VIII – Osnovne strukture podataka

- U C jeziku postoji mogućnost definisanja **strukturnih tipova** koji omogućavaju rad sa nizovima, slogovima, skupovima i datotekama.
- Sve strukture podataka mogu se podeliti u dve velike grupe:
  1. **Linearne** – predstavlja skup podataka gde svaki podatak ima tačno dva susedna elementa osim krajnjih elemenata sa krajeva strukture.
  2. **Nelinearne** – to je skup podataka u kome imamo podatke koji imaju više od dva susedna elementa
- U linearne strukture spadaju: **polja**, **magacini**, **redovi** i **lančane liste**
- U nelinearne strukture spadaju **stabla** i **grafovi**
- **Polje** (*array*) predstavlja najednostavniju linearnu strukturu podataka kod koga svi elementi poseduju isti tip.
- Svi podaci u jednom polju dele zajedničko ime a svaki podatak u polju jednoznačno je određen imenom polja i svojim indeksom.
- U programskim jezicima polja mogu da budu predstavljena sa **jednodimenzionalnom** ili **višedimenzionalnom** strukturom podataka
- Jednodimenzionalna polja obično se nazivaju **nizovima** ili **vektorima**
- Višedimenzionalna polja nazivamo **matricama**.

VIII - Jednodimenzionalna polja

- Rad s nizovima je "**prirodni**" način korišćenja računara, jer memorija računara nije ništa drugo nego sekvencijalni niz bajtova.
- U programiranju, kao i u matematici, zanimaju nas nizovi kao skup istih elemenata koji su poređani jedan za drugim.
- Elementi niza su promenljive koje se označavaju indeksom:

  $\mathbf{a_i}$ označava i-ti element niza u matematici

  a[i] označava i-ti element niza u C jeziku i=0,1,2....
- Niz je imenovana i numerčka kolekcija istih objekata koji se nazivaju elementi niza.
- Elementi niza mogu biti prosti skalarni ili korisnički definisani tipovi
- Označavaju se **imenom niza** i celobrojnim izrazom **- indeksom** koji označava poziciju elementa u nizu.
- Indeks se zapisuje u srednjim (uglastim) zagradama iza imena niza.

  Primer: **x[3]** označava element niza x indeksa 3.
- Sintaksa zapisa elementa jednodimenzionalnog niza je:

  element_niza: ime_niza [indeks]

  gde je indeks izraz celobrojnog tipa.

VIII – Jednodimenzionalna polja

- Kod C-a prvi element niza ima uvek indeks **0**, a n-ti ima indeks **n-1**
- Treba naglasiti da označavanje početnog indeksa nulom nije tipično za sve programske jezike.
- Prema tome, **x[3]** označava **četvrti** element niza.
- Sa elementima niza se manipuliše kao i sa običnim skalarnim promenljivama, uz uslov da je prethodno deklarisan tip elemenata niza.
- Da bi neki podatak bio deklarisan kao jednodimenzionalno polje - **niz**, treba navesti njegovo ime, broj elemenata kao i tip elemenata
- Opšta forma izraza kojom se deklariše niz je sledeća:

<tip elemenata> <ime> [<celobrojni izraz>];

gde je **ime** neki identifikator koji definiše naziv niza, **izraz** predstavlja broj elementa niza a **tip** određuje tip svakog pojedinog elementa u nizu

- Vrednost ovog **izraz-**a mora da bude pozitivan ceo broj i predstavlja veličinu niza a indeksiranje elemenata niza uvek počinje od 0.

Primer: **int Vektor_a [100] ;**

jednodimenzionalno polje **Vektor_a** čine **100** elemenata tipa **int**.

VIII – Jednodimenzionalna polja

- Elementi jednodimenzionalnog polja (vektora) se uvek smeštaju u memoriju računara neposredno jedan za drugim, pri čemu početni elemenat vektora ima najmanju a poslednji najveću adresu.
- Deklaracijom niza rezerviše se potrebna memorija, na način da elementi niza zauzimaju sukcesivne lokacije u memoriji.
- Važi pravilo: **adresa(A)=adresa(A[0])**
  adresa(A[n])=adresa(A[0]) + n*sizeof(A[0]))
- Elementima niza se pristupa pomoću celobrojnog indeksa,

Primeri:

```
A[0] = 7;
int i=5;
A[2]= A[i];
for(i=0; i<9; i++)
printf("%d ", A[i];
```

- Napomena: **int** zauzima 4 bajta

Memorijski raspored niza A

Adresa	*Sadržaj*
1000	data[0]
1004	data[1]
1008	data[2]
1012	data[3]
1016	data[4]
1020	data[6]
1024	data[5]
1028	data[7]
1032	data[8]

VIII-Jednodimenzionalna polja-Niz

- U C jeziku se ne vrši provera da li je vrednost indeksnog izraza unutar deklarisanog intervala.

Primer:

```
int A[5];
A[12]=5;
....
```

- ✓ Iskaz **A[12]=5;** je sintaktički ispravan i kompajler neće javiti grešku
- ✓ Međutim posle izvršenja ove naredbe može doći do greške u izvršenju programa, ili čak do pada operativnog sistema.
- ✓ Greška je u tome što se ovom naredbom zapisuje vrednost 5 na memorijsku lokaciju za koju nije rezervisano mesto u deklaraciji niza.

Primer:

U programu na sledećem slajdu prikazano je:

1. Kako se niz koristi za prihvatanje veće količine podataka - realnih brojeva.
2. Pokazano je kako se određuje suma elemenata niza
3. Pronađena je vrednost i indeks elementa koji ima najveću vrednost.

VIII – Jednodimenzionalna polja

```
#include <stdio.h>
#define N 5
int main()
{
int i, imax;
double suma, max;
double A[N];              /* niz od N elemenata */
/* 1. omogući korisniku da unese 5 realnih brojeva */
printf(“Unesi %d realnih brojeva:\n", N);
for (i=0; i<N; i++)
scanf("%lg", &A[i]);
/* 2. izračunaj sumu elemenata niza */
suma = 0;
for (i=0; i<N; i++)
suma += A[i];
printf("Suma unetih brojeva je %f\n", suma);
/*3.odredi indeks(imax) i vrednost(max) najvećeg elementa */
imax = 0;
max = A[0];
   for(i=1; i<N; i++) {
      if(A[i] > max ) {
         max = A[i];
         imax=i;
      }
   }
printf ("%d. element je najveći (vrednost mu je %f)\n, imax+1, max);
return 0;
}
```

Izvršenje programa može izgledati ovako:

Unesi 5 realnih brojeva:

5 6.78 7.1 8 0.17

Suma unetih brojeva je 27.050000

4. element je najveći (vrednost mu je 8.000000)

VIII – Inicijalizacija nizova

- Polja se mogu inicijalizovati navođenjem vrednosti elemenata u okviru velikih zagrada. Sintaksa inicijalizacije polja je sledeća:

  mem_klasa tip ime[izraz]={v_1,v_2,...,v_n};

  gde je **v_1** vrednost koja će biti pridružena prvom elemetu polja ime[0] **v_2** vrednost pridružena drugom ime[1], itd.

