Templight: A Clang Extension for Debugging and Profiling C++ Template Metaprograms

Zoltán Porkoláb, Zoltán Borók-Nagy

Eötvös Loránd University, Budapest
Ericsson Hungary
Agenda

- C++ Template Metaprogramming
- Possible debugging and profiling techniques
- Templight back-end tool
- Front-end tools
- 3rd party applications – please, contribute!
- Vision
C++ Template Metaprograms

- Expression templates (since 1995!)
- Active libraries, compile-time adaptation
- Static interface checking
- Simulating language extensions
- DSL embedding
- Many other areas ...
Motivation – a personal view

template <class T, class S>
? max( T a, S b) // How to define the return type?
{
    if ( a > b )
        return a;
    else
        return b;
}

int main()
{
    short is = 3; long il = 2; double d = 3.14;
    cout << max( il, is);
    cout << max( is, d);
}
Compile-time vs. Run-time

Compile-time

Run-time
Compile-time vs. Run-time

Compile-time

Run-time

3 < 3.14
Compile-time vs. Run-time

- $1.0 < 0$
- $3 > 2L$
- $3 < 3.14$

Compile-time

Run-time
Compile-time vs. Run-time

- int
- long
- std::string
- double

- $3 > 2L$
- $3 < 3.14$
- $-1.0 < 0$
- $3 > 2L$
- $3 < 3.14$
Motivation

template <class T, class S>
? max( T a, S b) // How to define the return type?
{
    if ( a > b )
        return a;
    else
        return b;
}

int main()
{
    short is = 3; long il = 2; double d = 3.14;
    cout << max( il, is); // long is "better" than short
    cout << max( is, d);  // double is "better" than short
}
Compile-time vs. Run-time

Template design time

<table>
<thead>
<tr>
<th>int</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::string</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 &gt; 2L</td>
</tr>
<tr>
<td>3 &lt; 3.14</td>
</tr>
</tbody>
</table>

Template Instantiation

Run-time
Compile-time vs. Run-time

T

S

int

long

std::string

double

Template Instantiation

3 > 2L

3 < 3.14

-1.0 < 0

Template design time

Compile-time

Run-time
Compile-time vs. Run-time

- $\text{sizeof}(T) < \text{sizeof}(S)$

- $3 > 2L$

- $3 < 3.14$

- $\text{int}$

- $\text{long}$

- $\text{std::string}$

- $\text{double}$

- Template design time

- Compile-time

- Template Instantiation

- Run-time
Compile-time vs. Run-time

T
sizeof(T) < sizeof(S)

S

int

std::string

double

long

-1.0 < 0

3 > 2L

3 < 3.14

Template design time

Compile-time Template Instantiation

Run-time
Motivation

template <class T, class S>
? max( T a, S b)  // How to define the return type?
{
    if ( a > b )
        return a;
    else
        return b;
}

int main()
{
    short is = 3; long il = 2; double d = 3.14;
    cout << max( il, is);  // long is ''better'' than short
    cout << max( is, d);   // double is ''better'' than short
}
(de)Motivation

template <class T, class S>
auto max( T a, S b) -> decltype(a+b) // C++11
{
    if ( a > b )
        return a;
    else
        return b;
}

int main()
{
    short is = 3; long il = 2; double d = 3.14;
    cout << max( il, is); // -> long
    cout << max( is, d); // -> double
}
(de)Motivation

template <class T, class S>

typename std::common_type<T,S>::type max( T a, S b) // C++11
{
    if ( a > b )
        return a;
    else
        return b;
}

int main()
{
    short is = 3; long il = 2; double d = 3.14;
    cout << max( il, is); // -> long
    cout << max( is, d); // -> double
}
The usual factorial program ...

template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };
};
template <>
struct Factorial<0>
{
    enum { value = 1 };
};
int main()
{
    const int fact5 = Factorial<5>::value;
}
Bugs!!! ...
template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };
};
template <>
struct Factorial<0>
{
    enum { value = 1 };
} //;
int main()
{
    const int fact5 = Factorial<5>::value;
}
The java programmer ...

template <int N>  
struct Factorial  
{
    enum { value = Factorial<
        N - 1>
    ::value *
        N
};
}  

template <>  
struct Factorial<0>  
{
    enum { value = 1
};
}  

int main()
{
    const int fact5 = Factorial<5>::value;
}
The vim user ...

