



Napredne tehnike programiranja

Vježba 13



Napredne tehnike programiranja



- STL spremniki (kontejneri)
- Sekvencijalni spremniki
- Asocijativni spremniki

Standard template library (STL)



- Godine 1990. Alex Stepanov i Meng Lee iz Hewlett Packard Laboratories proširili su C++ s bibliotekom predložaka klasa i funkcija koja je postala poznata kao STL.
- Godine 1994. STL je usvojen kao dio ANSI/ISO standarda C++.

Standard template library (STL)



- STL je ima tri osnovne komponente:
- Spremnici (kontejneri)
 - Generički predlošci klasa za pohranjivanje zbirke podataka.
- Algoritmi
 - Generički predlošci funkcija za rad na spremnicima.
- Iteratori
 - Pokazivači koji olakšavaju korištenje spremnika (pružaju sučelje koje je potrebno za STL algoritme za rad na STL spremnicima)

Standard template library (STL)



- STL nudi veliki izbor spremnika
- Poznata je vremenska i memorijska složenost spremnika
- STL spremnici automatski rastu i smanjuju se
- STL pruža ugrađene algoritme za obradu spremnika
- STL pruža iteratore koji čine spremnike i algoritme fleksibilnim i učinkovitim.
- STL je proširiv što znači da korisnici mogu dodati nove spremnike i nove algoritme

Tipovi spremnika u STL-u



Slijedni spremnici (sekvencijalni)

- služe za čuvanje uređene kolekcije elemenata određenog tipa
- osnovni tipovi: `vector`, `list`, `deque`
- adaptirani tipovi: `stack`, `queue`, `priority_queue`

Asocijativni spremnici

- pružaju podršku za efikasno pronalaženje elemenata na temelju ključa
- tipovi: `map`, `multimap`, `set`, `multiset`

Spremnnici



Sekvencijalni spremnnici:

- [array](#) Array class (class template)
- [vector](#) Vector (class template)
- [deque](#) Double ended queue (class template)
- [forward_list](#) Forward list (class template)
- [list](#) List (class template)

Adapteri:

- [stack](#) LIFO stack (class template)
- [queue](#) FIFO queue (class template)
- [priority_queue](#) Priority queue (class template)

Asocijativni spremnnici:

- [set](#) Set (class template)
- [multiset](#) Multiple-key set (class template)
- [map](#) Map (class template)
- [multimap](#) Multiple-key map (class template)

Nesortirani asocijativni spremnnici:

- [unordered_set](#) Unordered Set (class template)
- [unordered_multiset](#) Unordered Multiset (class template
- [unordered_map](#) Unordered Map (class template)
- [unordered_multimap](#) Unordered Multimap (class template)