  Primer: **float v[3]={1.17, 2.43, 6.11};**

  daje sledeće vrednosti: v[0]=1.17, v[1]=2.43, v[2]=6.11.
- Prilikom inicijalizacije polja dimenzija ne mora biti specicirana već će biti automatski izračunata.
- Zato možemo pisati **float v[]={1.17, 2.43, 6.11};** i prevodilac će sam kreirati polje dimenzije 3.
- Ako je vrednost inicijalizacije veća od dimenzije polja javlja se greška
- Ako je manja, onda će preostale vrednosti biti inicijalizovane nulom.
- Za globalno i statičko deklarisane nizove automatski se svi elementi postavljaju na vrednost nula.
- Kod lokalno deklarisanih nizova ne vrši se inicijalizacija početnih vrednosti elemenata niza i to mora obaviti programer.

VIII – Inicijalizacija nizova

- Za inicijal. elemenata niza na neku vrednost često se koristi **for** petlja

  Primer: **for (i = 0; i < 10; i++)**

  A[i] = 1;

  sve elemente niza A postaviće na vrednost 1.
- Niz se može inicijalizirati i sa deklaracijom sledećeg tipa:

  int A[9]= {1, 2, 23, 4, 32, 5, 7, 9, 6};
- Lista konstanti, napisana unutar velikih zagrada, redom određuje početnu vrednost elemenata niza.
- Ako se inicijaliziraju svi potrebni elementi niza, tada nije potrebno u deklaraciji navesti dimenziju niza.
- **int A[]={1, 2, 23, 4, 32, 5, 7, 9, 6};** je potpuno ekvivalentno prethodnoj inicijalizaciji niza **int A[9]**.
- Niz se može i parcijalno inicijalizirati.
- U deklaraciji int **A[10]= {1, 2, 23};** prva tri elementa imaju vrednost 1, 2 i 23, a ostale elemente prevodilac postavlja na vrednost nula.
- Kada se inicijalizira znakovni niz, tada se u listi inicijalizacije mogu navesti znakovne konstante: **char znakovi[2]= {'O', 'K'};**

VIII – Rad sa nizovima

- Broj elemenata nekog niza uvek se može odrediti pomoću iskaza:

```
int brojelemenata = sizeof(A)/sizeof(int);
```

- Operator **sizeof** primenjen na promenljivu daje broj okteta potrebnih za memorisanje te promenljive.
- Ako je promenljiva polje, onda **sizeof** daje broj okteta koji je potreban da se zapamti celo polje
- Zato se broj elemenata polja dobija deljenjem veličine polja sa brojem okteta koji zauzima jedna promenljiva tipa **float**.
- Međutim, ovakav način računanja broja elemenata polja nije moguće primeniti na **argument funkcije tipa polja**, budući da se on automatski konvertuje u pokazivač.

Primer: U programu **hex.c** korisnik unosi celi broj bez predznaka, zatim se vrši ispis broja u heksadecimalnoj notaciji.

VIII – Rad sa nizovima

```
#include <stdio.h>
int main()
{
int num, k;
unsigned broj;
char hexslova []= {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
char reverse[8] = {0}; /* zapis 8 hex znakova u inverznom poredku*/
printf(„Unesi celi broj bez predznaka: ");
scanf("%u", &broj);
if(broj <0) broj = -broj;
printf("Heksadecimalni zapis je: ");
num=0;
do {
k = broj % 16; /* iznos hex znaka na mestu najmanjeg*/
reverse[num++] = hexslova[k]; /* oznaka heksa znaka */
broj /= 16; /* odstrani ovaj znak */
} while(broj != 0);
/* num sadrži broj hex znakova */
/* ispis od krajnjeg (n-1) do nultog znaka */
for(k=num-1; k>=0; k--)
printf("%c", reverse[k]);
printf("\n");
return 0;
}
```

Izvršavanje ovog programa daje:

Unesi celi broj bez predznaka: 256001

Heksadecimalni zapis je: 3E801

VIII – Prenos nizova u funkciju

- Nizovi mogu biti **argumeti u funkcijama**.
- Pri deklaraciji ili definisanju funkcije formalni argument, koji ima tip niza, označava se na način da se deklariše niz bez oznake veličine niza, tj. u obliku **tip ime_niza[]**
- Pri pozivu funkcije, kao stvarni argument, navodi se samo ime niza bez srednjih zagrada.
- U C jeziku nizovi se u funkciju prenose kao **memorijske reference** (***by reference***), odnosno prenosi se adresa početnog elementa niza.
- Brigu o tome vodi kompajler u toku prevođenja programa.
- Memorijska referenca (adresa) promenljive se pamti "u njenom imenu" zato se pri pozivu funkcije navodi samo ime, bez srednjih zagrada.
- Nizovi, koji su argumenti funkcije, ne smeju se unutar funkcije tretirati kao lokalne promenljive.
- Promenom vrednosti elementa niza unutar funkcije ujedno se menja vrednost elementa niza koji je u pozivnom programu označen kao stvarni argument funkcije.

VIII – Prenos nizova u funkciju

- **Primer:** program izračunava sumu unetih elemenata niza, pomoću funkcije produkt(), i ispisuje rezultat.

```
#include <stdio.h>
#define N 5 /* radi s nizom od N elemenata */
double produkt(double A[], int brojelemenata)
{
int i;
double prod = 1;
for (i=0; i<brojelemenata; i++)
  prod += A[i];
return prod;
}
int main()
{
int i;
double A [N];
printf(“Unesi %d realnih brojeva:\n", N);
for (i=0; i<N; i++)
scanf("%lg", &A[i]);
printf("Suma unetih brojeva je %g\n", produkt(A, N));
return 0;
}
```

Uočite de se vrednost elemenata niza može menjati unutar funkcije jer se niz ne prenosi po vrednosti (*by value*), jer tada to ne bi bilo moguće.

