Programski jezik C

funkcije,
kazalci,
dinamična alokacija,
delo z nizi in polji

Funkcije – splošna oblika

```
[tip] ime(tip argument, tip argument2,...) {
    tip lokalnaSpremenljivka;
    tip lokalnaSpremenljivka;
    . . . . . .
    stavek;
    stavek;
     . . . . . .
    return izraz;
}
```

Vsaka funkcija ima svoje ime in ji ustreza nek podatkovni tip. Če tega ne navedemo, velja, da je tipa int. V programu mora biti ena in samo ena funkcija z imenom main.

Primer funkcije brez argumentov

```
void prikazMenuja(void) {
    printf("1....vnos podatkov\n");
    printf("2....racunanje \n");
    printf("3....Izpis rezultatov \n");
    printf("4....Izstop\n");
}

int main() {
    prikazMenuja();
}
```

Ker funkcija prikazMenuja ne vrača ničesar,tudi stavka return ni.

Posredovanje argumentov

Primer

```
void  izracun( char kaj, double x, double y) {
    switch (kaj) {
        case('+'):
            x+= y;
            break;
        case('*'):
            x*= y ;
     }
     printf("Rezultat:%lf ",x);
 }

int main() {
     double a,b;
     char  koda;
     scanf("%c %lf %lf", &koda, &a, &b);
     izracun(koda,a,b);
 }
```

*Prenos argumentov je bil izveden s "**klicem po vrednosti**".*
*V klicani funkciji je formalnim parametrom dodeljena **kopija** vrednosti argumentov, navedenih v klicu funkcije.*

Demo

Kako funkcije vračajo vrednost?

```
double fakulteta (int);

main () {
     double a;
     int k = 8;
     a = fakulteta (k);
     printf (“Fakulteta %d je %lf \n“,k,a);
}

double fakulteta (int n) {
    double x=1.0;
    int i;
    for(i=1;i<=n;i++) x = x*i;
    return x;
}
```

Predhodna deklaracija funkcije

Funkcija mora vsebovati stavek

return izraz ;

Prevajalnik predvideva, da funkcija vrača podatek tipa int, če funkcija ni pred njenim klicem definirana ali vsaj deklarirana.

Razredi pomnenja spremenljivk

auto (automatic)
Obstoj spremenljivke je omejen na obstoj funkcije oziroma bloka, v katerem je taka funkcija definirana.

register
Obstoj in področje spremenljivke je enako, kot pri avtomatičnih spremenljivkah. Prevajalnik skuša za take spremenljivke uporabiti delovne registre računalnika.

extern (external)
S to besedo označimo globalne spremenljivke,ki so definirane v neki drugi datoteki.

static
Lokalne spremenljivke, za katere želimo, da obstanejo tudi po izstopu iz njihove funkcije oziroma bloka. Eksterne spremenljivke, ki so dostopne le funkcijam za njihovo (eksterno) deklaracijo.

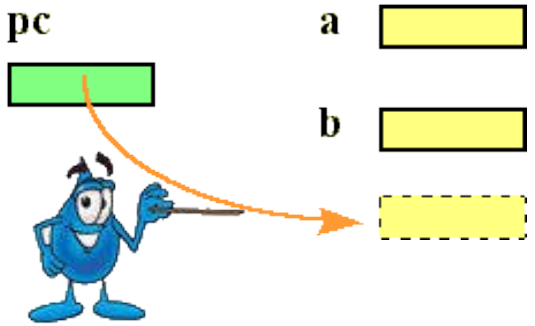
Kazalci

Deklaracija kazalca

char a,b , *pc ;

a in b sta spremenljivki tipa char, pc pa je ***kazalec*** *na spremenljivko enakega tipa (pri tem še ni jasno, na katero spremenljivko kaže).*

Kazalec je tudi spremenljivka danega tipa *(v našem primeru kazalec tipa char)*



Kazalčni operatorji

& Naslovni operator, da naslov spremenljivke

***** Operator indirekcije, da vsebino lokacije, katere naslov je v kazalcu.

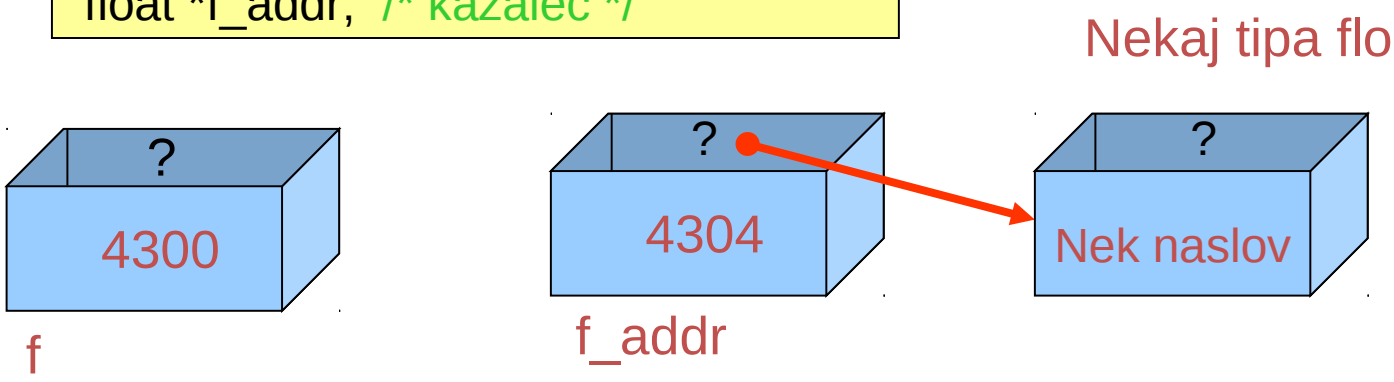
Primer uporabe:

```
pc = &a;        b = *pc;      /* isto bi naredili z b = a  */
```

