Urejeni pari

1

- urejeni pari se v matematiki pogosto pojavljajo
- C++ nudi tip `pair`
  - vključiti moramo `#include <utility>`
- deklaracija urejenega para

```
pair<tip1, tip2> ime;
```

  - elementa v urejenem paru ne rabita biti istega tipa

PRIMER:

```
// deklarirajmo par x (celo število, realno število)
pair<int, double> x;
```

Dostop do elementov v paru

2

- elementi v paru so javno dostopni
- imena komponent so naslednja:
  - `first` ... prvi element v paru
  - `second` ... drugi element v paru

PRIMER:

```
pair<int, double> x;
x.first = 2;
x.second = sqrt(2);
// x = (2,√2)
```

Še o parih

3

- pari so preprost način, da funkcija vrne dve vrednosti
  - namesto da vračamo dve vrednosti po referenci, vrnemo par
- funkcija `make_pair(a,b)`
  - ustvari in vrne urejen par (a,b)
  - a in b sta lahko poljubnega tipa
- urejene pare lahko primerjamo
  - z operatojema == in != (če sta definirana za posamezna tipa)
  - z operatorjem < (če je definiran za posamezna tipa)
    - v tem primeru je primerjava parov z < leksikografska

Pari - primer

4

```
#include <utility>
#include <cstdlib>

// funkcija, ki vrne meta dveh kock
pair<int, int> vrziKocki() {
      int x = rand()%6+1;
      int y = rand()%6+1;
      return make_pair(x,y);
}
```

Preslikava - map

5

- C++ omogoča, da definiramo preslikavo, ki iz poljubne (končne) množice slika v poljubno množico
- spomnimo se iz matematike
  - matematična funkcija $f: A \rightarrow B$ je množica urejenih parov $\{(a,b), a \in A, b \in B\}$
  - velja naslednja lastnost: $(a,b),(a,c) \in f \Rightarrow b = c$
  - a imenujemo neodvisna spremenljivka
  - b imenujemo odvisna spremenljivka

Preslikava - map

6

- v C++ je `map` podatkovna struktura, ki vsebuje urejene pare (ključ, vrednost)
  - z lastnostjo, da je za vsak ključ *k* lahko v map-u definiran največ en par *(k, vrednost_k)*
- za uporabo strukture map moramo vključiti

```
#include <map>
```

- deklaracija strukture map

```
map<tipKljuča, tipVrednosti> ime;
```

- tipa sta lahko poljubna z zahtevo:
  - `tipKljuča` mora imeti definiran operator <

Deklaracije za nadaljevanje

7

- naj v nadaljevanju veljajo naslednje deklaracije

```
// f slika iz podmnožice celih št.
// v realna št.
map<int, double> f;
int k;
double v;
```

Osnovne operacije za map

8

- **vstavljanje para (k, v)** oz. **definiranje f(k)=v**
  - najlažji način za vstavljanje para je uporaba operatorja [ ]
    `f[k] = v;`
  - uporabimo lahko tudi razredno funkcijo `insert(pair)`
    `f.insert(make_pair(k,v));`
    - če je vrednost f(k) že definirana, potem se s tem stara vrednost prepiše

Osnovne operacije za map

9

- **preveri, ali je *f(k)* za dani ključ *k* definiran**
  - uporabimo razredno funkcijo `count(ključ)`
  - funkcija vrne celo število
    - `0 – f(k)` ni definiran
    - `1 – f(k)` je definiran
- **odstrani definicijo *f(k)***
  - uporabimo razredno funkcijo `erase(ključ)`
    `f.erase(k)`

Osnovne operacije za map

10

- **koliko je *f(k)*?**
  - uporabimo lahko operator [ ], npr. f[k]
    - če je `f[k]` definiran, vrne ustrezno vrednost
    - če `f[k]` ni definiran, sam definira f[k] s privzeto vrednostjo
      - privzeta vrednost za števila je 0
      - sicer privzeti konstruktor

zaradi tega je potrebno biti previden, da se ne sklicujemo na nedefinirane vrednosti

PRIMER:

```
// izpiši vrednost f[100]
// prej preverimo, ali sploh obstaja
if (f.count(100)!=0)
  cout << f[100];
```

Osnovne operacije za map

11

- **določi število parov v preslikavi**
  - uporabimo razredno funkcijo `size()`
  - vrne število parov v preslikavi
- **odstrani vse pare v preslikavi**
  - uporabimo razredno funkcijo `clear()`
- **preveri, ali je preslikava prazna**
  - uporabimo razredno funkcijo `empty()`
  - vrne *`true`*, če je preslikava prazna, in *`false`*, sicer

map – iteratorji

12

- tudi map ima definirane iteratorje
- velja to, kar smo že povedali za množice in vektorje
- do elementov lahko dostopamo tudi preko iteratorjev
- preko iteratorjev
  - NE MOREMO spreminjati prve (first) vrednosti v paru
  - lahko spreminjamo drugo (second) vrednost v paru
- z iteratorji dostopamo do parov leksikografsko glede na ključe (prvi element v paru)

map – iteratorji

PRIMER:

```
// izpišimo vse elemente v preslikavi
map<int, double> m;
…
cout << "V preslikavi so pari: ";
map<int, double>::iterator it;
for(it=m.begin(); it!=m.end(); it++) {
  cout <<"("<< (*it).first << ", "
       << (*it).second << ") ";
}
cout << endl;
```