Template metaprograms

- First template metaprogram: Erwin Unruh 1994
  - Printed prime numbers as compiler error messages
- Later proved to be a Turing-complete sub language of C++
- Today many use cases exist
  - Expression templates
  - DSL embedding (boost::xpressive)
  - Generators (boost::spirit)
  - Compile-time adaptation (std::enable_if)
- Many more...

Factorial

- Compile time recursion
- Specialization to stop recursion

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

// example use
int main()
{
    const int fact5 = Factorial<5>::value;
    std::cout << fact5 << endl;
    return 0;
}
```

Factorial

- Compile time recursion
- Specialization to stop recursion

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

// example use
int main()
{
    const int fact5 = Factorial<5>::value;
    std::cout << fact5 << endl;
    return 0;
}
```

Meta-control structures

```
template <bool condition, class Then, class Else>
struct IF
{
    typedef Then RET;
};

template <class Then, class Else>
struct IF<false, Then, Else>
{
    typedef Else RET;
};

// example use
template <typename T, typename S>
IF< sizeof(T)<sizeof(S), S, T>::RET max(T t, S s)
{
    if ( t < s)
        return s;
    else
        return t;
}
```

Template metafunction as parameter

```
template <int n, template<int I> class F>
struct Accumulate
{
    enum { RET = Accumulate<n-1,F>::RET + F<n>::RET };
};

template <template<int> class F>
struct Accumulate<0,F>
{
    enum { RET = F<0>::RET };
};

template <int n>
struct Square
{
    enum { RET = n*n };
};

cout << Accumulate<3,Square>::RET << endl;
```

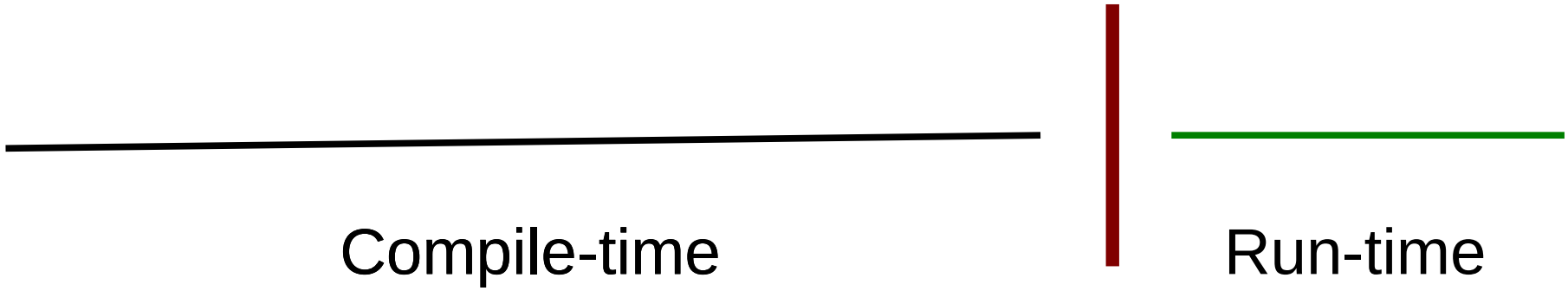
Why use it?

```
template <unsigned int N>
struct binary
{
    static const unsigned int value = binary<N/10>::value*2 + N%10;
};

template <>
struct binary<0>
{
    Static const unsigned int value = 0;
};

int main()
{
    const unsigned int di = 12;
    const unsigned int oi = 014;
    const unsigned int hi = 0xc;

    const unsigned int bi0 = binary_value("1101");  // run-time
    const unsigned int bi1 = binary<1100>::value;   // compile-time
}
```

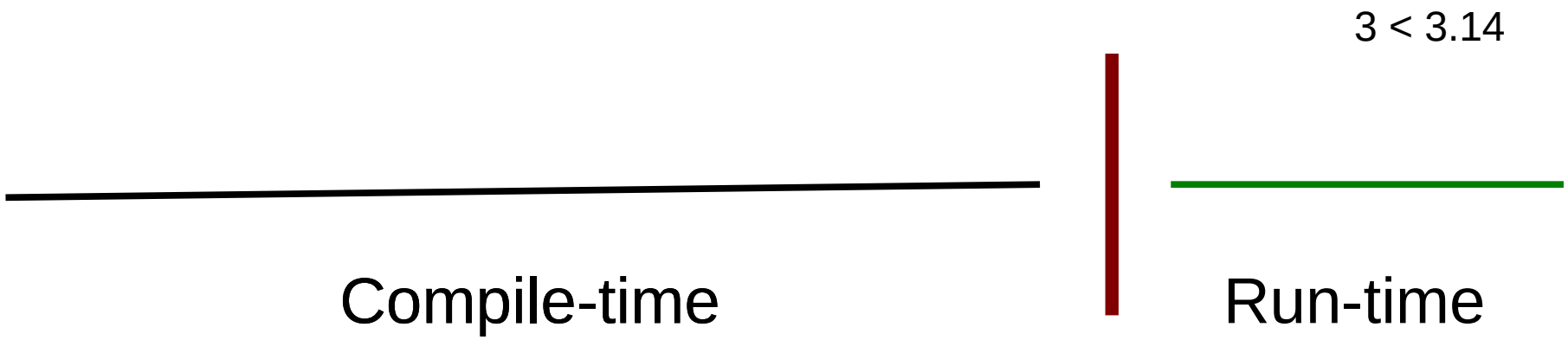
Motivation

```
template <class T, class S>
? max( T a, S b)  // How to define the return value?
{
    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 vs. Run-time



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

-1.0 < 0

3 > 2L

3 < 3.14

Compile-time

Run-time

Motivation

