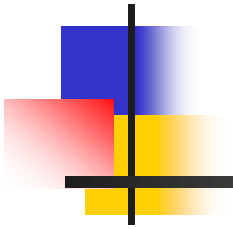
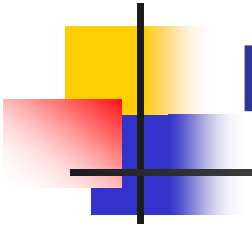


PROGRAMIRANJE II

P-02: Funkcije / napredni koncepti

prof. dr **Dražen Brđanin**

2023/24



P-02: Funkcije / napredni koncepti

- **Sadržaj predavanja**
  - pozivne konvencije
  - funkcije sa promjenljivim brojem argumenata
  - pokazivači na funkcije
  - argumenti komandne linije

Pozivne konvencije

- **Pozivna konvencija = skup pravila koja se primjenjuju prilikom pozivanja, prenošenja kontrole iz pozivajuće u pozvanu funkciju i vraćanje u pozivajuću funkciju**:
  - **način prenosa stvarnih argumenata u funkciju**:
    - preko steka, u registrima, kombinacijom stek-registar
  - **redoslijed kojim se prenose argumenti**:
    - **RTL** (*right-to-left* = zdesna ulijevo) i **LTR** (*left-to-right* = slijeva udesno)
  - **obaveze pozivajuće funkcije i obaveze pozvane funkcije**:
    - čuvanje sadržaja registara (ko čuva koje registre)
    - ko čisti stek (skidanje argumenata funkcije) nakon završetka izvršavanja pozvane funkcije
      - **caller clean-up** – pozivajuća funkcija čisti stek
      - **callee clean-up** – pozvana funkcija čisti stek
  - **način vraćanja rezultata iz pozvane u pozivajuću funkciju**:
    - u kojem registru / registarskom paru se vraća rezultat (rezultat ili adresa memorijskog bloka u kojem se nalazi rezultat)

Pozivne konvencije

- **Neke od najznačajnijih pozivnih konvencija**:
  - **caller clean-up**:
    - `cdecl (RTL)`
    - `syscall (RTL)`
    - `...`
  - **callee clean-up**:
    - `pascal (LTR)`
    - `stdcall (RTL)`
    - `fastcall (RTL)`
    - `vectorcall (LTR)`
    - `...`
- **U različitim računarskim arhitekturama i različitim kompajlerima primjenjuju se različite pozivne konvencije!**
- **cdecl** ("C declaration")
  - uobičajena pozivna konvencija koju koriste C kompajleri
  - svi argumenti prosljeđuju se preko steka
  - redoslijed prenosa argumenata: RTL
  - pozivajuća funkcija zadužena za pamćenje starog sadržaja EAX, ECX i EDX registara
  - pozvana funkcija zadužena za pamćenje ostalih registara
  - vraćanje rezultata: integer ili memorijska adresa vraća se u EAX, float/double vraća se u ST0
  - pozivajuća funkcija "čisti" stek (skida argumente funkcije sa steka)

Pozivne konvencije

Primjer:

```
...

int callee(int, int, int);

int caller()
{
   int ret;
   ret = callee(1, 2, 3) ;
   ret += 5;
   return ret;
}
```

Prevedeni (asemblerski) kod:

```
caller:
   push ebp       // zapamti sadrzaj registra koji
                     pokazuje pocetak stek segmenta
   mov ebp, esp   // novi pocetak stek segmenta
                     od trenutnog vrha steka
   push 3         // prvi argument na stek (RTL)
   push 2         // drugi argument na stek
   push 1         // treci argument na stek
   call callee    // poziv funkcije callee
   add esp, 12    // nakon povratka uvecaj esp za 12,
                     12 = broj bajtova koje na steku
                     zauzimaju argumenti (1,2,3)
   add eax, 5     // rezultat iz callee je u EAX
                     na to se dodaje 5 i to je
                     rezultat koji ce vratiti caller
   pop ebp        // sa steka skida adresu koja
                     pokazuje pocetak stek okvira
                     funkcije caller
   ret            // povratak iz funkcije caller
```

