C++11 Variadic Templates

markus fangerau

ellipsis-operator

```
try {
    // Try block.
}
catch( ... )
{
    // Catch block.
}
```

Already known: Variadic Functions

```
int printf(const char* format, ... );

#include <stdarg.h>

void PrintDoubles (int n, ...)
{
  va_list vl;
  va_start( vl , n );
  for ( int i=0 ; i<n ; i++)
  {
    double val = va_arg( vl , double );
    std::cout<< val << std::endl;
  }
  va_end( vl );
}
```

Already known: Preprocessor Macros

```
#define dbgprintf( ... ) \
  realdbgprintf ( __FILE__ , __LINE__ , __VA_ARGS__ )
```

Variadic Templates

```
template < typename ... Args >
class my_class
{
};

my_class< int > mc1;
my_class< double , char , std::string > mc2;
my_class<> mc3;
```

Variadic Templates

```
template <typename T,typename U,typename ... Args>
class test
{
};

test< int , char , double , std::string > x1; // Args is
                                    < double , std::string >
test< std::string , my_class< int > > x2; // Args is empty
```

Variadic Templated Functions

```
template < typename T>
void print_comma_separated_list(T value)
{
   std::cout<<value<<std::endl;
}

template <typename First, typename ... Rest>
void print_comma_separated_list( First first , Rest ... rest )
{
   std::cout<<first<<",";
   print_comma_separated_list( rest... );
}

print_comma_separated_list(42,"hello",2.3,'a');
```

sizeof...()

```
template < typename ... Args >
unsigned how_many_args(Args ... args)
{
    return sizeof...(args);
}
```

simple_tuple example 1

```
template<typename ... Types>
class simple_tuple;


template<>
class simple_tuple<>
{
};
```

simple_tuple example 2

```
template< typename First , typename ... Rest>
class simple_tuple < First , Rest... > : private
simple_tuple<Rest... >
{
   First member;
   public:
   simple_tuple(First const& f,Rest const& ... rest):
      simple_tuple<Rest...>(rest...),
      member(f) {}
   First const& head() const { return member; }
   simple_tuple<Rest...> const& rest() const { return *this; }
};
```

simple_tuple example 3

simple_tuple<int,char,double> st(42,'a',3.141);

- To access the first element of a tuple, you can call t.head()
- To access the second, you call t.rest().head()
- To access the third, you call t.rest().rest().head()

Danke

based on
An Introduction to Variadic Templates in C++0x
by Anthony Williams
http://www.devx.com/cplus/Article/41533