

# **Tema 15**

## **Algoritmi sortiranja i ažuriranja iz biblioteke šablona jezika C++**

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



# Sadržaj

1. Uvod
2. Sortiranje nizova metodima biblioteke STL
3. Stapanje (ažuriranje) nizova metodima biblioteke STL
4. Primeri



# 1. Uvod

- Sortiranje podataka
- Efikasni algoritmi sortiranja
- Metodi sortiranja u biblioteci STL
- Metodi spajanja u biblioteci STL



# Sortiranje podataka

- **Sortiranje** podataka je veoma čest zadatak programiranja, posebno u obradi transakcija
- **Algoritmi sortiranja** su neophodni kad je potrebno obezbediti određeni poredak podataka u kontejnerima sekvenci (nizova)
  - asocijativni kontejneri obezbeđuju uređenost elemenata po definiciji
- Metodi sortiranja u biblioteci STL omogućavaju sortiranje bilo kakvih objekata koji se mogu *porediti*, a njihova implementacija je za većinu primena dovoljno *efikasna*
- Sortiranje podataka često prethodi operacijama *ažuriranja*, jer ono može biti efikasnije kad se vrši u istom redosledu u kome su podaci na koje se primenjuju

# Efikasni algoritmi sortiranja

- Složenost različitih metoda sortiranja nizova objekata razmatra se u srednjem, najboljem i najgorem slučaju
  - Efikasni su algoritmi čija je srednja vremenska složenost reda  $O(n \log n)$
  - Stabilni su algoritmi sortiranja koji čuvaju originalni redosled elemenata koji imaju jednaki ključ, npr. efikasan algoritam *Quick Sort* nije stabilan

Algoritam	Najbolji slučaj	Prosečno	Nagori slučaj	Stabilnost
Quicksort	$n \log n$	$n \log n$	$n^2$	ne
Merge sort	$n \log n$	$n \log n$	$n \log n$	da
Heapsort	$n \log n$	$n \log n$	$n \log n$	ne
Insertion sort	$n$	$n^2$	$n^2$	da
Selection sort	$n^2$	$n^2$	$n^2$	ne
Bubble sort	$n$	$n^2$	$n^2$	da

# Metodi sortiranja u biblioteci STL

- Metodi sortiranja u biblioteci STL pogodni su kao *opšti* metodi sortiranja podataka u STL kontejnerima
  - postoji veliki broj specijalnih metoda sortiranja, koji nisu ugrađeni u standardnu biblioteku jezika C++
- Biblioteka STL u zaglavlju **<algorithms>** sadrži sledeće metode sortiranja:
  - `sort()`
  - `stable_sort()`
  - `partial_sort()`
  - `nth_element()`

*Napomena:* metod `sort()` obično koristi poboljšanu verziju algoritma *QuickSort*, dok metod `stable_sort` koristi verziju algoritma *MergeSort*

# Metodi spajanja u STL biblioteci

- Biblioteka STL u zaglavlju `<algorithms>` sadrži sledeće metode spajanja-ažuriranja nizova objekata:
  - `merge()`
  - `inplace_merge()`



## 2. Sortiranje nizova metodima biblioteke STL

1. Sortiranje nizova
2. Poredak jednakih elemenata
3. Parcijalno sortiranje
4. N-ti element sortiranog niza
5. Provera sortiranoosti nizova





## 2.1 Sortiranje nizova

- Veliki broj aplikacija zasniva se na *sortiranju* objekata
  - (1) zato što zahtevaju prethodno sortirane podatke ili
  - (2) zbog toga što sortiranje poboljšava njihove performanse
- Šablon `sort<Iter>`, koji postoji u zaglavlju `<algorithms>`, sortira nizove elemenata u (podrazumevajućem) *rastućem* poretku, pod uslovom da je definisan operator `<` za poređenje tipa objekata iz niza
- Uz to, objekti
  - moraju biti međusobno zamenjivi (*swapable*), pomoću funkcije šablona `swap()` definisane u zaglavlju `<utility>` i
  - moraju imati definisan konstruktor premeštanja (*move constructor*) i operator dodele i premeštanja (*move assignment operator*)