```
template <class T, class S>
? max( T a, S b)  // How to define the return value?
{
    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

int

long

std::string

double

-1.0 < 0

3 > 2L

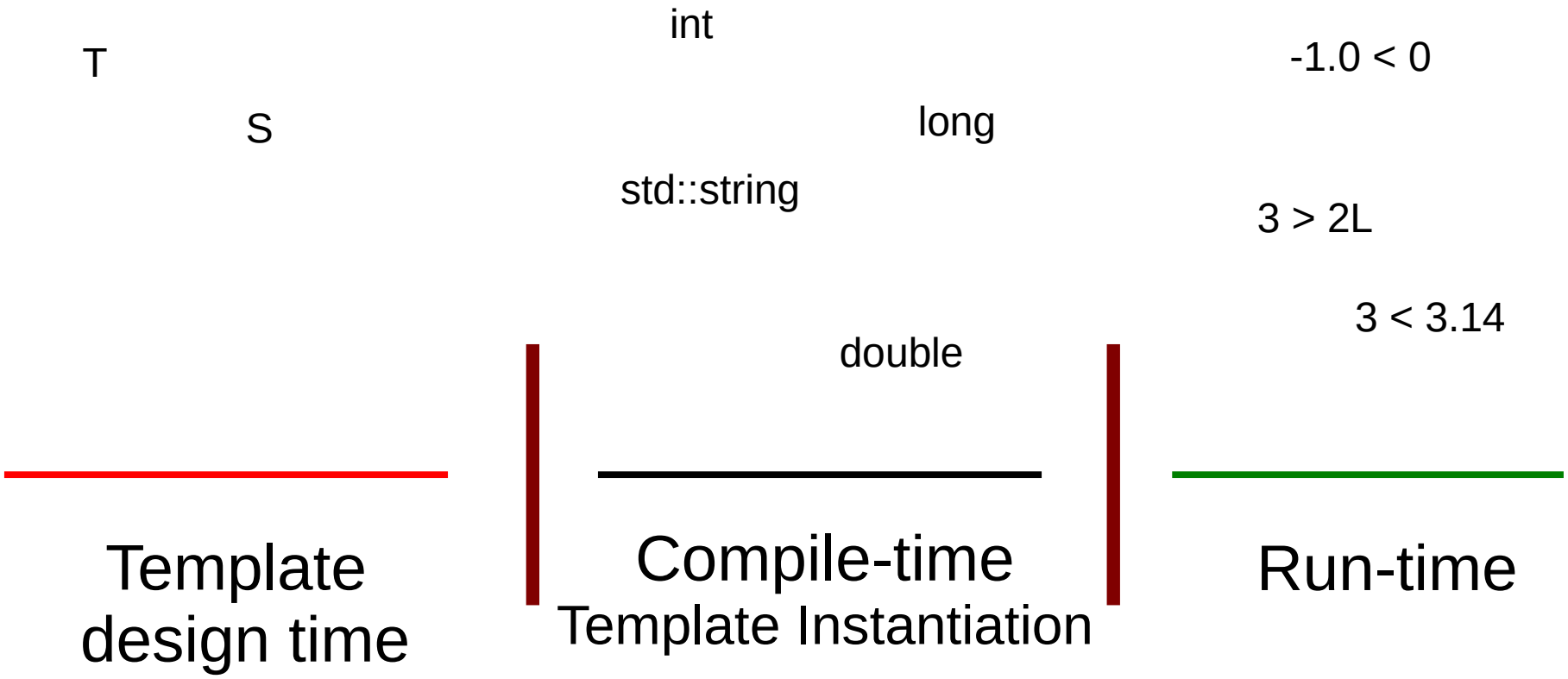
3 < 3.14

Template design time

Compile-time
Template Instantiation

Run-time

Compile-time vs. Run-time



Compile-time vs. Run-time

T

S

sizeof(T) < sizeof(S)

int

long

std::string

double

-1.0 < 0

3 > 2L

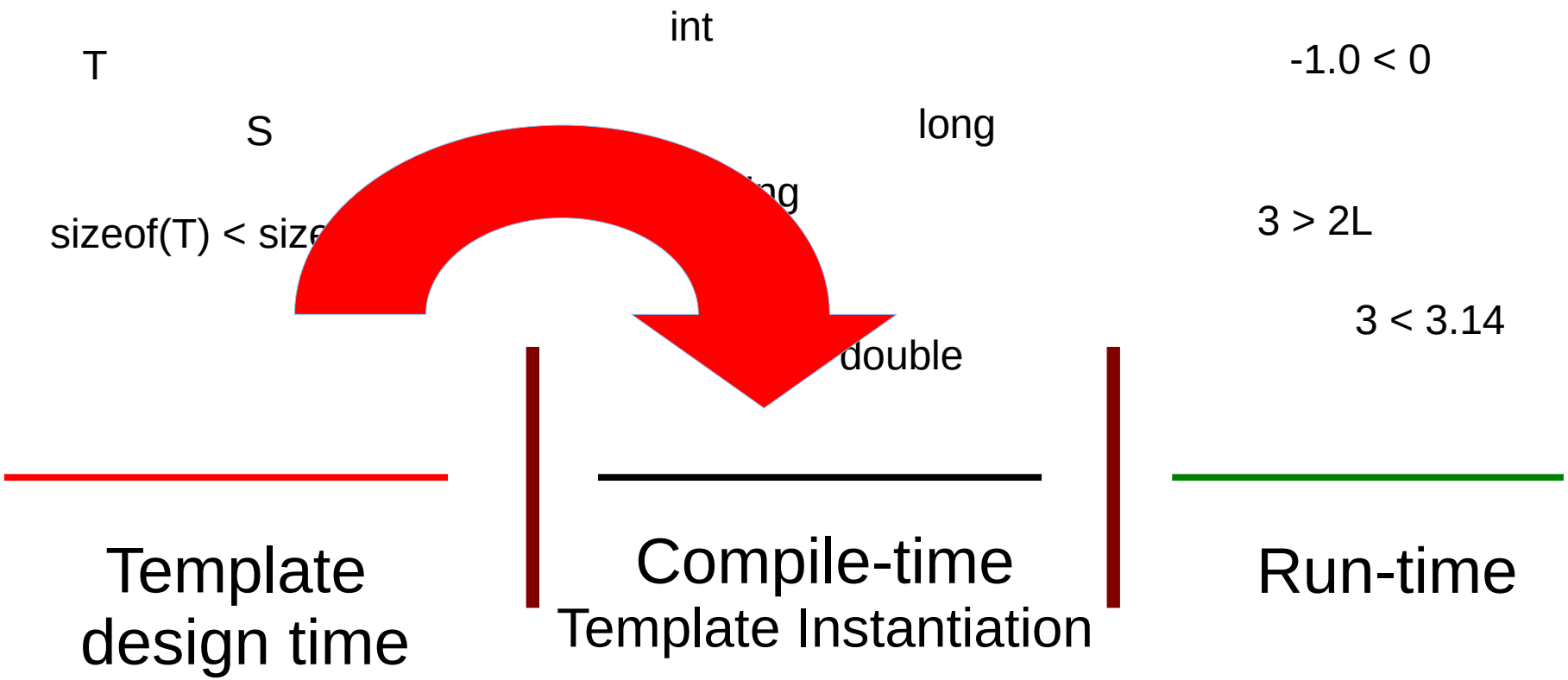
3 < 3.14

Template design time

Compile-time
Template Instantiation

Run-time

Compile-time vs. Run-time



Motivation

```
template <class T, class S>
? max( T a, S b)  // How to define the return value?
{
    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
}
```