template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };
};
template <>
struct Factorial<0>
{
    enum { ivalue = 1 };
};
int main()
{
    const int fact5 = Factorial<5>::value;
}
The vim user ...

template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };
};
template <>
struct Factorial<0>
{
    enum { ivalue = 1 };
};
int main()
{
    const int fact5 = Factorial<5>::value;
}

$ clang++ fact.cpp
fact.cpp:5:34: error: no member named 'value' in 'Factorial<0>'
    enum { value = Factorial<N-1>::value * N };
    ~~~~~~~~~~~~~~~~~^  
fact.cpp:5:18: note: in instantiation of template class 'Factorial<1>'
    requested here
    enum { value = Factorial<N-1>::value * N };
    ^  
fact.cpp:5:18: note: in instantiation of template class 'Factorial<2>'
    requested here
    enum { value = Factorial<N-1>::value * N };
    ^  
fact.cpp:5:18: note: in instantiation of template class 'Factorial<3>'
    requested here
    enum { value = Factorial<N-1>::value * N };
    ^  
fact.cpp:5:18: note: in instantiation of template class 'Factorial<4>'
    requested here
    enum { value = Factorial<N-1>::value * N };
    ^  
fact.cpp:16:21: note: in instantiation of template class 'Factorial<5>'
    requested here
    const int fact5 = Factorial<5>::value;
    ^
1 error generated.
The negative approach ...

template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };  
};
template <>
struct Factorial<0>
{
    enum { value = 1 };  
};
int main()
{
    const int fact5 = Factorial<-5>::value;
}
The negative approach ...

template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N };
};
template <>
struct Factorial<0>
{
    enum { value = 1 };
};

int main()
{
    const int fact5 = Factorial<-5>::value;
}

$ clang++ fact4.cpp
fact4.cpp:6:18: fatal error: recursive template instantiation exceeded maximum
    depth of 512
    enum { value = Factorial<N-1>::value * N };
    ^
fact4.cpp:6:18: note: in instantiation of template class 'Factorial<-517>'
    requested here
    enum { value = Factorial<N-1>::value * N };
Fact4.cpp:6:18: note: (skipping 503 contexts in backtrace; use
    -ftemplate-backtrace-limit=0 to see all)
fact4.cpp:18:21: note: in instantiation of template class 'Factorial<-5>'
    requested here
    const int fact5 = Factorial<-5>::value;
    ^
fact4.cpp:6:18: note: use -ftemplate-depth=N to increase recursive
    template
    instantiation depth
    enum { value = Factorial<N-1>::value * N };
    ^
1 error generated.
The greedy ...

```cpp
template <int N>
struct Factorial
{
    enum { value = Factorial<N-1>::value * N }
};
template <>
struct Factorial<0>
{
    enum { value = 1 }
};
int main()
{
    const int fact5 = Factorial<-5>::value;
}
```

```
$ clang++ -ftemplate-depth=10000 fact4.cpp
```
The greedy ...

template <int N> 
struct Factorial 
{ 
  enum { value = Factorial<N-1>::value * N }; 
}; 

template <> 
struct Factorial<0> 
{ 
  enum { value = 1 }; 
}; 

int main() 
{ 
  const int fact5 = 
} 

$ clang++ -ftemplate-depth=10000 fact4.cpp 
clang: error: unable to execute command: Segmentation fault 
clang: error: clang frontend command failed due to signal (use -v to see invocation) 
clang version 3.2 (branches/release_32 180710) 
Target: x86_64-unknown-linux-gnu 
Thread model: posix 
clang: note: diagnostic msg: PLEASE submit a bug report to 
http://llvm.org/bugs/ and include the crash backtrace, preprocessed 
source, and associated run script. 
clang: note: diagnostic msg: ****************** 

PLEASE ATTACH THE FOLLOWING FILES TO THE BUG REPORT: 
Preprocessed source(s) and associated run script(s) are located at: 
clang: note: diagnostic msg: /tmp/fact4-iy6zKp.cpp 
clang: note: diagnostic msg: /tmp/fact4-iy6zKp.sh 
clang: note: diagnostic msg: 

******************
We need tools

- C++ syntax is not designed for metaprogramming
- Compilers are not optimized for detecting and reporting template metaprogram errors
- Compilers are not optimized for template metaprogram execution
- Compiler internals are black box for most programmers
- Programmers have less experience with template metaprograms
Tool support

- Pretty good support for run-time C++
Tool support

• Pretty good support for run-time C++
  • Static analyzers, lint-like tools
  • Debuggers
  • Profilers
  • Code comprehension tools
  • Style checkers
Tool support

