C++ Standard Template Library (STL)

Pataki Norbert

Programozási Nyelvek és
Fordítóprogramok Tanszék

Programozási Nyelvek I.

Témák

C++ STL

- Ne fedezzük fel újra spanyolviaszt!
- Sok adatszerkezet/algoritmus implementált a könyvtárban
- Hatékonyság
- C-s alapok + C++ operátor túlterhelések + C++ template-ek
- További lehetőségek a template-ekkel kapcsolatban

C: find

```
int* find( int* first, int* last, int i )
{
  while( first != last )
  {
    if ( *first == i )
    {
      return first;
    }
    ++first;
  }
  return 0;
}
```

Template? Template :-)

```
template <class T>
T* find( T* first, T* last, const T& t )
{
  while( first != last )
  {
    if ( *first == t )
    {
      return first;
    }
    ++first;
  }
  return 0;
}
```

Még több template paraméter? Nyilván :-)

```
template <class It, class T>
It find( It first, It last, const T& t )
{
  while( first != last )
  {
    if ( *first == t )
    {
      return first;
    }
    ++first;
  }
  return last;
}
```

STL alapfogalmak

- Szabvány és implementációk (HP STL, SGI STL, Dinkumware, stb.)
- Aszimptotikus műveletigény garanciák
- Szabványos azonosítók
- Online referenciák
- Fő komponensek:
  - Konténerek (pl. `vector`)
  - Algoritmusok (pl. `for_each`)
  - Iterátorok (pl. `list<int>::iterator`)
  - Funktorok (Felhasználói osztályok `operator()`-ral.)
  - Stb.

Az STL bővítése

- Új algoritmusok: működnek a meglévő konténerekkel
- Új konténerek: működnek a meglévő algoritmusokkal
- Nem kell módosítani a könyvtár kódját
- Párhuzamos bővítés, OOP limitációk

Fejállományok

- Lényegében header-only implementáció
- Algoritmusok:
  - `#include <algorithm>`
  - `#include <numeric>`
- Konténerek:
  - `#include <vector>`
  - `#include <list>`
  - `#include <deque>`
  - `#include <string>`
  - `#include <set>`
  - `#include <map>`
- Hasznos holmik:
  - `#include <functional>`
  - `#include <utility>`

Fejállományok

Include-oljuk azt, amelyikre ténylegesen szükségünk van!

Ne include-oljuk feleslegesen!

Include-oljuk azt, amelyikre ténylegesen szükségünk van!
Akkor is, ha lefordul anélkül...

Hordozhatósági problémák

Konténerek az STL-ben

- Szekvenciális konténerek:
  - `vector`
  - `list`
  - `deque`
  - `string`
- Asszociatív konténerek:
  - `set`, `multiset`
  - `map`, `multimap`

Szekvenciális konténerek

- Eltérő memória felhasználás
- Hasonló műveletek:
  - `size`
  - `push_back`
  - `pop_back`
  - `push_front`
  - `pop_front`
  - indexelés
  - `insert`
  - stb.
- Jelentős különbségek

Asszociatív konténerek

- Rendezettség
- Jellemzően piros-fekete fák
- Műveletigények
- Speciális tagfüggvények
- Ekvivalencia, egyenlőség

Példa – map

```
#include <iostream>
#include <map>
#include <string>

int main()
{
  std::map<std::string, std::string> phones;
  phones[ "Kiss Bertalan" ] = "555-1234";
  phones[ "Nagy Kunigunda" ] = "555-6666";

  std::cout << phones[ "Kiss Bertalan" ]
            << std::endl;
  std::cout << phones[ "Unknown" ] << std::endl;
  std::cout << phones.size() << std::endl;
}
```

Rendezések az STL-ben

```
std::set<int> si;
si.insert( 7 );
si.insert( 2 );
si.insert( 8 );
si.insert( 3 );

std::copy( si.begin(),
           si.end(),
           std::ostream_iterator<int>( std::cout,
                                       " " ) );
// 2 3 7 8
```

Példa – multiset

```
class Employee
{

  std::string name;
  int salary;

public:

  //...

  const std::string& get_name() const
  {
    return name;
  }
};
```

Rendezések az STL-ben

```
struct EmployeeComp
{

  bool operator()( const Employee& lhs,
                   const Employee& rhs ) const
  {
    return lhs.get_name() < rhs.get_name();
  }

};
```

Rendezések az STL-ben

```
std::multiset<Employee, EmployeeComp> employees;
Employee e(...);

employees.insert( e );

std::multiset<Employee, EmployeeComp>::iterator i =
  employees.begin();
std::cout << i->get_name() << std::endl;
```

