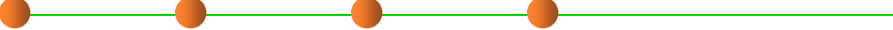




Programiranje I



Funkcije

Primjer: Dupliraj svaki drugi

- U ovom primjeru ćemo duplirati svaki drugi član niza.

```
void duplo(int *a, int n) {  
    int i;  
    for(i = 0; i < n; i += 2)  
        a[i] *= 2;  
}
```

Funkcija je deklarirana kao **void** jer ne vraća rezultat, ali sve promjene koje se izvrše na nizu se odražavaju i na argument (niz) u pozivajućoj funkciji.

- Druga tipična primjena pokazivača kod funkcija je u slučaju da funkcija treba da vrati više od jednog rezultata. Samo jedan rezultat se može vratiti direktno preko return-a, a ostali se rezultati moraju vraćati preko pokazivača.

Maksimum i pozicija maksimuma

- Napisati funkciju koja vraća maksimum i poziciju maksimuma niza. Jedan od rezultata vraćamo naredbom return, a drugi preko pokazivača. Moguća realizacija:

```
int max(int a[], int n, int *poz) {  
    int i, m = a[0];  
    *poz = 0;  
    for(i = 1; i < n; i++)  
        if(a[i] > m) {  
            m = a[i];  
            *poz = i;  
        }  
    return m;  
}
```

`poz` je pokazivač preko kojeg se vraća pozicija maksimuma. Sami protumačite funkciju.

Funkcija se može pozvati kao `a = max(x, n, &p)`.
Šta su sada `x`, `n` i `p`?

String kao argument funkcije

- String se funkciji proslijeđuje kao argument tipa **char ***, ali sada nije potrebno proslijeđivati njegovu dužinu.
- Razlog je taj što funkcija implicitno zna dužinu stringa preko terminacionog karaktera.
- Na primjer, realizacija naše verzije **strlen** funkcije bi bila:

```
int duzinaStringa(char *s) {  
    int i = 0;  
    while(s[++i] != '\0') ;  
    return i;  
}
```

