Functional extensions to the Boost metaprogram library

Ábel Sinkovics

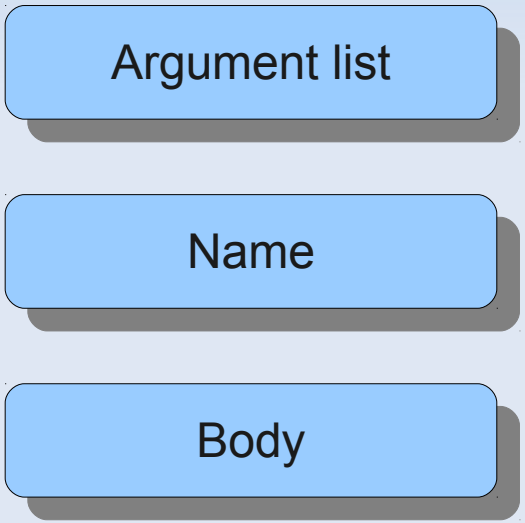
ELTE, Budapest
March 27, 2010

C++ template metaprograms

- Code evaluated at compilation time
- Erwin Unruh, 1994.
- Strong connection with functional programming
- Library support: Boost metaprogramming library
- The design of it is based on STL
- Basic building blocks of functional programming are missing

C++ template metaprograms

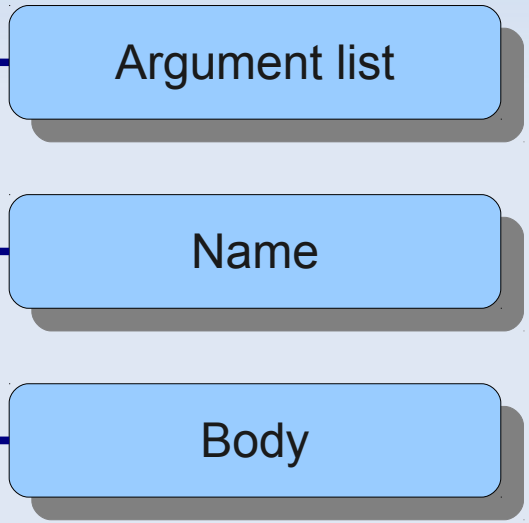
```
template <class T>
struct makeConst
{
  typedef const T type;
};
```




```
makeConst<int>::type
```

C++ template metaprograms

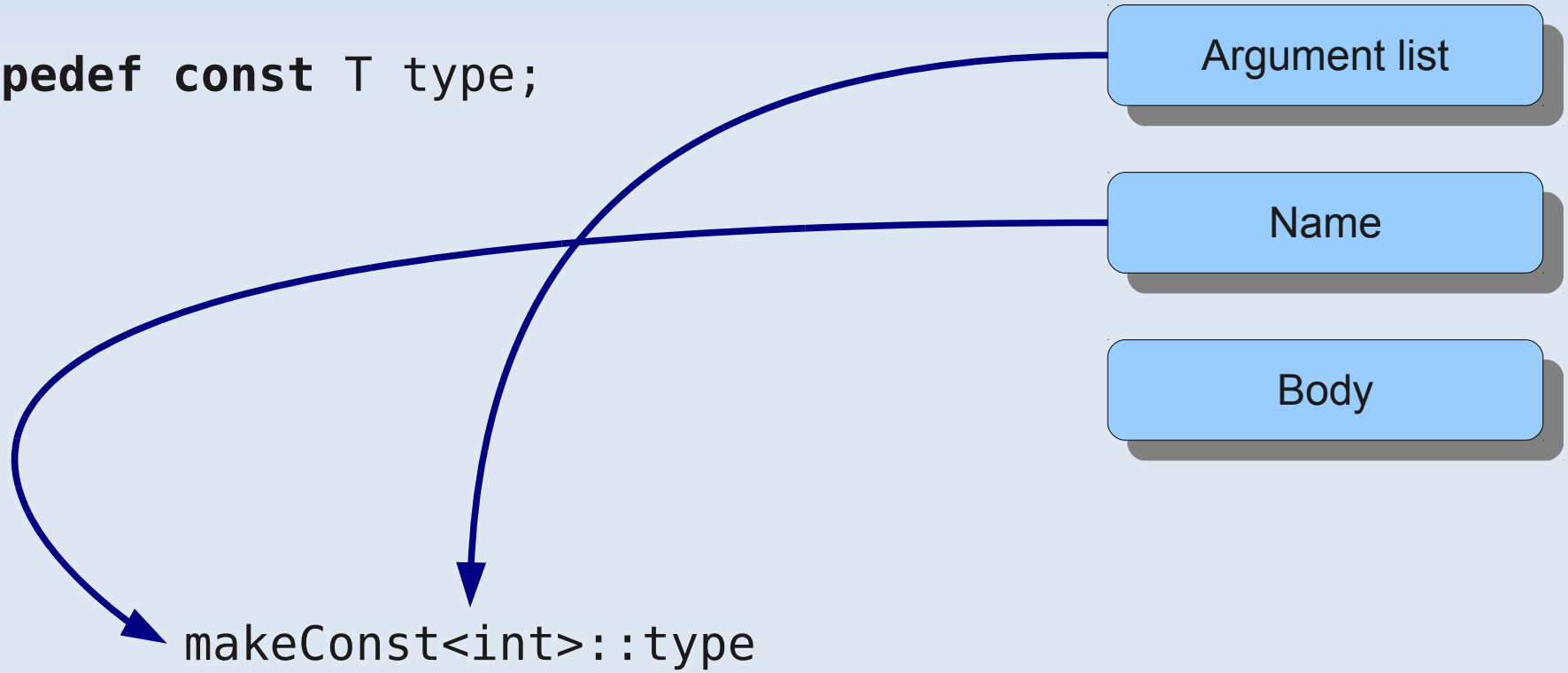
```
template <class T>
struct makeConst
{
  typedef const T type;
};
```




```
makeConst<int>::type
```

