Bevezetés a C++ Standard Template Library-be

Pataki Norbert

2013. május 17.

STL

- STL vs. STL implementáció
- Konténerek
- Funktorok
- Algoritmusok
- Iterátorok

STL Konténerek

- Szekvenciális: `vector`, `list`, `deque`, `basic_string`
- Asszociatív: `set`, `multiset`, `map`, `multimap`
- Adapter: `stack`, `queue`, `priority_queue`
- C++11:
  - `unordered_set`, `unordered_multiset`, `unordered_map`, `unordered_multimap`
  - `forward_list`
  - `array`

Vector

- Egybefüggő tárterületen azonos típusú elemek sorozata
- Elemek indexelve (0-tól kezdve)
- Elemek elérése – gyors
- Elem beszúrása vector végére – gyors
- Elem beszúrása máshova – lassabban

Vector

```
#include <vector>
using namespace std;

vector<int> v;     // üres vector létrehozása
v.push_back( 4 ); // elem beszúrása vector végére
v.pop_back();      // utolsó elem törlése

// Ötelemű vector létrehozása, minden eleme 3.2
vector<double> vd( 5, 3.2 );
int s = vd.size();

vd[1] = 4.76;      // Elemek elérése
vd.back() = 4.17; // Utolsó elem elérése
vd.front() = 1.2; // Első elem elérése
```

list

- Elemek szétszórtan helyezkednek el
- Elemek nincsen indexelve
- i-edik elérése lassú
- Elem beszúrása bárhova – gyors
- Következő / előző elem érhető el gyorsan

list

```
#include <list>
using namespace std;

list<double> d;       // üres list létrehozása
d.push_front( 4.5 ); // elem beszúrása lista elejére
d.pop_front();        // utolsó elem törlése
d.push_back( 1.1 );  // elem beszúrása lista végére

list<double> vl( 3, 6.32 );
int s = vl.size();

d.back() = 4.17;    // Utolsó elem elérése
d.front() = 1.2;    // Első elem elérése
d.sort();           // Lista rendezése
d.remove( 1.2 );    // Adott értékek törlése
d.reverse();        // Lista megfordítása
```

deque

- Kettős végű sor
- Egybefüggő tárterületek szétszórtan helyezkednek el
- Elemek indexelve vannak
- Elemek elérése – gyors
- Elem beszúrása tároló elejére / végére – gyors
- Elem beszúrása közepére – lassabban

deque

```
#include <deque>
#include <string>
using namespace std;

deque<string> d;
d.push_front( "Hello" );
d.push_back( "World" );
d[0] = "Goodbye";
d.back() = "Cruel World";
d.pop_back();
d.pop_front();
```

set

- Elemek sorrendje: rendezettség
- Minden elem legfeljebb egyszer szerepelhet
- Műveletek kihasználják a rendezettséget: gyors keresés, gyors beszúrás, stb.

set

```
#include <set>
#include <cstdlib>
using namespace std;

srand( time( 0 ) );
set<int> nums;
while( nums.size() < 5 )
{
  nums.insert( (rand() % 90 ) + 1 );
}
```

multiset

- Elemek sorrendje: rendezettség
- Azonos elemek többször is szerepelhetnek
- Műveletek kihasználják a rendezettséget: gyors keresés, gyors beszúrás, stb.

multiset

```
#include <set>
using namespace std;

multiset<int> m;
m.insert( 3 );

int s = m.size();
int i = m.count( 3 );
```

map

- Asszociatív tömb: elemek indexelve
- Nem feltétlenül 0-tól kezdve
- Nem feltétlenül egymás utáni indexek
- Nem feltétlenül egészek
- Kulcs alapján rendezett tárolás

map

```
#include <map>
#include <string>
#include <iostream>
using namespace std;

map<string, string> phones;
phones["Bela"] = "36(20)555-1234";
phones["Lajos"] = "36(30)555-5555;
// ...
cout << phones["Bela"];
```

Rendezés megadása

```
class Student
{
  std::string name;
  int height;
  double avg;
public:
  std::string get_name() const
  {
    return name;
  }
  int get_height() const
  {
    return height;
  }
  double get_average() const
  {
    return avg;
  }
};
```

