FUNKCIJE

Dragan dr Dinu

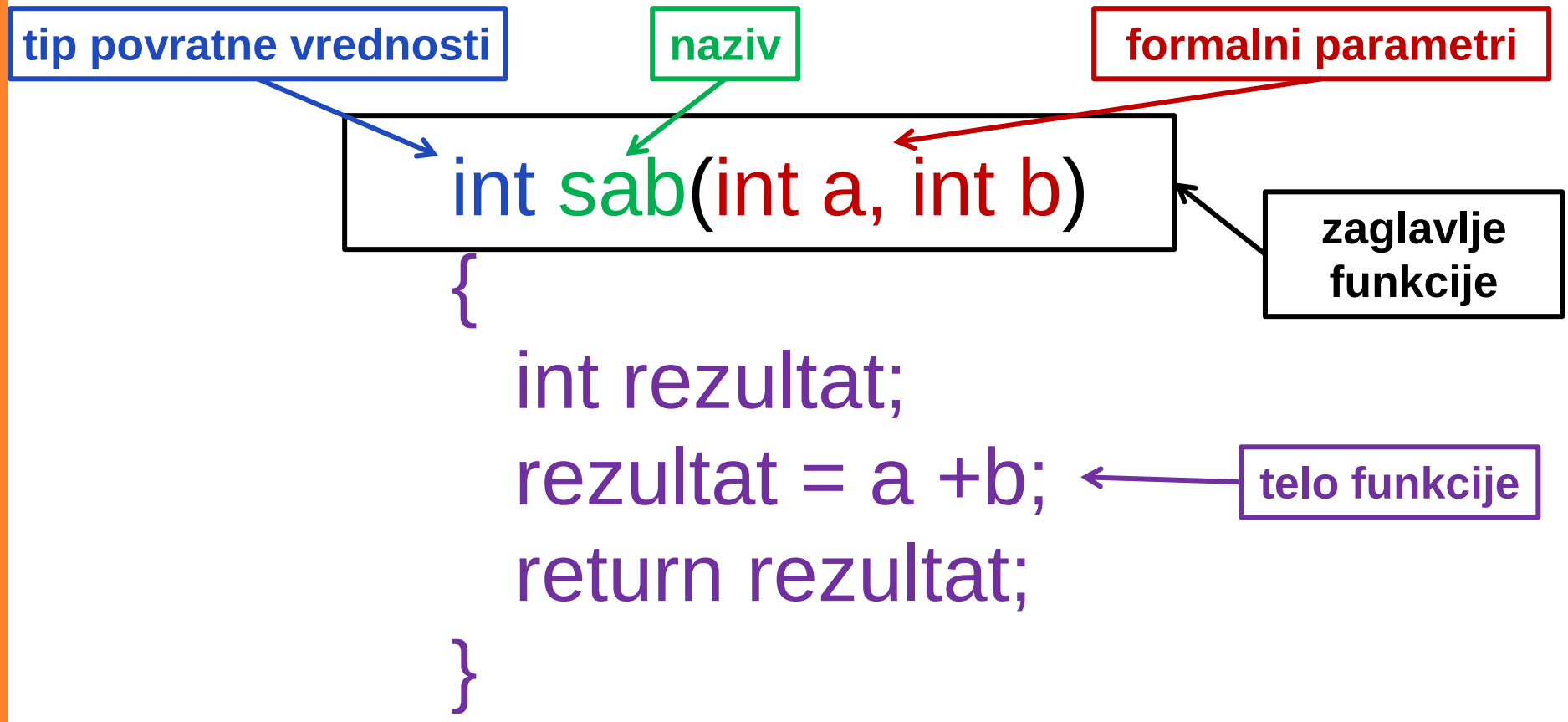
OSNOVNO O FUNKCIJAMA …

- Omogućava dekompoziciju problema na manje, lakše rešive probleme
- Omogućuju **modularnost**, ponovno **korišćenje**
- Implementiraju **semantički zatvoren posao** tako da se mogu koristiti u drugom rešenju
- Generalizuje često korišćeni deo koda, čine program mnogo razumljivijim i čitljivijim
- Omogućuje jedan nivo apstrakcije (možemo koristiti funkciju a da ne znamo kako je ona implementirana)
- Pružaju dodatni nivo apstrakcije (možemo koristiti funkciju, a da ne znamo detalje njene implementacije) – razdvajanje interfejsa od implementacije
- Dva tipa – procedure (skup naredbi) i funkcije (imaju povratnu vrednost)

… OSNOVNO O FUNKCIJAMA

- Implementiraju potprograme u jeziku C
- Identifikuje se **nazivom**
- Potrebne podatke za rad dobija putem **ulaznih** i/ili **ulazno/izlaznih** parametara
- Rezultate rada prosleđuje **nazivom** i/ili **izlaznim** parametrima
- Mogu biti ugrađene (deo pratećih C zaglavlja) ili ih program implementira i koristi
- Telo funkcije predstavlja njenu implementaciju

DEKLARISANJE FUNKCIJE



POVRATNA VREDNOST FUNKCIJE ...

