Programski jezik C

Zakaj naj bi se učili C?

Praktični razlogi:
C je *de facto* standard za sistemsko programiranje
Programi, pisani v jeziku C so bolj hitri

Izobraževalni razlogi:
Java ne omogoča vpogleda v nizkonivojske mehanizme
Računalnikar bi moral poznati izvajanje programov na različnih nivojih

Prednosti uporabe C:
boljši nadzor nad obnašanjem programa med njegovim izvajanjem
Več možnosti za uglaševanje performans programa
Dostop do nizkonivojskih mehanizmov sistema

Slabosti uporabe C:
Sami moramo skrbeti za upravljanje s pomnilnikom
tipično potrebujemo več vrstic kode za izvedbo neke naloge
več možnosti za napake

Kaj pa C++ ?

C++ je dodal objektno usmerjenost osnovnemu semantičnemu modelu C.
Java je predstavljala implementacijo novega, objektno usmerjenega jezika na osnovi nekaterih konceptov C.
Mnenje: C++ ni čist ... Če hočete programirati objektno usmerjeno, uporabljajte objektno usmerjen jezik.

Primer preprostega programa

```
/* malo komentarja */

#include <stdio.h>

main()
{
   printf(“Pozdravljen:\n”);
   printf(“Kako ti je ime:”);
}
```

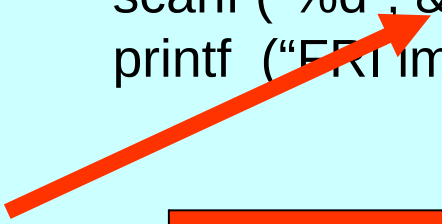
Prevajalnik razlikuje velike in male črke v našem programu.

Namesto besed *begin* in *end* uporabljamo zaviti oklepaj in zaklepaj.

Namesto glavnega programa imamo v kodi obvezno funkcijo *main()*.

Še en primer !

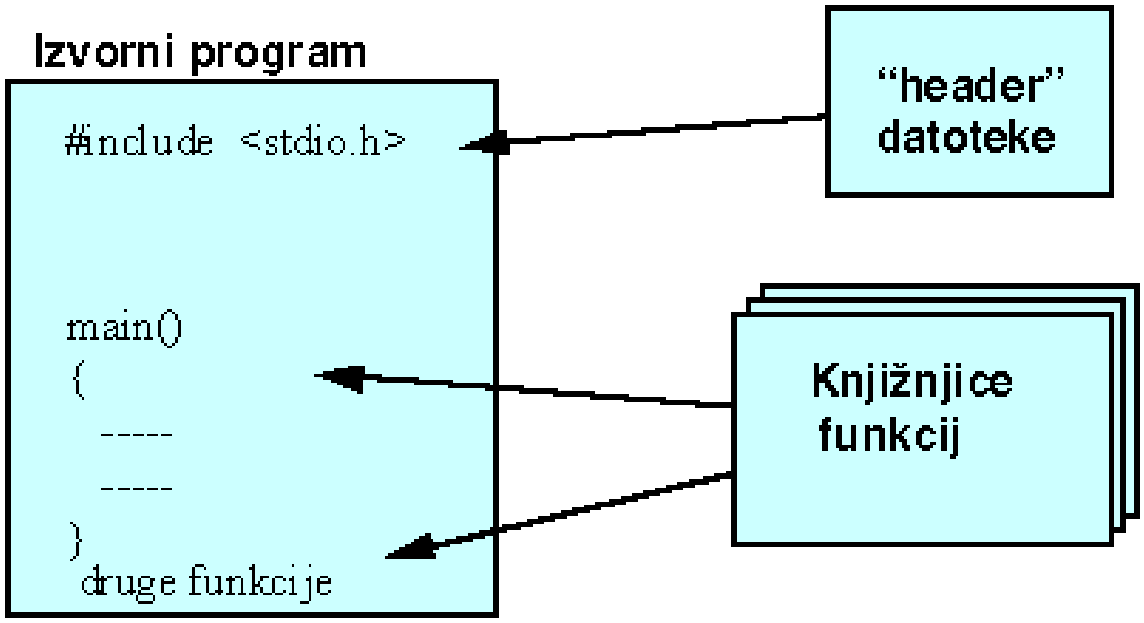
```
#include <stdio.h>

void main(void)
{
        int nStudents = 0; /* Initialization, required */

        printf  (“Koliko studentov ima FRI ?:”);
        scanf (“%d”, &nStudents);  /* Read input */
        printf  (“FRI ima %d studentov.\n”, nStudents);

}
```



Pozor na znak & pred imenom spremenljivke

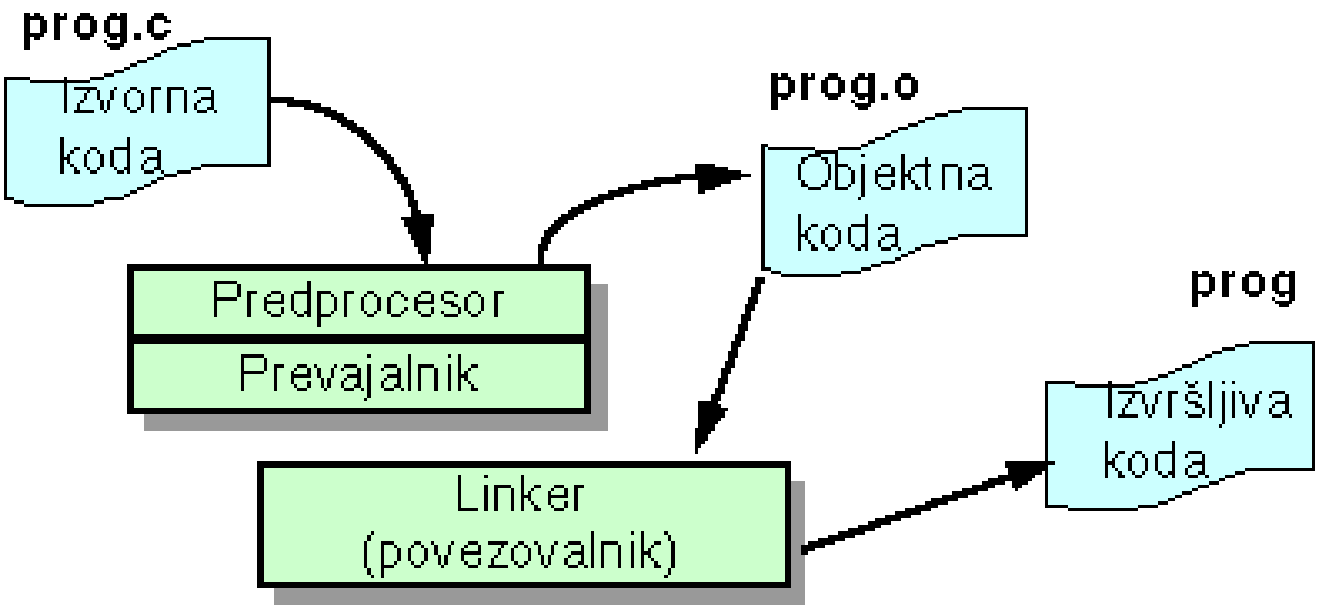
```
$Koliko študentov ima FRI ?: 1600 (enter)
FRI ima 1600 studentov.
$
```

Izgled programa v jeziku C



Skoraj ni programa, ki ne bi imel na začetku navedenih "header" datotek, ki jih mora prevajalnik vkljuciti v program. Prav tako je običajno, da v programu uporabljamo funkcije, ki so na voljo v posebnih "knjižnicah".

Priprava programa v jeziku C



Program v jeziku C zapisemo v datoteko, ki ima koncnico **.c** Sam prevajalnik ima vec podmodulov. Eden od njih je predprocesor, ki obravnava razlicna navodila oziroma direktive. Te se tipicno zacno z znakom #.
Datoteka s prevodom izvornega programa ima kratico **.o** (objektna datoteka).
Prevedeni program ima obicajno ime ***a.out***, vendar mu normalno damo bolj pametno ime.

Primeri ukazov za prevajanje

Spremenljivke in konstante

- **Konstante**
- **spremenljivke**

Atributi:
- ime
- vrednost
- podatkovni tip
- način pomnenja

Podatkovni tipi:
integer
int
long
short
unsigned
realni
float
double
znaki
char

Posebni operator sizeof() pove, koliko bytov zaseda neka podatkovna struktura

Kot pri vsakem programskem jeziku so tudi tu osnovni gradniki programa spremenljivke in konstante.

Preprosti tipi podatkov

Tip podatkov	*Velikost v bitih*	*Obseg*
char	8	-128 do 127
signed char	8	-128 do 127
unsigned char	8	0 do 255
short int	16	-32768 do 32767
unsigned int	16	0 do 65535
int	16	-32768 do 32767
long	32	-2147483648 do 21474836487
unsigned long	32	0 do 4294967295
float	32	3.4×10^{-38} do 3.4×10^{38}
double	64	1.7×10^{-308} do 1.7×10^{308}
long double	80	3.4×10^{-4932} do 1.1×10^{4932}

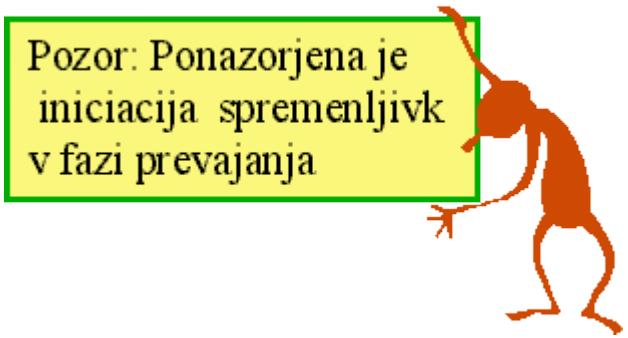
Imena spremenljivk

Začno s črko
Poleg črk lahko vsebujejo številke in znak _
Največ 255 znakov
Razlikujemo velike in male črke
Ne smemo uporabljati rezerviranih besed

Rezervirane besede

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

Primeri imen in deklaracij




```
int     a,  numPoints;
char    ch, answer = "\033";
double  value, max_value = 100.0;
```

Pri deklaraciji spremenljivk lahko pomagamo prevajalniku (optimizatorju), tako, da mu napovemo, ali bo neka spremenljivka imela stalno ali spremenljivo vrednost:

```
const double    e= 2.718281828;
volatile char   answer;
```

Razred *volatile prevajalniku pove, da se lahko spremenljivka spreminja v procesih, ki tečejo v ozadju in je torej ni dovoljeno* optimizirati.

