

Standartinės šablonų bibliotekos konteineriai (*angl. Standard Template Library (STL) Containers*)

Standartiniai konteineriai

- Konteineris (*angl.* [Container](#)) yra talpykla, kuri saugo grupę objektų (elementų).
- Konteineris dinamiškai rūpinasi jos elementams saugoti reikalinga vieta.
- Konteineriai atkartoja populiariausias duomenų struktūras:
 - dinaminiai masyvai ([vector](#))
 - dėklas ([stack](#))
 - eilutė ([queue](#))
 - krūva ([priority queue](#))
 - ir kt. ([list](#), [set](#), [map](#))

Bendri reikalavimai konteineriams

- Visi konteineriai turėtų suteikti galimybes atlikti šias operacijas:
 - sukurti naują konteinerį (*constructor*),
 - sužinoti esančių elementų skaičių (*size*),
 - išvalyti konteinerį (*clear*),
 - įterpti naują elementą į konteinerį (*insert*),
 - ištrinti elementą iš konteinerio (*delete*),
 - suteikti prieigą prie konteinerio elementų.

Konteinerių palyginimas

- Konteineriai dažniausiai lyginami pagal šiuos du pagrindinius kriterijus:
 1. **Elementų išrinkimas**. Pvz. masyvo elementai pasiekiami per indeksą, dėklo (LIFO principu), eilės (FIFO principu) ir t.t.
 2. **Elementų saugojimas**. Konteineriai gali būti realizuoti kaip statinės arba dinaminės talpyklos.
- Svarbiausias akcentas renkantis konteinerį turėtų būti atsižvelgta į naujo elemento įdėjimo/pašalinimo bei jo pasirinkimo sudėtingumus.

Dinaminiai masyvai (I) (vector)

- Kaip ir tradicinių masyvų atveju, konteinerio *vector* elementai saugomi atmintyje gretimai, tačiau atmintis valdoma dinamiškai.
- Vadinasi, pritrūkus vietos naujiems elementams įterpti, naujas masyvas gali būti sukurtas, o elementai iš seno masyvo perrašyti į naująjį.
- Kad nereiktų kiekvienam naujam elementui įterpimui perskirstinėti atminties, naujai sukurto masyvo dydis dažniausiai būna didesnis nei reikalingas esamiems elementams įterpti.

Dinaminiai masyvai (2) (vector)

- Todėl lyginant su tradiciniais masyvais, konteineris vector reikalauja daugiau atminties, mainais už gebėjimą dinamiškai augti.
- Nenuostabu, kad vector konteineris yra labai efektyvus išrenkant elementus ir gana efektyvus įterpian/trinant elementus iš konteinerio pabaigos.
- Tačiau įterpian/trinant elementus iš kitos vietos nei vektoriaus galas, jis yra mažiau efektyvus nei kiti sekos (angl. sequence

Vektorių sukūrimas

```
// constructing vectors
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    // constructors used in the same order as described above:
    vector<int> first; // empty vector of ints
    vector<int> second (4,100); // four ints with value 100
    vector<int> third (second.begin(),second.end()); // iterating through second
    vector<int> fourth (third); // a copy of third

    cout << "The contents of fourth are:";
    for (unsigned int i = 0; i<fourth.size(); ++i)
        cout << ' ' << fourth[i];
    cout << '\n';

    // the iterator constructor can also be used to construct from arrays:
    int myints[] = {16,2,77,29};
    vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

    cout << "The contents of fifth are:";
    for (vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
        cout << ' ' << *it;
    cout << '\n';

    return 0;
}
```

The contents of fourth are: 100 100 100 100
The contents of fifth are: 16 2 77 29

Priskyrir operatorius “=”

```
// vector assignment
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> foo (3,0);
    vector<int> bar (5,0);

    bar = foo;
    foo = vector<int>();

    cout << "Size of foo: " << int(foo.size()) << '\n';
    cout << "Size of bar: " << int(bar.size()) << '\n';
    return 0;
}
```

```
Size of foo: 0
Size of bar: 3
```


Vektoriaus el. skaičius, talpa (I)

```
// comparing size, capacity and max_size
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> myvector(100);
    cout << "size: " << myvector.size() << "\n";
    cout << "capacity: " << myvector.capacity() << "\n";
    cout << "max_size: " << myvector.max_size() << "\n";
    return 0;
}
```

```
size: 100
capacity: 100
max_size: 4611686018427387903
```

Vektoriaus el. skaičius, talpa (2)

```
// comparing size, capacity and max_size
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> myvector(100);

    // set some content in the vector:
    myvector.push_back(10);

    cout << "size: " << myvector.size() << "\n";
    cout << "capacity: " << myvector.capacity() << "\n";
    cout << "max_size: " << myvector.max_size() << "\n";
    return 0;
}
```

```
size: 101
capacity: 200
max_size: 4611686018427387903
```

