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

Példa: a find és find_if algoritmusok

19.4. Módosító szekvencia műveletek

Az std névtér begin és end függvényei (C++11-től)

- Egy vektor tároló v objektuma esetén a v.begin(), illetve v.end() határozza meg az első elemre, illetve az utolsó utánira hivatkozó bejárót.
- Egy n elemű t hagyományos tömb esetén a t, illetve t + n határozza meg az első, illetve utolsó utáni pozícióra hivatkozó mutatót.
- A két változat egységesíthető az std::begin, illetve std::end függvényekkel.

Az array tároló

- A szabványos könyvtár tárolói általában dinamikusak (futási időben változtatható a tárolóban elhelyezett elemek száma).
- C++11-től bevezetésre került az array tároló, amely statikus (az elemek száma rögzített).

Példa: std::begin, std::end

```
#include <array>
#include <vector>
#include <iterator>
#include <iostream>

using namespace std;
```

A begin és end meghívása

```
template <typename T>
void kiir(string s, const T& x)
{
   cout << s;
   cout << " elso: " << *begin(x);
   cout << " utolso: " << *(end(x) - 1) << endl;
   cout << "tipus: " << typeid(x).name() << endl;
}
```

A fő függvény

```
int main()
{
   int t[]{ 1, 2, 3, 4, 5 };
   array<int, 5> a{ 10, 20, 30, 40, 50 };
   vector<int> v{ 100, 200, 300, 400, 500 };
   kiir("hagyomanyos tomb", t);
   kiir("array", a);
   kiir("vector", v);
}
```

Kimenet

hagyomanyos tomb elso: 1 utolso: 5
tipus: int const [5]
array elso: 10 utolso: 50
tipus: class std::array<int,5>
vector elso: 100 utolso: 500
tipus: class std::vector<int,class std::allocator<int> >

Példa: find, find_if

- Mindkét algoritmus esetén a tartományt bemeneti bejáróval adjuk meg.
- A find esetén a keresendő értéket, a find_if esetén pedig egy predikátumot kell megadni.
- A visszatérített érték egy bejáró arra a pozícióra, ahol megtaláltuk az elemet, vagy az utolsó utáni pozíció ha nincs találat.

Példa: find, find_if

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

using namespace std;
```

Kiírás – általánosított for

```
template <typename T>
void kiir(const string &s, const T& x) {
    cout << s;
    for (const auto& elem : x) {
        cout << " " << elem;
    }
    cout << endl;
}
```

A find meghívása

```
template <class BemenetiBejaro, class T>
void kiir_find(BemenetiBejaro elso, BemenetiBejaro utolso,
    const T& ertek) {
    auto it = find(elso, utolso, ertek);
    cout << ertek;
    if (it == utolso)
        cout << " nem talalhato." << endl;
    else
        cout << " megtalalhato." << endl;
}
```

A find_if meghívása

```
template <class BemenetiBejaro, class Predikatum>
void kiir_find_if(BemenetiBejaro elso, BemenetiBejaro utolso,
    Predikatum p, const string& s) {
    auto it = find_if(elso, utolso, p);
    if (it == utolso)
        cout << s << " nem talalhato." << endl;
    else
        cout << "Az elso " << s << ": " << *it << endl;
}
```

A paratlan függvény

```
bool paratlan(int i) {
   return (i % 2) != 0;
}
```

A vegrehajt függvény

```
template <typename T>
void vegrehajt(const string& s, const T& x)
{
    kiir(s, x);
    kiir_find(begin(x), end(x), 3);
    kiir_find(begin(x), end(x), 6);
    kiir_find_if(begin(x), end(x), paratlan, "paratlan");
    kiir_find_if(begin(x), end(x),
        [](int i) { return (i % 2) == 0; }, "paros");
}
```

A fő függvény

```
int main()
{
    const int t[]{ 1, 2, 3, 4, 5};
    vector<int> v{ 1, 2, 3, 4, 5};
    vegrehajt("t =", t);
    vegrehajt("v =", v);
}
```

Kimenet:

```
t = 1 2 3 4 5
3 megtalalhato.
6 nem talalhato.
Az elso paratlan: 1
Az elso paros: 2
v = 1 2 3 4 5
3 megtalalhato.
6 nem talalhato.
Az elso paratlan: 1
Az elso paros: 2
```

19.4. Módosító szekvencia műveletek

copy: másolás (az első elemtől a tároló vége felé haladva)

copy_if (C++11): egy adott predikátumot teljesítő elemeket másol

copy_n (C++11): adott számú elemet másol

copy_backward: másolás fordított irányban (az utolsó elemmel kezdjük)

move (C++11): egy adott tartomány áthelyezése

move_backward (C++11): áthelyezés fordított irányban

fill: tartomány feltöltése ugyanazzal az értékkel

fill_n: adott számú elem feltöltése

Módosító szekvencia műveletek

transform: egy függvényt alkalmaz egy tartományra és az eredményt egy céltartományban tárolja

generate: egy függvény egymásutáni meghívása által generált értékeket tárolja egy tartományban

generate_n: adott számú elemet generál

remove: egy adott értékkel egyenlő elemek eltávolítása

remove_if: egy adott feltételnek megfelelő elemek törlése

remove_copy: átmásolja az elemeket, kivéve azokat, amelyek egy adott értékkel egyeznek meg

remove_copy_if: átmásol, de figyelmen kívül hagyja a feltételt teljesítőket.

Módosító szekvencia műveletek

replace: a régi értékkel megegyező elemeket helyettesíti az új értékkel

replace_if: egy feltételtől függően helyettesít

replace_copy: másol és helyettesíti az adott elemeket

replace_copy_if: másol és feltételtől függően helyettesít

swap: két elemet felcserél

swap_ranges: két tartományt felcserél

iter_swap: bejáróval meghatározott elemeket cserél fel

reverse: tükröz egy tartományt

reverse_copy: tükrözve másolja le a tartományt

Módosító szekvencia műveletek

rotate: körbeforgatja az elemeket

rotate_copy: egy új tartományba másolja a körbeforgatott elemeket

shift_left (C++20): balra eltolás adott számú pozícióval

shift_right (C++20): jobbra eltolás adott számú pozícióval

random_shuffle (C++17-ig): véletlenszerűen összekeveri az elemeket

shuffle (C++11): véletlenszerűen összekeveri az elemeket

sample(C++17): adott számú véletlenszerűen kiválasztott elemet határoz meg

Módosító szekvencia műveletek

unique: törli az egymásutáni azonos elemeket a tartományból úgy, hogy minden azonos elemekből álló szekvencia helyett egyetlen elem marad; az összehasonlítási feltételt predikátummal meg lehet adni

unique_copy: az egymásutáni azonos elemeket egyetlen elemmel helyettesíti a másolatban

Példa: fill

- A fill algoritmus tartományát előrehaladó bejáróval adjuk meg.

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

Kiírás

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

A fill meghívása

```
int main()
{
    vector <int> v{ 1,2,3,4,5,6,7,8,9 };
    cout << "v =" << v << endl;
    // Az első 4 és utolsó 2 elemen kívüliek nullázása
    fill(v.begin() + 4, v.end() - 2, 0);
    cout << "v =" << v << endl;
}
    // v = 1 2 3 4 5 6 7 8 9
    // v = 1 2 3 4 0 0 0 8 9
```

Példa: generate

- A generate algoritmus tartományát előrehaladó bejáróval adjuk meg.
- A harmadik paraméter egy hagyományos függvény vagy függvényobjektum, amely a generálást végzi.
- A függvény üres paraméterlistával kell rendelkezzen és a visszatérített értékének típusa a tároló elemeinek típusára konvertálható kell legyen.

Példa: generate

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

A kiir függvény

```
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 typename T::iterator az auto kulcsszóval helyettesíthető

A begin(v), illetve end(v) is használható a v.begin(), illetve v.end() helyett.

Generáló függvény

```
int f() {
   static int x = 0;
   return ++x;
}
// az x statikus jellege miatt az egyes
// függvénymeghívások
// egymásutáni egész számokat térítenek vissza
```

A fő függvény

```
int main() {
    vector <int> v(5);
    deque <int> q(5);
    srand(unsigned(time(0)));
    generate(v.begin(), v.end(), rand);
    kiir("v =", v);
    generate(q.begin(), q.end(), f);
    kiir("q =", q);
}
```

Lehetséges kimenet:

```
v = 17883 29905 11917 4912 16051
q = 1 2 3 4 5
```

Példa: iter_swap

- Az iter_swap előrehaladó bejárókkal megadott elemeket cserél fel.
- A felcserélendő elemek nem kell feltétlenül ugyanannak a tárolónak a részét képezzék.

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

Az Egesz osztály

```
class Egesz {
   int ertek;
public:
   Egesz(int n) : ertek(n) {}
   ostream& kiir(ostream& s) const;
};
```

Kiíró operátor

```
ostream& Egesz::kiir(ostream& s) const
{
   return s << "Egesz(" << ertek << ")";
}

inline ostream& operator<<(ostream& s, const Egesz& x)
{
   return x.kiir(s);
}
```

A kiir függvény

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

A fő függvény

```
int main() {
    deque<Egesz> q{ 3,7,11 };
    kiir("q =", q);
    cout << "q elso es utolso elemet felcserelve:\n";
    iter_swap(q.begin(), q.end() - 1);
    kiir("q =", q);
    vector <Egesz> v{ 2,4,6,8,10,12 };
    kiir("v =", v);
    cout << "v harmadik es q masodik elemet felcserelve:\n";
    iter_swap(v.begin() + 2, q.begin() + 1);
    kiir("v =", v); kiir("q =", q);
}
```

Kimenet

q = Egesz(3) Egesz(7) Egesz(11)

q elso es utolso elemet felcserelve:

q = Egesz(11) Egesz(7) Egesz(3)

v = Egesz(2) Egesz(4) Egesz(6) Egesz(8) Egesz(10) Egesz(12)

v harmadik es q masodik elemet felcserelve:

v = Egesz(2) Egesz(4) Egesz(7) Egesz(8) Egesz(10) Egesz(12)

q = Egesz(11) Egesz(6) Egesz(3)

Példa: fill_n

- A fill_n első paramétere egy kimeneti bejáróval megadott pozíció, ahonnan kezdjük a feltöltést.
- Ezt követi a feltöltendő elemek száma és az érték.
- C++11-től kezdődően az utolsó feltöltött hely utáni pozícióra hivatkozó bejárót térít vissza. Ez lehetőséget teremt a további elemek feltöltésére.