# Sortiranje nizova

- Parametar šablona `sort()` je tipa iteratora *niza s direktnim pristupom*, tako da je sortiranje elemenata moguće samo u kontejnerima tipa *polja, vektora i dvostrane liste*
  - liste i jednostruko povezane liste imaju funkcije-članove za sortiranje
- Tip elemenata za sortiranje ustanovljava se na osnovu iteratora, koji definišu niz koji treba sortirati, npr.

```
std::vector<int> niz {99, 77, 33, 66, 22, 11, 44, 88};  
std::sort(std::begin(niz), std::end(niz));
```

- Prikaz sortiranog niza može se izvršiti u petlji ili kopiranjem elemenata u izlazni tok algoritmom `copy()` i iteratora, npr.

```
std::copy(std::begin(niz), std::end(niz),  
          std::ostream_iterator<int> {std::cout, " "});  
// Rezultat: 11 22 33 44 66 77 88 99
```

# Sortiranje delova niza i opadajući poredak

- Iteratori omogućavaju sortiranje samo *delu niza*, npr. od drugog do preposlednjeg:

```
std::sort(++std::begin(niz), --std::end(niz));
```

- Sortiranje objekata u *opadajućem* poretku postiže se zadavanjem funkcijskog objekta, koji vrši međusobno poređenje elemenata niza, npr.

```
std::sort(std::begin(niz), std::end(niz),
```

```
std::greater<>());
```

- funkcija poređenja `greater<>` vraća rezultat tipa `bool` i ima dva argumenta istog tipa, koji se dobija dereferenciranjem *iteratora* (ili se u njega može implicitno konvertovati)
- funkcija poređenja se može zadati i kao *lambda funkcija*

# Primer 1: Sortiranje niza elemenata tipa string u opadajućem poretku

```
#include <iostream>
#include <string>
#include <deque>
#include <algorithm>
#include <iterator>

int main() {
    // Sortiranje niza elemenata tipa string u opadajućem poretku
    std::deque<std::string> reci{ "jedan", "dva", "devet", "devet",
                                "tri", "cetiri", "pet", "sest" };
    std::sort(std::begin(reci), std::end(reci),
        [](const std::string& s1, const std::string& s2) {
            return s1.front() > s2.front(); });
    std::copy(std::begin(reci), std::end(reci),
        std::ostream_iterator<std::string> {std::cout, " "});
    std::cout << std::endl;
    return 0;
}
```

tri sest pet jedan dva devet devet cetiri

← obrnuti leksikografski poredak

## Primer 2: Sortiranje liste imena u rastućem poretku (1/2)

```
#include <iostream>
#include <string>
#include <vector>      // kontejner vector
#include <iterator>    // iteratori stream i back insert
#include <algorithm>   // algoritam sort()

class ImePrezime {
private:
    std::string ime{};
    std::string prezime{};
public:
    ImePrezime(const std::string& i1, const std::string& i2) : ime(i1), prezime(i2){}
    ImePrezime()=default;
    std::string get_ime() const {return ime;}
    std::string get_prezime() const { return prezime; }

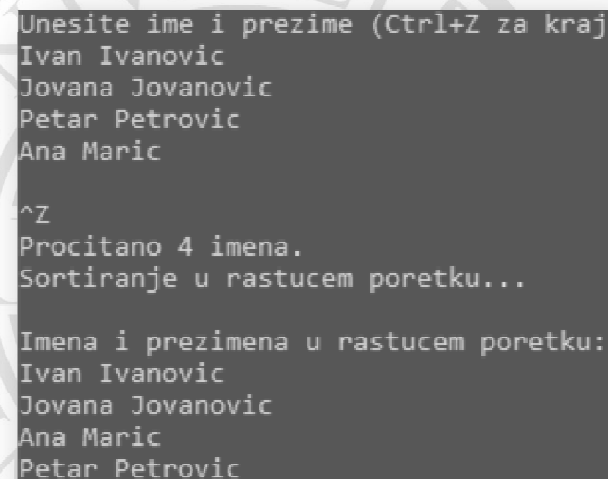
    friend std::istream& operator>>(std::istream& in, ImePrezime& ime);
    friend std::ostream& operator<<(std::ostream& out, const ImePrezime& ime);
};

// Ucitavanje objekata ImePrezime
inline std::istream& operator>>(std::istream& in, ImePrezime& ime) {
    return in >> ime.ime >> ime.prezime;
}
```

