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];
}
```

Funkcija *getDistanceFrom* treba da vrati udaljenost ovog grada od datog drugog grada. Ona je konstantna, jer ne menja spolja vidljivo stanje objekta

Pomoćna funkcija *computeDistanceFrom* sprovodi zahtevan postupak izračunavanja udaljenosti između gradova

Funkcija *insert* klase *map* nije konstantna funkcija članica, pa prevodilac ne bi dozvolio njen poziv za podatak član *memoizedDistances* unutar konsatante funkcije članice ove klase; međutim, ovaj podatak član je deklarisan kao *mutable*, pa je ovo dozvoljeno

Memoizacija izračunate vrednosti

Nestalni tipovi

- Kao *cv-kvalifikator* (*cv-qualifier*) objektnog tipa, potpuno analogno sa kvalifikatorom *const*, može da se koristi kvalifikator *volatile* (nestalan, nepostojan): označava da se vrednost ovog objekta može promeniti nezavisno od toka kontrole koda u kome se koristi, tipično od strane:
  - hardvera: u memoriju u kojoj je objekat uskladišten neki hardverski uređaj može *asinhrono*, tj. potpuno nezavisno od operacija programa, u proizvoljnim, nepredvidivim trenucima, upisati tj. promeniti njegovu vrednost
  - *signala*, odnosno rutina koje obrađuju asinhrone signale (*signal handler*) koji dolaze od operativnog sistema (ili hardvera, ali koje operativni sistem pretvara u asinhrone signale programu)
- Ovaj kvalifikator nalaže prevodiocu da ne vrši optimizacije koda i premeštanja operacija sa ovakvim objektom, što preciznije znači da sve operacije koje se po semantici izvršavaju pre neke date operacije čitanja ili promene ovakvog objekta moraju završiti pre te operacije, a one koje su po semantici iza te operacije ne smeju početi pre nego što se ta operacija završi
- Drugim rečima, prevodilac će svaku operaciju čitanja ili upisa u nestalan objekat izvršiti bez optimizacija i promena redosleda koda, ne sme je pročitati ili upisati pre nego što su sve operacije pre nje završene i svaki put je čita/upisuje iznova, iako program možda ne menja vrednost tog objekta
- Kvalifikator *volatile* se koristi potpuno analogno kao i kvalifikator *const*, i sva navedena pravila vezana za pokazivače, reference i konverzije važe na potpuno isti način
- Objekat može biti istovremeno i *const* i *volatile*