



Tema 09

Kontejneri u jeziku C++

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



Sadržaj

1. Uvod
2. Kontejnerski šabloni
3. Kontejneri sekvenci
4. Asocijativni kontejneri
5. Kontejnerski adapteri
6. Primeri programa



1. Uvod

- Komponente Standardne i STL biblioteke
- Upotreba šablona iz STL biblioteke
- Kategorije kontejnera



Komponente Standardne i STL biblioteke

- Osnovne komponente **Standardne biblioteke** jezika C++ su sastavni deo ISO standarda, deklarisanе u prostoru imena **std**
 - npr. generički kontejneri (strukture podataka), funkcije za njihovu upotrebu, stringovi, tokovi, niti, numerička i C biblioteka
- Osnovne komponente **STL biblioteke** (Standardne biblioteke šablona/templateja) jezika C++ su:
 - **kontejneri**
 - iteratori
 - algoritmi
 - funkcijski objekti



Upotreba šablona iz STL biblioteke

- Upotreba šablona iz **STL** biblioteke omogućava jednostavnija i kraća programska rešenja brojnih praktičnih problema
- Npr. deo programa koji učitava proizvoljni broj decimalnih vrednosti sa standardnog ulaza i računa njihov prosek:

```
std::vector<double> brojevi;    // zaglavlje <vector>
std::cout << "Unesite brojeve razdvojene prazninama (Ctrl+Z za kraj):\n ";
brojevi.insert(std::begin(brojevi),
               std::istream_iterator<double>(std::cin),
               std::istream_iterator<double>()); // zaglavlje <iterator>
std::cout << "Srednja vrednost = "
          << (std::accumulate(std::begin(brojevi), // zaglavlje <numeric>
                              std::end(brojevi), 0.0)/brojevi.size())
          << std::endl;
```

Kategorije kontejnera

- **Kontejneri** su objekti (strukture podataka) koji služe za smeštanje i organizovanje drugih objekata
- Osnovne kategorije kontejnera u STL biblioteci su
 - **Kontejneri sekvenci** (*sequence containers*), u kontinualnoj ili dinamičkoj memoriji (*contiguous storage/list storage*), gde su elementi organizovani u linearne strukture, a pristup elementima se ostvaruje preko funkcije člana ili pomoću iteratora
 - **Asocijativni kontejneri**, kao što su stabla pretraživanja (*search trees*) i heš tabelle (*hash tables*), gde se pristup elementima ostvaruje pomoću ključa ili iteratora
 - **Kontejnerski adapteri** (*container adapters*) su šabloni klasa koji predstavljaju alternativne mehanizme za pristup podacima u kontejnerima sekvenci ili asocijativnim kontejnerima

2. Šabloni kontejnerskih klasa

1. Kontejnerske klase
2. Alokatori
3. Komparatori
4. Zajedničke funkcije kontejnera



2.1 Kontejnerske klase

- Šabloni kontejnerskih klasa definisani su u zaglavljima:
 - `vector`, `array`, `deque`, `list`, `forward_list`, `map`, `unordered_map`, `set`, `unordered_set` i `bitset`
- Npr. šablon `vector<T>` je kontejner za jednodimenzionalna polja, koji po potrebi automatski povećava svoje dimenzije
 - povećanje dimenzija vektora (*capacity*) podrazumeva *kopiranje* postojećih elemenata vektora radi formiranja nove kontinualne memorijske strukture
 - način automatskog povećanja dimenzija zavisi od implementacije; osnovni cilj algoritma povećavanja je da se bitno ne produži prosečno vreme izvršavanja operacija
 - povećanje se vrši na $k \cdot N$ elemenata, gde je N postojeća dimenzija, a faktor k je obično 1.5 ili 2 (Visual C++ koristi $k=1.5$)

Podsetnik: Zaglavlja kontejnerskih klasa

Zaglavlje	Opis
<code>vector</code>	<code>vector<T></code> je proširivo polje, novi elementi dodaju se na kraj
<code>array</code>	<code>array<T,N></code> je polje <i>fiksni</i> h dimenzija od N elemenata (efikasnije)
<code>deque</code>	<code>deque<T></code> je red koji se može ažurirati s obe strane
<code>list</code>	<code>list<T></code> je dvostruko povezana lista elemenata tipa <code>T</code>
<code>forward_list</code>	<code>forward_list<T></code> je jednostruko povezana lista elemenata tipa <code>T</code>
<code>map</code>	<code>map<K,T></code> je asocijativna lista objekata tipa <i>pair</i> <K,T> (K je ključ)
<code>unordered_map</code>	<code>unordered_map<K,T></code> je asocijativna lista objekata tipa <i>pair</i> <K,T> za koje nije definisan poredak
<code>set</code>	<code>set<T></code> je kontejner <i>map</i> , gde je element liste istovremeno i ključ
<code>unordered_set</code>	<code>unordered_set<T></code> je kontejner <i>set</i> za čije elemente nije definisan poredak
<code>bitset</code>	<code>bitset<T></code> je klasa koja predstavlja niz bitova, npr. flegova

