Objektno orijentisano programiranje 1

Standardna biblioteka

Standardna biblioteka šablona

- Standard jezika C++ definiše biblioteku šablona
  - generičkih funkcija i generičkih klasa
- Biblioteka se zove *Standard Template Library* (STL)
  - STL je deo standardne bibioteke jezika C++
- Svrha je efikasno ostvarenje često korišćenih
  - struktura podataka → šablonske klase
  - postupaka → šablonske funkcije

  koji ne zavise od tipova
  - od kojih se struktura sastoji
  - koji se obrađuju u postupcima
- Svi identifikatori u STL-u su u prostoru imena `std`
  - ili u potprostorima prostora `std`
- Dokumentacija: https://www.cplusplus.com/reference/

Grupe šablona

- Uslužne funkcije i klase
  - relacioni operatori, funkcijske klase
- Jednostavne strukture podataka
  - par, kompleksan broj
- Zbirke podataka i iteratori
  - vektor, red, lista, stek, mapa, skup, tekst, niz bitova
- Algoritmi
  - pretraživanje (*search*), uređivanje (*sort*), spajanje *(merge)*

Šabloni relacionih operatora

- Podrška za automatizaciju generisanja relacionih operatora
- Postoji 6 relacionih operatora: `==` `!=` `<` `>` `<=` `>=`
  - 4 od 6 mogu da se izraze pomoću 2: `==` i `<` (i logičkih)
- STL definiše 4 izvedena operatora na osnovu 2
  - kao globalne šablonske funkcije
- Programeri treba samo da preklope `==` i `<`
  - ostala 4 operatora se generišu iz šablona
- Nalaze se u zaglavlju `<utility>`, u prostoru `std::rel_ops`
- Deklaracije:

```
template<typename T> bool operator@@(const T&, const T&);
```

gde umesto `@@` stoji neki od operatora: `!=`, `>`, `<=`, `>=`

Primer šablona relacionih operatora

- Pravougaonici se porede na osnovu površina

```
class Pravoug {
  float a,b;
public:
  Pravoug(float str_a=1, float str_b=1){a=str_a; b=str_b;}
  float P()const{return a*b;}
};
bool operator==(Pravoug x, Prvoug y) {return x.P()==y.P();}
bool operator<(Pravoug x, Pravoug y) {return x.P()<y.P();}
#include <utility>
using namespace std::rel_ops;
void main(){Pravoug p1,p2(2,1);
     if(p1>p2){…} //generise se >
}
```

Funkcijske klase/strukture

- Funkcijska klasa (FK):
  - klasa koja realizuje preklapanje operatorske funkcije `operator()`
- Funkcijska struktura (FS) se realizuje na isti način
- Šabloni u jeziku C++ ne mogu da se parametrizuju funkcijama
- Parametar šablona može da bude klasa/struktura,
  a to može da bude FK/FS
- FK/FS može da bude i generička (šablonska) klasa/struktura
- U STL-u neke generičke klase i funkcije traže FK/FS kao argumente
- FK/FS može da definiše korisnik, ali u STL-u postoji niz gotovih FS
- FS su definisane u zaglavlju `<functional>`
  - postoje unarne i binarne aritmetičke, relacione i logičke operacije

Opšti oblici funkcijskih struktura

- Unarna aritmetička operacija

```
template <typename T> struct ime{
      T operator()(const T&) const;}
```

- Binarna aritmetička operacija

```
template <typename T> struct ime{
      T operator()(const T&, const T&) const;}
```

- Unarna logička operacija

```
template <typename T> struct ime{
      bool operator()(const T&) const;}
```

- Binarna logička ili relaciona operacija

```
template <typename T> struct ime{
      bool operator()(const T&, const T&) const;}
```

Nazivi funkcijskih struktura

- Unarna aritmetička operacija:
  - `negate`
  - realizuje se pomoću unarnog operatora `-`
  - potrebno je da je `operator-()` realizovan za dati tip (argument)
- Binarne aritmetičke operacije:
  - `plus`, `minus`, `multiplies`, `divides`, `modulus`
  - realizuju se pomoću: `+`, `-`, `*`, `/`, `%`
- Relacijske operacije:
  - `equal_to`, `not_equal_to`, `greater`, `less`, `greater_equal`, `less_equal`
  - realizuju se pomoću: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Binarne logičke operacije:
  - `logical_and`, `logical_or`
  - realizuju se pomoću: `&&`, `||`
- Unarna logička operacija:
  - `logical_not`
  - realizuje se pomoću: `!`