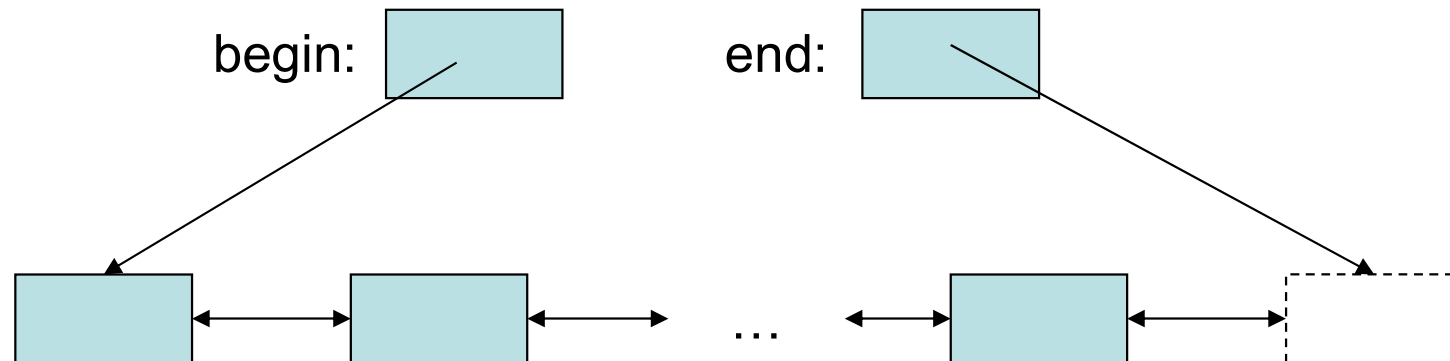
Spremnici



- Sučelje spremnika je napravljeno tako da oni podržavaju iste operacije
 - Dio operacija nude svi tipovi spremnika
 - Dio operacija je specifičan za slijedne tj. asocijativne spremnike
 - Dio operacija je specifičan za konkretan spremnik
- Zbog toga se svim spremnicima upravlja na sličan način

Model spremnika

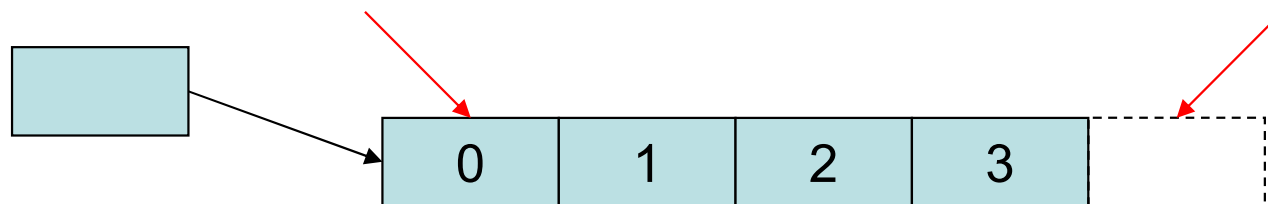
- Iteratori omogućavaju pristup elementima
 - begin (pokazuje na prvi element)
 - end (pokazuje iza zadnjeg elementa)



- Iterator je tip koji podržava “iterator operacije”
 - ++ (idi na slijedeći element)
 - * (vrati vrijednost)
 - == (da li iteratori pokazuju na isti element)
- Neki iteratori podržavaju više operacija (npr. --, +, [])

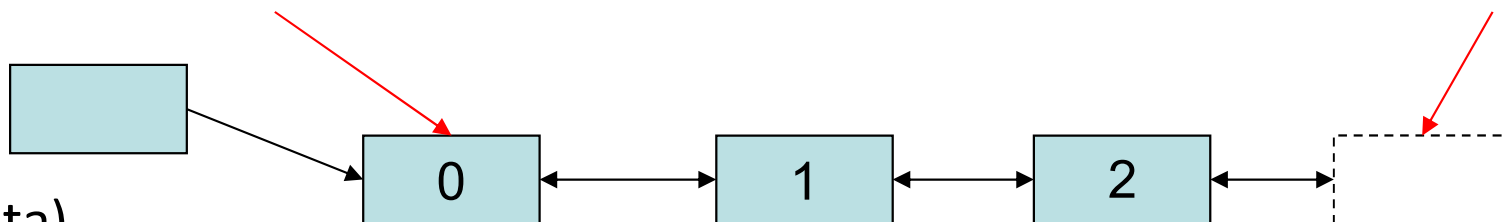
Spremnnici (različito organiziraju podatke)

