Programski jezik C

Strukture , njihova dinamična tvorba in uporaba

WEB

Uvod v strukture

Strukture omogočajo skupinjenje podatkov,ki so lahko različnega tipa, v enoto.

Splošna oblika:

struct etiketa { tip element; tip element; };

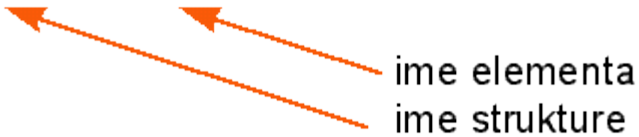
Primer 1:

```
struct oseba {
      char ime[10];
      char priimek[20];
      int starost;
      float visina;
} ;
```

Lahko deklariramo spremenljivke, katerih tip ustreza definirani strukturi:
struct oseba oseba1, oseba2;

Dostop do elementov oziroma členov strukture dobimo na naslednji način:

oseba1.starost
oseba2.visina



Demo

Structura (struct)

- *struct* je nov tip podatka, definiran s strani uporabnika
- *struct* je izpeljan podatkovni tip, sestavljen iz članov, ki so vsi bodisi osnovni ali izpeljani podatkovni tipi.
- Ena *struct* lahko pomni podatke za en objekt. Polje več *struct* lahko pomni podatke za več objektov.
- *struct* lahko definiramo na več načinov, kot bomo spoznali v nadaljevanju:

C strukture: sestavljenke, pa vendar skalarji

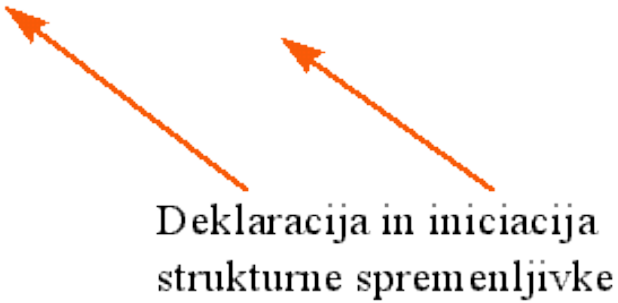
- Sestavljenke, saj lahko sočasno pomnijo več podatkov
  - Imenovani člani pomnijo podatke različnih tipov
  - V razliko od C++ teh članov ne moremo skrivati

- Skalarji, saj C obravnava strukturo kot eno celoto
  - imamo en kazalec na kolekcijo članov v pomnilniku
  - Cele strukture (ne le kazalce na strukture) lahko posredujemo kot argumente funkcij, lahko jim dodeljujemo spremenljivkam itd
  - Ne moremo pa jih primerjati z uporabo `==` (bilo bi preveč neučinkovito)

Še dva primera

Primer 2:

```
struct oseba oseba1={“Peter”,”Novak”,41,186} ;
```



Primer3:

```
oseba1.starost = 42;
strcpy(oseba1.ime,"Peter");
.......
oseba2 = oseba1;   /* Dopustno je kopiranje celotne strukture */
```

Deklaracije struktur

```
struct RojstniDan {
    int leto, mesec, dan; char *ime;
};
```

S tem nismo deklarirali nobene spremenljivke, imamo pa sedaj nov tip “*struct Novo*” , ki ga lahko navajamo kasneje

```
struct RojstniDan mama, oce, brat;
```

Uporabimo prej deklarirano *struct RojstniDan* , in tako deklariramo spremenljivke *mama,oce,brat* takega tipa

```
typedef struct {int leto, mesec, dan; char *ime;} RojstniDan;
```

Tako pa deklariramo preprosto ime RojstniDan (bolj prikladno od *struct RojstniDan*)

```
RojstniDan mama, oce, brat;
```