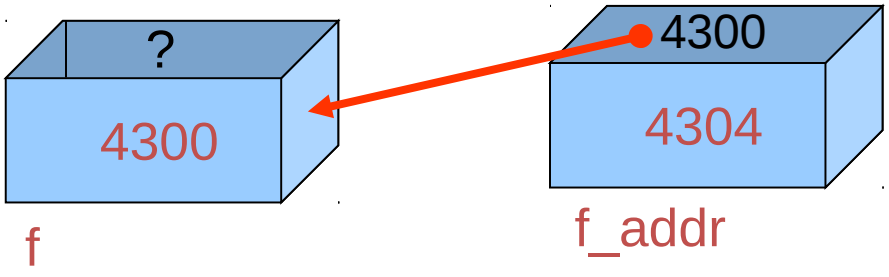
O kazalcih - 1

Kazalec ... spremenljivka, ki vsebuje naslov druge spremenljivke

```
float f;          /* spremenljivka */
float *f_addr;    /* kazalec */
```

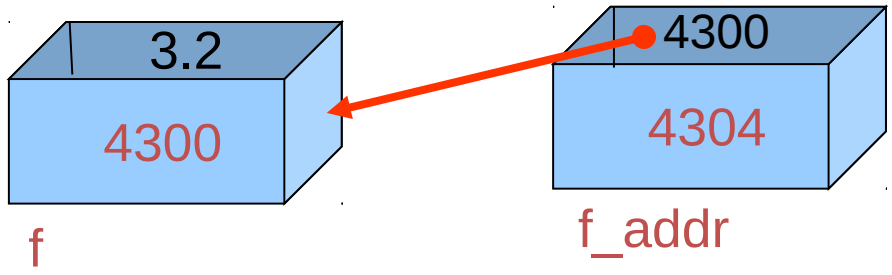



```
f_addr = &f; /* & ... operator naslavljanja*/
```

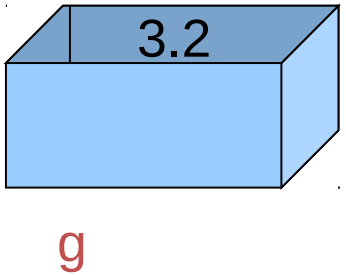


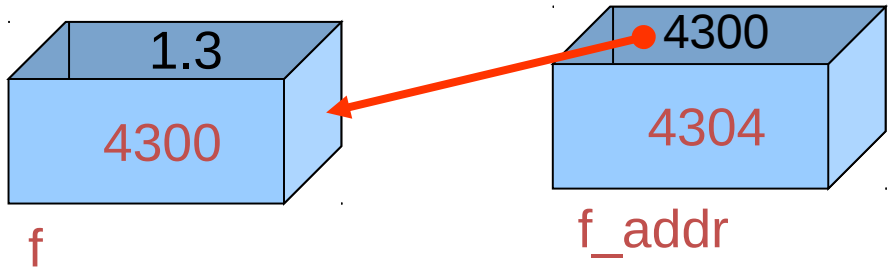
O kazalcih - 2

```
*f_addr = 3.2;    /* Operator indirekcije */
```

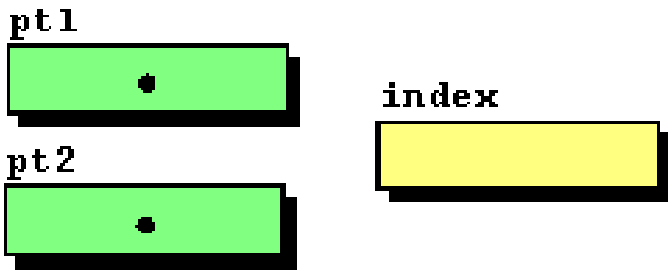



```
float g = *f_addr; /* indirekcija: g je sedaj 3.2 */
f = 1.3;
```





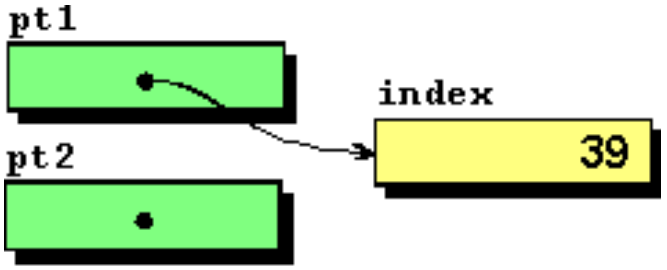
Primer s kazalci




```
#include <stdio.h>

int main() {
   int index, *pt1, *pt2;

   index = 39;                    /* any numerical value */
   pt1 = &index;                  /* the address of index */
   pt2 = pt1;
   printf("The value is %d %d %d\n", index, *pt1, *pt2);
   *pt1 = 13;                     /* this changes the value of index */
   printf("The value is %d %d %d\n", index, *pt1, *pt2);

   return 0;
}
```

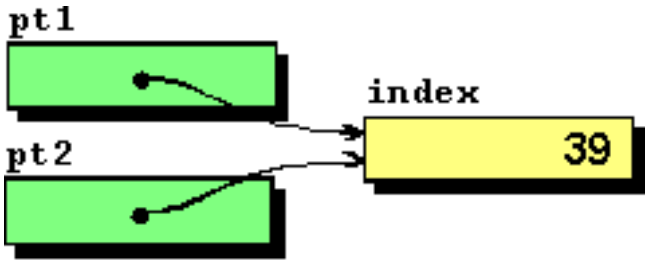
Demo

Primer s kazalci



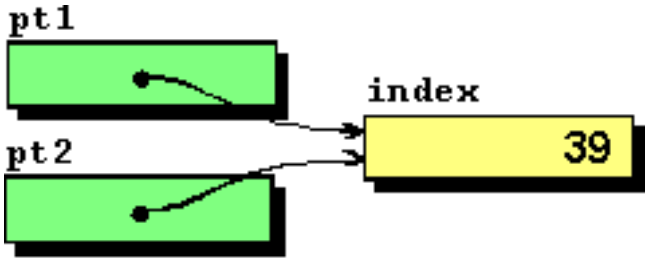

```
#include <stdio.h>

int main() {
   int index, *pt1, *pt2;

   index = 39;                          /* any numerical value */
   pt1 = &index;                        /* the address of index */
   pt2 = pt1;
   printf("The value is %d %d %d\n", index, *pt1, *pt2);
   *pt1 = 13;                  /* this changes the value of index */
   printf("The value is %d %d %d\n", index, *pt1, *pt2);

   return 0;
}
```

