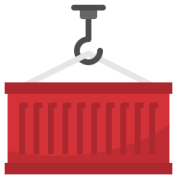
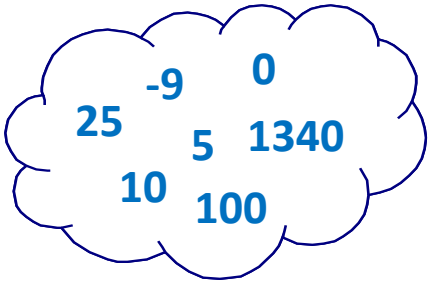
Programmētāju skola
3. līmeņa grupa

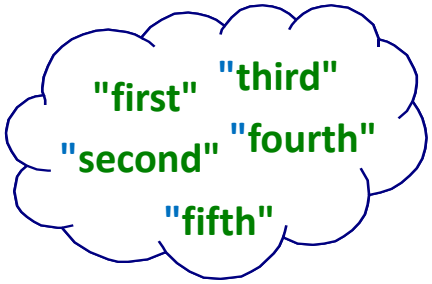
konteineri

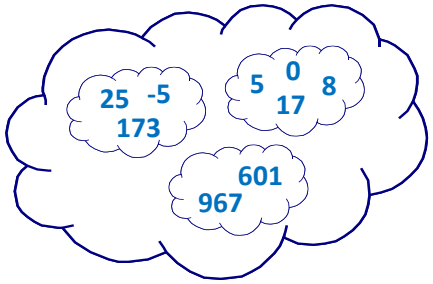
Konteiners



Konteiners – mainīgais, kas ir paredzēts **citu mainīgo glabāšanai**. Konteiners satur (contains) elementus, ļauj piekļūt tos (lasīt un mainīt), pievienot, dzēst, meklēt, ielīkt un veikt citas darbības ar šiem elementiem.







C++ valodas standarts ietver **populārus konteinerus**, kuri vecākās C++ versijās tika ieviestie atsevišķā **bibliotēkā STL** (Standard Template Library), bet tagad tie ir C++ **izpildlaikā bibliotēkas RTL** (Runtime Library) sastāvdaļas.

Konteiners	Apraksts
`vector`	Dinamiskais masīvs ar brīvpiekļuvi un automātisko izmēru maiņu
`list`	Saistītais saraksts ar secīgo piekļuvi
`set`	Unikālo elementu kopa
`map`	Asociatīvais masīvs ar unikālām atslēgām

vector

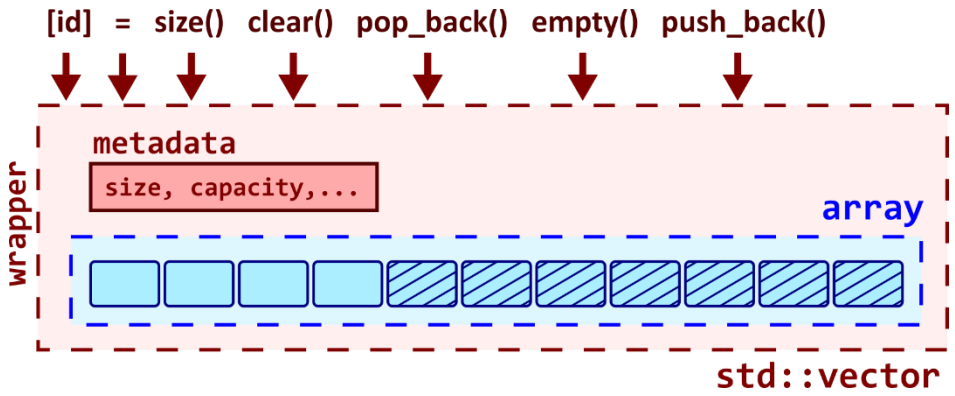
Konteiners vektors (`vector`) **paplašina standarta masīva iespējas**: saglabājot ātru brīvpiekļuvi elementiem, īsteno masīva kopēšanu, automātisku izmēra maiņu, elementus ievietošanu un noņemšanu.