Korišćenje funkcijskih struktura

- Navedene generičke FS
  - mogu da se koriste za standardne proste tipove
  - za klasne tipove je neophodno preklopiti odgovarajuće operatore
- U navedenim generičkim FS
  - operatorska funkcija `operator()` se neposredno ugrađuje u kod
  - ne gubi se na efikasnosti
- Često se kao podrazumevana vrednost argumenta bibliotečkih klasa navodi neka od navedenih bibliotečkih funkcijskih struktura

Primer

- Popunjavanje niza primenom operacije na elemente dva postojeća niza

```
#include <iostream>
#include <functional>
using namespace std;
template <typename T, class Op>
   void radi(const T a[], const T b[], T c[], int n){
     Op op; for(int i =0; i<n; i++) c[i]=op(a[i],b[i]);
   }
template<typename T> struct Zbir_kvadrata{
   T operator() (const T& a, const T& b) const{return a*a+b*b;}
};
void main(){
   int a[10],b[10],c[10],n; cin>>n; citaj(a,n); citaj(b,n);
   radi<int, plus<int> >(a,b,c,n); pisi(c,n);
   radi<int, Zbir_kvadrata<int> >(a,b,c,n); pisi(c,n);
}
```

Parovi podataka

- Par – dva podatka proizvoljnih tipova
- Definicija u zaglavlju `<utility>`

```
template<typename A, typename B> struct pair{
     A first; B second;
     pair():first(A()),second(B()){}
     pair(const A&, const B&);
     template<typename C, typename D>pair(const pair<C,D>&);
}
template<typename A, typename B>
     bool operator@@(const pair<A,B>&, const pair<A,B>&);
template<typename A, typename B>
     pair<A,B> make_pair(const A&, const B&);
```

Korišćenje parova podataka

- Tipovi `A` i `B` mogu da budu prosti ili klasni, ali ne mogu da budu nizovi
- Neophodno je da postoje konverzije `A(C)` i `B(D)`
- `operator@@` može da bude jedan od 6 relacionih: `==`, `!=`, `<`, `<=`, `>`, `>=`
  - `(a,b)==(c,d)` ako: `a==c && b==d`
  - `(a,b)<(c,d)` ako: `a<c || a==c && b<d`
  - ostali relacioni operatori se generišu iz `==` i `<`
  - u klasama `A` i `B` je dovoljno da se definišu operatori `==` i `<`
- Primer:

```
#include<iostream>
#include<utility>
using namespace std;
int main(){
     pair<float,int> p; pair<int,char>q(1,'a');
     cout<<p.first<<' '<<p.second<<endl;
     cout<<q.first<<' '<<q.second<<endl;
}
```

Kompleksni brojevi

- Za rad sa kompleksnim brojevima - generička klasa u biblioteci `<complex>`
  - `template <typename T> class complex;`
- Potpune specijalizacije šablona za: `float`, `double`, `long double`
- Klasa podržava sve operacije kao sa bilo kojim numeričkim tipom:
  - aritmetičke: `+`, `-`, `*`, `/`, `=`, `+=`, `-=`, `*=`, `/=` i relacione: `==` i `!=`
- Konstruktor: `complex(const T& re=T(), const T& im=T());`
- Metodi: `T real()const; T imag()const;`
- F-je:
```
T real(const complex<T>& z); T imag(const complex<T>& z);
T abs(const complex<T>& z); T arg(const complex<T>& z);
T norm(const complex<T>& z); //kvadrat potega
complex<T> polar(const T& r, const T& fi);
complex<T> conj(const complex<T>& z);
```
  - slično za: `sqrt`, `exp`, `log`, `log10`, `sin`, `cos`, `tan`, `sinh`, `cosh`, `tanh`
  - za: `pow(a,b)`
    - ako je `a` complex, `b` ∈ {`int`, realan broj, `complex`}
    - ako je `a` realno, `b` je tipa `complex`