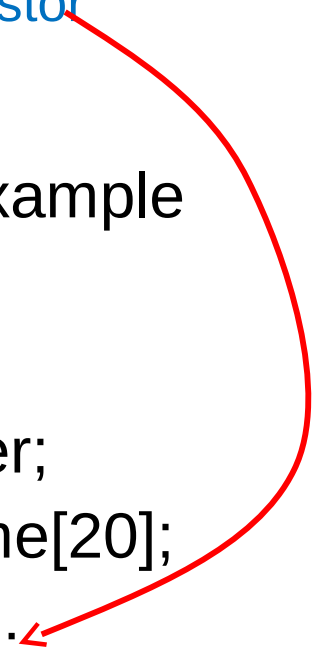
Deklaracija struktur (struct)

Ne rezervira prostora (le deklaracija)

```
struct myExample
{
    int label;
    char letter;
    char name[20];
} ;
```

Rezervira prostor

```
struct myExample
{
    int label;
    char letter;
    char name[20];
} mystruct ;
```



Imenu "myExample" pravimo structure tag

Dostop do članov strukture

- Do posameznih članov spremenljivke *struct* dostopamo s uporabo operatorja “pika” (“.”):

```
mojaStruktura.podatek ;
```

- Lahko pa uporabimo kazalec na *struct* , ki ga deklariramo in inicializiramo

Kazalec na strukturo

Naslov strukture

```
imeStrukture *mojKazalec = &mojaStruktura ;
```

in uporabimo kazalčni operator na strukturo (“->“):

Tako naslovimo člen strukture, ki jo kaže kazalec

```
mojKazalec -> podatek ;
```

Kar bi lahko zapisali tudi tako:

```
(*mojKazalec).podatek ;
```

Polja struktur

*Deklarirajmo polje 20 struktur tipa **oseba** in mu dajmo ime **delavec** :*

```
struct oseba {
    char ime[20];
    int starost;
    int visina ;
} delavec[20];
.............
delavec[0].starost = 41;
strcpy(delavec[0].ime,”Peter”);
.......
printf("Vpisi ime delavca:"); gets( delavec[1].ime);
.......
for(i=0; i<20; i++){
    printf("Delavec %s je star %d",   delavec[i].ime, delavec[i].starost);
}
```

Primer ponazoruje tudi vnos in izpis, skratka uporabo takega polja.

Demo1 Demo2

Strukture v strukturah

Elementi v strukturi so sami po sebi lahko tudi strukture:

```
struct podatki {
    int visina;
    int starost;
}

struct oseba {
   char ime [20]
   struct podatki lastnost;
}

. . . . . .
struct oseba oseba1;
. . . . . .
oseba1.lastnost.visina = 187;
oseba1.lastnost.starost = 35;
```

Deklaracija vsebovane strukture

Deklaracija strukture

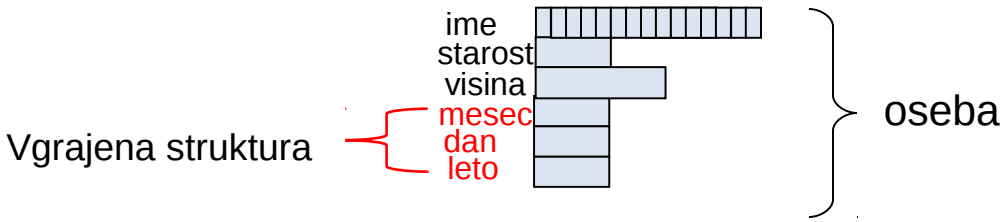
Deklaracija strukturne spremenljivke

Uporaba gneznede strukture

Vgrajene strukture

```
struct oseba{
    char ime[41];
    int starost;
    float visina;
    struct {
      int mesec;
      int dan;
      int leto;
    } rojstvo;
 };
```

Vgrajena struktura




```
struct oseba mama;

mama.rojstvo.leto=1950;……

struct oseba student[60];   /* Polje s podatki o studentih v letniku */

student[0].ime="Janez"; student[0].rojstvo.leto=1971;……
```

Strukture kot argumenti funkcij

- Strukture so skalarji, zato jih lahko vračamo in posredujemo kot argumente, tako kot *int, char, double*

```
struct BIG changestruct(struct BIG s);
```

