Invited talk at *Dansk Selskab for Datalogi*
Copenhagen, 13 June 2002

**Title:**
Software tools for program library development

**Speaker:**
Jyrki Katajainen

These slides are available at
http://www.cphstl.dk.

This bunch also contains slides that I did not have time to show.
Structure of this talk

1. What is the STL?
2. What is the CPH STL?
3. What tools do we use?
4. What tools have we developed?
5. What tools do we need?
Background

Kurt Mehlhorn about LEDA:

“Initially, I thought that the development of the LEDA library will take one year, but the project took 10 years.”

[A discussion in Göteborg, 2001]
The Standard Template Library (STL) is part of the ISO standard for C++ ratified in 1998.

Its main architect was Alexander A. Stepanov. The implementation written by him, Meng Lee, and David R. Musser was made freely available on the Internet in 1994.

- algorithms
- functors
- iterators
- adaptors
- sequences
- allocators
Source: David R. Musser, Gillmer J. Derge, and Atul Saini, *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, 2nd Edition, Addison-Wesley (2001), Figure 2.1
# Iterators

An iterator whose value type is $T$

$p, q$ objects of type $X$

t object of type $T$

<table>
<thead>
<tr>
<th>Category</th>
<th>Allowed expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>$X(p)$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$X p(q)$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$X p = q$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$p = q$ (assignment)</td>
</tr>
<tr>
<td></td>
<td>$p == q$ (equality)</td>
</tr>
<tr>
<td></td>
<td>$p != q$ (inequality)</td>
</tr>
<tr>
<td></td>
<td>*$p$ (read only once)</td>
</tr>
<tr>
<td></td>
<td>$p-&gt;m$ (equivalent to (*$p$).$m$)</td>
</tr>
<tr>
<td></td>
<td>++$p$ (preincrement)</td>
</tr>
<tr>
<td></td>
<td>(void) $p++$ (postincrement)</td>
</tr>
<tr>
<td>output</td>
<td>$X(p)$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$X p(q)$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$X p = q$ (copy constructor)</td>
</tr>
<tr>
<td></td>
<td>$p = q$ (assignment)</td>
</tr>
<tr>
<td></td>
<td>*$p = t$ (write only once)</td>
</tr>
<tr>
<td></td>
<td>++$p$ (preincrement)</td>
</tr>
<tr>
<td></td>
<td>$p++$ (postincrement)</td>
</tr>
</tbody>
</table>
Object of $X$'s difference type

<table>
<thead>
<tr>
<th>Category</th>
<th>Allowed expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>all earlier operations</td>
</tr>
<tr>
<td></td>
<td>$X \ p$ (default constructor)</td>
</tr>
<tr>
<td></td>
<td>$X()$ (default constructor)</td>
</tr>
<tr>
<td></td>
<td>multiple reads and writes</td>
</tr>
<tr>
<td>bidirectional</td>
<td>all earlier operations</td>
</tr>
<tr>
<td></td>
<td>$--r$ (predecrement)</td>
</tr>
<tr>
<td></td>
<td>$r--$ (postdecrement)</td>
</tr>
<tr>
<td>random access</td>
<td>all earlier operations</td>
</tr>
<tr>
<td></td>
<td>$p += i$ (iterator addition)</td>
</tr>
<tr>
<td></td>
<td>$p + i$ (iterator addition)</td>
</tr>
<tr>
<td></td>
<td>$i + p$ (iterator addition)</td>
</tr>
<tr>
<td></td>
<td>$p \ -= i$ (iterator subtraction)</td>
</tr>
<tr>
<td></td>
<td>$p \ - i$ (iterator subtraction)</td>
</tr>
<tr>
<td></td>
<td>$q \ - p$ (difference)</td>
</tr>
<tr>
<td></td>
<td>$p[i]$ (equivalent to $*(p + i)$)</td>
</tr>
<tr>
<td></td>
<td>$p &lt; q$ (less)</td>
</tr>
<tr>
<td></td>
<td>$p &gt; q$ (greater)</td>
</tr>
<tr>
<td></td>
<td>$p &lt;= q$ (less or equal)</td>
</tr>
<tr>
<td></td>
<td>$p &gt;= q$ (greater or equal)</td>
</tr>
</tbody>
</table>
Sequences

- list
- vector
- deque
Sorted sequences

- set
- multiset
- map \( \langle \text{key}, \text{value} \rangle \)
- multimap
A **functor** is a function pointer, or an object of any class that supports the operation `operator()`.