Primer s kazalci




```
#include <stdio.h>

int main() {
   int index, *pt1, *pt2;

   index = 39;                          /* any numerical value */
   pt1 = &index;                        /* the address of index */
   pt2 = pt1;
   printf("The value is %d %d %d\n", index, *pt1, *pt2);
   *pt1 = 13;                  /* this changes the value of index */
   printf("The value is %d %d %d\n", index, *pt1, *pt2);

   return 0;
}
```

Primer s kazalci



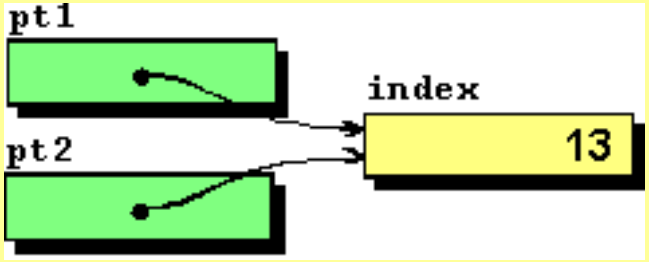

```
#include <stdio.h>

int main() {
   int index, *pt1, *pt2;

   index = 39;                    /* any numerical value */
   pt1 = &index;                  /* the address of index */
   pt2 = pt1;
   printf("The value is %d %d %d\n", index, *pt1, *pt2);
   *pt1 = 13;                     /* this changes the value of index */
   printf("The value is %d %d %d\n", index, *pt1, *pt2);

   return 0;
}
```

Demo

Izpis na zaslonu

```
The value is 39 39 39
```

Primer s kazalci




```
#include <stdio.h>

int main() {
   int index, *pt1, *pt2;

   index = 39;                    /* any numerical value */
   pt1 = &index;                  /* the address of index */
   pt2 = pt1;
   printf("The value is %d %d %d\n", index, *pt1, *pt2);
   *pt1 = 13;                     /* this changes the value of index */
   printf("The value is %d %d %d\n", index, *pt1, *pt2);

   return 0;
}
```

Izpis na zaslonu

```
The value is 39 39 39
The value is 13 13 13
```

Pogoste napake pri uporabi kazalcev

```
float a, b, *p ;
char c, *pc, niz[100] ;
. . . . . .
*p = 12.5;      /*p še ni definiran?*/
p = & 12.5;
*a = 12.5;
a = *b;
niz = "Pozdrav";
```

To pa je pravilno!

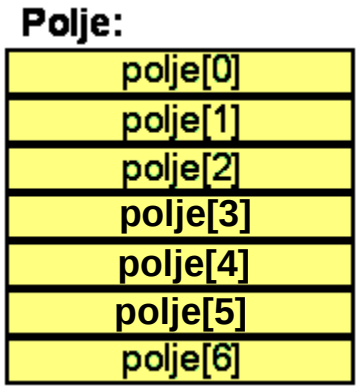
```
int *x, y[10];
char c, *pc;
 . . . . . .
x = y; /* je isto kot x = &y[0] */
pc = "Pozdrav";  /* ..OK, ker je pc spremenljivka */
```

Demo

Aritmetika s kazalci

Aritmetika s kazalci se razlikuje od aritmetike s spremenljivkami.

```
double polje[7], *p ;
int   i;
. . . . . .
p = polje
. . . . . .
```



p je enako kot **&polje[0]**

(p+4) je enako kot **&polje[4]**

p++... kazalec p bo kazal na naslednji element

polje je enako kot **&polje[0]**

***polje** je enako kot **polje [0]**

***(polje+1)** je enako kot **polje[1]**

&polje[5] - &polje[2] je enako **3**

polje[i] je enako kot ***(polje +i)**

je enako kot ***(i+polje)**

je enako kot **i[polje]**

To omogoča, da bo
p = p + 1
kazal na naslednji element tabele

O poljih in kazalcih

Ne glede na velikost posameznih elementov polja !

Zakaj kazalci kot argumenti funkcije?!

```
#include <stdio.h>

void zamenjaj(int, int);

int main() {
  int num1 = 5, num2 = 10;
  zamenjaj (num1, num2);
  printf("num1 = %d in num2 = %d\n", num1, num2);
}

void zamenjaj (int n1, int n2) { /* posredujemo vrednosti */
  int temp;

  temp = n1;
  n1 = n2;
  n2 = temp;
}
```

Napaka: posredovanje vrednosti

Demo

Zakaj kazalci kot argumenti? Zato

```
#include <stdio.h>

void zamenjaj (int *, int *);

main() {
  int num1 = 5, num2 = 10;
  zamenjaj (&num1, &num2);
  printf("num1 = %d in num2 = %d\n", num1, num2);
}

void zamenjaj (int *n1, int *n2) { /* Posredujemo naslove podatkov */
  int temp;

  temp = *n1;
  *n1 = *n2;
  *n2 = temp;
}
```

To je ekvivalent "klica po referenci", čeprav C striktno uporablja "klic po vrednosti". V tem primeru se mora zato uporabljati za argumente kazalce (prenaša se njihove vrednosti).

Demo

Kaj je tu narobe ?

```
#include <stdio.h>

void nekajNaredi(int *ptr);

main() {
        int *p;
        nekajNaredi(p);
        printf("%d", *p);     /* Bo to de
}

void nekajNaredi(int *ptr){ /* sprejme in vrne naslov */
  int temp=32+12;

  ptr = &(temp);
}

/* prevaja pravilno, med izvajanjem pa bo prislo do napake */
```

- p se prenese, čeprav ni inicializiran
- predpostavlja, da bo kazalec na temp legalen, čeprav po zaključku funkcije nekajNaredi, temp ne obstaja več in je njegov naslov nelegalen
- predpostavlja, da je naslov prenešen po referenci, in da bo dobil nazaj spremenjen kazalec p. Pravilno bi bilo torej, da se temp deklarira kot globalna spremenljivka, da se p inicializira in da se p prenese po referenci.