Primer: Promena dimenzija kontejnera

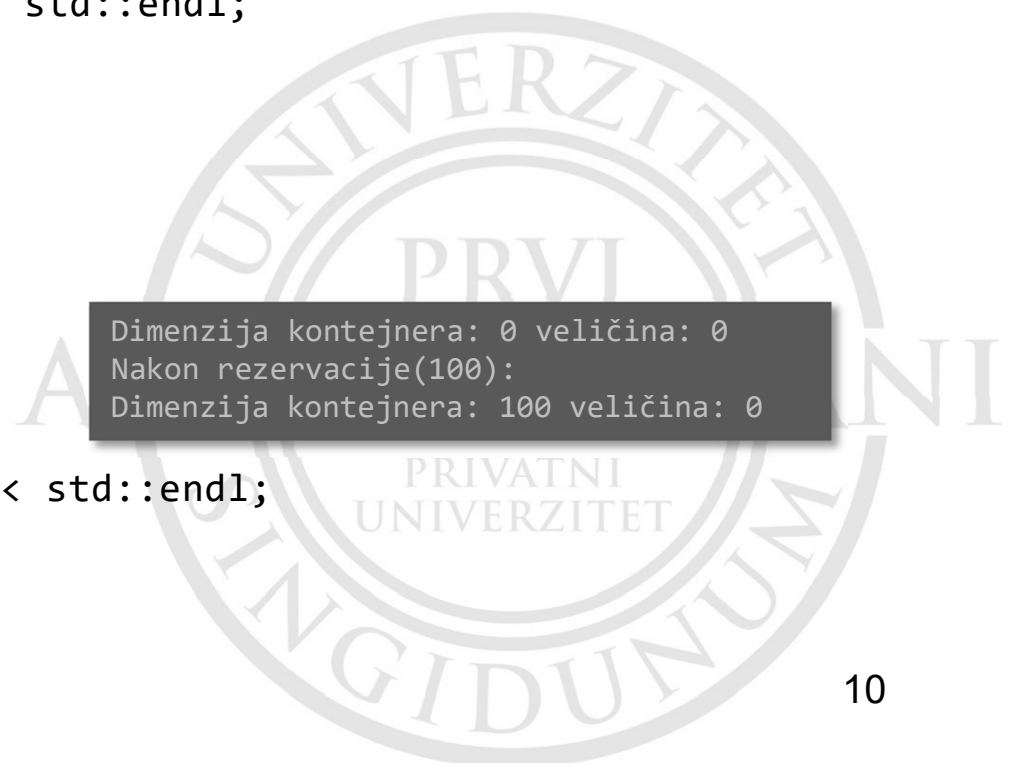
(1/2)

```
#include <iostream>
#include <vector>
using std::vector;

// Šablonska funkcija za prikaz dimenzija i broja elemenata vektora
template<class T>
void listInfo(const vector<T>& v) {
    std::cout << "Dimenzija kontejnera: " << v.capacity()
               << " velicina: " << v.size() << std::endl;
}

int main() {
    // Kreiranje osnovnog vektora
    vector<double> podatak;
    listInfo(podatak);

    // Kreiranje vektora od 100 elemenata
    podatak.reserve(100);
    std::cout << "Nakon rezervacije (100):" << std::endl;
    listInfo(podatak);
```



```
Dimenzija kontejnera: 0 velicina: 0
Nakon rezervacije(100):
Dimenzija kontejnera: 100 velicina: 0
```

Primer: Promena dimenzija kontejnera (2/2)

```
// Kreiranje i inicijalizacija vektora od 10 elemenata (-1)
vector<int> brojevi(10,-1);
std::cout << "Inicijalna vrednost vektora je: ";
for (auto n : brojevi) std::cout << " " << n; // petlja nad vektorom
std::cout << std::endl << std::endl;
```

auto - tip vrednosti određuje inicijalizator, u ovom slučaju tip elementa kontejnera

```
// Provera da li dodavanje elemenata utiče na obim vektora
auto staraDim = brojevi.capacity(); // stari obim
auto novaDim = staraDim; // novi obim, nakon dodavanja elementa
listInfo(brojevi);
for (int i=0; i<1000; i++) {
    brojevi.push_back(2*i);
    novaDim = brojevi.capacity();
    if (staraDim < novaDim) { // ako se
        staraDim = novaDim;
        listInfo(brojevi);
    }
}
return 0;
}
```

```
Dimenzija kontejnera: 0 velicina: 0
Nakon rezervacije (100):
Dimenzija kontejnera: 100 velicina: 0
Inicijalna vrednost vektora je: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Dimenzija kontejnera: 10 velicina: 10
Dimenzija kontejnera: 15 velicina: 11
Dimenzija kontejnera: 22 velicina: 16
Dimenzija kontejnera: 33 velicina: 23
Dimenzija kontejnera: 49 velicina: 34
Dimenzija kontejnera: 73 velicina: 50
Dimenzija kontejnera: 109 velicina: 74
Dimenzija kontejnera: 163 velicina: 110
Dimenzija kontejnera: 244 velicina: 164
Dimenzija kontejnera: 366 velicina: 245
Dimenzija kontejnera: 549 velicina: 367
Dimenzija kontejnera: 823 velicina: 550
Dimenzija kontejnera: 1234 velicina: 824
```

automatsko povećanje dimenzija vektora (capacity) na $k \cdot N$ elemenata (N je postojeći obim N , a faktor $k=1.5$)