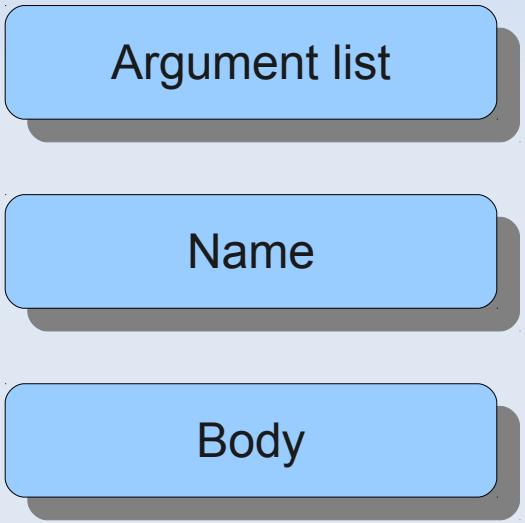
C++ template metaprograms

```
template <class T>
struct makeConst
{
  typedef const T type;
};
```



Template metafunction class

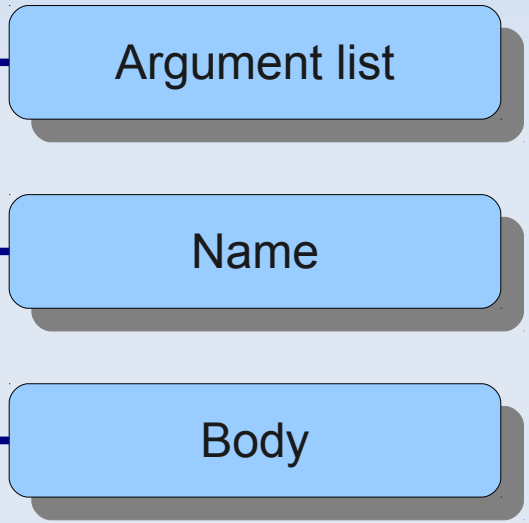
```
struct makeConst
{
  template <class T>
  struct apply
  {
    typedef const T type;
  };
};
```




```
makeConst::apply<int>::type
```