DEMO

Posredovanje (naslovov) 1D polj

```
/* Primer izracuna maksimuma tabelirane funkcije */
#include <stdio.h>
#define N 20 /* Namesto stevilcnih konstant raje uporabljamo simbole */
double max (double array[ ], int) ; /* vnaprejsnja deklaracija */

int main ( ) {
    double x[N], y[N], yMax;
    int i;

    for(i=0;i < N ; i++) {
        printf("Vnesi x in y:");
        scanf("%lf %lf", &x[i], &y[i]);
    }
     yMax = max(y,N);
    printf("Vrednost maksimuma:%lf \n",yMax);
}
double max(double value[ ] , int numValues) {
     int i;
     double maxValue;
     maxValue = value[0];
     for(i=0; i < numValues; i++)   if(value[i] > maxValue) maxValue = value[i];
     return  (maxValue ) ;
}
```

Opomba:
Primer ponazoruje tudi vnaprejšnjo deklaracijo funkcije . Tako deklaracijo bi lahko izvedli tudi s stavki naslednje oblike: double max(double *array, int)
ali kar
double max (double*, int)

Demo

Kazalci na večdimenzijska polja

Primer:

double A[4][5] ;

Pri enodimenzijskih poljih je ime polja naslov (kazalec na) prvi element polja.

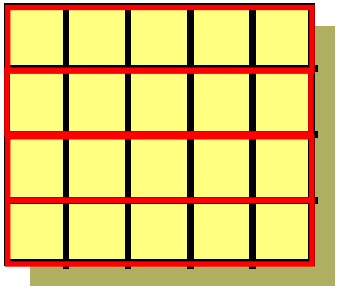
Tudi pri večdimenzijskih poljih je ime A naslov prvega elementa polja, A[0][0]. Prvi vrstici lahko rečemo A[0]. Velja:

A = A[0] = &A[0][0]

A in A[i] so konstantni kazalci (naslovi),ki jih ne moremo spreminjati.

A naslavlja prvo vrstico matrike. A+1 naslavlja naslednjo vrstico matrike, A[1]

A[4][5]



Spomnimo se: 2D polje lahko obravnavamo kot 1D polje, pri katerem je vsak element spet polje!

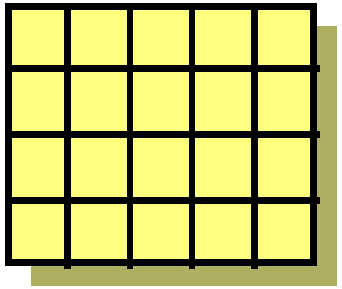
Več o kazalcih

Posredovanje naslovov 2D polj

Primer:

```
double A[4][5];

int main() {
     clearMat(A);
}

void clearMat(double  m[ ] [5]) {
    int row, col;
     for(row = 0; row <4; ++row){
         for(col = 0; col<5; ++col)  m[row][col] = 0.0;
    } }
```

A[4][5]



Pri večdimenzijskih poljih mora biti velikost dimenzij (razen skrajno leve) deklarirana tudi v klicani funkciji !!

DEMO

Dinamična alokacija pomnilnika

Kazalci torej kažejo na lokacijo v pomnilniku.
Kako **sami** alociramo del pomnilnika?

- **ptr = malloc(sizeof(struct node));**
  - **sizeof(struct node)** vrne velikost strukture v bytih
  - **malloc()** alocira navedeno število bytov v pomnilniku
  - Vrne kazalec n mesto v pomnilniku
  - Vrne **NULL**, če spomina ni več na voljo

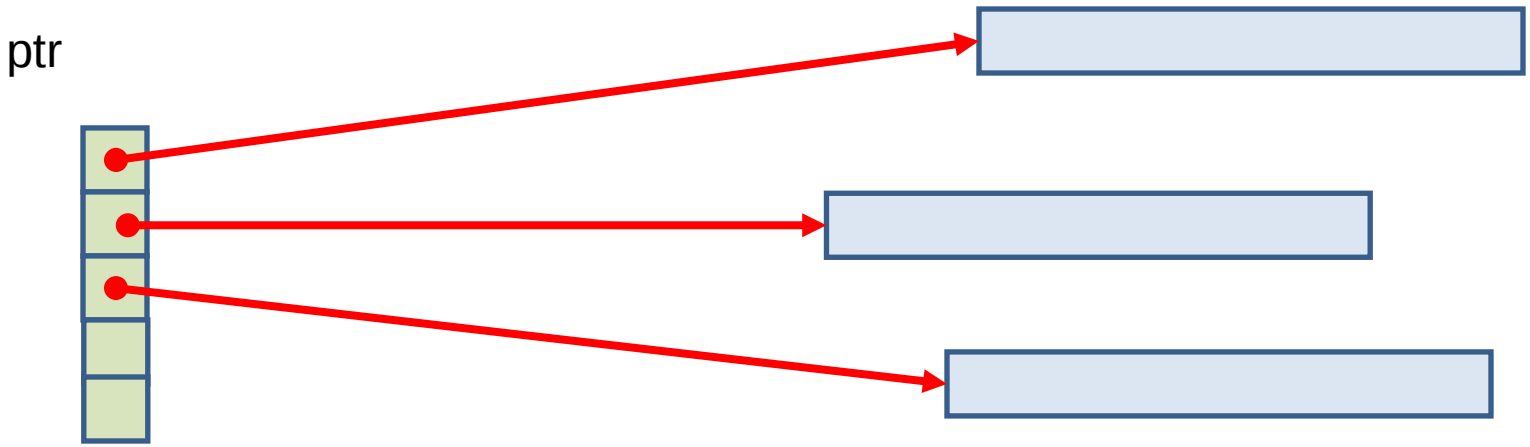
- **free(ptr);**
  - Sprosti pomnilnik, naveden s kazalcem

Demo 1

Demo 2

Preprost primer uporabe malloc

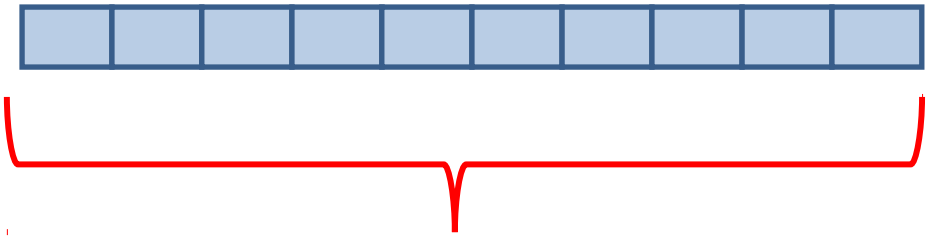
```
char *ptr[5]    /* polje 5 kazalcev na nekaj tipa char */
ptr[0] = (char*) malloc( 80);
ptr[1] = (char*) malloc( 80);
ptr[2] = (char*) malloc( 80);
……..
```



Alocirana področja v pomnilniku

Operator sizeof

Koliko bytov moramo alocirati ?