- **vector**



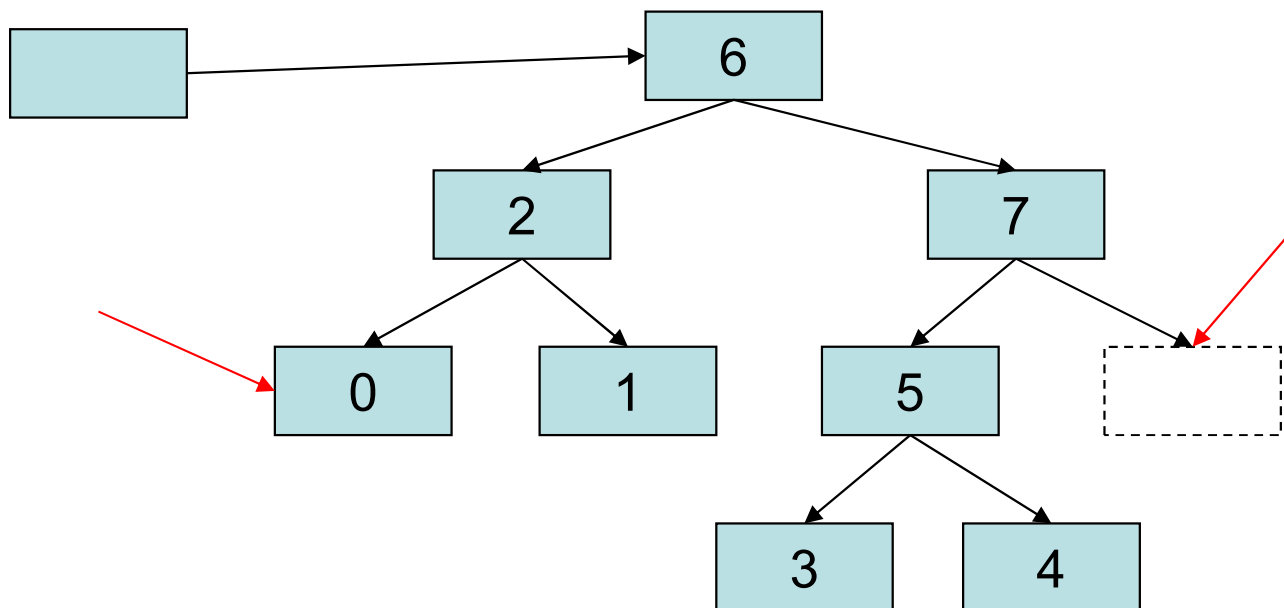
- **list**

(dvostuko vezana lista)



- **set**

(stablo)



Slijedni spremnici



- služe za čuvanje uređene kolekcije elemenata određenog tipa
- elemente dohvaćamo po poziciji (indeksu), npr. `a[i]`
- osnovni tipovi:
 - vector – “polje”: brz pristup pojedinom elementu
 - list – vezana lista: brzo ubacivanje i brisanje na krajevima
 - deque – red sa “dva kraja” (double-ended queue)

Slijedni spremnici



Adaptirani tipovi:

- stack – stog: LIFO
- queue – red: FIFO
- priority_queue – prioritetni red

Na temelju deque-a

Na temelju vector-a

Adaptori – prilagođuju slijedni spremnik koji se krije “ispod površine” tako da mu definiraju novo sučelje

Slijedni spremnici



Zaglavlja:

- vector – `#include <vector>`
- list – `#include <list>`
- deque – `#include <deque>`
- stack – `#include <stack>`
- Queue – `#include <queue>`

Deklaracija spremnika sastoji se od navođenja imena spremnika i tipa elemenata koje želimo čuvati npr. `vector<string>` rijeci;

Vektor



- Pruža alternativu C++ poljima
- Vektor je dinamična struktura (sam mijenja veličinu)
- Header datoteka <vector>
- Primjer:

```
#include <vector>  
vector<int> a(10);
```

Deklaracija vektora



Sintaksa: `vector<tip podataka>`

Primjer:

`vector<int>` - vektor cijelih brojeva

`vector<string>` - vektor stringova

`vector<int * >` - vektor pokazivača na int

`vector<Kvadrat>` - vektor objekata 'Kvadrat'

Konstrukcija vektora



`vector<T> v1;` – defaultni konstruktor – v1 je prazan vektor (s 0 elemenata)

`vector<T> v2 (v1);` – v2 sadrži kopije elemenata od v1 (v1 i v2 moraju biti istog tipa)

`vector<T> v3 (n, i);` – v3 sadrži n elemenata, svaki od kojih je inicijaliziran kopijom vrijednosti i

`vector<T> v4 (n);` – v4 sadrži n elemenata, svaki od kojih je defaultno konstruiran

Korištenje vektora



Dva načina korištenja vektora:

- Kao polje (array)
- STL način

Korištenje vektora



Vektor možemo koristiti kao polje

```
void primjer()
{
    const int N = 10;
    vector<int> brojevi(N);
    for (int i=0; i < 10; ++i)
        cin >> brojevi[i];
    int ia[N];
    for (int j = 0; j < N; ++j)
        ia[j] = brojevi[j];
}
```

Korištenje vektora



Vektor možemo koristiti na STL način:

Kreiramo prazan vektor

```
vector<string> proba;
```

umećemo elemente koristeći metodu `push_back`.

```
string word;  
while ( cin >> rijec ) {  
    proba.push_back(rijec);  
}
```

Operacije na vektorima



`v.empty()` - vraća true ako je v prazan; inače vraća false

`v.size()` - vraća broj elemenata u vektoru v

`v.clear()` - brisanje svih elemenata vektora v

`v[n]` - vraća element na poziciji n u vektoru v, povratni tip je T& (ili const T& ako je vektor konstantan)

`v1 = v2` - pridružuje vektoru v1 kopije elemenata iz v2 (tipovi vektora v1 i v2 moraju biti identični)

Operacije na vektorima



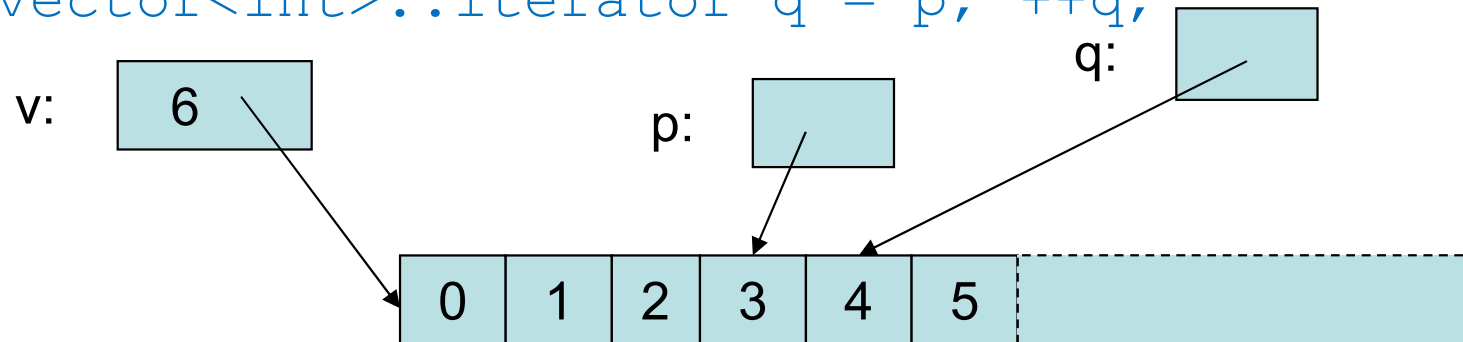
`v.push_back(t)` - dodaje kopiju od `t` kao novi element na kraj vektora i povećava mu veličinu za 1 (može implicirati alokaciju memorije), amortizirano konstantno vrijeme izvršavanja

`v.pop_back()` - izbacuje element s kraja vektora

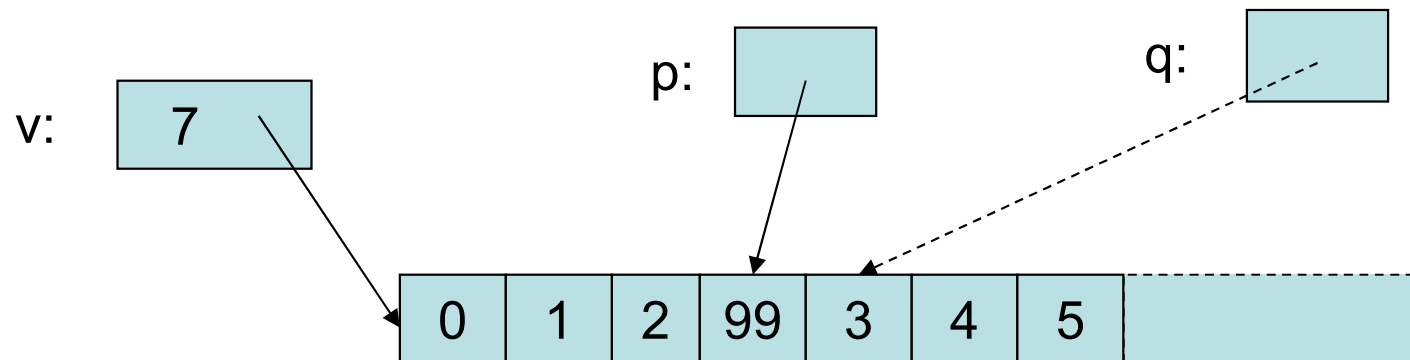
`==`, `!=`, `<`, `<=`, `>`, `>=` - relacijski operatori definirani tako da vektore uspoređuju leksikografski (analogno kao kod tipa string)