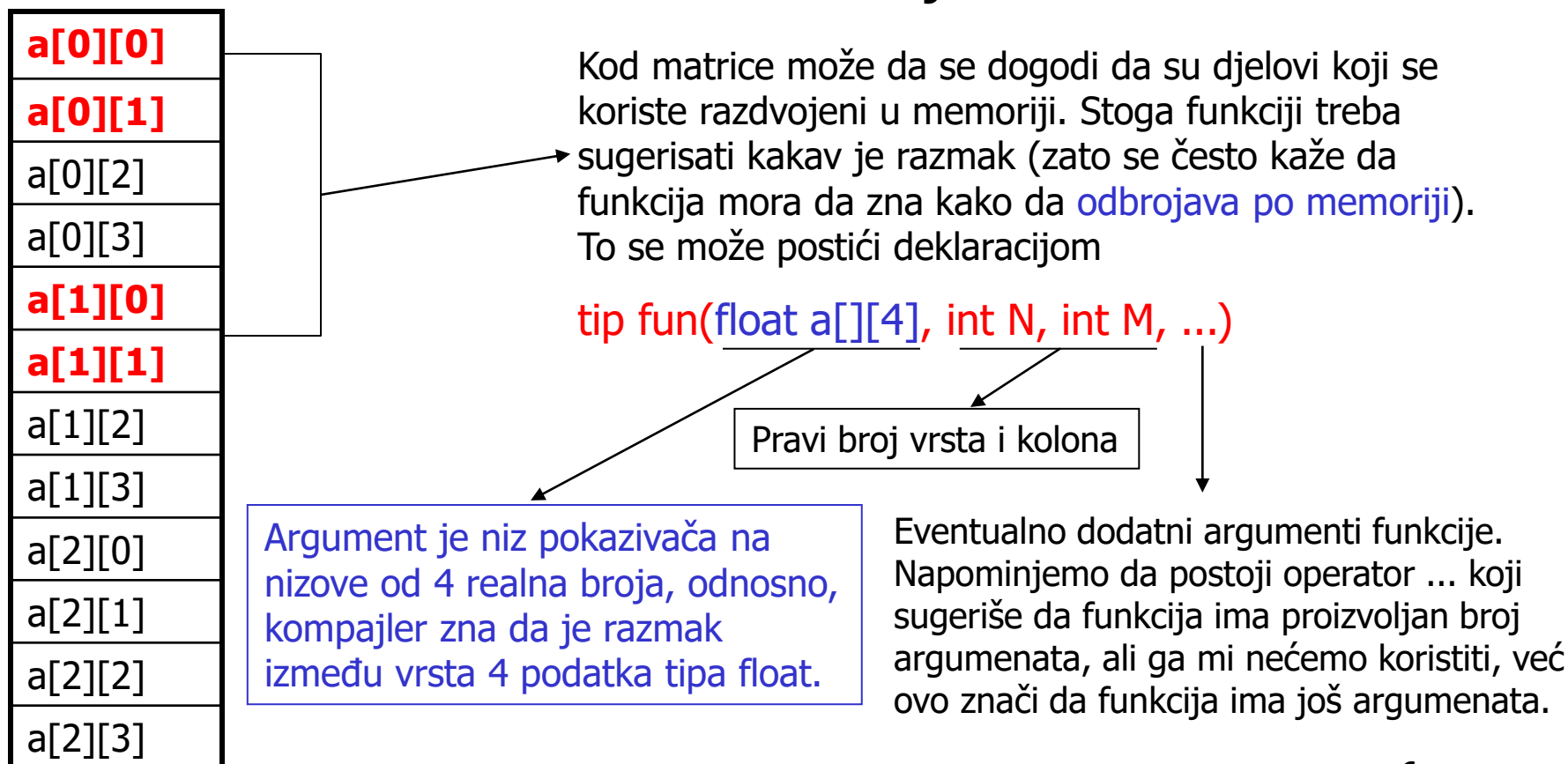
Protumačite kako ciklus koji sadrži samo praznu naredbu obavlja po nas korisnu aktivnost.

Matrica kao argument funkcije

- Na prvi pogled ne očekuje se ništa što bi razlikovalo rad sa matricama od rada sa nizovima jer se očekuje da se elementima matrice u funkciji pristupa preko pokazivača na isti način kao što je slučaj sa nizovima.
- Postoji, ipak, jedna sitna izmjena. Neka je u programu deklarirana matrica
`float m[3][4];`
- Dimenzionisanje se po pravilu vrši na najveću veličinu očekivane matrice u programu. Neka korisnik koristi dio matrice $N \times M = 2 \times 2$.

Matrica kao argument funkcije

- Elementi matrice su alocirani na sljedeći način:



Matrica - argument funkcije

- Alternativna deklaracija je:
`tip fun(float (*a)[4], int N, int M, ...)`
- Poziv naše funkcije **fun** u programu bi bio:
`fun(a, 2, 2,...)`
- Rad sa matricama u funkcijama očigledno nosi određene probleme, ali sa dosta opreza nije problem raditi ni sa ovim tipom argumenata funkcije.
- Vidjeli smo da funkcija može da vrati rezultat koji je standardnog tipa podataka uključujući modifikacije tipa unsigned, long, itd.
- Funkcija može da vrati i pokazivač.

