Uporabne podatkovne strukture

1

vector
set
map
...

Standard Template Library - STL

2

- C++ nudi predloge za razne podatkovne strukture
- predloge so v t.i. knjižnici STL
  - knjižnica predlog
- podatkovne strukture
  - poljem podobne strukture, dinamično prilagajanje velikosti
  - neurejene zbirke podatkov (set, multiset)
  - vrste, skladi, seznami
  - ...

Množice - set

3

- razred set predstavlja končno matematično množico
- vsi elementi v množici morajo biti istega tipa
- za uporabo, moramo vključiti knjižnico set
  - `#include <set>`
  - opisani podatkovni tipi so v imenskem prostoru `std`
- deklaracija množice ima naslednjo obliko
  - `set<tip_elementov> imeMnozice;`
- v množici lahko imamo podatke poljubnega tipa, če tip ustreza naslednji zahtevi
  - tip mora imeti definirana operatorja < in ==

Množice – osnovne operacije

4

- v nadaljevanju veljajo naslednje deklaracije:

```
set<int> m;
int x;
```

- **dodajanje elementa**

```
m.insert(x);
```

doda element x v množico m, seveda, če še tega elementa ni v množici

- **odstranjevanje elementa**

```
m.erase(x);
```

odstrani element x iz množice m, seveda, če je element v množici, če ga ni, se nič ne spremeni

Množice – osnovne operacije

5

- **odstranjevanje vseh elementov**

```
m.clear();
```

odstrani vse elemente iz množice m

- **ali je element v množici**

```
m.count(x);
```

vrne, kolikokrat se element x pojavi v množici m, torej 0 ali 1

- **koliko elementov je v množici**

```
m.size();
```

vrne število elementov v množici, torej moč množice

Množice – osnovne operacije

6

- **je množica prazna**

```
m.empty();
```

vrne bool vrednost, vrne true, če je množica prazna, in false, sicer

- **ali sta dve množici enaki**

uporabljamo lahko operatorja `==` oz. `!=`

množici sta enaki, če vsebujeta iste elemente

- **kopiranje ene množice v drugo**

operator = deluje kot pričakovano

stavek S=T; prepiše množico S s kopijo množice T

Množice – iteratorji

7

- iterator je objekt, ki omogoča zaporeden dostop do elementov v množici
- iterator določa "trenutni" element množice
- z iteratorji lahko izvedemo (med drugimi) naslednje tri pomembne operacije
  - vrednost na "trenutni" lokaciji
  - premik na naslednji element v množici
  - premik na prejšnji element v množici

Množice – iteratorji

8

- iterator definiramo na naslednji način:
  `set<tip_elementov>::iterator it;`
- z razredno funkcijo `begin()` dobimo iterator na prvi element v množici
- z razredno funkcijo `end()` dobimo iterator na mesto, takoj za zadnjim elementom (ne na zadnji element!)
- iterator si lahko predstavljamo kot kazalec na element v množici
- z operatorjem * (unarno) dostopamo do vrednosti, na katero "kaže" iterator

Množice – iteratorji

9

- z operatorjem * v množicah ne smemo spreminjati vrednosti

PRIMER:

```
set<int> m;
set<int>::iterator it;
…
if (!m.empty()) {
    it = m.begin();
        cout << "Prvi element je " << *it << endl;
} else
    cout << "Množica je prazna." << endl;
```

Množice – iteratorji

10

- operator ++
  - premakne iterator na naslednji element
- operator --
  - premakne iterator na predhodni element
- sami moramo paziti, da
  - ne gremo dlje od zadnjega elementa
  - ne gremo pred prvi element
- definirani so tudi iteratorji za sprehod od zadnjega k prvemu elementu (reverse_iterator)

Množice – iteratorji

11

PRIMER:

```
// izpišimo vse elemente v množici
set<int> m;
set<int>::iterator it;
…
for(it = m.begin(); it != m.end(); it++) {
  cout << *it << " ";
}
cout << endl;
```

Multimnožice

12

- `#include<set>` omogoča še en tip – `multiset`
- multimnožica lahko vsebuje več kopij istega elementa
- deluje identično kot `set`
- edina razlika je funkcija `count()`
  - pri množicah vrne 0 ali 1
  - pri multimnožicah pa število ponovitev elementa

Prilagodljiva polja - `vector`

13

- spoznali smo statična in dinamična polja
  - pri obojih je omejitev prostor, saj ni preprostega načina, da bi polju med delovanjem programa spremenili velikost
- v knjižnici STL obstaja podatkovna struktura `vector`
- `vector` nudi funkcionalnost polja
  - do elementov dostopamo kot pri poljih
  - velikost vektorja se lahko preprosto spremeni
  - z določenimi funkcijami se samodejno prilagaja
- vključiti moramo

```
#include <vector>
```

  - opisani tip je v imenskem prostoru `std`

vector

14

- `vector` v splošnem deklariramo kot

```
vector<tip_elementov> ime;
```

- `vector` ima več konstruktorjev
  - `vector<int> v1;  // privzeti je prazen vektor`
  - `vector<int> v2(100);// ustvari vector s 100 elementi`
- v drugem primeru lahko do elementov dostopamo preko indeksov: `v2[0], …, v2[99]`
- paziti moramo, da so indeksi ustrezni
- namesto operatorja `[]` lahko uporabimo funkcijo `at(i)`, ki pa preverja, ali je indeks `i` ustrezen

vector – nekatere funkcije

15

- v nadaljevanju veljajo naslednje deklaracije:

```
vector<int> v;
int x;
```

- **dodajanje elementa na konec vectorja**

```
v.push_back(x);
```

doda element x na konec vectorja v

- **odstranjevanje zadnjega elementa**

```
v.pop_back();
```

odstrani zadnji element iz vectorja v

- **odstranjevanje vseh elementov**

```
v.clear();
```

odstrani vse elemente v vectorju v

vector – nekatere funkcije

16

- **velikost vectorja**

```
v.size();
```

vrne celo število in sicer število elementov v vectorju

- **spreminjanje velikosti vectorja**

```
v.resize(nova_velikost);
```

spremeni število elementov v vectorju na želeno velikost

- **ali je vector prazen**

```
v.empty();
```

vrne `true`, če je vector prazen, in `false`, sicer

vector – iteratorji

17

- tudi vector ima definirane iteratorje
- velja to, kar smo že povedali za množice
- do elementov lahko dostopamo tudi preko iteratorjev
- preko iteratorjev lahko tudi spreminjamo vrednosti elementov (tega pri množicah ni bilo)

PRIMER:

```
// izpišimo vse elemente v vectorju
vector<int> v;
vector<int>::iterator it;
…
for(it = m.begin(); it != m.end(); it++) {
  cout << *it << " ";
}
cout << endl;
```

vectorji – funkcije, ki delujejo z iteratorji

18

- **vrivanje elementa na neko mesto v vectorju**

```
v.insert(it, x);
```

V tem primeru potrebujemo iterator `it` na mesto, kamor želimo vriniti element x. Funkcija je časovno neučinkovita.

- **brisanje elementa iz vectorja**

```
v.erase(it);
```

Iz vectorja izbriše element, na katerega kaže iterator `it`. Funkcija je časovno neučinkovita.

PRIMER:

```
// recimo, da je v vectorju v 10 elementov
vector<int> v;
…
vector<int>::iterator it = v.begin();
// odstranimo element na indeksu 5
v.erase(it+5);
```