Kopira se cela struktura

Vračamo celo strukturo

  - Klic po vrednosti: tvorjena je začasna kopija strukt
  - Pozor: posredovanje velikih struktur je lahko neučinkovito
    - vključuje mnogo kopiranja
- Temu se izognemo s posredovanjem kazalca na strukturo:

```
void changeStruct(struct BIG *p);
```

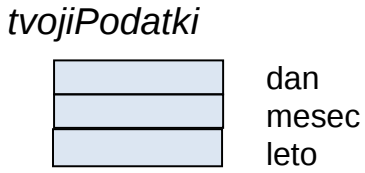
Kopira se le kazalec na strukturo

LINUX: Primeri dolgih struktur

Posredovanje, vračanje strukture

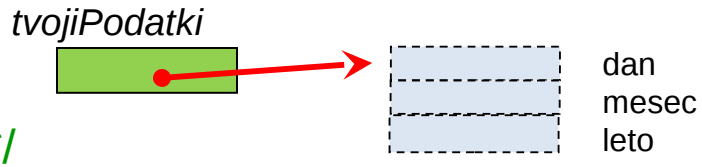
Posredovanje strukture po vrednosti

```
void prikazLeta (struct rojstvo tvojiPodatki) {
  printf("Rojen si %d\n", tvojiPodatki.leto);
}                              /* ni učinkovito, zakaj ? */
. . . .
```



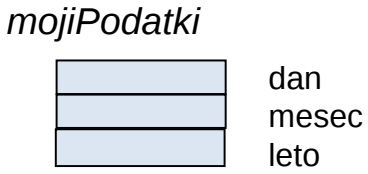
Posredovanje strukture po referenci

```
void prikazLeta2(struct rojstvo *tvojiPodatki) {
  printf("Rojen si %d\n", tvojiPodatki->leto);
      /* Pozor! '->', in ne '.',  po kazalcu na strukturo*/
}
```



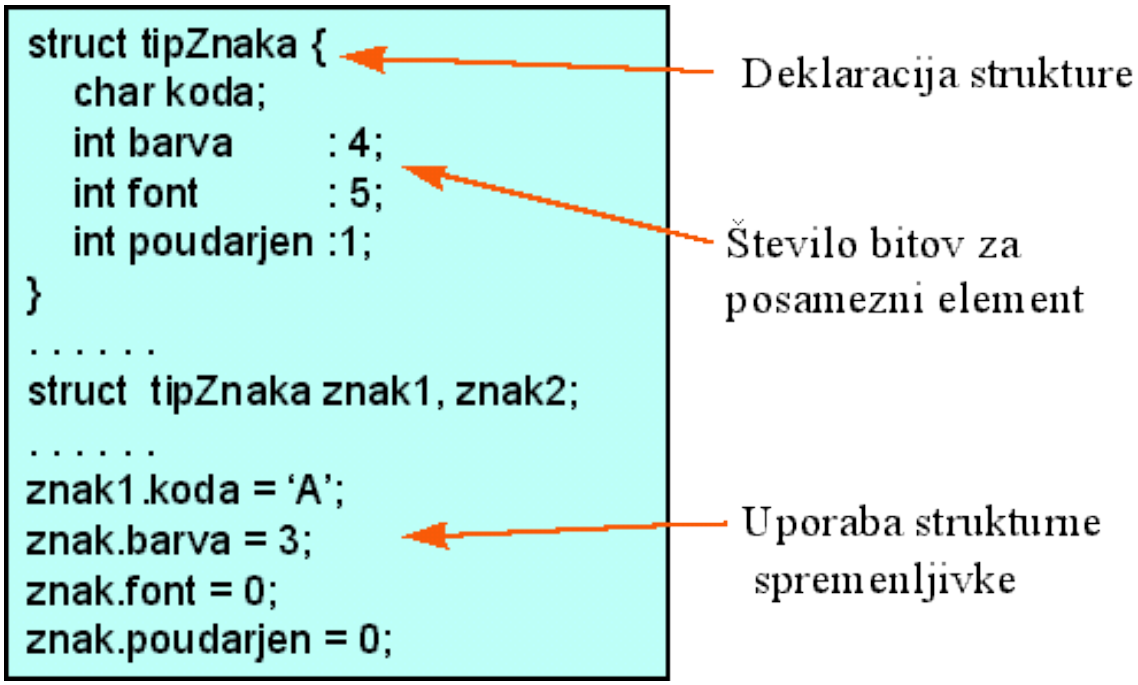
Vračanje strukture po vrednosti

```
struct rojstvo getRojstvo (void){
  struct rojstvo mojiPodatki;
  mojiPodatki.leto=1971;  /* '.' po strukturi */
  return mojiPodatki;
}                              /* Tudi neučinkovito, zakaj ? */
```