Vektora konteinera deklarācija atrodas galvenes failā `<vector>` un, tāpat kā visas C++ standarta identifikatori, tas atrodas nosaukumvietā `std`.

Metode	Apraksts	Piemērs
	Izveidot vektoru	`std::vector<int> v(100);`
`=`	Kopē viena vektora elementus citam	`v = v1;`
`[]`	Brīvpiekļuve noteiktam elementam	`v[10] = 10;`
`size`	Noteikt vektora elementu skaitu	`int c = v.size();`
`empty`	Noteikt, vai vektors ir tukšais	`if (v.empty()) ...`
`clear`	Noņem visus vektora elementus	`v.clear();`
`push_back`	Pievienot elementu vektora beigām	`v.push_back(10);`
`pop_back`	Noņemiet vektora pēdējo elementu	`v.pop_back();`

vector

Vektors ir aptinums (*wrapper*), kas izveidots ap masīvu, lai kontrolētu piekļuvi masīva elementiem un nodrošinātu programmētājam ērtu līdzekli masīva uzturēšanai.



Masīvs

```
int a[100]{};
int id = 200;
a[id] = 0; // UB
```

```
for (int i = 0; i < 1'000'000'000; i++)
    a[i % 100] = i;
```

izpildes laiks **3s** (tukšais cikls **1.8**s)

Vektors

```
vector<int> v(100);
int id = 200;
v[id] = 0; // Runtime error
```

```
for (int i = 0; i < 1'000'000'000; i++)
    v[i % 100] = i;
```

izpildes laiks **6s** (tukšais cikls **1.8**s)

`vector`: elementu iterēšana

Vektora elementu secīgai apstrādāšanai (iterēšanai) pielieto gan **ciklus uz indeksa pamatā** gan speciālus **iterātorus uz vieda rādītāja bāzes** (tai skaitā C++11 cikls `foreach`).

Piemēram:

```
vector<int> v { 1, 2, 3, 4, 5 };
```

Iterēšana pēc indeksa:

```
for (int i = 0; i < v.size(); i++) cout << v[i];
```

Iterēšana pēc iteratora jeb vieda rādītāja:

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

```
for (auto i = v.begin(); i != v.end(); i++) cout << *i;
```

Iterēšana ar ciklu `foreach`:

```
for (int i : v) cout << i;
```

```
for (int& i : v) i *= 2;
```

vector: piemērs

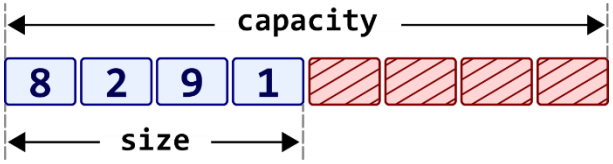
```cpp
#include <iostream>
#include <vector>

using namespace std;

void main() {
    vector<int> v{ 1, 2, 3, 4, 5 };      // 1 2 3 4 5
    v[2] = 0;                            // 1 2 0 4 5
    v.push_back(6);                      // 1 2 0 4 5 6
    v.push_back(7);                      // 1 2 0 4 5 6 7
    v.pop_back();                        // 1 2 0 4 5 6
    for (int& i : v) if (i % 2) i *= 2; // 2 2 0 4 10 6
    for (int i : v) cout << i << ' ';
}
```

vector: izmēra maiņa

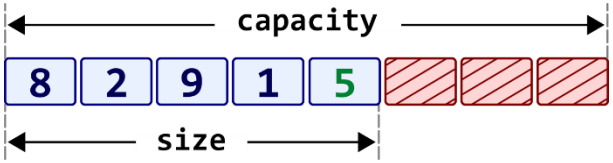
Atšķirībā no masīva, kas neļauj mainīt izmēru, vektora elementus var pievienot un noņemt, **mainot vektora izmēru**. Bet šīs operācijas vār būt ilgstošas, jo var pieprasīt vektora atmiņas pārdālišanu.

Jāpatur prātā, ka izmēra palielināšanai vektors izveido jauno masīvu.