Umetanje u vektor - insert()

```
vector<int>::iterator p = v.begin(); ++p; ++p; ++p;  
vector<int>::iterator q = p; ++q;
```



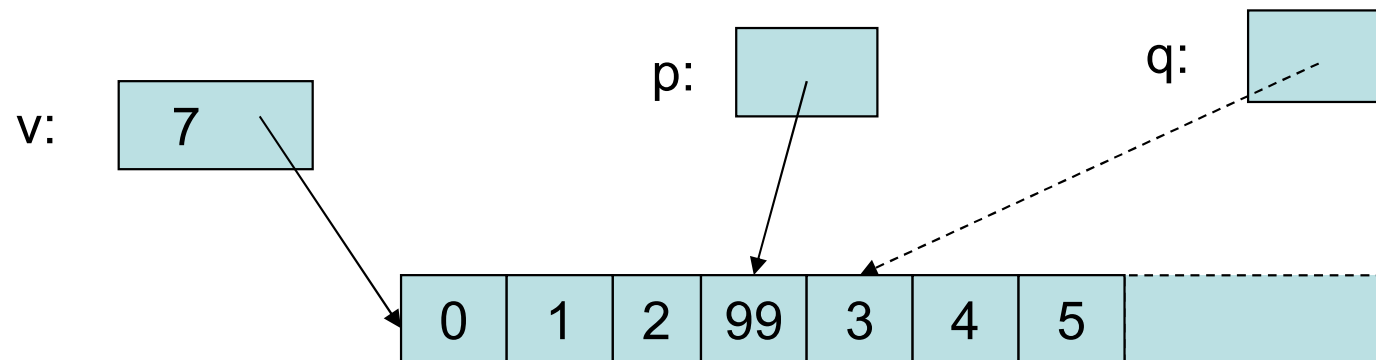
```
p=v.insert(p, 99);    //p pokazuje na novi element
```



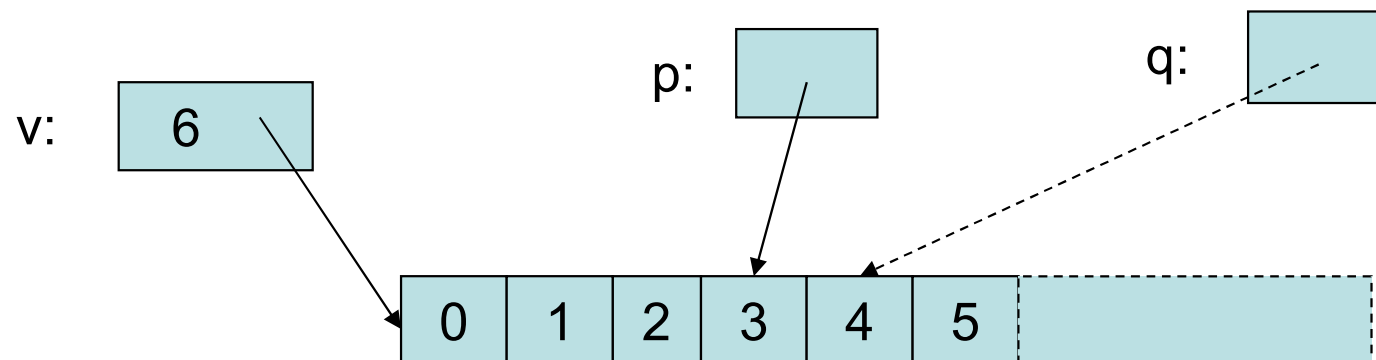
Napomena: `q` je neispravna nakon operacije `insert()`