Strukture z bitnimi polji

Elementi v strukturi so lahko tudi zlogi, dolgi le nekaj bitov. To je uporabno predvsem pri sistemskem programiranju.

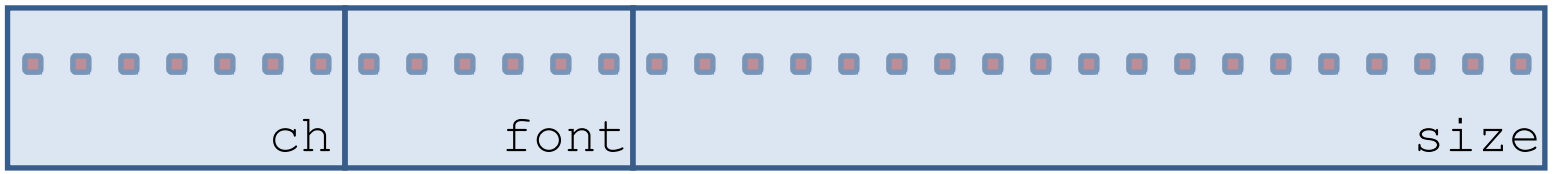


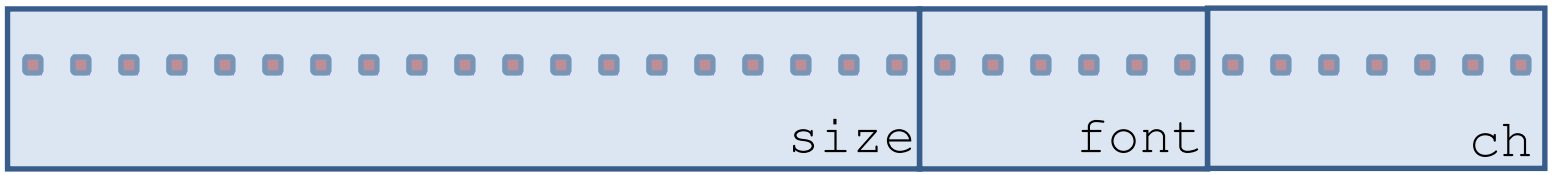
Strukture z bitnimi polji (primer)

Da prihranimo prostor, lahko za posamezne člane določimo število uporabljenih bitov

```
struct CHAR { unsigned ch: 7;
              unsigned font: 6;
              unsigned size: 19; };
```

Razpored v pomnilniku je odvisen od tipa računalnika





Unije

Podobne strukturam, vendar vsak član zaseda isto področje v pomnilniku!

- Strukture: člani so eden ob drugemu in pripadajo istemu lastniku
- Unije: člani zasedajo isti del pomnilnika

```
union VALUE {
  float f;
  int i;
  char *s;
};
/* ali float ali int ali niz */
```

Kaj vidiš: mlado ženo, staro ženo, moža?