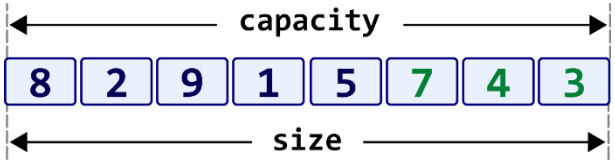
Elementu glabāšanai vektorā izveido masīvu, kura izmērs (capacity) ir lielāks par elementu skaitu (size).



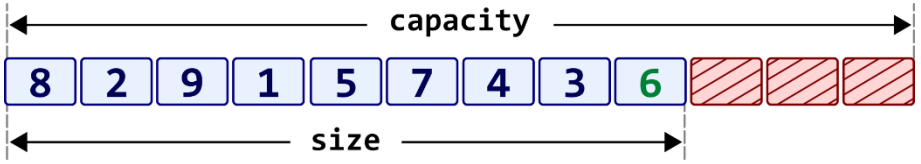
Tas ļauj pievienot jaunus elementus masīva ietilpības robežās.



Pēc masīva ietilpības izlietošanas…



…tiks izveidots jaunais lielāka izmēra masīvs un tajā tiks kopēti visi vecā masīva elementi.



Vektoru nodošana uz funkciju

Viena no vektoru priekšrocībām, salīdzinājumā ar masīviem, ir ērta to nodošana uz funkciju.

Masīvs

Masīvus var nodot uz funkciju tikai kā references.

```
int a[] {1, 2, 3, 4, 5};

void func(int arr[], int size) {…}

func(a, 5);
func({1, 2, 3, 4, 5}, 5); // error
```

Vektors

Vektora nodošanai pielieto tādas pašas noteikumus, kā vienkāršiem mainīgajiem.

```
vector<int> v {1, 2, 3, 4, 5};

void func(vector<int> arr) {…}

func(v);
func({1, 2, 3, 4, 5});

void func(vector<int>& arr) {…}

func(v);
func({1, 2, 3, 4, 5}); // error

void func(const vector<int>& arr) {…}

func(v);
func ({1, 2, 3, 4, 5}); // OK
```

Vektoru nodošanas ātrums

```
void func(int arr[], int size)
```

```
for (int i = 0; i < 1'000'000; i++) func(a, 5);
```
5 ms

```
void func(vector<int> arr)
```

```
for (int i = 0; i < 1'000'000; i++) func(v);
```
1000 ms

```
for (int i = 0; i < 1'000'000; i++) func({ 1, 2, 3, 4, 5 });
```
1000 ms

```
void func(vector<int>& arr)
```

```
for (int i = 0; i < 1'000'000; i++) func(v);
```
5 ms

```
void func(const vector<int>& arr)
```

```
for (int i = 0; i < 1'000'000; i++) func(v);
```
5 ms

```
for (int i = 0; i < 1'000'000; i++) func({ 1, 2, 3, 4, 5 });
```
1000 ms

Vektoru atgriešana no funkcijas

Vektors, atšķirībā no masīva, var būt funkcijas rezultāts.

Masīvs	Vektors
Funkcijai jānodod masīvs, kurā tā var ierakstīt rezultātu.	Vektora atgriešanai pielieto tādas pašas noteikumus, kā vienkāršiem datiem.

```
void func(int arr[], int size)
{
    for(int i = 0; i < size; i++)
        arr[i] = i;
}

int a[100]{};
func(a, 100);
```

```
vector<int> func(int size) {
    vector<int> v;
    for (int i = 0; i < size; i++)
        v.push_back(i);
    return v;
}

vector<int> v = func(100);
```

```
for (int i = 0; i < 100'000; i++)
    func(a, 100);
```

izpildes laiks **19 ms**

```
for (int i = 0; i < 100'000; i++)
     v = func(100);
```

izpildes laiks **1700 ms**

Vārdu saraksts (versija ar masīvu)