- Funkcija može svojim nazivom vratiti neku vrednost
- Ova vrednost postaje vrednost izraza u kome je poziv funkcije
- Tip povratne vrednosti u zaglavlju funkcije određuje kojeg tipa će biti vrednost koju funkcija vraća
- Ako funkcija ne vraća nikakvu vrednost, povratni tip je **void**

... POVRATNA VREDNOST FUNKCIJE

- Ako funkcija vraća vrednost u telu funkcije se mora nalaziti **return** iza koje sledi izraz čiji se rezultat vraća kao povratna vrednost funkcije

- **return** vraća vrednost tipa koji odgovara tipu povratne vrednosti naveden u zaglavlju funkcije (moraju se slagati)

- **return završava izvršavanje funkcije.**
  Sav kod posle return se ingoriše !

- Može biti više **return** naredbi, ali prva na koju se naiđe završava izvršavanje programa i vraća vrednost

- Ako funkcija ne vraća vrednost, ne mora sadržati **return**

POVRATNA VREDNOST – PRIMERI


```
void f1( );            // funkcija nema povratnu vrednost

int f2( );             // povratna vrednost tipa int
short f3( );           // povratna vrednost tipa short int
unsigned int f4( );    // povratna vrednost tipa unsigned int
char f5( );            // povratna vrednost tipa char

float f6( );           // povratna vrednost tipa float
double f7( );          // povratna vrednost tipa dougle

int *f8 ( );           // povratna vrednost tipa int *
                          (pokazivač na int)
float *f9 ( );         // povratna vrednost tipa float *
                          (pokazivač na float)
```

NAZIV FUNKCIJE

- Identifikuje funkciju
- Mora biti jedinstven
- Koristi se za pozivanje funkcije negde drugde u programu
- Trebalo bi da ima semantičko značenje tako da se iz naziva lako identifikuje zadatak funkcije, primeri:

sumirajNiz, pronadjiMax, ucitajMatricu

PARAMETRI FUNKCIJE ...

- Lista parametra služi da funkcija dobije podatke potrebne za rad ili da prosledi rezultate svog rada
- Potrebno je njihovu vrednost definisati (saopštiti funkciji) svaki put kada se poziva funkcija
- Postoje funkcije koje nemaju parametre
- Svaki parametar deklariše lokalnu promenljivu koja je vidljiva samo u telu funkcije
- Vrednost ove promenljive postavlja se na vrednost koja je saopštena prilikom prilikom poziva funkcije
- Prestaje da važi neposredno nakon izlaska iz funkcije

... PARAMETRI FUNKCIJE ...

- Parametar funkcije nije isto što i promenljiva čija vrednost se prosleđuje funkciji

- Prilikom poziva funkcije, **stvarni parametri** moraju se poklapati sa **formalnim parametrima:**
  - u broju parametara,
  - tipu podatka parametra, i
  - moraju biti u odgovarajućem redosledu

- **Nazivi formalnih i stvarnih parametara ne moraju se poklapati!**

... PARAMETRI FUNKCIJE

Deklaracija funkcije

```
int funk1(int par1, float par2)
{
   // telo funkcije
}
```

Pozivanje funkcije

```
// kod iz kojeg se poziva funkcija
int arg1 = 21;
float arg2 = 22.23;
int rez;
rez = funk1(arg1, arg2);
```

- **Formalni parametri** se navode u deklaraciji funkcije i nemaju konkretne vrednosti
- **Stvarni parametri** se navode u pozivu funkcije i donose funkciji konkretne vrednosti

PRENOS PARAMETARA FUNKCIJE ...

- **C funkcija dozvoljava prenos parametara samo po vrednosti**
- **Mehanizam steka i stek frejma**
- Prilikom poziva funkcije, vrednost promenljive kopira se u parametar funkcije (koliko god da ta promenljiva zauzima memorije), te funkcija nema pristup promenljivoj
- Zbog toga ako funkcija i promeni vrednost jednom od parametara, to ne utiče na promenljivu čija vrednost je prosleđena funkciji

... PRENOS PARAMETARA FUNKCIJE ...

- Primer kako se vrednost promenljive ne menja:

Pozivanje funkcije

```
// ...
int a = 21;
funk1(a);
printf("%d", a);
// ...
```

Funkcija

```
int funk1(int pa)
{
   pa *= 10;
   return pa;
}
```

U memoriji

```
a  == 21
pa == 21
pa == 210
a  == 21
```

... PRENOS PARAMETARA FUNKCIJE ...

- Na prvi pogled u C programima ne postoji mogućnost da se deklarišu izlazne i ulazno/izlazne promenljive ?!?!!!
- **A POKAZIVAČI???**
- Ako se želi da funkcija ima **bočni efekat** kao parametar se prenosi pokazivač na promenljivu čijom se vrednošću želi manipulisati
- Bočni efekat se koristi i kada je promenljiva suviše velika (zauzima suviše memorijskog prostora) da bi se prenosila po vrednosti
- Pošto se prenosi pokazivač, izmene u funkciji su vidljive u promenljivoj
- Kako je naziv niza adresa, on se uvek prenosi kao ulazno/izlazni ili izlazni

... PRENOS PARAMETARA FUNKCIJE

- Primer kako se vrednost promenljive menja:

- Primer kako se vrednost promenljive menja:

Pozivanje funkcije

```
// ...
int a = 21;
funk1(&a);
printf("%d", a);
// ...
```

