Objektumorientált programozás

Objektumalapú programozás a C++ programozási nyelvben

A standard könyvtár algoritmusai

Darvay Zsolt

A standard könyvtár

17. A standard könyvtár felépítése
18. Adatfolyamok
19. Algoritmusok

19. Algoritmusok

```
sort(v.begin( ),
v.end( ));
```

Áttekintés

19.1. Osztályozás

19.2. Alkalmazás hagyományos tömbökre

19.3. Nem módosító szekvencia műveletek

19.1. Osztályozás

- Az <algorithm> fejállományban megadott algoritmusok
- Numerikus műveletek a <numeric> fejállományban
- Inicializálatlan memóriára vonatkozó műveletek a <memory> fejállományban
- C-ben megadott algoritmusok a <cstdlib> fejállományban. Sablonok nélküli rendezés és keresés: qsort és bsearch. Az elnevezés ellenére a szabvány nem írja elő, hogy gyors rendezés, illetve bináris keresés legyen.

Az <algorithm> fejállomány algoritmusai

- Nem módosító szekvencia műveletek
- Módosító szekvencia műveletek
- Particionálás és rendezés
- Műveletek rendezett tartományokon: bináris keresés, összefésülés (merge), halmazműveletek
- Kupacműveletek
- Egyéb: minimum/maximum, összehasonlítás, permutáció

Az algoritmusok meghívása

- Általában tartományokra vonatkoznak.
- Egy tartományt mindig balról zárt és jobbról nyílt intervallumként adunk meg. Iterátorokkal határozzuk meg az intervallumot.
- Például: [első, utolsó)
- C++ 20-tól: korlátozott (constrained) algoritmusok. Az std::range névtér részeként hívható meg.
- Végrehajtási irányelvek (execution policies). Például párhuzamos végrehajtás, C++ 17-től.

19.2. Alkalmazás hagyományos tömbökre

- Egy objektumra hivatkozó mutatót közvetlen elérésű bejáróként használhatunk.
- Például a sort algoritmussal egy megadott tartományt rendezhetünk. A tartományt meghatározó iterátorok közvetlen elérésűek kell legyenek.

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

Hagyományos tömb rendezése

```
int main() {
    int t[] = { 9,2,7,1,5,4,3 };
    int n = 7;
    sort(t, t + n);
    cout << "Rendezve:";
    for (int i = 0; i < n; ++i)
          cout << " " << t[i];
    cout << endl;
}
```

Kimenet:
Rendezve: 1 2 3 4 5 7 9

19.3. Nem módosító szekvencia műveletek

all_of (C++11): igazat térít vissza, ha az adott predikátum igaz a tartománybeli összes elemre

any_of (C++11): igazat térít vissza, ha az adott predikátum igaz a tartomány legalább egy elemére

none_of (C++11): igazat térít vissza, ha az adott predikátum nem teljesül egyetlen tartománybeli elemre sem

for_each: az összes elemre elvégez egy műveletet (alkalmaz egy függvényt vagy függvényobjektumot)

for_each_n (C++17): az első n elemre alkalmazza a függvényobjektumot

Nem módosító szekvencia műveletek

count: az adott elem előfordulásainak a száma

count_if: azoknak az elemeknek a száma, amelyekre teljesül a feltétel

mismatch: az első különböző elempár két tartományból

find: egy elem keresése (első előfordulás)

find_if: egy feltételtől függő keresés

find_if_not (C++11): az első elem keresése, amely nem teljesíti a feltételt

find_end: egy tartomány utolsó előfordulása egy másik tartományban

Nem módosító szekvencia műveletek

find_first_of: megkeresi az első tartományban azt az első elemet, amely benne van a második tartományban

adjacent_find: megkeresi azt a két egymásutáni elemet, amelyek egyenlőek (vagy teljesítenek egy predikátum által megadott feltételt)

search: egy tartomány első előfordulása egy másik tartományban

search_n: ugyanazzal az értékkel rendelkező n hosszúságú részsorozat első előfordulása egy tartományban

Példa: count (a vektor)

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.push_back(10);
    v.push_back(30);
    v.push_back(10);
    v.push_back(50);
    v.push_back(10);
```

- A count és a count_if algoritmus esetén a tartományt bemeneti iterátorok segítségével kell megadni.
- A count_if esetén a feltételt függvényobjektummal határozzuk meg.

Példa: count (az eredmény)