Motivation

```
template <class T, class S>
IF< sizeof(T)<sizeof(S), S, T>::RET max( T a, S b)
{
    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>
auto max( T a, S b) -> decltype(1==1?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>::value 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
}
```

(de)Motivation

```
template <class T, class S>
auto max( T a, S b)                    // C++14
{
    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
}
```

Use case example

```
template <class T>
class matrix
{
public:
    matrix( int i, int j );
    matrix( const matrix &other);
    ~matrix();
    matrix operator=( const matrix &other);
private:
    int  x;
    int  y;
    T   *v;
    void copy( const matrix &other);
    void check( int i, int j) const throw(indexError);
};
```

Specialization for POD* types

*POD is depricated since C++20

```
template <class T>
void matrix<T>::copy( const matrix &other)
{
    x = other.x;
    y = other.y;
    v = new T[x*y];
    for ( int i = 0; i < x*y; ++i )
       v[i] = other.v[i];
}
// specialization for POD types
template <>
void matrix<long>::copy( const matrix &other)
{
    x = other.x;
    y = other.y;
    v = new long[x*y];
    memcpy( v, other.v, sizeof(long)*x*y);
}
template <>
void matrix<int>::copy( const matrix &other) ...
```

Trait

```
template <typename T> struct copy_trait
{
    static void copy( T* to, const T* from, int n)  {
        for( int i = 0; i < n; ++i ) to[i] = from[i];
    }
};
template <> struct copy_trait<long>
{
    static void copy( long* to, const long* from, int n) {
        memcpy( to, from, n*sizeof(long));
    }
};
template <class T, class Cpy = copy_trait<T> >
class matrix {  ... }

template <class T, class Cpy>
void matrix<T,Cpy>::copy( const matrix &other) {
    x = other.x;
    y = other.y;
    v = new T[x*y];
    Cpy::copy( v, other.v, x*y);
}
```

char_trait

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;

template<
    class CharT
> class char_traits;

struct char_traits<char>     {  }; // specializations for
struct char_traits<wchar_t>  {  }; // char_type, int_type, pos_type,
struct char_traits<char16_t> {  }; // assign, eq, lt, eof,
struct char_traits<char32_t> {  }; // compare, length, find, ...


struct ci_char_traits : public std::char_traits<char> {
  static char to_upper(char ch) { return std::toupper((unsigned char) ch); }
  static bool eq(char c1, char c2)  { return to_upper(c1) == to_upper(c2); }
  static bool lt(char c1, char c2)  { return to_upper(c1) < to_upper(c2);  }
  static int compare(const char* s1, const char* s2, std::size_t n) {  ... }
  static const char* find(const char* s, std::size_t n, char a)     {  ... }
};
```

char_trait

```
struct ci_char_traits : public std::char_traits<char> {
  static char to_upper(char ch) { return std::toupper((unsigned char) ch); }
  static bool eq(char c1, char c2)  { return to_upper(c1) == to_upper(c2); }
  static bool lt(char c1, char c2)  { return to_upper(c1) < to_upper(c2);  }
  static int compare(const char* s1, const char* s2, std::size_t n) {  ... }
  static const char* find(const char* s, std::size_t n, char a)     {  ... }
};

template<class DstTrait, class CharT, class SrcTrait>
constexpr std::basic_string_view<CharT, DstTrait>
traits_cast(const std::basic_string_view<CharT, SrcTrait> src) noexcept {
    return {src.data(), src.size()};
}

int main()
{
    using namespace std::literals;

    constexpr auto s1 = "Hello"sv;
    constexpr auto s2 = "heLLo"sv;

    if (traits_cast<ci_char_traits>(s1) == traits_cast<ci_char_traits>(s2))
        std::cout << s1 << " and " << s2 << " are equal\n";
}
```

Policy

```
template <typename T, bool B> struct copy_trait
{
    static void copy( T* to, const T* from, int n) {
        for( int i = 0; i < n; ++i )
            to[i] = from[i];
    }
};
template <typename T> struct copy_trait<T, true>
{
    static void copy( T* to, const T* from, int n)    {
        memcpy( to, from, n*sizeof(T));
    }
};

template <typename T> struct is_pod {  enum { value = false }; };

template <> struct is_pod<long>     {  enum { value = true }; };

template <class T, class Cpy = copy_trait<T,is_pod<T>::value> >
class matrix  { ... }
```

Typelist

```
class NullType {};

// We now can construct a null-terminated list of typenames:
typedef Typelist< char,
           Typelist<signed char,
             Typelist<unsigned char, NullType>
           >
        > Charlist;

// For the easy maintenance, precompiler macros are defined
// to create Typelists:

#define TYPELIST_1(x)           Typelist< x, NullType>
#define TYPELIST_2(x, y)        Typelist< x, TYPELIST_1(y)>
#define TYPELIST_3(x, y, z)     Typelist< x, TYPELIST_2(y,z)>
#define TYPELIST_4(x, y, z, w)  Typelist< x, TYPELIST_3(y,z,w)>

// usage example
typedef TYPELIST_3(char, signed char, unsigned char)  Charlist;
```

Typelist operations

```
//  Length
template <class TList> struct Length;

template <>
struct Length<NullType>
{
    enum { value = 0 };
};

template <class T, class U>
struct Length <Typelist<T,U> >
{
    enum { value = 1 + Length<U>::value };
};

static const int len = Length<Charlist>::value;
```

Typelist operations

```
// IndexOf
template <class TList, class T> struct IndexOf;

template <class T>
struct IndexOf< NullType, T>
{
    enum { value = -1 };
};
template <class T>
struct IndexOf< Typelist<T, Tail>, T>
{
    enum { value = 0 };
};
template <class T, class Tail>
struct IndexOf< Typelist<Head, Tail>, T>
{
private:
    enum { temp = IndexOf<Tail, T>::value };
public:
    enum { value = (temp == -1 ? -1 : 1+temp) };
};

static const int IndexOf<Charlist, int>::value;            // -1
static const int IndexOf<Charlist, char>::value;           //  0
static const int IndexOf<Charlist, unsigned char>::value;  //  2
```

Matrix revisited

```
typedef TYPELIST_4(char, signed char, unsigned char, int)  Pod_types;

template <typename T> struct is_pod
{
    enum { value = IndexOf<Pod_types,T>::value != -1 };
};
template <typename T, bool B> struct copy_trait
{
    static void copy( T* to, const T* from, int n)   {
        for( int i = 0; i < n; ++i )
            to[i] = from[i];
    }
};
template <typename T> struct copy_trait<T, true>
{
    static void copy( T* to, const T* from, int n)   {
        memcpy( to, from, n*sizeof(T));
    }
};
template <class T, class Cpy = copy_trait<T,is_pod<T>::value> >
class matrix { ... }
```

SFINAE

