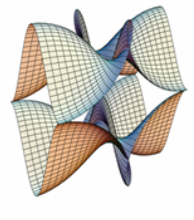

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 06 - Standard Template Library (2)

v2018/2019.

Sastavio: Zvonimir Bujanović

pair

`pair` je generička struktura koja predstavlja uređeni par.
Gotovo je identična kao naša ranija implementacija strukture `par`.

```
#include <utility>
```

Konstruktori.

```
pair<int, int> A;
pair<int, string> B( 10, "rp1" ), C( 5, "x" );
```

Primjer korištenja.

```
A = make_pair( 2, 3 );
B.first = 20;
C.second = "vjezbe";
if( B == C ) { ... }
B = C;
if( B < C ) { ... }
```

tuple

`tuple` je generička struktura u C++11 koja predstavlja uređenu n-torku.

```
#include <tuple>
```

Konstruktori.

```
tuple<int, char, int> A;
tuple<int, char, string> B( 10, 'b', "rp1" ), C( 5, 'c', "x" );
```

Primjer korištenja.

```
A = make_tuple( 2, 'a', 3 );
get<0>( B ) = 20;
get<2>( B ) = "blabla";
if( B == C ) { ... }
B = C;
if( B < C ) { ... }
```

set

`set<T>` je generička struktura koja predstavlja skup elemenata tipa `T`.

Na tipu `T` mora biti definiran operator `<`.

```
#include <set>
```

Usporedi s ATP `Set` iz Struktura podataka i algoritama.

Implementacija u STL-u je efikasnija od onih napravljenih na SPA.
`set` je obično implementiran u STL-u pomoću tzv. red-black stabala.

set

Konstruktori.

- Prazan skup.

```
set<int> A;
```

- Skup koji sadrži sve elemente nekog polja, vectora, liste ...
(Ovakav konstruktor postoji i za `vector` i `list`.)

```
vector<string> A; // napunimo nekako vektor
set<string> S1( A.begin(), A.end() );

int B[5] = { 5, 10, 15, 20, 25 };
set<int> S2( B, B+5 );
```

set

Ubacivanje elemenata.

```
set<int> A;
A.insert( 5 );  // A = {5}
A.insert( 10 ); // A = {5, 10}
A.insert( 15 ); // A = {5, 10, 15}
A.insert( 10 ); // A = {5, 10, 15}
```

Funkcija `insert` vraća `pair`.

- Prvi element para je iterator na ubačeni element.
- Drugi element para je tipa `bool`. Vrijednost mu je `false` ako je element koji ubacujemo već bio element skupa, a `true` inače.

Možemo odjednom ubaciti i sve elemente nekog drugog "kontejnera".

```
vector<int> v;
A.insert( v.begin(), v.end() );
set<int> B;
A.insert( B.begin(), B.end() );
```

set

Skup možemo zamišljati kao sortiranu listu po kojoj se krećemo iteratorima.

```
// ispis elemenata skupa (od manjeg prema većem):
set<int>::iterator si;
for( si = A.begin(); si != A.end(); ++si )
    cout << *si << " ";
```

Sličnost s iteratorima kod liste:

- U `A.begin()` postoji element skupa, u `A.end()` ne postoji.
- Iteratori su dvosmjerni, tj. smijemo raditi i `--si`.

```
// ispis elemenata skupa (od većeg prema manjem):
set<int>::iterator si = A.end();
do {
    --si;
    cout << *si << " ";
} while( si != A.begin() );
```

set

Za razliku od liste, ne možemo mijenjati vrijednost elementa pomoću iteratora!
(To bi moglo narušiti "sortiranost" skupa.)

```
set<int> S;
S.insert( 5 );

set<int>::iterator si = A.begin();
cout << *si; // OK: ispise "5".

*si = 3; // NIJE OK: !!! compile error !!!
```

Dakle, iteratori u skupu su "read-only".

Broj pojavljivanja elementa: `count`.

```
// A.count(x) vraca broj pojavljivanja elementa x u skupu A.
// (dakle, vraca 0 ili 1)
if( A.count( 5 ) )
    cout << "Skup A sadrzi element 5";
```

Iterator na traženi element: `find`.

```
// A.find(x) vraca iterator elementa x u skupu A.
// Ako x nije element skupa, vraca A.end()
if( A.find( 1 ) == A.end() )
    cout << "1 nije element skupa A";
```

set

Broj elemenata skupa: `size`.

```
set<int> A;
A.insert( 5 ); A.insert( 10 ); A.insert(5); // A = {5, 10}
cout << A.size(); // ispise: "2"
```

Provjera je li skup prazan: `empty`.

```
// A.empty() vraca true ako je A prazan, a false inace.
if( A.empty() ) { ... }
```

Brisanje svih elemenata skupa: `clear`.

```
A.clear();
if( A.empty() ) { ... } // sada je prazan
```

Sve ove funkcije postoje i u `vector` i u `list`.