Funkcija

```
int funk1(int *pa)
{
   *pa *= 10;
   return *pa;
}
```

U memoriji

```
 a   == 21
*pa  == 21
*pa  == 210
 a   == 210
```

PROTOTIPOVI

- Zbog linearne strukture C programa
- Da bi se omogućilo kompajliranje u jednom prolazu
- Da bi se omogućilo pozivanje funkcija iz main funkcije pre njihove implementacije
- Deklarišu se tako što se navodi samo zaglavlje bez tela funkcije:

```
int sab(int a, int b);
```

- Moguće je u potpunosti izostaviti nazive promenljive jer oni nisu bitni za slaganje formalnih i stvarnih parametara:

```
int sab(int , int);
```

UMETNUTE (ENGL. INLINE) FUNKCIJE

- Česta upotreba malih funkcija usporava izvršavanje – mehanizam poziva funkcija korišćenjem steka (engl. stack) i stek frejma (engl. stack frame)
- Rezervisana reč inline je sugestija prevodiocu da pokuša da na mesto poziva funkcije direktno “umetne” mašinski kod funkcije – ubzanje rada
- Uvedene sa C99 standardom

```
inline int mnozi(int a, int b) {  // inline deklaracija funkcije
   return(a*b);
}
int main(){
   int c;
   c = mnozi(2,3);
   printf("Rezultat množenja:%d\n", c);
   return 0;
}
```

FUNKCIJE – PRIMERI ...

- **ZADATAK1:**
  Implementirati program koji računa sumu, razliku, proizvod i količnik dva broja. Brojeve zadaje korisnik preko tastature. Implementirati unos, sabiranje, oduzimanje, množenje, deljenje i prikaz rezultata kao zasebne funkcije.

- **VEŽBA1:**
  Izmeniti prethodni program tako da korisnik može da bira željenu operaciju (sabiranje, oduzimanje, množenje ili deljenje), pri čemu za svaku operaciju ponovo definiše operande. Omogućiti ponavljanje ovih operacija sve dok korisnik ne odluči da izađe iz programa.

… FUNKCIJE – PRIMERI (ZADATAK1) ...


```
#include <stdio.h>
#include <stdlib.h>

void UnesiOperande(float *op1, float *op2);
float Sabiranje(float op1, float op2);
float Oduzimanje(float op1, float op2);
float Mnozenje(float op1, float op2);
float Deljenje(float op1, float op2);
void PrikaziRezultate(float op1, float op2, float zb, float rz, float pr, float kl);

int main()
{
   float operand1 = 0, operand2 = 0;
   float zbir = 0, razlika = 0, proizvod = 0, kolicnik = 0;

   printf("\n--Program za simuliranje kalkulatora--\n\n");

   UnesiOperande(&operand1, &operand2);

   zbir = Sabiranje(operand1, operand2);
   razlika = Oduzimanje(operand1, operand2);
   proizvod = Mnozenje(operand1,operand2);
   kolicnik = Deljenje(operand1, operand2);

   PrikaziRezultate(operand1, operand2, zbir, razlika, proizvod, kolicnik);

   return 0;
}
```

... FUNKCIJE – PRIMERI (ZADATAK1) ...


```
void UnesiOperande(float *op1, float *op2)
{
   printf("Unesite prvi operand:\t");
   scanf("%f", op1);

   printf("Unesite drugi operand:\t");
   scanf("%f", op2);

   printf("\n-------------------------------------");
}

void PrikaziRezultate(float op1, float op2, float zb, float rz, float pr, float kl)
{
   printf("\n\nRezultati osnovnih operacija su:\n");

   printf("\n%4.2f + %4.2f = %4.2f\n", op1, op2, zb);

   printf("\n%4.2f - %4.2f = %4.2f\n", op1, op2, rz);

   printf("\n%4.2f * %4.2f = %4.2f\n", op1, op2, pr);

   printf("\n%4.2f / %4.2f = %4.2f\n", op1, op2, kl);

   printf("\n\n-------------------------------------\n");
}
```

... FUNKCIJE – PRIMERI (ZADATAK1)


```
float Sabiranje(float op1, float op2)
{
   return op1 + op2;
}
float Oduzimanje(float op1, float op2)
{
   return op1 - op2;
}
float Mnozenje(float op1, float op2)
{
   return op1 * op2;
}
float Deljenje(float op1, float op2)
{
   return op1 / op2;
}
```

- Šta sa deljenjem?

... FUNKCIJE – PRIMERI ...

- **ZADATAK2:**
  Implementirati program za računanje sume vrednosti elemenata niza prirodnih brojeva koji sadrži maksimum 50 elemenata. Program prihvata od korisnika broj elemenata ($0 < N \leq 50$) i vrednost svakog pojedinačnog elementa. Unos elemenata i računanje sume realizovati kao zasebne funkcije.

- **VEŽBA2:**
  Implementirati program koji od korisnika prihvata N elemenata niza (koji maksimalno može sadržati do 30 elemenata). Omogućiti korisniku da bira neku od sledećih operacija: izračunavanje sume elemenata niza, računanje srednje vrednosti niza, nalaženje minimuma, nalaženjene maksimuma. Omogućiti ponavljeno izvršavanje ovih akcija.

