template++

Pataki Norbert

Programozási Nyelvek és
Fordítóprogramok Tanszék

Programozási Nyelvek (C++)

Konténer adaptorok

- Szekvenciális konténerek 
- Asszociatív konténerek 
- Konténer adaptorok:
  - `std::stack`
  - `std::queue`
  - `std::priority_queue`

Nem önálló konténerek.

Példák

```
#include <stack>
#include <iostream>

int main()
{
  std::stack<int> s;
  s.push( 4 );
  s.push( 7 );

  std::cout << s.top() << std::endl; // 7
  s.pop();
  std::cout << s.top() << std::endl; // 4
}
```

Példák

```
#include <queue>
#include <iostream>

int main()
{
  std::queue<int> q;
  q.push( 4 );
  q.push( 7 );
  std::cout << q.front(); // 4
  q.pop();
  std::cout << q.front(); // 7
}
```

Koncepció

```
template <class T, class Cont = std::deque<T> >
class queue
{

  Cont c;

// ...

};
```

Konténerek iterátorai

- `iterator`
- `const_iterator`
- `reverse_iterator`
- `const_reverse_iterator`

Példák

```
std::vector<int> v;
v.push_back( 7 );
v.push_back( 3 );
std::vector<int>::iterator i = v.begin();
*i = 8;
std::vector<int>::const_iterator ci = v.begin();
std::cout << *ci << std::endl;
```

Példák

```
std::set<int> s;
s.insert( 7 );
s.insert( 3 );
s.insert( 4 );
std::set<int>::iterator i = s.begin();
std::set<int>::reverse_iterator ri = s.rbegin();
std::cout << *i  << *ri << std::endl; // 37
```

const_iterator

```
template <class T>
class Foo
{
  std::list<T> s;

public:

  void bar() const
  {
    for( typename std::list<T>::const_iterator i = s.begin();
         i != s.end();
         ++i )
    {
      // ...
    }
  }
};
```

typename

```
int i;

template <class T>
class X
{
  // ....
  void f()
  {
    Y<T>::N * i;
    // ...
  }
};
```

```
int i;

template <class T>
class X
{
  // ....
  void f()
  {
    typename Y<T>::N * i;
    // ...
  }
};
```

Unruh, 1994 (részlet)

```
<source>: In member function 'void Prime_print<i>::f() [with int i = 13]':
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 14]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 15]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 16]'
<source>:56:   instantiated from here
<source>:36: error: invalid conversion from 'int' to 'void*'
<source>:36: error:   initializing argument 1 of 'D<i>::D(void*) [with int i =
<source>: In member function 'void Prime_print<i>::f() [with int i = 11]':
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 12]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 13]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 14]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 15]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 16]'
<source>:56:   instantiated from here
<source>:36: error: invalid conversion from 'int' to 'void*'
<source>:36: error:   initializing argument 1 of 'D<i>::D(void*) [with int i =
<source>: In member function 'void Prime_print<i>::f() [with int i = 7]':
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 8]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 9]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 10]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 11]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 12]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 13]'
<source>:37:   instantiated from 'void Prime_print<i>::f() [with int i = 14]'
```

Fogalmak

- Sablonok, template-ek
- Példányosítás fordítási időben
- Template paraméterek
- Specializációk
- Template metaprogramozás

Specializációk

```
template <class T>
class Matrix
{
  // ...
};

template <class T>
class Matrix<T*>
{
  // ...
};

template <>
class Matrix<bool>
{
  // ...
};
```

Általános változat

Parciális specializáció

Teljes specializáció

Faktoriális számítás

```
template <int N>
struct Factorial
{
  static const int value = N * Factorial<N - 1>::value;
};

template <>
struct Factorial<0>
{
  static const int value = 1;
};

int main()
{
  int i = Factorial<5>::value;
}
```

Generált assembly (részlet)

```
$ g++ -W -Wall -pedantic -ansi -save-temps fac.cpp
$ cat fac.s

main:
.LFB0:
    .cfi_startproc
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
    .cfi_def_cfa_register 6
    movl $120, -4(%rbp)
    movl $0, %eax
    popq %rbp
```

Faktoriális számítás

```
template <int N>
struct Factorial
{
  enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>
{
  enum { value = 1 };
};
```

Stack overflow

```
// so.cpp:
int fac( int n )
{
  return n * fac( n - 1 );
}

int main()
{
  fac( 3 );
}
```

```
$ g++ -W -Wall -pedantic -ansi so.cpp
$ ./a.out
Segmentation fault (core dumped)
```

Végtelen rekurzió

```
template <int N>
struct Factorial
{
  static const int value = N * Factorial<N - 1>::value;
};

int main()
{
  int i = Factorial<5>::value;
}
```

Végtelen rekurzió

```
fac.cpp: In instantiation of 'const int Factorial<-894>::value':
fac.cpp:5:30:   recursively required from 'const int Factorial<4>::value'
fac.cpp:5:30:   required from 'const int Factorial<5>::value'
fac.cpp:11:25:   required from here
fac.cpp:5:30: fatal error: template instantiation depth exceeds maximum of 900 (use -ftemplate-depth= to increase the maximum)
   static const int value = N * Factorial<N - 1>::value;
                            ^
compilation terminated.
```