VIII – Rad sa nizovima

Primer: Često je potrebno odrediti da li u nekom nizu od N elemenata postoji element vrednosti x. U tu svrhu zgodno je definisati funkciju **int search(int A[], int N, int x);** koja vraća indeks elementa niza A koji ima vrednost x. Ako ni jedan element nema vrednost x, funkcija vraća negativnu vrednost -1.

```
int search (int A[], int N, int x)
{
int indx;
for(indx = 0; indx < N; indx++) {
if( A[indx] == x) /* element pronađen – prekini */
break;
}
if(indx == N) /* tada ni jedan element nema vrednost x*/
return -1;
else
return indx;
}
```

Veza između nizova i pokazivača

- Ime niza je konstantni pokazivač na početak niza, tj. adresa njegovog nultog tj. početnog elementa.
- Ime niza je samo po sebi jedna adresa, ili vrednost pointera (pointer je promenljiva koja uzima adrese za svoje vrednosti).
- Kada je niz deklarisan, kompajler alocira baznu adresu i dovoljan memorijski prostor za smeštanje svih elemenata niza.
- Kompajler jezika C sam prevodi oznake niza u pokazivače, pa se korišćenjem pokazivača povećava efikasnost u radu sa nizovima.

  Primer: Neka je data deklaracija

```
#define N 100
long a[N], *p;
```

- Pretpostavimo da je prva adresa za smeštanje niza **a** jednaka 300 (ostale su 304,308,…,696).
- Naredbe **p = a; p=&a[0];** su potpuno ekvivalentne.
- U oba slučaja promenljiva **p** uzima za svoju vrednost adresu nultog elementa niza **a**.
- Preciznije, promenljivoj **p** se dodeljuje vrednost 300.

Veza između nizova i pokazivača

- Takođe, naredbe **p=a+1; p=&a[1];** su ekvivalentne i dodeljuju vrednost 304 promenljivoj **p**.
- Analogno, naredbe **p = a + i** i **p = &a[i]** su ekvivalentne, za svaki element **a[i]** niza **a**.
- Ako su elementima niza **a** pridružene vrednosti, one se mogu sumirati koristeći pointer **p**, na sledeći način:

```
sum=0;
for(p=a; p<&a[N]; sum += *p, ++p);
```

- Isti efekat se može postići sledećim postupkom:

```
sum=0;
for(i=0;i<N;++i) sum += *(a+i);
```

- Napomenimo da je izraz ***(a+i)** ekvivalentan sa **a[i]**, a takođe može se pisati **p = &(*(a+i))** umesto izraza **p = a + i**, odnosno **p = &a[i].**

Primer: Napisati program za sumiranje elemenata nizova.

```
p=a; sum=0;
for(i=0;i<N;++i) sum += p[i];
```

Veza između nizova i pokazivača

- Međutim, postoji razlika između pointera i nizova.
- Kako je ime niza **a** konstantni pointer, a ne promenljiva, ne mogu se koristiti izrazi **a = p++a a+=2**.
- To znači da se adresa niza ne može menjati.

Primer: Napisati program koji transformiše niz celih brojeva tako da na početku budu negativni a na kraju nenegativni elementi tog niza.

```
#include<stdio.h>
void main(void)
{ int niz[100], p,i,k,n;
printf("\nBroj elemenata --> "); scanf("%d",&n);
for(i=0; i<n; i++)
{ printf("niz[%d]=--> ",i+1); scanf("%d",niz+i); }
p=0; k=n-1;
while(p<k)
{ while(niz[p]<0 && p<k)p++;
while(niz[k]>=0 && p<k)k--;
if(p<k)
{ int pom=niz[p]; niz[p]=niz[k]; niz[k]=pom; }
}
for(i=0; i<n; i++)printf("niz[%d] = %d ",i,niz[i]); printf("\n");
}
```

Veza između nizova i pokazivača

- Isti problem rešen tako što se koristi ime niza kao pokazivač.

```
#include<stdio.h>
void main(void)
{ int niz[100], *p,*k,i,n;
printf("\nBroj elemenata --> "); scanf("%d",&n);
for(i=0; i<n; i++){printf("niz[%d]= --> ",i+1);
   scanf("%d",niz+i); }
p=niz; k=niz+n-1;
while(p<k)
{ while(*p<0 && p<k)p++;
while(*k>=0 && p<k)k--;
if(p<k) { int pom; pom=*p; *p=*k; *k=pom; }
}
for(i=0; i<n; i++)printf("niz[%d]= %d ",i,*(niz+i));
   printf("\n");
}
```

Veza između nizova i pokazivača

- **Pointerska aritmetika** predstavlja veliku prednost jezika C u odnosu na druge jezike visokog nivoa.
- U jeziku C dozvoljene su aritmetičke operacije nad pokazivačima.

Dozvoljeno je:

- ✓ dodeliti pokazivaču adresu promenljive ili nulu (vrednost NULL);
- ✓ uvećati ili umanjiti vrednost pokazivača;
- ✓ dodati vrednosti pokazivača neki ceo broj;
- ✓ oduzeti od vrednosti pokazivača neki ceo broj;
- ✓ porediti dva pokazivača pomoću operacija **==, !=,** i tako dalje;
- ✓ oduzeti od jednog pokazivača drugi, ako ukazuju na objekte istog tipa.

VIII - Stringovi

- String je naziv za memorijski objekat koji sadrži niz znakova, a poslednji znak u nizu mora biti nulti znak ('\0').
- Deklariše se kao niz znakova (pr. **char str[10]**), ili kao pokazivač na znak (**char *str**), ali pod uslovom da se pri inicijalizaciji niza i kasnije u radu sa nizom uvek vodi računa o tome da poslednji elemenat niza mora biti jednak nuli.
- Zbog ove se karakteristike stringovi u C jeziku nazivaju se **ASCIIZ** stringovi jer se sastoje od niza ASCII znakova i nule (**Z - zero**).
- Dužina stringa je celobrojna vrednost koja je jednaka broju znakova u stringu (bez nultog znaka).
- Indeks "nultog" znaka jednak je broju znakova u stringu, odnosno dužini stringa.