… FUNKCIJE – PRIMERI (ZADATAK2) ...


```
#include <stdio.h>
#include <stdlib.h>

#define MAXELNIZA 50

void UnosElemenataNiza(int *pN, int pNizPrirordnihBrojeva[MAXELNIZA]);
int SrednjaVrednost(int pN, int pNizPrirordnihBrojeva[MAXELNIZA]);

int main()
{
    int N = 1;
    int NizPrirordnihBrojeva[MAXELNIZA];

    printf("\n Program za racunanje sume elemenata niza N prirodnih brojeva.\n\n");

    UnosElemenataNiza(&N, NizPrirordnihBrojeva);

    printf("\n Suma elemenata niza izosi %d.\n\n", SrednjaVrednost(N, NizPrirordnihBrojeva));

    return 0;
}
```

... FUNKCIJE – PRIMERI (ZADATAK2)


```
void UnosElemenataNiza(int *pN, int pNizPrirordnihBrojeva[MAXELNIZA])
{
    int i;

    printf("Unesite N:\t");
    scanf("%d", pN);

    for (i = 0; i < *pN; i++)
    {
        printf("Unesite %d. element niza:\t", i+1);
        scanf("%d", &pNizPrirordnihBrojeva[ i ]);
    }
}

int SrednjaVrednost(int pN, int pNizPrirordnihBrojeva[MAXELNIZA])
{
    int i;
    int suma = 0;

    for (i = 0; i < pN; i++)
        suma += pNizPrirordnihBrojeva[ i ];

    return suma;
}
```

ZADACI ZA VEŽBU ...

- **ZADATAK3:**
  Napraviti program koji prihvata podatke o polaznicima (ime, prezime, jmbg, grad), dobijene podatke sortira po jmbg-u i prikazuje ih korisniku. Podaci se smeštaju u niz, može biti maksimalno 40 pošiljalaca. Implementirati unos, prikaz i sortiranje kao zasebne funkcije.

- **VEŽBA4:**
  Modifikovati prethodnu vežbu tako da korisnik može da bira kriterijum po kojem će se vršiti sortiranje. Implementirati program oslanjajući se na funkcije.

FUNKCIJE – PRIMERI (ZADATAK3) ...


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXPOLAZNIKA 40

typedef char TString30[31];
typedef char Tstring13[14];

typedef struct Polaznik
{
   TString30 Ime;
   TString30 Prezime;
   Tstring13 jmbg;
   TString30 grad;
} TPolaznik;

typedef TPolaznik TPolaznici[MAXPOLAZNIKA];

void UnosPolaznika(TPolaznici polaznici, int *brPolaznika);
void PrikazPolaznika(TPolaznici polaznici, int brPolaznika, TString30 vrstaSorta);
void SortiranjeJmbg(TPolaznici polaznici, int brPolaznika);
void Zamena(TPolaznici polaznici, int koga, int kim);
```

… FUNKCIJE – PRIMERI (ZADATAK3) ...


```
int main()
{
   TPolaznici listaPolaznika;
   int brPolaznika = 0;

   printf("\n--Program za unos i sortiranje liste polaznika--\n\n");

   UnosPolaznika(listaPolaznika, &brPolaznika);

   SortiranjeJmbg(listaPolaznika, brPolaznika);

   PrikazPolaznika(listaPolaznika, brPolaznika, "JMBG");

   return 0;
}
```

… FUNKCIJE – PRIMERI (ZADATAK3) ...


```
void UnosPolaznika(TPolaznici polaznici, int *brPolaznika)
{
   int i;

   printf("Unesite broj polaznika (maksimalno 40):\t");
   scanf("%d", brPolaznika);
   printf("\n-----------------------------------------------");
   for (i = 0; i < *brPolaznika; i++)
   {
      printf("\nUnesite podatke za %d. polaznika:\n", i+1);
      printf("\nIme:\t\t");

      scanf(" %[^\t\n]s",polaznici[i].Ime);
      printf("\nPrezime:\t");

      scanf(" %[^\t\n]s",polaznici[i].Prezime);
      printf("\nJMBG:\t\t");

      scanf(" %[^\t\n]s",polaznici[i].jmbg);
      printf("\nGrad:\t\t");

      scanf(" %[^\t\n]s",polaznici[i].grad);
      printf("\n-----------------------------------------------");
   }
}
```

… FUNKCIJE – PRIMERI (ZADATAK3) ...


```
void PrikazPolaznika(TPolaznici polaznici, int brPolaznika, TString30 vrstaSorta)
{
   int i;
   printf("\n\n\n\nLista polaznika sortirana prema %s:\n", vrstaSorta);
   printf("-----------------------------------------------");
   for (i = 0; i < brPolaznika; i++)
   {
      printf("\nR.br:\t\t%d", i+1);
      printf("\nIme:\t\t%s", polaznici[i].Ime);
      printf("\nPrezime:\t%s", polaznici[i].Prezime);
      printf("\nJMBG:\t\t%s", polaznici[i].jmbg);
      printf("\nGrad:\t\t%s", polaznici[i].grad);
      printf("\n-----------------------------------------------");
   }
}

void SortiranjeJmbg(TPolaznici polaznici, int brPolaznika)
{
   int i, j;
   for(i = 0; i < brPolaznika-1; i++)
      for(j = i+1; j < brPolaznika; j++)
         if (strcmp(polaznici[i].jmbg, polaznici[j].jmbg) > 0)
            Zamena(polaznici, i, j);
}
```

… FUNKCIJE – PRIMERI (ZADATAK3)


