# Лекция 5. Примеры применения метапрограммирования

## Метапрограммирование в C++

# Генерирование при помощи mpl::fold

## Пример

```
typedef
  mpl::vector
  <
    short [2], long, char *, int
  > member_types;

template <class T, class More>
  struct store : More
{
  T value;
};
```

## Пример (окончание)

```
struct empty { };

typedef
  mpl::fold
  <
    member_types,
    empty,
    store <_2, _1>
  >::type generated;
```

# Генерирование при помощи mpl::fold

## Пример

```
typedef
  mpl::vector
  <
    short [2], long, char *, int
  > member_types;

template <class T, class More>
  struct store : More
{
  T value;
};
```

## Пример (окончание)

```
typedef
  mpl::fold
  <
    member_types,
    mpl::empty_base,
    store <_2, _1>
  >::type generated;
```

# Сгенерированная специализация

## Пример

```
store
<
  int,
  store
  <
    char *,
    store
    <
      // ...
    >
  >
>
```

# Доступ к полю

## Пример

```
int main()
{
  generated g;
  // ...
  long &rl = static_cast <store <long, store <short[2], empty> > &>
    (g).value;
  // ...
}
```

# Непосредственное генерирование

## Пример

```
typedef
  mpl::vector
  <
    short [2], long, char *, int
  > member_types;

template <class T>
  struct wrap
{
  T value;
};
```

## Пример (окончание)

```
template <class U, class V>
  struct inherit : U, V
{ };

typedef
  mpl::fold
  <
    member_types, empty,
    inherit <wrap <_2>, _1>
  >::type generated;
```

# Непосредственное генерирование

## Пример

```
typedef
  mpl::vector
  <
    short [2], long, char *, int
  > member_types;

template <class T>
  struct wrap
{
  T value;
};
```

## Пример (окончание)

```
typedef
  mpl::inherit_linearly
  <
    member_types,
    mpl::inherit <wrap <_2>, _1>
  >::type generated;
```

# Сгенерированная специализация

## Пример

```
inherit
<
  wrap <int>,
  inherit
  <
    wrap <char *>,
    inherit
    <
      // ...
    >
  >
>
```

# Доступ к полю

## Пример

```
int main()
{
  generated g;
  // ...
  long &rl = static_cast <wrap <long> &> (g).value;
  // ...
}
```

# Основные размерности

## Пример

```
typedef mpl::vector_c <int,  1,  0,  0,  0,  0,  0,  0> mass;
typedef mpl::vector_c <int,  0,  1,  0,  0,  0,  0,  0> length;
typedef mpl::vector_c <int,  0,  0,  1,  0,  0,  0,  0> time;
typedef mpl::vector_c <int,  0,  0,  0,  1,  0,  0,  0> charge;
typedef mpl::vector_c <int,  0,  0,  0,  0,  1,  0,  0> temperature;
typedef mpl::vector_c <int,  0,  0,  0,  0,  0,  1,  0> intensity;
typedef mpl::vector_c <int,  0,  0,  0,  0,  0,  0,  1> angle;

typedef mpl::vector_c <int,  0,  0,  0,  0,  0,  0,  0> scalar;
```

# Основные размерности

## Пример

```
//                                m   l   t
typedef mpl::vector_c <int,  0,  1, -1,  0,  0,  0,  0> velocity;
//                                                       l/t
typedef mpl::vector_c <int,  0,  1, -2,  0,  0,  0,  0> acceleration;
//                                                       l/t^2
typedef mpl::vector_c <int,  1,  1, -1,  0,  0,  0,  0> momentum;
//                                                       m l/t
typedef mpl::vector_c <int,  1,  1, -2,  0,  0,  0,  0> force;
//                                                       m l/t^2
// ...
```

# Представление величины

## Пример

```
template <class T, class TDimensions>
  struct quantity
{
  explicit quantity(T x) :
    m_value(x)
  { }
  T value() const
    { return m_value; }
private:
  T m_value;
};
```

# Использование величин

## Пример

```
int main()
{
  quantity <float, length> l(1.0f);
  quantity <float, mass>   m(2.0f);
  m = l;     // ошибка компиляции
  // ...
}
```

# Реализация сложения

## Пример

```
template <class T, class D>
  quantity <T, D> operator + (quantity <T, D> x, quantity <T, D> y)
{
  return quantity <T, D> (x.value() + y.value());
}
```

# Использование сложения

## Пример

```
int main()
{
  quantity <float, length> l1(1.0f);
  quantity <float, length> l2(2.0f);
  quantity <float, mass>   m1(3.0f);
  l1 = l1 + l2;
  l1 = l2 + m1;    // ошибка компиляции
  // ...
}
```

