Osnovi programiranja

2024/25

Rekurzija

- U matematici ili informatici označava postupak ili funkciju koji u svojoj definiciji koriste sami sebe
- Rekurzivne definicije su prisutne i u matematici:

Definicija prirodnih brojeva:

- 1 je prirodan broj
- Ako je n prirodan broj, onda je to i n+1

Fibonačijev niz:

- 0. član niza je 1
- 1. član niza je 1
- Svaki n-ti član niza (n>1) je suma prethodna dva ($a_n = a_{n-1} + a_{n-2}$)

Rekurzija

- U matematičkom smislu rekurzija predstavlja definisanje problema uz pomoć samog tog problema.
- Drugi način definisanja rekurzije kaže da rekurzija predstavlja način definisanja problema preko pojednostavljene verzije istog tog problema.
- Jedan primer kojim se ovo može opisati je rešavanje problema pronalaska puta do kuće (problem ćemo označiti izrazom “pronađi put do kuće”). Rekurzijom bi se ovaj problem mogao opsati u tri koraka, na sledeći način:
  1) ako si kod kuće, ostani u mestu,
  2) inače, napravi jedan korak prema kući,
  3) pronađi put do kuće.
- Tačka pod brojem 3 u ovom opisu problema predstavlja poziv istog problema iz definicije, ali posle učinjene jedne jednostavne akcije, a to je jedan korak prema kući, problem je pojednostavljen.

Rekurzija

- Opisani postupak se može uopštiti, čime dobijamo generalni algoritam koji rešava probleme rekurzijom, i on se sastoji od sledeća tri koraka:

  1) trivajlni slučaj (kojim se prekida proces izračunavanja),
  2) izvršavanje jedne akcije koja nas vodi ka trivijalnom slučaju,
  3) rekurzivni poziv.

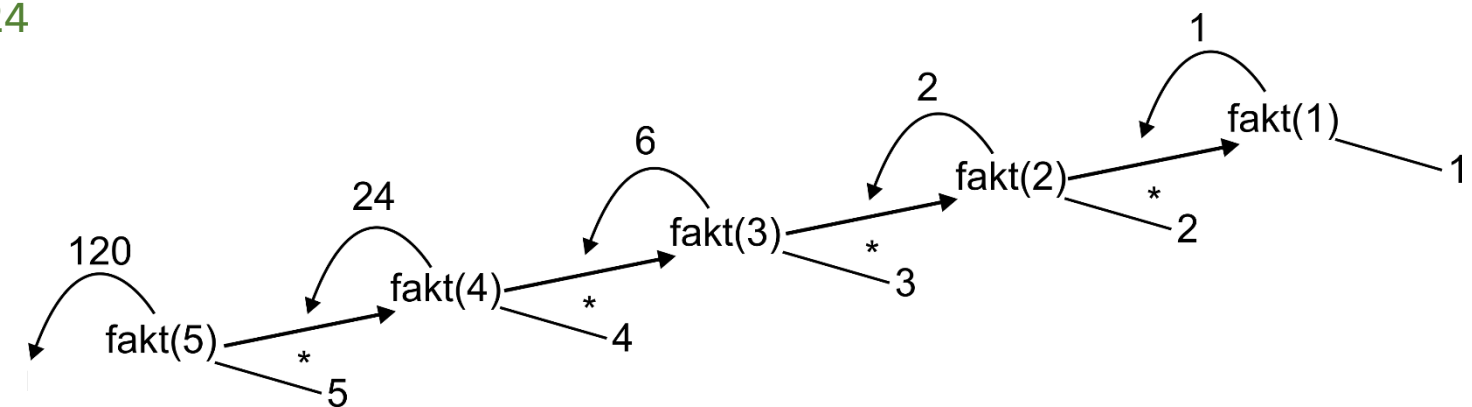
- Ovako opisani postupak rešavanja problema ima algoritamski oblik i za većinu problema se može implementirati.