```
template <class T, class U>  // is convertible from T to U
struct Conversion
{
    typedef char Small;
    class Big { char dummy[2]; };
    static Small Test(U);
    static Big   Test(...);
    static T     MakeT();
    static const bool exists = sizeof(Test(MakeT()))==sizeof(Small);
    static const bool sameType = false;
};
template <class T>
struct Conversion<T,T>
{
  static const bool exists = 1;
  static const bool sameType = 1;
};
int main()
{
  std::cout << std::boolalpha
     << Conversion<double, int>::exists << '\n'
     << Conversion<char, char*>::exists << '\n'
     << Conversion<std::vector<int>,std::list<int>>::exists << '\n';
}
```

SFINAE

```
template <class T, class U>  // is convertible from T to U
struct Conversion
{
    using Small = char;
    Using Big   = struct { char dummy[2]; };
    static Small Test(U);
    static Big   Test(...);
    static const bool exists =  /* declval is in <utility> header */
                    sizeof(Test(std::declval<T>()))==sizeof(Small);
    static const bool sameType = false;
};
template <class T>
struct Conversion<T,T>
{
  static const bool exists = 1;
  static const bool sameType = 1;
};
int main()
{
  std::cout << std::boolalpha
     << Conversion<double, int>::exists << '\n'
     << Conversion<char, char*>::exists << '\n'
     << Conversion<std::vector<int>,std::list<int>>::exists << '\n';
}
```

type_traits

```
template <class T, class U>  // is convertible from T to U
struct Conversion
{
    using Small = char;
    Using Big   = struct { char dummy[2]; };
    static Small Test(U);
    static Big   Test(...);
    static const bool exists =  /* declval is in <utility> header */
                    sizeof(Test(std::declval<T>()))==sizeof(Small);
    static const bool sameType = false;
};
#include <type_traits>
// is_convertible
// is_assignable
// is_pod
// ...
int main()
{
  std::cout << std::boolalpha
     << std::is_convertible<double, int>::value << '\n'
     << std::is_convertible<char, char*>::value << '\n'
     << std::is_convertible<std::vector<int>,std::list<int>>::value
     << '\n';
}
```

- **Run-time**
- **Compile-time**

- **Run-time**
- **Compile-time**

- Functions
- Values, literals
- Data structures
- If/else
- Loop
- Assignment
- May depend on input

- **Run-time**

- Functions
- Values, literals
- Data structures
- If/else
- Loop
- Assignment
- May depend on input

- **Compile-time**

- Metafunctions (type)
- Const, enum,constexpr
- Typelist (type)
- Pattern matching
- Recursion
- Ref. Transparency
- Deterministic

- **Run-time**
- **Compile-time**

- Imperative
- Object-oriented
- Generative
- (some) Functional

- **Run-time**

- Imperative
- Object-oriented
- Generative
- (some) Functional

- **Compile-time**

- Pure Functional

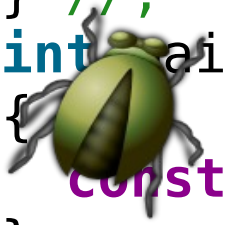
The usual factorial program ...

```
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!!! ...

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 java programmer ...

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

```
$ clang++ fact.cpp
fact.cpp:14:2: error: expected ';' after class
}
 ^
 ;
1 error generated.
```

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 = Fact
};
template <>
struct Factorial<0>
{
  enum { ivalue = 1 }
};
int main()
{
  const int fact5 = F
}
```

```
$ 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 = Fact
};
template <>
struct Factorial<0>
{
  enum { value = 1 };
};
int main()
{
  const int fact5 = F
}
```

```
$ 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 ...

```
template <int N>
struct Factorial
{
  enum { value = Fact
};
template <>
struct Factorial<0>
{
  enum { value = 1 };
};
int main()
{
  const int fact5 = F
}
```

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

The greedy ...

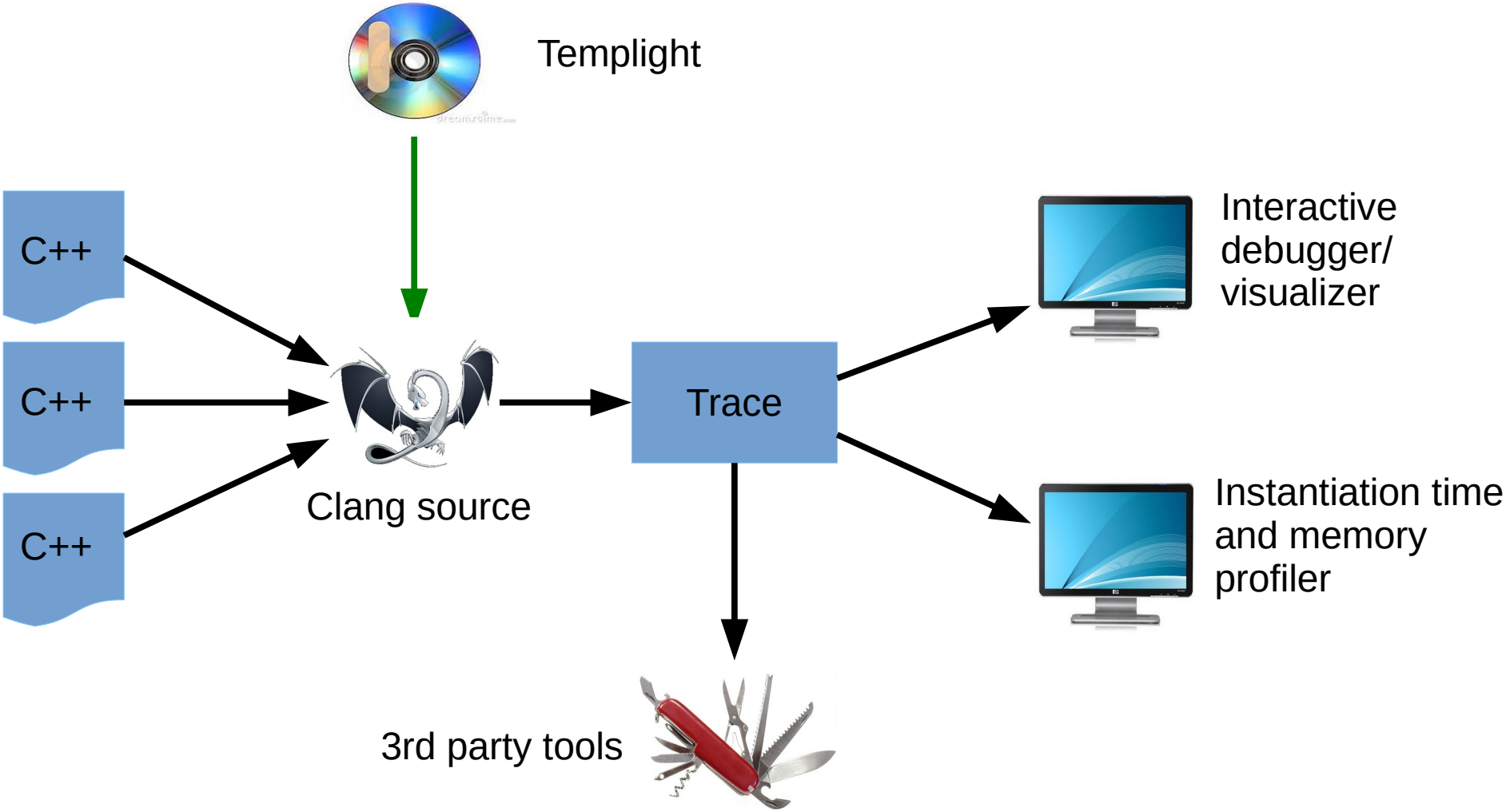
```
template <int N>
struct Factorial
{
  enum { value = Fact
};
template <>
struct Factorial<0>
{
  enum { value = 1 };
};
int main()
{
  const int fact5 = F
}
```

```
$ 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:

********************
```

