Summary

- Control Schemes
  - manual: explicit user-directed
  - semi-automatic: user/compiler cooperate
  - automatic: compiler-managed
- Granularity of instantiation
  - on a per-file basis
  - on a per-class basis
  - on a per-unit basis
- Support for Libraries and Large Projects
  - end-user must instantiate
  - library provider may instantiate
  - repositories of previous instantiations
  - specializations discoverable at link time
  - separable declarations and definitions

GNU g++
reno 2.0

(spring 94)

- A Fully Manual Scheme

  #pragma interface and #pragma implementation fully implemented and *-fexternal-templates* the only behavior.

  A new keyword *implement* allows for direct user intervention in the instantiation process.

  implement *NAME*;

  can appear anywhere in a top-level file that a declaration that can appear. *NAME* refers to a class template or (member) function template *using any arguments that are in currently in scope*.

  implement MapCH<CISRSignalName, CISRProcess::Input>;
  extern MapCH<CISRSignalName, CISRProcess::Input> name2input;

- Scalable to a Automatic Scheme

  The **implement** storage specifier allows separate tools to be built that will manage repositories through auto-generation and compilation of instantiation files.

GNU g++
reno 1.2

-fexternal-templates

- A Fully Manual Scheme

  All template class and template function definitions must be presented in order to be used; template class or template function declaration indicate a user-provided instantiation.

  Expand class and (all) class member functions of all class templates referenced; do the same for function templates according to #pragma interface and #pragma implementation. Use *external* linkage.

  g++ -c -fexternal-templates uses.cc

- Supporting Pragmas

  #pragma interface

  #pragma implementation "file.h"

- Instantiation occurs in an implementing file by mention. Use *typedef* for classes to trigger

  typedef *NAME* gensym;

GNU g++
reno-1.2

(default behavior)

- Fully Automatic Scheme

  All template class and template function definitions must be presented in order to be used; template class or template function declaration indicate a user-provided instantiation.

  Expand class and (all) class member functions of all class templates referenced; do the same for function templates. Use *internal* linkage.

  g++ -c file.cc

- Works well for small templates which have internal linkage (*e.g.* iostream templates)

  Unworkable for libraries and large projects

Symantec C++
THINK C (for the Mac)

- A Fully Manual Scheme

  Template declarations are required for compilation.

  At final-link time an object file must provide the definition of all templates used in the executable.

  Pragmas allow direct control over instantiation

- Supporting Pragmas

  #pragma template_access { public | private }

  #pragma template *class*
  #pragma template *ret function(signature)*

  where *class* is a class template name with parameters.
  where *ret function(signature)* is a function template name with parameters in the *signature*

Borland C++

- A Fully Automatic Scheme

  All template class and template function definitions must be presented at compile time. Instantiations are created for all template references in the current object file with external linkage.

  At final link time the link editor figures out how to filter out the multiple definitions so that there is one unique one in the final executable.

- Advantage: 100% ARM compliant

  No user intervention required; only one definition of each template in the final executable.

- Extremely expensive

IBM xlC
version 1

- Manual *and* automatic instantiation scheme

  1. Template function definitions are presented in all files that use them and multiple instances are created (redundant copies are ignored)
  2. Template function declarations are presented in all files that need them. In a separate file the template function definitions are presented along with a triggering mention
  3. Template function declarations are presented in special include files. The compiler automatically instantiates.

  xlC -c -qtempinc=*directory* file.C

- Supporting pragmas

  #pragma implementation (file.C)

  #pragma define (*INSTANCE*)

  INSTANCE is a class template name with parameters
  Function templates use a mentioning *extern*

- The tempinc directory is a sort of repository

DEC cxx version 1.2

- A completely manual instantiation scheme

  cxx -c -define_templates definitions.cxx
  cxx -c uses.cxx

- Supporting Pragmas

  #pragma define_templates ***NAME***

  where *NAME* is a class template name *or function template name* including parameters

- The #pragma it is processed when seen; instantiation occurs for all needed template definitions

- The #pragma is transitive so encapsulation is preserved

- Allows function templates for static template functions

AT&T CC 3.0.1

- A Completely Automatic Scheme

  Compile individual files against a set of repositories

  At final-link, convert unresolved references to templates into template instantiation requests. Instantiate new templates into the repository.

  Repeat until done.

- Advantage: fully ARM compliant.
  No explicit user interaction is required

- Allows for link-time discovery of user specializations

- Instantiation can occur at differing levels of granularity

- Disadvantage: forces the end user to manage the instantiation obligation

Template Terms

class template:
a template for a class

function template:
a template for a function

template class:
an instance of a class template

template function:
an instance of a function template

template declaration file:
a (header) file with the class or function template definition

template definition file:
a (source) file with the class or function template function bodies

argument declaration file:
a (header) file declaring a class which is used as a template argument

ARM Requirements for Template Support

Stoustrup, pg 598

There must be exactly one definition for each template of a given name in a program. There can be many declarations. The definition is used to generate specific template classes and template functions to match the uses of the template.

Using a template-class-name constitutes a declaration of a template class.

Calling a function template or taking its address constitutes a declaration of a template function. There is no special syntax for calling or taking the address of a template function; the name of a template function is used exactly as is a function name. Declaring a function with the same name as a function template with a matching type constitutes a declaration of a specific template function.

An Informal Survey of C++ Template Implementations

Wendell C. Baker
CAD Group
Electrical Engineering and
Computer Sciences
U.C. Berkeley