Zbirke podataka

- Zbirke su objekti koji sadrže podatke nekog tipa
- Drugi nazivi: kontejneri, kolekcije
- Zbirke omogućavaju:
  - dodavanje novih elemenata,
  - dohvatanje postojećih elemenata
  - izbacivanje nepotrebnih elemenata iz zbirke
- Zbirke imaju različite unutrašnje strukture koje utiču na:
  - efikasnost (brzinu) pojedinih operacija nad zbirkom:
    - pristup elementu, dodavanje, izbacivanje
  - zauzeće memorije
- Primeri:
  - vektori – mali utrošak memorije, brz pristup, sporo umetanje i izbacivanje
  - liste – veći utrošak memorije, spor pristup, brzo umetanje i izbacivanje

Vrste zbirki

- Osnovne grupe zbirki
  - **sekvencijalne** – strukture podataka kojima može da se pristupa sekv.
  - **asocijativne** – uređene strukture podataka sa brzom pretragom: red složenosti O(log n)
  - **neuređene asocijativne** – neuređene str. pod. sa brzim pristupom: O(1), O(n) u najgorem slučaju.
- Klase raznih zbirki imaju sličan interfejs, ali su nezavisne
  - nemaju zajedničkog pretka
- Za zbirke klasnih tipova,
  od projektanta klase elemenata zbirke se očekuje da obezbedi:
  - podrazumevani konstruktor, kopirajući i premeštajući konstruktor
  - destruktor, `operator=` (kopirajući i premeštajući)
  - za asocijativne zbirke i `operator==` i `operator<`

Elementi zbirke

- Prilikom stavljanja elemenata u zbirku – pravi se kopija (konstruktor kopije)
- Prilikom izbacivanja elemenata iz zbirke – element se uništava (destruktor)
- Ako su elementi zbirke pokazivači, kopiraju se samo pokazivači, ne objekti
  - ne pozivaju se konstruktor kopije, odnosno destruktor
- Osim prvog i poslednjeg svaki element zbirke ima prethodnika i sledbenika
  - ovo ne zavisi od strukture zbirke
- Redni broj prvog elementa zbirke je 0
- Tekući je onaj element zbirke kojem je trenutno moguć pristup

Iteratori

- Iterator je objekat pridružen objektu zbirke radi njenog obilaženja i izvršavanja neke radnje nad svakim obiđenim elementom zbirke
- Klase iteratora u STL-u: ugnežđene klase u klase zbirki
- Objekat iteratora u svakom trenutku omogućava pristup tekućem elementu zbirke
- Radnje nad iteratorima:
  - stvaranje uz postavljanje na početak/kraj zbirke
  - pomeranje sa jednog na drugi element
  - pristup tekućem elementu
  - upoređivanje (da li 2 iteratora pokazuju na isti element)

Iteratori kao pametni pokazivači

- Iteratori mogu da se zamisle kao pametni pokazivači koji pokazuju na elemente nizova
- Operacije nad iteratorima su realizovane preklapanjem operatora
  - radnje sa iteratorima izgledaju notacijski isto kao sa pokazivačima
  - ako `i` pokazuje na tekući element zbirke,
    - `i+1` pokazuje na naredni
    - `i-1` pokazuje na prethodni
    - `*i` predstavlja vrednost tekućeg elementa zbirke
- Iterator može da pokazuje ispred prvog ili iza poslednjeg elementa
  - tada nije dozvoljen pristup elementu zbirke

Vrste iteratora

- Izlazni
  - omogućava samo postavljanje vrednosti tekućeg elementa i kretanje unapred
- Ulazni
  - omogućava samo dohvatanje vrednosti tekućeg elementa i kretanje unapred
- Jednosmerni
  - omogućava radnje izlaznih i ulaznih operatora
- Dvosmerni
  - omogućava radnje jednosmernih iteratora + kretanje unazad
- Sa proizvoljnim pristupom
  - omogućava radnje dvosmernih iteratora + pristup po proizvoljnom redosledu

Iteratori i dozvoljene operacije

Izlazni	Ulazni	1-smerni	2-smerni	Proizvoljni
`I(i) =`	`I(i) =`	`I(i) =`	`I(i) =`	`I(i) =`
`*i=`	`=*i`	`*i`	`*i`	`*i`
				`[]`
	`->`	`->`	`->`	`->`
`++`	`++`	`++`	`++ --`	`++ -- + - += -=`
	`== !=`	`== !=`	`== !=`	`== != < > <= >=`