Templight

- Based on LLVM/Clang compiler infrastructure
- Instrument source code
  - Inject templates for begin-end of every instantiation
  - Generate warnings by specific protocol
  - Collect and present warnings
- Modify the compiler
  - First as a patch for Clang++
  - Since Clang++ 7.0 part of the main branch
  - Detect memoization

Templight


Templight
C++
C++
C++
Clang source
Trace
Interactive debugger/ visualizer
Instantiation time and memory profiler
3rd party tools

How to use

```
$ clang++-7 --cc1 -templight-dump fib.cpp

---
name:            'fact<5>'
kind:            TemplateInstantiation
event:           Begin
orig:            'fact.cpp:2:8'
poi:             'fact.cpp:15:10'
---
name:            'fact<5>'
kind:            TemplateInstantiation
event:           End
orig:            'fact.cpp:2:8'
poi:             'fact.cpp:15:10'
---
name:            'fact<5>'
kind:            TemplateInstantiation
event:           Begin
orig:            'fact.cpp:2:8'
poi:             'fact.cpp:15:10'
---
name:            'fact<4>'
kind:            TemplateInstantiation
event:           Begin
orig:            'fact.cpp:2:8'
poi:             'fact.cpp:4:32'
```

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
16 {
17   static const int value = 1;
18 };
19
```

Event type:

Kind:

Name:

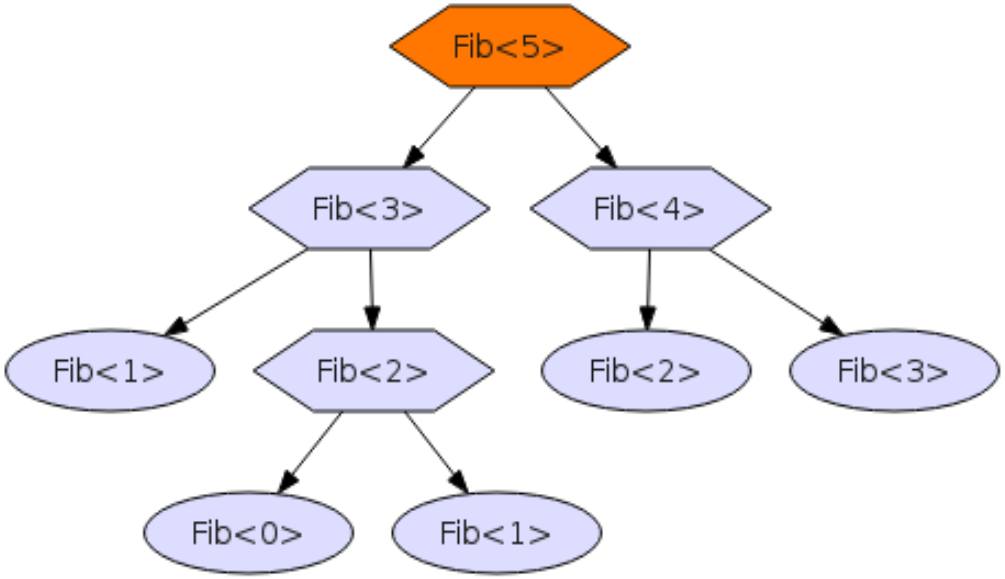
File position:

Templar

File Help

Breakpoint Filter Reset