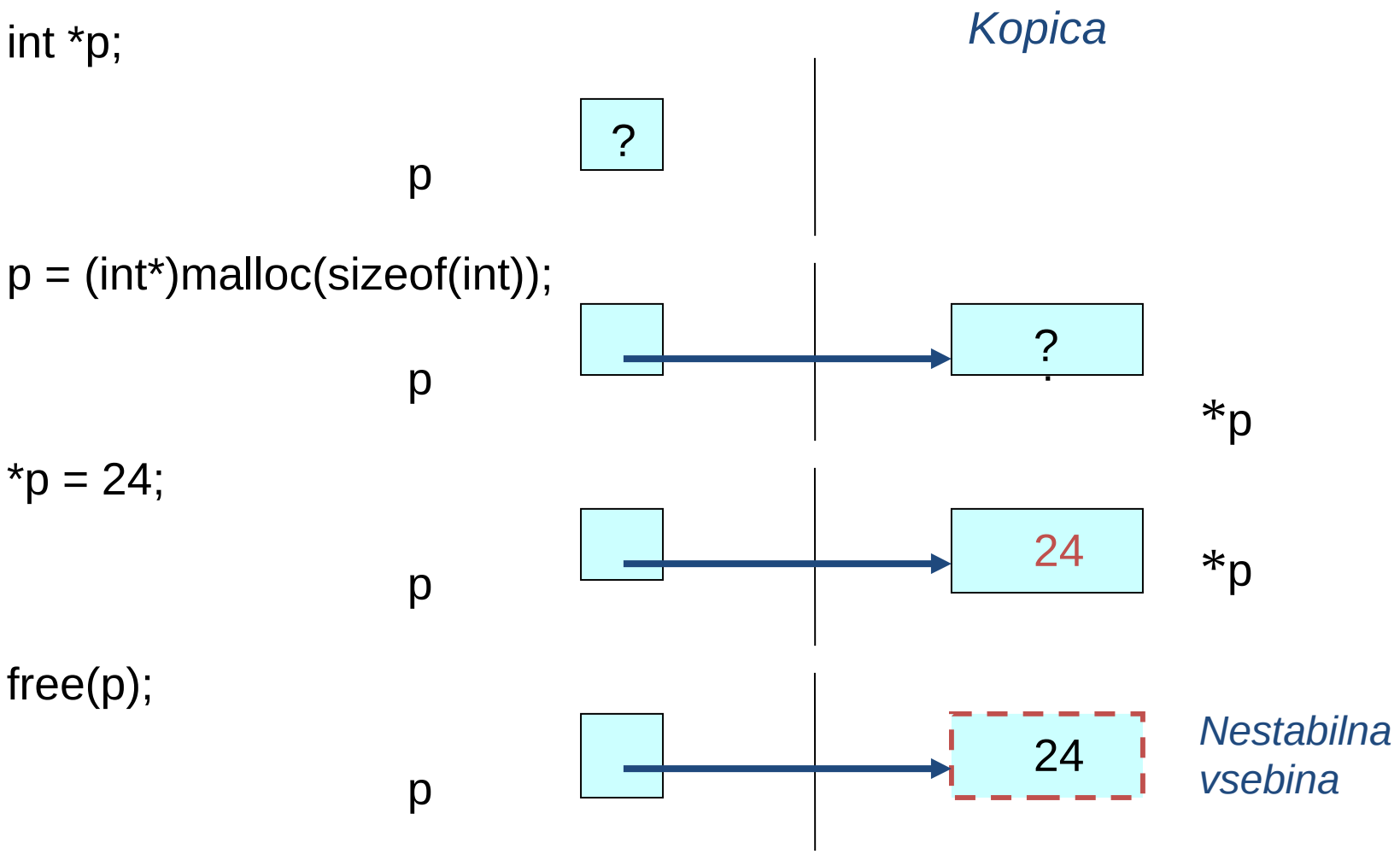
Ne trudimo se s štetjem,To bo ugotovil operator **sizeof**

- **ptr = malloc(sizeof(struct node));**
  - **sizeof(struct node)** vrne velikost strukture v bytih
  - **malloc()** alocira navedeno število bytov v pomnilniku

Demo 1

Demo 2

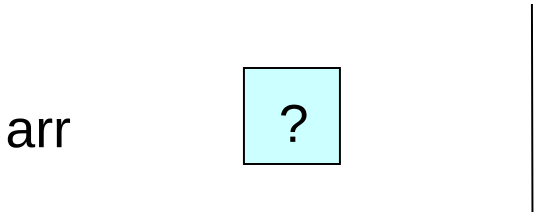
Primer dinamične alokacije



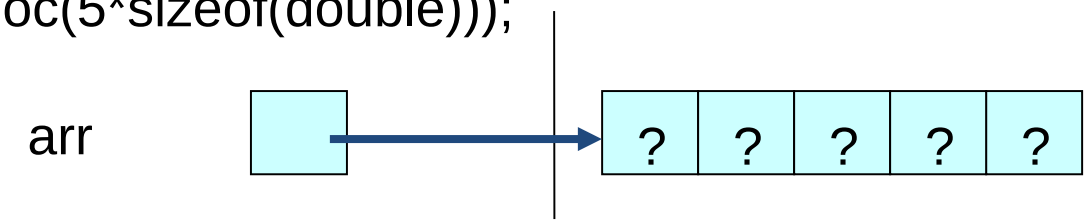
Primer 2: kazalec na polje

double * arr;

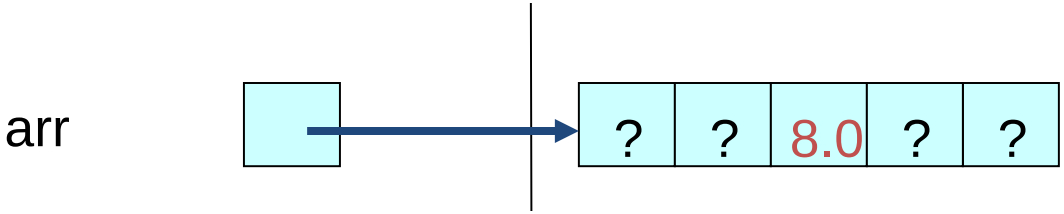
Kopica



arr = (double*)(malloc(5*sizeof(double)));



arr[2] = 8.0;



Funkcija realloc

Funkcija realloc() spremeni velikost bloka, ki smo ga predhodno alocirali s funkcijo malloc() ali calloc(). Prototip funkcije je:

void *realloc(void *ptr, size_t *size*);

Arrgument *ptr* je kazalec na originalni blok pomnilnika. *Size* pa pove novo velikost bloka v bytih.