Složene deklaracije funkcija

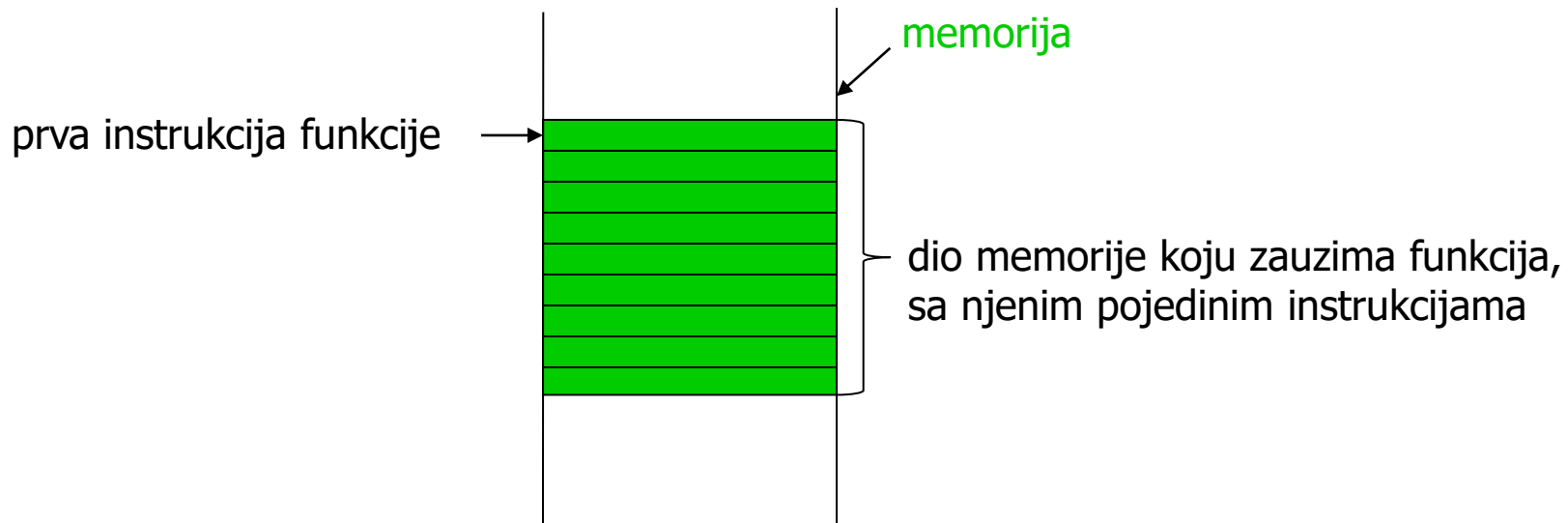
- `int *f()` je funkcija koja vraća pokazivač na cijeli broj.
- Ovo zahtijeva veliki oprez u radu, jer promjenljiva na koju dati pokazivač pokazuje ne smije da bude privremena i da nestane izlaskom iz funkcije.
- Dakle, rezultat rada funkcije koja vraća pokazivač mora da bude adresa promjenljive koju koristi funkcija (ili glavni program) koja je pozvala predmetnu funkciju.
- Česta je greška da se u funkciji deklarira niz i da funkcija vraća pokazivač na taj niz. Izlaskom iz funkcije, dealociraju se sve promjenljive alocirane u funkciji.
- Mi ćemo, kad god je to moguće (a neće biti uvijek moguće), izbjegavati ovakve funkcije.

Složene deklaracije funkcija

- Kombinacija niz-pokazivač-funkcija dovodi do veoma komplikovanih deklaracija.
- Pored pravila koja se koriste kod nizova uvodi se dodatno pravilo:
 - Par [] ima isti prioritet kao par ().
- Posmatrajmo primjer `int (*f) ()` (zanemarimo parametre):
 - U prvom koraku eliminišemo desnu zagradu; dakle zaključujemo da je `*f` funkcija koja vraća cijeli broj.
 - U drugom koraku eliminišemo `*` i zaključujemo da je `f` pokazivač na cjelobrojnu funkciju, odnosno adresa cjelobrojne funkcije.
- Pogledajte ostale primjere iz knjige.

Adresa funkcije

- Svi elementi programa zauzimaju memoriju.
- Tako i kôd funkcije zauzima memoriju.



Adresa funkcije je adresa prvog bajta funkcije, odnosno pokazivač na funkciju sadrži adresu prvog bajta prve instrukcije funkcije, što je analogno pokazivačima na sve tipove promjenljivih.