- Pretty good support for run-time C++
  - Static analyzers, lint-like tools
  - Debuggers
  - Profilers
  - Code comprehension tools
  - Style checkers
- Tools for template metaprogramming
Tool support

- Pretty good support for run-time C++
  - Static analyzers, lint-like tools
  - Debuggers
  - Profilers
  - Code comprehension tools
  - Style checkers
- Tools for template metaprogramming
  - ?
## Tool support

<table>
<thead>
<tr>
<th>Run-time</th>
<th>Compile-time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tool support

Run-time

Compile-time
Tool support

Run-time

Compile-time
Tool support

Run-time

Compile-time
Tool support

Run-time

Compile-time
Related work

- Debugging
  - Static assert/Concept check (Siek-Lumsdaine, McNamara-Smaragdakis, Alexandrescu, others...)
  - Warning generation (many attempt)
  - Instrumentation

- Profiling
  - Measuring full compilation (Gurtovoy-Abramhs)
  - Measuring warning appearance (Watanabe)

- Visualize
  - Source execution
  - Instantiation graph
GPCE 2006: Porkoláb, Mihalicza, Sipos: Debugging C++ template metaprograms

template<int i>
struct Factorial
{
    /* ------------------ begin inserted ------------ */
    struct _TEMPLIGHT_0s { int a; };
    enum { _TEMPLIGHT_0 = Templight::ReportTemplateBegin<_TEMPLIGHT_0s, &_TEMPLIGHT_0s::a>::Value };
    /* ------------------ end inserted ------------ */
    enum { value = Factorial<i-1>::value };
    /* ------------------ begin inserted ------------ */
    struct _TEMPLIGHT_1s { int a; };
    enum { _TEMPLIGHT_1 = Templight::ReportTemplateEnd<_TEMPLIGHT_1s, &_TEMPLIGHT_1s::a>::Value };
    /* ------------------ end inserted ------------ */
};

template<>
struct Factorial<1>
{
    /* ------------------ begin inserted ------------ */
    struct _TEMPLIGHT_2s { int a; };
    enum { _TEMPLIGHT_2 = Templight::ReportTemplateBegin<_TEMPLIGHT_2s, &_TEMPLIGHT_2s::a>::Value };
    /* ------------------ end inserted ------------ */
    enum { value = 1 };
    /* ------------------ begin inserted ------------ */
    struct _TEMPLIGHT_3s { int a; };
    enum { _TEMPLIGHT_3 = Templight::ReportTemplateEnd<_TEMPLIGHT_3s, &_TEMPLIGHT_3s::a>::Value };
    /* ------------------ end inserted ------------ */
};
Instrumentation

• Advantages
  • Light-way approach (compared to compiler hack)
  • Grammar support (we used wave)
  • Easier to port: just change the warning generator
Instrumentation

• Advantages
  • Light-way approach (compared to compiler hack)
  • Grammar support (we used wave)
  • Easier to port: just change the warning generator

• Disadvantages
  • Complex constructs are hard (e.g. inheritance)
  • Serious distortion in profiling information
  • Memoization is not detected
Templight 2.0

• Based on LLVM/Clang compiler infrastructure

• Patch to
  • Detect/measure instantiation
  • Detect memoization
  • Put timestamp on events
  • Measure memory consumption (optional)

• Emit trace in various formats (txt, YAML, XML)

• Front-end tools
  • Visual debugger
  • Profiler data viewer
Templight 2.0

Templight patch

C++

Clang source

Trace

Interactive debugger/visualizer

Instantiation time and memory profiler

3rd party tools
Installation

- Visit [http://plc.inf.elte.hu/templight](http://plc.inf.elte.hu/templight)
- Download `templight-<timestamp>.tar.gz`
  - Contains clang patch and the two frontends
- Download Clang source
- Patch and build clang
- Build front-end tools (optional)
  - `>=Qt 4.6` and `>=Graphviz 2.28.0` required
  - `$ qmake; make`
How to use

template<int N>
struct Fib
{
    static const int value = Fib<N-2>::value + Fib<N-1>::value;
};
template<> struct Fib<0>
{
    static const int value = 0;
};
template<> struct Fib<1>
{
    static const int value = 1;
};
int main()
{
    static const int fib5 = Fib<5>::value;
}
How to use