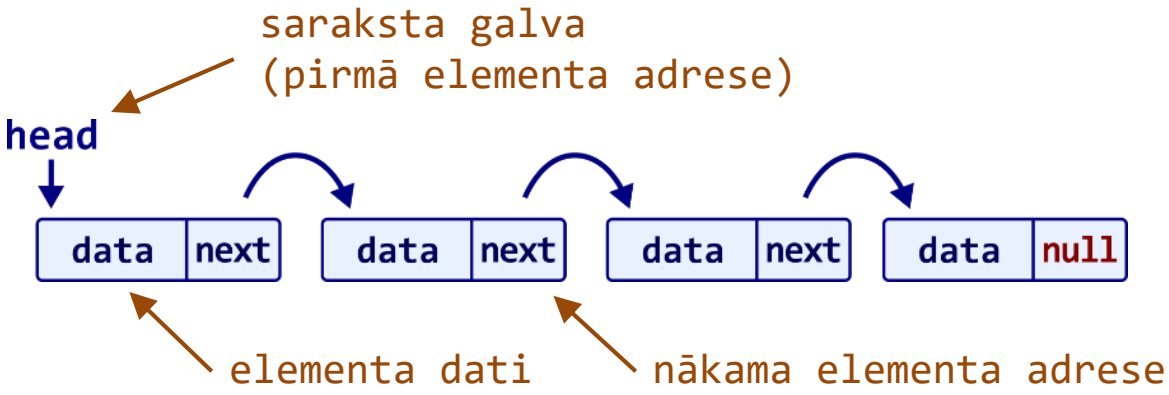
```
void collectWords(const string& str, string words[], int &count)
{
    string word{ "" };
    for (char chr : str) {
        if (isalpha(chr)) word += tolower(chr);
        else if (word != "") {
          words[count++] = word;
          word = "";
        }
    }
    if (word != "") words[count++] = word;
}
```

Vārdu saraksts (versija ar vektoru)

```
void collectWords(const string& str, vector<string>& words)
{
    string word{ "" };
    for (char chr : str) {
        if (isalpha(chr)) word += tolower(chr);
        else if (word != "") {
          words.push_back(word);
          word = "";
        }
    }
    if (word != "") words.push_back(word);
}
```

Saistītais saraksts

Saistītais saraksts – ir datu struktūra, kura elementu apvienošanai pielieto radītājus. Lai to īstenotu katrs saraksta elements, papildinājumā datiem, glabā nākama elementa adresi.



Atšķirībā no masīva, saraksta elementiem **nav jāatrodas nepārtraukta atmiņas apgabalā**, kas ļauj vieglāk pievienot sarakstam jaunus elementus. Taču saraksta **elementiem nav brīvpiekļuves**, tādēļ lai piekļūtu saraksta elementam ir jāizlasa visi elementi, kas atrodas pirms tā.

list

Konteiners saraksts (`list`) ir līdzīgs vektoram, taču elementu glabāšanai masīva vietā pielieto **saistīto sarakstu**. Tas ļauj paātrināt izmēra maiņu, bet padara neiespējamu brīvpiekļuvi elementiem.

Saraksta konteinera deklarācija atrodas galvenes failā `<list>` un, tāpat kā visas C++ standarta identifikatori, tas atrodas nosaukumvietā `std`.

Metode	Apraksts	Piemērs
	Izveidot sarakstu	`std::list<int> lst;`
`=`	Kopē viena saraksta elementus citam	`lst = lst1;`
`size`	Noteikt saraksta elementu skaitu	`int c = lst.size();`
`empty`	Noteikt, vai saraksts ir tukšais	`if (lst.empty()) ...`
`clear`	Noņem visus saraksta elementus	`lst.clear();`
`push_back`	Ievietot elementu saraksta beigās	`lst.push_back(10);`
`pop_back`	Noņemiet saraksta pēdējo elementu	`lst.pop_back();`

`list` vs `vector`

Vektoram un sarakstam ir līdzīgi uzdevumi, taču katram ir savas priekšrocības un trūkumi, kas jāņem vērā, izvēloties konteineru programmai:

Vektors			Saraksts
brīvpiekļuve	+	-	secīgā piekļuve
grūti mainīt elementu skaitu	-	+	viegli mainīgais elementu skaits
sarežģīti dzēst un ielikt elementus	-	+	viegli dzēst un ielikt elementus
kompaktā glabāšana	+	-	rādītājiem tiek tērēta papildu atmiņa
glabāšanai ir nepieciešams nepārtrauktais atmiņas apgabals	-	+	elementi var būt izplatīti pa brīviem atmiņas apgabaliem

`list`: elementu iterēšana

Atšķirībā no vektora, saraksta elementu iterēšanai nevar izmantot ciklus uz indeksa pamatā. Taču cikli ar iterātoriem (tostarp cikls `foreach`) var būt pielietoti sarakstiem tāpat kā āri vektoriem.

