ANSI/ISO C++ Standards Committee.
- There is nothing standard about SGL!
Standard Template (Generic) Library

- Generic programming refers to the style of programming in which container classes/packages and algorithms are parameterized by type.

- The two basic approaches used are:
  - dynamic typing and inheritance (Smalltalk)
  - static typing and a facility for arguments of type $T$ (C++/Ada)
Stadard Template Library

- STL is a general-purpose library of generic algorithms and data structures communicating through iterators
- Contains a lot of different 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
Overview of STL Components

- Containers
  - C++ array, vector<T>, deque<T>, list<T>
  - set<Key>, map<Key, T>

- Algorithms
  - find, count, copy, remove, reverse, sort ...

- Iterators
  - input: ++, *, +=
  - output: ++, =
  - forward: ==, &
  - bidirectional: --
  - random access: +, -, <, >
• **Function Objects**
  
  – arithmetic, comparison, logical operations

• **Adaptors**
  
  – Function: binders, negators, adaptors for pointers to functions
  
  – Container: stack, queue, priority queue

• **Allocators**
  
  – Enapsulate information about the memory model the program is using (pointers, references, sizes of objects, difference types, allocation/deallocation functions)
Sample STL Program

```cpp
#include <iostream.h>
#include <assert.h>
#include <algo.h>    // include STL algorithms
#include <vector.h> // include STL vector container

// Function Object
struct least_digit_less : public binary_function<int, int, bool> {
    bool operator()(const int &x, const int &y) {
        return ((x % 10) < (y % 10));
    }
};
```
```cpp
int main() {
    vector<int> V;       // implicit instantiation
    ostream_iterator<int> out(cout, " ");

    for (int i=0; i<25; i++) V.push_back(i);

    vector<int>::iterator x = find(V.begin(), V.end(), 10);

    reverse(V.begin(), x);
    copy(V.begin(), V.end(), out); cout << endl;

    sort(V.begin(), V.end(), least_digit_less());
    copy(V.begin(), V.end(), out); cout << endl;
}
```

```
9 8 7 6 5 4 3 2 1 0 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
20 0 10 21 1 11 22 2 12 23 3 13 24 14 4 15 5 16 6 17 7 18 8 19 9
```
Ada

- Named after Countess Augusta Ada Lovelace, who is considered the world’s first computer programmer (Charles Babbage’s Difference Engine)

- The Ada programming language was designed to:
  - improve correctness, safety, and reliability
  - reduce software development and maintenance costs
  - provide a syntax which is readable and easy to maintain.
Ada Features

- Fully specified and standardized (ANSI/ISO)
  - Compilers validated by U.S. Gov. & other agencies
- Supports modern Software Engineering methods:
  - Abstract Data Types (Modular programming)
  - Object-Oriented Programming (classes, inheritance, polymorphism, late binding)
  - Built-in tasking constructs/exceptions
  - Generic Programming
- Strongly typed (contractual model)
Evolution of Ada 95

- Revision initiated in January 1988 (Ada 9X)
- Requirements document broken into two groups:
  - Internationalization, Programming Paradigms (OO), Real-Time Requirements, Systems Programming, and General Requirements
  - Parallel/Distributed Processing, Safety-Critical Applications, Information Systems, and Scientific/Mathematical Applications
- Revised international standard (ISO/IEC 8652:1995)
SGL Relevant Ada (83 & 95) Features

- Generic Programming Support
  - Generic packages and subprograms (83)
  - Generic package parameters (95)

- Object Oriented Programming Support
  - Controlled types (95)

- Systems Programming Support
  - Storage Pool Management (95)
Some Ada Syntax

Generic Function Specification & Body

-- basic_algorithms.ads
generic
  type T is private;
procedure Swap(A, B: in out T);
pragma Inline(Swap);