Template metafunction class

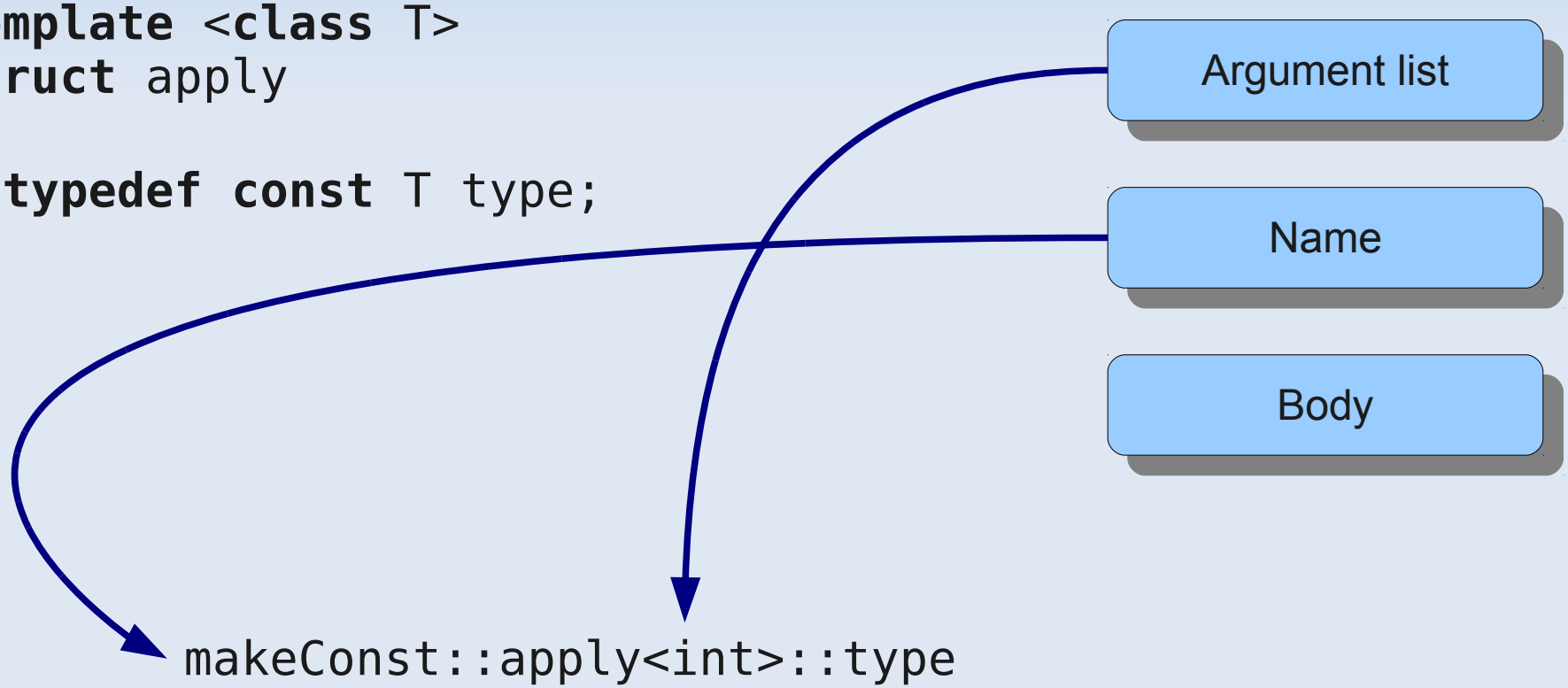
```
struct makeConst
{
  template <class T>
  struct apply
  {
    typedef const T type;
  };
};
```




```
makeConst::apply<int>::type
```

Template metafunction class

```
struct makeConst
{
   template <class T>
   struct apply
   {
      typedef const T type;
   };
};
```



Functional extensions to the Boost metaprogram library

- Laziness
- Function composition
- Currying

Nullary metafunctions

```
struct nullary_metafunction
{
  typedef int type;
};
```

Nullary metafunctions

```
struct nullary_metafunction
{
  typedef int type;
};

some_metafunction<int, double>
```

Nullary metafunctions

```
struct nullary_metafunction
{
  typedef int type;
};

some_metafunction<int, double>

nullary_metafunction::type
some_metafunction<int, double>::type
```