Unije (2)

- Prostor v pomnilniku
  - Velikost unije je velikost največjega člana
  - Izogibajmo se unijam z zelo različnimi velikostmi članov;

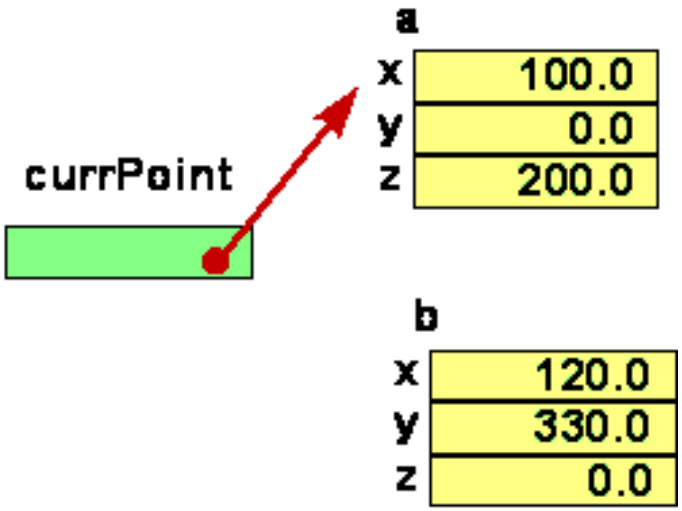
    za večje podatkovne tipe raje uporabljamo kazalce
- Initializacija
  - Unijo lahko inicializiramo le na vrednost, primerno tipu njenega prvega člana

Kazalci na strukture

*Definirajmo **strukturo point**, ki ima kot člene koordinate točke. Imejmo dve taki točki oziroma strukturi: a in b. Imejmo tudi kazalec na tako strukturo. Končno s pomočjo tega kazalca **naslavljajmo** obe **strukturi** in **nastavimo vrednosti** njunih **elementov**:*

```
struct point {
    double x,y,z;
};

struct point a, b, *currPoint;
 . . . . . .
currPoint = &a;
(*currPoint).x = 100.0 ; (* currPoint).y= 0.0;
(*currPoint).z = 200.0;
 currPoint =&b;
currPoint->x = 120.0;  /* bolj pregledno */
currPoint-> y = 330.0;
currPoint-> z = 0.;
```



Rekurzivno definirane strukture

Če dve strukturi naslavljata ena drugo, mora biti ena deklarirana s prototipom

```
struct clovek;

struct zival {
  char ime[10];
  char pasma[20];
  struct clovek *lastnik;
} fido = {" Fido" , " labradorec" };

struct clovek {
  char ime[20];
  struct zival zivali[4];
} janez = {" Janez" , {fido}};
```

Lastnika še ne moremo navesti, saj ga še nismo deklarirali

Janez ima samo Fida, lahko pa bi imel več živali

Dostop do članov

- Operator za direktni dostop `s.m`
  - *Indeks* in operator *pika* imata enako prednost in jih upoštevamo od leve k desni. Zato ne potrebujemo oklepajev za

    `janez.zivali[0].pasma`
- Indirektni dostop `s->m`: kar je enako kot `(*s).m`
  - Preusmerimo kazalec na strukturo in nato navedemo člana te strukture
  - .Operator pika ima višjo prednost pred operatorjem indirekcije. Zato potrebujemo oklepaje: (*s).m

`( *fido.lastnik ).ime` ali `fido.lastnik->ime`

. Najprej ocenimo in dobimo naslov lastnika
** Ocenimo in nato preusmerimo kazalec na cloveka*

. in -> imata enako prednost in ju upoštevamo od leve na desno

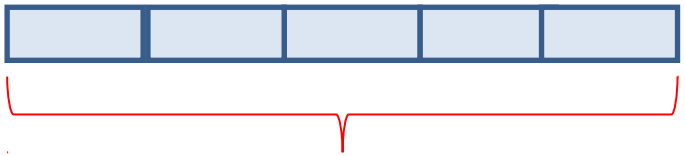
Uporaba calloc za tvorbo struktur

```
#include < stdio.h >
struct oseba{
    char ime[15];
    int ocena;
};

struct oseba  *student;
int num, i;
void vnos(struct oseba *p) {
    printf("Vpisi ime in oceno:");
    scanf("%s %d",  p->ime, &p->ocena);
}

int main( int argc, char *argv[ ]) {
    if(argc<2) exit(1);
    num = atoi(argv[1]) ;
    student =(struct oseba*) calloc(num,sizeof(struct oseba));
    for(i =0 ;i< num; i++) vnos(&student[i]);
    . . . . . .
for (i = 0; i< num;i++)       printf("%s %d \n", student[i].ime, student[i].ocena);
    exit(0)
 }
```