Oštevilčeni tipi spremenljivk

Definicija oštevilčenih tipov (enumerated types) ima naslednjo splošno obliko:

enum etiketa {seznam vrednosti};
enum etiketa imeSpremenljivke;

Primer: enum dnevi
{poned,torek,sreda,cetrtek,petek,sobota}; enum dnevi dan; . .
dan= sreda;

Prevajalnik C obravnava oštevilčene označbe kot celoštevilčne konstante (0, 1, 2,..)

Definicija novih tipov operandov

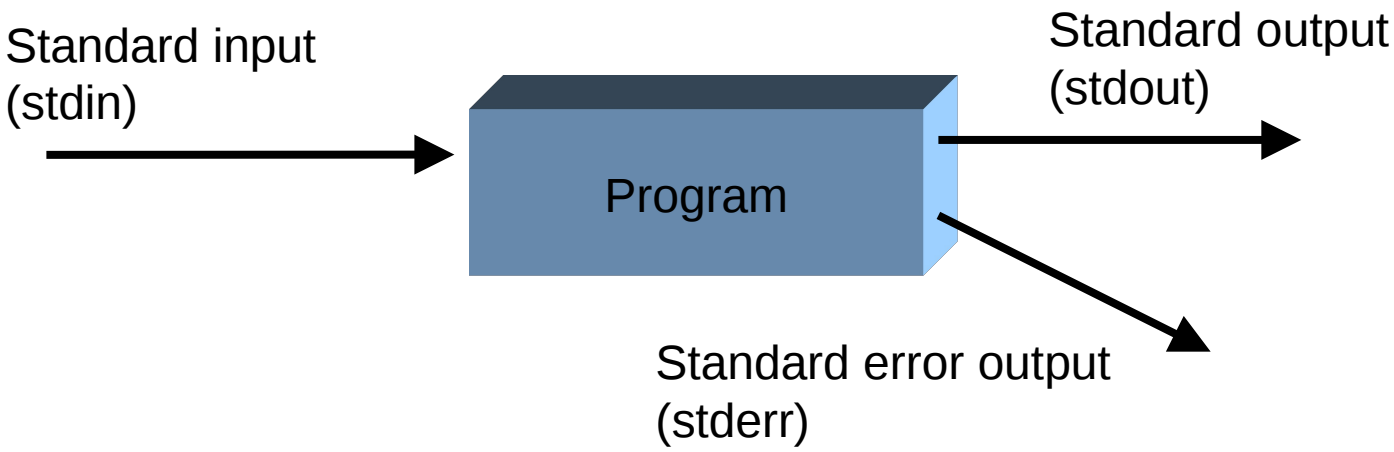
Splošno:
typedef oldType newName;

Primer:
enum logical {FALSE, TRUE};
typedef enum logical boolean;
boolean status;

Vhodno izhodne funkcije

Funkcije s standardnim vhodom, izhodom:

int printf (format [,arg, arg,..arg])	Formatiran izpis na standardni izhod
int scanf (format [,kazalec, kazalec, ..])	Formatirano branje s standardnega vhoda
int getchar( )	Branje znaka s standardnega vhoda
int putchar (int)	Izpis znaka na standarni izhod
char *gets(char str[80])	Branje niza s standardnega vhoda
char *puts(char str[80])	Izpis niza na standardni izhod



Formatirano branje in izpis

Primer:

```
#include <stdio.h>
int   starost, stevCevljev;
main( ) {
    printf ("Koliko imas stevilko cevljev:");
    scanf ("%d", &stevCevljev);
    printf ("Torej rabis copate stev %d \n",stevCevljev);
}
```

Pozor na znak & pred imenom spremenljivke v stavku scanf, ker mora biti za vhodni parameter podan naslov in ne vrednost spremenljivke

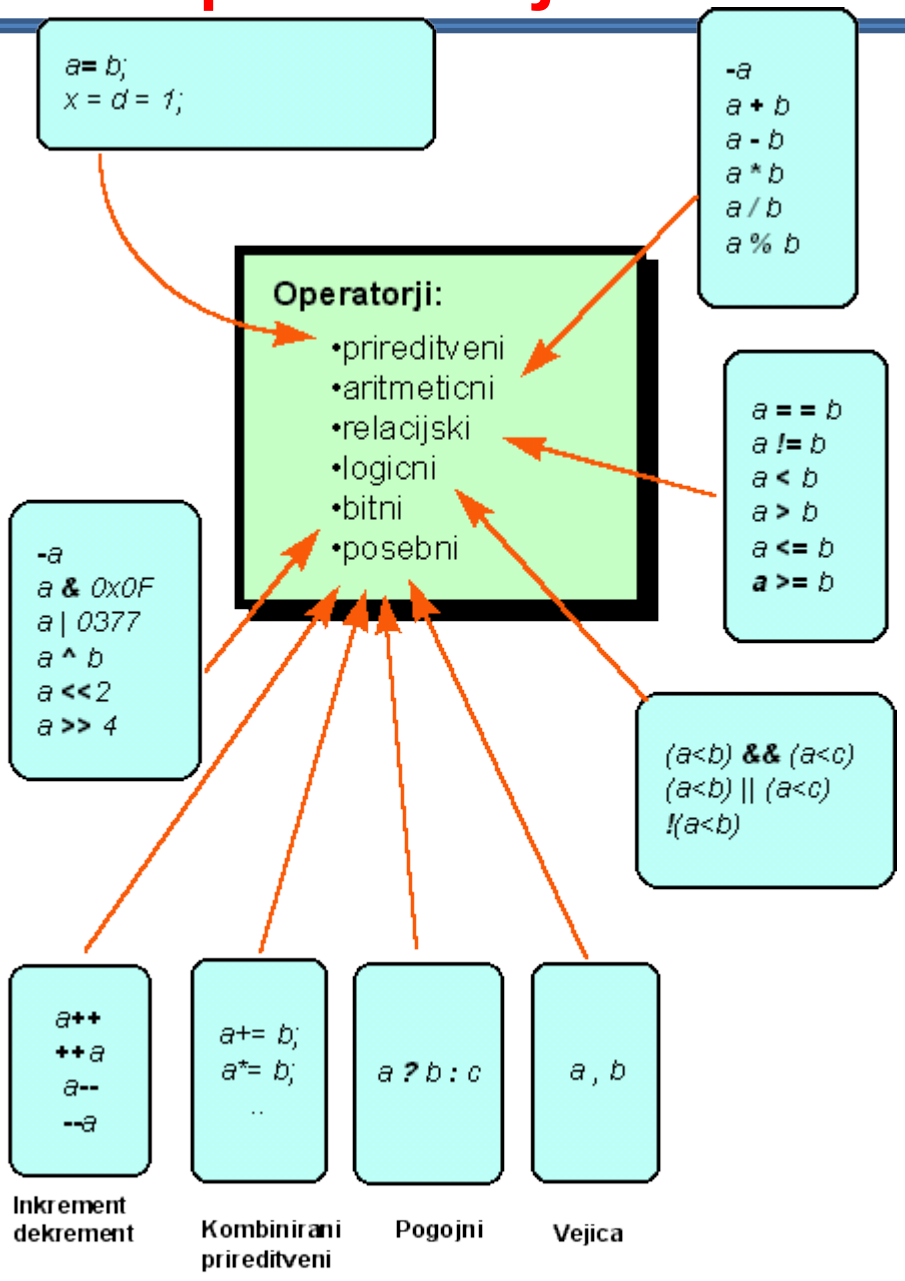
Splošna oblika:

printf (format, seznam spremenljivk ali konstant);
scanf(format, seznam naslovov spremenljivk);

printf in scanf - konverzijske specifikacije

d	Desetiška cela stevila
u	Desetiška cela števila brez predznaka
o	Osmiška števila
x	Šestnajstiška števila (male črke abcdef)
X	Šestnajstiška števila (velike črke ABCDEF)
i	Cela števila, osnova definirana z notacijo
f	Realna števila tipa float
e	Realna števila,znanstvana notacija (črka e)
E	Realna števila,znanstvena notacija (črka E)
g	Realna števila, format odvisen od velikosti
G	Isto, le črka E namesto e
c	Posamezni znaki
s	Nizi, ki so zaključeni s kodo 0

Pregled operatorjev in izrazov



Aritmetični operatorji

Operator	Pomen	Primer
+	seštevanje	a+b
-	odštevanje	a-b
*	množenje	a*b
/	deljenje	a/b
%	modulo (ostanek celoštevilčnega deljenja)	a%b
++	pred inkrement (poveča vrednost spremenljivke za 1 in jo nato uporabi)	++a
++	po inkrement (uporabi spremenljivko in ji nato poveča vrednost za 1)	a++
--	pred dekrement (zmanjša vrednost spremenljivke za 1 in jo nato uporabi)	--a
--	po dekrement (uporabi spremenljivko in ji nato zmanjša vrednost za 1)	a--
-	unarni minus	-a
+	unarni plus	+a
+=	seštevanje in prirejanje (a+=b pomeni a=a+b)	a+=b
-=	odštevanje in prirejanje (a-=b pomeni a=a-b)	a-=b
*=	množenje in prirejanje (a*=b pomeni a=a*b)	a*=b
/=	deljenje in prirejanje (a/=b pomeni a=a/b)	a/=b
%=	modulo in prirejanje (a%=b pomeni a=a%b)	a%=b

Logični operatorji

Operator	Pomen	Primer
&	bitni IN (AND)	a&b
|	bitni ALI (OR)	a|b
^	bitni ekskluzivni ALI (XOR)	a^b
~	komplement	~a
!	negacija	!a
<<	pomik levo (a<<b pomakne a za b bitov v levo)	a<<b

Posebni operatorji

Kombinirani prireditveni operatorji:

Splošna oblika:
izraz1 op= izraz2;

Pomeni isto kot:

izraz1 = izraz1 op izraz2;

*(velja za operatorje: + - * / % << & | ^)*

Pogojni operator:

Splošna oblika:
izraz1 ? izraz2 : izraz3
Pomen:

Če je vrednost izraz1 TRUE (ni nič), potem je celotni izraz po vrednosti in tipu enak izrazu2 sicer je celotni izraz po tipu in vrednosti enak izrazu3

Operator vejica:
Splošna oblika:
izraz1 , izraz2
Pomen:
Ocenita se oba izraza, celoten izraz ima vrednost in tip desnega izraza.