Piemēram:

```
list<int> l { 1, 2, 3, 4, 5 };
```

Iterēšana pēc iteratora jeb vieda rādītāja:

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

```
for (auto i = l.begin(); i != l.end(); i++) cout << *i;
```

Iterēšana ar ciklu `foreach`:

```
for (int i : l) cout << i;
```

```
for (int& i : l) i *= 2;
```

list: piemērs

```
#include <iostream>
#include <list>

using namespace std;

void main() {
   list<int> lst{ 1, 2, 3, 4, 5 };        // 1 2 3 4 5
   lst.push_back(6);                      // 1 2 3 4 5 6
   lst.push_back(7);                      // 1 2 3 4 5 6 7
   lst.pop_back();                        // 1 2 3 4 5 6
   for (int& i : lst) if (i % 2) i *= 2; // 2 2 6 4 10 6
   for (auto i : lst) cout << i << ' ';
}
```

`list`: vārdu saraksts

```
void collectWords(const string& str, list<string>& words)
{
    string word{ "" };
    for (char chr : str) {
        if (isalpha(chr)) word += tolower(chr);
        else if (word != "") {
          words.push_back(word);
          word = "";
        }
    }
    if (word != "") words.push_back(word);
}
```

set

Konteiners kopa (`set`), atšķirībā no vektora un saraksta, ietver **unikālus elementus**, katrs no kuriem var parādīties konteinerā tikai vienu reizi. Kopas elementi ir nemainīgie, kas nozīmē, ka tos nevar mainīt pēc pievienošanas kopai.

Kopas konteinera deklarācija atrodas galvenes failā `<set>` un, tāpat kā vektors un saraksts, izvietota standarta nosaukumvietā `std`.

Metode	Apraksts	Piemērs
	Izveidot kopu	`std::set<int> s;`
`=`	Kopē vienas kopas elementus citam	`s = s1;`
`size`	Noteikt kopas elementu skaitu	`int c = s.size();`
`empty`	Noteikt, vai kopa ir tukša	`if (s.empty()) ...`
`clear`	Noņem visus kopas elementus	`s.clear();`
`insert`	Pievienot elementu kopai	`s.insert(10);`
`erase`	Noņemiet elementu no kopas	`s.erase(10);`
`count`	Noteikt elementu skaitu ar uzdoto vērtību (0 vai 1)	`if (s.count(10))...`

`set`: elementu iterēšana

Atšķirībā no vektora un saraksta, kopas elementi pieejami iterēšanas ciklā **tikai lasīšanai**. Taču tāpat kā āri sarakstiem iterēsanai var būt pielietoti tikai cikli ar iterātoriem (tostarp cikls `foreach`).

Piemēram:

```
set<int> s { 1, 2, 3, 4, 5 };
```

Iterēšana pēc iteratora jeb vieda rādītāja:

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

```
for (auto i = s.begin(); i != s.end(); i++) cout << *i;
```

Iterēšana ar ciklu `foreach`:

```
for (int i : s) cout << i;
```

```
for (const int& i : s) cout << i;
```

set: piemērs

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

using namespace std;

void main() {
   std::set<int> s {1, 2, 3, 2, 4};     // 1 2 3 4
   s.insert(5);                         // 1 2 3 4 5
   s.insert(5);                         // 1 2 3 4 5
   s.erase(2);                          // 1 3 4 5
   for (auto i: s) cout << i << ' ';
}
```

set: unikālo vārdu saraksts