*Program ponazoruje uvod v preprosto "podatkovno bazo". Zapisi v tej bazi so podatki o študentih. Vsak zapis pomnimo v **strukturi** tipa **oseba**, ki jo moramo prej definirati. Program **prebere iz komandne vrstice**, koliko študentov imamo. Za vsakega moramo **dinamično alocirati strukturo**. Vpis in izpis podatkov demonstrira **naslavljanje teh struktur s kazalcem**.*

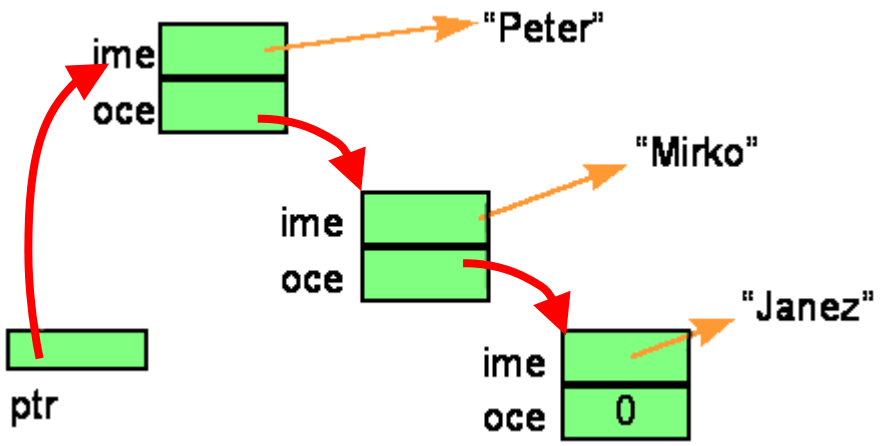


V bistvu je to polje num struktur

PPT Demo

Kazalci v strukturah

Posamezni členi v strukturah so lahko tudi kazalci na druge strukture. Tako lahko realiziramo cele sezname struktur, povezanih s kazalci.



Rdeč kazalec kaže na strukturo enakega tipa

Primer:
Uvedimo strukturo, ki podaja sorodstvene relacije. Osnovni podatek v vsaki strukturi je ime. Ker so ***imena*** *običajno* ***različne dolžine****, je bolje, če zanje* ***alociramo pomnilnik*** *posebej -* ***dinamično****. V strukturi pa imamo* ***kazalec*** *(naslov) na tako pomnjeno ime. Drugi podatek v strukturi je tudi* ***kazalec na strukturo enakega tipa****. To omogoča* ***povezovanje struktur****.*

Uporaba malloc za tvorbo struktur

```
struct person {
        char zacetnice[4];
        long int emso;
        int starost;
        struct person *oce;
        struct person *mama;
} *janez, *metka, *miha;

int main() {
        janez = (struct person *)malloc( sizeof( struct person ) );
        miha= (struct person *)malloc( sizeof( struct person ) );
        metka= (struct person *)malloc( sizeof( struct person ) );

        strncpy(janez->zacetnice, “JN“, 2);
        miha->emso= 555235512;
        miha->oce = janez;
        miha->mama = metka;
        metka->starost= 68;
        /* ker je miha kazalec, je pravilno miha->mama->starost */
        printf(“mama od mihe je stara  %d let ", miha->mama->starost);
}
```

PPT

Demo

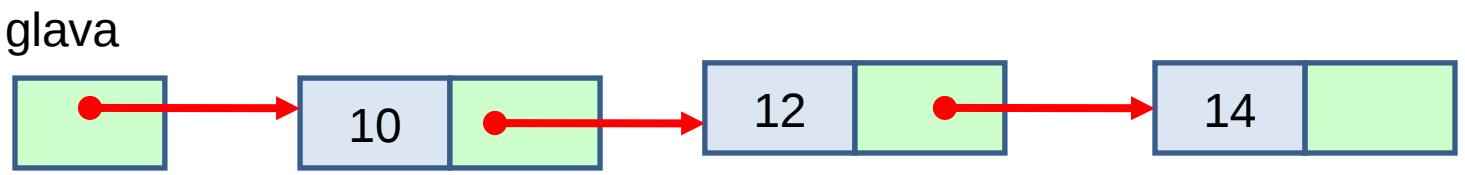
Sklad: demo

Primer implementacije sklada s poljem (stack.c)
Primer organizacije programa z dvema datotekama (projekt)
Primer uporabe "header" datoteke (stack.h)