Fordítási ellenőrzések javítása

```
template <class T>
class Vector
{
  int cap, size;
  T* p;

public:

  Vector( int i )
  {
    cap = i;
    p = new T[ cap ];
  }
  //...
};
```

```
Vector<int> v( -5 );
```

Fordítási ellenőrzések javítása

```
template <class T>
class Vector
{
  int cap, size;
  T* p;

public:
  Vector( unsigned int i )
  {
    cap = i;
    p = new T[ cap ];
  }
  //...
};
```

```
Vector<int> v( -5 );
```

Kis átalakítás után

```
template <class T,
          int N = 16>
class Vector
{
  //...

public:

  Vector()
  {
    cap = N;
    p = new T[ cap ];
  }
};
```

```
Vector<int, -5> v;
```

Validáció

```
template <int N>
class Check
{
  char v[ N ];
};
```

```
error:  size of array is negative
```

```
warning:  ISO C++ forbids zero-size array
[-Wpedantic]
```

Validáció

```
template <class T, bool b>
struct ERROR_if_not { };

template <class T>
struct ERROR_if_not<T, false>
{
  ERROR_if_not() { T().invalid_allocation(); }
};

template <int N>
struct Check
{
  ERROR_if_not<int, ( N > 0 ) > ___;
};
```

Fordítási hiba

```
template <class T,
          int N = 16>
class Vector
{

  Vector()
  {
    Check<N>();
    cap = N;
    p = new T[ N ];
  }
  // ...
};
```

```
Vector<int, -5> v;
```

```
a.cpp: In instantiation of 'ERROR_if_not<T, false>::ERROR_if_not() [with T = int]':
a.cpp:17:8:   required from 'Vector<T, N>::Vector() [with T = int; int N = -5]'
a.cpp:44:19:   required from here
a.cpp:12:20: error: request for member 'invalid_allocation' in '0', which is of non-class type 'int'
   ERROR_if_not() { T().invalid_allocation(); }
```

Emlékeztető

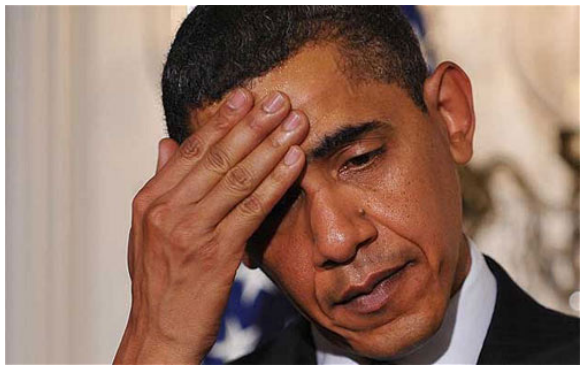
```
template <class T>
const T& max( const T& a, const T& b )
{
  return a < b ? b : a;
}

int main()
{
  std::cout << max( 3.5, 25 );
}
```

```
max.cpp: In function 'int main()':
max.cpp:11:29: error: no matching function for call to 'max(double, int)'
   std::cout << max( 3.5, 25 );
                             ^
max.cpp:4:10: note: candidate: template<class T> const T& max(const T&, const T&)
 const T& max( const T& a, const T& b )
          ^
max.cpp:4:10: note:   template argument deduction/substitution failed:
max.cpp:11:29: note:   deduced conflicting types for parameter 'const T' ('double' and 'int')
   std::cout << max( 3.5, 25 );
                             ^
```

A `max` sablon

```
template <class T, class U>
"T xor U" max( const T& t, const U& u )
{
  // ...
}
```

`T` xor `U`?



Elágazás fordítási időben

```
template <bool cond, class T, class F>
struct If
{
  typedef T Ret;
};

template <class T, class F>
struct If<false, T, F>
{
  typedef F Ret;
};
```

A `max` sablon

```
template <class T, class U>
typename If<sizeof( T ) < sizeof( U ), U, T>::Ret
  max( const T& t, const U& u )
{
  return t < u ? u : t;
}
```

Adatszerkezetek fordítási időben

```
class NullType { };

template <class Head, class Tail>
struct Typelist
{
};

typedef
  Typelist<int,
           Typelist<double,
             Typelist<long, NullType> > >
  types;
```

A `Typelist` elemszáma

```
template <class T>
struct Size;

template <class Head, class Tail>
struct Size<Typelist<Head, Tail> >
{
  static const int value = 1 + Size<Tail>::value;
};

template <>
struct Size<NullType>
{
  static const int value = 0;
};

//...
std::cout << Size<types>::value;
```

Template metaprogramozás tulajdonságai

- Programozási nyelv a programozási nyelvben
- Fordítás idejű számítások
- Turing-teljes
- Funkcionális megközelítés
- Limitációk

Template metaprogramozás alkalmazásai

- Fordítási hibák előállítása prímszámokkal
- Adaptív könyvtárak
- Futási idő hatékonyságának javítása:
  - Érdemi számítások fordítási időben
  - Expression template-ek
  - Másolások optimalizációja
- DSL beágyazások
- Fordítási ellenőrzések
- C++11: `constexpr`, `static_assert`