```
void Zamena(TPolaznici polaznici, int koga, int kim)
{
   TPolaznik tmpPolaznik;

   strcpy(tmpPolaznik.Ime,polaznici[koga].Ime);
   strcpy(tmpPolaznik.Prezime, polaznici[koga].Prezime);
   strcpy(tmpPolaznik.jmbg, polaznici[koga].jmbg);
   strcpy(tmpPolaznik.grad, polaznici[koga].grad);

   strcpy(polaznici[koga].Ime, polaznici[kim].Ime);
   strcpy(polaznici[koga].Prezime, polaznici[kim].Prezime);
   strcpy(polaznici[koga].jmbg, polaznici[kim].jmbg);
   strcpy(polaznici[koga].grad, polaznici[kim].grad);

   strcpy(polaznici[kim].Ime, tmpPolaznik.Ime);
   strcpy(polaznici[kim].Prezime, tmpPolaznik.Prezime);
   strcpy(polaznici[kim].jmbg, tmpPolaznik.jmbg);
   strcpy(polaznici[kim].grad, tmpPolaznik.grad);
}
```


C

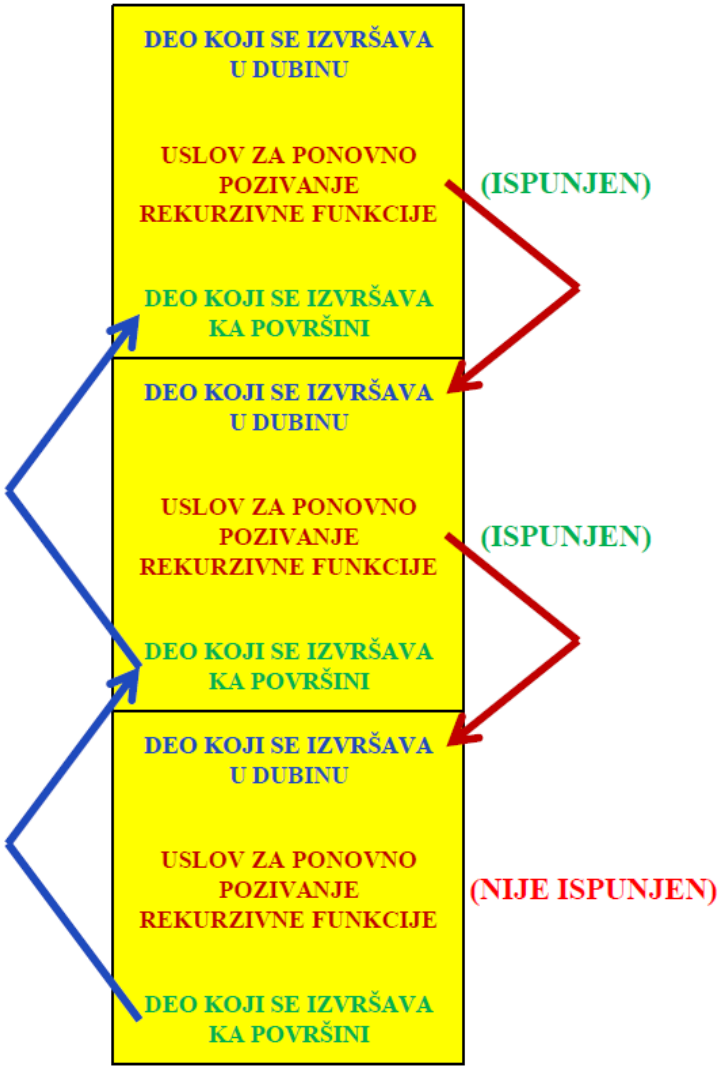
REKURZIJA

REKURZIJA

- Rekurzija nastaje kada se pojam definiše pomoću sebe samog
- Javlja se u različitim oblastima, od lingvistike i logike, preko matematike i računarstva, do umetnosti
- Primeri:
  - Binarno pretraživanje, faktorijel, fraktali, Fibonačijev niz, trougao Sierpinskog, Hanojske kule, ...
- Iterativna funkcija je ona koja koristi iteracije kako bi izvršila određeni kod veći broj puta, dok rekurzivna funkcija poziva samu sebe kako bi izvršila određeni kod veći broj puta
- Svaka iteracija može se pretvoriti u rekurziju i obratno
- Rekurzija se na nivou izvršavanja programa modeluje putem petlji (tj. uslovnih i bezuslovnih skokova) i steka

REKURZIVNE FUNKCIJE

- Situacija u kojoj je funkcija sama sebi i nadređena i podređena, tj. situacija kada funkcija poziva samu sebe
- Postoje tri dela rekurzivne funkcije:
  - **deo koji se izvršava u dubinu**
  - **uslov za ponovno pozivanje rekurzivne funkcije**
  - **deo koji se izvršava ka površini**
- Samo je uslov za ponovno pozivanje obavezan deo rekurzivne funkcije

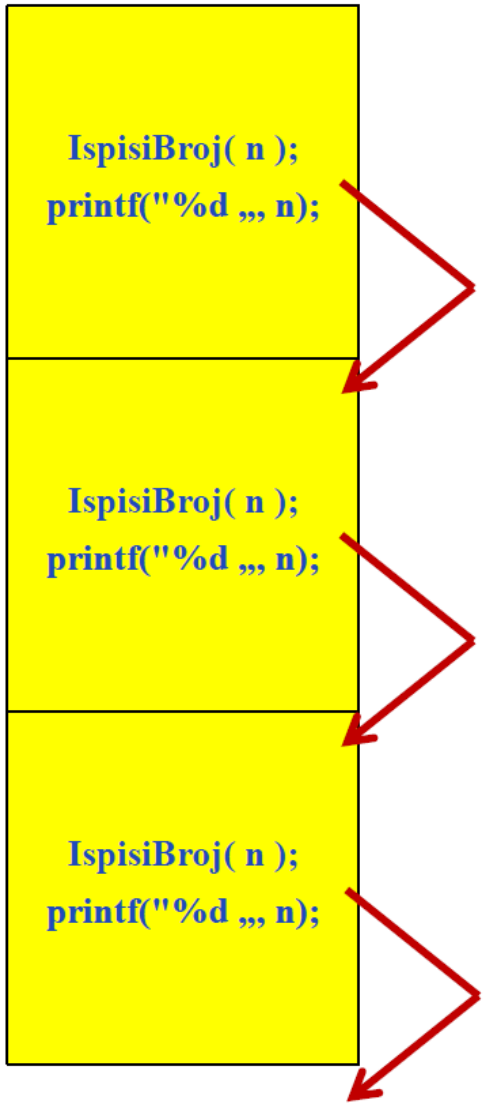


REKURZIJA – PRIMER …

- Program za ispisivanje svih celih brojeva koji slede nakon navedenog broja u obrnutom redosledu do nule (ali ne uključuje i nulu)