Primer programa, ki tak sklad uporabi (stacktest.c)

Povezani seznami (linked lists)

- Povezan seznam je seznam, ki ga lahko širimo ali krčimo med samim tekom programa
- Povezan seznam naredimo s pomočjo kazalcev
- Povezan seznam je pogosto sestavljen iz struktur, ki vsebujejo kazalec, ki povezuje tako strukturo na druge dinamične spremenljivke (oziroma strukture)
- Povezane sezname prikazujemo kot škatle, med seboj povezane s puščicami

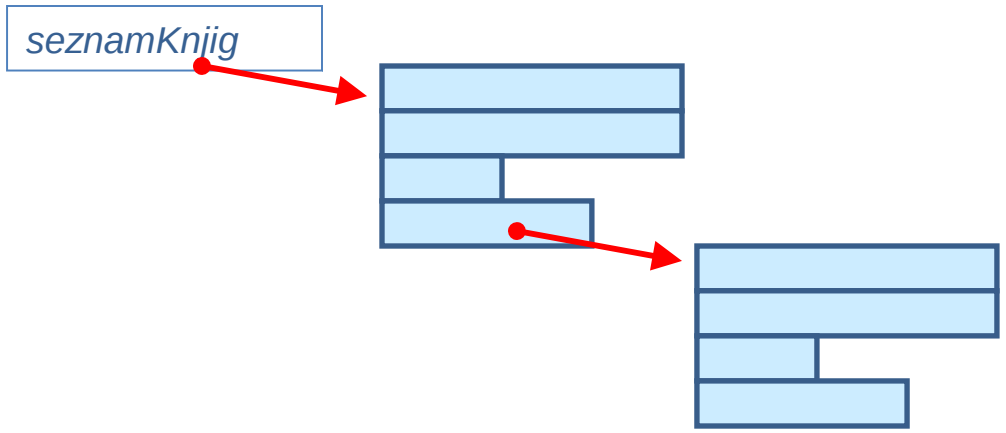


Primer

```
struct knjiga {
        char naslov[32];
        char avtor[32];
        int cena;
        struct knjiga * next;
};

struct knjiga * seznamKnjig;

void addKnjiga(void) {
    struct knjiga * temp;
        char buffer[8];
        temp = (struct knjiga*) malloc(sizeof(struct knjiga));
        printf("Vnesi naslov naslov:"); gets(temp->naslov);
        printf("Vnesi avtorja:"); gets(temp->avtor);
        printf("Vnesi ceno:");   gets(buffer);   temp->cena = atoi(buffer);
        printf("Knjigo %s dodajam v seznam...", temp->naslov);
        temp->next = seznamKnjig;
        seznamKnjig = temp;
}
```



seznamKnjig

Povezani seznami: demo

Tvorba povezanih seznamov v C
Program prikazuje, kako tvorimo preprost linearen povezan seznam s pomočjo dinamične alokacije pomnilnika in kazalcev PPT

Zakaj je izpis v obratnem vrstnem redu, kot smo podatke vnašali?
Kako bi naredili izpis v drugačnem vrstnem redu? PPT

Kako bi poiskali element v seznamu? PPT

Kako bi brisali element iz seznama? PPT

Cgrasp: vizualizacija podatkovnih struktur

Več o povezanih seznamih

- Sekcija 1 — Osnovne strukture povezanih seznamov in koda
- Sekcija 2 — Osnove tvorbe seznamov
- Sekcija 3 — Tehnike kodiranja povezanih seznamov

WEB