```
  cout << "v =";
  for (vector<int>::iterator i = v.begin(); i != v.end(); ++i)
      cout << " " << *i;
  cout << endl;

  int result = count(v.begin(), v.end(), 10);
  cout << "A 10 - esek szama: " << result << endl;
}
// v = 10 30 10 50 10
// A 10 - esek szama : 3
```

Példa: count_if (feltétel)

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
// hagyományos függvény
bool nagyobb_mint_5(int x)
{
   return x > 5;
}
```

Példa: count_if (kiírás)

```
void kiir(string s, vector<int>& v)
{
  cout << s;
  for (vector<int>::iterator i = v.begin(); i != v.end(); ++i)
      cout << " " << *i;
  cout << endl;
}
```

Példa: count_if (eredmény)

```
int main()
{
    vector<int> v{ 9,2,6,3,5,4,7,0,8 };
    kiir("v =", v);
    int r = count_if(v.begin(), v.end(), nagyobb_mint_5);
    cout << "Az 5 - nel nagyobbak szama: " << r << endl;
}
// v = 9 2 6 3 5 4 7 0 8
// Az 5 - nel nagyobbak szama : 4
```

Függvényobjektumot meghatározó osztály

```
class Paros {
public:
   bool operator()(int x)
   {
        return x % 2 == 0;
   }
};
```

A count_if algoritmus függvényobjektummal

```
int main()
{
    vector<int> v{ 9,2,6,3,5,4,7,0,8 };
    kiir("v =", v);
    int r = count_if(v.begin(), v.end(), Paros());
    cout << "Parosak szama: " << r << endl;
}
// v = 9 2 6 3 5 4 7 0 8
// Parosak szama : 5
```

Példa: find_end (feltétel)

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <list>
using namespace std;
bool ketszeres(int a, int b)
{
   return 2 * a == b;
}
```

- A find_end algoritmus mindkét tartományát előrehaladó bejáróval kell megadni.
- Ha egy feltétel szerint történik az összehasonlítás, akkor függvényobjektumot használunk.

Példa: find_end (kiírás)

```
template <typename T>
void kiir(string s, T& v)
{
  cout << s;
  for (typename T::iterator i = v.begin(); i != v.end(); ++i)
      cout << " " << *i;
  cout << endl;
}
```

Meg kell adni, hogy T egy típusnév

Adatszerkezetek létrehozása

```
int main()
{
    vector <int> v1{ 4,8,12,16,20,4,8,12,16,20 };
    vector <int> v2{ 8,16,24 };
    list <int> w{ 8,12,16 };

    kiir("v1 =", v1);
    kiir("w =", w);
    kiir("v2 =", v2);
```

A find_end meghívása

```
vector <int>::iterator r;
r = find_end(v1.begin(), v1.end(), w.begin(), w.end());

if (r == v1.end())
    cout << "Nem talalhato meg w a v1-ben" << endl;
else
    cout << "A w utolso elofordulasanak pozicioja v1-ben: "
         << r - v1.begin() << endl;
```

A ketszeres függvénnyel

```
vector <int>::iterator p;
p = find_end(v1.begin(), v1.end(),
                    v2.begin(), v2.end(), ketszeres);
if (p == v1.end())
   cout << "Nem talalhato meg v2 a v1-ben" << endl;
else
   cout << "A v2 a v1 reszsorozatanak ketszerese.\n"
        << "Pozicio v1-ben (utolso elofordulas): "
        << p - v1.begin() << endl;
}
```

Kimenet

v1 = 4 8 12 16 20 4 8 12 16 20

w = 8 12 16

v2 = 8 16 24

A w utolso elofordulasanak pozicioja v1 - ben: 6

A v2 a v1 reszsorozatanak ketszerese.

Pozicio v1 - ben(utolso elofordulas) : 5

Példa: find_first_of

- A find_first_of algoritmus tartományait előrehaladó bejáróként adjuk meg.
- Ha szükség van összehasonlításra, akkor fügvényobjektummal dolgozhatunk.