Adresa funkcije

- Adresa funkcije koja se zove `fun()` je `&fun`, ali se može koristiti i samo `fun`.
- Ovo je slično adresi niza, kod kojeg ime niza ima vrijednost adrese (prvog elementa) niza.
- Najvažnija primjena adresa funkcija je u proslijeđivanju funkcija kao argumenta drugih funkcija.
- Naime, ponekad je zgodno omogućiti jednoj funkciji da poziva više drugih funkcija u zavisnosti od situacije.

Funkcija sa argumentom funkcijom

- Zamislimo sljedeću situaciju. Napisali smo funkciju koja na osnovnu nekog sofisticiranog metoda računa integral funkcije **$\sin(x)$** .
- U slučaju da nam treba funkcija koja računa integral kosinusa morali bi da napišemo (kopiramo) prethodnu funkciju za računanje i da editujemo funkciju koja je argument (**$\cos(x)$**).
- Ovaj postupak bi morali obaviti za svaku funkciju za koju želimo da nađemo integral.
- Mala greška u našem početnom dizajnu dovodi do potrebe da prepravljamo sve funkcije, što je neprihvatljivo.

Funkcija sa argumentom funkcijom

- Drugi problem koji postoji kod pretpostavljenog rješenja je situacija kada želimo da našu funkciju komercijalizujemo.
- Korisniku bismo morali omogućiti da primijeni naš algoritam na bilo koju funkciju, a to bi značilo da korisnik mora biti upoznat sa našim kodom.
- Ovim bismo zapravo potpuno razotkrili svoju programersku vještinu i u potpunosti ugrozili komercijalizaciju našeg programa.
- Umjesto toga, naša funkcija treba da ima kao argument funkciju i da korisnik, pa i mi, sa istom funkcijom realizuje svaki projekat, a jedino znanje koje je tada potrebno o našoj funkciji je njeno zaglavlje (lista i tipovi argumenata).

Primjer

- Pretpostavimo sljedeći jednostavan problem (u svakom slučaju jednostavniji nego što je postavka problema o kojem smo diskutovali).
- Želimo da sumiramo niz:

$$\sum_{i=0}^{n-1} f(a[i])$$

gdje je $a[i]$, $i=0, \dots, n-1$ dati niz, recimo, cijelih brojeva, a $f()$ proizvoljna funkcija.

Primjer - Realizacija

```
#include <stdio.h>
int sumaFunkcija(int (*)(()), int*, int);
int linija(int);
int kvadrat(int);
```

← pretprocesor i prototipovi funkcija

```
int linija(int n) { return n; }
int kvadrat(int n) { return n*n; }
int sumaFunkcija(int (*r)(), int *a, int n) {
    int i, s = 0;
    for(i = 0; i < n; i++)
        s += r(a[i]);
    return s;
}
```

↑
Realizacija funkcija - obratite pažnju na funkciju **sumaFunkcija**, a posebno na način sumiranja.

```
int main(){
    int a[5] = {1,2,2,3,1};
    int b = sumaFunkcija(&linija,a,5);
    int c = sumaFunkcija(kvadrat,a,5);
    printf("%d %d", b, c);
}
```

↑
Glavni program sa pozivima funkcije. Obratite pažnju na podvučeni dio i pokušajte da odgovorite na pitanje zašto rade ispravno obje varijante i zašto se nigdje ne naglašava da je podvučeni argument funkcija?

Funkcija **main**

- Glavni program ima oblik bilo koje druge funkcije – ima zaglavlje, tijelo funkcije koje obično započinje sa sekcijom za deklaraciju, nastavlja se naredbama i završava sa return.
- Postavlja se pitanje - kome funkcija main vraća rezultat?
- Funkcija main, kao i svaka druga funkcija, vraća rezultat onome ko je tu funkciju pozvao, a u ovom slučaju je to **operativni sistem**.
- UNIX/LINUX operativni sistemi lijepo rade sa ovakvim rezultatima, koji obično sugerišu da je funkcija sa uspjehom obavila posao.

