Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Predmet: Algoritmi i strukture podataka

Pripremna zadaća za vježbu 5

Rok se nalazi u informacionom sistemu zamger.

Cilj vježbe je upoznavanje sa pojmom rekurzije. Studenti će riješiti nekoliko prolema koristeći rekurzivne funkcije, pri čemu će analizirati njihovu vremensku kompleksnost i memorijsko zauzeće.

Primjer
U nastavku slijede dvije varijante funkcije koja izračunava faktorijel (faktorijel je definisan samo nad skupom prirodnih brojeva, tako da nije bitan slučaj sa negativnim **n**):

Rekurzivna varijanta	Nerekurzivna varijanta

`int faktorijel (int n) {`
`  if (n<=1) return 1; // 0 i 1`
`  return n * faktorijel(n-1);`
`}`
	`int faktorijel (int n) {`
	`  int f(1);`
	`  for (int i(2); i<=n; i++)`
	`  f=f*i;`
	`  return f;`
	`}`

Šta se može uočiti u ova dva primjera?

- Rekurzivna varijanta je lakša za razumijevanje i bliža je matematskoj definiciji faktorijela.
- Rekurzivna varijanta je kraća.
- Broj operacija množenja koje se izvrše je isti: *n-1*.
- Nerekurzivna varijanta prividno ima dvije pomoćne varijable više.
- Svaki rekurzivni poziv zauzima određeni dodatni prostor na steku. Veličina steka je ograničena, previše rekurzije može dovesti do prekoračenja steka. Također, poziv funkcije je sporiji od prolaska kroz petlju.
- U gornjem primjeru u pitanju nije *repna rekurzija (tail-recursion)* - rekurzivni poziv funkcije faktorijel treba da bude posljednja naredba koja se izvršava u funkciji, a u ovom slučaju se nakon poziva funkcije još mora izvršiti množenje.
- Kada se koristi repna rekurzija kompajler je u mogućnosti da optimizuje kod tako da se on izvršava podjednako brzo i efikasno kao i nerekurzivna varijanta. Voditi računa da do ove optimizacije neće doći ako postoje objekti za koje se mora pozvati destruktor.

Funkcija faktorijel koja koristi repnu rekurziju glasila bi ovako (mora se uvesti drugi parametar):

```
int faktorijel( int n, int a = 1 ) {
    if ( n == 0 ) return a;
    return faktorijel( n-1, a * n );
}
```

Primjer 2

Napraviti funkciju **int fib(int n)** koja računa n-ti Fibonaccijev broj.

Fibonaccijev broj se definiše kao broj fib(n) takav da je:

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-2) + fib(n-1), & n \geq 2 \end{cases}$$

Očigledna implementacija je sljedeća:

```
int fib (int n) {
      if (n<=1) return n; // 0 i 1
      return fib(n-1) + fib(n-2);
}
```

Ova funkcija ima *eksponencijalno vrijeme izvršenja* približno[1] vremenu $O(2^n)$. Isti problem se može riješiti sljedećom funkcijom koja koristi petlju i ima O(n) vrijeme izvršenja:

```
int fib_petlja (int n) {
      if (n<=1) return n; // 0 i 1
      int pretprosli(0), prosli(1), rezultat;
      for (int i(2); i<=n; i++) {
            rezultat = pretprosli+prosli;
            pretprosli = prosli;
            prosli = rezultat;
      }
      return rezultat;
}
```

Zaključak: rekurziju treba koristiti tamo gdje ona doprinosi boljoj čitljivosti koda, ali vodeći računa o performansama rekurzije i mogućem eksponencijalnom rastu kompleksnosti kada funkcija više puta poziva samu sebe. Pokušati koristiti repnu-rekurziju kad god je to moguće.

Zadatak 1.

Koristeći funkciju fib_petlja, napraviti rekurzivnu verziju fib2_0(int n) koja ima O(n) vrijeme izvršenja i koristi *repnu* rekurziju. Dozvoljeno je dodati nove parametre u prototip pod uslovom da imaju default vrijednost.

Zadatak 2.

Napisati funkciju **int nzd(int x, int y)** koja računa *najveći zajednički djelilac (NZD)* dva broja koristeći Euklidov algoritam.

Euklidov algoritam glasi: nzd dva broja x i y je broj:

$$nzd(x, y) = \begin{cases} x & za\ y = 0 \\ nzd(y,\ x \mod y) & za\ y > 0 \end{cases}$$

1 Vremenska složenost datog algoritma preciznije iznosi $O(((1+\sqrt{5})/2)^n) \approx O(1.618^n)$.