```
#include <iostream>
#include <algorithm>
#include <deque>
#include <list>
#include <functional> // a greater predikátumhoz
using namespace std;
```

Függvényobjektumot meghatározó osztály

```
template <typename T, int M>
class Tobbszoros {
public:
    bool operator()(T a, T b)
    {
        return M * a == b;
    }
};
```

Példa: find_first_of (kiírás)

```
template <typename T>
void kiir(string s, T& v)
{
  cout << s;
  for (typename T::iterator i = begin(v); i != end(v); ++i)
      cout << " " << *i;
  cout << endl;
}
```

A begin és end függvénysablonok ebben a formában a C++11-től kezdődően használhatóak.

Adatszerkezetek

```
int main()
{
    deque <int> q{ 4,8,12,16,20,4,8,12,16,20 };
    list <int> w{ 7,33,5,19,12,80,11 };

    kiir("q =", q);
    kiir("w =", w);
```

Függvényobjektum nélkül

```
deque <int>::iterator r;
r = find_first_of(q.begin(), q.end(), w.begin(), w.end());

if (r == q.end())
    cout << "Nem talalhato meg w egyetlen eleme sem q-ban."
         << endl;
else
    cout << "Az elso w-beli elem pozicioja q-ban: "
        << r - q.begin() << endl;
```

Szabványos függvényobjektummal

```
deque <int>::iterator z;
z = find_first_of(q.begin(), q.end(), w.begin(), w.end(),
    greater<int>());
if (z == q.end())
    cout << "A q egyetlen eleme sem nagyobb egy w-belinel."
         << endl;
else
    cout << "A q-ban van olyan elem, "
         << "amely nagyobb egy w-belinel.\n"
         << "Pozicio q-ban (elso elofordulas): "
         << z - q.begin() << endl;
```

Saját függvényobjektummal

```
deque <int>::iterator p;
p = find_first_of(q.begin(), q.end(), w.begin(), w.end(),
      Tobbszoros<int, 5>());
if (p == q.end())
      cout << "A q egyetlen elemenek 5-szorose sincs w-ben."
           << endl;
else
      cout << "A q-ban van olyan elem, "
           << "amelynek 5-szorose w-ben van.\n"
           << "Pozicio q-ban (elso elofordulas): "
          << p - q.begin() << endl;
}
```

Kimenet

q = 4 8 12 16 20 4 8 12 16 20

w = 7 33 5 19 12 80 11

Az elso w-beli elem pozicioja q-ban: 2

A q-ban van olyan elem, amely nagyobb egy w-belinel.

Pozicio q-ban (elso elofordulas): 1

A q-ban van olyan elem, amelynek 5-szorose w-ben van.

Pozicio q-ban (elso elofordulas): 3

Példa: for_each

- A for_each algoritmus esetén a tartományt bemeneti bejáróval határozzuk meg.
- A megadott függvényobjektumot alkalmazzuk minden elemre.
- A visszatérített érték maga a függvényobjektum lesz, amelynek a tartalma idő közben változhatott.
- C++17-től kezdődően egy végrehajtási irányelvet is megadhatunk. Ebben az esetben nincs visszatérített érték.

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

A Szorzas predikátum

```
template <typename T>
class Szorzas {
private:
    T szorzo;
public:
    Szorzas(const T& sz) : szorzo(sz) {
    }
    void operator () (T& elem) const {
        elem *= szorzo;
    }
};
```

Az Atlag predikátum

```
class Atlag {
private:
    int elemszam;
    int osszeg;
public:
    Atlag() : elemszam(0), osszeg(0)
    {
    }
    void operator () (int elem);
    operator double();
};
```

Az Atlag osztály operátorai

```
void Atlag::operator () (int elem) {
    elemszam++;
    osszeg += elem;
}

Atlag::operator double() {
    return  static_cast <double> (osszeg) /
            static_cast <double> (elemszam);
}
```

Kiírást megvalósító operátor

```
template <typename T>
ostream& operator << (ostream& s, vector <T>& v)
{
   for (typename vector <T>::iterator i = begin(v);
                  i != end(v); ++i)
        s << " " << *i;
   return s;
}
```

A for_each meghívása

```
int main()
{
    vector <int> v{ 1, 2, 3, 4 };
    cout << "v =" << v << endl;
    for_each(v.begin(), v.end(), Szorzas<int>(2));
    cout << "Megszorozva 2-vel: " << endl;
    cout << "v =" << v << endl;
    double atlag = for_each(v.begin(), v.end(), Atlag());
    cout << "atlag = " << atlag << endl;
}
```

Kimenet:

```
v = 1 2 3 4
Megszorozva 2-vel:
v = 2 4 6 8
atlag = 5
```