Laziness

```
template <class a, class b>
struct divide :




{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :




{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    // condition,


    // true branch,
    // false branch
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    equal_to<
      b, int_<0>
    >,
    // true branch,
    // false branch
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    typename equal_to<
      b, int_<0>
    >::type,
    // true branch,
    // false branch
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    typename equal_to<
      b, int_<0>
    >::type,
    infinite,
    typename divides<a, b>::type
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    typename equal_to<
      b, int_<0>
    >::type,
    infinite,
    typename divides<a, b>::type
  >
{};
```

```
divide<int_<7>, int_<0> >::type
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  if_<
    typename equal_to<
      b, int_<0>
    >::type,
    infinite,
    typename divides<a, b>::type
  >
{};
```

```
divide<int_<7>, int_<0> >::type
```

```
In file included from /usr/include/boost/mpl/aux_/include_pr
eprocessed.hpp:37,
                 from /usr/include/boost/mpl/aux_/arithmetic
                 _op.hpp:34,
                 from /usr/include/boost/mpl/divides.hpp:19,
                 from t1.cpp:2:
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of ‘boost::mpl::divides_impl<mpl_::integral_c
 _tag, mpl_::integral_c_tag>::apply<mpl_::int_<7>, mpl_::int
 _<0> >’:
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
   instantiated from ‘boost::mpl::divides<mpl_::int_<7>, mpl
   _::int_<0>, mpl_::na, mpl_::na, mpl_::na>’
t1.cpp:16:   instantiated from ‘divide<mpl_::int_<7>, mpl_::
int_<0> >’
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:142
: error: ‘(7 / 0)’ is not a valid template argument for type
 ‘int’ because it is a non-constant expression
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of ‘boost::mpl::divides<mpl_::int_<7>, mpl_::
 int_<0>, mpl_::na, mpl_::na, mpl_::na>’:
t1.cpp:16:   instantiated from ‘divide<mpl_::int_<7>, mpl_::
int_<0> >’
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
 error: no type named ‘type’ in ‘struct boost::mpl::divides_
 impl<mpl_::integral_c_tag, mpl_::integral_c_tag>::apply<mpl
 _::int_<7>, mpl_::int_<0> >’
t1.cpp: In instantiation of ‘divide<mpl_::int_<7>, mpl_::int
_<0> >’:
t1.cpp:20:   instantiated from here
t1.cpp:16: error: no type named ‘type’ in ‘struct boost::mpl
::divides<mpl_::int_<7>, mpl_::int_<0>, mpl_::na, mpl_::na,
mpl_::na>’
t1.cpp: In function ‘int main()’:
t1.cpp:20: error: ‘type’ is not a member of ‘divide<mpl_::in
t_<7>, mpl_::int_<0> >’
t1.cpp:20: error: expected ‘;’ before ‘x’
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    infinite,
    typename divides<a, b>::type
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    infinite,
    divides<a, b>
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    divides<a, b>
  >
{};
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    divides<a, b>
  >
{};
```

```
divide<int_<7>, int_<0> >::type
```

Laziness

```
struct infinite {};

template <class a, class b>
struct divide :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    divides<a, b>
  >
{};
```

```
divide<int_<7>, int_<0> >::type
```

Laziness

```
template <class a, class b>
struct some_calculation :




        {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    // when b is zero,
    // when b is not zero




      > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    // when b is not zero
```

```
      > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      // condition



      // ...,
      // ...
    > > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename less<
        typename divides<a, b>::type,
        int_<10>
      >::type,
      // ...,
      // ...
    > > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename less<
        typename divides<a, b>::type,
        int_<10>
      >::type,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename less<
        typename divides<a, b>::type,
        int_<10>
      >::type,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

