Inline funkcije

- Kada se u složenom programu temeljno sprovedu svi elementi objektne dekompozicije, tipična situacija jeste ta da većina metoda (tela fukcija) ima vrlo malo linija koda, vrlo retko više od 5-10 linija; sve preko toga može biti signal da je propuštena prilika za apstrakcijom ili dekompozicijom (makar algoritamskom). Duža tela funkcija svakako nisu preporučljiva, jer smanjuju razumljivost
- Tipični ekstremni primeri su metode koje rade vrlo proste operacije, na primer:
  - samo vraćaju ili postavljaju vrednost atributa (*getter* i *setter* operacije):

```
string Person::getName () const { return this->name; }
Person& Person::setName (const string& newName) { this->name = newName; return *this; }
```

  - predstavljaju "omotače" (*wrapper*) oko neke druge operacije, sa ciljem enkapsulacije:

```
Clock* Clock::create (…) { return new Clock(…); }
```

  - delegiraju poziv jednoj ili nekim drugim operacijama, uz eventualne konverzije argumenata, kao posedica dekompozicije i lokalizacije zajedničkih delova, odnosno svođenja na već postojeće:

```
Clock::Clock (int hh, int mm, int ss) { setTime(hh,mm,ss); }
bool operator!= (const complex& c1, const complex& c2) { return !(c1==c2); }
```

- Nije neobičan utisak koji se može steći o dobro dekomponovanom programu, da na neki način "većina metoda ne radi ništa posebno, već sve prepušta drugima, samo delegira pozive drugima", a ceo program ipak radi složen posao!
- Ovo je posledica dobre dekompozicije, ali i puno implicitne semantike koja je sadržana u konstruktima jezika, kao što su npr. polimorfni pozivi, pozivi konstruktora osnovnih klasa, konverzije i slično

Copyright 2018-2024 by Dragan Milićev

Inline funkcije

- Međutim, takve funkcije koje samo delegiraju poziv drugim funkcijama prave nepotreban režijski trošak u vreme izvršavanja: smeštanje povratne adrese na stek, izvršavanje instrukcije skoka u potprogram, indirektan povratak iz potprograma preko povratne adrese skinute sa steka, pa u mnogim slučajevima i kopiranje argumenata na stek i njihovo skidanje sa steka su potpuno nepotreban trošak za funkciju koja će samo ponovo izvršiti poziv druge funkcije ili prosto pročitati ili upisati podatak
- Zbog ovoga je još odavno osmišljena optimizaciona tehnika u prevodiocima *neposrednog ugrađivanja koda* pozvanog potprograma na mesto poziva (*inlining*): umesto koda za skok u potprogram, sa prenosom argumenata i čuvanjem povratne adrese na steku, kod pozvanog potprograma se neposredno ugrađuje u kod pozivaoca; ovo uzrokuje kraće vreme izvršavanja, ali i nešto veći "memorijski otisak" (*memory footprint*), odnosno veličinu programa
- U svakom slučaju, navedeni primeri trivijalnih metoda koje delegiraju pozive drugim ili samo čitaju/upisuju podatak sprovođenjem ove optimizacije ne prave nikakav trošak, a imaju značaj u dizajnu programa (bolja dekompozicija, enkapsulacija)
- Na neki način, ovo se može smatrati korakom unazad u evolutivnom razvoju programiranja: koncept potprograma je u najstarije programske jezike uveden da bi se smanjila redundantnost i unapredila dekompozicija, tj. da se ne bi ponavljao isti kod na svakom mestu korišćenja, već se on izdvaja u potprogram koji se poziva, potencijalno parametrizovano, sa različitih mesta. Neposredno ugrađivanje u kod radi obrnutu stvar i pravi redundansu, ali je razlika velika:
    - ovaj postupak radi prevodilac potpuno atuomatski i skriveno od programera
    - redundansa postoji samo u mašinskom kodu, dok je izvorni kod dekomponovan i bez redundanse

Copyright 2018-2024 by Dragan Milićev