

C++: Tecniche di (meta)programmazione

Prerequisiti:

- C++ standard (template compresi)
- OOP

Introduzione

Definizione di metaprogrammazione da wikipedia:

Metaprogramming is the writing of programs that write or manipulate other programs (or themselves) as their data or that do part of the work that is otherwise done at runtime during compile time. In many cases, this allows programmers to get more done in the same amount of time as they would take to write all the code manually.

Introduzione

Un esempio in bash:

```
#!/bin/bash
# metaprogram
echo '#!/bin/bash' >program
for ((I=1; I<=992; I++)); do
    echo "echo $I" >>program
done
chmod +x program
```

Introduzione

Runtime:

- Imperativa
- Value

Compile time:

- Funzionale
- Type

Richiami di C++

Esempi di template:

```
// function template  
template<class T> void f(T &);
```

```
// class template  
template<class T> struct X { };
```

Richiami di C++

Template specialization (completa e parziale):

```
template<class T, class U>  
struct X { };
```

```
template<class T>  
struct X<T, int> { }; // Partial  
    specialization
```

```
template<>  
struct X<int, int> { }; // Complete
```

Compile time If

```
// if(cond) A else B  
template<bool>  
struct X;
```

```
template<>  
struct X<true> { }; /* A */
```

```
template<>  
struct X<false> { }; /* B */
```

Compile time assert

Spesso alcune assertion che potrebbero essere valutate compile time sono controllate runtime. Ad esempio:

- `sizeof(void *) == sizeof(int)`

Compile time assert

```
// Forward declaration
template<bool>
class StaticAssert;

// Specialization with true
template<>
class StaticAssert<true>
{ };
```

- Utilizza l'inesistenza di una specializzazione con false per generare un errore ct

Compile time assert

Nel mondo reale:

```
BOOST_STATIC_ASSERT((  
    sizeof(int) > sizeof(char)  
));
```

Compile time predicate

Equality:

```
template<class T, class U>
struct IsEqual {
    enum { value = 0 };
};
```

```
template<class T>
struct IsEqual<T, T> {
    enum { value = 1 };
};
```

Compile time predicate

```
template<class T, class U>
struct IsDerived {
private:
    struct Yes { int __dummy__; };
    struct No { };
    static Yes convert(const U *);
    static No convert(const void *);
public:
    enum { value =
        (sizeof(convert((T *)0)) == sizeof(Yes))
    };
};
```

Compile time predicate

Realmente e' possibile utilizzare le implementazioni della libreria type traits di boost per i predicati sopra descritti:

- `is_base_and_derived`
- `is_same`

Compile time predicate

Ad esempio, possiamo realizzare un predicato che ritorna true se un tipo e' POD (plain old data):

```
template<class T>
struct IsPod {
    enum { value =
        IsEqual<T, bool>::value ||
        IsEqual<T, char>::value ||
        ...
        IsDerived<T, POD>::value
    };
};
```

Container

Sono definiti ricorsivamente:

```
struct NullType { };
```

```
template<class T, class U>  
struct TypeList {  
    typedef T Type;  
    typedef U Next;  
};
```

Container

```
typedef TypeList<
    int, TypeList<
        bool, TypeList<
            char, TypeList<
                float, NullType
            >
        >
    >
> SomeTypes;
```

Container

Utilizzando la ricorsione compile time, si possono realizzare funzioni che iterano:

```
template<
  class TypeList,
  class Type,
  int TypePosition = 0,
  int Test = boost::is_same<
    Type,
    typename TypeList::Type
  >::value
> struct Position;
```

Container

```
template<
  class TypeList,
  class Type,
  int TypePosition
> struct Position<TypeList, Type,
  TypePosition, 0> {
  enum { value = Position<
    typename TypeList::Next,
    Type,
    TypePosition + 1>::value };
};
```

Container

```
template<
  class TypeList,
  class Type,
  int TypePosition
> struct Position<TypeList, Type,
  TypePosition, 1> {
  enum { value = TypePosition };
};
```

Container

Nel mondo reale, si usa il *template metaprogramming framework* di boost:

- `boost::mpl::vector`
- `boost::mpl::find_if`
- `boost::mpl::distance`

Esempio: copy

Spesso e' possibile specializzare un'azione specifica per un tipo, ad esempio:

```
template<
    class T,
    bool J = IsPod<T>::value
> struct MyTraits;
```

Esempio: copy

```
template<class T>
struct MyTraits<T, false> {
    static void copy(T *b, T *e, T *d) {
        while(b != e) *d++ = *b++;
    }
};
```

```
template<class T>
struct MyTraits<T, true> {
    static void copy(T *b, T *e, T *d) {
        memcpy(d, b, sizeof(T) * (e - b));
    }
};
```

Esempio: copy

Utilizzando la copy iterativa:

real	0m2.909s
user	0m1.216s
sys	0m1.692s

Utilizzando la copy veloce:

real	0m1.962s
user	0m0.136s
sys	0m1.824s

Esempio: albero espressione

```
struct Image {
    uint8_t *m_data; // Image data
    unsigned m_w, m_h; // Width, height
    ...

    Image operator+(const Image &t) const
    {
        Image r(m_w, m_h);
        for(unsigned i = 0; i < m_w * m_h; ++i) {
            r.m_data[i] = m_data[i] + t.m_data[i];
        }
        return r;
    }
};
```

Esempio: albero espressione

```
int main() {  
    Image a(4096, 4096);  
    Image b(4096, 4096);  
    Image c(4096, 4096);  
    Image d(4096, 4096);  
    uint8_t s(128);  
  
    a = ((b + c) * s + d) + a;  
}
```

Esempio: albero espressione

Le performance sono basse perche' vengono generati piu' temporanei:

- $(b + c)$
- $(b + c) * s$
- $(b + c) * s + d$
- $((b + c) * s + d) + a$

Esempio: albero espressione

```
template<class T, class U>
struct Sum {
    const T &m_t;
    const U &m_u;

    template<class J>
    inline Sum<Sum, J>
    operator+(const J &t) const {
        return Sum<Sum, J>(*this, t);
    }
}
```

Esempio: albero espressione

```
inline uint8_t execute(  
    const unsigned i  
) const {  
    return m_t.execute(i) +  
           m_u.execute(i);  
};
```

```
Sum(const T &t, const U &u)  
: m_t(t), m_u(u)  
{ }  
};
```

Esempio: albero espressione

```
struct Image {
    uint8_t *m_data;
    unsigned m_w, m_h;

    uint8_t execute(const uint i) const {
        return m_data[i];
    }
};

template<class T>
Sum<Image, T>
operator+(const T &t) const {
    return Sum<Image, T>(*this, t);
}
```

Esempio: albero espressione

Utilizzando la versione semplice:

real	0m1.781s
user	0m1.244s
sys	0m0.540s

Utilizzando la versione metaprogrammata:

real	0m0.754s
user	0m0.608s
sys	0m0.144s