Svojstva STL kontejnera

- U STL kontejnerima pamte se *kopije* objekata, osim kad su u pitanju privremeni objekti koji se mogu premeštati
 - na taj način se originalni objekti mogu nezavisno menjati
- Kopiranje složenih objekata može biti neefikasno, pa je bolje u kontejnerima pamtiti *pokazivače* ili koristiti objekte koji se mogu premeštati
- Osim toga, u kontejnere koji služe za pamćenje objekata osnovnih klasa ne treba smeštati objekte izvedenih klasa jer dolazi do gubitka informacija izvedenih klasa (*object slicing*)
- Konstruktori STL kontejnerskih klasa ne prijavljuju izuzetke (*noexcept*)

Svojstva elemenata kontejnera

- **Element** kontejnera tipa *T* mora imati neka osnovna svojstva, minimalno kao u primeru:

```
class T {  
    public:  
        T();                // podrazumevajući konstruktor  
        T(const T& t);      // konstruktor kopije  
        ~T();              // destruktor  
        T& operator=(const T& t); // operator dodele  
};
```

- Predvodilac najčešće generiše podrazumevajuće elemente za većinu postojećih tipova
- U kontejnerima map i set neophodna je još definicija *funkcije poređenja* elemenata, koja omogućava realizaciju sortiranja

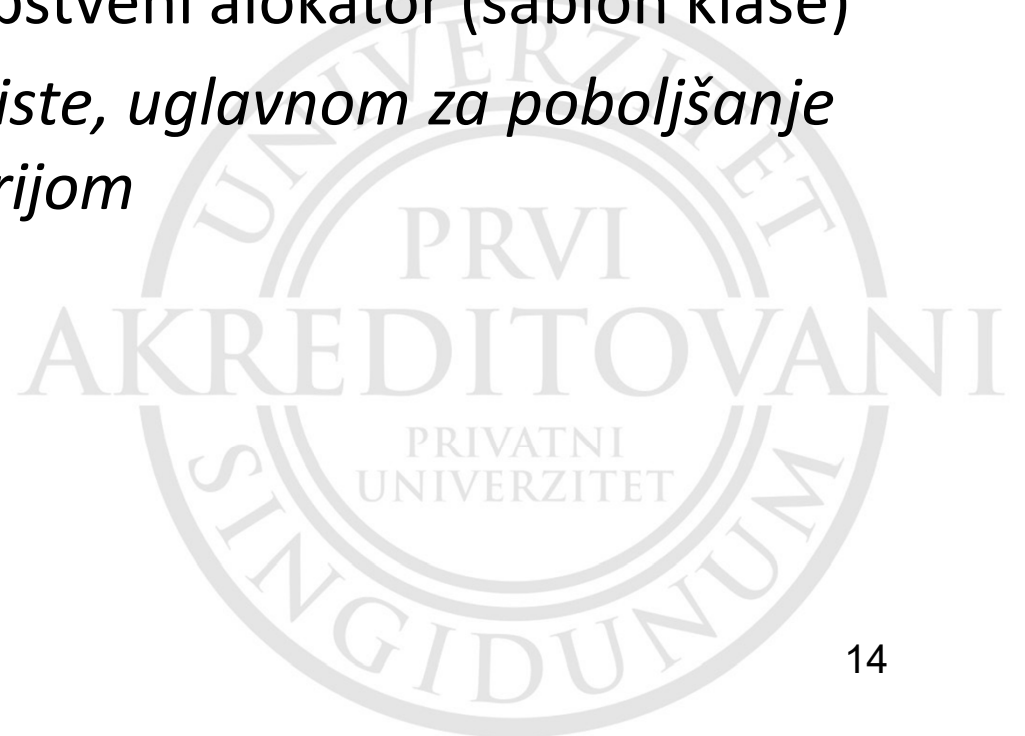
2.2 Alokatori

- Većina kontejnerskih struktura automatski se prilagođava broju elemenata koji se u njih smeštaju
- Npr. stvarna forma šablona `vector<T>` ima još jedan argument

```
vector<T, Allocator=allocator<T>>
```

tako da je moguće definisati sopstveni alokator (šablon klase)

- *Sopstveni alokatori se retko koriste, uglavnom za poboljšanje performansi upravljanja memorijom*