Napomena : Kad se pomakne jedan element, svi se elementi mogu pomaknuti

Brisanje iz vektora - erase()



```
p = v.erase(p);    // p pokazuje na element iza obrisanog
```



- Nakon operacija `insert()` i `erase()` iteratori se mogu 'izgubiti'

Kretanje kroz vektor



```
for(int i=0; i<v.size(); ++i//bolje size_type umjesto int
... // Napravi nešto sa v[i]
```

```
for(vector<T>::size_type i=0; i<v.size(); ++i)
... // Napravi nešto sa v[i]
```

```
for(vector<T>::iterator p = v.begin(); p!=v.end(); ++p)
... // Napravi nešto sa *p
```

```
for(vector<T>::value_type x : v)
... // Napravi nešto sa x
```

```
for(auto& x : v)
... // Napravi nešto sa x
```


Veličina vektora



- Vektor se tijekom rada s njim povremeno automatski povećava

Terminologija:

- kapacitet – ukupan broj elemenata koji se mogu nalaziti u spremniku prije opetovanog povećavanja – `capacity()`
- veličina – trenutni broj elemenata u spremniku – `size()`

Veličina vektora




- Ručno povećanje vektora

```
vector.reserve(nova_velicina);
```

- Reserve može uništiti iteratore
- capacity i reserve su specifični za vektor i ne pojavljuju se kod drugih spremnika

Rad sa vektorom



```
vector<int> S(2); // S=(0,0)
S.push_back(3); // S=(0,0,3)
S.pop_back(); // S=(0,0)
S.push_back(4); // S=(0,0,4)
S.resize(5); // S=(0,0,4,0,0);
S.reserve(10); // S=(0,0,4,0,0);
cout<<S.front(); // S[0]
S.back()=5; // S[S.size()-1]
S.clear(); // S=();
cout<<S.empty(); // da li je S prazan?
```

Rad sa stack i queue spremnikom

- stack

```
#include <stack>
```

```
...
```

```
stack<int> S;
```

```
S.push(3);
```

```
S.push(5);
```

```
int a = S.top();
```

```
S.pop();
```

```
if (S.empty())
```

```
{ ... }
```

```
int zz = S.size();
```

- queue

```
#include <queue>
```

```
...
```

```
queue<string> Q;
```

```
Q.push("abc");
```

```
Q.push("xy");
```

```
string a = Q.front();
```

```
Q.pop();
```

```
if (Q.empty())
```

```
{ ... }
```

```
int zz = Q.size();
```

Pair



- Omogućava predložak za stvaranje uređenih parova dvaju tipova podataka

```
pair<string, string> student("Pero", "Peric");  
pair<int, int> rezultat(3, 2);  
pair<double, int> tocka(1.2, 3);
```

- Tipovi ne moraju biti isti
- Potrebno je uključiti zaglavlje < utility>

```
#include <utility>
```

Pair



Kreiranje para:

```
pair<string, string> student("Pero", "Peric");
```

Pristup prvom elementu u paru:

```
cout << student.first;
```

Pristup drugom elementu u paru:

```
cout << student.second;
```

Pair (primjer)



Napišite funkciju `min()` koja za dobiveni vector int-ova vraća najmanji element tog vectora, te broj njegovog pojavljivanja.

```
typedef pair<int,int> min_val_pair;
min_val_pair min(const vector<int>& ivec) {
    int minVal = ivec[0];
    int brojac = 0;
    int size = ivec.size();
    for (int i = 0; i < size; ++i) {
        if (minVal == ivec[i])
            ++brojac;
        else
            if (minVal > ivec[i]) {
                minVal = ivec[i];
                brojac = 1;}
    } return make_pair(minVal, brojac);}
```