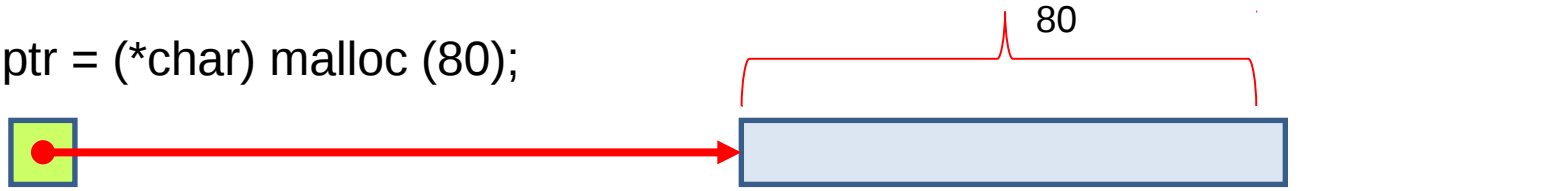
```
puts("Vpisi en niz.");
gets(buf);
message = realloc(NULL, strlen(buf)+1); /* tu bi lahko uporabili tudi malloc */
strcpy(message, buf);
puts(message);   /* Izpis besedila. */
puts("Vpisi nov niz.");
gets(buf);
/* Povecaj alokacijo in vanjo dodaj novi niz. */
message = realloc(message,(strlen(message) + strlen(buf)+1));
strcat(message, buf);
puts(message); /*Prikazi celotno obvestilo. */
```

DEMO

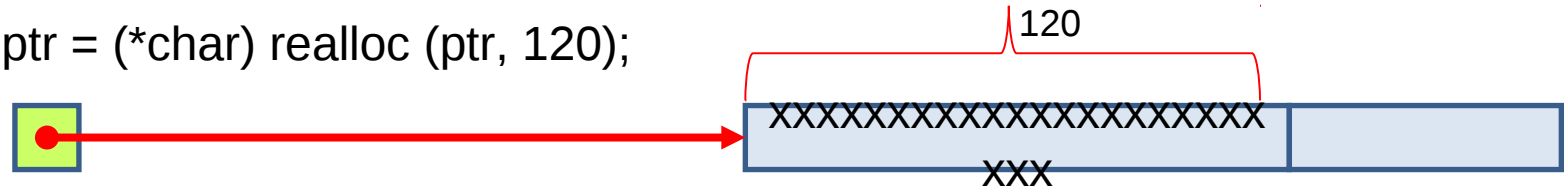
Uporaba realloc

char *ptr;

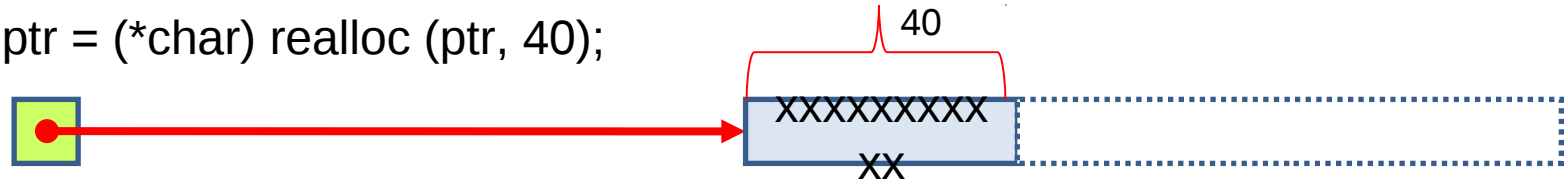
ptr = (*char) malloc (80);



ptr = (*char) realloc (ptr, 120);



ptr = (*char) realloc (ptr, 40);



Realloc tipično uporabi vsebino že alociranega pomnilnika in tega poveča (če je nova zahtevana velikost večja od prejšnje), ohrani isto vsebino (če je zahteva enaka), lahko pa tudi "odreže" (če je nova zahtevana velikost manjša od prejšnje)

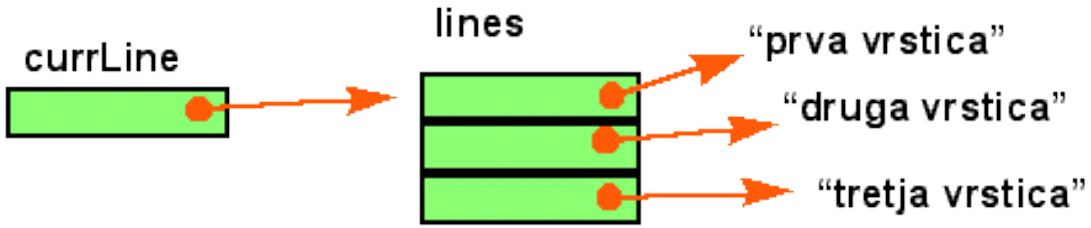
Kazalci na kazalce

Najprej si poglejmo, kako rezerviramo kar celo polje kazalcev:

Primer:

```
char *lines[3] = {"prva vrstica",
                  "druga vrstica",
                   "tretja vrstica"};
```

In sedaj še primer uporabe kazalcev na kazalce:



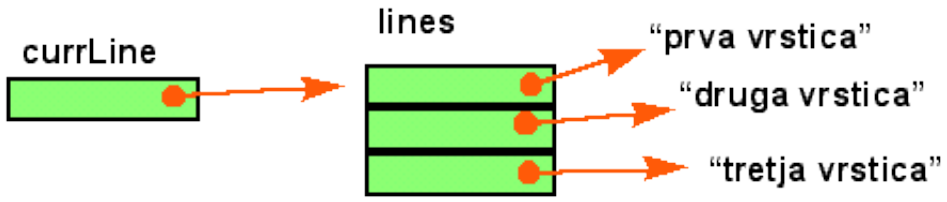
Demo

Primer uporabe kazalcev na kazalce

Kako bi take podatke deklarirali?
Kako lahko uporabimo kazalec za izpis vrstic?

```
char  **currLine, *lines[3] = {"prva vrstica",
                               "druga vrstica",
                               "tretja vrstica"};
currLine = lines;
for (i=0;i<3;i++) printf("%s \n", *currLine++);
```