```
void collectWords(const string& str, set<string>& words)
{
    string word{ "" };
    for (char chr : str) {
        if (isalpha(chr)) word += tolower(chr);
        else if (word != "") {
         words.insert(word);
          word = "";
        }
    }
    if (word != "") words.insert(word);
}
```

Asociatīvais masīvs

Pēc būves un piekļuves principa masīvus iedala **indeksētajos** un **asociatīvajos**.

Array

Indeksētais masīvs (indexed array) – brīvpiekļuves masīvs, kurā piekļuvei elementam ir nepieciešams tā kārtas numurs jeb **indekss** (index):

```
Id => Value
```

```
array[0] = 25; array[1] = 32; array[2] = 17; array[3] = 40;
```

Map

Asociatīvais masīvs (associative array, map, dictionary) – secīgas piekļuves struktūra, kas glabā "**atslēga/vērtība**" veida pārus un atbalsta vērtības ar uzdoto atslēgu meklēšanu un dzēšanu.

```
Key => Value
```

Piekļuvei elementam asociatīvajā masīvā ir jānorada **atslēga** (key) indeksa vietā:

```
array["Sunday"] = 25; array["Monday"] = 32; array["Tuesday"] = 17;
```

```
array[0] = 25; array[48921] = 32; array[10762386] = 17;
```

map

Asociatīvais masīvs (**map**) satur elementu pārus: **unikālā atslēga** (key, first) un tām **atbilstošā vērtība** (value, second). Piekļuvei elementa vērtībai indeksa vietā ir jānorāda elementa atslēga.

Asociatīvā masīva konteiners atrodas galvenes failā `<map>` un, tāpat kā visas C++ standarta identifikatori, atrodas nosaukumvietā `std`.

Metode	Apraksts	Piemērs
	Izveidot asociatīvo masīvu	`map<string, int> m;`
`=`	Kopē viena asociatīvā masīva elementus citam	`m = m1;`
`[]`	Piekļuve noteiktam elementam pēc atslēgas	`m["second"] = 2;`
`size`	Noteikt asociatīvā masīva elementu skaitu	`int c = m.size();`
`clear`	Noņem visus asociatīvā masīva elementus	`m.clear();`
`insert`	Ievietot jauno elementu	`m.insert({"fourth", 40});`
`erase`	Noņem elementu ar norādīto atslēgu	`m.erase("third");`
`count`	Noteikt elementu skaitu ar uzdoto atslēgu (0 vai 1)	`if (m.count("second"))...`

map: elementu iterēšana

Asociatīva masīva elementu iterēšanai var būt pielietoti cikli ar iterātoriem (tostarp cikls `foreach`). Ciklā elementi būs pieejami kā **atslēga/vērtība pāri** (`pair`). Lai piekļūtu atslēgai, ir jānorāda vārds `first`, vērtībai – `second`. Elementu vērtības var mainīt, atslēgas ir paredzēti tikai lasīšanai.

Piemēram:

```
map<string, int> m { {"key1", 1}, { "ke2", 2 }, { "key3", 3 } };
```

Iterēšana pēc iteratora jeb vieda rādītāja:

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

```
for (auto i = m.begin(); i != m.end(); i++) cout << i->second;
```

Iterēšana ar ciklu `foreach`:

```
for (auto i: m) cout << i.first << " => " << i.second;
```

```
for (auto& i: m) if (i.first[3] == '2') i.second = 0;
```

map: piemērs

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

using namespace std;

void main() {
    map<string, int> m {
        { "first", 100 },
        { "second", 200 },
        { "third", 300 }
    };
    m.insert({ "fourth", 400 });
    m["fifth"] = 500;
    m["second"] = 2;
    m.erase("third");
    for (auto i : m) cout << i.first << " => " << i.second << endl;

    // fifth => 500, first => 100, fourth => 400, second => 2

}
```

map: vārdu skaitītāji

```
void collectWords(const string& str, map<string, int>& words)
{
    string word{ "" };
    for (char chr : str) {
        if (isalpha(chr)) word += tolower(chr);
        else if (word != "") {
            words[word]++;
            word = "";
        }
    }
    if (word != "") words[word]++;
}
```