```
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
```



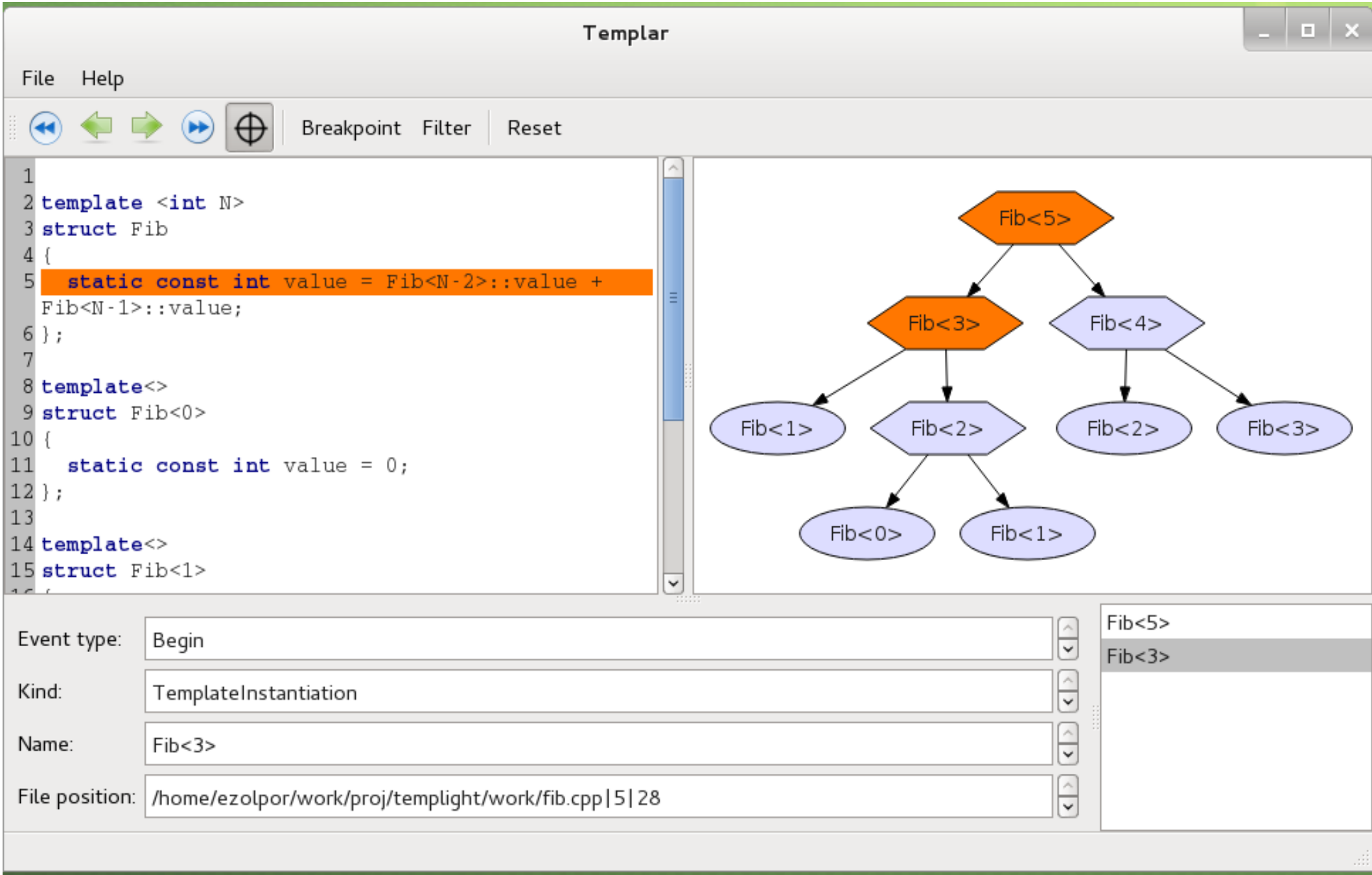
Event type: Begin

Kind: TemplateInstantiation

Name: Fib<5>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|22|14

Fib<5>

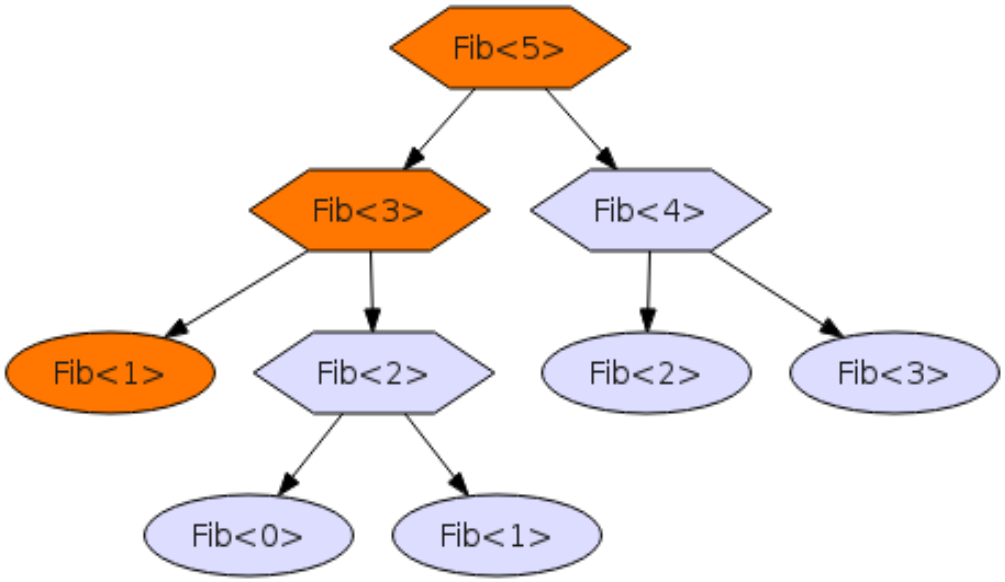

Templar
File
Help
Breakpoint
Filter
Reset
template <int N>
struct Fib
{
static const int value = Fib<N-2>::value +
Fib<N-1>::value;
};
template<>
struct Fib<0>
{
static const int value = 0;
};
template<>
struct Fib<1>
Fib<5>
Fib<3>
Fib<4>
Fib<1>
Fib<2>
Fib<2>
Fib<3>
Fib<0>
Fib<1>
Event type:
Begin
Kind:
TemplateInstantiation
Name:
Fib<3>
File position:
/home/ezolpor/work/proj/templight/work/fib.cpp|5|28
Fib<5>
Fib<3>

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
```