  Primer: string koji sadrži tekst: **Hello, World!**, u memoriji zauzima 14 bajta. Njegova dužina je 13, jer je indeks nultog znaka jednak **13**.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
H	e	l	l	o	,		W	o	r	l	d	!	\0

VIII - Stringovi

- String se može inicijalizirati i pomoću literalne konstante:

  char hello[]="Hello, World!“;

- Elementi stringa se mogu menjati naredbom dodele vrednosti,
- Tako, naredbama **hello[0] = 'h'; i hello[6] = 'w';** možemo promeniti prethodni string u "**hello world**";
- Ako je potrebno više mesta za string, nego što se to navodi inicijalnim literalnim stringom, tada treba eksplicitno navesti dimenziju stringa. **Primer**: deklaracijom **char hello[50] = "Hello, World!";** kompajler rezervše 50 mesta u memoriji a u prvih 14 upisuje "**Hello, World!“**
- String je i svaki pokazivač koji se inicijalizira na adresu memorijskog objekta koji ima karakteristike stringa.
- Zato i sledeći iskaz predstavlja deklaraciju stringa:

  char *digits ="0123456789ABCDEF";

- Ovakva inicijalizacija je moguća jer kompajler interno literalni string tretira kao referencu, pa se njegova adresa dodeljuje pokazivaču **digits**.
- Dozvoljeno je čak literalnu string konstantu koristiti kao referencu: **printf("%c", "0123456789ABCDEF"[n]);** ispisuje n-ti znak stringa

VIII – Stringovi

Primer: Pogledajmo primer u kome se string tretira kao niz znakova.

```
#include <stdio.h>
#include <string.h>
int main()
{
char hello[14] = { 'H', 'e', 'l', 'l', 'o', ',', ' ‘,'W‘,'o‘,'r‘,'l‘,'d‘,'!‘,'\0’};
printf("%s\n", hello);
printf("Dužina stringa je %d.\n", strlen(hello));
return 0;
}
```

Nakon izvršenja programa, dobija se poruka:

Hello, World!
Dužina stringa je 13.

- U programu je prvo definisana i inicijalizirana promenljiva **hello**.
- Ona je tipa **znakovnog niza (string)** od 14 elemenata.
- Inicijalizacijom se u prvih 13 elemenata upisuju znakovi (**Hello World!**), a poslednji element se inicira na nultu vrednost.
- Ispis ove promenljive se vrši pomoću **printf()** funkcije sa specifikatorom formata ispisa **%s**.
- Dužina stringa je određena korišćenjem standardne f-je strlen(char *s);

VIII – Operacije sa stringovima

- C nema dobru podršku za rad sa stringovima;
- Praktično stringovi ne postoje, već se predstavljaju kao polje karaktera

Primer:

```
char ime[50];
char prezime[50];
char punoime[100];
ime = "Arnold";
prezime = " Schwarznegger";
punoime = "Mr " + ime + prezime;
```

- Ako su **s1** i **s2** C "stringovi" program **ne može**:
  1. da dodeli vrednost jednog stringa drugom → **s1 = s2;**
  2. da ih upoređuje → **. . . s1 < s2 . .**
  3. da uradi konkatenaciju u jedan string → **. . . s1 + s2 . . .**
  4. da funkcija kao rezultat vrati string.
- Da bi smo uradili ove operacije moramo se koristiti funkcijama iz biblioteke **#include <string.h>**

VIII – Operacije sa stringovima

- Stringovi su podržani u svim modernim programskim jezicima.
- Tipične operacije nad stringovima su:
  1. određivanje dužine stringa (length);
  2. upoređenje na jednakost (equality comparison);
  3. leksikografsko poređenje (lexicographic comparison);
  4. selekcija karaktera u stringu (character selection);
  5. selekcija podstringa (substring selection);
  6. nadovezivanje (concatenation);
  7. konverzija stringa u numeričku vrednost i obratno.
  8. traženje određenog karaktera ili dela stringa
  9. kopiranje stringova ili neke vrednosti u string
- Da bi se sve ove operacije primenile nad stringovima potrebno je korišćenje funkcija iz standardne biblioteke **#include <string.h>**

VIII – Operacije sa stringovima

1. **Konkatenacija** – nadovezivanje vrednosti stringova
   Ako označimo operaciju konkatenacije sa ||, rezultat primene operacije konkatenacije na stringove:
   "Moja najbolji drug je " || " moj prvi komsija Pera“ je:
   "Moja najbolji drug je moj prvi komsija Pera“
   char *strcat(const char *string1, char *string2)
   Dodaje string2 iza string1
   char *strncat(const char *string1, char *string2, size_t n)
   Dodaje n karaktera iz string2 u string1

2. **Kopiranje** (dodela vrednosti stringu)
   char *strcpy(const char *string1, const char *string2)
   Kopira string2 u string1, uključujući oznaku kraja stringa.
   char *strncpy(const char *string1,const char *string2, size_t n)
   Kopira prvih n karaktera iz string2 u string1.

VIII – Operacije sa stringovima

3. **Traženje** u stringu - operacija traženja omogućava da se u zadatom stringu pronađete i/ili izdvoje delovi tj. drugi stringovi ili karakteri

// Nalazi prvo pojavljivanje karaktera u stringu.

char *strchr(const char *string, int c)

// Pronalazi poslednje pojavljivanje karaktera **c** u stringu.

char *strrchr(const char *string, int c)

// Locira prvo pojavljivanje stringa **s2** u stringu **s1**.

char *strstr(const char *s1, const char *s2)

// Vraća pointer na prvo pojavljivanje u stringu **s1** nekog karaktera iz
// stringa **s2**, ili null pointer ako nema karaktera iz **s2** u **s1**

char *strpbrk(const char *s1, const char *s2)

// Vraća broj karaktera na početku **s1** koji se poklapaju sa **s2**.

size_t strspn(const char *s1, const char *s2)

// Vraća broj karaktera na početku **s1** koji se **ne** poklapaju sa **s2**

size_t strcspn(const char *s1, const char *s2)

// Deli string **s1** u sekvencu tokena, svaki od njih je ograničen jednim ili više karaktera iz stringa **s2**. **char *strtok(char *s1, const char *s2)**

VIII – Operacije sa stringovima

4. **Poređenje** vrednosti stringova - vrednosti stringova se upoređuju leksikografski, po engleskoj abecedi.

Primer: **String "A" je pre stringa “B“**

String “Pera" je posle stringa “Mika“

(poređenje se vrši redom, slovo po slovo)

String “Perci" je posle stringa “Peric“

(slovo **c** je pre slova **i**, nadalje nije bitno)

Funkcije poređenja:

int strcmp(const char *string1,const char *string2)

Upoređuje string1 i string2 za određivanje *alphabetic* redosleda.

int strncmp(const char *string1, char *string2, size_t n)

Upoređuje (leksički) prvih n karaktera dva stringa. Vrati 0 ako su string1=string2, <0 ako string1< string2 i >0 ako string1>string2

int strcasecmp(const char *s1, const char *s2)

Case insensitive verzija za strcmp().

int strncasecmp(const char *s1, const char *s2, int n)

Case insensitive verzija za strncmp().