Tipovi iteratora

- Tipovi iteratora su nezavisni od "vrste" iteratora
- Pod tipom se podrazumeva jezički tip (klasa)
- Vrsta iteratora je određena samom zbirkom,
  a tip iteratora navodi programer kada ga deklariše
- Postoje sledeći tipovi:
  - `iterator` – direktni za promenljive zbirke
  - `const_iterator` – direktni za nepromenljive zbirke
  - `reverse_iterator` – obrnuti za promenljive zbirke
  - `const_reverse_iterator` – obrnuti za nepromenljive zbirke

Osobine tipova iteratora

- Direktni:
  - i+1 pokazuje na sledeći element u odnosu na tekući
- Obrnuti (reverzni):
  - i+1 pokazuje na prethodni element u odnosu na tekući
- Konstantni (za nepromenljive zbirke)
  - preko njega zbirka ne može da se menja ni umetanjem, ni izbacivanjem, ni menjanjem vrednosti elemenata

Tipovi zbirki

- Nizovi (`array`)
- Vektori (`vector`)
- Redovi sa dva kraja (`deque`)
- Liste (`list`) i jednosmerne liste (`forward_list`)
- Specijalne zbirke (`queue`, `priority_queue`, `stack`)
- Preslikavanja (`map`, `multimap`)
- Skupovi (`set`, `multiset`)
- Tekstovi (`string`)
- Nizovi bitova (`vector<bool>`, `bitset`)

Vektori

- Jednodimenzioni nizovi neograničenog kapaciteta
- Omogućavaju:
  - efikasan pristup elementima po proizvoljnom redosledu
  - efikasno dodavanje na kraj i izbacivanje sa kraja
  - neefikasno dodavanje i izbacivanje sa početka i u sredini
  - automatsku promenu kapaciteta
    - standard ne propisuje inkrement promene kapaciteta
- Spadaju u sekvencijalne zbirke
- Koriste iteratore sa proizvoljnim pristupom

Klasa `vector`

- Deklaracija se nalazi u zaglavlju `<vector>`

```
template<typename E,
         class Alloc=std::allocator<T> > class vector;
```

`E` je tip elemenata vektora, `Alloc` – klasa alokatora

- Klasa poseduje veliki broj metoda i:
  - konstruktore
    - podrazumevani (stvara prazan vektor)
    - konstruktor kopije i konstruktor premeštanja
    - `vector(unsigned vel, const E& x=E());`
    - `template<class It> vector(It prvi,It posl);`
      kopira elemente neke zbirke iz intervala [`prvi`, `posl`)
  - destruktor
  - `operator=` (kopiranjem i premeštanjem)

Metodi klase `vector` (1)

```
void assign(unsigned n, const E& x=E());
template <class It> void assign (It prvi, It posl);
unsigned size()const;
unsigned capacity()const;
unsigned max_size()const;
bool empty()const;
void resize(unsigned vel, const E& x=E());
void reserve(unsigned n);
```

Primer

```
#include<iostream>
#include<vector>
using namespace std;
int main(){ vector<int> v(1000);
  cout<<"size="<<v.size()<<"  max_size="<<v.max_size()<<
      "  capacity="<<v.capacity()<<endl;
   v.push_back(1);
   cout<<"size="<<v.size()<<"  max_size="<<v.max_size()<<
      "  capacity="<<v.capacity()<<endl; return 0;
}
Izlaz ( na 32-bitnoj mašini):
size=1000  max_size=1073741823  capacity=1000
size=1001  max_size=1073741823  capacity=1500
```

Metodi klase `vector` (2)

```
iterator begin();
const_iterator begin()const;
iterator end();
const_iterator end()const;
reverse_iterator rbegin();
const_reverse_iterator rbegin()const;
reverse_iterator rend();
const_reverse_iterator rend()const;
```

Primer

```
#include<vector>
#include<iostream>
usage namespace std;
vector<int> v(10);
typedef vector<int>::iterator Vit;
typedef vector<int>::reverse_iterator Vrit;
for(Vit i=v.begin(); i!=v.end(); cin>>*i++);
for(Vrit i=v.rbegin(); i!=v.rend(); cout<<*i++<<' '); cout<<endl;

// v.begin()   - iterator koji pokazuje na prvi element v
// v.end()     - iterator koji pokazuje iza poslednjeg elementa v
// v.rbegin()  - iterator koji pokazuje na poslednji element v
// v.rend()    - iterator koji pokazuje ispred prvog elementa v
```