## Primer 2: Sortiranje liste imena u rastućem poretku (2/2)

```
// Prikaz objekata klase ImePrezime
inline std::ostream& operator<<(std::ostream& out, const ImePrezime& ime) {
    return out << ime.ime << " " << ime.prezime;
}

int main() {
    std::vector<ImePrezime> imena;
    std::cout << "Unesite ime i prezime (Ctrl+Z za kraj):";
    std::copy(std::istream_iterator<ImePrezime>(std::cin),
              std::istream_iterator<ImePrezime>(),
              std::back_inserter_iterator<std::vector<ImePrezime>>(imena));
    std::cout << "Procitano " << imena.size() << " imena.\n"
              << "Sortiranje u rastucem poretku...\n";
    // Sortiranje po prezimenima u rastucem poretku
    std::sort(std::begin(imena), std::end(imena),
              [](const ImePrezime& i1, const ImePrezime& i2){
                  return i1.get_prezime() < i2.get_prezime(); });
    // Prikaz sortirane liste imena i prezimena
    std::cout << "\nImena i prezimena u rastucem poretku:\n";
    std::copy(std::begin(imena), std::end(imena),
              std::ostream_iterator<ImePrezime>(std::cout, "\n"));
    return 0;
}
```



```
Unesite ime i prezime (Ctrl+Z za kraj)
Ivan Ivanovic
Jovana Jovanovic
Petar Petrovic
Ana Maric
^Z
Procitano 4 imena.
Sortiranje u rastucem poretku...

Imena i prezimena u rastucem poretku:
Ivan Ivanovic
Jovana Jovanovic
Ana Maric
Petar Petrovic
```