VIII – Operacije sa stringovima

Ostale funkcije

char *strerror(int errnum) - Poruka o grešci za zadati broj greške.

int strlen(const char *string) - Određuje dužinu stringa.

Sintaksa: len = strlen(ptr); gde je len ceo broj (int) i ptr je pointer na char

Namena: strlen() vraća dužinu niza bez oznake kraja.

Primer: Dužina stringa je 13 tj. *len* dobija vrednost13.

```
int len;
 char str[15];
 strcpy(str, "Hello, world!");
 len = strlen(str);
```

strcpy() – kopira ceo ili deo niza u drugi niz

Sintaksa: strcpy(ptr1, ptr2); gde su ptr1 i ptr2 pointeri na char

Ako imamo definisana dva niza: char S[25]; i char D[25]; postoje sledeće varijante ove naredbe:

1. strcpy(S, "This is String 1."); - Upis proizvoljnog teksa u string **S**
2. strcpy(D, S); - Kopiranje celog stringa iz **S** u **D**
3. strcpy(D, &S[8]); - Kopiranje ostatka stringa **S** u **D**

VIII – Operacije sa stringovima

strncpy() - kopiranje dela stringa

Sintaksa: strncpy(ptr1,ptr2, n); n je ceo broj a ptr1 i ptr2 pointeri na char

Varijante korišćenja: definisana su dva niza char S[25]; i char D[25]; i uz pretpostavku da se sledeći deo koda izvršava pre svake navedene stavke:

```
strcpy(S, "This is String 1.");
```

1. Kopiranje 4 karaktera od početka S u D i stavljanje null na kraju:

```
strncpy(D, S, 4);
D[4] = '\0';
```

2. Kopiranje dva karaktera iz sredine stringa S u D:

```
strncpy(D, &S[5], 2);
D[2] = '\0';
```

3. Kopiranje kraja stringa S u D:

```
strncpy(D, &S[8], 15);   //daje isti rezultat kao strcpy(D, &S[8]);
```

VIII – Operacije sa stringovima

Primer: Korišćenje ovih funkcija:

```
char *str1 = “ZDRAVO";   // deklaracija stringa - može i ovako !!
char *str2;              // posto je ime stringa ukazatelj na prvi znak
int duzina;
duzina = strlen(“ZDRAVO");          /* duzina = 5 */
(void) strcpy(str2,str1);
 char *str1 = “ZDRAVO";
 char *str2;
 int duzina = 2;
 (void) strncpy(str2,str1, duzina);          /* str2 = “ZD" */
```

- Obratiti pažnju da **str2** nije završen sa null oznakom!!!
- Funkcije strncat(), strncmp,() i strncpy() su restriktivnija verzija originalnih funkcija (bez “**n**” u imenu) – obavljaju istu funkciju, ali za **n** karaktera

VIII – Operacije sa stringovima

strcmp() - koristi se za poređenje dva stringa

Sintaksa: diff = strcmp(ptr1, ptr2); gde je diff ceo broj a ptr1 i ptr2 pointeri na char

Primeri korišćenja:

```
char s1[25] = "pat";
char s2[25] = "pet";
```

//diff će imati *negativnu* vrednost nakon izvršenja sledećeg koda:

```
diff = strcmp(s1, s2);
```

//diff će imati *pozitivnu* vrednost nakon izvršenja sledećeg koda:

```
diff = strcmp(s2, s1);
```

//diff će imati vrednost nula (0) nakon izvršenja sledećeg koda:

```
diff = strcmp(s1, s1);
```

- Poređenje se vrši karakter po karakter.
- Ako su stringovi identični, rezultat je nula (0).
- Kada se pronađe razlika, prestaje se sa poređenjem, i ako je taj karakter u prvom stringu "manji" tj pre (po ASCII) karaktera iz drugog stringa vraća se negativna vrednost a ako nije pozitivna vrednost

VIII – Operacije sa stringovima

strncmp() – koristi se za poređenje prvih n karaktera dva stringa

Sintaksa: diff = strncmp(ptr1, ptr2, n);

gde su diff, n celi brojevi a ptr1 i ptr2 pointeri na char

Primeri korišćenja:

```
char s1[25] = "pat";
char s2[25] = "pet";
```

//diff će imati *negativnu* vrednost nakon izvršenja sledećeg koda:

```
diff = strncmp(s1, s2, 2);
```

//diff će imati *pozitivnu* vrednost nakon izvršenja sledećeg koda:

```
diff = strncmp(s2, s1, 3);
```

//diff će imati vrednost nula (0) nakon izvršenja sledećeg koda:

```
diff = strncmp(s1, s1, 1);
```

- Poređenje se vrši karakter po karakter.
- Ako su stringovi identični, rezultat je nula (0).
- Kada se pronađe razlika, prestaje se sa poređenjem, i ako je taj karakter u prvom stringu "manji" tj. pre (po ASCII) karaktera iz drugog stringa vraća se negativna vrednost a ako nije pozitivna

VIII-Rad sa nizovima i stringovima

Primer: Sledeća dva primera pokazuju kako se mogu napisati standardne funkcije za kopiranje stringa (**strcpy**) i leksičko upoređivanje dva stringa (**strcmp**). Funkcija **strcpy()** kopira znakove stringa **src** u string **dest**, a vraća adresu od stringa **dest**.

1. **Verzija s indeksnim operatorom**:

```
char *strcpy( char *dst, const char *src)
{
int i;
for (i = 0; src[i] != '\0'; i++)
dst[i] = src[i];
dst[i] = '\0';
return dst;
}
```

2. **Verzija s pokazivačkom aritmetikom**

```
char *strcpy(char *dest, const char *src)
{
char *d = dest;
while(*d = *src) /* true dok src ne bude '\0'*/
{
d++;
src++;
}
return dest;
}
```

VIII-Rad sa nizovima i stringovima

Funkcija **strcmp()** služi poređenje dva stringa s1 i s2. Deklarisana je:

int strcmp(const char *s1, const char *s2)

F-ja vraća vrednost 0 ako je sadržaj oba stringa isti, negativnu vrednost ako je **s1** leksički manji od **s2**, ili pozitivnu vrednost ako je **s1** leksički veći od **s2**. Leksičko poređenje vrši se znak po znak, prema ASCII kodu

```
/*1. verzija sa indeksnim operatorom */
int strcmp( char *s1, char *s2)
{
int i = 0;
while(s1[i] == s2[i] && s1[i] != '\0')
i++;
if (s1[i] < s2[i])
return -1;
else if (s1[i] > s2[i])
return +1;
else
return 0;
}
```

```
/*2. verzija s inkrementiranjem pokazivača*/
int strcmp(char *s1, char *s2)
{
while(*s1 == *s2)
{
if(*s1 == '\0')
return 0;
s1++;
s2++;
}
return *s1 - *s2;
}
```

- Kod starijih CPU verzija 2 rezultirala je bržim izvršenjem programa.
- Kod novijih CPU to nije slučaj, pa se preporučuje korišćenje verzije 1.