Event type: Begin

Kind: Memoization

Name: Fib<1>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28

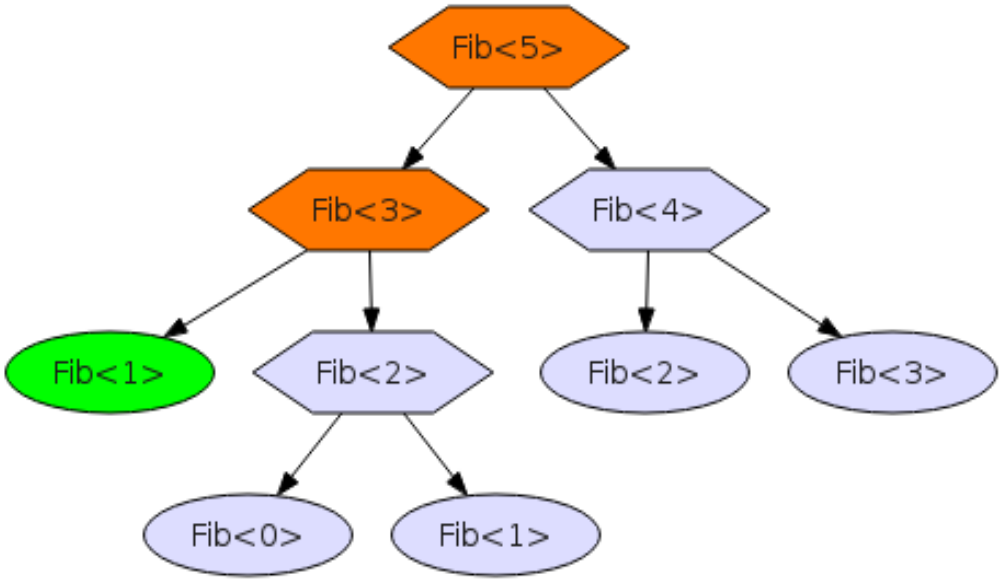
Fib<5>
Fib<3>
Fib<1>

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
```



Event type: End

Kind: Memoization

Name: Fib<1>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28

Fib<5>

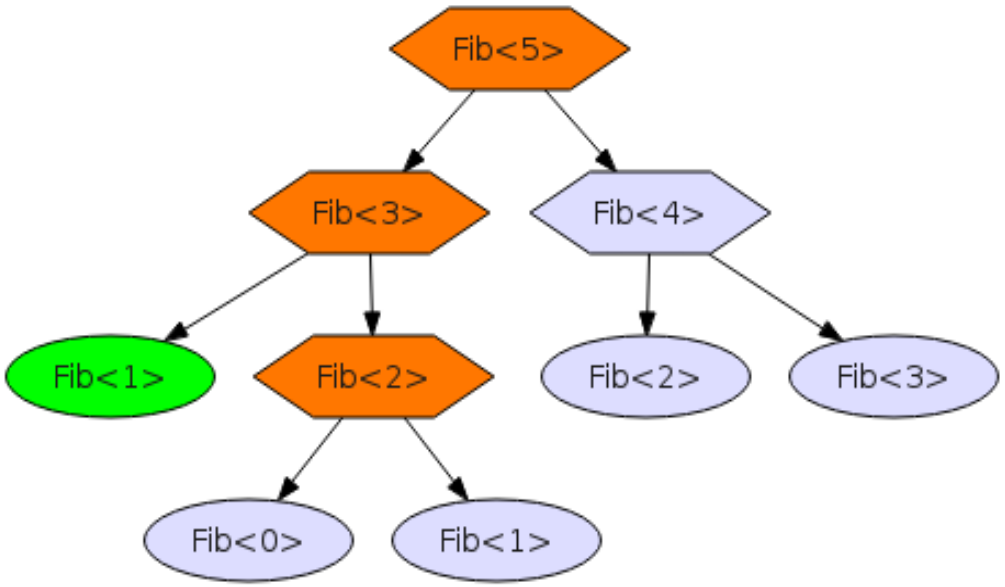
Fib<3>

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
```



Event type: Begin

Kind: TemplateInstantiation

Name: Fib<2>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|46

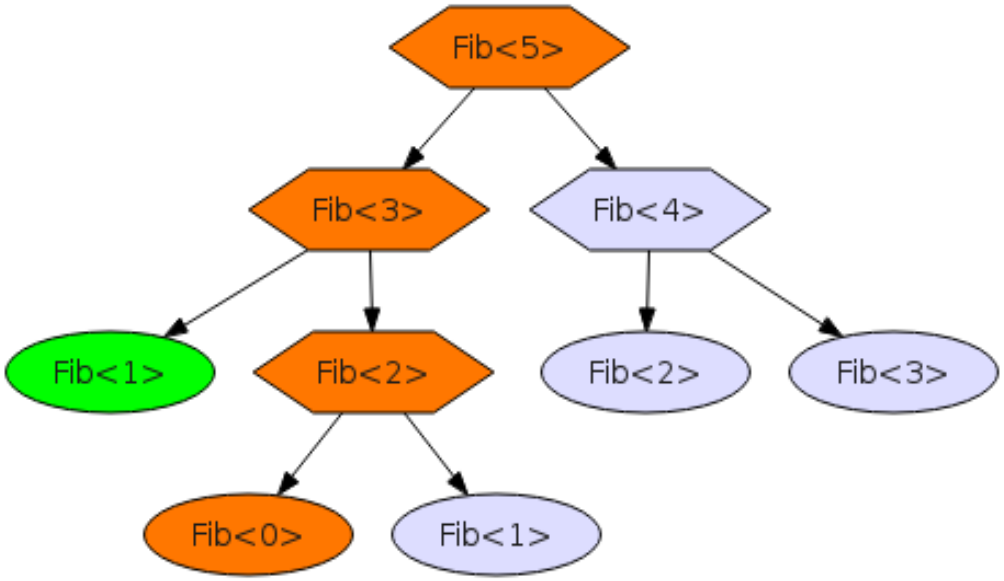
Fib<5>
Fib<3>
Fib<2>

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
```



Event type: Begin

Kind: Memoization

Name: Fib<0>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28

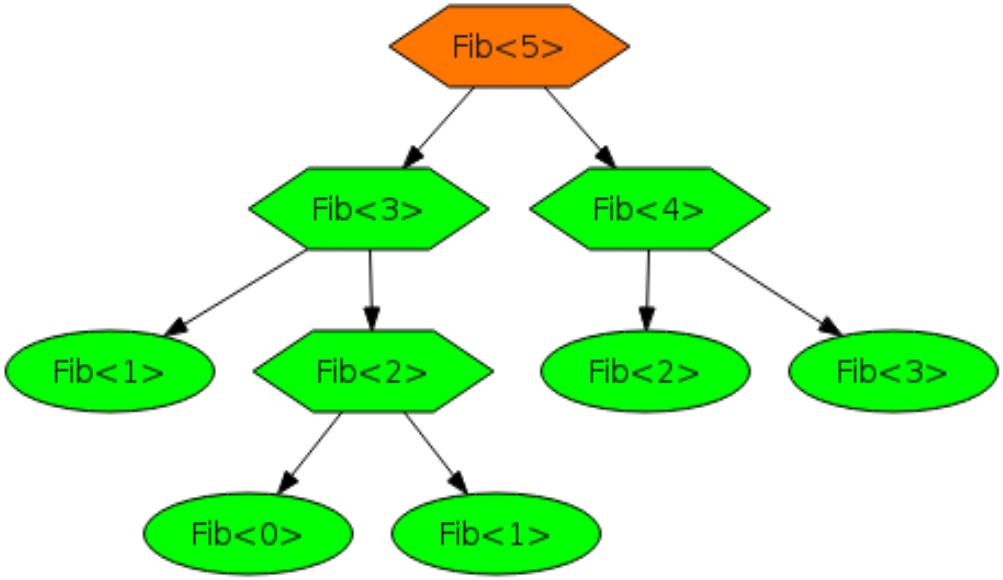
Fib<5>
Fib<3>
Fib<2>
Fib<0>

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
```