For example, the `std::sort` function can take a functor, which defines an ordering on the set of elements, as its third parameter.

```cpp
template <
    typename random_access_iterator
>
void sort (  
    random_access_iterator,  
    random_access_iterator
);

template <
    typename random_access_iterator,  
    typename ordering
>
void sort (  
    random_access_iterator,  
    random_access_iterator,  
    ordering
);
```

---

© Performance Engineering Laboratory

10
Adaptors

Iterator adaptors

- E.g., reverse iterators

Container adaptors

- queue
- priority queue
- stack

Function adaptors

- E.g., create a unary function from a binary function by fixing one of the parameters
Allocators

Make dynamic sequences independent of the memory management.

\texttt{X} an allocator whose value type is \texttt{T}
\texttt{a} object of type \texttt{X}
\texttt{t} object of type \texttt{T}
\texttt{n} value of type \texttt{X::size\_type}
\texttt{p} object of type \texttt{X::pointer}

\texttt{a.allocate(n)}
\begin{itemize}
  \item Allocates \(n \times \text{sizeof}(T)\) bytes of memory
\end{itemize}

\texttt{a.deallocate(p,n)}
\begin{itemize}
  \item Dealslocate the memory that \texttt{p} points to
\end{itemize}

\texttt{a.construct(p,t)}
\begin{itemize}
  \item Equivalent to \texttt{new ((void*) p) T(t)}
\end{itemize}

\texttt{a.destroy(p)}
\begin{itemize}
  \item Equivalent to \((T*) p)\rightarrow\text{\~T()}
\end{itemize}
STL header files

<algorithm> 103 functions; most are trivial, but there are some though ones, e.g., sort, inplace_merge, etc.

<deque> A doubly resizable array

<functional> Functor utilities

<iterator> Iterator utilities

<list> A doubly linked list

<map> Sorted sequences with satellite data

<memory> Memory-management utilities

<numerics> 4 numeric functions

<queue> An interface to a queue and a priority queue

<set> Sorted sequences without satellite data

<stack> An interface to a stack

<utility> pair and rel_ops

<vector> A singly resizable array
#include <list>
#include <deque>
#include <algorithm>
#include <cassert>

template <typename sequence>
sequence make (const char s[]) {
    return sequence(&s[0], &s[std::strlen(s)]);
}

int main () {
    char* vowels = "aeiouy";
    int len = std::strlen(vowels);

    std::list<char> consonants =
        make<list<char> >("bcdfghjklmnpqrstvwxz");

    std::deque<char> alphabet(26, ' ');

    std::merge(
        &vowels[0], &vowels[len],
        consonants.begin(), consonants.end(),
        alphabet.begin()
    );

    assert(alphabet ==
        make<deque<char> >("abcdefghijklmnopqrstuvwxyz");
    return 0;
}

shell> g++ merge.cpp
shell> a.out
Stepanov’s contributions

“the task of the library designer is to find all interesting algorithms, find the minimal requirements that allow these algorithms to work, and organize them around these requirements”

[Stepanov 2001]

– Algorithm algebra
– Generic programming
– Programming with concepts
– Semi-formal specification of the components, including complexity requirements
– Generality so that every program works on a variety of types, including C++ built-in types
– Efficiency close to hand-coded, type-specific programs
Goals of the CPH STL project

The purpose of the project is

- to study and analyse existing specifications for and implementations of the STL to determine the best approaches to optimization,

- to provide an enhanced edition of the STL and make it freely available on the Internet,

- to provide cross-platform benchmark results to give library users better grounds for assessing the quality of different STL components,

- to develop software tools that can be used in the development of component libraries, and

- to carry out experimental algorithmic research.
Development history

The CPH STL: weekly team meetings
Autumn 2000; credit points for 12 students; of those 7 wrote written projects (5 projects in all)

Performance engineering
Spring 2001; credit points for 13 students; 4 finished their development exercise

The CPH STL: weekly team meetings
Spring 2001; credit points for 9 students; one B.Sc. project, one written project

My favourite software development tools
Autumn 2001; credit points for 16 students; 2 finished their development exercise; three written projects
Where are the challenges?