Fiksan broj argumenata u funkcijama

- **Funkcija sa fiksnim brojem argumenata**:
  - funkcija kod koje se broj argumenata ne mijenja
  - broj stvarnih argumenata mora da odgovara broju formalnih argumenata
  - deklaracije funkcija sa fiksnim brojem argumenata:

```
tip f();                                    // funkcija bez argumenata
tip f1(tip1 arg1);                          // funkcija sa jednim argumentom
tip f2(tip1 arg1, tip2 arg2);               // funkcija sa dva argumenta
tip f3(tip1 arg1, tip2 arg2, tip3 arg3);    // funkcija sa tri argumenta
...
```

  - primjer:

Deklaracija> `double suma_niza(double niz[], int n);`

Poziv> `suma_niza(temperature, 31);`

Obavezno slaganje formalnih i stvarnih argumenata po broju i redoslijedu.

Ako se tipovi ne slažu, primjenjuju se pravila za implicitne konverzije

Promjenljiv broj argumenata u funkciji

- **Funkcija sa promjenljivim brojem argumenata**:
  - Pored (uobičajenih) **obaveznih argumenata**, funkcija može da ima i **"neobavezne" argumente**
  - **Ako funkcija može da ima neobavezne argumente, kaže da se je to funkcija sa promjenljivim brojem argumenata**
  - **Funkcija** sa promjenljivim brojem argumenata **mora sama da otkrije broj i tipove neobaveznih argumenata**
  - primjeri nekih standardnih funkcija sa promjenljivim brojem argumenata:

```
// funkcija za standardni izlaz
// prima konverzioni string iz kojeg "zaključuje" koliko ima neobaveznih argumenata
// kao rezultat vraća broj ispisanih znakova
    int printf(const char *format, ...);
```

Prototip funkcije printf()

```
// funkcija za standardni ulaz
// prima konverzioni string iz kojeg "zaključuje" koliko ima neobaveznih argumenata
// kao rezultat vraća broj učitanih podataka
    int scanf(const char *format, ...);
```

Prototip funkcije scanf()

Promjenljiv broj argumenata u funkciji

- Primjer korišćenja standardnih funkcija sa promjenljivim brojem argumenata:

```
#include <stdio.h>

int main()
{
   int n;
   char tekst[100];

   printf("Unesite tekst: ");
   printf("Broj ucitanih rijeci: %d.\n", scanf("%s", tekst));

   n = printf("%s\n", tekst);
   printf("Broj znakova ispisan posljednjim printf(): %d.\n", n);

   return 0;
}
```

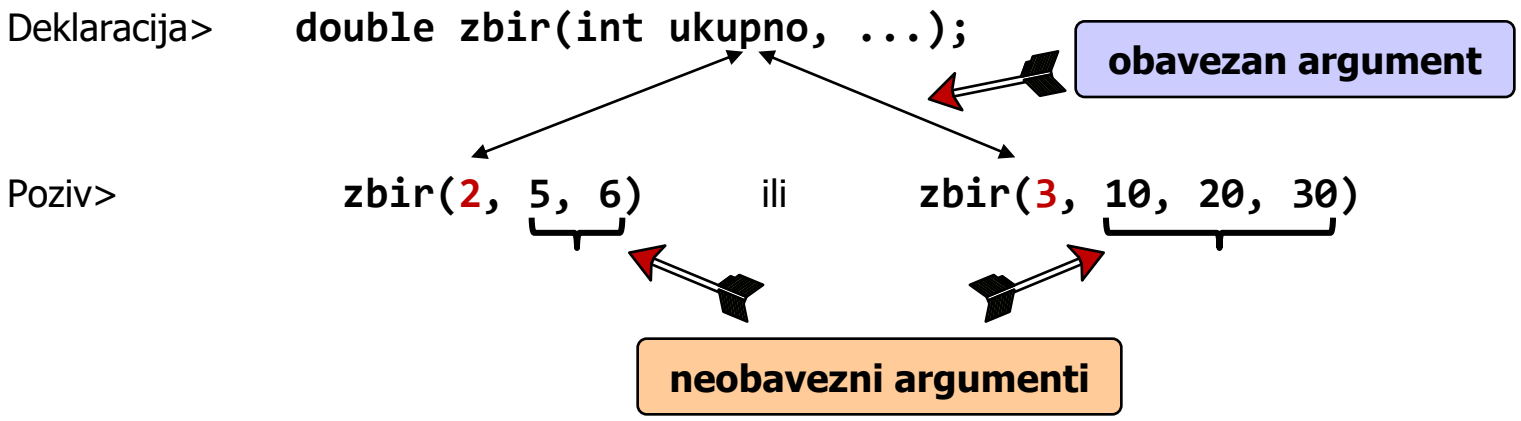
Primjer izvršavanja:

```
Unesite tekst: Banja Luka
Broj ucitanih rijeci: 1.
Banja
Broj znakova ispisan posljednjim printf(): 6.
```