Pair (operacije)



Operacije na pair-u:

- `pair<T1, T2> p;`
- `pair<T1, T2> p(v1, v2);`
- `make_pair(v1, v2);` //stvara i vraća novi par
- `p1 < p2` //leksikografsko uspoređivanje
- `p1 == p2` //uspoređivanje po elementima
- `p.first` //prvi element para
- `p.second` //drugi element para

Asocijativni spremnici



Asocijativni spremnici podržavaju gotovo sve operacije kao i slijedni spremnici:

- Konstruktori:

```
C<T> c; C<T> c1(c2); C<T> c(b,e);
```

- Relacijski operatori:

```
== !=
```

- Iteratorii:

```
c.begin(); c.end();
```

- Zamjena: `c1.swap(c2);`

- Brisanje: `c.clear(); c.erase(i);`

- Veličina: `c1.size;`

Asocijativni spremnici



Nepodržane operacije su:

`front, push_front, pop_front`

`back, push_back, pop_back`

- Elementi su u asocijativnim spremnicima poredani po ključu, bez obzira na redoslijed kojim ih ubacujemo u spremnik

Map



Zaglavlje

```
#include <map>
```

Mapa je asocijativno polje

- sadrži parove (ključ, vrijednost)
- Ključ se upotrebljava za indeksiranje elemenata
- vrijednost predstavlja korisni podatak koji želimo čuvati

Primjeri:

```
map<string, short> ocjene;  
map<string, short> zaposlenik;  
map< pair<int,int>, boja> slika;
```

Map



Konstruktori:

```
map<k, v> m; //mapa (ključ, vrijednost)
```

```
map<k, v> m1(m2); //mapa m1 je kopija m2
```

- Tip koji koristimo kao ključ mora podržavati operator<

Map



Dodavanje elemenata u mapu sa operatorom []:

```
map<string, int> word_count;  
word_count["abc"] = 1;
```

Korištenje vrijednosti koju vraća operator[]:

```
cout << word_count["abc"];  
++word_count["abc"];
```

- Oprez: operator[] ubacuje element u mapu ako on tamo još nije bio

Map



Dodavanje elemenata u mapu:

`m.insert(e)`

- Ubacuje par `e` u mapu `m` ako ključ od `e` još nije u mapi
- Ako je ključ od `e` u mapi, onda se ne dogodi ništa
- Vraća par (iterator, bool)
 - Iterator pokazuje na ubačeni element
 - Bool kazuje da li je element bio ubačen ili ne

`m.insert(b, e)`

- Svaki element u rasponu iteratora `b` i `e` se ubacuje u mapu `m`
- Vraća void

Map



Primjeri:

```
word_count.insert(make_pair("abc", 1));
```

– Insert očekuje par

Dohvat elementa iz mape:

```
int occurs = word_count["xyz"];
```

Problem: ako ključ “xyz” nije postojao u mapi, operator[] ga je upravo dodao u mapu, što ne bismo htjeli

Map



Operacije koje provjeravaju da li je ključ u mapi bez da ga ubace:

`m.count(k)`

- Vraća broj pojavljivanja ključa k u mapi m
- Vraća ustvari 0 ili 1

`m.find(k)`

- Vraća iterator na ključ k, ako k postoji u mapi
- Ako ključ ne postoji u mapi, vraća `end()` iterator

Map



Brisanje elemenata iz mape:

`m.erase(k)`

- Briše elemente sa ključem k u mapi m

`m.erase(i)`

- Briše element na kojeg pokazuje iterator i

Map (primjer)

```
#include <iostream>
#include <algorithm>
#include <utility>
#include <map>
using namespace std;
void PrintOut(const pair<int,int>& p) { cout << "printout: [" << p.first
<< "," << p.second << "]" << endl;}
int main() {
    int a=1, b=2, c=3, d=4, e=5, f=6;
    map<int,int> table;
    map<int,int>::iterator it;
    table.insert(pair<int,int>(a, b));
    table[c] = d;
    table[e] = f;
    cout << "table[e]:" << table[e] << endl;
    it = table.find(c);
    cout << "PrintOut(*it), where it = table.find(c)" << endl;
    PrintOut(*it);
    cout << "iterating:" << endl;
    for_each(table.begin(), table.end(), &PrintOut); return 0; }
```

