STL

Elsődleges komponensek:
- Tárolók
- Algoritmusok
- Bejárók

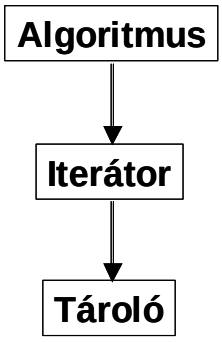
Másodlagos komponensek:
- Függvény objektumok
- Adapterek
- Allokátorok (helyfoglalók)
- Tulajdonságok

Tárolók:
- Vektor (vector)
- Lista (list)
- Halmaz (set)
- Kétirányú sor (deque)
- Stack (adapter)
- Queue (adapter)

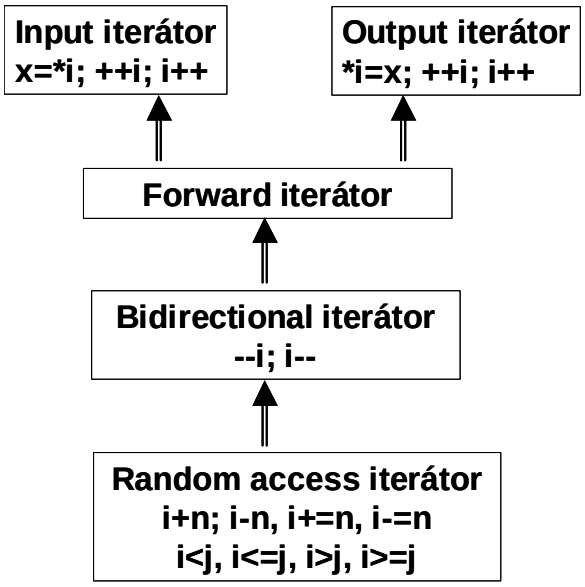
Algoritmus: másolás, keresés, hozzáadás, törlés, transzformálás, konvertálás

Bejáró típusok:
- író
- olvasó
- előre haladó
- kétirányú
- közvetlen elérésű

Komponensek kapcsolata



Bejáró (iterátor) típusok:



Vektor

```
namespace std {
   template < typename T, typename Allocator = allocator<T> >
   class vector {
      . . .
   }
}
```

Vektor használata

```
#include <vector>
  ...
vector<int> v1;        // üres vektor
vector<int> v2(5,-3); // 5 elemű vektor, minden eleme –3 értékű
vector<int> v3(5);     // 5 elemű vektor, az elemeknek nincs kezdeti értékük
vector<int> v4(v2);    // v2 vektorral inicializált új vektor
vector<int> v5(v2.begin()+1, v2.begin()+5);   // v2.begin()+5==v2.end()

vector<int>::iterator i;
for( i=v2.begin(); i != v2.end(); ++i) { cout<< *i; cout <<' '; }
int k=v2[3];
v2.front()=1;     // első elem átírása
v2.back()=5;      // utolsó elem átírása
v2.push_back(1); // új elem a vector végére
v2.pop_back();    // utolsó elem eltávolítása
```

Beszúrás

```
vector<int> v1(5,7);            // eredmény 77777
v1.insert(v1.begin()+1, 9);     //          797777
v1.insert(v1.begin()+3, 2, 8); //          79788777
```

Törlés

```
vector<int> v;
for( int k=1; k<9; k++) v.push_back(k); // 12345678
v.erase(v.begin()+5);                   // 1234578
```

Durva típushiba:

```
vector<int>(5,3);  // v.size()==5
vector<int>::iterator i = v.begin()+2;
v.erase(i);
*i = 9;  // katasztrófa
```

Kétirányú lista (list)

```
#include <list>
...

list<int> l1;       // üres lista
list<int> l2(5,-3); // 5 elemű lista, minden eleme -3 értékű
list<int> l3(5);    // 5 elemű lista, az elemeknek nincs kezdeti értékük
list<int> l4(l2);   // l2 listával inicializált új lista
list<int> l5(l2.begin(), l2.end());

list<int>::iterator i;
for( i=l2.begin(); i != l2.end(); ++i) { cout<< *i; cout <<' '; }
```

A listák létrehozása kísértetiesen hasonlít a vektorok létrehozásához. Különbség, hogy a bejáró nem közvetlen elérésű, így nem lehet hozzáadni egész számot.