-- basic_algorithms.adb
procedure Swap(A, B: in out T) is
  Tmp: T := A;
begin
  A := B;
  B := Tmp;
end Swap;
Generic Package Specification

generic
   type T1 is private;
   type T2 is private;
   with function "="(A, B: T1) return Boolean is <>;
   with function "="(A, B: T2) return Boolean is <>;

package Pairs is
   subtype Value_Type1 is T1;
   subtype Value_Type2 is T2;

   type Pair is record
      First : Value_Type1;
      Second : Value_Type2;
   end record;

   function "="(A, B: Pair) return Boolean;
   pragma Inline ("=");
end Pairs;
Designing the Ada Standard Generic Library

- **Containers**
  - Similar interface to STL; export iterators

- **Iterators Signatures**
  - Provide the iterator interface specification

- **Algorithms**
  - Parameterized by iterator signatures:

    ```ada
generic
    with package Iterators is new Forward_Iterators(<>);
    procedure Forward_Distance(First, Last: Iterators.Iterator;
                               N: in out Iterators.Distance_Type);
    ```
• **Function Objects**
  
  – Become simple functions in SGL

• **Adaptors**
  
  – *Should* be similar (unimplemented)

• **Allocators**
  
  – Are modeled almost directly from STL!
-- Bidirectional iterators signature package

generic
  type Value_Type is private;
  type Iterator is private;
  type Pointer is private;
  type Distance_Type is (<>);

  with procedure Inc( I: in out Iterator );
  with procedure Dec( I: in out Iterator );

  with function "="( i: Iterator; j: Iterator ) return Boolean;

  with function Val( I: in Iterator ) return Value_Type ;
  with procedure Assign( I: in Iterator; V: Value_Type );
  with function Ref( I: in Iterator ) return Pointer;

package Bidirectional_Iterators is end;
with Gnat.IO;
with Basic_Algorithms;
with Algorithms;
with Vectors;
with Put_Iterators;

package Integer_Vectors is new Vectors(Integer);

procedure Example is
  use Gnat.IO;
  use Integer_Vectors;

  -- Local Functions
  procedure Put_Space(I : in Integer) is
    begin
      Put(V);
      Put(" ");
    end Put_Space;
function Least_Digit_Less(A, B: in Integer) return boolean is begin
    return (A mod 10) < (B mod 10);
end Least_Digit_Less;

-- Instantiate Packages
package Put_Space_Iterators is new Put_Iterators(Integer, Put_Space);
use Put_Space_Iterators;

-- Instantiate algorithms
function Find is new Algorithms.Find
    (Integer_Vectors.Input_Iterator, "=");

procedure Sort_LD is new Algorithms.Sort
    (Integer_Vectors.Random_Access_Iterator, Least_Digit_Less);

procedure Copy is new Basic_Algorithms.Copy
    (Integer_Vectors.Input_Iterator,
     Put_Space_Iterators.Output_Iterator);
V : Integer_Vectors.Vector;
X : Integer_Vectors.Iterators.Iterator;
OS : Put_Space_Iterators.Iterator;
begin
  for I in 0..24 loop
    Push_Back(V, I);
  end loop;

  X := Find(Start(V), Finish(V), 10);

  reverse(Start(V), X);
  OS := Copy(Start(V), Finish(V), OS);

  Sort_LD(Start(V), Finish(V));
  OS := Copy(Start(V), Finish(V), OS);
end Example;
Wish List

- Language features:
  - C++ friend construct
  - Overloading of generic functions
  - Some mechanism for delaying type checking until instantiation time (i.e. IterSwap)

- Compiler features (GNAT):
  - Better error reporting in generic package/subprogram instantiation errors
Summary (C++ vs. Ada)

- The Ada syntax for parameter passing and function adaptors is simpler and more intuitive
- Iterator signatures provide a cleaner interface
- Explicit instantiation has certain advantages, provided the library design helps out
- The strong typing split personality syndrome:
  - Strong typing is good!
  - Strong typing is bad!
- SGL cannot quite achieve the same orthogonality as STL
• The Ada syntactic separation of functions and procedures presents problems and results in a cluttering of the interface

• gnat is a more stable compiler than g++