Metodi klase `vector` (3)

```
E& operator[](unsigned ind);
const E& operator[](unsigned ind)const;
E& at(unsigned ind); //0<=ind<=size(), out_of_range
const E& at(unsigned ind)const;
E& front();
const E& front()const;
E& back();
const E& back()const;
void push_back(const E& x);
void pop_back();
```

Metodi klase `vector` (4)

```
iterator insert(iterator poz, const E& x=E());
void insert(iterator poz, unsigned n,
            const E& x=E());
template<class It> void insert(iterator poz,
            It prvi, It posl);
iterator erase(iterator poz);
iterator erase(iterator prvi, iterator posl);
void clear();
void swap(vector<E> & v);
```

Globalna funkcija sa `vector` arg

- Leksikografsko poređenje 2 vektora:

```
bool operator@@(const vector<E>&v1, const vector<E>&v2);
// @@ ∈ {==,!=,<,<=,>,>=}
// v1==v2  ako imaju jednak broj elemenata
//         i ako su odgovarajući elementi e1==e2
// v1<v2   ako kod prvog para razlicitih elementa e1<e2
//         ili ako je v1 kraci od v2, a svi e1==e2
```

Primer

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
  vector<int> v; int k;
   cout<<"niz? ";
   while(cin>>k, k!=9999){
     if(v.size()==v.capacity()) v.reserve(v.size()+10);
        // bez eksplicitne rezervacije - dodaje se 1/2 kapac.
     v.push_back(k);
   }
   v.reserve(v.size()); //ne mora da ima efekat
   int s=0; for(unsigned i=0; i<v.size(); s+=v[i++]);
   cout<<"S="<<s<<endl; return 0;
}
```

Redovi sa dva kraja

- Dinamički nizovi koji pored mogućnosti vektora omogućavaju i
  - efikasno dodavanje elementa na i uklanjanje sa početka niza
- Definisani su u biblioteci `<deque>`

```
template<typename E> class deque;
```

- Naziv potiče od *double-ended queue*
- Spadaju u sekvencijalne zbirke, koriste iteratore sa proizvoljnim pristupom
- U odnosu na šablon `vector<>`, ne postoje samo metodi:

```
capacity() i reserve()
```

- Postoje metodi:

```
void push_front(const E& x);
void pop_front();
```

- Pogodna struktura za realizaciju FIFO bafera

Liste

- Omogućavaju efikasno umetanje i izbacivanje elemenata i u unutrašnjosti zbirke
- Efikasan sekvencijalni pristup
- Neefikasan pristup po proizvoljnom redosledu
- U zaglavlju `<list>`:

```
template <typename E> class list;
```

- Spadaju u sekvencijalne zbirke i koriste dvosmerne iteratore
- Iste funkcije kao za `deque`, sa izuzetkom **`operator`**`[]()` i `at()`

Metodi klase `list`

```
void splice(iterator poz, list<E>& lst);
void splice(iterator poz, list<E>& lst, iterator i);
void splice(iterator poz, list<E>& lst,
            iterator prvi, iterator posl);
void remove(const E& x);
template<class U>void remove_if(U u);
void unique();
template<class U> void unique(U u);
void merge(list<E>& lst);
template<class U> void merge(list<E>& lst, U u);
void sort();
template<class U> void sort(U u);
void reverse();
```

Red, prioritetni red i stek

- Red, prioritetni red i stek su specijalne zbirke
  - kod reda (`queue`) se podaci dodaju na kraj, a uzimaju sa početka
  - kod prioritetnog reda (`priority_queue`) podaci se uzimaju po opadajućem prioritetu
  - kod steka (`stack`) se podaci dodaju na kraj (vrh) i uzimaju sa kraja
- Specijalne zbirke koriste ranije opisane zbirke
- U zaglavljima `<queue>`, odnosno `<stack>`:

```
template<typename E,class Z=deque<E>> class queue;
template<typename E,class Z=vector<E>,class U=less<E>>
    class priority_queue;
template<typename E, class Z=deque<E>> class stack;
```

Preslikavanja

- Elementi zbirki preslikavanja su parovi vrednosti (ključ, podatak)
- Elementi su uređeni po vrednosti ključa
  - to omogućava brzo pronalaženje podatka na osnovu ključa
- Preslikavanja spadaju u asocijativne zbirke
- Preslikavanja koriste dvosmerne iteratore
- Preslikavanje ključa u podatak može da bude:
  - jednoznačno (jedinstvena vrednost ključa) – zbirka `map`
  - višeznačno (više podataka u zbirci za jednu vrednost ključa) – `multimap`
- U zaglavlju `<map>`:

```
template<typename K,typename P,class U=less<K>> class map;
template<typename K,typename P,class U=less<K>> class multimap;
```

Skupovi

- Brzo pronalaženje elemenata na osnovu njihovih vrednosti
- Preslikavanja koja ključeve preslikavaju u sebe (elementi sadrže samo ključ, odnosno ključ je i podatak)
- Skupovi su uređeni – elementi su uređeni po vrednostima
- Spadaju u asocijativne zbirke i koriste dvosmerne iteratore
- Slično preslikavanjima, skupovi mogu da budu:
  - bez ponavljanja elemenata – `set`
  - sa mogućim duplikatima elemenata – `multiset`
- U zaglavlju `<set>`:

```
template<typename K, class U=less<K>> class set;
template<typename K, class U=less<K>> class multiset;
```

Tekstovi (stringovi)

- Tekstovi se sastoje od niza znakova nekog tipa
- Za rad sa tekstovima promenljive dužine – šablon `basic_string`
- Klasa u zaglavlju `<string>`:
  - obratiti pažnju da su `<string>` i `<string.h>` različite biblioteke

```
template <typename Z> class basic_string;
```

- Omogućava generisanje klasa za razne tipove znakova (npr. za 8-bitne ASCII znake, za 16-bitne UNICODE znake)
- Vrlo često se koriste `char` znaci u tekstovima, pa je definisan tip:

```
typedef basic_string<char> string; //generisana klasa
```

- Postoje svi metodi klase `vector` izuzev `push_back()` i `pop_back()`
- Mnogo dodatnih metoda i operatora za rad sa tekstovima
  - `assign(=)`, `append(+=)`, `+`, `<<`, `>>`, *rel.ops.*, `getline`, `compare`, `substr`, `copy`, `insert`, `replace`, `erase`, `find`

Nizovi bitova

- Dve varijante: (1) `vector<bool>` i (2) `bitset`
  - (1) specijalizacija šablona `vector` za elemente tipa **`bool`**
    - dinamički nizovi promenljivih dužina sa efikasnim skladištenjem bitova
    - bitovima se pristupa kao i bilo kojim drugim elementima vektora
    - primer:

```
vector<bool> v(20);
for(unsigned i=0; i<v.size(); cin>>v[i++]);
for(vector<bool>::iterator i=v.begin(); i!=v.end(); cout<<*i++);
```

  - (2) generička klasa koja nije zbirka (nema iteratore)
    - deklaracija u zaglavlju `<bitset>`

```
template<unsigned N> class bitset;
```

    - na podatke se primenjuju std. bitski operatori `~`, `&`, `|`, `^`, `<<`, `>>`, `&=`,...
    - elementi (bitovi) se numerišu sdesna-ulevo, krajnji desni element – pozicija 0
    - metodi i op.: `set`, `reset`, `flip`, `to_ulong`, `count`, `size`, `test(poz)`, `any`, `none`, `[]`, `>>`, `<<`, `==`, `!=`

Algoritmi

- Generičke funkcije u zaglavlju `<algorithm>`
- STL podržava sledeće kategorije algoritama
  - algoritmi nad pojedinačnim podacima (`min`, `max`, `swap`)
  - obrada pojedinačnih elemenata zbirki (`for_each`, `fill`, `generate`,...)
  - pretraživanje zbirki (`find`, `min/max_element`, `count`, `search`,...)
  - obrade zbirki:
    - kopiranje (`copy`, `copy_backward`)
    - zamena (`replace`, `replace_if`,...)
    - uklanjanje (`remove`, `remove_if`, `remove_copy`,..., `unique`,...)
    - obrtanje (`reverse`, `reverse_copy`)
    - poređenje (`equal`, `lexicographical_compare`)
    - permutacije (`next_permutation`, `prev_permutation`)
  - obrade uređenih sekvencijalnih zbirki (`sort`, `merge`, `binary_search`)