$ clang++ -templight fib.cpp

$ ls
fib.cpp.trace.xml

$ wc fib.cpp.trace.xml
  123  275 3838 fib.cpp.trace.xml

$ head fib.cpp.trace.xml
<?xml version="1.0" standalone="yes"?>
<Trace>
  <TemplateBegin>
    <Kind>TemplateInstantiation</Kind>
    <Context context = "Fib&lt;5&gt;"/>
    <PointOfInstantiation>fib.cpp|22|14</PointOfInstantiation>
    <TimeStamp time = "421998401.188854"/>
    <MemoryUsage bytes = "0"/>
  </TemplateBegin>
</TemplateBegin>
template <int N>
struct Fib
{
  static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
  static const int value = 0;
};

template<>
struct Fib<1>
{
  static const int value = 1;
};
{  
  static const int value = 0;
}

template<> struct Fib<1>
{
  static const int value = 1;
};

int main()
{
  int fib5 = Fib<5>::value;
}
template <int N>
struct Fib
{
  static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
  static const int value = 0;
};

template<>
struct Fib<1>
```cpp
template <int N>
struct Fib
{
    static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
    static const int value = 0;
};

template<>
struct Fib<1>
{

}
```

Event type: Begin
Kind: Memoization
Name: Fib<1>
File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28
```cpp
template <int N>
struct Fib
{
  static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
  static const int value = 0;
};

template<>
struct Fib<1>
{
};
```

Event type: End
Kind: Memoization
Name: Fib<1>
File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28
```cpp
template <int N>
struct Fib
{
  static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
  static const int value = 0;
};

template<>
struct Fib<1>
{
};
```

Event type: Begin
Kind: TemplateInstantiation
Name: Fib<2>
File position: `/home/ezolpor/work/proj/templight/work/fib.cpp|5|46`
template <int N>
struct Fib
{
  static const int value = Fib<N-2>::value + 
  Fib<N-1>::value;
};

template<>
struct Fib<0>
{
  static const int value = 0;
};

template<>
struct Fib<1>
```cpp
template <int N>
struct Fib
{
    static const int value = Fib<N-2>::value + Fib<N-1>::value;
};

template<>
struct Fib<0>
{
    static const int value = 0;
};

template<>
struct Fib<1>
{

};
```

Event type: End

Kind: TemplateInstantiation

Name: Fib<4>

File position: `/home/ezolpor/work/proj/templight/work/fib.cpp` | 5 | 46
```cpp
10 {
11   static const int value = 0;
12 }
13
14 template<>
15 struct Fib<1>
16 {
17   static const int value = 1;
18 }
19
20 int main()
21 {
22   int fib5 = Fib<5>::value;
23 }
24
25
Event type: End
Kind: TemplateInstantiation
Name: Fib<5>
File position: /home/ezolpor/work/proj/templight/work/fib.cpp|22|14
Major features

• Debugging
  • Breakpoints: Step in/out/over, forward or backward
  • Filtering out unwanted events
  • Safe mode – flush output after each events

• Profiling
  • Cumulative instantiation times
  • Memory usage at each events
  • Distortion < 3%
    – Heap allocated, not growing, default size is 500.000
    – Flush at the end of compilation
Forks, Applications

- Martin Schulze modified client tools
  http://github.com/schulmar/Templar

- Malte Skarupke's blog: comparing instantiation time of unique_ptr, boost::flat_map, etc.
  http://probablydance.com/2014/04/05/reinventing-the-wheel-for-better-compile-time/
Metashell – interactive TMP REPL

- Ábel Sinkovics and András Kucsma
- Metashell https://github.com/sabel83/metashell
- Online demo: http://abel.web.elte.hu/shell
Mikael Persson's Templight fork

- https://github.com/mikael-s-persson/templight
- https://github.com/mikael-s-persson/templight-tools

- Refactored and greatly improved Templight
- Patch is under review
- Tools: KCachegrind format
Our vision

[Diagram showing Trace as a central node with arrows pointing to Debugger/visualizer, Other tools, 3rd party tools, GCC, Intel C++ Compiler, and Windows.]
Summary

- Tool support for C++ metaprogramming
- Debugger/profiler requires compiler support
- Templight 2.0 based on clang
- Mikael's patch for clang is under review
- Please use it, give us feedback
- Compiler vendors, will you support Templight?
Q/A

Templight: A Clang Extension for Debugging and Profiling C++ Template Metaprograms

http://plc.inf.elte.hu/templight

gsd@elte.hu