Vektoriaus el. skaičius, talpa (3)

```
// comparing size, capacity and max_size
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> myvector;

    // set some content in the vector:
    for (int i=0; i<100; i++) myvector.push_back(i);

    cout << "size: " << myvector.size() << "\n";
    cout << "capacity: " << myvector.capacity() << "\n";
    cout << "max_size: " << myvector.max_size() << "\n";
    return 0;
}
```

```
size: 100
capacity: 128
max_size: 4611686018427387903
```

Rūšiavimas (I)

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

bool comparison (int i,int j) { return (i<j); }

int main () {
    int myints[] = {32,71,12,45,26,80,53,33};
    vector<int> myvector (myints, myints+8);
    vector<int>::iterator it;

    // using default comparison (operator <):
    sort (myvector.begin(), myvector.begin()+4);

    // using function as comparison
    sort (myvector.begin(), myvector.end(), comparison);

    // print out content:
    cout << "myvector contains:";
    for (it=myvector.begin(); it!=myvector.end(); ++it)
        cout << " " << *it;

    return 0;
}
```

// 32 71 12 45 26 80 53 33

// (12 32 45 71)26 80 53 33

// 12 26 32 33 45 53 71 80

myvector contains: 12 26 32 33 45 53 71 80

Rūšiavimas (2)

```
// sort algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct point {
    double x[2];
    double f;
};

bool comparison (const point& i, const point& j) { return (i.f<j.f); }

int main () {

    vector<point> taskas(4);
    taskas[0].x[0] = 0.0; taskas[0].x[1] = 0.0; taskas[0].f = 10.0;
    taskas[1].x[0] = 1.0; taskas[1].x[1] = 0.0; taskas[1].f = 0.0;
    taskas[2].x[0] = 0.0; taskas[2].x[1] = 1.0; taskas[2].f = -10.0;
    taskas[3].x[0] = 1.0; taskas[3].x[1] = 1.0; taskas[3].f = 20.0;

    cout << "myvector contains before sort:\n";
    for (int i=0; i < 4; i++)
        cout << taskas[i].f << ", ";
    cout << "\n" << endl;

    sort (taskas.begin(), taskas.end(), comparison);
    cout << "myvector contains after sort:\n";
    for (int i=0; i < 4; i++)
        cout << taskas[i].f << ", ";

    return 0;
}
```

myvector contains before sort:
10, 0, -10, 20,

myvector contains after sort:
-10, 0, 10, 20,

Kartotinių paieškų metodas (*angl. multi start*)

1. Papildyti Jūsų turimas atsitiktinės paieškos (*angl. pure random search or Monte Carlo*) programas, kad jos iš 3 geriausių surastų sprendinių atliktų lokalias paieškas - sprendinių patikslinimus naudojantis greičiausio nusileidimo metodu (*angl. steepest descent*).

`git clone https://github.com/Remziukas/SteepestDescent.git`

2. Papildyti dar (bent) vienu laisvai pasirinktu lokalsios paieškos metodu: <http://www.mymathlib.com/optimization/nonlinear/unconstrained/>
3. Perdarykite (papildykite) savo programas, kad jos naudotų *vector* konteinerius.

Reikalavimai galutiniam darbui

1. **[2 balai]**. C++ (C) aplinkoje realizuoti dvi versijas atsitiktinės paieškos algoritmo:
 - a. su fiksuotu (pvz. 100, 1000) iteracijų skaičiumi (**for**). Jokių reikalavimų masyvams nėra.
 - b. su pasirinkta stabdymo sąlyga (**while**). Reikia naudoti dinaminis masyvus (duomenys saugomi HEAP atmintyje).
2. **[2 balai]**. Iš visų pradinių bandymo taškų (vektorių) kaip įmanoma efektyviau išrinkti fiksuotą skaičių (pvz. 3, 5, 10) geriausių taškų (t.y. tuos, kuriuose tikslo funkcijos reikšmė yra mažiausia)
3. **[3 balai]**. Papildyti Jūsų turimas atsitiktinės paieškos programas, kad jos iš 3 (5, 10) geriausių surastų sprendinių atliktų lokalias paieškas - sprendinių patikslinimus naudojantis:
 - a. greičiausio nusileidimo metodu (angl. steepest descent)
 - b. laisvai pasirinktu lokalaus optimizavimo metodu iš: <http://www.mymathlib.com/optimization/nonlinear/unconstrained/>
4. **[2 balai]**. Perdarykite savo programas, kad jos naudotų **vector** konteinerius.
5. **[1 balas]**. Pademonstruokite, kaip Matlab aplinkoje esančias funkcijas naudoti C++ aplinkoje, o C++ funkcijas naudoti Matlab aplinkoje.

Visas darbas turi būti padarytas naudojantis GIT versijų kontrolės sistema, t.y. turi matytis atlikto darbo nuoseklumas, o pats darbas publikuotas Jūsų www.github.com paskyroje.