Tema 10
Iteratori u jeziku C++

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2

Sadržaj

1. Uvod

- Pojam iteratora u jeziku C++
- Ponašanje iteratora
- Osnovni metodi iteratora

Pojam iteratora u jeziku C++

- Pronalaženje i obrada nizova podataka u starijim verzijama jezika C++ realizovala se korišćenjem *for* petlje
- Pozicija elemenata u nizovima čuvala se u posebnim promenljivima
- Od verzije jezika C++11 uvode se iteratori, objekti koji omogućavaju realizaciju obilaska različitih STL kontejnerskih struktura, uz istovremeno pamćenje pozicije elemenata
- Iteratori omogućavaju različite načine obilaska celih struktura ili njihovih delova, element po element

Ponašanje iteratora

- Ponašanje iteratora definiše se pomoću osnovnih operatora:
  - `*` vraća element na tekućoj poziciji; ako objekt ima članove, pristupa im se pomoću operatora `->`
  - `++` pomera iterator na sledeći element; većina iteratora omogućava prelazak na prethodni element pomoću operatora `--`
  - `==` proverava da li dva iteratora pokazuju na istu poziciju (operator `!=` proverava da li su pozicije različite)
  - `=` dodeljuje vrednost iteratora, u stvari *poziciju* na koju pokazuje
- Iteratori obezbeđuju isti interfejs i različito ponašanje za različite strukture kontejnera na koji se odnose
- Zavisnost od strukture podataka lako se implementira pomoću *šablona* (templejta)

Osnovni metodi iteratora

- Osnovni metodi iteratora definišu poluotvoreni interval elemenata kontejnera:
  - `begin()` - pozicija početka kontejnera, odnosno prvog elementa kontejnera, ako postoji
  - `end()` - pozicija kraja kontejnera, odnosno pozicija *iza* poslednjeg elemenata kontejnera
- Za svaki tip kontejnera definisana su dva tipa iteratora:
  - `container::iterator` - za obilazak elemenata kontejnera radi čitanja i pisanja (*read/write*)
  - `container::const_iterator` - za obilazak elemenata kontejnera samo radi čitanja (*read only*)

2. Kontejneri i iteratori

1. Kontejneri i iteratori
2. Iteratori asocijativnih kontejnera
3. Obilazak kontejnera

2.1 Kontejneri i iteratori

- Pristup tekućem elementu kontejnera vrši se pomoću iteratora `*pozicija`, a prelazak na sledeći pomoću operatora `++` (prefiksno ili postfiksno)
- Prefiksna upotreba operatora inkrementa (npr. `++pozicija`) može biti efikasnija, jer ne zahteva kreiranje privremenog objekta koji pamti i vraća tekuću poziciju elementa
- Tip iteratora se ne mora eksplicitno definisati, već se može ustanoviti na osnovu prve vrednosti elementa korišćenjem ključne reči `auto`, npr.

```
for (auto poz=lista.begin(); poz!=lista.end(); ++poz) {
  cout << *poz << ' ';
}
```

Primer: Prikaz svih elemenata kontejnera

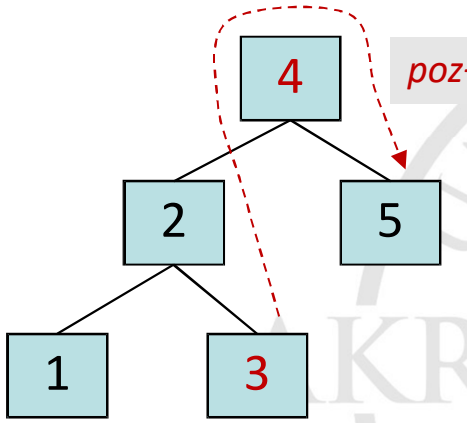
```
#include <list>
#include <iostream>
using namespace std;

int main() {

  list<char> lista; // kontejner liste elemenata tipa char

  // Formiranje liste elemenata 'a'-'z'
  for (char c='a'; c<='z'; ++c) {
    lista.push_back(c);
  }
  // Prikaz svih elemenata liste
  list<char>::const_iterator poz; // iterator liste (čitanje)
  for (poz = lista.begin(); poz != lista.end(); ++poz) {
    cout << *poz << ' ';
  }
  cout << endl;
}
```

