Rekurzija

Rekurzija

- Rekurzijom se, u opštem slučaju, naziva način definisanja (ili objašnjavanja) nekog pojma posredstvom istog tog pojma
- U programiranju se rekurzijom naziva (neposredni ili posredni) poziv jednog metoda iz tela tog istog metoda
- Rekurzija je jedan od čestih načina rešavanja problema u programiranju
- Njome se prvobitan (težak) problem deli na manje probleme, koji su po svojoj prirodi istovetni sa polaznim, ali su jednostavniji
- Novi (jednostavniji) problem se dalje deli na nove jednostavnije i tako redom

Rekurzija

- Međutim, jasno je da se postupak pojednostavljivanja ne može nastavljati beskonačno, i da se u jednom momentu mora zaustaviti
- Nizom pojednostavljenja polazni problem se svodi na trivijalni, koji se lako rešava
- Rešenje trivijalnog problema uslovljava povratni lanac formiranja rešenja od jednostavnijih ka složenijim sve do rešenja početnog problema
- Rekurzivno rešenje je najprirodnije ako su po svojoj prirodi rekurzivni
  - Problem koji se rešava, ili
  - Struktura podataka koja se koristi u rešenju problema

Rekurzija

- Rekurzivno rešenje, naravno, nije uvek ni jedino moguće, niti najefikasnije rešenje problema
- Na primer, za svako rekurzivno rešenje može se naći odgovarajuće nerekurzivno (iterativno) - koje od ta dva rešenja je bolje, ponekad je teško prosuditi
- Često je međutim rekurzivno rešenje elegantnije, kraće i čitljivije, a nerekurzivno efikasnije
- Konkretna efikasnost zavisi od računara na kom se program izvršava, od realizacije programskog jezika (kompajlera, interpretera), kao i od samog rešenja

Rekurzija

- U Javi (i ostalim programskim jezicima koji dozvoljavaju rekurziju), razlikujemo dva tipa rekurzije:
  - Direktna rekurzija (samorekurzija), kada metod neposredno poziva samog sebe
  - Indirektna (uzajamna) rekurzija, kada metod poziva samog sebe posredno preko drugih metoda koje poziva

Rekurzija: rešavanje problema

- U svakom rekurzivnom metodu mora da postoji:
  - Rešenje opšteg problema njegovom dekompozicijom na manje probleme koji se rešavaju rekurzivnim pozivom (pozivima),
  - Rešenje trivijalnog slučaja kojim se problem rešava direktno bez daljih rekurzivnih poziva,

  pri čemu dekompozicija opšteg problema mora postepeno svoditi početni problem na trivijalan slučaj (ili trivijalne slučajeve)

- Posmatrajmo za početak način funkcionisanja i efekte rekurzije na jednom jednostavnom, standardnom i nezaobilaznom primeru: izračunavanju faktorijela nekog prirodnog broja

Rekurzija: faktorijel

- Izračunavanje faktorijela broja *n* (u oznaci *n*!) je problem koji se prirodno rešava rekurzijom. Po definiciji,

$$n! = \begin{cases} 1, & n = 1 \\ n(n-1)!, & n > 1 \end{cases}$$

  što znači da pri izračunavanju *n*! rešenje svodimo na izračunavanje (*n*–1)!, potom rešavanje (*n*–1)! svodimo (na isti način) na rešavanje (*n*–2)! itd.

- Trivijalan slučaj je ako je *n* = 1, kada je rešenje 1

- Ispunjen je i uslov da se dekompozicijom početni problem postepeno svodi na trivijalan slučaj

Rekurzija: faktorijel

```
class Faktorijel {
  static int fakt(int n) {
    if (n == 1) {
      return 1;
    }
    else {
      return fakt(n - 1) * n;
    }
  }
  public static void main(String[] args) {
    int n;
    do {
      System.out.print("Unesite broj n za koji se racuna n! (n>0): ");
      n = Svetovid.in.readInt();
    } while (n <= 0);
    System.out.println("n! = " + fakt(n));
  }
}
```

Rekurzija: faktorijel

- Možemo primetiti da se nizom rekurzivnih poziva problem pojednostavljuje i izračunavanje odlaže, a da se, kada je rešen trivijalni problem, u nizu povrataka iz rekurzivnih poziva, izračunava konačno rešenje

- Memorijski prostor se zauzima za svaki novi rekurzivni poziv metoda, on je aktuelan dok se metod izvršava i oslobađa se kada je rekurzivni poziv metoda završen

- Rezultat izračunavanja iz jednog poziva prenosi se u prethodni poziv