Alapértelmezett rendezések megvalósítása

```
#include <functional>
// ...

template <class Key,
          class Comp = std::less<Key>,
          /* ... */ >
class set
{
  // ...
};
```

Paraméterezett rendezések használata

```
template <class Key, class Comp = std::less<Key> >
class set
{
  // ...
  void foo()
  {
    Key a(...);
    Key b(...);
    if ( Comp()( a, b ) ) // "a < b"
    {
      // ...
    }
  }
};
```

Speciális konstruktorok

```
std::multiset<int> m(
  std::istream_iterator<int>( std::cin ),
  std::istream_iterator<int>( ) );

std::copy(
  m.begin(),
  m.end(),
  std::ostream_iterator<int>( std::cout, " " ) );

std::deque<int> d( m.begin(), m.end() );
```

Intervallum konstruktorok megvalósítása

```
template <class T, /*...*/>
class deque
{

  // ...

public:

  template <class InputIterator>
  deque( InputIterator first,
         InputIterator last )
  {
    // ...
  }
};
```

vector

```
std::vector<int> v;
v.push_back( 4 );
int *ip = &v[ 0 ];
```

vector

```
std::vector<bool> x;
x.push_back( false );
bool* p = &x[ 0 ];
```

Hibaüzenet

```
v.cpp: In function 'int main()':
v.cpp:7:19: error: taking address of temporary [-fpermissive]
   bool* p = &x[ 0 ];
                   ^
v.cpp:7:19: error: cannot convert 'std::vector<bool>::reference* {aka std::_Bit_reference*}'
to 'bool*' in initialization
```

A jelenség mögötti fogalom

- *Template specializáció* (template osztály):
  - Parciális specializáció
  - Teljes specializáció
- A `vector<bool>` egy önálló konténer
- Template metaprogramozás

Iterátorok

- C-ben: pointerek a tömbök bejárásához
- C++ STL: iterátorok a konténerek bejárásához
- Konténerek indexelés nélkül szintén bejárható (pl. `list`)
- Konténerek belső típusa
- `begin()` és `end()`
- Iterátor kategóriák a konténerek esetében
- Példák:

```
#include <list>
#include <vector>
//...
std::vector<int> v;
std::vector<int>::iterator i = v.begin();

std::list<double> d;
std::list<double>::iterator s = d.begin();
```

Példa – `find`

```
std::list<int> c;
int s;
// ...
std::list<int>::iterator i =
  std::find( c.begin(), c.end(), s );
if ( c.end() != i )
{
  std::cout << *i;
}
```

Példa – `find`

```
std::deque<int> d;
int x;
// ...
std::deque<int>::iterator i =
  std::find( d.begin(), d.end(), x );
if ( c.end() != i )
{
  std::cout << *i;
}
```

Konténerek bejárása

```
std::set<int> m;
// ...
for( std::set<int>::iterator i = m.begin();
     i != m.end();
     ++i )
{
  if ( 0 == *i % 2 )
  {
    std::cout << *i << ' ';
  }
}
```

Példa – map

```
#include <map>
// ...
typedef std::map<std::string, std::string> dict;
// ....
dict phones;
phones[ "Kiss Bertalan" ] = "555-1234";
// ...
for( dict::iterator it = phones.begin();
     it != phones.end();
     ++it )
{
  std::cout << it->first << ":"
            << it->second << std::endl;
}
```

Iterátor kategóriák

- Input iterator
- Output iterator
- Forward iterator
- Bidirectional iterator
- Random access iterator

Felhasználói kódrészletek

- Gyakran kell felhasználói kódrészletet átadni az STL-nek
- Rendezések (pl. `std::sort`, asszociatív konténerek)
- Keresések (pl. `std::find_if`, `count_if` algoritmusok)
- `std::for_each` algoritmus
- stb.
- Hatékonyság, paraméterezhetőség
- `operator()` – tetszőleges aritás

Példa

```
class Sum
{
  double ps;

public:

  Sum( double d = 0.0 ) : ps( d ) { }

  void operator()( double d )
  {
    ps += d;
  }
  double get() const
  {
    return ps;
  }
};
```

Példa

```
std::deque<double> d;
//...
std::cout <<
  std::for_each( d.begin(),
                 d.end(),
                 Sum() ).get();
```

Példa

```
std::set<double> s;
//...
std::cout <<
  std::for_each( s.begin(),
                 s.end(),
                 Sum( 3.14 ) ).get();
```