Rezultat:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

2.2 Iteratori asocijativnih kontejnera

- Osim kontejnera sekvenci, iteratori se koriste i za obilazak *asocijativnih* kontejnera, kao što je *skup* (`set`)
- Npr. nakon dodavanja elemenata `1,2,3,4,5` u skup (u proizvoljnom redosledu) elementi će biti interno sortirani, tako da je uvek desni element veći od levog:



  - *Napomena*: kontejner `set` ne dopušta duplikate elemenata skupa. Asocijativni kontejner `multiset` može da sadrži više identičnih elemenata

Primer: Obilazak elemenata kontejnera *set* (1)

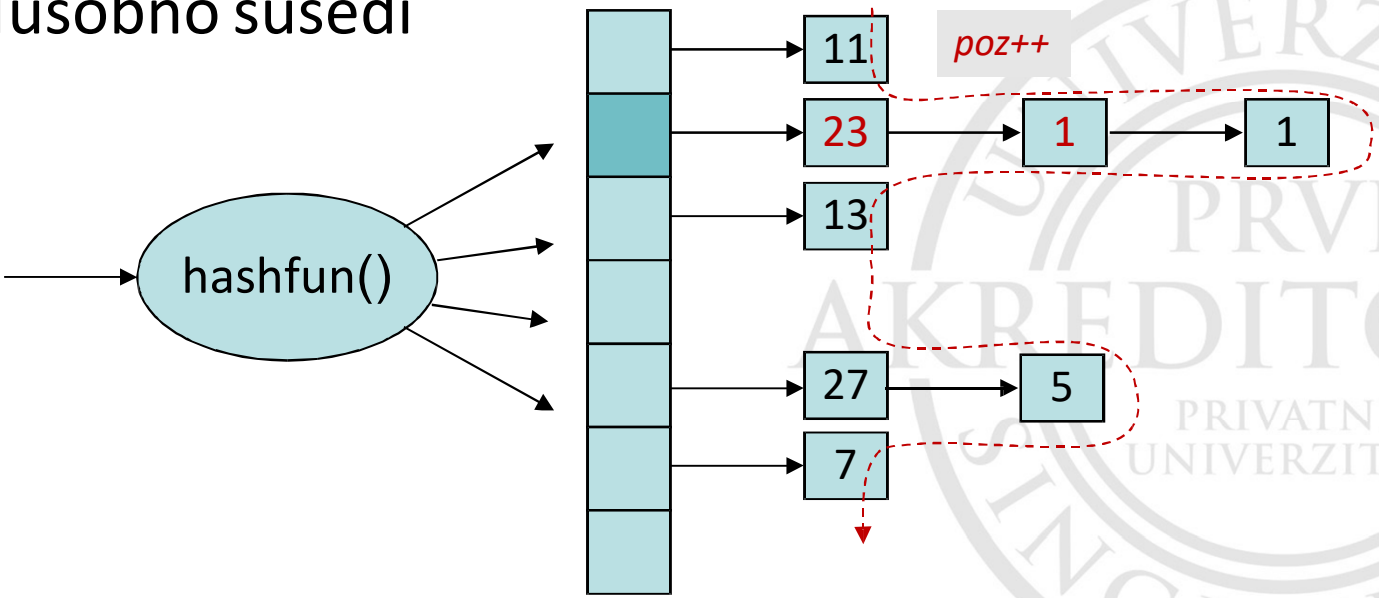
```
#include <list>
#include <set>
#include <iostream>

int main() {
  // Definisanje skupa elemenata
  typedef std::set<int> SkupCelih; // set<int,greater<int>>
  SkupCelih skup;                  // kontejner skupa celobrojnih elemenata

  // Dodavanje 6 elemenata u proizvoljnom redosledu
  skup.insert(3);
  skup.insert(1);
  skup.insert(5);
  skup.insert(4);
  skup.insert(1); // element 1 drugi put
  skup.insert(2);

  // Prikaz svih elemenata skupa
  SkupCelih::const_iterator poz;
  for (poz = skup.begin(); poz != skup.end(); ++poz) {
    std::cout << *poz << ' ';
  }
  std::cout << std::endl;
}
```