VIII – Operacije sa stringovima

strchr() i strrchr()

Primer korišćenja ovih funkcija:

```
char *str1 = “Hello";
char *ans;
ans = strchr(str1,'l');
```

➢Nakon izvršenja, ans ukazuje na lokaciju str1 + 2

strpbrk() - generalnija funkcija, koja traži prvo pojavljivanje bilo koje grupe karaktera:

```
char *str1 = "Hello";
char *ans;
ans = strpbrk(str1,'aeiou');
```

➢Sada ans pokazuje na lokaciju str1 + 1, lokaciju prvog e.

strstr() - vraća pointer na specificirani string za traženje ili null pointer ako taj string nije nađen. Ako s2 ukazuje na string dužine 0, f-ja vraća s1:

```
char *str1 = "Hello";
char *ans;
ans = strstr(str1,'lo');      // U ovom slučaju, ans = str + 3.
```

VIII – Operacije sa stringovima

- Parametar funkcije je pokazivač na **char**, odnosno, pri pozivu funkcije argument je adresa početnog elementa stringa.
- Funkcija vraća vrednost dužine stringa (**size_t** je sinonim za unsigned).
- Funkcija **strlen()** se može implementirati na sledeći način:

```
unsigned strlen(const char *s)
{
unsigned i=0;
while (s[i] != '\0') /* prekini petlju za s[i]==0, inače */
i++; /* inkrementiraj brojač znakova */
return i; /* i sadrži dužinu stringa */
}
```

ili pomoću pokazivačke aritmetike:

```
unsigned strlen(const char *s)
{
unsigned i = 0;
while (*s++ != '\0') /* prekini petlju za *s==0, inače */
i++; /* inkrementiraj brojač i pokazivač*/
return i; /* i sadrži dužinu stringa */
}
```

Standardne F-je za rad sa stringovima

- U standardnoj biblioteci "**string.h**". postoji niz funkcija za rad:

1. **size_t strlen(const char *s)** - Vraća dužinu stringa s.
2. **char *strcpy(char *s, const char *t)** - Kopira string t u string s, uključujući '\0'; vraća s.
3. **char *strncpy(char *s, const char *t, size_t n)** - Kopira najviše n znakova stringa t u s; vraća s. Dopunjava string s sa '\0' znakovima
4. **char *strcat(char *s, const char *t)**-Dodaje string t na kraj stringa s;
5. **char *strncat(char *s, const char *t, size_t n)** - Dodaje najviše n znakova stringa t na string s, i znak '\0'; vraća s.
6. **int strcmp(const char *s, const char *t)** - Upoređuje string s sa stringom t, vraća <0 ako je s<t, 0 ako je s==t, ili >0 ako je s>t. Upoređivanje je leksikografsko, prema ASCII "abecedi".
7. **int strncmp(const char *s, const char *t, size_t n)** - Upoređuje najviše n znakova stringa s sa stringom t; vraća <0 ako je s<t, 0 ako je s==t, ili >0 ako je s>t.
8. **char *strchr(const char *s, int c)** - Vraća pokazivač na prvu pojavu znaka **c** u stringu **s**, ili NULL znak ako **c** nije sadržan u stringu **s**.

VIII-Rad sa nizovima i stringovima

9. char *strrchr(const char *s, int c) - Vraća pokazivač na zadnju pojavu znaka c u stringu s, ili NULL znak ako c nije sadržan u stringu s.
10. char *strstr(const char *s, const char *t) - Vraća pokazivač na prvu pojavu stringa t u stringu s, ili NULL ako string s ne sadrži string t.
11. size_t strspn(const char *s, const char *t) - Vraća dužinu prefiksa stringa **s** koji sadrži znakove koji čine string t.
12. size_t strcspn(const char *s, const char *t) - Vraća dužinu prefiksa stringa **s** koji sadrži znakove koji nisu prisutni u stringu t.
13. char *strpbrk(const char *s, const char *t) - Vraća pokazivač na prvu pojavu bilo kojeg znaka iz string **t** u stringu **s**, ili NULL ako nije prisutan ni jedan znak iz string t u stringu s.
14. char *strerror(int n) - Vraća pokazivač na string kojeg interno generiše kompajler za dojavu greške u nekim sistemskim operacijama. Argument je obično globalna promenljiva **errno**, čiju vrednost takođe postavlja kompajler pri sistemskim operacijama.

VIII-Rad sa nizovima i stringovima

15. char *strtok(char *s, const char *sep) - vrši razlaganje stringa **s** na niz **leksema** koji su razdvojeni znakovima-separatorima.

- ✓ Skup znakova-separatora se zadaje u stringu **sep**.
- ✓ Funkcija vraća pokazivač na leksem ili NULL ako nema leksema.
- ✓ Korišćenje funkcije **strtok()** je specifično jer u stringu može biti više leksema, a ona vraća pokazivač na jedan leksem.
- ✓ Da bi se dobili sledeći leksemi treba ponovo zvati istu funkciju, ali s prvim argumentom jednakim NULL.
- ✓ Tako na primer, za string **char * s = "Prvi drugi,treci";** ako odaberemo znakove separatore: razmak, tab i zarez, tada sledeći iskazi daju ispis tri leksema (Prvi drugi i treči):

```
char *leksem = strtok(s, " ,\t");        /* nađi prvi leksem */
while( leksem != NULL) {                 /* ukoliko postoji */
printf("", leksem);                      /* ispiši ga i nađi sledeći */
lexem = strtok(NULL, " ,\t");            /* pa ponovi postupak */
}
```

Ulaz./izlaz.operacije sa stringovima

- Najjednostavniji način nalaženja i ispisivanja stringa je da se koriste **printf()** i **scanf()** funkcije s oznakom formata %s.