```
void IspisiBroj( unsigned n ) {
   IspisiBroj( n ); // Rekurzivni poziv
   printf("%d ", n);
}
```

- Šta je loše u ovoj rekurzivnoj funkciji?



… REKURZIJA – PRIMER …


```
void IspisiBroj( unsigned n ) {
  unsigned s = n – 1;
  if (s > 0)
    IspisiBroj( s ); // Rekurzivni poziv
  printf("%d ", n);
}
```

- Za:

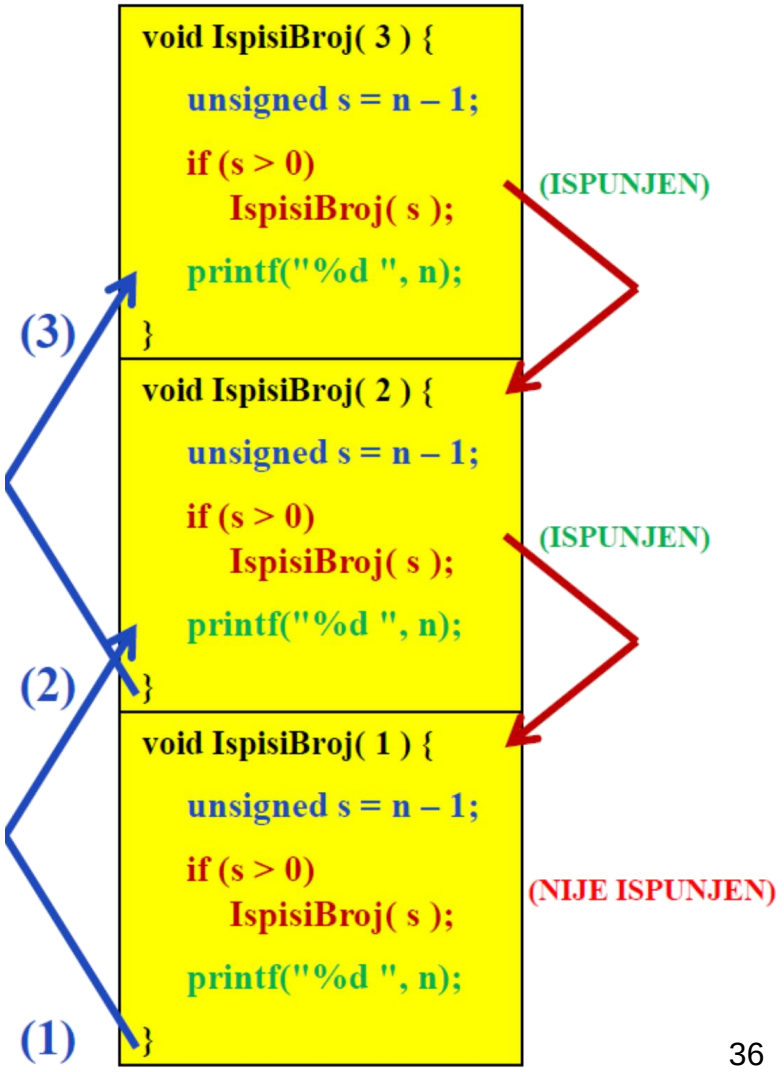
```
void main(){
   IspisiBroj( 3 )
}
```

- Da li će ispisivati:

  3 2 1

  ili

  1 2 3 ?



REKURZIJA – FAKTORIJEL …

- Rekurzivna definicija faktorijela:

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

- Izlazak iz rekurzije omogućen je osnovnim slučajem (n = 0)
- Rekurzivna funkcija za računanje faktorijela:

```
int faktorijel(int n) {
if (n <= 1)
    return 1;
else
    return n*faktorijel(n-1); // rekurzivni poziv
}
```

… REKURZIJA – FAKTORIJEL

- Iterativna funkcija za računanje faktorijela:

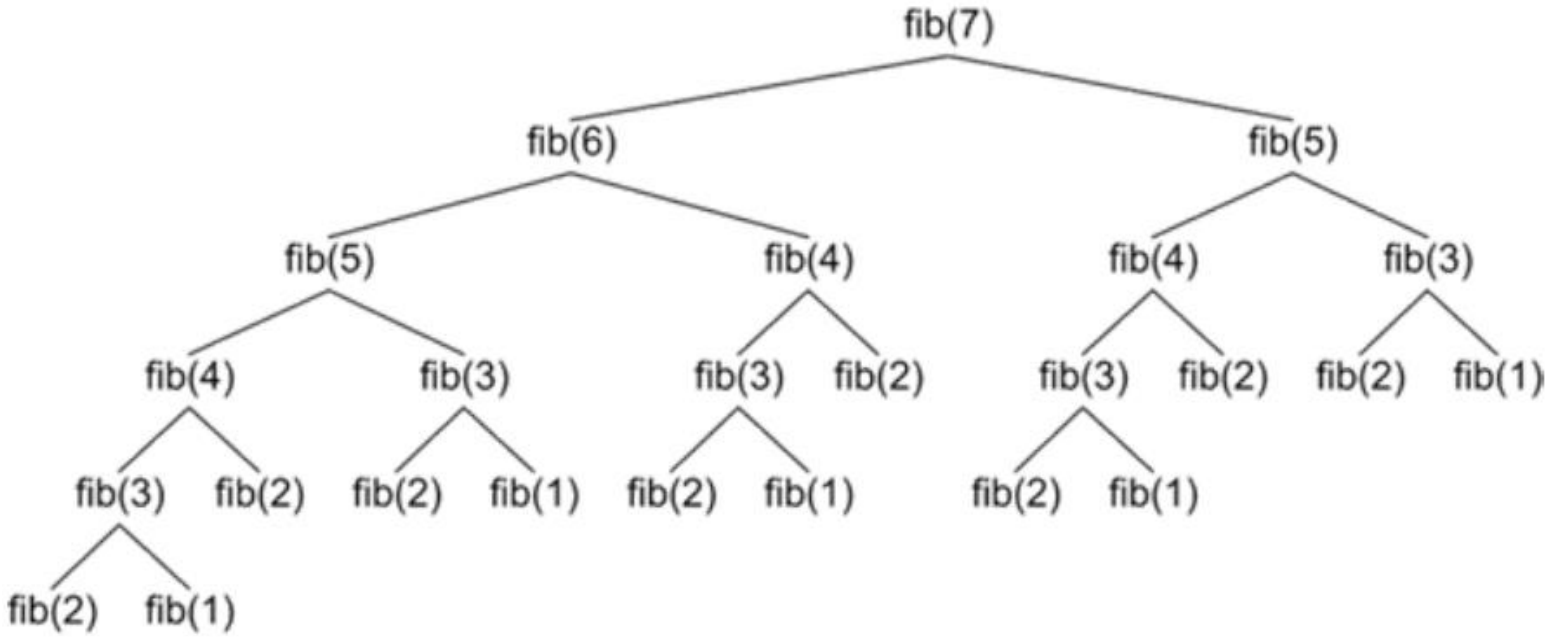
```
int faktorijel(int n) {
    int i;
    int fakt = 1;
    if (n == 1) return fakt;
    for (i = 2; i <= n; i++) {
        fakt*=i;
    }
    return fakt;
}
```

REKURZIJA – FIBONAČIJEV NIZ …

- Fibonačijevi brojevi:

$$F_n = \begin{cases} 0, n = 0 \\ 1, n = 1 \\ F_{n-1} + F_{n-2}, n \geq 2 \end{cases}$$

- Rekurzivni proces u vidu stabla (engl. tree recursion):



… REKURZIJA – FIBONAČIJEV NIZ …

- Rekurzivna funkcija za generisanje Fibonačijevog niza:

```
int fibonacciRekurzivno(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciRekurzivno(n - 1)+fibonacciRekurzivno(n - 2);
}
```

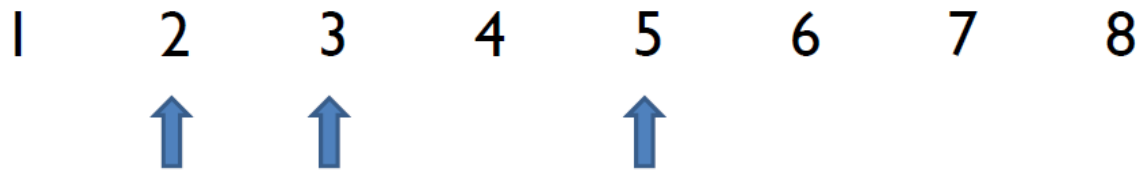
… REKURZIJA – FIBONAČIJEV NIZ

- Iterativna funkcija za generisanje Fibonačijevog niza:

```
int fibonacciIterativno(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    int pretPret = 0, pret = 1, rezultat = 0;
    for (int i = 2; i <= n; i++) {
        rezultat = pret + pretPret;
        pretPret = pret;
        pret = rezultat;
    }
    return rezultat;
}
```

REKURZIJA – BINARNO PRETRAŽIVANJE …

- Binarno pretraživanje niza (engl. binary search):



- Podeli-pa-vladaj (engl. divide-and-conquer) algoritam
- bsearch() – deo standardne C biblioteke - stdlib.h
- Različite podvarijante npr. uniformno binarno pretraživanje

… REKURZIJA – BINARNO PRETRAŽIVANJE