Rezultat:

```
1 2 3 4 5
```

Iteratori asocijativnih kontejnera

- Neki kontejneri nemaju definisan poredak elemenata, čak i kad se dodaju prethodno sortirani elementi, npr. asocijativni kontejner `unordered_multiset`
- Dodavanje novog elementa u kontejner menja poredak postojećih elemenata, s tim da su identični elementi uvek međusobno susedi



2.3 Obilazak kontejnera

- Primena iteratora može biti nezavisna od kontejnera, npr.

```
for (auto poz=kont.begin(); poz!=kont.end(); ++poz){
  ...
}
```

- Upotreba redosleda elemenata ne važi za sve kontejnere, npr.

```
for (auto poz=kont.begin(); poz<kont.end(); ++poz){
  ...
}
```

nije odgovarajući za *liste*, *skupove* i *mape*, za koje *redosled* elemenata nije definisan, tako da se ne može se ustanoviti pozicija kraja kontejnera

– greška prevođenja

Obilazak kontejnera

- Obilazak i prikaz elemenata kontejnera može se realizovati sekvencom

```
for (auto elem : kont) {
  std::cout << elem << ' ';
}
```

koja je kraća od standardnog načina

```
for (auto poz=kont.begin(); poz!=kont.end(); ++poz){
  auto elem = *poz;
  std::cout << elem << ' ';
}
```

- Svaki kontejner definiše sopstvene tipove iteratora, tako da je posebno zaglavlje `<iterator>` potrebno samo za specijalne iteratore, kao što su reverzni i spoljašnji iteratori

Primer: Obilazak elemenata kontejnera *set* (2)

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

int main() {
  // Definisanje skupa elemenata
  typedef std::set<int> SkupCelih; // set<int,greater<int>>
  SkupCelih skup;                  // kontejner skupa celobrojnih elemenata

  // Dodavanje 6 elemenata u proizvoljnom redosledu
  skup.insert(3);
  skup.insert(1);
  skup.insert(5);
  skup.insert(4);
  skup.insert(1); // element 1 drugi put
  skup.insert(2);

  // Prikaz svih elemenata skupa s auto iteratorom
  for (auto element : skup) {
    std::cout << element << ' ';
  }
  std::cout << std::endl;
}
```

Rezultat:

```
1 2 3 4 5
```

3. Kategorije iteratora

1. Izlazni iteratori (*output*)
2. Ulazni iteratori (*input*)
3. Jednosmerni iteratori (*forward*)
4. Bidirekcioni iteratori (*bidirectional*)
5. Iteratori s direktnim pristupom (*random-access*)

Vrste iteratora

- Iteratori imaju neka opšta svojstva, kao i svojstva koja zavise od *tipa kontejnera* i mogu biti:
  1. Jednosmerni (*forward*) - omogućavaju iteraciju samo u jednom smeru pomoću operatora inkrementa
     - za klase kontejnera: `forward_list`, `unordered_set`, `unordered_multiset`, `unordered_map` i `unordered_multimap`
  2. Dvosmerni (*bidirectional*) - omogućavaju iteraciju u oba smera, unapred i unazad, koristeći operatore inkrementa i dekrementa
     - za klase kontejnera `list`, `set`, `multiset`, `map` i `multimap`
  3. Iteratori s direktnim pristupom (*random-access*) - imaju sva svojstva bidirekcionih iteratora, ali omogućavaju i direktni pristup elementima
     - za klase kontejnera `vector`, `deque`, `array` i `string`
  4. Izlazni (*output*) - jednosmerni iteratori za izlazne klase tokova
  5. Ulazni (*input*) - jednosmerni za ulazne klase tokova

3.1 Izlazni iteratori (*output*)

- Omogućavaju samo upis pojedinačnih elemenata u jednom smeru, npr. za upis na standardni izlaz
- Ne može se dva puta dodeliti vrednost bez pomeranja iteratora na sledeću poziciju, odnosno upis se vrši kao

