Programski jezik I

Stringovi (nastavak)

Funkcije

Korisne funkcije iz string.h

- strcpy(tar, or) kopira string or (od origin) u string tar (od target).
- strcat(tar, or) nadovezuje string or na kraj stringa tar.
- strcmp(a, b) leksikografski poredi stringove a i b.
  - ako je string a veći od stringa b vraća pozitivan broj;
  - ako su stringovi jednaki vraća nulu;
  - ako je string a manji od stringa b vraća negativan broj.
  - String a je leksikografski veći od stringa b ako je:
    - ASCII kod prvog karaktera stringa **a** veći od ASCII koda prvog karaktera stringa b;
    - ukoliko su prvi karakteri isti porede se naredni.
    - Stringovi su jednaki ako su im svi karakteri jednaki.
  - ASCII kod terminacionog karaktera je manji od bilo kog drugog karaktera i u C-u je jednak 0.

Korisne funkcije iz string.h

- strstr(tar, or) traži podstring or u stringu tar i vraća pokazivač na prvo pojavljivanje takvog podstringa.
- strchr(tar, c) i strrchr (tar, c) traže prvo i posljednje pojavljivanje karaktera c u stringu tar, respektivno.
- strspn(tar, or) i strcspn(tar, or) prebrojavaju početne karaktere stringa or koji se pojavljuju u stringu tar, odnosno koji se ne pojavljuju u njemu, respektivno.
- Funkcije atoi(s), atol(s) i atof(s) konvertuju broj sadržan u stringu s u int, long int i double broj, respektivno.
- Funkcije itoa(N, s, B) i ltoa(N, s, B) konvertuju int N i long N u string s u brojnom sistemu sa osnovom B (zaglavlje stdlib).

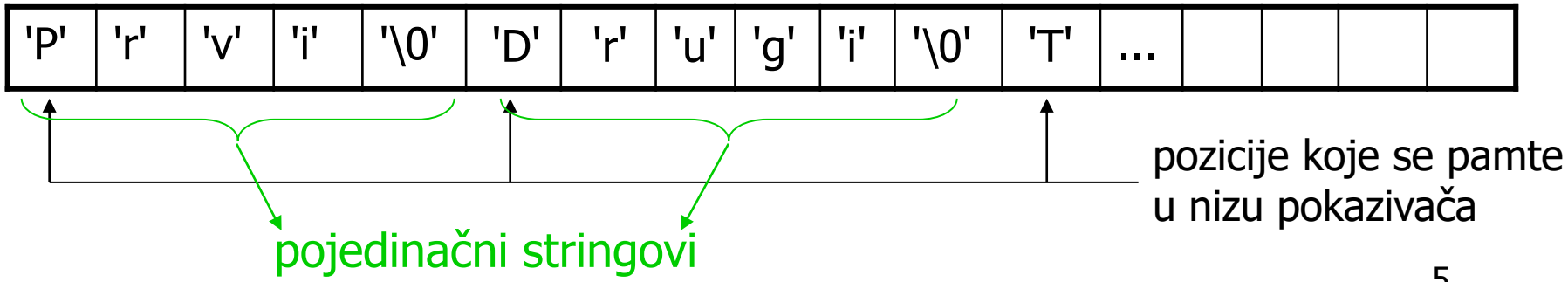
Rad sa mnoštvom stringova

- Aplikacije koje rade sa stringovima obično rade sa velikim brojem stringova.
- Svaki string se mora dimenzionisati na najveću očekivanu dužinu.
- Na primjer, baza prezimena se mora dimenzionisati na najveće očekivano prezime od, recimo, 15 slova, dok većina prezimena kod nas ima manje od 10.
- Gora varijanta je za riječi opšte namjene, gdje se dimenzionisanje mora obaviti sa 20 i više slova, dok mnogo riječi, npr. veznici, imaju samo po nekoliko slova.

Rad sa mnoštvom stringova

- Da bi se izbjegla ova nepotrebna memorijska zahtjevnost, uobičajeno se koristi sljedeći trik.
- U jednom nizu karaktera (namjerno sada koristimo ovaj pojam) se čuvaju svi stringovi razdvojeni terminacionim karakterima. Pozicije početaka pojedinih stringova se pamte preko pokazivača.