Rekurzija - Faktorijel

- $0! = 1 \quad n! = n \cdot (n-1)!$

```
fakt(5) = 5 · fakt(4)
        = 5 · (4 · fakt(3))             // (jer je fakt(4) = 4 · fakt(3))
        = 5 · (4 · (3 · fakt(2)))       // (jer je fakt(3) = 3 · fakt(2))
        = 5 · (4 · (3 · (2 · fakt(1)))) // (jer je fakt(2) = 2 · fakt(1))
        = 5 · (4 · (3 · (2 · 1)))       // (jer je fakt(1) = 1)

        = 5 · (4 · (3 · (2 · 1)))       // množenje se sada vrši redom
        = 5 · (4 · (3 · 2))             // kojim se f-je završavaju, tj. unazad
        = 5 · (4 · 6)
        = 5 · 24
        = 120
```



Rekurzija

- Svako rešenje rekurizvnog problema se sastoji od dva glavna dela:
  - **Bazni slučaj(evi)** - rešenja trivijalnih instanci problema
  - **Rekurzivni slučajevi** - zavisnost rešenja problema od rešenja manjih instanci

Definicija Rekurzija predstavlja metod u kome rešenje polaznog problema nalazimo koristeći rešenja podproblema iste strukture - jednostavnije instance istog problema.

Rekurzija – često postavljana pitanja

- Funckija nije završila sa radom i ja ne mogu ponovo da je pozovem?
- Ukoliko funkcija pozove samu sebe, dobiću beskonačnu petlju?
- Kako da odredim koji su mi bazni slučajevi potrebni ili mi možda neki nedostaje?

Primer 6

- Napisati rekurzivnu i iterativnu funkciju koja racuna faktorijel celog broja.

```
#include <stdio.h>
long factorial(int n) {
    if (n == 1) return 1;
    else
    return n*factorial(n-1);
}
long factorial_iterative(int n) {
    long f = 1;
    int i;
    for (i = 1; i<=n; i++)
    f *= i;
    return f;
}
main() {
    printf("5! = %d\n", factorial(5));
}
```

```
#include <stdio.h>
long factorial(int n) {
return n == 1 ? 1 : n*factorial(n-1);
}
```

Primer 7

- Napisati rekurzivnu i iterativnu funkciju koja racuna k-ti stepen realnog broja.

```
#include <stdio.h>
float power(float f, int k)
{
    if (k == 0) return 1;
    else return f*power(f, k-1);
}
float power_iterative(float x, int k) {
    int i;float s = 1;
    for (i = 0; i<k; i++)
    s*=x;return s;
}
main()
{
    printf("%f", power(2.5, 8));
}
```

```
#include <stdio.h>
float power(float f, int k)
{
return k==0 ? 1 : f*power(f, k-1);
}
```

Primer 8

- Napisati rekurzivnu i iterativnu funkciju koja
  - računa sumu neparnih cifara datog broja,
  - određuje maksimalnu cifru broja,
  - od datog broja formira broj sa ciframa u obrnutom poretku.

Primer 8

```
#include <stdio.h>

int sumNep(int n){
        if(n==0) return 0;
        else
                if ((n%10)%2==1)
                        return n%10 + sumNep(n/10);
                else
                        return sumNep(n/10);
}
```

Primer 8

```
int maxCifra(int n, int m){

        if (n==0) return m;
        else
                return maxCifra(n/10, m>n%10?m:n%10);
}
```

Primer 8

```
int reverseNumber(int num, int reversed) {
    if (num == 0) {
        return reversed;
    }

    int lastDigit = num % 10;
    reversed = reversed * 10 + lastDigit;

    return reverseNumber(num / 10, reversed);
}
main()
{
       printf("%d\n",reverseNumber(123,0));
       printf("%d\n",sumNep(123));
       printf("%d\n",maxCifra(35082,0));
}
```

Nestandardni tipovi podataka

Nestandardni tipovi podataka

- Prosti tipovi podataka su takvi da njihove vrednosti ne možemo rastavljati na jednostavnije komponente.
- Standardni prosti tipovi podataka su:
  - celobrojni (int),
  - realni (float, double, long double),
  - logički (bool),
  - znakovni (char) .
- Iako ovi tipovi podataka u velikom broju slučajeva mogu biti dovoljni, pri rešavanju složenijih problema od velike pomoći mogu biti i neki nestandardni tipovi podataka o kojima će u nastavku biti reči.