Opomba:
currline je tipa *****char***
currline je tipa* ****char*
currLine je tipa* ***char



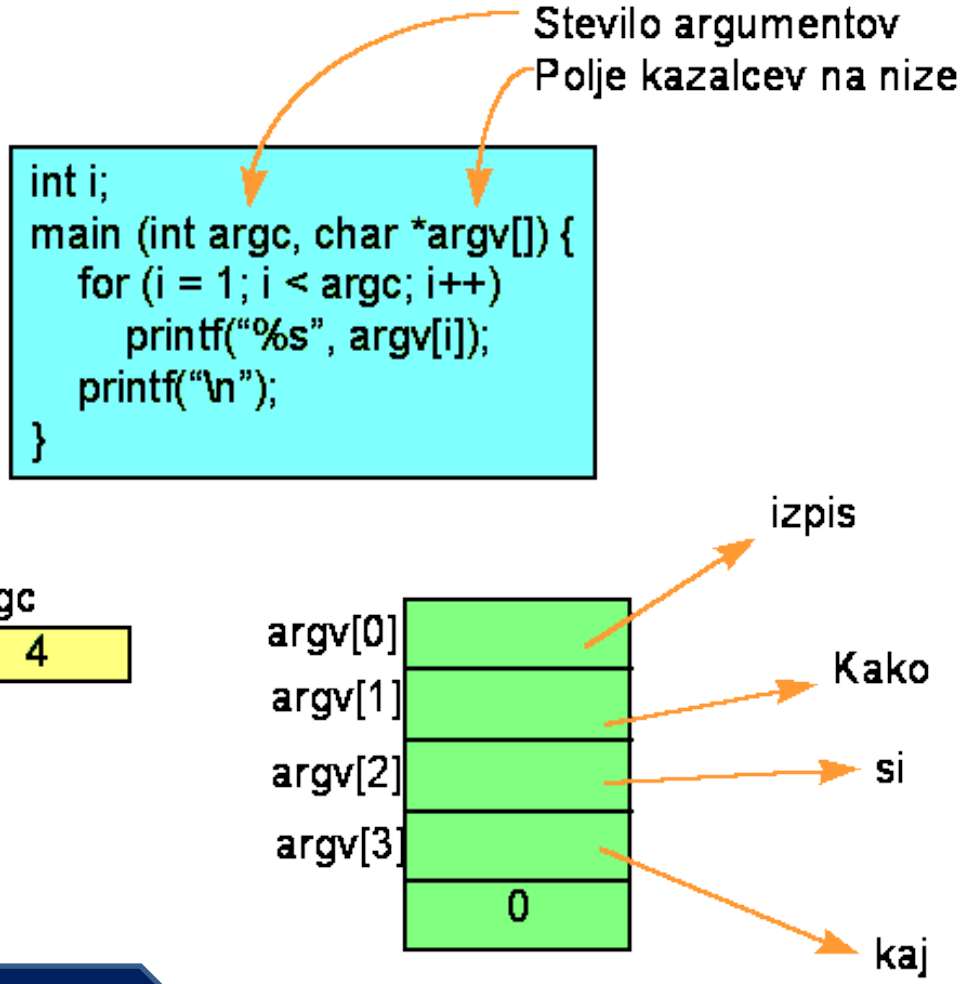
DEMO

Argumenti v ukazni vrstici

Primer ukazne vrstice:

$izpis Kako si kaj

In še ustrezni program:



Demo

Demo, kalkulator

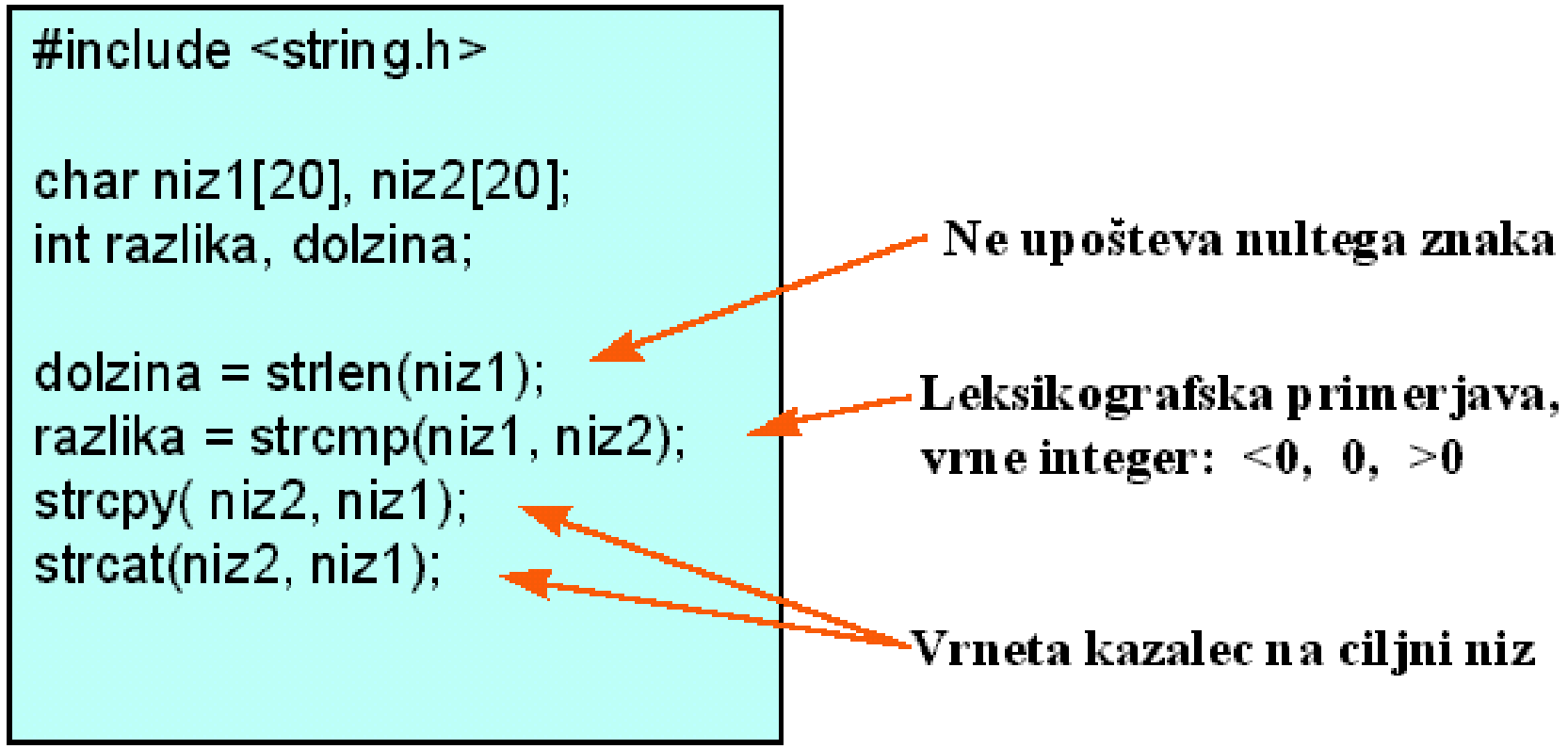
Funkcije z nizi

int strlen(s)	Vrne število znakov v nizu s (brez nultega znaka).
char *strchr(s, c)	Vrne kazalec na prvi nastop znaka c v nizu s. (sicer vrne NULL)
char *strrchr(s, c)	Vrne kazalec na zadnji nastop znaka c v nizu s.
char * strcpy (s2, s1)	Kopira niz s1 v niz s2.
char* strncpy(s2, s1, n)	Kopira niz s1 v niz s2, vendar največ n znakov.
int strcmp (s2, s1)	Primerja niza in vrne: •pozitivno vrednost če je s2>s1, •0..če je s2=s1 •negativno vrednost sicer
int strncmp(s2, s1)	Podobno kot strcmp, vendar primerja največ n znakov
char * strstr(s2, s1)	V nizu s2 išče podniz s1 in vrne kazalec nanj

WEB

Funkcije z nizi (nadaljevanje)

```
#include <string.h>

char niz1[20], niz2[20];
int razlika, dolzina;

dolzina = strlen(niz1);
razlika = strcmp(niz1, niz2);
strcpy( niz2, niz1);
strcat(niz2, niz1);
```

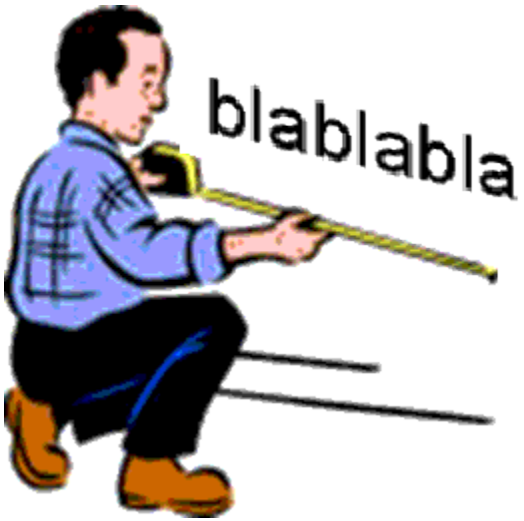


Primer 1: Primerjava nizov

```
#include <stdio.h>
#include <stdlib.h>
/*Ce uporabljamo funkcije z nizi, dodamo naslednjo vrstico..*/
#include <string.h>

int main()    {
      char odgovor[4];
      printf(“Zelis pognati ta program? [da/ne] ");
      scanf("%s", odgovor);
      if(strcmp(odgovor, "ne") == 0)
           /* 0 pomeni enakost obeh nizov */
          exit(1);
      else       /* nadaljujemo s programom... */
}
```

Demo

Primer 2: dolžina niza

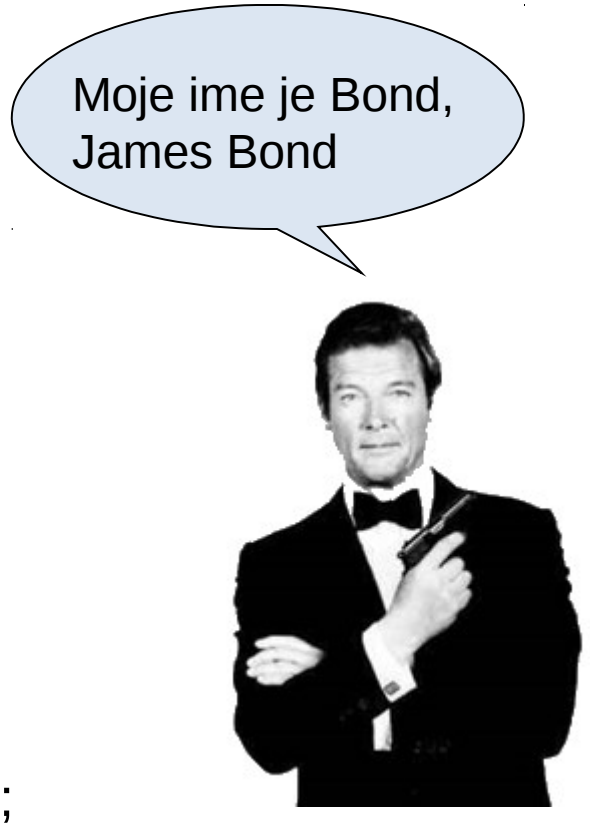



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()    {
        char str[100];
        printf("Vnesi niz");
        scanf("%s", str);
        printf("Dolzina niza (%s) je %d \n", str, strlen(str) );
        exit(0);
}
```

Demo

Primer: kopiranje in povezovanje nizov

```
/* Primer: kopiranje in povezovanje nizov */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()    {
        char ime[20], priimek[20];
        char imePriimek[42], priimekIme[42];
        strcpy(ime, "James");
        strncpy(priimek, "Bond", strlen("Bond");
        strcpy(imePriimek, ime);
        strcpy(priimekIme, priimek);
        strcat(imePriimek, priimek);
        strcat(priimekIme, ime);
        printf("Ime in priimek = %s\n", imePriimek);
        printf("Priimek in ime = %s\n", priimekIme);
        exit(0);
}
```



Demo