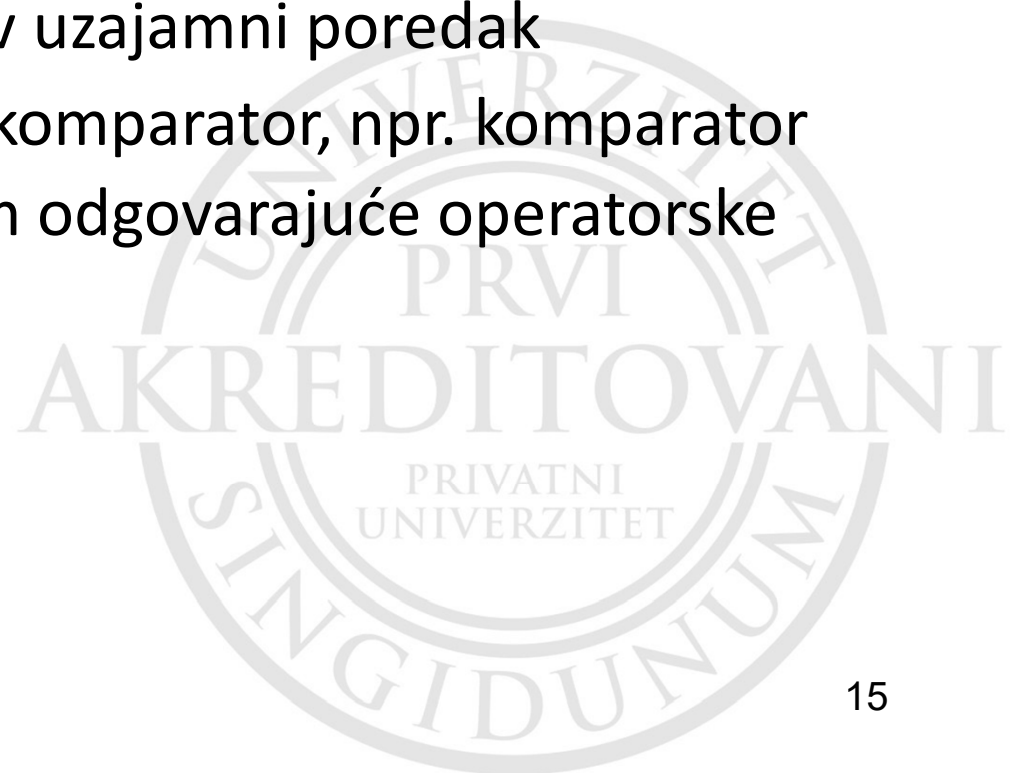


2.3 Komparatori

- Neki kontejneri pretpostavljaju poredak svojih elemenata, kao npr. kontejner *map*

```
map<K, T, Compare=less<K>, Allocator=allocator<pair<K,T>> >
```

- Element strukture *Compare* je funkcijski objekt koji poredi ključeve tipa *K* i određuje njihov uzajamni poredak
- Moguće je definisati sopstveni komparator, npr. komparator "veći od" dobija se definisanjem odgovarajuće operatorske funkcije *operator>()*



3. Kontejneri sekvenci

1. Svojstva kontejnera sekvenci
2. Zajedničke funkcije kontejnera sekvenci
3. Polja
4. Vektori
5. Liste



3.1 Svojstva kontejnera sekvenci

- Kontinualni kontejneri sekvenci omogućavaju direktan pristup svakom elementu za *konstantno* vreme $O(1)$
- **Polje** je kontejner `std::array<T,N>` i predstavlja standardno polje fiksne veličine N
- **Vektor** je kontejner `std::vector<T>` i predstavlja strukturu čija se veličina menja automatski
 - za kreiranje vektora koristi se dinamička memorija (*heap*)
 - povećanje veličine i brisanje nekog elemenata vektora prouzrokuje *kopiranje* sadržaja u drugi kontinualni blok memorije odgovarajuće veličine, uz istovremeno *oslobađanje* prethodno zauzetog memorijskog bloka

Svojstva kontejnera sekvenci

- Kontejneri tipa **liste** koriste se kada su operacije dodavanja i brisanja elemenata česte
- **Dvostrana lista** (*double-ended queue*), kontejner `std::deque`, predstavlja red čekanja koji čuva elemente u kontinualnim, ali međusobno nezavisnim blokovima dinamičke memorije
 - to omogućava veoma brzo dodavanje elemenata na početak ili kraj, bez njihovog premeštanja
- **Dvostruko povezana lista**, kontejner `std::list`, omogućava obilazak elemenata u dva smera, unapred i unazad
- **Jednostruko povezana lista**, kontejner `std::forward_list`, može se obići za vreme reda $O(n)$, dok je vreme dodavanja i brisanja elemenata konstantno, $O(1)$

3.2 Zajedničke funkcije kontejnera sekvenci (1/2)

- Zajedničke funkcije za kontejnerske klase `array`, `vector` i `deque` su:

Funkcija	Opis
<code>begin(), end()</code>	vraća iterator begin/end
<code>rbegin(), rend()</code>	vraća reverzni iterator begin/end
<code>cbegin(), cend()</code>	vraća <code>const</code> begin/end iterator (elementi nepromenjivi)
<code>crbegin(), crend()</code>	vraća <code>const</code> reverse begin/end iterator (elementi nepromenjivi)
<code>assign()</code>	prepisuje sadržaj novim skupom elemenata
<code>operator=()</code>	prepisuje element kopijom drugog elementa istog tipa ili liste inicijalizacije
<code>size(), max_size()</code>	vraća aktuelni, odnosno najveći broj elemenata
<code>capacity()</code>	vraća broj alociranih elemenata
<code>empty()</code>	vraća <code>true</code> ako u kontejneru nema elemenata
<code>resize()</code>	menja aktuelni broj elemenata

Zajedničke funkcije kontejnera sekvenci (2/2)