**Challenge 1:** C++ itself

**Challenge 2:** correctness

**Challenge 3:** efficiency

**Challenge 4:** extensions

**Challenge 5:** tools
Challenge 1: C++ itself

```cpp
template <
    typename element
>
const element&
min (>
    const element&,
    const element&
);

vs.

#define min(a, b) ((a) < (b) ? (a) : (b))
```

Develop `min` that satisfies the following requirements:

1. Offers function call semantics (including type checking), not macro semantics.

2. Supports both `const` and non-`const` arguments (including mixing the two in a single call).

3. Supports arguments of different types where that makes sense.
Alexandrescu’s solution

template <class L, class R>
typename MinMaxTraits<L, R>::Result
min(L& lhs, R& rhs) {
    if (lhs < rhs)
        return lhs;
    return rhs;
}

template <class L, class R>
typename MinMaxTraits<const L, R>::Result
min(const L& lhs, R& rhs) {
    if (lhs < rhs)
        return lhs;
    return rhs;
}

... two more overloads ...

It would all be so nice, but there is a little detail worth mentioning. Sadly, \texttt{min} does not work with any compiler the author had access to. In fairness, each compiler chokes on a different piece of code. For more details, see Andrei Alexandrescu, \textit{Generic Programming}: Min and Max Redivivus, \textit{C++ Experts Forum}, April 2001. Available at \url{www.cuj.com/experts}. 
Conformance to the standard

Figures missing; sorry

Challenge 2: correctness

- memory leakage
- exception safety
- thread safety
- iterator validity
- constant correctness
- concept checking
Challenge 3: efficiency

Example:

```cpp
template <
    typename input_iterator,
    typename output_iterator
>
output_iterator
    copy (
        input_iterator,
        input_iterator,
        output_iterator
    );
```

This is trivial, ikke os'.
copy.cpp File Reference

This file defines the function copy. More...

```cpp
#include <iterator>
#include <type>
#include <algorithm>
```

Go to the source code of this file.

Namespaces

```cpp
namespace cppnl;
```

Detailed Description

This file defines the function copy.

Original author

Jyrki Katajainen <jyrki@diku.dk>, December 2001

Sources

Andrei Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley (2001), see Section 2.11.3.


---

Generated on Mon Dec 17 00:53:19 2001 for The C++ Standard Template Library by Doug Lea, 12.3 written by Doug Lea on Herresh, © 1999-2000
```cpp
#include "cassert"  /* defines std::isalnum(true) */
#include <type_traits>  /* defines std::string and std::string_view */
#include <algorithm>  /* defines std::copy */

namespace std { namespace

  template <typename input_iterator, typename output_iterator>
  inline output_iterator copy (input_iterator first, input_iterator last, output_iterator result) {
    static const int conservative = 0;
    static const int fast = 1;
    for (; first != last; ++first) {  /* is *first */
      *result = *first;
      ++result;
    }
    return result;
  }

  template <typename input_iterator, typename output_iterator>
  inline output_iterator copy (input_iterator first, input_iterator last, output_iterator result, std::string_view conservative, fast);  

}  // namespace
```
Challenge 4: extensions

- `<hash_set>`, `<hash_multiset>`, `<hash_map>`, `<hash_multimap>`
- `<slist>`
- `min_and_max_element` in `<algorithm>`
- Most algorithms should support forward iterators if efficiency requirements are not violated
- `tuple` in `<utility>`
- C++ without built-in types: `natural<1>`, `natural<8>`, `natural<16>`, etc.
- Similar integer type instead of `short`, `int`, `long`
- Infinite precision arithmetic `integer<∞>`
- `real` as a class
- `array` as a class

But not much more!
Challenge 5: tools

Next I will discuss about the tools

– used by us,

– developed by us, and

– needed by us.
Course on software tools in 2001

17.9 Version management with CVS
   Delta algorithms

24.9 Shell programming: Bourne Again shell
      Python: PE-lab’s talk announcement system

1.10 Regular expressions in grep, sed, awk, Perl, and JavaScript; Regex engines

8.10 Enterprise application integration: XML
      Database programming: MySQL

22.10 Web programming: C, Perl, PHP, Python

29.10 Autoconf, automake, and libtool
      Make utility

5.11 Macro processing: m4, C, \TeX, \LaTeX

12.11 XEmacs and Elisp
      Stack programming: PostScript

19.11 UML
Feedback from the students

“Jyrki, you are trying to teach us far too many tools in such a short time.”