Niz karaktera



Rad sa mnoštvom stringova

- Realizacija načina rada sa mnoštvom stringova nije stvar kompajlera, već programerske vještine.

```
#include<stdio.h>
int main(){
char s[1000], *p[200], temp[20];
int size = 1, tp = 0, i, j = 0;
p[0] = s;
while(size) {
    gets(temp);
    size = strlen(temp);
    for(i=0; i<size; i++) s[tp+i] = temp[i];
    s[tp+size] = '\0';
    tp += size+1;
    if(size) p[++j] = s+tp;
}
for(i=0; i<j; i++)
    printf("%s\n", p[i]);
}
```

nastavak gore

Rad sa mnoštvom stringova

- Objasnimo prethodni primjer.
- Deklarisali smo string i niz pokazivača na stringove.
- U while petlji ostajemo sve dok je učitani string dužine veće od nula, odnosno dok korisnik ne unese prazan string.
- Svaki uneseni string se pozicionira na odgovarajuće mjesto, uz ažuriranje promjenljive tp, koja pamti poziciju dokle se stiglo u "velikom stringu".
- Nakon unosa stringa postavlja se terminacioni karakter na odgovarajuće mjesto i podesi da odgovarajući pokazivač iz niza pokazivača pokaže na naredni string čiji se unos očekuje.
- Na kraju ovog demonstracionog programa smo iz "velikog stringa" štampali pojedinačne stringove.
- Program nije optimalan ni direktno praktično upotrebljiv, već je samo ilustrativni primjer.

String literali

- Pokazivač na tip char se može inicijalizovati na sljedeći način:

  ```
  char *p = "Test";
  ```

- Na ovaj način je deklarisan pokazivač na read-only memoriju koja sadrži string literal "Test".
- Pokušaj izmjene ovog stringa, na primjer

  ```
  p[0] = 'B';
  ```

  dovodi do greške prilikom kompajliranja.
- Tumačimo izlaz sljedećeg koda:

```
char a[] = "Test";
char *b = "Test";
printf("%p\n%p\n%p\n%p", &a, a, &b, b);
```

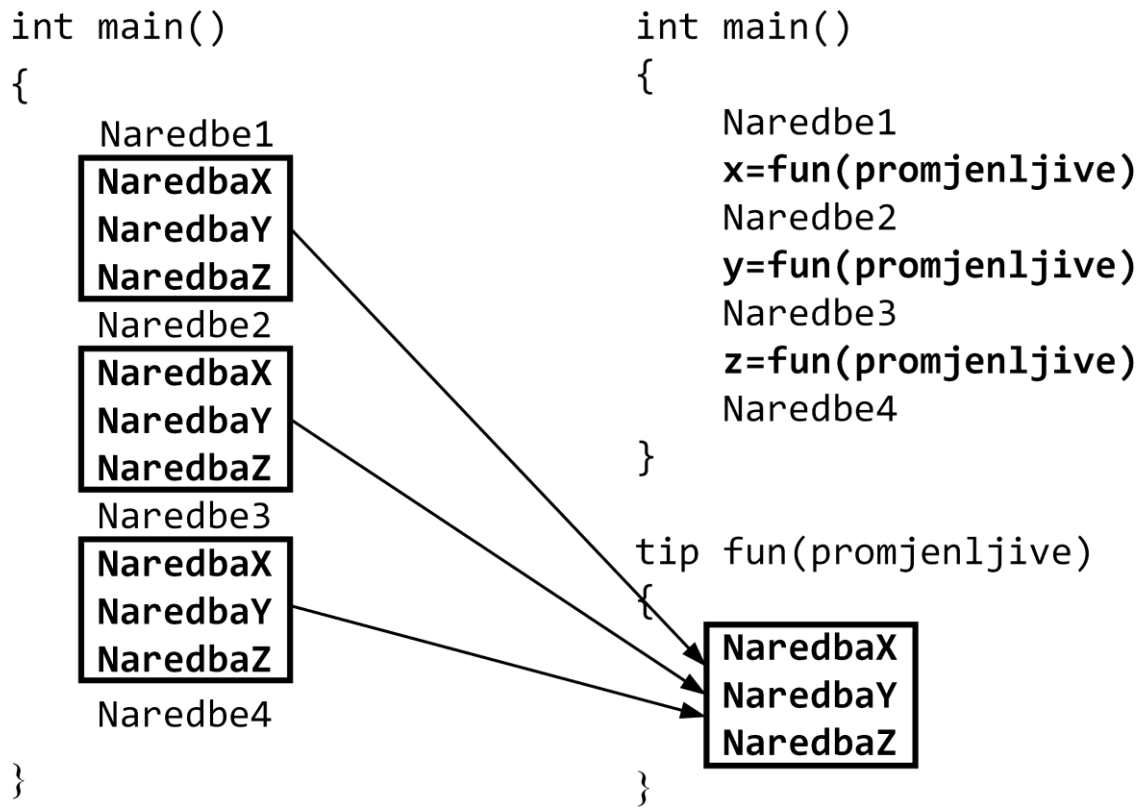
Izlaz

```
000000000061FE1B
000000000061FE1B
000000000061FE10
0000000000404000
```