Izrazi

Imajo tip in vrednost. So kombinacija operandov in operatorjev.

Primeri:

```
pogoj =  a < b;
rezultat = (a > 4) < 6;
stanje = !(a < b);
rezultat = x >> 2; ++a;  /* kar je enako a = a+1 */;
a = --b - 2;
predznak = (x < 0) ? -1 : 1 ;
```

Nepravilna uporaba operatorjev

Izraze pišemo pregledno in nedvoumno!

Slabo:

```
z= x++ - y/x--;
z= -x / y;
z= x++ + ++y / z-- *5;
z= (x++ ==4 || y-- <= 5);
```

Dobro:

```
x++;  z=x-y/x;  x--;
z= (-x) /y;
z=((x++) + (++y)) /( (z--)*5)
z=(x==4||y<=5); x++;  y--;
```

Konverzija tipa podatka

Avtomatična:

Do avtomatične konverzije pride med tipi:
char, short int, int

Potrebna:

V naslednjih dveh primerih imejmo dve spremenljivki:

int a; float b;

Jasno je, da mora priti v naslednjem stavku do konverzije tipa izraza iz **float** v **int**: **a = b;**

Zahtevana:

V naslednjem primeru konverzijo eksplicitno zahtevamo:
a = (int) b;

Konverzija tipa podatka po standardu ANSI

Če je eden od operandov **long double**, bo tak tudi drugi.

Sicer če je en operand **double**, bo tak tudi drugi.

Sicer če je en operand **float**, bo tak tudi drugi.

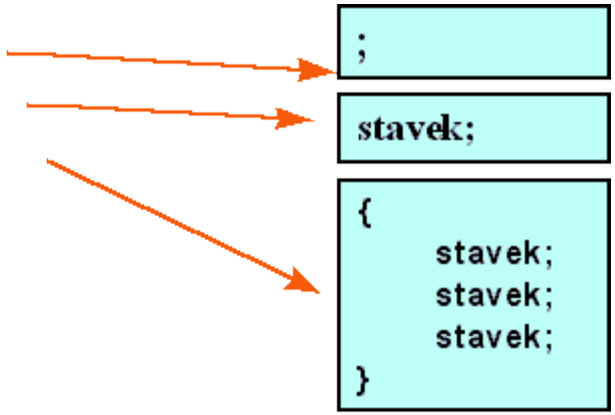
Operand tipa **char** ali **short int** postane tipa **int**.

Če je en operand tipa **long int**, bo tak tudi drugi.

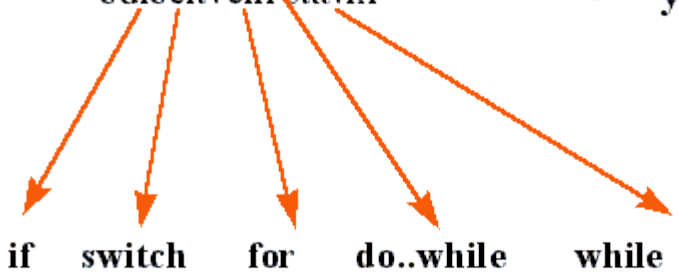
Sicer pa bo izraz tipa **int**.

Vrste stavkov

- prazen stavek
- preprost stavek
- sestavljen stavek



- **Preprosti stavki:**
  - •klic funkcije
  - •odlocitveni stavki



Na hitro nekaj o funkcijah

(Ker brez njih pač ne gre)

Splošna oblika klica funkcije:

[vrednost =] imeFunkcije(arg1, arg2,..,argn);

Primeri:
ch = getchar();
printf("Pozdravljeni");

Opomba:
getchar() bere znak, vendar ga dobimo šele po vtipkanju ENTER. Rezultat getchar() je tipa int. To omogoča, da lahko vemo, kdaj je branje neuspešno (ne moremo na primer brati po koncu vhoda, tedaj vrne -1 (kar pomeni EOF (end of file)))

Odločitveni stavek if

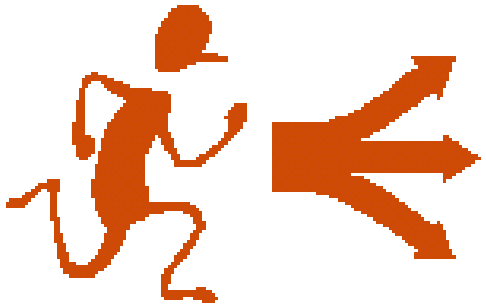
Splošna oblika

if (izraz) stavek1; else stavek2;

Primer:

```
printf("Vpisi x in y:");
scanf("%d %d",&x, &y);
if(x==y)
    printf("Enaki vrednosti\n");
else    printf("Vrednosti sta razlicni \n");
```

Odločitveni stavek switch



Splošna oblika:

```
switch(izraz) {
     case (A):
          stavek_A;
          break;
    case (B):
          stavek_B;
          break;
    default:
          stavek_X;
}
```

Opomba:
Stavek default normalno pišemo na koncu. Lahko pa tudi manjka.

Stavek switch: primer

```
printf ("Izberi eno od moznosti:");
switch( getchar() )   {
        case ('1'):
                vnosPodatkov();
                break;
         case ('2'):
                izracun();
                break;
         case('3');
                izpisRezultatov();
}
```

Iterativni stavki (zanke)

Splošne oblike

for (Inicializacija; Pogoj; Inkrement) stavek;
while (izraz) stavek;
do stavek while (izraz);

Zanke: primeri

```
for(i=1; i<=10;i++)printf("2 X %d = %d\n",i,2*i);/* postevanka */

while ( (ch = getchar())!= EOF) putchar(ch);     /* Prepis vhoda na izhod */

float stevilo, vsota = 0;
do{
      printf("Vpisi stevilo:");
      scanf("%f", &stevilo);
      vsota += stevilo;
}while (stevilo != 0) ;
printf(" Vsota je %f\n ", vsota);
```

Stavki break, continue, goto

Stavek *break*
Povzroči izstop iz (najbolj notranje) zanke tipa for, while ali do..while.
Uporabljamo ga tudi pri zaključku alternativ v odločitvenih stavkih switch

Stavek *continue*
Je podoben stavku break in ga uporabljamo v zankah (for, while, do..while). V razliko od stavka break ne povzroči izstopa iz zanke ampak le preskok vseh stavkov (ki so za njim) v dani iteraciji.

Stavek *goto*
Povzroči direkten prehod na stavek z dano etiketo

Primer:

```
if( failure)  goto errorMessage ;
 .....
errorMessage:  printf( "Action failed");
```

Polja

Primer deklaracije polja:

```
double  vsota, cena[20];    /* polje ima 20 elementov */
int i;
```

Uporaba:

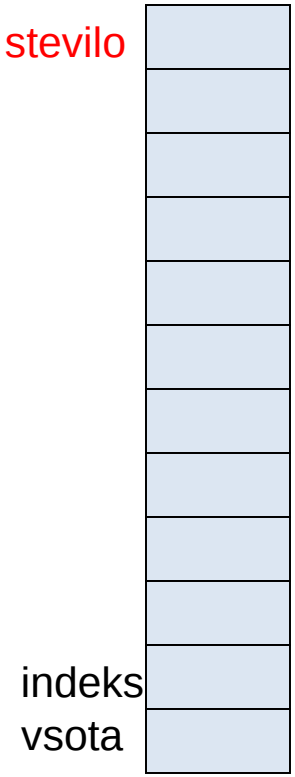
```
cena[0] = 100.0;            /* prvi element ima indeks 0 */
cena[1] = 150.0;
.....
vsota = 0;
for(i=0,i<20;i++) vsota  += cena[i];
```

Primeri deklaracije in istočasno iniciacije vrednosti:

```
int dnevi[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
char pozdrav[ ] = {'P','o','z',d','r','a','v','l','j','e','n'};
```

