C++ Template Meta Programming

Francesco Nidito

Programmazione Avanzata AA 2005/06
Outline

1 Introduction

2 Template Metaprogramming

3 A Bad Example

4 A Good Example

5 Conclusions

At this moment is quite difficult to give a definition. First define two terms:
- Template
- Metaprogramming
What Are Templates?

- Templates are a C++ technique to create Generic Code
- Some templates are in C++ Standard and can be found in STL
- The first version of STL architecture was made by Alexander Stepanov
- Some examples of templates in STL are: std::vector, std::map and std::list
C++ Templates Example: Very Simple Array

template<typename T, unsigned int L>
class VerySimpleArray{
    T a[L];
    public:
        VerySimpleArray(){
            for(int i=0; i < L; ++i) a[i] = T();
        }
        T& Get(unsigned int i){return a[i];}
        void Set(unsigned int i, T& v){a[i] = v;}
};

VerySimpleArray<float, 42> vsa;
vsa.Set(3, 6.0);
C++ Templates Example: Template specialization

template<typename _Tp, typename _Alloc=allocator<_Tp> >
class vector : protected _Vector_base<_Tp, _Alloc>
{
    __glibcpp_class_requires(_Tp, _SGIAssignableConcept)

    typedef _Vector_base<_Tp, _Alloc> _Base;

    ...

template <typename _Alloc>
class vector<bool,_Alloc>: public _Bvector_base<_Alloc>
{
    public:
        typedef bool value_type;

    ...

/usr/include/c++/3.3.2(bits/std_vector.h
/usr/include/c++/3.3.2(bits/std_bvector.h
What Is Metaprogramming?

- **Metaprogramming** means writing programs that can modify other programs or themselves.
- The modification can happen at run time or at compile time.
- Compilers are an example of metaprogramming: they take a program in an input language (C, Java...) and produce another program (with same **semantic**) in an output language (machine code, bytecode...).
- **Metaprograms** have three advantages:
  - Produce code quickly
  - Lift abstraction
  - Produce correct code (if the metaprogram is correct).
What Is Template Metaprogramming?

- Now it is time for the replay, Template Metaprogramming is...
- Producing metapograms in C++ using templates
- But... wait a second... why do I need Template Metaprogramming?
The Need For Speed

```cpp
int Fact(int x) {
    int acc = 1;
    for(;x>0;--x) acc*=x;
    return acc;
}

int a, b;

a = read_int_from_stdin();
b = Fact(a); //we compute at run time
...

b = Fact(5); //we can compute at compile
    //time but we do not
```

We pay the time to compute something that is a constant!
Naive Solution

... 

b = 120; // b = Fact(5);

The solution is not elegant. We need to place a lot of Magic Numbers in the code. And the magic number does not keep its meaning (5! = 120 or 42 + 78 = 120 ...).
We can use a #define but it is not better: we must provide a define for all possible value of the input of Fact.

... 
#define FACT_4 (24)
#define FACT_5 (120)
... 

b = FACT_5;
The general idea is to use the compiler to do some computation at compile time.

To do this we need a only a C++ compiler that provides Templates

This can be done because the compiler, when compiles Templates, produces code

If we produce the right code (e.g. the sum of constant terms), the compiler optimize and do constant folding

But we can do more...
First Step

template< int i>
struct C {
    enum { RES = i };
};

cout << C<2>::RES;

C<2>::RES is substituted by the compiler with 2: it is a cost ant and we can optimize.
Back To Factorial

```cpp
template<int n>
struct Fact {
    enum { RET = n * Fact<n-1>::RET };
};

template<>
struct Fact<1> {
    enum { RET = 1 };
};

int b = Fact<5>::RET; // == 120
```

To do computation we must unroll templates: a kind of recursion
enum { RET = 5 * Fact<4>::RET };  
enum { RET = 5 * 4 * Fact<3>::RET };  
enum { RET = 5 * 4 * 3 * Fact<2>::RET };  
enum { RET = 5 * 4 * 3 * 2 * Fact<1>::RET };  
enum { RET = 5 * 4 * 3 * 2 * 1 };  
enum { RET = 120 };  

b = 120;  //Fact<5>::RET;
template <bool C, class T, class E>
struct IF {
    typedef T RET;
};

template <class T, class E>
struct IF<false, T, E> {
    typedef E RET;
};
class CopyingGC {
public:
    void Collect() /*...*/
};

class MarkAndSweepGC {
public:
    void Collect() /*...*/
};

IF<GC == COPY, CopyingGC, MarkAndSweepGC>::RET gc;
gc.Collect();
C++ Template Metaprogramming Functions

Template<int n1, int n2>
struct Max {
    enum { RET = (n1 > n2) ? n1 : n2 };
};

cout << Max<42, 6>::RET; //prints 42
C++ Template Metaprogramming Code Generation

```c++
inline int Power(const int x, int n){
    int p = 1;
    for(;n>0; --n) p *= x;
    return p;
}
```