```
while (...) {
  *pozicija = ... ; // dodela vrednosti
  ++pozicija;       // pomeranje, za sledeću dodelu
}
```

- Nijedan iterator klase `const_iterators` nije izlazni i ne omogućava upis

Expression	Effect
**iter = val*	Writes *val* to where the iterator refers
++iter	Steps forward (returns new position)
iter++	Steps forward (returns old position)
TYPE(iter)	Copies iterator (copy constructor)

3.2 Ulazni iteratori (*input*)

- Omogućavaju samo čitanje pojedinačnih elemenata u jednom smeru, npr. sa standardnog ulaza
  - ne može se dva puta pročitati vrednost bez pomeranja iteratora na sledeću poziciju
- Operatori `==` i `!=` proveravaju da li je vrednost iteratora jednaka ili različita od pozicije iza poslednjeg elementa, tj.

```
InputIterator poz, end;
while (poz != end) {
   ... // pristup radi
       // čitanja pomoću
       // *poz
   ++poz;
}
```

Expression	Effect
**iter*	Provides read access to the actual element
iter ->member	Provides read access to a member of the actual element
++iter	Steps forward (returns new position)
iter++	Steps forward
iter1 == iter2	Returns whether two iterators are equal
iter1 != iter2	Returns whether two iterators are not equal
TYPE(iter)	Copies iterator (copy constructor)

3.3 Jednosmerni iteratori (*forward*)

- Iteratori s dodatnim osobinama u odnosu na ulazne; dva jednaka jednosmerna iteratora i nakon inkrementiranja pokazuju na istu vrednost, tj.

```
ForwardIterator poz1, poz2;
poz1 = poz2 = begin; // pokazuju na isti element
if (poz1 != end) {
  ++poz1; // poz1 je za jedan element ispred poz2
  while (poz1 != end) {
    if (*poz1 == *poz2) {
      ... // obrada jednakih na
          //susednim pozicijama
    ++poz1;
    ++poz2;
  }
}
```

Expression	Effect
**iter*	Provides access to the actual element
iter->member	Provides access to a member of the actual element
++iter	Steps forward (returns new position)
iter++	Steps forward (returns old position)
iter1 == iter2	Returns whether two iterators are equal
iter1 != iter2	Returns whether two iterators are not equal
TYPE()	Creates iterator (default constructor)
TYPE(iter)	Copies iterator (copy constructor)
iter1 = iter2	Assigns an iterator

- Jednosmerni iteratori se mogu koristiti za `forward_list` i kontejnere bez poretka elemenata

3.4 Bidirekcioni iteratori (*bidirectional*)

- Jednosmerni iteratori s dodatnim svojstvom da omogućavaju iteriranje elemenata *u suprotnom smeru* pomoću operatora dekrementa, i to
  - `--iter` pomera se jedan korak unazad i vraća *novu* poziciju
  - `iter--` pomera se jedan korak unazad i vraća *staru* poziciju
- Dvosmerni iteratori mogu se koristiti za *liste* i *asocijativne* kontejnere

3.5 Iteratori s direktnim pristupom (*random-access*)

- Imaju sva svojstva dvosmernih iteratora, uz mogućnost *direktnog* pristupa
- Podržavaju poređenje i iteratorsku aritmetiku
- Koriste se za kontejnere s direktnim pristupom (`array`, `vector`, `deque`), stringove i standardna fiksna *C*-polja

Expression	Effect
iter[*n*]	Provides access to the element that has index *n*
iter+=*n*	Steps *n* elements forward (or backward, if *n* is negative)
iter−=*n*	Steps *n* elements backward (or forward, if *n* is negative)
iter+*n*	Returns the iterator of the *n*th next element
n+*iter*	Returns the iterator of the *n*th next element
iter−*n*	Returns the iterator of the *n*th previous element
iter1−*iter2*	Returns the distance between *iter1* and *iter2*
iter1<*iter2*	Returns whether *iter1* is before *iter2*
iter1>*iter2*	Returns whether *iter1* is after *iter2*
iter1<=*iter2*	Returns whether *iter1* is not after *iter2*
iter1>=*iter2*	Returns whether *iter1* is not before *iter2*

Primer: Upotreba iteratora s direktnim pristupom (1/2)

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

int main() {
  vector<int> kont;
  // Umetanje elemenata -3 do 9
  for (int i=-3; i <= 9; ++i) {
    kont.push_back (i);
  }
  // Prikaz broja elemenata na osnovu distance između početka i kraja
  // - koristi operator '-' za iteratore
  cout << "Broj elemenata/distanca: " << kont.end() - kont.begin() << endl;
  // Prikaz svih elemenata
  // - koristi operator '<' umesto '!='
  vector<int>::iterator poz;
  for (poz= kont.begin(); poz < kont.end(); ++poz) {
    cout << *poz << ' ';
  }
  cout << endl;
```