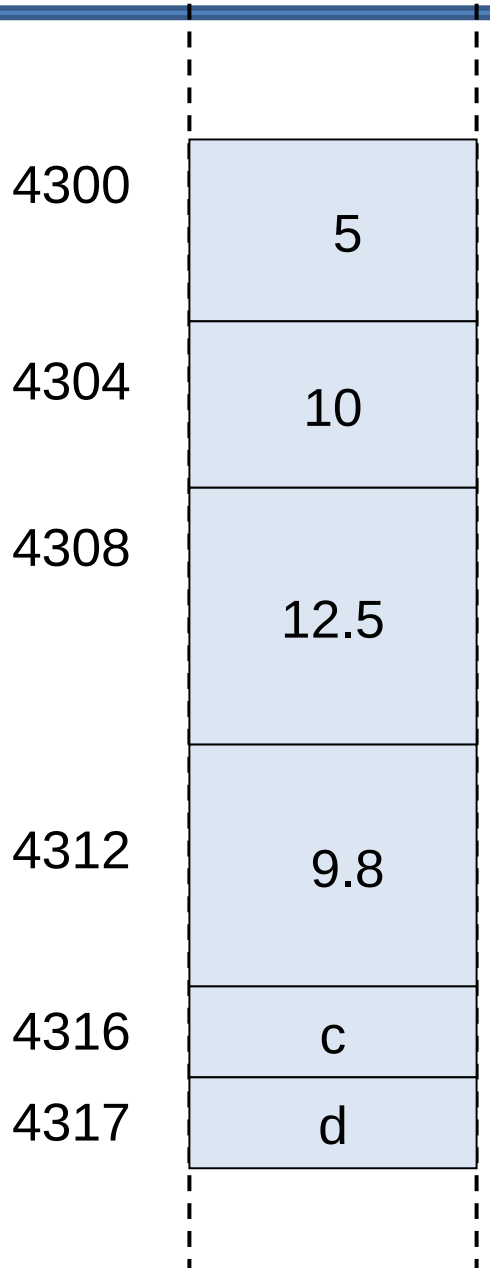
Enodimenzijska polja

```
#include <stdio.h>

void main(void)
{
    int stevilo[12];          /* 12 elementov polja */
    int indeks, vsota = 0;
    /* Vedno inicializiraj vrednosti pred uporabo */
    for (indeks = 0; indeks < 12; indeks++) {
            stevilo[indeks] = indeks;
    }
    /* stevilo[indeks]=indeks bi sedaj povzročil napako, zakaj ?*/

    for (indeks = 0; indeks < 12; indeks = indeks + 1) {
            vsota += stevilo[indeks];  /* vsota elementov polja */
    }

    return;
}
```

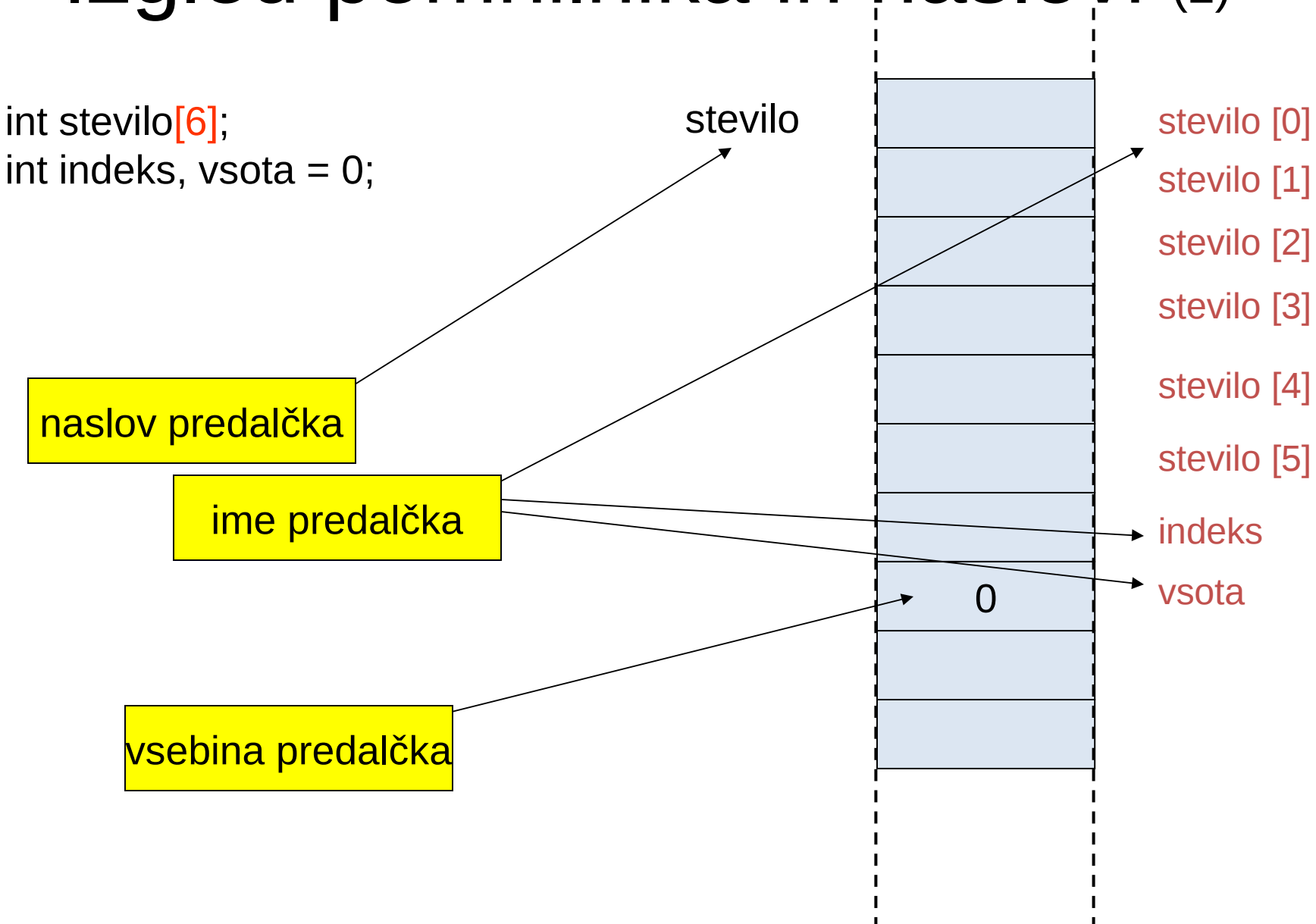


Izgled pomnilnika in naslovi

```
int    x = 5, y = 10;
float f = 12.5, g = 9.8;
char c = ‘c’, d = ‘d’;
```



Izgled pomnilnika in naslovi (2)



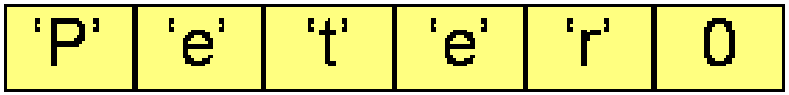
Večdimenzijska polja

Primeri polj z numeričnimi vrednostmi:

```
char screen[25][80];
int mat[2][3] = {{1,2 3},{2,4,6}};

for(i=0; i<2;i++){
    for(j=0; j<3;j++) screen[i][j] = mat [i][j]+'0';
}
```

1	2	3
2	4	6

Polja znakov (nizi)



char Priimek[6] = {'P','e','t','e','r','\0'};

Lahko pa to deklariramo tudi na boljši način:

char Priimek[ ] = "Peter";

Nize zaključuje znak '\0'

char name[6] = {'L','j','u','b','l' ,'j' ,'a' ,'n' ,'a','\0'}; /* '\0'= konec niza */
printf("%s", name); /* izpisuje do '\0' */

Vhodno izhodne operacije z nizi

Primer 1:

```
char ime[20]; printf("Kako ti je ime:");
scanf("%s",ime);   /* pozor, ni znaka & pred ime */
printf("Pozdravljen %s", ime);
```

In računalnik bo napisal: Pozdravljen Peter

Primer 2:

```
char ime[20]; printf("Kako ti je ime:");
gets(ime);
printf("Pozdravljen %s",ime);
```

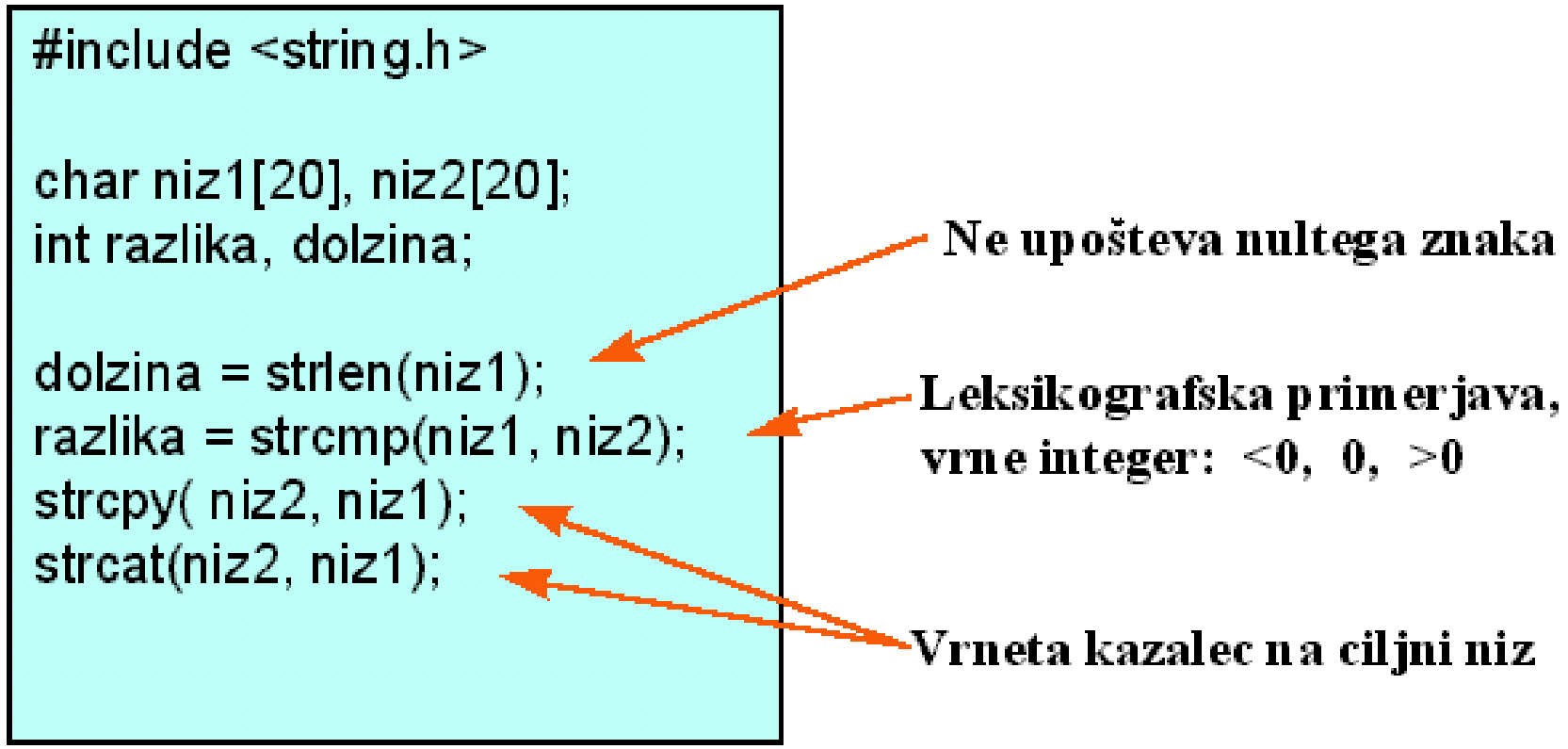
In računalnik bo napisal: Pozdravljen Peter Novak