```
int trazi(int *podaci, int broj, int brojac) {
    // pocetak = 0 (pocetni indeks) kraj = brojac - 1 (krajnji indeks)
    return binarnoPretrazivanje(podaci, broj, 0, brojac-1);
}
int binarnoPretrazivanje(int *podaci, int broj, int pocetak, int kraj) {
    //pronadji sredinu
    int sredina = pocetak + (kraj - pocetak)/2; //celobrojno deljenje
    //uslov za zaustavljanje
    if (pocetak > kraj)
        return -1;
    else if (podaci[sredina] == broj) //pronadjen?
        return sredina;
    else if (podaci[sredina] > broj) //pod. veći od broja, trazi u "nizoj" polovini
        return binarnoPretrazivanje(podaci, broj, pocetak, sredina-1);
    else //pod. je manji od broja, trazi u "visoj" polovini
    return binarnoPretrazivanje(podaci, broj, sredina+1, kraj);
}
```

PARAMETRI KOMANDNE LINIJE

PARAMETRI KOMANDNE LINIJE …

- C programski jezik omogućuje da se od korisnika preuzmu podaci tokom pokretanja programa iz komandne linije (u savremenim računarima to bi bilo iz Command Prompt programa):

  c:\naziv_programa.exe prvi_param drugi_param treci_param

- Da bi se iskoristili parametri komandne linije, tj. da bi se parametri preuzeli, koristi se sledeći oblik main funkcije:

  int main(int BrArg , const char* NaziviArg[])

- Ukupan broj parametara prenetih main funkciji jednak je broju parametara navedenih u komandnoj liniji uključujući i sam naziv programa koji se pokreće

- Prazno mesto se koristi kao separator

… PARAMETRI KOMANDNE LINIJE

- To znači da je u slučaju navedenog primera (c:\naziv_programa.exe prvi_param drugi_param treci_param):

  BrArg == 4

  NaziviArg[0] == "naziv_programa.exe"

  NaziviArg[1] == "prvi_param"

  NaziviArg[2] == "drugi_param"

  NaziviArg[3] == "treci_param"

- Često se koristi kao deo API-ja, omogućuje pokretanje programa sa korisničkim predefinisanim parametrima ili u režimu koji korisnik zahteva

- Parametre programu prosleđuje operativni sistem

MAKROI

MAKROI ...

- Mehanizam deklarisanja netipskih konstanti, **#define**
- Omogućuje zamenu jednog identifikatora sekvencom izraza, tj. parčetu koda je dato ime
- Pretprocesor kad naiđe na identifikator, menja ga sadržajem makroa
- Postoje dve vrste makroa:
  - object-like – makroi koji ne sadrže listu parametara (u ovu grupu spadaju i netipske konstante)
  - function-like – makroi, koji poput funkcija, sadrže listu parametara
- Mogu značajno da ubrzaju program jer ne postoji redirekcija kao kod funkcija
- Kako se kod kopira onoliko puta koliko se makro poziva, mogu značajno da povećaju veličinu programa (kod funkcija je memorija samo jednom zauzeta za kod programa)

… MAKROI

- Makroi se moraju pažljivo koristiti jer mogu da dovedu do različitih ne predviđenih situacija
- Često se moraju koristiti zagrade kako bi se obezbedila korektnost upotrebljenih izraza i pravilan redosled u njihovom izvršavanju
- Izrazi sa bočnim efektom (pokazivačima) ne mogu uvek korektno da se upotrebe sa makroima
- Makroi ne mogu da se redefinišu na novu vrednost, te prvo moraju da se unište (sa **#undef** direktivom), pa da se napravi nova definicija
- Moguće je definisati više makroa sa potpuno istim sadržajem

FUNCTION-LIKE MAKROI …

- Definicija:

  #define id_makroa(lista_param) tekst_zamene

- Prvi deo definicije mora biti bez razmaka jer pretprocesor uzima da je idenfikitor makroa (sve sa listom paramara) ne prekinuti niz znakova a da je sve posle prvog razmaka tekst zamene
- Tekst zamene može biti bilo koji niz izraza (uključujući i razmake)
- Primeri:

  #define MIN(a,b) ((a < b) ? a : b)

  #define MAX(a,b) ((a > b) ? a : b)

  #define MCR(x) x * (x + 5)

… FUNCTION-LIKE MAKROI …

- Primer korišćenja za **#define MCR(x) x * (x + 5)**

```
int a = 2;
printf("a = %d \n", MCR(a));
-----------------------------------
>> a = 14
>> |
```

- Ali šta ako se desi sledeće:

```
int a = 2; int b = 3;
int c = MCR(a+b);
```

- •Da li je rešenje 50?

… FUNCTION-LIKE MAKROI …


```
c = MCR(2+3);
// c = 2 + 3 * (2 + 3 + 5) = 2 + 3 * 10
//   = 2 + 30 = 32
```

- Kako ispraviti tu situaciju?

```
#define MCR(x) (x) * (x + 5) // c = (2 + 3) * (2 + 3 + 5)
```

- Za svaki slučaj dodati svugde zagrade (nije obavezno za ovaj primer)

```
#define MCR(x) (x) * ((x) + 5) // c = (2 + 3) * ((2 + 3) + 5)
```

- Kad koristiti makro a kada funkciju?