```
In file included from /usr/include/boost/mpl/aux_/include_pr
eprocessed.hpp:37,
                 from /usr/include/boost/mpl/aux_/arithmetic
                 _op.hpp:34,
                 from /usr/include/boost/mpl/divides.hpp:19,
                 from t1.cpp:2:
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of 'boost::mpl::divides_impl<mpl_::integral_c
 _tag, mpl_::integral_c_tag>::apply<mpl_::int_<7>, mpl_::int
 _<0> >':
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
   instantiated from 'boost::mpl::divides<mpl_::int_<7>, mpl
   _::int_<0>, mpl_::na, mpl_::na, mpl_::na>'
t1.cpp:16:   instantiated from 'divide<mpl_::int_<7>, mpl_::
int_<0> >'
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:142
: error: '(7 / 0)' is not a valid template argument for type
 'int' because it is a non-constant expression
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of 'boost::mpl::divides<mpl_::int_<7>, mpl_::
 int_<0>, mpl_::na, mpl_::na, mpl_::na>':
t1.cpp:16:   instantiated from 'divide<mpl_::int_<7>, mpl_::
int_<0> >'
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
 error: no type named 'type' in 'struct boost::mpl::divides_
 impl<mpl_::integral_c_tag, mpl_::integral_c_tag>::apply<mpl
 _::int_<7>, mpl_::int_<0> >'
t1.cpp: In instantiation of 'divide<mpl_::int_<7>, mpl_::int
_<0> >':
t1.cpp:20:   instantiated from here
t1.cpp:16: error: no type named 'type' in 'struct boost::mpl
::divides<mpl_::int_<7>, mpl_::int_<0>, mpl_::na, mpl_::na,
mpl_::na>'
t1.cpp: In function 'int main()':
t1.cpp:20: error: 'type' is not a member of 'divide<mpl_::in
t_<7>, mpl_::int_<0> >'
t1.cpp:20: error: expected ';' before 'x'
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename apply<
        less< divides<a, _1>, int_<10> >,
        b
      >::type,
      // ...,
      // ...
    > > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename apply<
        less< divides<a, _1>, int_<10> >,
        b
      >::type,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    eval_if<
      typename apply<
        less< divides<a, _1>, int_<10>
        b
      >::type,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

```
In file included from /usr/include/boost/mpl/aux_/include_pr
eprocessed.hpp:37,
                 from /usr/include/boost/mpl/aux_/arithmetic
                 _op.hpp:34,
                 from /usr/include/boost/mpl/divides.hpp:19,
                 from t1.cpp:2:
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of 'boost::mpl::divides_impl<mpl_::integral_c
 _tag, mpl_::integral_c_tag>::apply<mpl_::int_<7>, mpl_::int
 _<0> >':
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
   instantiated from 'boost::mpl::divides<mpl_::int_<7>, mpl
   _::int_<0>, mpl_::na, mpl_::na, mpl_::na>'
t1.cpp:16:   instantiated from 'divide<mpl_::int_<7>, mpl_::
int_<0> >'
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:142
: error: '(7 / 0)' is not a valid template argument for type
 'int' because it is a non-constant expression
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp: In
 instantiation of 'boost::mpl::divides<mpl_::int_<7>, mpl_::
 int_<0>, mpl_::na, mpl_::na, mpl_::na>':
t1.cpp:16:   instantiated from 'divide<mpl_::int_<7>, mpl_::
int_<0> >'
t1.cpp:20:   instantiated from here
/usr/include/boost/mpl/aux_/preprocessed/gcc/divides.hpp:70:
 error: no type named 'type' in 'struct boost::mpl::divides_
 impl<mpl_::integral_c_tag, mpl_::integral_c_tag>::apply<mpl
 _::int_<7>, mpl_::int_<0> >'
t1.cpp: In instantiation of 'divide<mpl_::int_<7>, mpl_::int
_<0> >':
t1.cpp:20:   instantiated from here
t1.cpp:16: error: no type named 'type' in 'struct boost::mpl
::divides<mpl_::int_<7>, mpl_::int_<0>, mpl_::na, mpl_::na,
mpl_::na>'
t1.cpp: In function 'int main()':
t1.cpp:20: error: 'type' is not a member of 'divide<mpl_::in
t_<7>, mpl_::int_<0> >'
t1.cpp:20: error: expected ';' before 'x'
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    lazy_eval_if<
      apply<
        less< divides<a, _1>, int_<10> >,
        b
      >,
      // ...,
      // ...
    > > {};
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    lazy_eval_if<
      apply<
        less< divides<a, _1>, int_<10> >,
        b
      >,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

Laziness

```
template <class a, class b>
struct some_calculation :
  eval_if<
    typename equal_to<
      b, int_<0>
    >::type,
    identity<infinite>,
    lazy_eval_if<
      apply<
        less< divides<a, _1>, int_<10> >,
        b
      >,
      // ...,
      // ...
    > > {};
```