Zadatak 1

Napišite program koji:

1. Učitava stringove sve dok ne unesemo string koji je već ranije bio unešen.
2. Ispisuje koliko je različitih stringova bilo unešeno, te sve te stringove.

set

Izbacivanje elemenata: `erase`.

```
set<int> A;
... // napravimo npr. A = {5, 15, 20}
A.erase( 15 ); // A = {5, 20}
A.erase( 10 ); // A = {5, 20}
```

Izbacivati možemo i pomoću iteratora na element.

```
// Izbacujemo najmanji element.
A.erase( A.begin() );

// Izbacujemo sve elemente (krace je pomocu A.clear()).
set<int>::iterator si;
for( si = A.begin(); si != A.end(); )
    A.erase( si++ );
```

Kao i kod liste, brisanje uništi iterator.

Zadatak 2

Napišite program koji:

1. Učitava stringove sve dok ne unesemo string `"kraj"`.
2. Ispisuje koliko je različitih stringova uneseno (zajedno s `"kraj"`), te ih sve ispisuje.
3. Nakon toga izbacuje sve stringove parne duljine, ispisuje koliko je stringova ostalo u skupu, te ih sve ispisuje.

multiset

`multiset` je generička struktura koja predstavlja multiskup.
U multiskupu možemo imati više istih elemenata.

```
#include <set>
```

Operacije su analogne onima iz `set`-a.

```
multiset<int> A;
A.insert( 7 ); A.insert( 5 ); A.insert(5); // A = {5, 5, 7}
cout << A.size();      // 3
cout << A.count( 5 ); // 2
```

multiset

Svaka kopija pojedinog elementa ima svoj iterator.

- `find(x)` vraća iterator na *neki* element koji je $= x$.
- `lower_bound(x)` vraća iterator na prvi element koji je $\geq x$.
- `upper_bound(x)` vraća iterator na prvi element koji je $> x$.

```
multiset<int> A;
... // napunimo do A = {5, 10, 10, 15, 20}
multiset<int>::iterator si1, si2, si3;
si1 = A.find(10); // iterator na prvi ili drugi element 10
si2 = A.lower_bound(10); // iterator na prvi element 10
++si2;                   // iterator na drugi element 10
si3 = A.upper_bound(10); // iterator na element 15
```

Brisanje elemenata.

- `erase(x)` briše sve kopije elementa `x`.
- `erase(si)` briše samo onu kopiju na koju pokazuje iterator `si`.

map

`map<D, K>` je generička struktura koja predstavlja preslikavanje elemenata domene `D` u kodomenu `K`.

Na elementima domene mora biti definiran operator `<`.
Kodomena može biti proizvoljna.

Elementi domene = "ključevi", elementi kodomene = "vrijednosti".

```
#include <map>
```

Usporedi s ATP `Mapping` iz Struktura podataka i algoritama.

Implementacija u STL-u je efikasnija od onih napravljenih na SPA.
`map` je obično implementiran u STL-u pomoću tzv. red-black stabala.

map

`map` je primjer asocijativnog polja. Indexi u polju su elementi domene.

```
map<string, int> ocjena;
ocjena[ "Mirko" ] = 3;
ocjena[ "Maja" ] = 5;

cout << ocjena[ "Mirko" ]; // ispise "3"

// !!! oprez kod citanja nepridruzenih elemenata !!!
// ispise "0" i doda ocjena[ "Pero" ] = 0 u mapu.
// (opcenito, poziva se defaultni konstruktor za kodomenu)
cout << ocjena[ "Pero" ];

map<int, int> polje;
polje[ 1 ] = 123;
polje[ 10000000 ] = 456;
polje[ -1000000 ] = 789;
```

map

`map` također možemo zamišljati kao sortiranu listu.

- U svakom čvoru nalazi se uređeni par (`d`, `k`), gdje je `d` element domene, a `k` njemu pridružen element kodomene.
- Lista je uzlazno sortirana po elementima domene.