Promjenljiv broj argumenata u funkciji

Funkcija sa promjenljivim brojem argumenata mora sama da otkrije broj i tipove neobaveznih argumenata!

- **"Otkrivanje" neobaveznih argumenata**:
  - **Da bi mogla da "otkrije" neobavezne argumente, funkcija mora da ima bar jedan obavezan argument na osnovu kojeg može da otkrije neobavezne argumente**
  - **Nije propisano kakav mora da bude obavezan argument**, npr.
    - **konverzioni string**: kao u slučaju printf() i scanf()
    - **broj neobaveznih argumenata**, kao u sljedećem primjeru:



Promjenljiv broj argumenata u funkciji

- **Mehanizam za pristup neobaveznim argumenatima**:

  <stdarg.h> sadrži makroe za bezbjedno dohvatanje neobaveznih argumenata:

  - **va_list – "pointerski" tip na listu neobaveznih argumenata**
    - da bi se omogućio pristup neobav. argumentima, treba **definisati promjenljivu tipa va_list**

      ```
      va_list pok_arg
      ```

  - **va_start – inicijalizacija pointera na listu neobaveznih argumenata**
    - argumenti makroa za inicijalizaciju su identifikatori pointera i posljednjeg obaveznog argumenta

      ```
      va_start(pok_arg, posljednji_obavezni_argument)
      ```

  - **va_arg – sukcesivno dohvatanje neobaveznih argumenata**
    - neobavezni argumenti se dohvataju jedan po jedan, a argumenti makroa su pointer i tip argumenta koji se dohvata