```
some_calculation<int_<7>,int_<0> >::type
```

Laziness

```
template <class condition, class true_b, class false_b>
struct lazy_eval_if :
  eval_if<
    typename condition::type,
    true_b,
    false_b
  >
{};
```

Function composition

- Common abstraction
- Simplifies code
- Could be implemented using lambda expressions

Function composition

```
// compose f4, f3, f2, f1
```

```
struct some_name_that_avoids_name_collision
{
  template <class x>
  struct apply :
    apply<
      f4,
      apply<f3, apply<f2, apply<f1, x> > >
    >
  {};
};
```

Function composition

```
// compose f4, f3, f2, f1
```

```
struct some_name_that_avoids_name_collision
{
  template <class x>
  struct apply :
    apply<
      f4,
      apply<f3, apply<f2, apply<f1, x> > >
    >
  {};
};
```

```
some_name_that_avoids_name_collision
```

Function composition

```
// compose f4, f3, f2, f1
```

```
template f4::apply< template f3::apply<
  template f2::apply< template f1::apply<
    _1
  > >
> >
```

compose<f4, f3, f2, f1>

Function composition

```
// compose f4, f3, f2, f1
```

```
template f4::apply< template f3::apply<
  template f2::apply< template f1::apply<
    _1
  > >
> >
```

```
compose<f4, f3, f2, f1>
```

Function composition

```
// compose f4, f3, f2, f1

template <class f4, class f3, class f2, class f1>
struct compose
{
  template <class x>
  {
    typedef
      template f4::apply< template f3::apply<
        template f2::apply< template f1::apply<
          x
        > >
      > >
      type;
  };
};
```

compose<f4, f3, f2, f1>

Currying

- Special form of function evaluation
- Commonly used
- When porting code written in a functional language to C++ template metaprogramming we need to express it in a simple way

Currying

```
template <class x1, class y1, class x2, class y2>
struct area :
  multiplies<minus<x2, x1>, minus<y2, y1> >
{};


area< int_<1>, int_<2>, int_<3>, int_<4> >::type
```

```
area::apply<
  int_<1>
>::type::apply<
  int_<2>
>::type::apply<
  int_<3>
>::type::apply<
  int_<4>
>::type
```

Currying

```
struct area {
  template <class x1> struct apply {
    template <class y1> struct apply {
      template <class x2> struct apply {
        template <class y2> struct apply :
          multiplies<minus<x2, x1>, minus<y2, y1> >
        {};
      };
    };
  };
};
```

```
area::apply<
  int_<1>
>::type::apply<
  int_<2>
>::type::apply<
  int_<3>
>::type::apply<
  int_<4>
>::type
```

Currying

```
struct area {
  template <class x1> struct apply {
    template <class y1> struct apply {
      template <class x2> struct apply {
        template <class y2> struct apply :
          multiplies<minus<x2, x1>, minus<y2, y1> >
        {};
      };
    };
  };
};
```

```
curry<area>
```

```
area::apply<
  int_<1>
>::type::apply<
  int_<2>
>::type::apply<
  int_<3>
>::type::apply<
  int_<4>
>::type
```

Currying

```
struct area {
  template <class x1> struct apply {
    template <class y1> struct apply {
      template <class x2> struct apply {
        template <class y2> struct apply :
          multiplies<minus<x2, x1>, minus<y2, y1> >
        {};
      };
    };
  };
};
```

```
curry<area, int_<4> >
```

```
area::apply<
  int_<1>
>::type::apply<
  int_<2>
>::type::apply<
  int_<3>
>::type::apply<
  int_<4>
>::type
```

Related work

- Todd Veldhuizen
- Loki
- FC++
- Bartosz Milewski
- E-Clean

Currying

```
template <
  class UnpackedMetafunctionClass,
  class ArgumentsLeft,
  class ArgumentList
>
struct curryImpl :
  eval_if<
    typename equal_to<ArgumentsLeft, int_<0> >::type,
    apply<UnpackedMetafunctionClass, ArgumentList>,
    nextCurryingStep<
      UnpackedMetafunctionClass,
      ArgumentsLeft,
      ArgumentList
    >
  >
{};
```

Functional extensions to the Boost metaprogram library

- Laziness
- Function composition
- Currying

Functional extensions to the Boost metaprogram library

Ábel Sinkovics
abel@elte.hu