This might be relevant for you

Jens Tofteskov, *I gang med kandidatafhandlingen — og hva' så?*, Web dokument (1995–2001).
`http://www.edu.cbs.dk/cm/kapjece_dk.shtml`

Performance Engineering Laboratory

List of members

Christopher Curry <cdcurry@diku.dk>
Jyrki Katajainen <jyrki@diku.dk>
Jeppe Nejsum Madsen <jnm@arraytechnology.com>
Hans Henrik Stærfeldt <hhs@cbs.dtu.dk>
Lars Yde <lars_yde@worldonline.dk>

Former members and guests

Jesper Bojesen <jbb@medical-insight.com> (M.Sc. 2000)
Anders Sewerin Johansen <sewerin@get2net.dk> (M.Sc. 1999)
Tomi Pasanen (Post Doc., total 2 months in 2001)
Sofus S. Mortensen <sofus@lambdasoft.dk> (M.Sc. thesis 2001)
Morten Nicolaj Pedersen <Morten.Pedersen@capgemini.dk> (M.Sc. 1999)
Maz Spork <halgrim@diku.dk> (M.Sc. 1999)

Academic partners

Gerth Stølting Brodal, Ph.D., Assoc. Prof., Århus
Martin Dietzfelbinger, Ph.D., Professor, Ilmenau
Torben Hagerup, Ph.D., Professor, Frankfurt am Main
Jukka Teuhola, Ph.D., Senior Lecturer, Turku
Tomi Pasanen, Ph.D. Lecturer, Turku
Peter Sanders, Ph.D., Saarbrücken

Industrial partners

Jesper Bojesen, M.Sc., Medical Insight, Copenhagen
Sofus S. Mortensen, B.Sc., lambdasoft, Copenhagen
Jesper Larsson Träff, Ph.D., NEC, St. Augustin

This page was last modified by Jyrki on 14.11.2001.

Copenhagen STL

The Standard Template Library, or STL for short, is a library of generic algorithms and data structures that has been incorporated in the C++ language standard and now ships with all modern C++ compilers.

In existing STL implementations many STL components boast good performance, some even outdo most of their hand-crafted competition. In many cases, however, there is still room for improvement on both an algorithmic and an implementational level as suggested, for example, by research conducted here at DIKU (Performance Engineering Laboratory).

The purpose of this project is:

- to study and analyse existing specifications for and implementations of STL to determine the best approaches to optimization,
- to design alternative/enhanced versions of individual STL components using standard algorithmic and performance engineering techniques, and
- to implement and document the new versions in C++.

Those will be the main strands of activity, but the project will also be an exercise in software development using up-to-date methodology and tools.

--

Last modified December 10 2001 01:19:10 PM

Possible topics for B.Sc. projects

- Memory-leak detection
- Exception safety
- Concept checking

For other proposals, see the CPH STL home-page `http://www.cphstl.dk` under menu item "New Developers".

Problem: memory leaks

```
template <typename T>
void f() {
  T* p(new T);

  /* ...more code... */

  delete p;
}
```

if `f()` never executes the `delete` statement, either because of an early return or because of an exception thrown during execution of the function body, then the allocated object is not deleted and we have a classic memory leak.

Problem: unsafe code

```
// Exception-safe?
//
String f() {
  String result;
  result = "some value";
  std::cout << "some output";
  return result;
}
```

This function has two side effects: it emits some output, and it returns a String. The goal we want to achieve is the **strong exception-safety guarantee**, which boils down to ensuring that the function acts atomically — even if there are exceptions, either all side effects happen or none of them do.

There is one minor quibble, as illustrated by the following client code:

```
String theName;
theName = f();
```

The String copy constructor is invoked because the result is returned by value, and the copy assignment operator is invoked to copy the result into `theName`. If either copy fails, then `f()` has completed all of its work and all of its side effects (good), but the result has been irretrievably lost (oops).

Problem: poor error messages

```
#include <list>
#include <algorithm>

int main(int, char*[]) {
  std::list<int> v;
  std::stable_sort(v.begin(), v.end());
  return 0;
}
```

```
grimer> g++ concept-check.cpp
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
In function `void __inplace_stable_sort<_List_iterator<int,int
*> >(_List_iterator<int,int &,int *>, _List_iterator<int,int &
*>)':
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
instantiated from here
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
no match for `_List_iterator<int,int &,int *> & -
_List_iterator<int,int &,int *> &'
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
In function `void __insertion_sort<_List_iterator<int,int &,in
>(_List_iterator<int,int &,int *>, _List_iterator<int,int &,in
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
instantiated from `__inplace_stable_sort<_List_iterator<int,in
*> >(_List_iterator<int,int &,int *>, _List_iterator<int,int &
*>)'
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
instantiated from here
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
no match for `_List_iterator<int,int &,int *> & + int'
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
In function `void __linear_insert<_List_iterator<int,int &,int
int>(_List_iterator<int,int &,int *>, _List_iterator<int,int &
*>, int *)':
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
instantiated from `__insertion_sort<_List_iterator<int,int &,i
>(_List_iterator<int,int &,int *>, _List_iterator<int,int &,in
/usr/local/stow/lib/gcc-lib/hppa2.0-hp-hpux10.20/2.95.2/../../
```

How to get started?

Send me e-mail <jyrki@diku.dk> or visit my office N334.