Rezultat:

```
Broj elemenata/distanca: 13
-3 -2 -1 0 1 2 3 4 5 6 7 8 9
```

Primer: Upotreba iteratora s direktnim pristupom (2/2)

```
  // Prikaz svih elemenata
  // - koristi operator [] umesto operatora *
  for (int i=0; i < kont.size(); ++i) {
    cout << kont.begin()[i] << ' ';
  }
  cout << endl;
  // Prikaz svakog drugog elementa
  // - koristi operator +=
  for (poz = kont.begin(); poz < kont.end()-1; poz += 2) {
    cout << *poz << ' ';
  }
  cout << endl;
}
```

Rezultat:

```
Broj elemenata/distanca: 13
-3 -2 -1 0 1 2 3 4 5 6 7 8 9
-3 -2 -1 0 1 2 3 4 5 6 7 8 9
-3 -1 1 3 5 7
```

4. Spoljašnje iteratorske funkcije

1. Pomeri (*advance*)
2. Sledeći (*next*) i prethodni (*prev*)
3. Razmak (*distance*)
4. Zamena (*iter_swap*)

4.1 Pomeri (*advance*)

- Standardna biblioteka obezbeđuje nekoliko funkcija za rad s iteratorima

  `advance(),next(),prev(),distance(),iter_swap()`

- Fukcija `advance()` menja poziciju zadanog iteratora unapred i unazad za zadani broj elemenata:

```
#include <iterator>
void advance (InputIterator& poz, Dist n)
```

Primer: Upotreba spoljašnje iteratorske funkcije advance()

```
#include <iterator>
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

int main() {

 list<int> kont;

  // Umetanje elemenata 1 do 9
  for (int i=1; i<=9; ++i) {
    kont.push_back(i);
  }
  list<int>::iterator poz = kont.begin();

  // Prikaz tekućeg elementa
  cout << *poz << endl;

  // Premeštanje za tri elementa unapred
  advance(poz, 3);

  // Prikaz tekućeg elementa
  cout << *poz << endl;

  // Premeštanje za jedan element unazad
  advance(poz, -1);

  // Prikaz tekućeg elementa
  cout << *poz << endl;

}
```

Rezultat:

```
1
4
3
```

4.2 Sledeći (*next*) i prethodni (*prev*)

- Premešta iterator na poziciju sledećeg `next()`, odnosno prethodnog elementa `prev()`
- Premeštanje može biti za jedan ili za zadani broj elemenata (pozicija)
- Za bidirekcione iteratore, zadani broj pozicija može biti *negativan*, za premeštanje u suprotnom smeru