Tipovi nastali preimenovanjem standardnih tipova podataka

- Programski jezik C omogućava da pored postojećih definiše i nove tipove podataka.
- Jedan od načina za definisanje novih tipova podataka jeste preimenovanje postojećih standardnih tipova.
- Za definisanje novog tipa podataka u programskom jeziku C koristi se rezervisana reč **typedef** iza koje se navodi naziv postojećeg tipa podataka, a zatim naziv novog tipa podataka. Na taj način definiše se novi tip podataka jednak nekom od postojećih tipova.

```
typedef <postojeci_tip> <novi_tip>;
```

Tipovi nastali preimenovanjem standardnih tipova podataka

```
typedef int ceo_broj;
typedef float realan_broj;
typedef bool logicki;
typedef char znak;
```

```
int i;
ceo_broj a,b;
znak c;
```

Nabrojivi tipovi podataka

- Često je slučaj da podaci koje želimo da koristimo u programu nisu ceobrojnog, realnog, logičkog ili znakovnog tipa. Ukoliko se radi o nabrojivom tipu podataka, jedno od rešenja je da svakoj mogućoj vrednosti novog tipa pridružimo određenu vrednost nekog od standardnih prostih tipova podataka.
- Na primer, dane u nedelji možemo obeležiti brojevima od 1 do 7, a zatim u programu koristiti brojeve kao sinonime za dane:

```
int dan;
...
if (dan==6 || dan==7) printf("Vikend\n");
```

Nabrojivi tipovi podataka

- Zamislimo sada da je potrebno napraviti program koji će računati premiju za osiguranje automobila. Visina premije koju osiguranik plaća osiguravajućem društvu zavisi od marke automobila, a za pojedine marke je potrebno doplatiti i određeni bonus zbog rizika od krađe.

  Da bismo u programu mogli da određujemo visinu premije na osnovu marke automobila, svaku marku označićemo određenim celim brojem, kao što je to prikazano u tabeli:

Marka automobila	Broj
Audi	1
BMW	2
Citroen	3
Fiat	4
Mercedes	5
Peugeot	6
Volkswagen	7
Ostali	0

Nabrojivi tipovi podataka

```
// Marku automobila oznacavamo celim brojem
int marka;
float premija;
...
//Ukoliko je marka Audi, Mercedes ili Volkswagen, uracunavamo bonus
if (marka==1 || marka==5 || marka==7) premija=premija*1.1;
```

- Problem sa ovakvim načinom predstavljanja marke automobila je u tome što bi svaki programer koji radi na ovakvom programu pored sebe morao da ima tabelu sa markama automobila i odgovarajućim brojevima.
- Drugi ozbiljan problem koji se javlja je ažuriranje spiska marki automobila. Recimo da na spisak želimo da ubacimo Mazdu, tako da zadržimo abecedni red.

Nabrojivi tipovi podataka

- Ovi problemi se mogu prevazići na taj način što bi se definisao novi, **nabrojivi**, tip podataka, koji bi mogao imati neku od vrednosti iz navedenog skupa.
- Definisanje novog nabrojivog tipa podataka obavlja se korišćenjem rezervisane reči **enum** iza koje sledi naziv novog tipa, a zatim između vitičastih zagrada spisak mogućih vrednosti.

```
enum <novi_tip> {vrednost_1,vrednost_2,...,vrednost_n};
```

Primer 1

```c
#include <stdio.h>

enum dani {ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja};
enum automobili {audi, bmw, citroen, fiat, mercedes, peugeot, volkswagen, ostali};

main()
{
    float premija = 500;

    enum dani dan = petak;
    enum automobili marka = mercedes;

    if(dan >= subota)
        printf("Ne radimo vikendom.\n");

    if(marka==audi || marka==mercedes || marka==volkswagen)
        premija = premija * 1.1;

    printf("%.2f\n", premija);
}
```

Nabrojivi tipovi podataka

- Važno je napomenuti i to da jedna ista vrednost ne može biti navedena unutar dva različita nabrojiva tipa podataka.

```
enum domaci_automobili {fica, skala, jugo, fiat};
enum strani_automobili {audi, bmw, fiat, mercedes, volkswagen};
```

Primer 2

- Napisati program koji štampa pozivne telefonske brojeve većih gradova u Srbiji.

```
#include <stdio.h>

enum gradovi {beograd, novisad, nis, kragujevac, kraljevo, cacak};

main()
{
    enum gradovi grad;
    for(grad = beograd; grad <= cacak; grad = (enum gradovi)(grad + 1))
    switch(grad)
    {
        case beograd:
                printf("011\n");
                break;
        case novisad:
                printf("021\n");
```

Primer 2

```
            break;
        case nis:
            printf("018\n");
            break;
        case kragujevac:
            printf("034\n");
            break;
        case kraljevo:
            printf("036\n");
            break;
        case cacak:
            printf("032\n");
            break;
    }
}
```