  Primer:

```
char line[100];
scanf("%s",str);
printf("%s", str);
```

- Funkcija **scanf()** nije pogodna za unos stringova u kojima ima blanko znakova, jer oni znače kraj unosa, pa se za unos i ispis stringa koriste:

```
char *gets(char *str);
int puts(char *str);
```

- Funkcija **gets()** prihvata liniju teksta sa standardnog ulaza (unos se prekida kada je na ulazu '\n') i smešta je u string str.
- F-ja vraća pokazivač na taj string ili NULL ako nastupi greška ili EOF.
- Funkcija **puts()** ispisuje string **str** na standardni izlaz.
- Vraća pozitivnu vrednost ili -1 (EOF) ako nastupi greška.
- Jedina razlika između funkcije printf("%s", str) i puts(str) je u tome da funkcija puts(str) uvek na kraju ispisa dodaje i znak nove linije.

Ulaz./izlaz.operacije sa stringovima

Primer: Prikazana je upotreba raznih funkcija za rad sa stringovima. Program prihvata liniju po liniju teksta sve dok se ne otkuca: "kraj".

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAXCH 2000
int main( void)
{
char str[100]; /*unosi se jedna linija teksta*/
char text[MAXCH]; /*ukupno uneseni tekst*/
/* iniciraj string text sa sledećim tekstom */
strcpy(text, "Uneli ste tekst;\n");
puts("Otkucaj nekoliko linija teksta.");
/* obavesti korisnika */
puts("Za kraj unosa otkuca: kraj");
/* kako se vrši unos */
while (gets(str) != NULL ) /* unesi string */
{
if(strcmp(str, "kraj") == 0) /* prekini */
break;
/*prekini ako dužina texta premaši MAXCH*/
if(strlen(str)+strlen(text) >= MAXCH)
break;
strcat(text, str); /*dopuni ukupan tekst+str*/
strcat(text, "\n"); /* i označi novi red */
}
puts(text); /* ispiši tekst koji je unesen */
return 0;
}
```

Korisnički definisane U/I operacije

- Veliki nedostatak funkcija **scanf()** i **gets()** je da ne mogu ograničiti broj unetih znakova, pa treba koristiti stringove kod kojih je rezervisan veliki broj bajtova u memoriji.
- Zato mnogi programeri sami definišu funkciju za unos stringa:

  int getstring(char *str, int maxchar);
- Parametri ove funkcije su string **str** i celi broj **maxchar**, kojim se zadaje maksimalno dozvoljeni broj znakova koji će biti unet u string.
- Funkcija vraća broj znakova ili 0 ako je samo pritisnut znak nove linije ili EOF ako je na početku linije uneto Ctrl-Z.
- Funkcija **getstring()** se može implementirati na sledeći način:

```
#include <stdio.h>
int getstring(char *str, int maxchar)
{
int ch, nch = 0; /* početni broj znakova=0 */
--maxchar;       /* obezbedi mesto za '\0' */
while((ch = getchar()) != EOF)
{
if(ch == '\n')   /* prekini unos na kraju linije*/
Break;
/* prihvati ako je broj znakova < maxchar */
if(nch < maxchar)
str[nchar++] = ch;
}
if(ch == EOF && nch == 0)
return EOF;
str[nch] = '\0';  /* dodaj na kraj nul znak */
return nch;
}
```

Pretvaranje stringa u numeričku vrednost

- Funkcija **getstring()** se može iskoristiti i za unos numeričkih vrednosti, jer u standardnoj biblioteci postoje funkcije

```
int atoi(char *str);

double atof(char *str);
```

  koje vrše pretvaranje znakovnog zapisa u numeričku vrednost.
- Ove funkcije su deklarisane u <stdlib.h>.
- Funkcija **atoi()** pretvara string u vrednost tipa int a funkcija **atof()** pretvara string u vrednost tipa double.
- Podrazumeva se da string sadrži niz znakova koji se koriste za zapis numeričkih literala.
- U slučaju greške ove funkcije vraćaju nulu.
- Greška se uvek javlja ako prvi znak nije znak ili znakovi + i -.

Pretvaranje stringa u numeričku vrednost

Primer: U donjem programu prikazano je kako se može vršiti unos numeričkih vrednosti pomoću funkcija **getstring(), atoi() i atof():**

```
#include <stdio.h>
#include <stdlib.h>
int getstring(char *, int);
#define NCHARS 30
int main()
{
int i;
double d;
char str [NCHARS];
puts("Unesite celi broj");
getstring(str, NCHARS);
i = atoi(str);
printf("Uneli ste %d\n", i);
puts("Unesite realni broj");
getstring(str, NCHARS);
d = atof(str);
printf("Uneli ste %lg\n", d);
return 0;
}
```

- Drugi način da se iz stringa dobije num.vrednost i obrnuto su funkcije:

int sscanf (char *str, char *format, ...); - numer.vrednost iz stringa

int sprintf(char *str, char *format, ...); - string iz numer.vrednosti

- Ove funkcije imaju isto dejstvo kao funkcija scanf() i printf().
- Razlika je u tome da sscanf()/sprintf() prima/ispisuje znakove iz string str, dok funkcija scanf/printf prima/ispisuje znakove sa standard.ulaza

Primer: **sscanf(str, "%d", &i);** - pretvaranje stringa i u num.vrednost

VIII- Nizovi stringova

- Nizovi stringova se jednostavno deklarišu i inicijaliziraju kao nizovi pokazivača na **char**.

  Primer: za rad sa kartama može se koristiti dva niza stringova: jedan za boju, a drugi za lik karata. Deklarišu se na sledeći način:

```
char *boja[] = {"Srce", "Tref", "Karo", "Pik "};
char *lik[] = {"As", "2", "3", "4", "5“,"6", "7", "8", "9", "10", "Jack",
                "Dama", "Kralj" };
```

- Pojedinom stringu se pristupa pomoću indeksa, na primer boja[2] označava string "Karo".
- Pri deklaraciji se koristi pravilo da operator indirekcije ima niži prioritet od srednjih zagrada, tj. da je deklaracija **char *boja[]** ekvivalentna deklaraciji: **char *(boja [])**, pa se išćitava: **boja** je niz pokazivača na **char**.
- Isto važi i za upotrebu promenljive **lik**.
- **Primer**: ***lik[i]** se odnosi na prvi znak i-tog stringa (tj. **lik[i][0]**).

VIII- Nizovi stringova

Primer: Program karte.c služi da se po slučajnom uzorku izmešaju karte.

- Za mešanje karata koristi se proizvoljna metoda.
- Postoje 52 karte i one se pamte u nizu celih brojeva **int karte[52].**
- Svaka **karta[i]** sadrži vrednost iz intervala 0..51 koja označava kombinaciju boje i lika karte, prema pravilu :

  oznaka boje: boja[karte[i] / 13]; /* string boja[0 do 3] */

  oznaka lika: lik [karte[i] % 13]; /* string lik[0 do 12] */

  jer ima 13 likova i 4 boje. To znači da ako se početni raspored inicijalizuje sa:

```
for (i = 0; i < BROJKARATA; i++)
    karte[i] = i;
```

  karte će biti složene po bojama, od asa prema kralju (prva će biti as-srce a poslednja kralj-pik).
- Za mešanje karata program koristi standardnu funkciju **int rand();** koja je deklarisana u **<stdlib.h>.**
- Ova funkcija pri svakom pozivu vraća celi broj iz intervala 0 do RAND_MAX (obično je to vrednost 32767) koji se generiše po slučajnom uzorku.

VIII- Program Karte.c

```
#include <stdio.h>
#include <stdlib.h>
char *boja[] = {"Srce", "Tref", "Karo",
"Pik "};
char *lik[] = {"As", "2", "3", "4", "5", "6",
"7“,"8“,"9”,"10", "Jack", "Dama”,"Kralj" };
#define BROJKARATA 52
void swap(int *x, int *y)
{
int t = *x;
*x = *y;
*y = t;
}
int main( void)
{
int i, karte[BROJKARATA]; /* ukupno 52
karte */
for (i = 0; i < BROJKARATA; i++)
karte[i] = i;
for (i = 0; i < BROJKARATA; i++)
{
int k = rand() % BROJKARATA;
swap (&karte[i], &karte[k]);
}
for (i = 0; i < BROJKARATA; i++)
printf("%s - %s\n", boja[karte[i]/13],
lik[karte[i]%13]);
return 0;
}
```

Dobije se ispis:

Pik - 3
Srce - 9
Tref - 10
Srce - 10
Tref - 5
Tref - 8
Tref - Kralj
. . . . .
Pik - 7
Srce - 3
Tref - Dama
Tref - Jack
Tref - 6
Karo - 8

VIII-Rad sa nizovima i stringovima

Primer: Korišćenje nekih navedenih funkcija za traženje

```
#include < string.h>
void main(){
    char linija[100], *deo_teksta;
    /* !!!! inicijalizacija stringa u kodu !!!!*/
    strcpy(linija,“zdravo, ja sam string;");
    printf("Linija: %s\n", linija);
    /* dodavanje na kraj stringa */
    strcat(linija," Ko si ti?");
    printf("Linija: %s\n", linija);
    /* pronađji duzinu linije -  strlen vraca duzinu kao tip size_t */
    printf(“Duzina linije: %d\n", (int)strlen(linija));
    /* pronađi pojavljivanje podnizova */
    if ( (deo_teksta = strchr ( linija, ‘K' ) )!= NULL )
          printf("String koji pocinje sa \“K\" ->%s\n", deo_teksta);
}
```

Zadatak : nalaženje najkraće reči

Zadatak:

Napisati program na C-u za nalaženje najkraće od *n* reči unetih sa tastature.

Rešenje:

Učitavaće se reč po reč sa tastature, izračunavati njihova dužina i porediti sa dužinom do tada najkreće unete reči. Ukoliko dužina tekuće reči bude manja od dužine najkraće, tekuća reč će se kopirati u najkraću reč. Na početku će se za dužinu najkraće reči uzeti vrednost veća od maksimalne moguće dužine reči.

Napomena:

U navedenom rešenju nisu korišćene funkcije iz biblioteke <string.h>. Sa strane je navedeno gde se delovi koda mogu zameniti odgovarajućim funkcijama iz ove biblioteke.

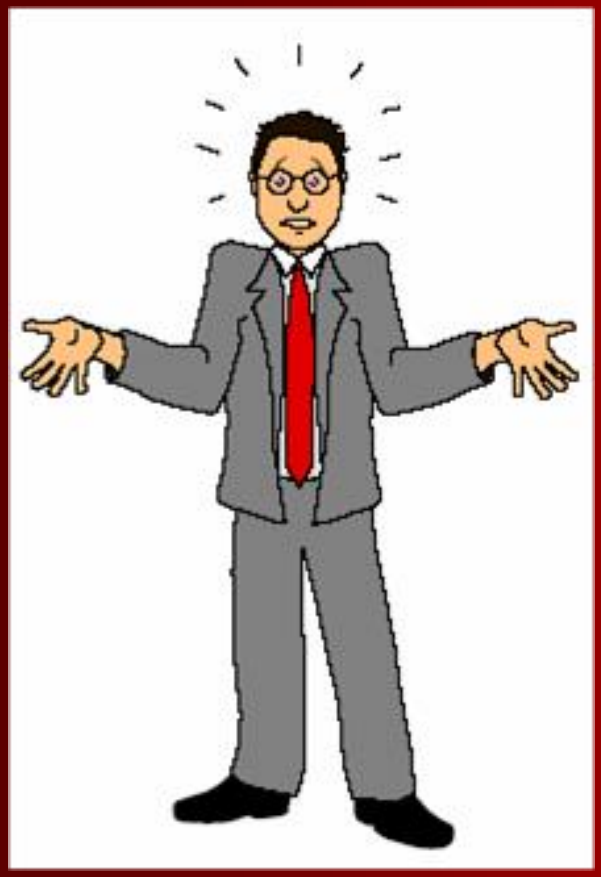
Zadatak : nalaženje najkraće reči

```
#include <stdio.h>
main()
{
   char rec[20],minrec[20];
   int i,j,n,minduzina,duzina;
   printf("unesite broj reci\n");
   scanf("%d", &n);
   /* kako je za rec predviđeno maksimalno 20 karaktera, duzina reci moze biti do 19 slova*/
   minduzina=20;
for (i=0; i<n; i++)
   {
      printf("unesite sledecu rec\n");
      scanf("%s", rec);
     // određivanje duzine reci
      for( duzina=0; rec[duzina]!='\0'; duzina++);
     // da li je uneta rec kraca od pre toga određenje najkrace
     if( duzina<minduzina )
                {          // kopiranje reci
                           j=0;
                           do
                              minrec[j]=rec[j];
                           while( rec[j++] != '\0' );
                }
              }
              printf("najkraca rec je: %s", minrec);
}
```

strlen

strcpy

Hvala na pažnji !!!



Pitanja

? ? ?