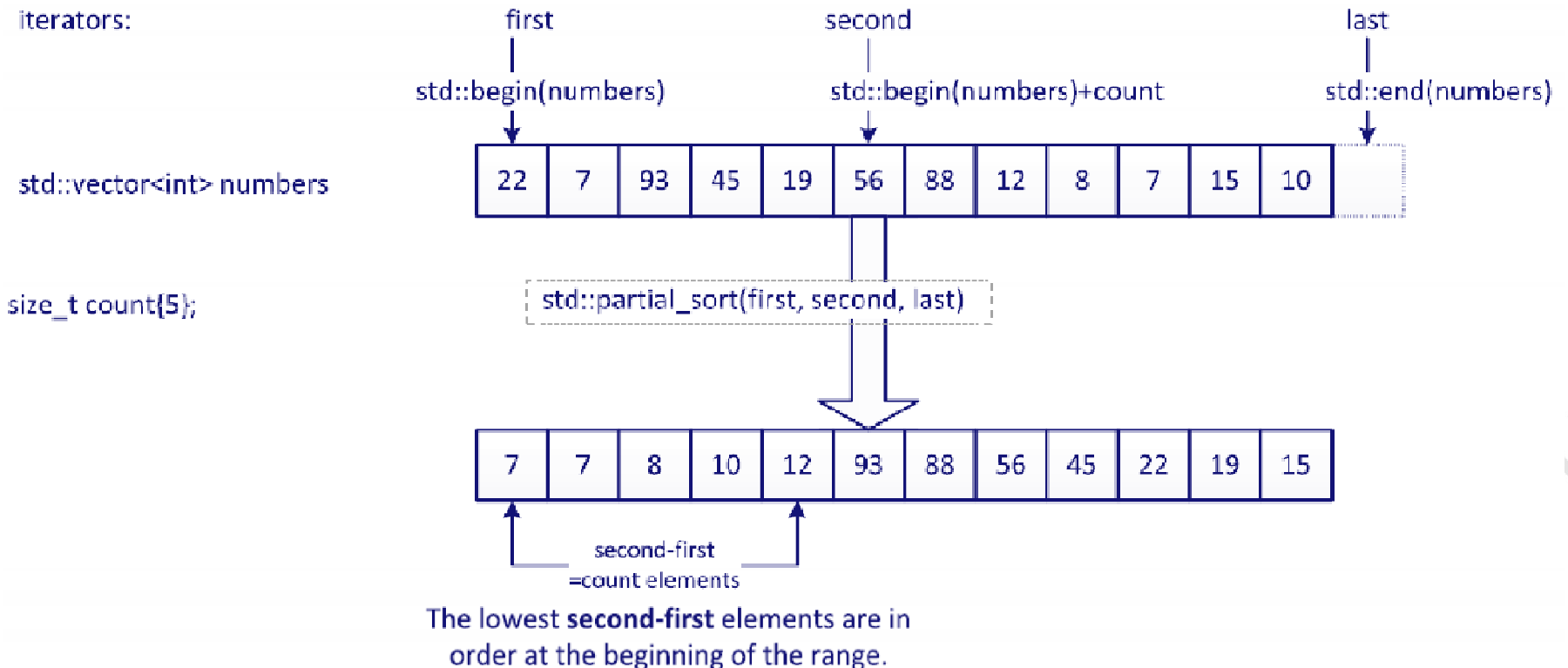
## 2.2 Poredak jednakih elemenata

- Standardni algoritam sortiranja *ne garantuje stabilnost*, tako da međusobno jednaki elementi niza ne zadržavaju isti poredak, što može imati neželjene posledice
  - poredak istih elemenata može da nosi neku dodatnu informaciju, koji je u nekim primenama veoma važan. Npr. promena redosleda transakcija istog klijenta može da dovede do prepisivanja podataka
- Algoritam `stable_sort()` garantuje očuvanje originalnog redosleda elemenata koji imaju isti ključ
- Postoje dve verzije algoritma, jedna koja ima dva argumenta za opis niza i druga, koja ima dodatni argument za definisanje načina međusobnog poređenja elemenata



## 2.3 Parcijalno sortiranje

- Algoritam `partial_sort()` omogućava efikasnije dobijanje sortiranog niza od  $n$  najmanjih elemenata iz obimnog niza od  $N$  elemenata,  $n \ll N$  (originalni redosled jednakih se ne čuva)





# Parcijalno sortiranje

- Algoritam parcijalnog sortiranja ima tri argumenta, iteratora s direktnim pristupom

- Primer upotrebe:

```
size_t n {5};    // broj elemenata koji se sortira
std::vector<int>niz{22,7,93,45,19,56,88,12,8,7,15,10};
std::partial_sort(std::begin(niz), std::begin(niz) + n,
                  std::end(niz));
```

- Dodatni argument se koristi za drugačiji poredak elemenata:

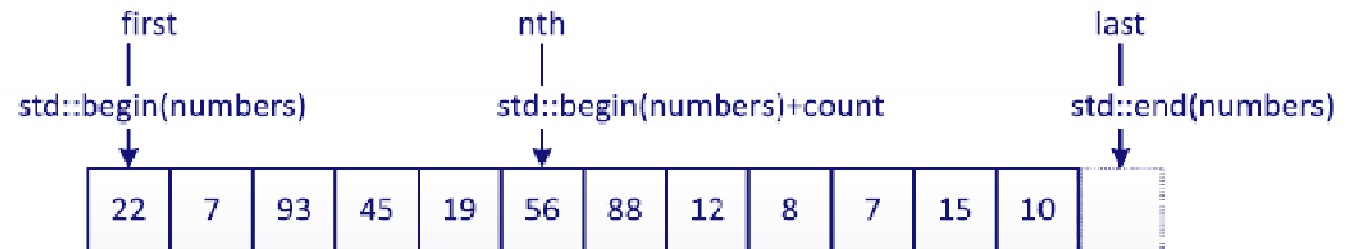
```
std::partial_sort(std::begin(niz), std::begin(niz) + n,
                  std::end(niz), std::greater<>());
```