Funkcija	Opis
<code>shrink_to_fit()</code>	smanjuje memoriju potrebnu za aktuelni broj elemenata
<code>front()</code>	vraća referencu na prvi element
<code>back()</code>	vraća referencu na poslednji element
<code>operator[]()</code>	pristupa elementu prema zadanom indeksu
<code>at()</code>	pristupa elementu prema zadanom indeksu uz proveru granica
<code>push_back()</code>	dodaje element na kraj sekvence
<code>insert()</code>	umeće element od zadane pozicije
<code>emplace()</code>	kreira element na zadanoj poziciji
<code>emplace_back()</code>	kreira element na poslednoj poziciji
<code>pop_back()</code>	uklanja element s kraja sekvence
<code>erase()</code>	uklanja jedan ili više elemenata sekvence
<code>clear()</code>	uklanja sve elemente sekvence čija veličina postaje 0
<code>swap()</code>	menja vrednosti dvaju elemenata sekvence
<code>data()</code>	vraća pokazivač na interno polje koje sadrži elemente

3.3 Polja

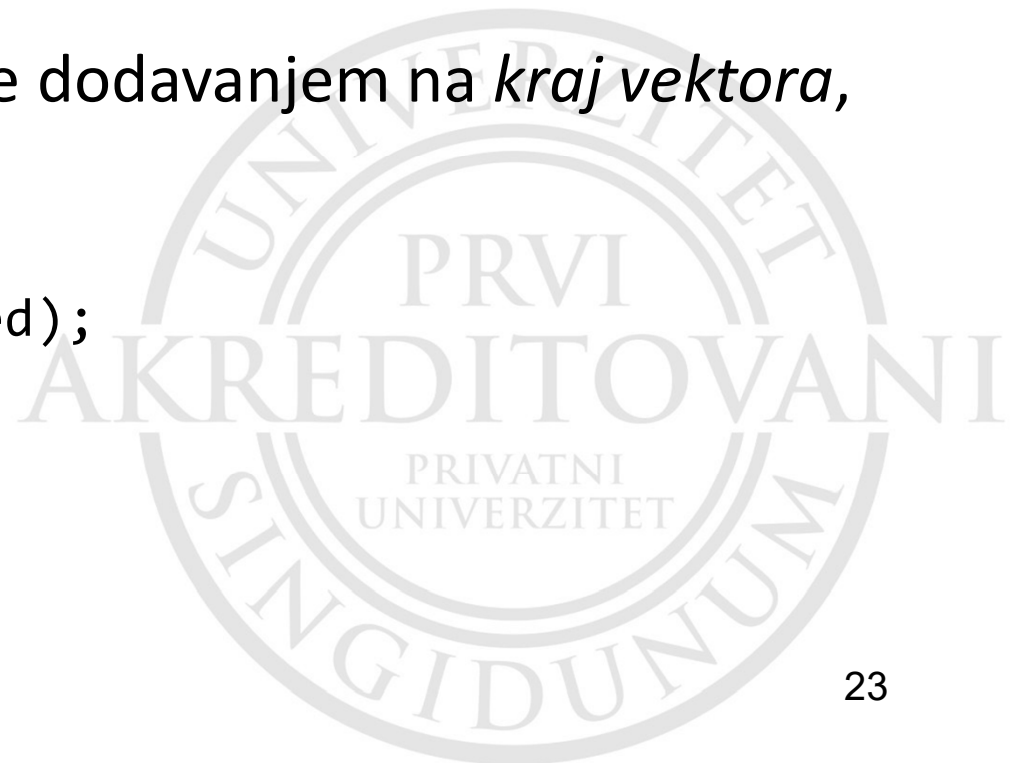
- Polje `array<T, N>` je najjednostavnija standardna kontejnerska klasa, čiji su parametri šablona tip elemenata `T` i fiksna dimenzija polja `N`
 - pristup elementima polja ima složenost $O(1)$, kao i za ugrađeni tip polja, ali kontejnerska klasa ima dodatne metode linearne složenosti $O(N)$, koje ugrađeni tip polja nema: `begin()`, `end()` i preklopljene operatore `=`, `==` i `<`
- Inicijalizacija polja vrši se pomoću dva para zagrada: spoljašnji par je za objekt `array<T, N>`, a unutrašnji za član `T[N]`
`array<string, 3> {{ "jedan", "dva", "tri" }};`
- Ovo omogućava upotrebu polja kao rezultata funkcije, npr.
`return {{ "jedan", "dva", "tri" }};`

3.4 Vektori

- Vektor `vector<T>` je kontejner sekvenci koji predstavlja kontinualno polje podataka, čija se veličina može menjati
 - kontinualni raspored podataka omogućava direktan pristup elementima za konstantno vreme $O(1)$
- Dodavanje elemenata zahteva *realokaciju* vektora, kako bi se očuvao njihov kontinualni raspored u memoriji
 - memorija vektora izuzima se iz dinamičke memorije (*heap*)
 - realokacije se ne vrši za svaki novi element koji se dodaje na kraj; radi efikasnosti, unapred se alocira $k \cdot N$ novih elemenata, gde je N tekuća dimenzija vektora, a k obično 1.5 ili 2
 - broj elemenata koje vektor može da sadrži je njegov kapacitet `capacity()`, a broj vrednosti u vektoru je njegova veličina `size()`, tako da važi $size() \leq capacity()$