Event type: End

Kind: TemplateInstantiation

Name: Fib<4>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|46

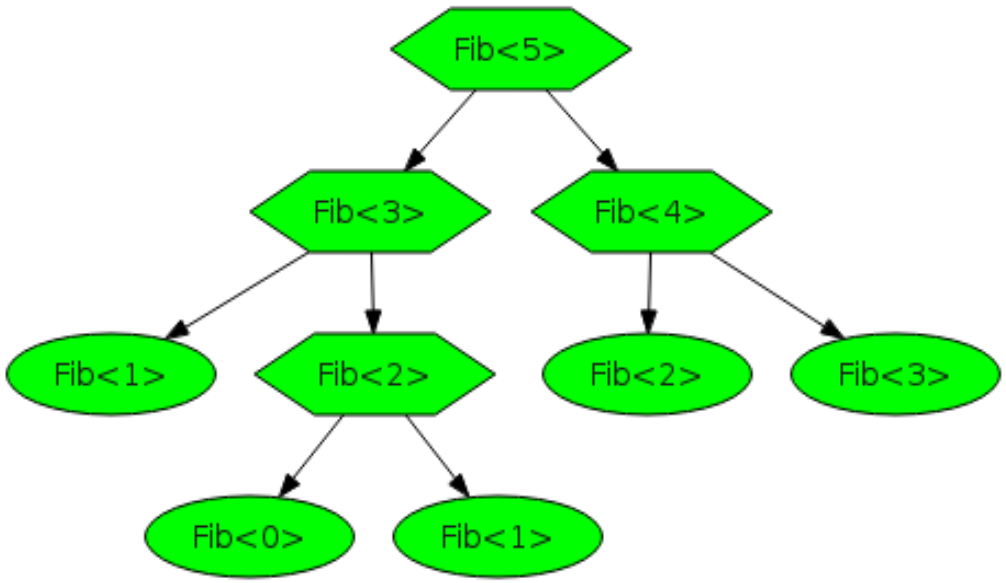
Fib<5>

Templar

File Help

Breakpoint Filter Reset

```
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

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
16 {
17   static const int value = 1;
18 };
19
```

Event type:

Kind:

Name:

File position:

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
16 {
17   static const int value = 1;
18 };
19
```

Dialog

Add

Delete

Delete All

Add Item

Enter Regexp:

Fib<1>

Cancel OK

Done

Event type:

Kind:

Name:

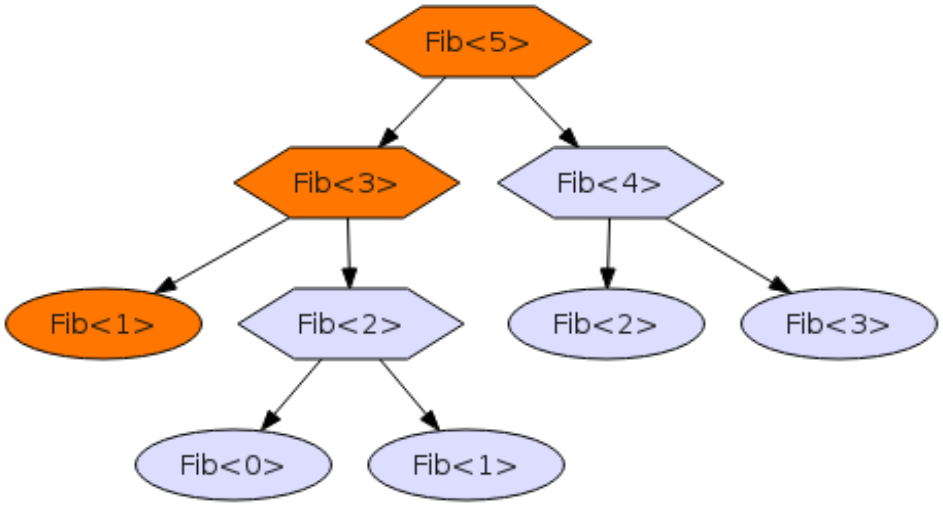
File position:

Templar

File Help

Breakpoint Filter Reset

```
1
2 template <int N>
3 struct Fib
4 {
5   static const int value = Fib<N-2>::value +
  Fib<N-1>::value;
6 };
7
8 template<>
9 struct Fib<0>
10 {
11   static const int value = 0;
12 };
13
14 template<>
15 struct Fib<1>
16 {
17   static const int value = 1;
18 };
19
```



Event type: Begin

Kind: Memoization

Name: Fib<1>

File position: /home/ezolpor/work/proj/templight/work/fib.cpp|5|28

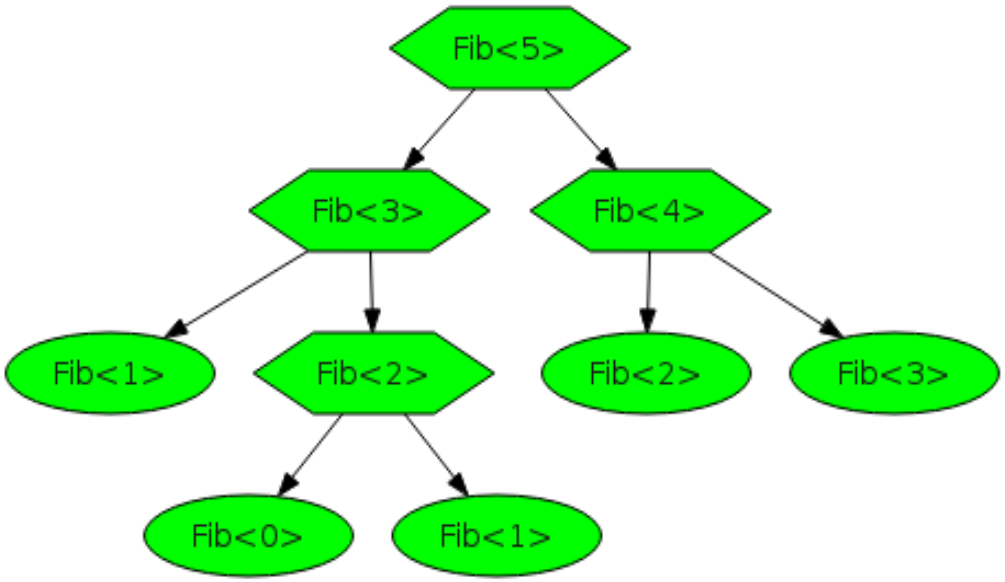
Fib<5>
Fib<3>
Fib<1>

Templar

File Help

Breakpoint Filter Reset

```
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

Related work

- Debugging
  - Static assert/Concept check (Siek-Lumsdaine, McNamara-Smaragdakis, Alexandrescu, others...)
  - Warning generation (many attempt)
  - Metashell http://metashell.org/
  - Instrumentation
- Profiling
  - Measuring full compilation (Gurtovoy-Abrahams)
  - Measuring warning appearance (Watanabe)
- Visualize
  - Source execution
  - Instantiation graph

Metashell – interactive TMP REPL

- Ábel Sinkovics and András Kucsma
- Metashell https://github.com/sabel83/metashell