További tagfüggvények :

push_back, pop_back, push_front, pop_front, front, back, insert, erase, clear, size

Kétirányú sor (deque)

Olyan sor, ami véletlen elérésű, mint a vektor, és mind a két végére azonos idő alatt lehet egy elemet feltenni, illetve törölni.

Adapter működése

```
namespace std {
   template< class T, class Container = deque<T> >
   class stack {
      public :
      typedef Container::valus_type value_type;
       . . .
      protected :
         Container c;
   public:
      explicit stack( const Container& = Container() );
      bool empty() const { return c.empty(); }
      size_type size() const { return c.size(); }
      value_type& top() { return c.back(); }
      const value_type& top()  const { return c.back(); }
      void push( const value_type& x ) { c.push_back(x); }
      void pop() { c.pop_back(); }
   };
}
```

Halmaz (set)

```
#include <set>
#include <iostream>

using namespace std;

class adat
{
   int k;
   int a;
public:
   adat(int arg1, int arg2 ) { k=arg1, a=arg2; }
   friend bool operator<( const adat& a, const adat& b )
   {
      return a.k < b.k;
   }
   friend operator<<( ostream& os, const adat& a )
   {
      os << "(" << a.k <<"," << a.a <<") ";
   }
};

int main() {
   set<adat> s; // üres set
   s.insert(adat(5,2));
   s.insert(adat(2,5));
   s.insert(adat(100,3));
   s.insert(adat(88,10));
   s.insert(adat(2,3));  // ezt nem teszi be mert már létezik
   set<adat>::iterator i, j;

   for( i=s.begin(); i != s.end(); ++i) { cout<< *i; cout <<' '; }
   cout << endl;
   if( (i = s.find(adat(2,0))) != s.end() ) cout << *i;
   else cout << "nem talalt";

   return 0;
}
```

Multiset esetén több azonos kulcsú elemet is fel lehet venni.

```
#include <set>
#include <iostream>

using namespace std;

int main()
{
   multiset<int> ms;
   multiset<int>::const_iterator l, u, i;

   int a[] = {7,3,4,1,7,7,9,0,8,7,9 };
   ms.insert(a, a+sizeof(a)/sizeof(int) );

   l = ms.lower_bound(7);
   u= ms.upper_bound(8);
   for( i=l; i != u; ++i) { cout << *i; cout <<" "; }

   pair< multiset<int>::const_iterator, multiset<int>::const_iterator> lu
       = ms.equal_range(9);
   for( i=lu.first; i != lu.second; ++i) { cout << *i; cout <<" "; }
   return 0;
}
```

Asszociatív tömb (map)

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

using namespace std;

int main()
{
   map<const string, unsigned int> telefonkonyv;
   map<const string, unsigned int>::iterator i;
   telefonkonyv["Kiss Gizi"] = 210078;
   telefonkonyv["Csonttoro Jani"] = 210578;
   telefonkonyv["Hoher Jozsi"] = 43078;

   i = telefonkonyv.find("Hoher Jozsi");
   if( i==telefonkonyv.end() ) cout << "nem talalt";
   else cout<< i->first << " " << i->second;
   return 0;
}
```

Algoritmus

```
namespace std
{
   template < class InIt, class Fun>
   Fun for_each(InIt first, InIt last, Fun f );
}
```

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

using namespace std;

class func {
   int db;
public:
   func() { db=0; }
   int get_db() const { return db; }
   void operator()( pair<const string, unsigned int>& a )
   {
      cout<< a.first << " " << a.second << endl;
      db++;
   }
};

int main()
{
   map<const string, unsigned int> telefonkonyv;
   telefonkonyv["Milo Viki"] = 210078;
   telefonkonyv["Csonttoro Jani"] = 210578;
   telefonkonyv["Hoher Jozsi"] = 43078;
   telefonkonyv["Madar"] = 578071;

   func f = for_each( telefonkonyv.begin(), telefonkonyv.end(), func() );
   cout <<f.get_db();
   return 0;
}
```

Vigyázz, a függvényobjektumot az algoritmus érték szerint veszi át!