- algoritam `partial_sort_copy()` kreira u drugom kontejneru kopiju sortiranog dela niza

## 2.4 N-ti element sortiranog niza

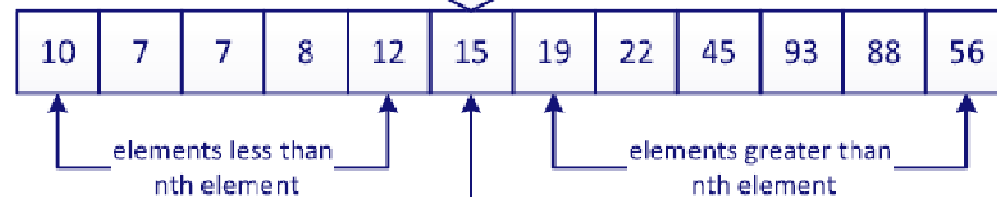
- Algoritam `nth_element()` primenjuje se na niz definisan prvim i trećim argumentom, dok je drugi argument iterator, koji pokazuje na  $n$ -ti element sortiranog niza

iterators:



`size_t count{5};`

`std::nth_element(first, nth, last)`



This is the element that would be here if the range was fully sorted

## 2.5 Provera sortiranosti nizova

- Ustanovljavanje da li je neki niz već sortiran omogućava izbegavanje nepotrebnih operacija sortiranja
- Funkcija `is_sorted()` vraća *true* ako su elementi niza već sortirani u rastućem poretku. Ako se upotrebi dodatni argument i zada drugačija funkcija poređenja, npr. `greater<>()`, može se proveriti da li je niz sortiran u opadajućem poretku
- Funkcija `is_sorted_until()` vraća iterator koji predstavlja *gornju granicu* niza sortiranog u rastućem redosledu, odnosno prvi element koji je manji od svog prethodnika (ili kraj niza)

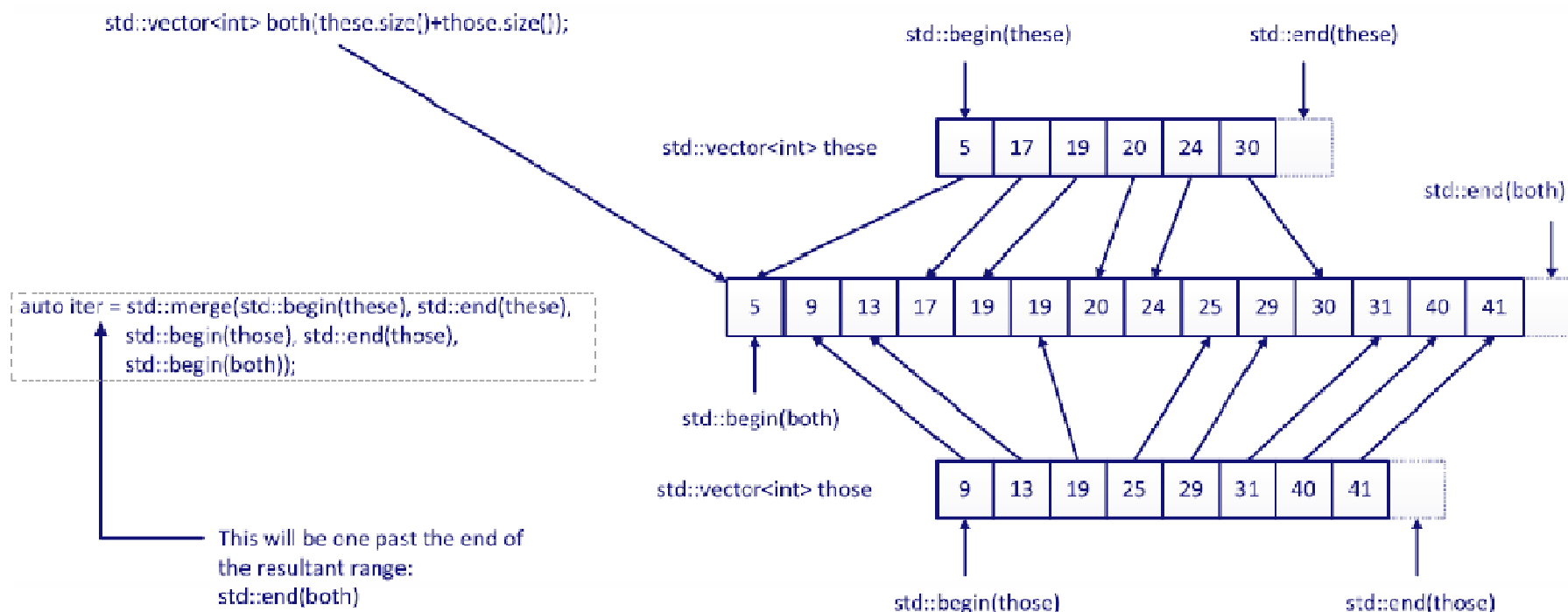
## 3. Stapanje (ažuriranje) nizova metodima STL biblioteke