4.3 Razmak (*distance*)

- Funkcija `distance()` vraća razliku pozicija dva iteratora u okviru jednog kontejnera (razmak, rastojanje)
- Npr.

```
list<int> kontejner;
// Umetanje elemenata -3 do 9
for (int i=-3; i<=9; ++i) {
  kontejner.push_back(i);
}

// Pozicija elementa 5
list<int>::iterator pos;
poz = find(kontejner.begin(), kontejner.end(), 5); // poz. 5

// Razmak -3..5
cout << distance(kontejner.begin(),poz) << endl;    // d = 8
```

4.4 Zamena (*iter_swap*)

- Funkcija omogućava međusobnu zamenu vrednosti elemenata jednog kontejnera na pozicijama na koje pokazuju dva iteratora
- Npr. zamena vrednosti prvog i drugog elementa

```
iter_swap( kontejner.begin(), next(kontejner.begin()) );
```

5. Iteratorski adapteri

1. Iteratori umetanja (*insert*)
2. Iteratori tokova (*stream*)
3. Reverzni iteratori (*reverse*)
4. Iteratori premeštanja (*move*)

5.1 Iteratori umetanja (*insert*)

- Iteratori umetanja (*inserters*) zasnivaju se na standardnim iteratorima, koji pristupaju *postojećim* elementima kontejnera, ali se koriste za dodavanje *novih* elemenata
- Inserteri umesto prepisivanja elementa *dodaju* novi element
  - `back_insert_iterator` dodaje nove elemente na kraj kontejnera koji imaju metod `push_back()`
  - `front_insert_iterator` dodaje nove elemente na početak kontejnera koji imaju metod `push_front()`
  - `insert_iterator` se može koristiti za umetanje elemenata na bilo koje mesto u kontejnere koji imaju funkciju člana `insert()`

Expression	Effect
**iter*	No-op (returns *iter*)
iter = value	Inserts *value*
++iter	No-op (returns *iter*)
iter++	No-op (returns *iter*)

Primer: Opšti iterator umetanja (inserter)

```
#include <set>
#include <list>
#include <algorithm>
#include <iterator>
using namespace std;

template <typename T>
inline void printElems(const T& kont, const std::string& txt=""){
  std::cout << txt;
  for (const auto& elem : kont) {
    std::cout << elem << ' ';
  }
  std::cout << std::endl;
}

int main(){
  set<int> kont;
```

Primer: Opšti iterator umetanja (inserter)

```
// Kreiranje insert iteratora za kontejner kont
//  (neodgovarajući  način)
insert_iterator<set<int>> iter(kont, kont.begin());

// Umetanje elemenata s uobičajenim intefejsom iteratora
*iter = 1;
iter++;
*iter = 2;
iter++;
*iter = 3;
printElems(kont,"set: ");

// Kreiranje insertera i umetanje elemenata
// (odgovarajući način)
inserter(kont, kont.end()) = 44;
inserter(kont, kont.end()) = 55;
printElems(kont,"set: ");
```

Rezultat:

```
set: 1 2 3
set: 1 2 3 44 55
```

Primer: Opšti iterator umetanja (inserter)

```
  // Upotreba insertera za umetanje svih elemenata u listu
  list<int> kont2;
  copy (kont.begin(), kont.end(),          // koji elementi
        inserter(kont2, kont2.begin())); // gde se kopiraju
  printElems(kont2, "list: ");

  // Upotreba insertera za ponovno umetanje svih elemenata
  // u listu ispred drugog elementa
  copy (kont.begin(), kont.end(),            // koji elementi
        inserter(kont2, ++kont2.begin())); // gde se kopiraju
  printElems(kont2, "list: ");
}
```