String literal se smješta u drugoj sekciji RAM-a

Funkcije - Osnovno

Kada se u programu ponavlja više linija koda, dobra praksa je grupisati te linije u obliku funkcije, čime se pojednostavljuje održavanje i modifikacija programa.



Funkcije - Osnovno

- Funkcija predstavlja programsku cjelinu koja izvršava određeni zadatak.
- Pomoću funkcija se složeni programski zadaci dijele na jednostavnije cjeline. Time se postiže veća jasnoća programa i olakšava se njegova modifikacija i održavanje.
- Dobro osmišljena funkcija obavlja jedan jasno definisan zadatak.
- Korisnik dobro osmišljene funkcije ne mora poznavati detalje njene implementacije da bi je koristio.
- Funkcija može da ima ulazne podatke, i može da vrati rezultat svog izvršavanja onoj funkciji koja ju je pozvala.

Definicija funkcije

- Definicija funkcija ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1, ..., tip_n arg_n) {
        tijelo funkcije
}
```

gdje:

  - tip_podatka predstavlja tip podatka koji će funkcija vratiti kao rezultat svog izvršavanja;
  - ime_funkcije mora biti pravilan identifikator;
  - tip_1 arg_1, ... , tip_n arg_n je lista ulaznih parametara funkcije. Deklaracije pojedinih parametara se odvajaju zarezima.

- Unutar vitičastih zagrada se navodi tijelo funkcije koje se sastoji od deklaracije promjenljivih i izvršnih naredbi, isto kao kod funkcije main.

Vraćanje rezultata funkcije

- Funkcija može da vrati rezultat svog izvršavanja, što se postiže pomoću ključne riječi **return** na sljedeći način:

  return izraz;

  gdje izraz može biti proizvoljan izraz (konstanta, promjenljiva, matematički izraz, poziv druge funkcije).
- Tip izraza treba da odgovara tipu podatka koji funkcija vraća.
- Naredba return predstavlja tačku izlaska iz funkcije, tj. ako ima naredbi nakon return, one se neće izvršiti!
- Ukoliko funkcija ne vraća rezultat, ona se deklariše kao **void**:

```
void ime_funkcije(tip_1 arg_1, ..., tip_n arg_n) {
        tijelo funkcije
}
```

Primjer 3 funkcije

```
int zbir1(int x, int y)
{
        return x+y;
}
```

```
void zbir2(int x, int y)
{
        printf("%d", x+y);
}
```

```
int zbir3(int x, int y)
{
        printf("%d", x+y);
        return 1;
}
```

Crvenom bojom su označena **zaglavlja funkcija**, gdje se, pored imena funkcije, navodi tip njenog rezultata, a u zagradi se definišu parametri funkcije.

Ove tri funkcije su slične, ali rade tri različite stvari:

- Prva funkcija vraća rezultat koji je suma parametara;
- Druga funkcija ne vraća rezultat, već samo štampa zbir;
- Treća funkcija štampa zbir i vraća rezultat 1 koji može predstavljati indikaciju da je operacija obavljena uspješno.

Primjeri – Napomene

- Rezultati funkcija koje nijesu void mogu da se pridruže dozvoljenoj lijevoj strani izraza, ali i ne moraju:

```
c = zbir1(3,4);
zbir3(c,12);
```

- U drugom slučaju funkcija će vratiti vrijednost, ali pošto nema lijeve strane izraza, privremena promjenljiva u kojoj je sačuvana vrijednost rezultata će biti dealocirana.
- Iz glavnog programa se funkcije tipa void pozivaju bez navođenja promjenljivih na lijevoj strani znaka jednakosti:

```
zbir2(5,c);
```

Tip void

- Funkcija koja ne vraća vrijednost deklariše se kao void.
- Promjenljiva koja se deklariše kao void nema vrijednost i gotovo se nikad ne koristi.
- Funkcija koja se deklariše da vraća void * vraća pokazivač na neodređeni tip podatka.
- Promjenljiva koja se deklariše kao pokazivač na void, tj. void *b, pokazuje na promjenljivu koja je nekog od mogućih tipova podataka. Prije upotrebe ovog pokazivača, potrebno ga je konvertovati u konkretan tip.
- Ovo će biti jasnije kada budemo učili dinamičku alokaciju memorije.

Funkcija – Neki detalji

- Unutar funkcije se mogu deklarisati pomoćne promjenljive, kao u funkciji main. Tako se naša funkcija zbir1 mogla definisati kao:

```
int zbir1(int x, int y) {
        int zbir = x + y;
        return zbir;
}
```

Promjenljiva zbir se nakon završetka funkcije dealocira – briše iz memorije.

- Funkcije tipa void mogu se završiti svojom posljednjom naredbom ili im se izvršavanje može prekinuti naredbom return, bez argumenta nakon return.

Parametri i argumenti funkcije

- Parametri funkcije su promjenljive koje se navode u listi argumenata u zaglavlju, dok su argumenti vrijednosti koje se prosljeđuju funkciji iz drugih funkcija.
- Prilikom prosljeđivanja, vrijednost argumenata se kopira u parametre!

```
int main() {
   ...
   a = fun(x, 3);
   ...
}
```

Argumenti

```
int fun(int x, int y) {
   ...
}
```

Parametri

Ako u funkciji piše x++, ta promjena ni na koji način ne utiče na originalnu vrijednost argumenta iz pozivajuće funkcije, jer su to različite promjenljive!

Prototipovi funkcija

- Da bismo osigurali ispravan poziv funkcije, tj. da se ne dozvoli poziv sa neočekivanim argumentima, prije definisanja funkcije (i prije funkcije main) se navodi **prototip funkcije**.
- Prototip liči na zaglavlje, s tim što se ne moraju navoditi imena argumenata. Prototip se završava sa ;
- Primjer prototipa:

```
int zbir(int, int);   // ponekad se navode imena parametara
                      // da bi se objasnilo njihovo značenje
```

Prototip – Upotreba

```
#include <stdio.h>

int fun(int);

int main() {
    printf("%d\n", fun());
    return 0;
}

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

Prototip funkcije

Pokušaj poziva funkcije **fun** bez argumenata dovodi do greške, jer ta funkcija očekuje za argument cio broj.

Bez prototipa se ne vrši provjera prilikom poziva funkcije, pa poziv može rezultovati u besmislenim rezultatima ili „pucanju" programa.

U slučaju da nije naveden prototip, kompajler će pretpostaviti deklaraciju. U liniji prvog poziva ćemo dobiti upozorenje **implicit declaration** o tome. Drugo upozorenje će biti tipa **conflicting types** i javiće se na mjestu definicije funkcije.

Poziv po referenci

- Pri radu sa nizovima, ne možemo proslijediti funkciji svaki član niza pojedinačno.
- Stoga je programerska praksa razvila **poziv po referenci** (eng. call by reference), gdje se proceduri prosljeđuje čitav memorijski objekat (npr. niz), a ne njegova kopija.
- Promjene koje se izvrše nad referencom (memorijskim objektom) u pozvanoj proceduri reflektuju se na originalni objekat u pozivajućoj proceduri.
- Programski jezik C ne podržava poziv po referenci, već samo **poziv po vrijednosti** (eng. call by value), gdje se u parametre funkcije upisuje kopija vrijednosti argumenata.
- U C-u se poziv po referenci simulira koristeći pokazivače.

Prosljeđivanje preko pokazivača

```
int zbir(int *x, int *y) {
        (*x)++;
        (*y)++;
        return (*x) + (*y);
}
```

promjenljive **x** i **y** su pokazivači

x i **y** i dalje ostaju iste adrese, ali se sad mijenja (uvećava za jedan) ono što se nalazi na tim adresama. To ima efekat i na glavni program. ***x*** je ono što se nalazi na adresi x.

Poziv funkcije zbir se obavlja:

```
int x=1, y=2;
c = zbir(&x, &y);
```

argument je adresa promjenljive

Primjer: Suma niza

- Prethodni primjer sa prosljeđivanjem promjenljivih preko pokazivača (adresa) nije tipičan.
- Dajmo sad dva mnogo karakterističnija primjera. Prvi je sumiranje članova niza:

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

Funkciji prosljeđujemo adresu niza (tj. adresu njegovog prvog člana), kao i broj članova niza (kod nizova gotovo uvijek postoji jedan ovakav par argumenata funkcije).

alternativna deklaracija kod nizova
int sumaniza(int a[], int n)

članovima niza se pristupa na uobičajen način