# Реализация умножения

## Пример

```
template <typename D1, typename D2>
  struct multiply_dimensions :
    mpl::transform <D1, D2, mpl::plus <_, _> > { };

template <typename T, typename D1, typename D2>
  quantity <T, typename multiply_dimensions <D1, D2>::type>
    operator * (quantity <T, D1> q1, quantity <T, D2> q2)
{
  typedef typename multiply_dimensions <D1, D2>::type Dim;
  return quantity <T, Dim>(q1.value() * q2.value());
}
```

# Использование умножения

## Пример

```
int main()
{
  quantity <float, mass>         m(5.0f);
  quantity <float, acceleration> a(9.8f);
  //
  cout << "Сила = " << (m * a).value() << endl;
  //
  quantity <float, force>        f = m * a;    // ошибка компиляции
  // ...
}
```

# Шаблонный конструктор величины

## Пример

```
template <class T, class TDimensions>
  struct quantity
{
  // ...
  template <typename TOtherDimensions>
    quantity(quantity <T, TOtherDimensions> q)
    : m_value(q.value())
  {
    BOOST_MPL_ASSERT((mpl::equal <TDimensions, TOtherDimensions>));
  }
  // ...
};
```

# Использование умножения со сложением

## Пример

```
int main()
{
  quantity <float, mass>         m(5.0f);
  quantity <float, acceleration> a(9.8f);
  quantity <float, force>        f(0.0f);
  // ...
  f = f + m * a;     // ошибка компиляции
  // ...
}
```

# Реализация присваивания

## Пример

```
template <class T, class D1, class D2>
  quantity <T, D1> operator + (quantity <T, D1> x, quantity <T, D2> y)
{
  BOOST_MPL_ASSERT((mpl::equal <D1, D2>));
  return quantity <T, D1> (x.value() + y.value());
}
```

# Проблемно-ориентированные языки

## Определения

Проблемно-ориентированный язык: (*Domain-specific language, DSL*) — язык, использующий обозначения и абстракции предметной области для решения специфичных для неё задач.

Проблемно-ориентированный встроенный язык: (*Domain-specific embedded language, DSEL*) — библиотека, обладающая теми же свойствами.

# Матричные вычисления

## Пример (BLAS)

```
int main()
{
  double dA, dB;
  double *pdA, *pdB, *pdC;
  // ...
  cblas_dgemm(
    CblasRowMajor, CblasNoTrans,
    CblasNoTrans, n, n, n,
    dA, pdA, 1, pdB, 1, dB, pdC, 1);
  // ...
}
```

## Пример

```
int main()
{
  Matrix a, b, c, x;
  double dA, dB;
  // ...
  x = dA * a * b + dB * c;
  // ...
}
```

# Реализация сложения векторов

## Пример

```
Vector operator + (const Vector &rcA, const Vector &rcB)
{
  const std::size_t cn = rcA.size();
  Vector result(cn);
  for (std::size_t i = 0; i != cn; ++ i)
    result[i] = rcA[i] + rcB[i];
  //
  return result;
}
```

# Использование сложения векторов

## Пример

```
int main()
{
  Vector a, b, c, x;
  // ...
  x = a + b + c;
  // ...
}
```

# Реализация ленивого выражения

## Пример

```
template <class L, class OpTag, class R>
  struct Expression
{
  Expression(const L &rcL, const R &rcR)
    : m_rcL(rcL), m_rcR(rcR)
  { }
  float operator [] (unsigned index) const
  {
    return OpTag::apply(m_rcL[index], m_rcR[index]);
  }
  const L &m_rcL;
  const R &m_rcR;
};
```

# Реализация ленивого сложения

## Пример

```
template <class L, class R>
  Expression <L, plus, R> operator + (
    const L &rcL, const R &rcR)
{
  return Expression <L, plus, R> (rcL, rcR);
}
```

## Пример

```
Expression <Expression <Vector, plus, Vector>, plus, Vector>
```

# Реализация функции сложения

## Пример

```
struct plus
{
  static float apply(float a, float b)
  {
    return a + b;
  }
};
```

# Реализация присваивания вектора

## Пример

```
template <class TExpr>
  Vector &Vector::operator = (const TExpr &rcX)
{
  for (unsigned i = 0; i < this->size(); ++ i)
    (*this)[i] = rcX[i];
  //
  return *this;
}
```

# Хранение последовательностей вычислений

## Пример

```
Expression
<
  Expression <Vector, plus, Vector>,
  plus,
  Expression <Vector, minus, Vector>
>
  intermediate = a + b + (c - d);
```

## Пример

```
auto intermediate = a + b + (c - d);
```