Példa: fill_n

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

Kiírás

```
template <typename T>
void kiir(const T& v)
{
   cout << "v =";
   for (const auto& e : v)
        cout << " " << e;
   cout << endl;
}
```

A fill_n meghívása

```
int main() {
    vector <int> v(9, -1); // 9 db -1  értékkel tölti fel
    // ugyanaz mint
    // vector<int> v;
    // for (auto i = 0; i < 9; ++i)
    //   v.push_back(-1);
    kiir(v);
    auto poz = fill_n(v.begin(), 3, 10);
    kiir(v);
```

Fő függvény kimenet

```
    fill_n(poz, 4, 20);  // poz pozíciótól
    kiir(v);
    fill_n(v.end() - 2, 2, 30);
    kiir(v);
}
```

Kimenet:

```
v = -1 -1 -1 -1 -1 -1 -1 -1 -1
v = 10 10 10 -1 -1 -1 -1 -1 -1
v = 10 10 10 20 20 20 20 -1 -1
v = 10 10 10 20 20 20 20 30 30
```

Pszeudovéletlen számok

- A sorozat egy determinisztikus algoritmussal generálódik, de a felhasználó számára véletlenszerűnek tűnik. Erre utal a pszeudo az elnevezésben.
- Hagyományos lehetőség: rand függvény a cstdlib fejállományból: 0 és RAND_MAX közötti egész számot generál (a RAND_MAX értéke általában 32767).
- C++11-től a random fejállományban további (sablonokon alapuló) lehetőségek.

A random fejállomány

- A random_device osztály: nem determinisztikus véletlenszámot generál, ha ez támogatott az adott rendszeren. Általában a pszeudovéletlen generátor inicializálására használjuk.
- A uniform_int_distribution osztálysablon: egész számokat generál diszkrét egyenletes eloszlás szerint.
- Az mt19937 osztály: a Mersenne Twister algoritmuson alapuló pszeudovéletlen számot generáló motor. Az algoritmus periódusa egy Mersenne prím: $2^{19937} - 1$.
- További eloszlásokat meghatározó osztálysablonok és generáló motorok léteznek.

Példa: pszeudovéletlen szám

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

using namespace std;
```

Dobás dobókockával

```
int main() {
    map<int, int> hisztogram;
    for (int k = 1; k <= 6; ++k)
        hisztogram[k] = 0;
    const int dobas = 12000;
    random_device rd;
    uniform_int_distribution<int> egyenletes(1, 6);
    mt19937 mersenne(rd());
```

Hisztogram

```
  for (int i = 0; i < dobas; ++i)
     ++hisztogram[egyenletes(mersenne)];

  for (auto e : hisztogram) {
     cout << e.first << ":" << setw(5) << e.second;
     cout << " " << string(e.second / 100, '*') << endl;
  }
}
```

Lehetséges kimenet

1: 1999 **********

2: 2035 **********

3: 1979 **********

4: 1963 **********

5: 2030 **********

6: 1994 **********

Példa: generate_n

- Egy függvényobjektum által generált elemeket adja át rendre egy tároló elemeinek.
- A generate_n algoritmus első paramétere egy kimeneti iterátor, amely meghatározza azt a pozíciót ahonnan kezdjük a módosítást.
- Legfennebb n darab elemet módosít. Az n értékét a második paraméterben adjuk meg.
- A harmadik paraméter tartalmazza a predikátumot.
- C++11-től kezdődően a visszatérített érték az utolsó módosított pozíció utáni helyre hivatkozó bejáró.

Példa: generate_n

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

Kiírás

```
template <typename T>
void kiir(const string& s, const T& x) {
    cout << s;
    for (const auto& e : x) {
        cout << " " << e;
    }
    cout << endl;
}
```

A fő függvény

```
int main()
{
   const int elemszam = 5;
   vector<int> v(elemszam);
   deque<int> q(elemszam);
   random_device rd;
   mt19937 motor(rd());
   uniform_int_distribution<int> eloszlas(0, 9);
```

A generate_n meghívása

```
    generate_n(v.begin(), elemszam,
        [&]() { return eloszlas(motor); });
    kiir("v =", v);
    generate_n(q.begin(), elemszam,
        [&]() { return eloszlas(motor); });
    kiir("q =", q);
}
// Lehetséges kimenet:
// v = 4 9 1 5 0
// q = 7 0 1 3 4
```