1. Stapanje (ažuriranje) nizova
2. Metod spajanja merge()
3. Metod spajanja inplace\_merge()



## 3.1 Stapanje (ažuriranje) nizova

- Operacija stapanja (*merge*) kombinuje elemente dva niza, uređena na isti način, u opadajućem ili rastućem poretku
- Rezultat je novi niz, koji sadrži kopije elemenata dva niza u istom poretku (koristi se operator `<` za poređenje elemenata)



## 3.2 Metod spajanja merge()

- Algoritam merge( ) očekuje pet argumenata, koji su iteratori
  - prva četiri argumenta definišu dva niza koji se stapaju - po dva za prvi i drugi niz, a peti argument je iterator koji definiše element kontejnera u koji se smešta prvi element objedinjenog niza
  - ulazni nizovi se ne smeju preklapati (nepredvidivi rezultati)
- Algoritam nema informaciju o kontejneru objedinjenog niza i ne može da kreira njegove elemente, već samo smešta postojeće. To se obezbeđuje kreiranjem objedinjenog niza koji ima broj elemenata jednak zbiru broja elemenata ulaznih nizova ili automatski, pomoću insert iteratora
- Algoritam vraća iterator, koji pokazuje na poslednji element objedinjenog niza (peti argument u pozivu funkcije)

# *Funkcije poređenja Standardne biblioteke*

- Može se koristiti i drugačija funkcija poređenja, koja se zadaje kao šesti argument, npr. `greater<>` iz zaglavlja **<functional>**
- Ovo zaglavlje Standardne biblioteke sadrži *funkcionalne objekte* za aritmetičke, bit i logičke operacije, funkcije negacije, pretraživanja (npr. algoritam Boyer-Moore) i *poređenja*:

- |                              |                                  |
|------------------------------|----------------------------------|
| – <code>equal_to</code>      | realizuje <code>x == y</code>    |
| – <code>not_equal_to</code>  | realizuje <code>x != y</code>    |
| – <code>greater</code>       | realizuje <code>x &gt; y</code>  |
| – <code>less</code>          | realizuje <code>x &lt; y</code>  |
| – <code>greater_equal</code> | realizuje <code>x &gt;= y</code> |
| – <code>less_equal</code>    | realizuje <code>x &lt;= y</code> |