Parametri funkcije **main**

- Do sada smo koristili funkciju **main** bez parametara. Međutim, ova funkcija može imati parametre, a to su uvijek **cijeli broj** i **niz pokazivača na karaktere**.
- Ako funkcija **main** ima parametre, ona se deklariše kao:
int main (int argc, char *argv[])
- Što su ovi parametri i kako im zadati vrijednost?
- Objasnimo na primjeru. Kreirali smo program i kompajlirali ga, što je rezultovalo fajlom **test.exe**. U komandnoj liniji oper. sistema, program se izvršava pozivom imena **test**.
- Zamislimo sada da je program pozvan sa
test abc -4.5 57

Parametri funkcije **main**

- Vrijednost `argc` će biti **4**, dok će pojedini stringovi biti:

`argv[0] = "test"`

`argv[1] = "abc"`

`argv[2] = "-4.5"`

`argv[3] = "57"`

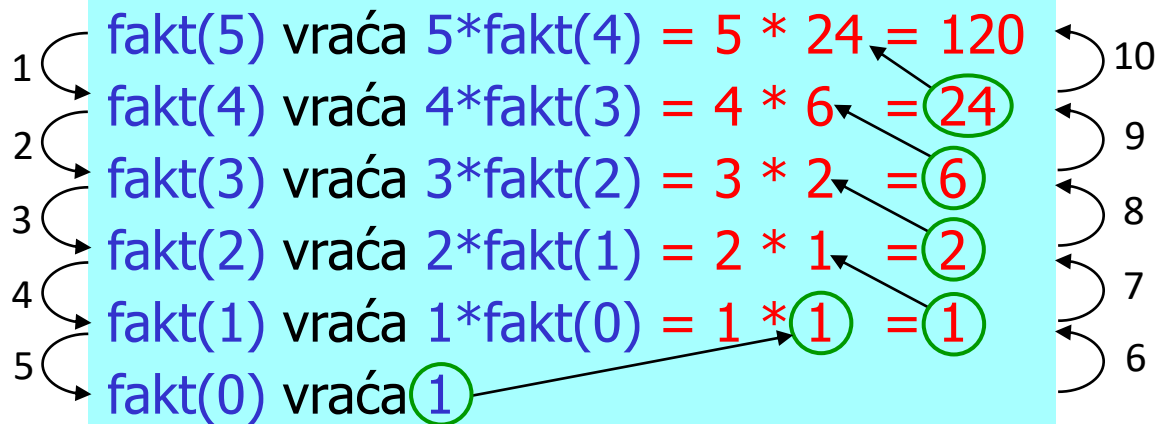
← Ime exe fajla je uvijek prvi string

- Za vježbu napisati program koji sa komandne linije učitava dva stringa. Program vrši njihovo nadovezivanje i štampanje.
- Za vježbu napisati program koji sa komandne linije učitava niz cijelih brojeva. Potrebno je izvršiti štampanje ovog niza.

Rekurzija

- Savremeni programski jezici dozvoljavaju da funkcija pozove samu sebe, što se naziva **rekurzijom**.
- Rekurzivno se funkcija faktorijel može realizovati korišćenjem pravila: $n! = n * (n-1)!$ i $0! = 1$.

```
int fakt(int n) {  
    if(n==0)  
        return 1;  
    else  
        return n * fakt(n-1);  
}
```



Rekurzija

- Rekurzivna rješenja su veoma elegantna. Neki programerski problemi se veoma teško rješavaju bez korišćenja rekurzije.
- Rekurzija može biti memorijski zahtjevna (što se dešava prilikom poziva `fakt(120)`), ali i vremenski, jer operacije sa stekom i alokacionim zapisom traju neko vrijeme.
- Stoga, rekurzivno treba rješavati one probleme kod kojih je prirodno koristiti rekurzivni pristup (binarna pretraga, quick sort itd.).
- Kao primjer loše rekurzije, realizovati rekurzivnim putem računanje **Fibonacci-evih brojeva**, za koje važi $FIB(1) = 0$, $FIB(2) = 1$ i $FIB(I) = FIB(I-1) + FIB(I-2)$ za $I > 2$.
- Program realizujte rekurzivno i odredite broj pozivanja funkcije FIB za $I=6$ ili $I=15$. Zatim dajte alternativno iterativno rješenje.