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.
int strcpy (s2, s1)	Kopira niz s1 v niz s2.
int strncpy(s2, s1, n)	Kopira niz s1 v niz s2, vendar največ n znakov.
char *strcmp (s2, s1)	Primerja niza in vrne: •pozitivno vrednost če je s2>s1, •0..če je s2=s1 •negativno vrednost sicer
char *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

Funkcije z nizi (nadaljevanje)

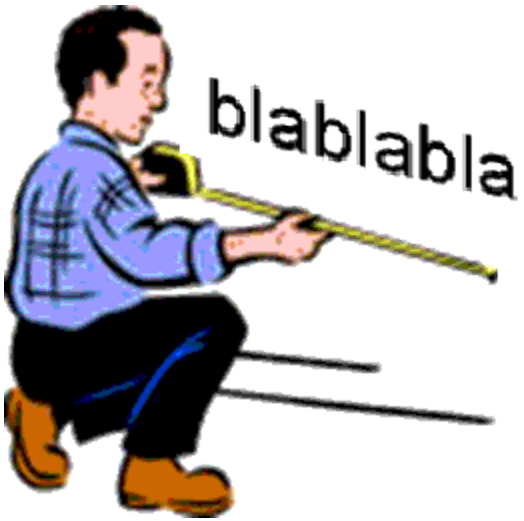
```
#include <string.h>

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... */
}
```

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);
}
```

Primer 3: kopiranje in povezovanje nizov

```
/* Primer 3: 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, "Tom");
        strncpy(priimek, "Jones", strlen("Jones");
        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);
}
```

Uvod v strukture

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

Splošna oblika:

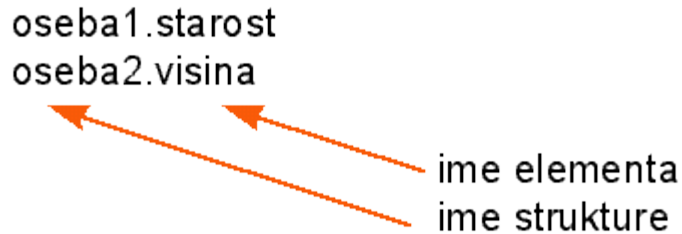
struct etiketa { tip element; tip element; };

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;

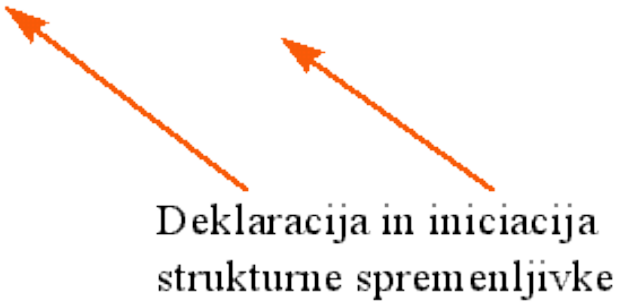
Dostop do elementov oziroma členov strukture dobimo na naslednji način:



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 */
```

Deklaracija struktur (struct)

Ne rezervira prostora

```
struct my_example
{
     int label;
    char letter;
     char name[20];
} ;
/* imenu "my_example" pravimo
     structure tag    */
```

Rezervira prostor

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

Polja struktur

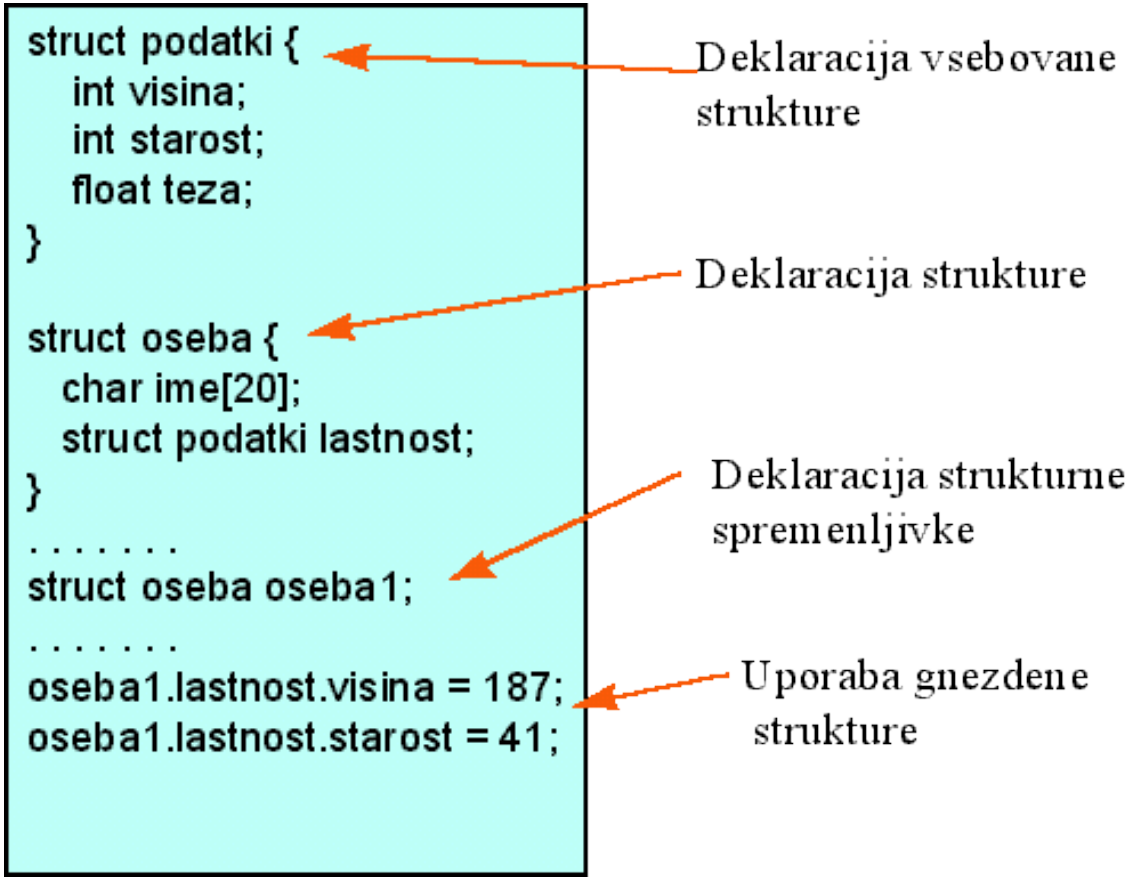
*Deklarirajmo polje 20 struktur tipa **oseba** in mu dajmo ime **delavec** :*

```
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.

Strukture v strukturah

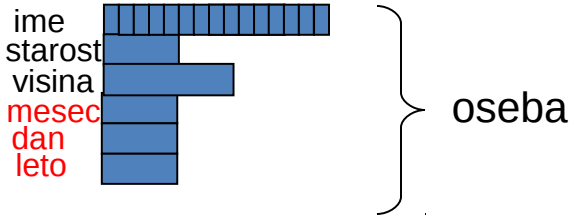
Elementi v strukturi so sami po sebi lahko tudi strukture:



Še o strukturah