Rendezés megadása

```
struct NameComparator
{
  bool operator()( const Student& a,
                   const Student& b ) const
  {
    return a.get_name() < b.get_name();
  }
};

std::set<Student, NameComparator> a;
```

Rendezés megadása

```
struct HeightComparator
{
  bool operator()( const Student& a,
                   const Student& b ) const
  {
    return a.get_height() < b.get_height();
  }
};

std::set<Student, HeightComparator> b;
```

Rendezés megadása

```
struct AverageComparator
{
  bool operator()( const Student& a,
                   const Student& b ) const
  {
    return a.get_average() < b.get_average();
  }
};

std::set<Student, AverageComparator> c;
```

STL Algoritmusok

- Globális műveletek
- Konténer-függetlenek
- Nem biztos, hogy egy algoritmus az összes konténerrel működik

Algoritmusok

```
#include <algorithm>
using namespace std;
// ...

vector<int> v;
list<double> s;
multiset<char> m;

int i = count( v.begin(), v.end(), 2 );
int j = count( s.begin(), s.end(), 6.5 );
int k = count( m.begin(), m.end(), 'T' );
```

Algoritmusok – folyt. köv.

```
#include <algorithm>
using namespace std;

bool even( int x )
{
  return 0 == x % 2;
}

bool upper( char c )
{
  return c>='A' && c<='Z';
}

int a = count_if( v.begin(), v.end(), even );
int b = count_if( m.begin(), m.end(), upper );
```

Algoritmusok

```
void print( int i )
{
  cout << i << ' ';
}

for_each( v.begin(), v.end(), print );
```

Algoritmusok

```
#include <numeric>
#include <algorithm>

bool comp( int i, int j )
{
  return i > j;
}

vector<int> v;
// ...
int sum = accumulate( v.begin(), v.end(), 0 );
sort( v.begin(), v.end() );
sort( v.begin(), v.end(), comp );
```

Iterátorok

- Elemek elérése a konténerekben
- Összekapcsolja a konténereket és az algoritmusokat

Iterátorok

```
vector<int> v;
list<double> l;
set<int> s;
// ...

vector<int>::iterator vi = v.begin();
list<double>::iterator li = l.begin();
set<int>::iterator i = s.begin();

cout << *vi << ' ' << *li
     << ' ' << *si << endl;
```

Iterátorok

```
++vi; // következő elemre lép
vi++; // következő elemre lép
--vi; // előző elemre lép
vi--; // előző elemre lép
++li; // következő elemre lép
si++; // előző elemre lép
```

Konténerek bejárása

```
list<int> l;
set<double> s;
// ...
for( list<int>::iterator i = l.begin();
     i != l.end();
     ++i )
{
  cout << *i;
}

for( set<double>::iterator i = s.begin();
     i != s.end();
     ++i )
{
  cout << *i;
}
```

Konténerek bejárása

```
map<int, string> m;
// ...
for( map<int, string>::iterator i = m.begin();
     i != m.end();
     ++i)
{
  cout << i->first << ' ' << i->second;
}
```

Algoritmusok

```
list<int> l;
int x, y;
// ...
// y első előfordulásának megkeresése
list<int>::iterator i = find( l.begin(), l.end(), y );
l.erase( i );
i = find( l.begin(), l.end(), x );
if ( i != l.end() )
  *i = y;
```

Asszociatív konténerek

```
set<int> s;
int i;
// ...

set<int>::iterator it = s.find( i );
if ( it != s.end() )
{
  s.erase( it, s.end() );
}
```

Asszociatív konténerek

```
map<string, int> m;
string str;
// ...

map<string, int>::iterator it = m.find( str );
if ( it != l.end() )
{
  m.erase( m.begin(), it );
}

m.erase( "Hello" );
```

Iterátorok

const_iteratorok nem módosíthatják a konténereket:

```
vector<int> v;
...
vector<int>::const_iterator i = v.begin();
*i = 10; // ford. hiba.
```