Višedimenzionalni vektori

- Višedimenzionalni vektori definišu se kao vektori vektora; npr. dvodimenzionaln matrica celih brojeva može se definisati kao
`vector<vector<int>> matrica;`
- Pristup pojedinačnim elementima se vrši pomoću više indeksa
`matrica[i][j] = broj;`
- Proširenje dimenzija se realizuje dodavanjem na *kraj vektora*, a ne pojedinačnog podatka
`vector<int> red(10)`
`matrica[i][j] = push_back(red);`



3.5 Liste

- Kontejner `list<T>` predstavlja dvostruko povezanu listu elemenata tipa `T`, tako da je moguć obilazak liste u oba smera
 - iterator `list<T>::iterator` je *bidirekcioni*
- Lista podržava sve operacije koje ima vektor izuzev direktnog pristupa elementima
- Vreme pristupa elementu liste je reda $O(N)$, ali se *dodavanje* na kraj i *izuzimanje* elemenata s kraja liste vrši za konstantno vreme $O(1)$
- Osim dodavanja i uklanjanja elemenata, efikasna operacija je *sortiranje* liste, jer se sastoji samo od međusobne zamene niza pokazivača

4. Asocijativni kontejneri

1. Svojstva asocijativnih kontejnera
2. Skupovi i kolekcije
3. Heš tabele



4.1 Svojstva asocijativnih kontejnera

- Asocijativni kontejneri omogućavaju *sortiranje* i *pretraživanje* elemenata na osnovu relacije poretka i vreme pristupa $O(\log(n))$
 - **Skup**, kontejner `std::set`, najjednostavnija je struktura jedinstvenih, ali uporedivih elemenata
 - Kontejner `std::map` čuva elemente u parovima ključ-vrednost (ključ se koristi za sortiranje i pronalaženje vrednosti elemenata)
 - Kontejneri `std::multiset` i `std::multimap` ne zahtevaju da elementi strukture budu jedinstveni, odnosno međusobno različiti
 - **Heš table**, kontejneri `std::unordered_set` i `std::unordered_map`, omogućavaju pristup (pronalaženje) elemenata za konstantno vreme $O(1)$, s tim da ne zahtevaju jedinstvenost elemenata i ne bave se njihovim poređenjem

4.2 Skupovi i kolekcije

- **Skupovi** se mogu predstavljati kontejnerskim klasama `set`, `unordered_set` i `multiset`
 - skupovi sadrže samo jedinstvene, međusobno različite elemente, dok se u kolekcijama isti elementi mogu ponavljati
- Pristup elementima ostvaruje se za konstantno vreme $O(1)$
- Između elemenata kontejnera `set` definisana je relacija poretka, tako da su elementi uređeni, dok su kontejneri `unordered_set` skupovi neuređenih elemenata
- **Kolekcije** se mogu predstaviti kontejnerima `multiset` i `unordered_multiset`, u kojima se isti elementi mogu ponavljati

4.3 Heš tabele

- Heš tabele predstavljaju asocijativne kontejnere kod kojih se pristup elementima realizuje za konstantno vreme $O(1)$
 - elementi heš tabele se pronalaze prema *ključu*, ali ne postoji njihov prirodni poredak, pa se heš tabele ne mogu jednostavno obilaziti u sortiranom redosledu
 - korisnik može da upravlja veličinom heš tabele i da izabere heš funkciju
- Heš tabele se predstavljaju kontejnerskim klasama `unordered_set` i `unordered_map`
 - u mnogim situacijama mogu se zameniti klasama `set` i `map`, koje imaju definisan *poredak* elemenata, odnosno relaciju `smaller<`
 - ako je neophodno ponavljanje ključeva, mogu se koristiti kontejnerske klase `unordered_multiset` ili `unordered_multimap`

Kontejner map

- Kontejner `map<K, T>` služi za smeštanje elemenata tipa `pair<const K, T>` koji sadrži parove elemenata ključ/objekt, gde je ključ tipa `K`, a objekt tipa `T`
 - vrednost ključa mora biti *jedinstvena*, nema duplih ključeva. Poredak objekata definisan je porekom ključeva, koji se međusobno porede funkcijom `less<K>` ili korisnički definisanom funkcijom
- Primer: podaci o starosti osoba u obliku para `<ime, starost>`, gde je *ime* tipa `string`, a *starost* vrednost tipa `size_t`
`std::map<std::string, size_t> osobe`
`{ {"Ana", 25}, {"Ivana", 33}, {"Bojan", 32} };`
- Tipični metodi kontejnera `map` i `multimap` su:
`size()`, `empty()`, `max_size()`, `count(k)`, `find(k)`, `lower_bound(k)`,
`upper_bound(k)`, `equal_range(k)` i `swap(k1, k2)`