Rezultat:

```
set: 1 2 3
set: 1 2 3 44 55
list: 1 2 3 44 55
list: 1 1 2 3 44 55 2 3 44 55
```

5.2 Iteratori tokova (*stream*)

- Iteratori tokova su iteratorski adapteri koji omogućavaju korišćenje tokova kao izvor ili odredište algoritama
- Iterator ulaznog toka `istream` može se koristiti za čitanje elemenata iz ulaznog toka, a iterator izlaznog toka `ostream` za upis vrednosti u izlazni tok
- Posebna vrsta iteratora toka je *stream buffer iterator*, koji se može koristiti za direktni upis ili čitanje iz bafera toka

5.3 Reverzni iteratori (*reverse*)

- Reverzni iteratori redefinišu operatore inkrementa i dekrementa da promene smer u odnosu na obične iteratore, tako da algoritmi obrađuju elemente obrnutim redosledom
- Sve kontejnerske klase osim jednosmernih lista i neuređenih kontejnera omogućavaju upotrebu reverznih iteratora, npr.

```
void print (int elem) { cout << elem << ' ' }
int main() {
  list<int> kont = {1,2,3,4,5,6,7,8,9}; // lista elemenata
  // Prikaz svih elemenata u normalnom poretku pomoću alg.
  // for_each(range, fun) i korisničke funkcije print(elem)
  for_each(kont.begin(), kont.end(), print);
  cout << endl;
  // Prikaz u obrnutom poretku zamenom iteratora reverznim
  for_each(kont.rbegin(), kont.rend(), print);
  cout << endl;
}
```

5.4 Iteratori premeštanja (*move*)

- Iteratori premeštanja su iteratorski adapteri koji svaki pristup izabranom elementu pretvaraju u operaciju *premeštanja*, npr.
  - kopiranje stringa s u string v1

```
std::list<std::string> s;
std::vector<string> v1(s.begin(), s.end());
```

  - premeštanje (*move*) stringa s u string v2

```
std::vector<string> v2(make_move_iterator(s.begin()),
                       make_move_iterator(s.end()));
```

- Jedna od primena ovih iteratora je da omoguće algoritmima da umesto kopiranja *premeštaju* elemente iz zadanog opsega
- Pri tome treba obezbediti da se svakom elementu pristupa samo jednom

6. Upotreba iteratora

- Prenosivost koda s iteratorima
- Korisnički definisani iteratori

Prenosivost koda s iteratorima

- Prenosivost koda s iteratorima ostvaruje se izbegavanjem eksplicitne upotrebe operatora inkrementa i dekrementa
- Npr. za strukturu

```
std::vector<int> kontejner;
```

radi sortiranja od *drugog* elementa, umesto operatora ++

```
// Sortiranje od drugog elementa
if (kontejner.size() > 1) {
  std::sort(++kontejner.begin(), kontejner.end());
}
```

bolje je koristiti funkciju next()

```
// Sortiranje od drugog elementa
if (kontejner.size() > 1) {
  std::sort(std:next(kontejner.begin()), kontejner.end());
}
```

Korisnički definisani iteratori

- Korisnički definisani iteratori kreiraju se kao objekti posebne strukture i svojstava, koji nasleđuju osobine odgovarajućih postojećih iteratora, npr.

```
namespace std {
  template <typename T>
    struct iterator_traits {
     typedef typename T::iterator_category iterator_category;
     typedef typename T::value_type value_type;
     typedef typename T::difference_type difference_type;
     typedef typename T::pointer pointer;
     typedef typename T::reference reference;
   };
}
```

- Generičke funkcije omogućavaju prilagođavanje tipova elemenata različitih kontejnera

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. Josuttis N. M. , *The C++ Standard Library*, 2nd Ed, Pearson Education, 2012
8. Web izvori
   - http://www.cplusplus.com/doc/tutorial/
   - http://www.learncpp.com/
   - http://www.stroustrup.com/
9. Vikipedija www.wikipedia.org
10. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019