We can specialize `Power` code for particular cases:

```c++
inline int Power3(const int x){
    int p = 1;
    p *= x;
    p *= x;
    p *= x;
    return p;
}
```
C++ Template Metaprogramming Code Generation

template<int n>
inline int Power(const int x){
    return Power<n-1>(x) * x;
}

template<>
inline int Power<1>(const int x){
    return 1;
}

template<>
inline int Power<0>(const int x){
    return 1;
}

cout << Power<4>(m);
C++ Template Metaprogramming Code Generation

cout << Power<4>(m);

cout << Power<3>(m) * m;

cout << Power<2>(m) * m * m;

cout << Power<2>(m) * m * m * m;

cout << Power<1>(m) * m * m * m * m;

cout << 1 * m * m * m * m;

cout << m * m * m * m;
C++ Template Metaprogramming Loop Unrolling(1)

template<int n, class B>
struct FOR {
    static void loop(int m) {
        for (int i = 0; i < m/n; i++) {
            UNROLL<n, B>::iteration(i * n);
        }
        for (int i = 0; i < m%n; i++) {
            B::body(n * m/n + i);
        }
    }
};
template <int n, class B>
struct UNROLL{
    static void iteration(int i) {
        B::body(i);
        UNROLL<n-1, B>::iteration(i + 1);
    }
};

template <class B>
struct UNROLL<0, B> {
    static void iteration(int i) {} 
};
C++ Template Metaprogramming Code Generation

- We can generate code for multiple purposes:
  - Vector operations
  - Math functions
  - ...

- **Exercise**: try to write some C++ Template Metaprogramming functions and control structures (**WHILE**)
Lisp lovers know very well this code:
(cons 1 (cons 2 (cons 3 (cons 4 (cons nil)))))

The previous code represents the list:
(1 2 3 4)

C++ Template Metaprogrammers have this too...
```cpp
template <int n, class T>
struct Cons {
    enum { item = n };
    typedef T next;
};

struct Nil {
};

typedef Cons<1, Cons<2, Nil()>> V;
//V::item == 1;
//V::next::item == 2;
//V::next::next == Nil;

Exercise: try to create a Tree
We want to copy all elements of an int array into another
We want to do it fast
We can use an approach similar to FOR template metastructure
Total Loop Unroll (C++ Code 1)

template<int N>
inline void Copy(int n, int* from, int* to){
    for (int i = 0; i < n/N; ++i) {
        TMP_COPY_UNROLL<N>::iteration(i * N, from, to);
    }
    for (int i = 0; i < n%N; ++i) {
        *to++ = *from++;
    }
}
template <int N>
struct TMP_COPY_UNROLL {
    static void iteration(int i, int *f, int* t){
        *t++ = *f++;
        TMP_COPY_UNROLL<N-1>::iteration(i + 1, f, t);
    }
};

template <>
struct TMP_COPY_UNROLL<1> {
    static void iteration(int i, int* f, int* t){
        *t++ = *f++;
    }
};
Performance Results

We measure the “Execution Time” of various values of unroll and...

...surprise

<table>
<thead>
<tr>
<th># Loop Unroll</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.16</td>
</tr>
<tr>
<td>10</td>
<td>1.79</td>
</tr>
<tr>
<td>50</td>
<td>2.23</td>
</tr>
<tr>
<td>100</td>
<td>2.70</td>
</tr>
<tr>
<td>500</td>
<td>2.81</td>
</tr>
<tr>
<td>1000</td>
<td>2.84</td>
</tr>
<tr>
<td>2000</td>
<td>4.83</td>
</tr>
<tr>
<td>5000</td>
<td>19.48</td>
</tr>
</tbody>
</table>
Performance Results (Chart)
Performance Results (An Explanation)

- Why?
- Loop unroll produce very big executable files
- Big executable files cannot be kept in the code cache
- We have a lot of cache misses
Meta Programming Library is part of Boost library

Boost is a portable C++ library that provides a big number of utilities:
- Threads
- Containers
- Math
- I/O
- Much more (circa 70 sub libraries)...

“The Boost.MPL library is a general-purpose, high-level C++ template metaprogramming framework of compile-time algorithms, sequences and metafunctions”
typedef list_c<int,0,1,2,3,4,5,6,7,8,9>::type numbers;
typedef list_c<int,0,1,2,3,4>::type answer;
typedef copy_if<numbers,
    vector_c<int>,
    push_back<_1,_2>,
    less<_1,int_<5> >
>::type result;

BOOST_STATIC_ASSERT(size<result>::value == 5);
BOOST_STATIC_ASSERT((equal<result,answer>::type::value));
Conclusions

- C++ template metaprogramming is a powerful method to do computational tasks at compile time.
- First approach is not very easy.
- Some lib is present (general, matrix/math...)
- Must be careful on compile errors (the templates tree is unrolled!)