      ```
      va_arg(pok_arg, tip)
      ```

  - **va_end – završetak dohvatanja neobaveznih argumenata**

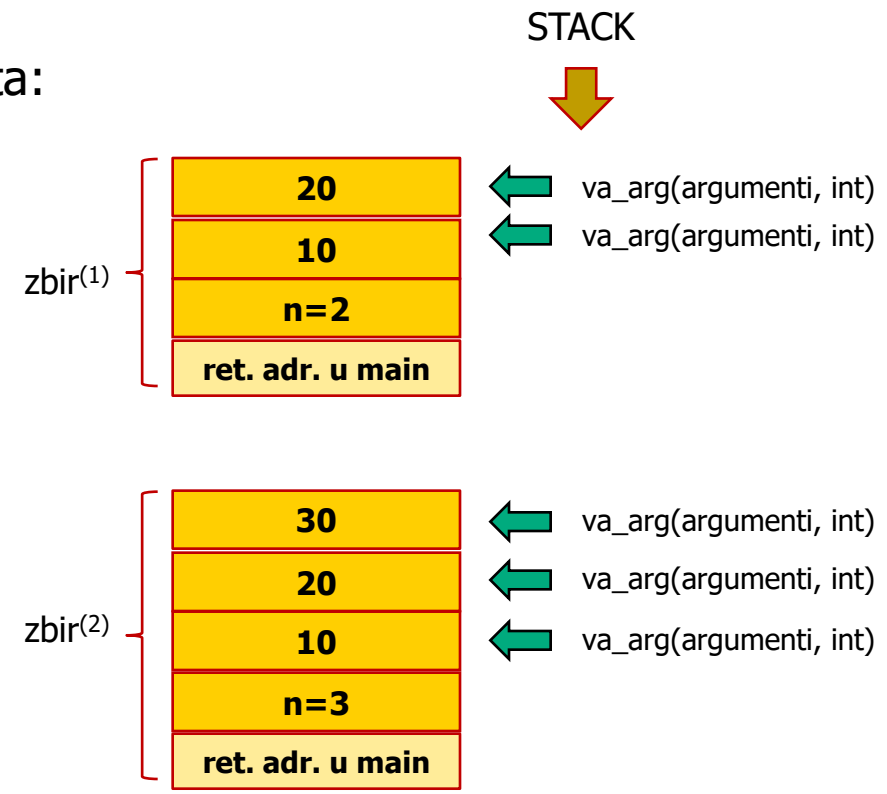
    ```
    va_end(pok_arg)
    ```

Promjenljiv broj argumenata u funkciji

- Primjer dohvatanja neobaveznih argumenata:

```
#include <stdio.h>
#include <stdarg.h>
int zbir(int n, ...)
{
   int i, s=0;
   va_list argumenti;
   va_start(argumenti, n);
   for (i=1; i<=n; i++)
      s += va_arg(argumenti, int);
   va_end(argumenti);
   return s;
}

int main()
{
   printf("Zbir(10,20)=%d\n", zbir(2,10,20));
   printf("Zbir(10,20,30)=%d\n", zbir(3,10,20,30));
   return 0;
}
```



Rezultat izvršavanja:

```
Zbir(10,20)=30
Zbir(10,20,30)=60
```

Promjenljiv broj argumenata u funkciji

- Primjer (funkcija koja spaja proizvoljan broj stringova):

```
char *konkatenacija(int n, ...)
{
   char *rezultat="", *next;
   va_list argumenti;
   va_start(argumenti, n);
   for (int i=0; i<n; i++)
   {
      int j, k;
      next = va_arg(argumenti, char*);
      int len = strlen(rezultat) + strlen(next) + 1;
      char *tmp = (char *) malloc(len);
      for (j=0; rezultat[j]; j++)
         tmp[j] = rezultat[j];
      for (k=0; tmp[j+k]=next[k]; k++);
      if (i) free(rezultat);
      rezultat = tmp;
   }
   va_end(argumenti);
   return rezultat;
}
```

Rezultat izvršavanja:

```
Banja Luka
```

```
// primjer poziva
char *s;
s = konkatenacija(3, "Banja", " " , "Luka");
printf("%s", s);
...
```

Pokazivači na funkcije

- **Definicija pokazivača:**
  - **Pokazivač je promjenljiva pokazivačkog tipa** – sadrži adresu nekog objekta

    `tip* pokazivac` ili `tip *pokazivac`
  - **Operatori za rad sa pokazivačima**
    - & – operator referenciranja (adresni operator) – daje adresu nekog objekta

      `tip x, *px=&x;` ili `tip x, *px; px=&x;`
    - * – operator dereferenciranja (operator indirekcije) – indirektan pristup lokaciji preko pointera

```
tip x, y, z, *px=&x;
y=*px;   // promjenljivoj y dodjeljuje vrijednost promjenljive x
*px=z;   // mijenja vrijednost promjenljive x
```

  - **Specijalna vrijednost koju pokazivač može da ima jeste 0 (NULL)**
    - NULL je specijalna konstanta definisana u <stdio.h>
    - Ako pokazivač ima vrijednost 0, to ne znači da pokazuje na početak memorije (nulta adresa), već da pokazivač ne pokazuje na neki objekat
    - Moguće je porediti pokazivače i u situacijama kad imaju NULL vrijednost
    - Pokušaj dereferenciranja (indirektni pristup lokaciji na adresi koju pokazuje) NULL pokazivača dovodi do *run-time* greške (greška tokom izvršavanja programa)

Pokazivači na funkcije

- **Do sada smo imali funkcije koje kao rezultat vraćaju pokazivač**
  - Prototip funkcije koja vraća pokazivač:

```
tip *funkcija(lista_argumenata);
```

ili

```
tip* funkcija(lista_argumenata);
```

  - primjeri nekih standardnih funkcija koje vraćaju pokazivač:

```
// funkcija za dinamičku alokaciju memorije
// prima zahtijevanu veličinu memorijskog bloka koji treba da se alocira
// kao rezultat vraća adresu alociranog memorijskog bloka
    void *malloc(size_t n);

// funkcija za kopiranje stringova
// prima adrese ciljnog i izvornog stringa i kopira izvorni string u ciljni
// kao rezultat vraća adresu ciljnog stringa
    char *strcpy(char *destination, char *source);
```