# Primer: Stapanje dva niza u opadajućem poretku

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <functional>

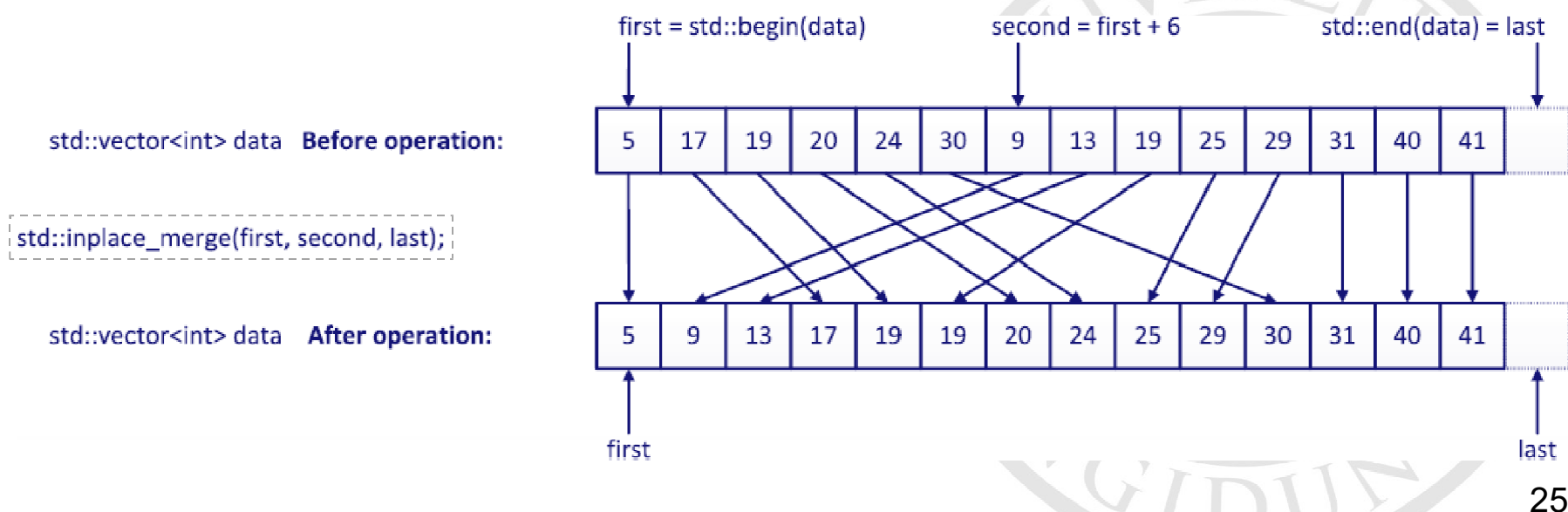
int main() {
    std::vector<int> niz1 {2, 15, 4, 11, 6, 7};    // prvi niz
    std::vector<int> niz2 {5, 2, 3, 2, 14, 11, 6}; // drugi niz
    // Sortiranje nizova u opadajućem poretku
    std::stable_sort(std::begin(niz1), std::end(niz1), std::greater<>());
    std::stable_sort(std::begin(niz2), std::end(niz2), std::greater<>());
    // Kreiranje objedinjenog niza dovoljne veličine
    std::vector<int> niz3 (niz1.size() + niz2.size() + 10);
    // Stapanje dva niza u treći, u istom poretku
    auto end_iter = std::merge(std::begin(niz1), std::end(niz1),
                               std::begin(niz2), std::end(niz2),
                               std::begin(niz3), std::greater<>());
    // Prikaz rezultata
    std::copy(std::begin(niz3), end_iter,
              std::ostream_iterator<int>{std::cout, " "});
    return 0;
}
```

15 14 11 11 7 6 6 5 4 3 2 2 2



## 3.3 Metod spajanja inplace\_merge()

- Algoritam `inplace_merge()` objedinjava dva uzastopna sortirana niza elemenata u istom nizu u kome se nalaze
  - metod očekuje tri parametra, *prvi*, *drugi* i *poslednji*, koji predstavljaju bidirekzione iteratore. Parametri definišu dva desno otvorena niza elemenata: `[ prvi, drugi )` i `[ drugi, poslednji )`
  - rezultat spajanja je niz `[ prvi, poslednji )`



## 4. Primeri

1. Pretraživanje i sortiranje: anagrami
2. Obrada kreditnih i debitnih transakcija



## 4.1 Pretraživanje i sortiranje: lista anagrama

- **Anagrami** su nizovi reči (izrazi) koji se dobiju permutacijama slova drugih nizova reči, npr. "logaritam" je anagram reči "algoritam"
- Program koji za zadani skup reči vraća skup svih mogućih anagrama tih reči zasniva se na proveru da li je neka reč anagram neke druge reči
- Provera da li je jedna reč anagram druge reči može se efikasno realizovati poređenjem stringova koji se dobiju *sortiranjem* svake od reči, npr. i "logaritam" i "algoritam" nakon sortiranja daju isti string "aagilmort"

# Pretraživanje i sortiranje: lista anagrama

- **Algoritam pronalaženja anagrama** neke reči u zadanom skupu reči poredi svaki string sa ostalim stringovima iz zadanog niza reči. Kad se pronađe string koji je anagram, nije ga potrebno ponovo testirati
- Sortirani string se može koristiti kao ključ *heš tabele* (kontejner `unordered_map`). Vrednost u tabeli je polje stringova iz ulaznog niza
- Računanje se sastoji od  $n$  umetanja u heš tabelu
- Sortiranje ključeva ima složenost  $O(n \cdot m \log m)$ , a umetanje  $O(n \cdot m)$ , gde je  $n$  broj stringova i  $m$  maksimalna dužina stringa. Složenost algoritma pronalaženja anagrama je  $O(n^2 \cdot m \log m)$

# Lista anagrama (program)