[Anonymous student 2001]
Reflections on three tools

- CVS
- Make
- Doxygen
CVS tutorial

Checkout:

ask> setenv CVS_RSH ssh
ask> setenv CVSROOT :ext:jyrki@cphstl.dk:/usr/local/CPHSTL/
ask> cvs checkout cphstl

Commit after some changes:

ask> cvs -q update
ask> cvs commit -m "A mandatory note; let it be meaningful"

Creating a new directory:

ask> mkdir newdir
ask> cvs add newdir

Removing a directory:

ask> cd newdir
ask> rm *
ask> cvs remove
ask> cvs commit -m "removed all files"
ask> cd ..
ask> cvs update -P
CVS reflections

- There are some startup problems since at this point the manual is not good.

- It takes some time before one starts to trust to the system.

- Now we move the files inside the repository in order not to loose the development history.

- Now and then we still get some mysterious problems due to access privileges (when adding new directories into the repository).
# Original author: Jyrki Katajainen <jyrki@diku.dk>,
# February - June 2001
# Spell-checking was inspired by Steffen Nissen
# <lukesky@math-tech.dk>.

# Here are the ways how you could use this description
# file.
...
# Spell-ckeck your text.
# gmake spell file=<your LaTeX-file>
# or
# gmake spell
...
# public:

language=english #dansk also possible
...
spell:
ifdef file
  ispell -d $(language) -p ./$(file:.tex=.dict) \ 
  -t $(file)
else
  ifeq ($(words $(latex-files)), 1)
    ispell -d $(language) -p ./$(dictionary-files) \ 
    -t $(latex-files)
  else
    @echo "Usage: gmake spell file=<your LaTeX-file>"
  endif
endif

...
Make reflections

- make and gmake are two separate tools.
- There are problems with absolute paths.
- One makefile per directory is not always good. Our original makefile for generating reports has had several errors, and the updates have required a lot of work.
- I have started to use Python instead of gmake.

```shell
shell> cat run
#!/usr/bin/env python

# Usage: run program.cpp

CC = "g++ "
options = " -Wall -W -pedantic -ansi "

import os, sys

program = sys.argv[1]
base = program[:-4]
object = base + ".o"

os.system(CC + options + " -I. " + " -c " + program)
os.system(CC + options + " -L. " + object + " -ltutnew")
o.s.system("./a.out")
```
Doxygen example

/*! \file
 * \brief This file defines the function
 * \code copy\endcode.
 *
 * \h3
 * Original author
 * \endcode
 *
 * Jyrki Katajainen <jyrki@diku.dk>, December 2001
 ...

namespace cphstl {

namespace {

    enum copy_algorithm_selector {conservative, fast};

    /*! \brief Function \code copy\endcode for input
 * iterators.
 */

    template <
        typename input_iterator,
        typename output_iterator
    >
    output_iterator
    copy ( ...
Doxygen reflections

– Seems that I am almost the only user of this tool. Students have not had time to learn it.

– Doxygen does not include a full C++ parser.

– It is an open-source product still containing many errors.

– It would be nice if XML could be used instead of HTML.
Tools developed by us

- Maz’ cache profiler
- Jyrki’s benchmarking framework (under development)
Maz’ cache profiler

We use Bjarke’s implementation, which is 1000 times faster than the original one, but it is still slow for production use.

shell> newprof -h
Usage: newprof [options]
Options are:
  -mN: set cache capacity to N words (default 1024)
  -aN: set cache associativity to N (default 1)
  -bN: set block size to N words (default 8)
  -pS: set replacement policy [rnd,nru,lru] (default rnd)
  -wN: set word size in bits (default 32)
  -fS: read from file S (default is stdin)
  -l: produce LaTeX output
  -h: this page
  -n: just report the miss count
  -v: verbose output (all memory references)

Cache associativity is 0 to N where 0 and N are fully associative, 1 is direct-mapped, and 2 to N-1 are set associative.
Example program

```c
#define CPROF "profile.dat"
#include <newprof.h>

int main() {
    int size = 100;
    int A[size];

    // Read the same element again and again ...
    for (int i = 0; i < size; ++i) {
        READ(A[50]);
    }

    // Sequential access
    for (int i = 0; i < size; ++i) {
        WRITE(A[i], 0);
    }

    // Arbitrary access
    for (int i = 0; i < size; ++i) {
        int next = (int) (double(rand()) / RAND_MAX * size);
        READ(A[next]);
    }

    // Write the profile to a file.
    writetofile();
}
```
Profiler output
Jyrki’s benchmarking framework

**Vision:** To carry out program comparisons

- one should only fill a form,
- send it to a benchmarking system, and
- then one will receive the results in a file or via e-mail.
Earlier workflow

- Write the functions to be compared.
- Write a driver in C++ which performs one single benchmark case.
- Write a shell script to create a benchmark suite.
- Write a gnuplot control file to get a nice plot of the results.
- Write a makefile to compile the files, carry out all experiments, and create the plot.
- Run the same experiment in different computers.