Dakle, iteratori u mapi su pokazivači na `pair`.

```
map<string, int> ocjena;
ocjena[ "Mirko" ] = 3;
ocjena[ "Maja" ] = 5;

map<string, int>::iterator mi;
for( mi = ocjena.begin(); mi != ocjena.end(); ++mi )
    cout << mi->first << " ima ocjenu "
         << mi->second << endl;

// ispis je sortiran po elementima domene:
// Maja ima ocjenu 5
// Mirko ima ocjenu 3
```

map

Traženje ključa u mapi: `find`.

```
map<string,int>::iterator it = ocjena.find( "Pero" );
if( it != ocjena.end() ) // ima li Pero ocjenu?
    cout << "Pero ima ocjenu " << it->second;
else
    cout << "Pero nema ocjenu";
```

Smijemo promijeniti vrijednost pridruženu nekom ključu, ali ne i sam ključ jer to može narušiti sortiranost mape.

```
map<string,int>::iterator it = ocjena.find( "Pero" );
if( it != ocjena.end() )  // ima li Pero ocjenu?
{
    it->second = 5;       // OK, Perina ocjena je sada 5.
    it->first = "Slavko"; // NIJE OK, !!! compile error !!!
}
```

map

Brisanje elemenata u mapi: `erase`.

```
// erase(x) brise element s kljucem x
ocjena.erase( "Marija" );

// erase(mi) brise element na koji pokazuje iterator mi
ocjena.erase( ocjena.begin() );
```

Funkcije `clear`, `empty`, `size`, `=`, `==` rade kao i s ostalim kontejnerima.

Zadatak 3

Napišite program koji:

1. Učitava imena i ocjene sve dok ne učitamo string `"kraj"` za ime.
2. Izbacuje sve osobe s ocjenom 1.
3. Ispisuje sva imena i ocjene preostalih osoba.

multimap

`multimap<D, K>` je preslikavanje koje jednom elementu domene `D` pridružuje više elemenata kodomene `K`.

```
#include <map>
```

Operacije su kombinacija onih iz `map` i `multiset`.
Umjesto `[]` koristimo `insert`. Iterator pokazuje na uređeni par (`d`, `k`).

```
multimap<string, int> ocjene;

ocjene.insert( make_pair( "Ana", 3 ) );
ocjene.insert( make_pair( "Pero", 4 ) );
ocjene.insert( make_pair( "Ana", 5 ) );

multimap<string, int>::iterator it;
it = ocjene.find( "Ana" );         // jedna od Aninih ocjena
it = ocjene.lower_bound( "Ana" ); // prva Anina ocjena
it = ocjene.upper_bound( "Ana" ); // prva iza Aninih ocjena
```

Preostali kontejneri

U STL-u postoji još kontejnera koje nećemo detaljnije obraditi:

- `deque` - lista s efikasnim operacijama dodavanja i brisanja elemenata s oba kraja (kao **`stack`** i **`queue`** istovremeno).

C++11 uvodi još neke kontejnere:

- `array` - efikasna implementacija polja fiksne veličine.
- `forward_list` - jednostruko povezana lista.
- `unordered_set`, `unordered_multiset` - (multi)skup u kojem elementi nisu nužno sortirani, već je pristup do njih implementiran pomoću hash funkcije.
- `unordered_map`, `unordered_multimap` - preslikavanje elemenata domene na kodomenu, ne nužno sortirano po elementima domene; pristup pomoću hash funkcije.

Složeni predlošci

Parametar u predlošku može i sam biti složeni tip.
Prilikom korištenja samo treba voditi računa o tipovima podataka.

```
map<pair<int, int>, stack<string>> M;

pair<int, int> p( 3, 7 );
stack<string> S; S.push( "RP1" );

M[p] = S;

map<pair<int, int>, stack<string>>::iterator mi;
for( mi = M.begin(); mi != M.end(); ++mi )
{
    // tip od mi->first je pair<int, int>
    pair<int, int> pp = mi->first;

    // tip od mi->second je stack<string>
    mi->second.push( "Jupi!" );
}
```

Ključna riječ auto

C++11 uvodi ključnu riječ `auto`.

U donjoj deklaraciji, kompajler odredi tip od izraza `nesto`, te automatski pridruži taj tip varijabli `var`.

```
auto var = nesto;
```

Ovo može jako skratiti neke deklaracije.

```
auto i = 5;     // isto kao: int i = 5;
auto d = 3.14; // isto kao: double d = 3.14;

map<int, string> M;
// isto kao: map<int, string>::iterator mi = M.begin();
auto mi = M.begin();
```

Iz edukativnih razloga, korištenje `auto` nije dozvoljeno na kolokvijima!