```
#include <iostream>           // tokovi podataka
#include <vector>              // kontejner vector
#include <unordered_map>       // kontejner unordered_map
#include <iterator>            // iteratori
#include <algorithm>           // algoritam sort()
#include <string>              // klasa string
using std::vector;
using std::string;
using std::unordered_map;


// Funkcija pronalazi anagrame u zadanom skupu reci

vector<vector<string>> PronadjiAnagrame(const vector<string>& recnik) {
    std::unordered_map<string, vector<string>> sortiraniStrAnagrami;

    for (const std::string& s : recnik) {
        // Sortira string i koristi ga kao kljuc za dodavanje
        // originalnog stringa kao vrednosti u hes tabelu
        string sortiraniStr(s);
        sort(sortiraniStr.begin(), sortiraniStr.end());
        sortiraniStrAnagrami[sortiraniStr].emplace_back(s);
    }
}
```

# Lista anagrama (program)

```
vector<vector<string>> grupAnagrama;  
for (const auto& p : sortiraniStrAnagrami)  
    if (p.second.size() >= 2) // ako ima više od jedne reči  
        grupAnagrama.emplace_back(p.second); // pronadjen anagram  
return grupAnagrama;  
  
}  
  
int main() {  
  
    // Skup reci u kojima se traže anagrami  
    vector<string> reci { "debitcard", "elvis", "silent", "badcredit",  
                          "lives", "freedom", "listen", "levis", "money" };  
  
    vector<vector<string>> anagrami = PronadjiAnagrame(reci);  
  
    // Prikaz liste anagrama  
    for (auto& grupa : anagrami) {  
        for (std::string& rec : grupa)  
            std::cout << rec << ' ';  
        std::cout << std::endl;  
    }  
    return 0;  
}
```



```
debitcard badcredit  
elvis lives levis  
silent listen
```

## 4.2 Obrada kreditnih i debitnih transakcija

- *dodaće se naknadno*



# Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
5. Horton I., *Beginning C++*, Apress, 2014
6. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
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. Vikipedija [www.wikipedia.org](http://www.wikipedia.org)
10. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