Pokazivači na funkcije

- **Pokazivač na funkciju**
  - Slično kao što pokazivač može da pokazuje promjenljivu, pokazivač može da pokazuje i na funkciju (jer je funkcija objekat koji smješten u memoriji i počinje na nekoj adresi)
  - **Čemu služe pokazivači na funkcije?**
    - **Funkciju nije moguće proslijediti drugoj funkciji.**
    - **Funkcija ne može da se vrati kao rezultat.**
    - **Funkcija ne može da se dodijeli promjenljivoj.**

Sve ovo je moguće pomoću pokazivača na funkcije

  - **Kako se deklariše pointer na funkciju?**

```
tip (*)(deklaracija_argumenata)
```

Primjer:

```
double (*) (int)     // deklaracija pointera na funkciju koja prima int i vraća double

void (*) ()          // deklaracija pointera na funkciju koja nema argumenata i ne vraća rezultat

int f(flaot (*) (double));      // deklaracija funkcije f() koja ima jedan formalni argument tipa
                                // pointer na funkciju koja prima double i vraća float
```

Pokazivači na funkcije

Pokazivač na funkciju ↔ **Funkcija koja vraća pokazivač**

`tip (*pf) (deklaracija_argumenata)` ↔ `tip *f(deklaracija_argumenata)`

- **Operatori za rad sa pokazivačima**

  & – operator referenciranja (adresni operator)

```
tip funkcija(deklaracija);
tip (*pf1)(deklaracija) = &funkcija;
tip (*pf2)(deklaracija) = funkcija   // moze i ovako, jer je identifikator
                                        funkcije ujedno i pokazivac na funkciju
```

  *** – operator dereferenciranja (operator indirekcije)**

```
(*pf)(stvarni_argumenti)      // poziv funkcije preko pokazivaca
```

- **Niz pokazivača na funkcije**

```
tip1 f1(tip2);
tip1 f2(tip2);
tip1 (*niz1[2])(tip2);                 // neinicijalizovani niz sa dva pokazivača
tip1 (*niz2[2])(tip2) = {&f1, &f2};    // inicijalizovani niz sa dva pokazivača
tip1 (*niz3[2])(tip2) = {f1, f2};      // moze i ovako
```

Pokazivači na funkcije

Primjer:

```
#include <stdio.h>

int inc(int x) { return ++x; }
int mul(int x) { return 2*x; }

void map(int a[], int n, int b[], int (*f)(int))
{
   for (int i=0; i<n; i++) b[i]=(*f)(a[i]);
}

void ispis(int niz[], int n)
{
   for (int i=0; i<n; i++) printf("%d ", niz[i]);
   printf("\n");
}

int main()
{
   int a[5]={1,2,3,4,5};
   int b[5];
   map(a,5,b,inc); ispis(b,5);
   map(a,5,b,mul); ispis(b,5);
   return 0;
}
```

Rezultat izvršavanja:

```
2 3 4 5 6
2 4 6 8 10
```

Pokazivači na funkcije

Primjer:

```
#include <stdio.h>

int inc(int x) { return ++x; }
int mul(int x) { return 2*x; }

void map(int a[], int n, int b[], int (*f)(int))
{
   for (int i=0; i<n; i++) b[i]=(*f)(a[i]);
}

void ispis(int niz[], int n)
{
   for (int i=0; i<n; i++) printf("%d ", niz[i]);
   printf("\n");
}

int main()
{
   int a[5]={1,2,3,4,5}, b[5];
   int (*niz[])(int) = {inc, mul};
   for (int i=0; i<2; i++)
   { map(a,5,b,niz[i]); ispis(b,5); }
   return 0;
}
```

Rezultat izvršavanja:

```
2 3 4 5 6
2 4 6 8 10
```

Pokazivači na funkcije

Primjer (funkcija za tebalarni ispis vrijednosti matematičkih funkcija):

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

double user_def(double x) { return 2*x+1; }

void tabela(double (*f)(double), char *ime, double start, double end, double dx)
{
   printf("    x      %10s\n", ime);
   printf("---------- ----------\n");
   for (double x=start; x<=end; x+=dx)
       printf("%10.4lf %10.4lf\n", x, (*f)(x));
}

int main()
{
   tabela (sin, "sin(x)", 0, 0.4, 0.1);
   printf("\n");
   tabela (user_def, "2x+1", 0, 0.2, 0.1);
   return 0;
}
```

Rezultat izvršavanja:

```
 x          sin(x)
---------- ----------
    0.0000     0.0000
    0.1000     0.0998
    0.2000     0.1987
    0.3000     0.2955
    0.4000     0.3894

    x            2x+1
---------- ----------
    0.0000     1.0000
    0.1000     1.2000
    0.2000     1.4000
```

Argumenti komandne linije

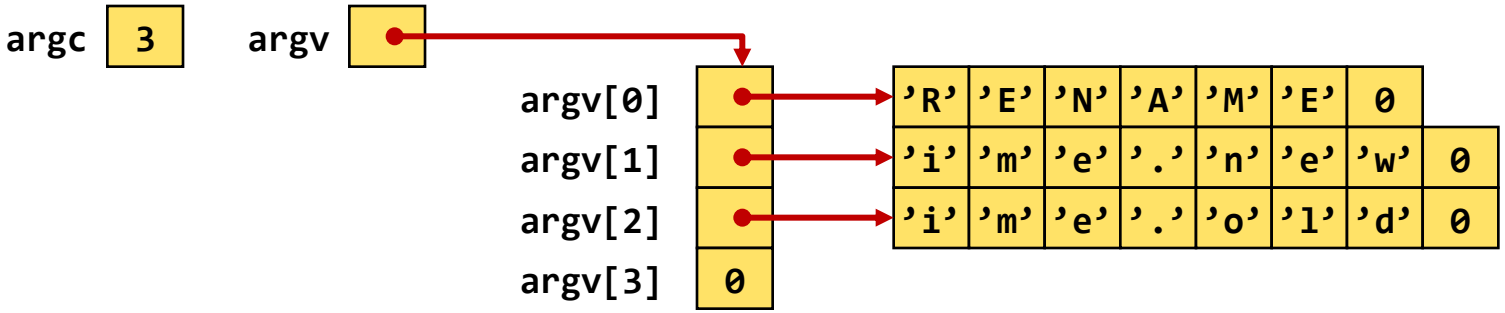
- **Funkcija main()**
  - Svaki C program mora da ima glavnu funkciju – **main()**
  - Izvršavanje C programa = izvršavanje main() funkcije
  - Funkcija main() može da vraća rezultat
    - ako ne vraća rezultat, main() je tipa void
    - ako vraća rezultat, main() je tipa int
    - rezultat se vraća operativnom sistemu i tipično reprezentuje status izvršavanja programa (bez greške – 0, greška tokom izvršavanja $\neq 0$)
  - Poziv C programa u nekim operativnim sistemima vrši se iz komandne linije
    - prilikom poziva iz komandne linije, **programu je moguće proslijediti tekstualne parametre**
    - **main() prima argumente komandne linije preko dva formalna argumenta**

Dozvoljeni prototipovi funkcije main():

```
int main();     int main(int argc, char *argv[]);     int main(int argc, char **argv);
```

Argumenti komandne linije

- **Argumenti funkcije main()**
  - prvi formalni argument: **ukupan broj stringova u komandnoj liniji**
    - uobičajeno se naziva **argc** (eng. *argument count*)
    - ne mora da se zove argc, može i drugačije
  - drugi formalni argument: **adresa niza pokazivača na stringove**
    - uobičajeno se zove **argv** (eng. *argument vector*)
    - može da se zove i drugačije
    - ukupno ima **argc+1** pointera u vektoru
    - prvi pointer(**argv[0]**) pokazuje na prvi string u komandnoj liniji = naziv pokrenutog programa
    - posljednji pointer (**argv[argc]**) je NULL pointer
- **Primjer:** `RENAME ime.new ime.old`



Argumenti komandne linije

Primjer:

```
#include <stdio.h>

int main(int argc, char **argv)
{
   printf("argc=%d\n", argc);

   for (int i=0; i<=argc; i++)
      printf("argv[%d]: %s\n", i, argv[i]);

   return 0;
}
```

Primjer izvršavanja:

```
D:\>help BL
argc=2
argv[0]: help
argv[1]: BL
argv[2]: (null)
```