```
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;……
```

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

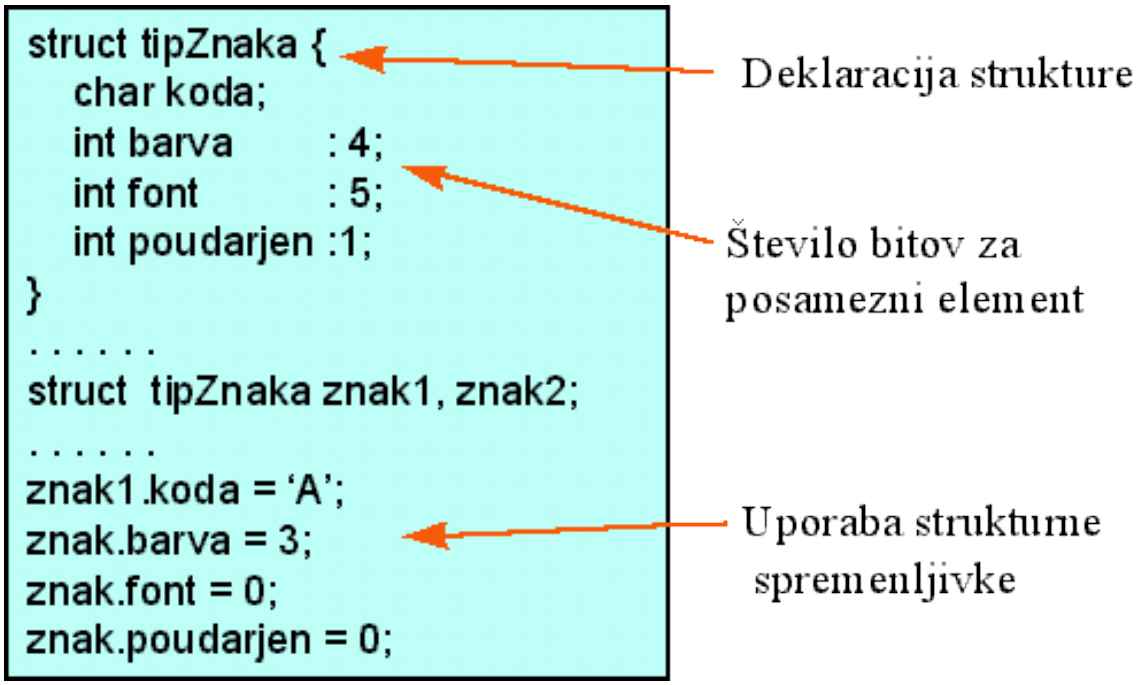
```
void prikazLeta2(struct rojstvo *tvojiPodatki) {
  printf("Rojen si %d\n", tvojiPodatki->leto);
       /* Pozor! '->', in ne '.',  po kazalcu na strukturo*/
}
```

Vračanje strukture po vrednosti

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

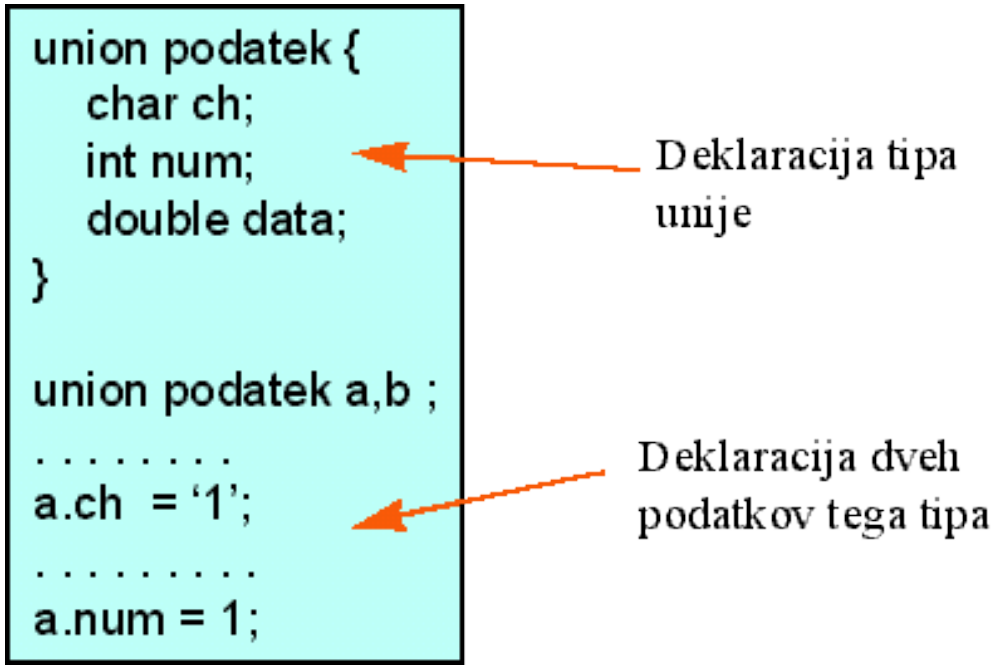
Strukture z bitnimi polji

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



Unije

Deklaracija unije je enaka deklaraciji strukture. Unija omogoča shranjevanje ***podatkov različnih tipov v isto pomnilno področje****.*



Funkcije – splošna oblika

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

Vsaka funkcija ima svoje imein 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");
}

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);
}

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.

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

Predhodna deklaracija funkcije (po ANSI C navajamo tudi tip argumentov)

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.

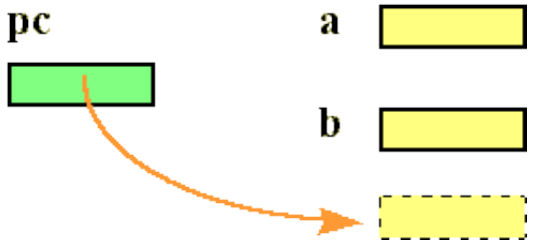
Kazalci

Deklaracija kazalca

char a,b , *pc ;

a in b sta spremenljivki tipa char, pc pa je kazalec na spremenljivko enakega tipa (pri tem se 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.

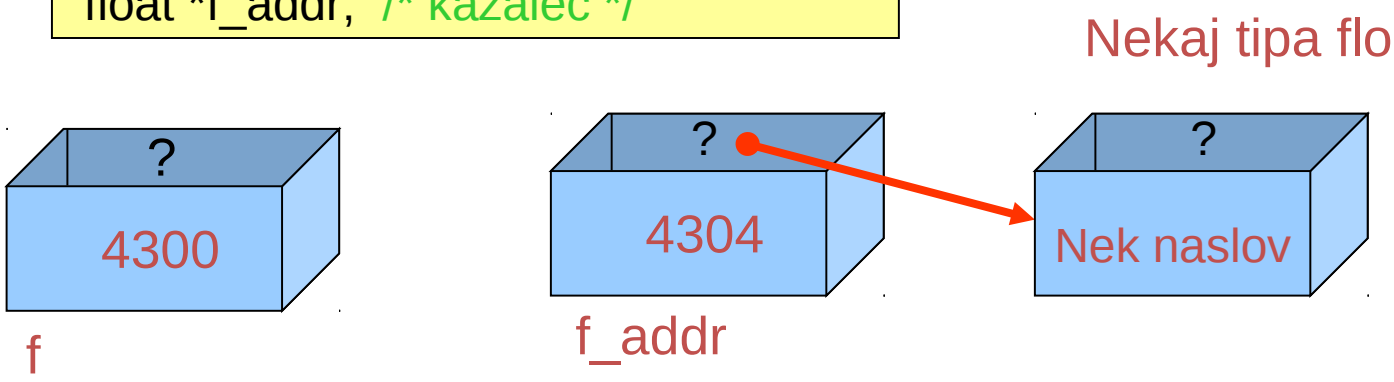
Primer uporabe:

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

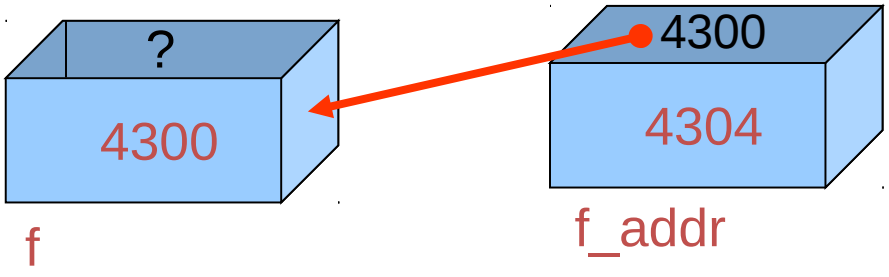
O kazalcih - 1

Kazalec ... spremenljivka, ki vsebuje naslov druge spremenljivke

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

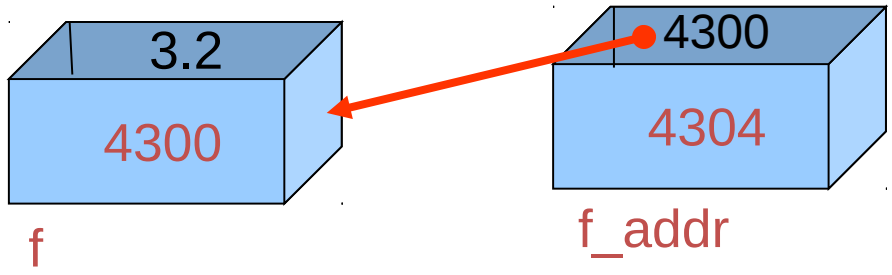



```
f_addr = &f; /* & ... operator naslavljanja*/
```

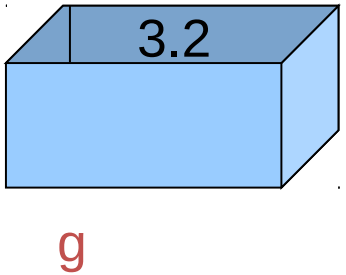


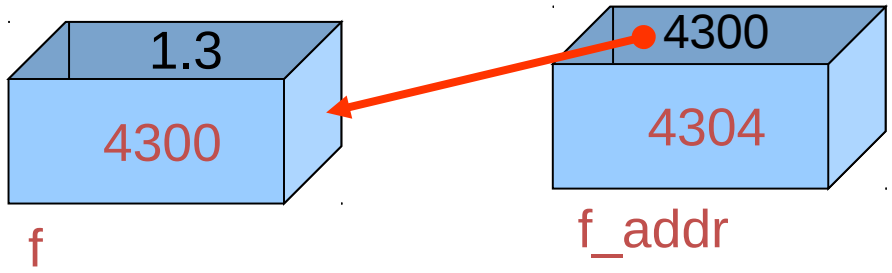
O kazalcih - 2

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

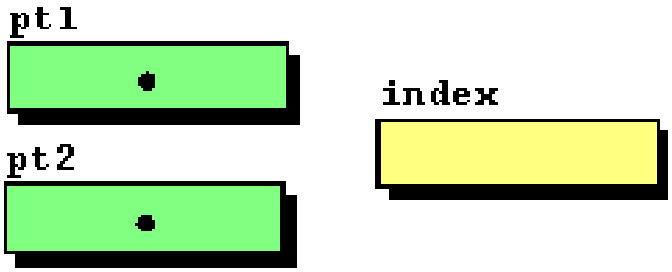