Set (skup)



Zaglavlje:

```
#include <set>
```

- Skup je samo kolekcija ključeva
- Primjeri:

```
set<string> rjecnik;
```

```
set< pair<int,int> > tocke
```

- Operacije definirane na skupu su identične operacijama na mapi, samo nema operatora []

Sr. No	Data Structure	Sub Type	Operations	Time Complexity	Space Complexity
1	Priority Queue	Max Heap	Q.top()	O(1)	O(1)
		Min Heap	Q.push()	O(log n)	O(1)
			Q.pop()	O(log n)	O(1)
			Q.empty()	O(1)	O(1)
2	Map	Ordered Map	M.find(x)	O(log n)	O(1)
		Unordered Map	M.insert(pair<int, int> (x, y)	O(log n)	O(1)
		Ordered Multimap	M.erase(x)	O(log n)	O(1)
		Unordered Multimap	M.empty()	O(1)	O(1)
			M.clear()	Theta(n)	O(1)
			M.size()	O(1)	O(1)
3	Set	Ordered set	s.find()	O(log n)	O(1)
		Unordered set	s.insert(x)	O(log n)	O(1)
		Ordered Multiset	s.erase(x)	O(log n)	O(1)
		Unordered Multiset	s.size()	O(1)	O(1)
			s.empty()	O(1)	O(1)
4	Stack		s.top()	O(1)	O(1)
			s.pop()	O(1)	O(1)
			s.empty()	O(1)	O(1)
			s.push(x)	O(1)	O(1)
5	Queue		q.push(x)	O(1)	O(1)
			q.pop()	O(1)	O(1)
			q.front()	O(1)	O(1)
			q.back()	O(1)	O(1)
			q.empty()	O(1)	O(1)
			q.size()	O(1)	O(1)
6	Vector	1D vector	sort(v.begin(), v.end())	Theta(nlog(n))	Theta(log n)
			reverse(v.begin(), v.end())	O(n)	O(1)
			v.push_back(x)	O(1)	O(1)
			v.pop_back()	O(1)	O(1)
		2-dimensional vector	v.size()	O(1)	O(1)
			v.clear()	O(n)	O(1)
			v.erase()	O(n^2)	O(1)
7	List		L.emplace_front(val)	O(1)	
			L.emplace_back(val)	O(1)	
			L.push_front(val)	O(1)	
			L.push_back(val)	O(1)	
			L.pop_front()	O(1)	
			L.pop_back()	O(1)	
			L.insert(iterator, val)	O(1)	
			L.erase(iterator)	O(1)	
			L.clear()	O(n)	

Vremenska i
prostorna složenost
operacija u
spremnici



Zadatak



1. Učitavajte znak po znak sa tipkovnice sve dok ne učitajte točku, upitnik ili uskličnik. Ako ste pročitali slovo, spremite ga u vektor. Vektor neka se sastoji od uređenih parova <slovo, broj pojavljivanja slova>. Dakle, kada pročitate novo slovo, potrebno je pregledati da li u vektoru već postoji navedeno slovo. Ako postoji, samo treba povećati odgovarajući brojač; inače treba ubaciti novo slovo (sa odgovarajućim brojačem) na kraj vektora.
2. Napišite program koji čita niz riječi sa tipkovnice (dok se ne učitava riječ 'kraj'). Pohranite u vektor parove (riječ, redni broj riječi). Zatim korisnik treba upisati još jednu riječ, a program treba napisati da li je ta riječ bila upisana prije sa tipkovnice, koliko puta i koji su bili redni brojevi tih upisa.
3. Napišite program koji čita niz riječi sa tipkovnice (dok se ne učitava riječ 'kraj'). U mapi program bilježi broj pojavljivanja svakog slova i riječi. Na kraju se ispiše znak i riječ koji su najčešće pojavljivali.