嵌入式技术

高阶编程

邢超

<EC.1>

1 Standard Template Library (STL)

Expression Template

```
template <int dim, class T>
struct inner_product
{
    T operator()(T* a, T* b)
    {
        return inner_product<dim - 1,
            T>()(a + 1, b + 1) +
            inner_product<1, T>()(
                a, b);
    }
};

template <class T>
struct inner_product<1, T>
{
    T operator()(T* a, T* b)
    { return (*a) * (*b); }
};

inner_product<4, int>()(a, b);
```

<EC.2>

Fibonacci Sequence at compile time

```
#include <iostream>
#include <cassert>
using namespace std;

template<int stage> struct Fib {
  static const uint64_t value =
                Fib<stage-1>::value +
                  Fib<stage-2>::value
                  ;
   static inline uint64_t getValue(
      int i)
  {
    if (i == stage)
    {
      return value;
    } else {
      return Fib<stage-1>::getValue(i
        );
    }
  }
};
```

<EC.3>

Fibonacci Sequence at compile time

```
template<> // Template specialization
     for the 0's case.
struct Fib<0>
{
  static const uint64_t value = 1;

  static inline uint64_t getValue(int
      i)
  {
    assert(i == 0);
    return 1;
  }
};
```

<EC.4>

Fibonacci Sequence at compile time

```
template<> // Template specialization
     for the 1's case
struct Fib<1>
{
  static const uint64_t value = 1;

  static inline uint64_t getValue(int
      i)
  {
    if (i == 1)
    {
      return value;
    } else {
      return Fib<0>::getValue(i);
    }
  }
};
```

<EC.5>

Fibonacci Sequence at compile time

```
int main(int, char *[])
{
  //Generate (at compile time) 100
     places of the Fib sequence.
  //Then, (at runtime) output the 100
      calculated places.
  //Note: a 64 bit int overflows at
     place 92
  for (int i = 0; i < 100; ++i)
  {
    cout << "n:=" << i << "␣=>␣" <<
        Fib<100>::getValue(i) << endl
        ;
  }
}
```

<EC.6>

2 Lisp Macro

Lisp Macro

- comp.lang.lisp and any other comp.lang.* group with macro in the subject
  - Lispnik: "Lisp is the best because of its macros!"
  - Othernik: "You think Lisp is good because of macros?! But macros are horrible and evil; Lisp must be horrible and evil."
- Usage
  - function
  - lazy evaluation
  - syntax
  - Domain Specific Language (DSL)

<EC.7>

defmacro

```
(defmacro backwards (expr) (reverse
    expr))
(macroexpand '(backwards ("hello␣
    world!" t format)))
```

<EC.8>

3 camlp4

Caml Preprocessor and Pretty-Printer one of its most important applications is the definition of domain-specific extensions of the syntax of OCaml author: Daniel de Rauglaudre

4 思考

思考

- 当前常见程序设计语言的新特性是什么？
- 高阶编程的优缺点是什么？

<EC.9>