5. Kontejnerski adapteri

- Kontejnerski **adapteri** su šabloni klasa koji se definišu na osnovu *postojećih* kontejnerskih klasa, najčešće ograničavanjem njihovih svojstava
- Primeri kontejnerskih adaptera su **red** (*queue*), **stek** (*stack*) i **prioriteni red** (*priority queue*), koji se definišu ograničavanjem operacija samo na jednu stranu nekog osnovnog kontejnera
 - **red** se može definisati na osnovu kontejnera `deque<T>` ili `list<T>`
 - **stek** se može definisati na osnovu kontejnera `deque<T>`, `vector<T>` ili `list<T>`



Primeri kontejnerskih adaptera

- Adapter **red** (*queue*, dostupan preko zaglavlja **<queue>**)
dodaje elemente na kraj, a izuzima s početka reda
 - ima metode : **empty()**, **size()**, **front()**, **back()**, **push_back()** i **pop_front()**
- Adapter **prioriteni red** (*priority_queue*, dostupan preko zaglavlja **<queue>**)
 - ima metode:
- Adapter **stek** (*stack*, dostupan preko zaglavlja **<stack>**)
 - ima metode:



Primer: Definisanje sopstvenog kontejnerskog adaptera *sortirani vektor*

```
// Sopstveni kontejner: sortirani vektor celih brojeva
// - dodavanje elemenata u sortiranom poretku
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> // std::lower_bound, std::sort
#include <iterator>
using namespace std;

void insert_sorted(vector<int> &v, const int &item) {
    const auto insert_pos(lower_bound(begin(v), end(v), item)); // insert na poziciju prvog koji nije
    manji v.insert(insert_pos, item);
}

int main() {
    vector<int> v {6,5,8,2,4};
    sort(begin(v), end(v)); // pocetno sortiranje

    insert_sorted(v, 7);
    insert_sorted(v, 1);
    insert_sorted(v, 0);

    for (const auto &w : v) {
        cout << w << " ";
    }
    cout << '\n'; // 0 1 2 4 5 6 7 8
}
```



Primer: Definisanje sopstvenog kontejnerskog adaptera *sortirani vektor*

```
// Sopstveni kontejner: sortirani vektor
// - dodavanje elemenata u sortiranom poretku
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <iterator>
using namespace std;
```

```
template <typename C, typename T>
void insert_sorted(C &v, const T &item) {
    const auto insert_pos(lower_bound(begin(v), end(v), item)); // insert na poziciju prvog koji nije
    manji v.insert(insert_pos, item);
}
```

```
int main() {
    vector<int> v {6,5,8,2,4};
    sort(begin(v), end(v)); // pocetno sortiranje
    insert_sorted(v, 7); insert_sorted(v, 1); insert_sorted(v, 0);
    for (const auto &w : v) cout << w << " "; cout << '\n'; // 0 1 2 4 5 6 7 8
```

```
    vector<string> v1 {"neke", "slucajne", "reci", "bez", "nekog", "reda"};
    sort(begin(v1), end(v1)); // pocetno sortiranje
    insert_sorted(v1, "zaba"); insert_sorted(v1, "baba");
    for (const auto &w:v1) cout << w << " "; cout << '\n';
}
```

```
0 1 2 4 5 6 7 8
baba bez neke nekog reci reda slucajne zaba
```

6. Primeri programa

1. Vektori korisničkih klasa [5]
2. Program za računanje izraza u obrnutoj poljskoj notaciji (RPN) [7]



6.1 Vektori korisničkih klasa

- Definicija klase **Osoba** [Osoba.h]
- Unos podataka o osobama u kontejner **vektor**



Definicija klase Osoba [Osoba.h](1/4)

```
// Klasa definiše osobe po imenima i prezimenima
#pragma once
#include <cstring>
#include <iostream>

class Osoba {
public:
    // Konstruktor, uključuje default
    Osoba(const char* aIme = "Petar", const char* aPrezime = "Petrovic") {
        initImePrezime(aIme, aPrezime);
    }
    // Konstruktor kopije i konstruktor premeštanja
    Osoba(const Osoba& p) {
        initImePrezime(p.ime, p.prezime);
    }
    Osoba(Osoba&& p) {
        ime = p.ime;
        prezime = p.prezime;
        // Brisanje pokazivača na objekt radi prevencije brisanja
        p.ime = nullptr;
        p.prezime = nullptr;
    }
}
```



Definicija klase Osoba [Osoba.h] (2/4)

```
// Destruktor
virtual ~Osoba() {
    delete[] ime;
    delete[] prezime;
}

// Preklopljeni operator <
bool operator<(const Osoba& p) const {
    int result = strcmp(prezime, p.prezime);
    return (result < 0 || result == 0 && strcmp(ime, p.ime) < 0);
}

// Operator dodele
Osoba& operator=(const Osoba& p) {
    // Prevencija dodele oblika p = p
    if (&p != this) {
        delete[] ime;
        delete[] prezime;
        initImePrezime(p.ime, p.prezime);
    }
    return *this;
}
```



Definicija klase Osoba [Osoba.h] (3/4)

```
// Operator dodele i premeštanja
Osoba& operator=(Osoba&& p) {
    // Prevencija dodele p = p
    if (&p != this) {
        // Oslobođanje memorije
        delete[] ime;
        delete[] prezime;
        ime = p.ime;
        prezime = p.prezime;
        p.ime = nullptr;
        p.prezime = nullptr;
    }
    return *this;
}

// Prikaz podataka osobe
void prikaziOsobu() const {
    cout << ime << " " << prezime << endl;
}
```



