Konstantni tipovi i funkcije članice

- Prevodilac zapravo sprovodi ista opšta pravila za pokazivače na (ne)konstantne objekte, jer je:
  - u nekonstantnoj, nestatičkoj funkciji članici klase *X*, pokazivač *this* implicitno deklarisan kao pokazivač tipa *X* const* (konstantan pokazivač na *nekonstantan* objekat)
  - u konstantnoj, nestatičkoj funkciji članici klase *X*, pokazivač *this* je implicitno deklarisan kao pokazivač tipa *const X* const* (konstantan pokazivač na *konstantan* objekat)

```
class X {
public:
  X (int ii) { set(ii); }

  int read () const { return i; }
  void write (int ii) { i = ii; }

private:
  int i;
};

...

X x(0);
const X cx(1);

x.read();
x.write();
cx.read();
cx.write();
```

Prilikom poziva, *this* se inicijalizuje ovako:
const X const this = &x*, što je dozvoljeno, jer je *x* tipa *X*

Prilikom poziva, *this* se inicijalizuje ovako:
X const this = &x*, što je dozvoljeno, jer je *x* tipa *X*

Prilikom poziva, *this* se inicijalizuje ovako:
const X const this = &cx*, što je dozvoljeno, jer je *cx* tipa *const X*

Greška u prevođenju, jer se *this* inicijalizuje ovako:
X const this = &cx*, što nije dozvoljeno, jer je *cx* tipa *const X*

Copyright 2018-2024 by Dragan Milićev

Konstantni tipovi i funkcije članice

- Konstantne funckije članice su one operacije koje (konceptualno) ne menjaju "spolja vidljivo stanje objekta", što ne mora obavezno značiti da ne mogu da upišu vrednost u neki podatak član objekta
- Tipičan primer jeste situacija kada neka operacija samo izračunava neki podatak, odnosno vraća informaciju (podatak) o stanju objekta, ali je izračunavanje tog podatka veoma složena i zahtevna operacija (memorijski ili vremenski), pa je zgodno uraditi tzv. *memoizaciju* (*memoization*): pamćenje izračunatog podatka, kako se sledeći put može samo vratiti ta upamćena vrednost, bez ponovnog izračunavanja. Da bi se ta izračunata vrednost zapamtila u objektu, potrebno je upisati je u neki podatak član predviđen za čuvanje te vrednosti
- U konstantnoj funkciji članici bi prevodilac sprečio upis u taj podatak član, što se može prevazići eksplicitnom konverzijom operatorom *const_cast*:

```
const_cast<X const*>(this)->member = …
```

- U opštem slučaju, ovo može imati nedefinisane efekte, pa na jeziku C++ postoji i direktniji pristup: ovakav podatak član može se označiti kao promenljiv čak i u konstantnim objektima, navođenjem specifikatora *mutable* u deklaraciji tog podatka člana. Na primer:

```
class City {
public:
  …
  Distance getDistanceFrom (City* other) const;
protected:
  Distance computeDistanceFrom (City* other) const;
private:
  mutable map<City*,Distance> memoizedDistances;
};

…

Distance City::getDistanceFrom (City* other) const {
  if (memoizedDistances.count(other)==0)
    memoizedDistances.insert(computeDistanceFrom(other));
  return memoizedDistances[other];
}
```


Copyright 2018-2024 by Dragan Milićev