For me it took two days or more to get the results for a single function.

For a student it normally took longer, since there were at least one tool that he/she did not know earlier.
Gnuplot example: `find on Pentium`
Gnuplot example: \texttt{find on HP9000}
Gnuplot example: find on Sun
Design decisions

- Web interface
- XML
- gmake
- shell scripts

I will do it in **Python**. Then a form writer has the full power of a programming language at his/her disposal.

I will rely on Python’s unittest module (earlier called PyUnit), cf., test case object \( \approx \) benchmark case object, test suite object \( \approx \) benchmark suite object, and test runner object \( \approx \) benchmark runner object.
Possible extensions

– Integration with PAPI to get the number of cache misses,

– the number of branch miss-predictions, and

– the number of instructions.

– Integration with Tutnew to get information about the memory usage.

– Integration with gprof to get how many times a specific function is called.
PAPI

An API that gives a uniform access to the hardware counters of modern computers. At the moment the following computers are supported:

- Pentium Pro, II, III, P6
- AMD Athlon
- IBM Power 3, 604, 604e
- Sun UltraSparc
- MIPS R10K, R12K
- Cray T3E, SV1, SV2
- (On the way: Alpha EV6, EV67, IA-64, Microsoft Windows)

PAPI is aware of 104 different counters, but not all are supported by all architectures.
For example, for Pentium III running under Linux the following counters related to L1 cache are provided:

- **PAPI\_L1\_DCM**: L1 data cache misses
- **PAPI\_L1\_ICM**: L1 instruction cache misses
- **PAPI\_L1\_TCM**: L1 cache misses
- **PAPI\_L1\_LDM**: L1 load misses
- **PAPI\_L1\_STM**: L1 store misses
- **PAPI\_L1\_DCH**: L1 data cache hits
- **PAPI\_L1\_DCA**: L1 data cache accesses
- **PAPI\_L1\_ICH**: L1 instruction cache hits
- **PAPI\_L1\_ICA**: L1 instruction cache accesses
- **PAPI\_L1\_ICR**: L1 instruction cache reads
- **PAPI\_L1\_TCA**: L1 total cache accesses

```c
void main() {
    int events[2]={PAPI\_L1\_DCM, PAPI\_L1\_ICM};
    long\_long results[2] = {0,0};
    int status;

    status = PAPI\_start\_counters(&events, 2);
    do\_something();
    status = PAPI\_stop\_counters(&results, 2);
}
```
Tools we need

– Automating program transformations: manual loop unrolling is tedious.

– Memory leakage detector: Tutnew looks fine, but most documentation is only in Finnish. The tool is available at http://www.cs.tut.fi/%7Ebitti/tutnew/.

– Unittesting framework: one student has used CppUnit.

– \LaTeX{} style to create interactive PDF-documents easily.

– Prettyprinter: I cannot get my students to use cweb; interactive prettyprinting.
Tutnew example

```cpp
shell> cat leak.cpp
#include <iostream>
#include <tutnew.h>

void print_content(int* p) {
    std::cout << *p << std::endl;
}

int main() {
    int* p = new int;
    *p = 3;
    print_content(p);
    std::cout << "The end" << std::endl;
    return 0;
}

shell> run leak.cpp
3
The end
Tutnew: the following normal memory blocks have not been deleted:
Tutnew: 4 byte(s) allocated with new on line 9 in file leak.cpp
```
“While STL is widely used, my hopes for the creation of many libraries of generic components have not been fulfilled. As far as I can determine the reason that such libraries are not created is that there are no financial mechanisms for supporting the work.”

[Stepanov 2001]
Future of C++

Will C++ be the language of the elite programmers in 20 years? I doubt that. Already now good programmers are seeking for languages, the usage of which makes them more productive.

Can C++ be improved to meet the future challenges? I think so, but I do not believe in committee work.
Library writer’s wish list

- A language should have only one official compiler.

- The kernel of a programming language should be smaller than that of C++, e.g., built-in types should be part of the standard library.

- The library writer should have access to the facilities provided by the compiler, i.e., there should be a bridge between a library and the compiler. (E.g., give a warning when the user is using a CPH STL specific extension and warnings are on.)