Definicija klase Osoba [Osoba.h] (4/4)

```
private:
    char* ime{};
    char* prezime{};

    // Pomoćna privatna funkcija za inicijalizaciju imena i prezimena nove osobe
    void initImePrezime(const char* aIme, const char* aPrezime) {
        size_t length = strlen(aIme) + 1;
        ime = new char[length];
        strcpy_s(ime, length, aIme);
        length = strlen(aPrezime) + 1;
        prezime = new char[length];
        strcpy_s(prezime, length, aPrezime);
    }
};
```



Unos podataka o osobama u kontejner Vektor(1/2)

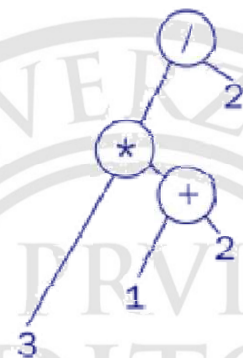
```
#include <iostream>
#include <vector>
#include "Osoba.h"
using namespace std;

int main() {
    vector<Osoba> ljudi;           // Vektor objekata klase Osoba
    const size_t maxduz = 50;
    char ime[maxduz];
    char prezime[maxduz];
    // Unos podataka o ljudima
    while (true) {
        cout << "Unesite ime osobe (Enter za kraj): ";
        cin.getline(ime, maxduz, '\n'); // var,len,delimiter
        if (strlen(ime) == 0)
            break;
        cout << "Unesite prezime osobe: ";
        cin.getline(prezime, maxduz, '\n');
        ljudi.emplace_back(ime, prezime);
    }
```

```
Unesite ime osobe (Enter za kraj): Petar
Unesite prezime osobe: Petrovic
Unesite ime osobe (Enter za kraj): Mitar
Unesite prezime osobe: Mitrovic
Unesite ime osobe (Enter za kraj): Ivana
Unesite prezime osobe: Ivanovic
Unesite ime osobe (Enter za kraj):
```


6.2 Program za računanje izraza u obrnutoj poljskoj notaciji (RPN)

- Obrnuta poljska notacija (*Reverse Polish Notation*) je način predstavljanja izraza u funkcionalnom obliku, bez zagrada
 - npr. infiksni aritmetički izraz : $(1 + 2) * 3 / 2$
u funkcionalnom obliku se predstavlja kao: $/ 2 * + 1 2 3$
dok je u obrnutoj (poljskoj) notaciji : $3 2 1 + * 2 /$
 - rezultat računanja izraza je $(1+2)*3/2 = 9/2 = 4.5$
 - izračunavanje (evaluacija) ovakvih izraza vrši se rekurzivnim postupkom, koji se iterativno realizuje pomoću strukture *stek* (kontejnerski adapter iz STL biblioteke)
- U realizaciji programa koriste se kontejnerski adapter *stack*, kontejner *map* i *lambda* funkcije



Program za računanje izraza u obrnutoj poljskoj notaciji (RPN)

```
#include <iostream>
#include <stack>
#include <iterator>
#include <map>
#include <sstream>
#include <vector>
#include <stdexcept>
#include <cmath>
using namespace std;

template <typename IT>
double evaluate_rpn(IT it, IT end) {
    stack<double> val_stack;
    map<string, double (*)(double, double)> ops{
        { "+", [](double a, double b) { return a + b; } },
        { "-", [](double a, double b) { return a - b; } },
        { "*", [](double a, double b) { return a * b; } },
        { "/", [](double a, double b) { return a / b; } },
        { "^", [](double a, double b) { return pow(a, b); } },
        { "%", [](double a, double b) { return fmod(a, b); } },
    };
};
```

Program za računanje izraza u obrnutoj poljskoj notaciji (RPN)

```
auto pop_stack( [&]()  
                { auto r(val_stack.top()); val_stack.pop(); return r; } );  
for (; it != end; ++it) {  
    stringstream ss{ *it };  
    double val; ss >> val;  
    if (val) {  
        val_stack.push(val);  
    }  
    else {  
        const auto r { pop_stack() };  
        const auto l { pop_stack() };  
        try {  
            val_stack.push(ops.at(*it)(l, r));  
        }  
        catch (const out_of_range &) {  
            throw invalid_argument(*it);  
        }  
    }  
}  
return val_stack.top();  
}
```



Program za računanje izraza u obrnutoj poljskoj notaciji (RPN)

```
int main() {  
    try {  
        cout << evaluate_rpn(istream_iterator<string>{cin}, {}) << '\n';  
    }  
    catch (const invalid_argument &e) {  
        cout << "Neispravan operator " << e.what() << '\n';  
    }  
    system("pause");  
}
```

```
3 2 1 + * 2 /  
^Z  
4.5  
Press any key to continue . . .
```



Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Horton I., *Beginning C++*, Apress, 2014
5. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. Galowitz J., *C++ 17 STL Cookbook*, Packt, 2017
8. Veb izvori
 - <http://www.cplusplus.com/doc/tutorial/>
 - <http://www.learncpp.com/>
 - <http://www.stroustrup.com/>
9. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