```
float g = *f_addr; /* indirekcija: g je sedaj 3.2 */
f = 1.3;
```



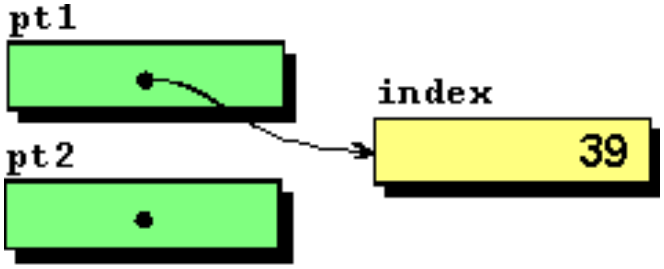


Primer s kazalci



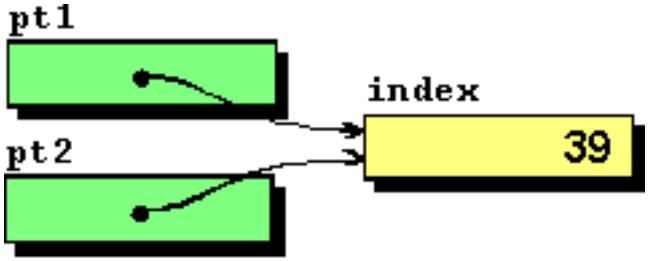

```
#include <stdio.h>

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;
}
```

Primer s kazalci



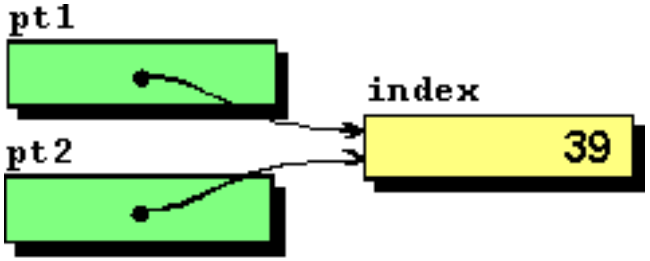

```
#include <stdio.h>

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

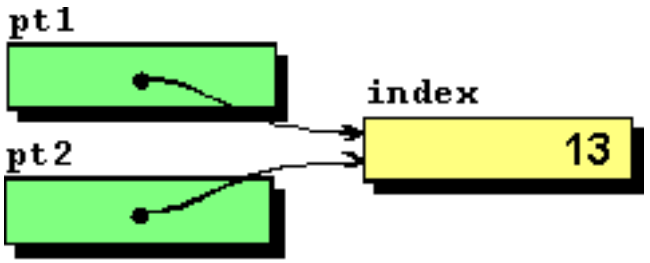



```
#include <stdio.h>

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;
}
```

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";
```

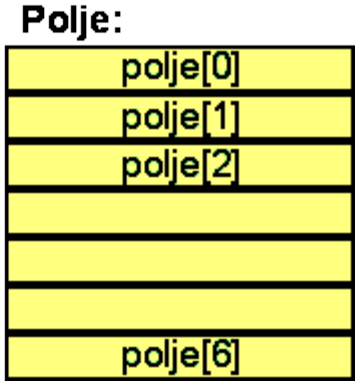
To pa je pravilno!

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

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]**

Ne glede na velikost posameznih elementov polja !

Zakaj kazalci kot argumenti funkcije?!

```
#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 vrednosti */
  int temp;

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

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).

Kaj je tu narobe ?

```
#include <stdio.h>

void nekajNaredi(int *ptr);

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

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

  ptr = &(temp);
}

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

Polja kot argumenti funkcije

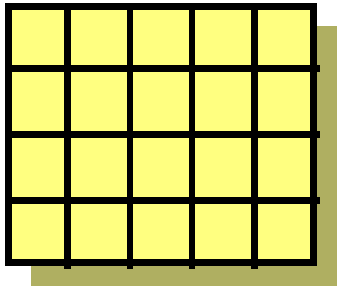
```
/* Primer izracuna maksimuma tabelirane funkcije */
#define N 20  /* Namesto stevilcnih konstant raje uporabljajmo simbole */
void main() {
    double x[N], y[N], yMax;
    int i;
    double max (double array[ ], int) ; /* vnaprejsnja deklaracija */

    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)

Kazalci na večdimenzijska polja

Primer:

A[4][5]



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]

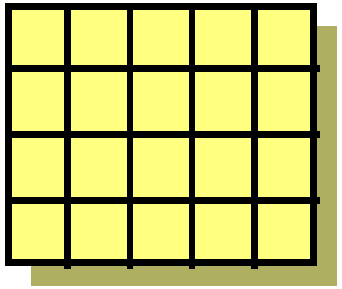
A in A[i] so konstantni kazalci (naslovi),ki jih ne moremo spreminjati.

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

Večdimenzijska polja kot argumenti funkcije

Primer:

A[4][5]



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

main() {
     clearMat(A);
}
```

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

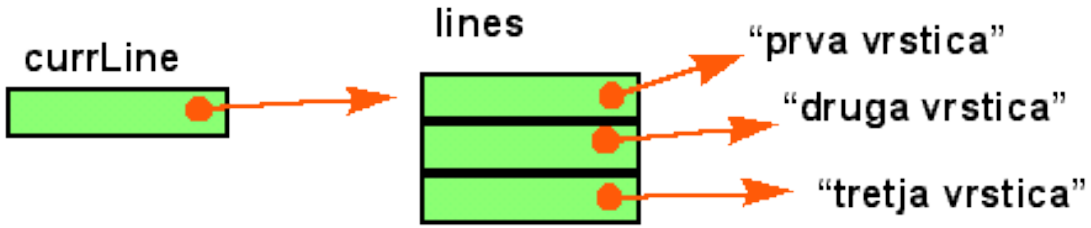
Kazalci na kazalce

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:

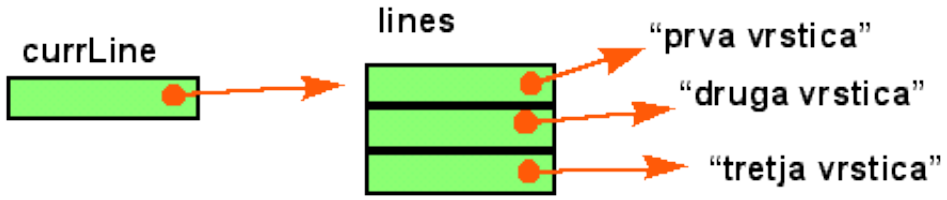


Primer uporabe kazalcev na kazalce

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

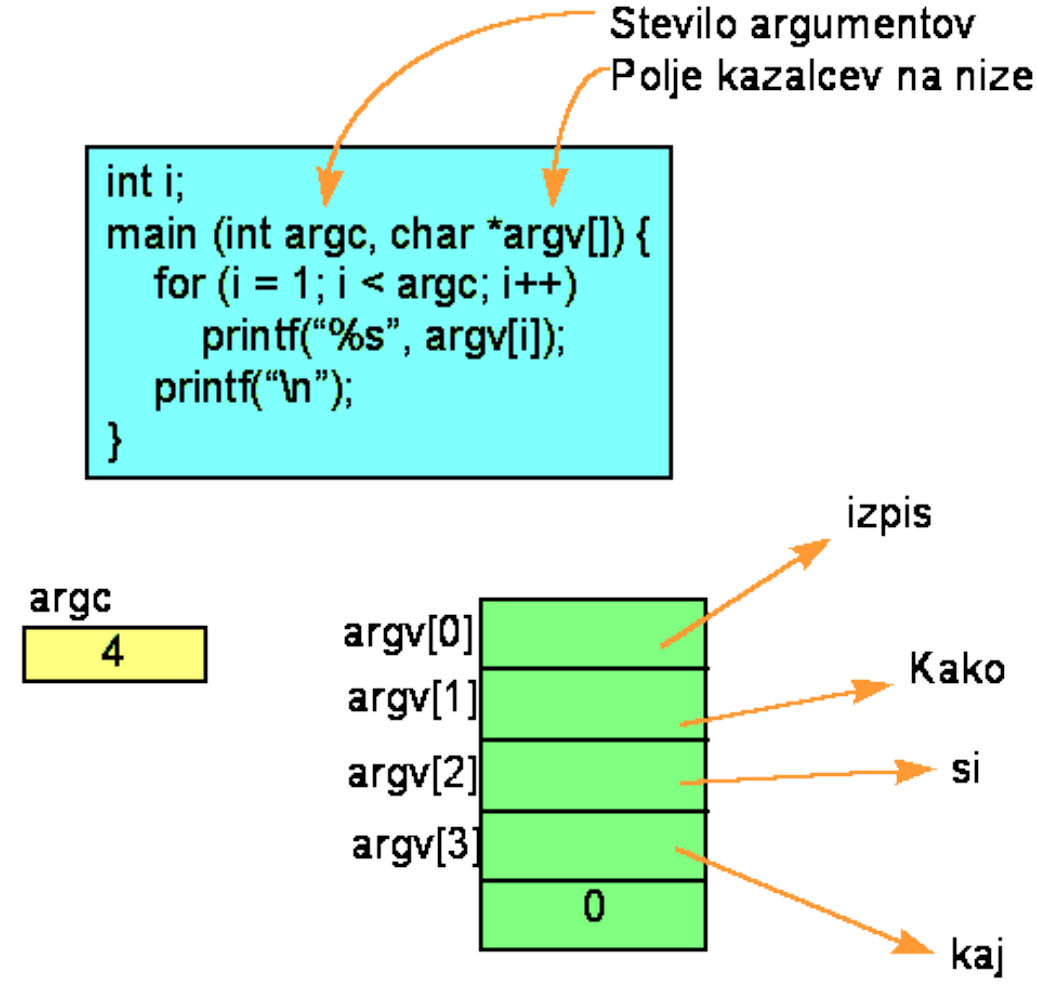


Argumenti v ukazni vrstici

Primer ukazne vrstice:

$izpis Kako si kaj

In še ustrezni program:



Dinamična alokacija pomnilnika

Primeri:

```
#include < stdlib.h >
double *p1, *p2, *p3;
 . . . . . .
p1 = (double*) malloc (100);
. . . . . .
p2 = (double*) calloc (100, sizeof(double));
. . . . . .
p3 = realloc (p1,200); . . . . . .
free(p2); free(p3);
```

*Funkcija **malloc** ima en sam argument, ki pove, koliko bytov pomnilnika rabimo.*

*Funkcija **calloc** ima dva argumenta: prvi pove število elementov, drugi pove velikost posameznega elementa (v bytih), zato uporaba operatorja sizeof().*
*Oba klica vrneta kazalec tipa "**void**". Zato praviloma kazalec pretvorimo v pravi tip z operatorjem "**cast**" (ki je v nasem primeru (double*))*

*Funkcija **realloc** spremeni polje, naslovljeno s kazalcem,za definirano število bytov. Če spomina ni dovolj, vrnejo vse tri funkcije vrednost 0.*

Eksplicitna alokacija in dealokacija

```
#include <stdio.h>

void main(void) {
        int *ptr;
        ptr = malloc(sizeof(int)); /* alociramo prostor za nek integer */

        *ptr=4;      /* ... Pomnilnik uporabimo... */

        free(ptr);    /* alocirani prostor spet sprostimo */
}
```

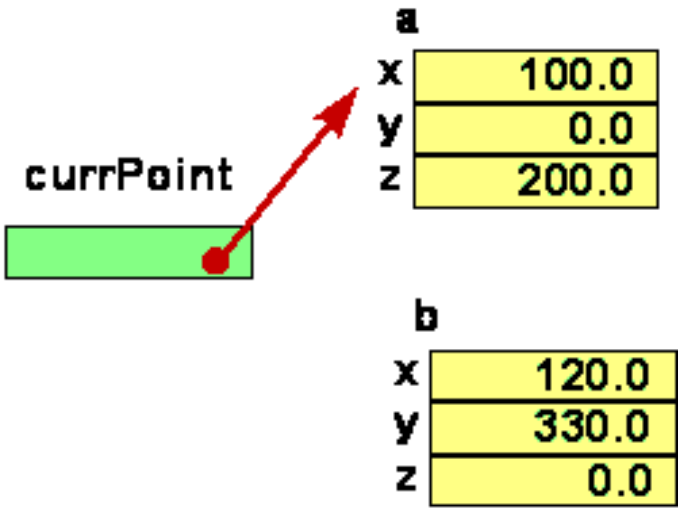
Demo C

Demo Java

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.;
```



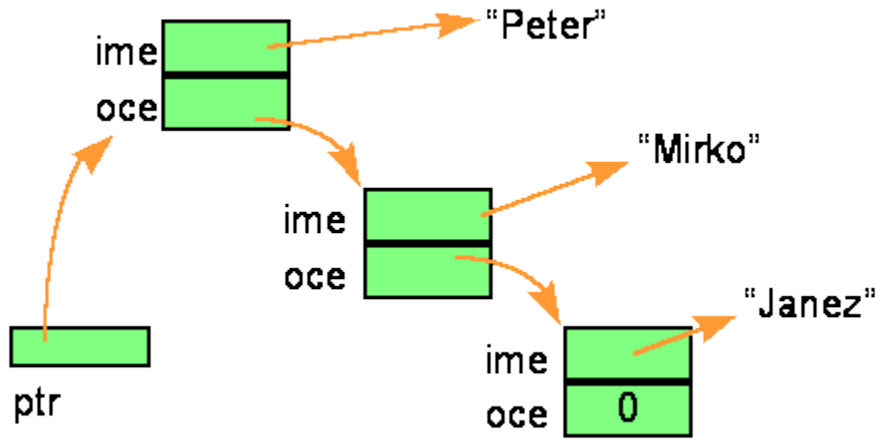
Primer 2

*Program ponazoruje uvod v preprosto "podatkovno bazo". Zapisi v tej bazi so podatki o študentih. Vsak zapis pomnimo v **strukturi** tipa **student**, 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**.*

```
#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);
}

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)
 }
```

Kazalci v strukturah

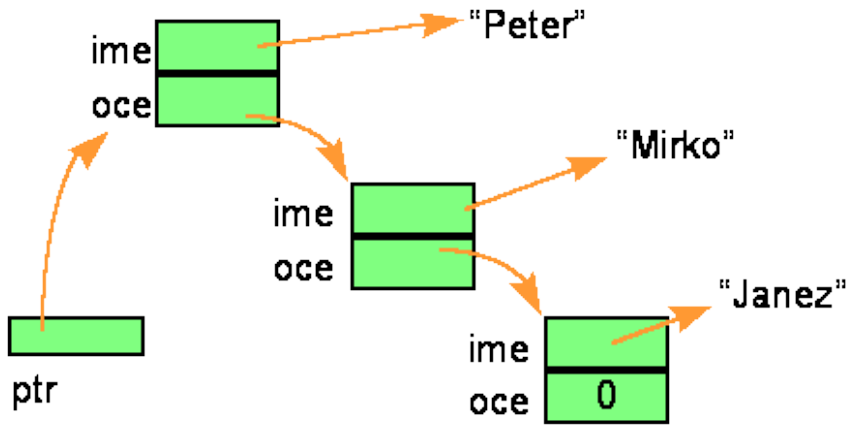
Posamezni členi v strukturah so lahko tudi kazalci na druge strukture. Tako lahko realiziramo cele sezname struktur, povezanih s kazalci.



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****.*

Primer - nadaljevanje

```
#include < stdio.h >
struct oseba {
    char *ime;
    struct oseba *oce;
}
oseba1 = {"Janez", NULL},
oseba2 = {"Mirko", &oseba1},
oseba3 = {"Peter", &oseba2};

struct oseba  *ptr; . . . . . ptr = &oseba3;
printf("sin je %s \n",ptr->ime);
printf("oce je %s \n", ptr->oce->ime);
printf("ded je %s \n",ptr->oce ->oce->ime);
/********* lahko pa tudi tako.. *********/
ptr = ptr->oce;
printf("ded je %s \n", ptr-> oce -> ime);
```



Vhodno izhodne operacije

V datoteki stdio.h so definirani:

kazalci na datoteke (FILE): stdin, stdout, stderr
NULL (ki je enak 0)
EOF (ki je enak -1)
FILE (ki je typedef za podatkovno strukturo)

Funkcije s standardnim vhodom, izhodom:

int printf (format [,arg, arg,..arg])	Formatiran izpis na standardni izhod
int scanf (format [,kazalec, kazalec, ..])	Formatirano branje s standardnega vhoda
int getchar( )	Branje znaka s standardnega vhoda
int putchar (int)	Izpis znaka na standarni izhod
char *gets(char str[80])	Branje niza s standardnega vhoda
char *puts(char str[80])	Izpis niza na standardni izhod

Delo z datotekami

Odpiranje datoteke
Primer:
Odprimo za branje datoteko z imenom **datoteka**.

FILE *fd ; fd = fopen("datoteka", "r");

Najprej smo morali definirati kazalec fd, ki ga bomo kasneje uporabljali pri vseh operacijah z odprto datoteko. Temu kazalcu pravimo "opisnik datoteke" (file descriptor). Drugi parameter v klicu funkcije fopen je v danem primeru "r", zato bo datoteka odprta za branje.

Seznam vseh možnosti

"r" Odpri datoteko za branje
"w" Odpri datoteko za pisanje. Če je še ni, jo tvori, če je že, jo povozi
"a" Odpri datoteko za pisanje. Če je se ni, jo tvori, sicer dodajaj na konec.
"r+" Datoteka bo odprta za branje in pisanje
"w+" Datoteka bo odprta za branje in pisanje
"a+" Datoteka bo odprta za branje in pisanje

Zapiranje datoteke
fclose (fd);

Branje iz datoteke... Zapis v datoteko

Pri navedbi možnih funkcij predpostavimo, da smo deklarirali dva kazalca na datoteki (fp1 in fp2), en niz (s) in eno spremenljivko tipa char:

```
FILE  *fp1, *fp2;   /* kazalca na datoteki */char *s          /* kazalec na niz */int c;int size, n;
```

Seznam funkcij za branje in zapis v datoteko:

c = getc (fp1);	**/* To je makro */**
c = fgetc(fp1);	**/* To je funkcija */ Branje znaka iz datoteke (podobno getchar())**
ungetc(c,fp1)	**Vrne znak na standardni vhod**
fgets (s,n,fp1);	**Branje (največ n) znakov iz datoteke v niz s**
fscanf (fp1, format, &arg1, &arg2,..);	**Formatirano branje, podobno kot scanf()**
fread (s, size, n, fp1);	**Neformatirano (binarno) branje n elementov,velikosti size, pomnjenih v polju s. Funkcija vrne število prebranih elementov.**
putc (ch, fp2)	**Zapis znaka v datoteko**
fputs (s,fp2);	**zapis niza v datoteko**
fprintf (fp2,format, arg1, arg2,..);	**Formatiran zapis v datoteko**
fwrite (s, size, n, fp2);	**Neformatirani (binarni) zapis n elementov velikosti size. s kaže na polje elementov. Funkcija vrne število zapisanih elementov.**

Primer

Kopiranje ene datoteke, znak po znak, v drugo:

```
#include < stdio.h >      /* Kopiranje datoteke */

void main() {
        FILE *fp1,  *fp2;
        int ch;
        if ((fp1= fopen("inpFile", "r")) ==NULL) {
            fprintf(stderr,"Datoteke ne morem odpreti");
            exit (1);
        }
        fp2 = fopen ("outFile", "w");
        while ( (ch = getc(fp1)) != EOF) putc(ch, fp2);
        fclose( fp1); fclose (fp2);
        exit (0);
}
```

Naključno branje in pisanje datoteke

Pri naslednjih funkcijah uporabljamo spremenljivko tipa "long integer", ki predstavlja odmik (in torej položaj v datoteki), merjen v bytih

```
FILE *fp;
 long odmik;      /* v datoteki */
 int odkod ;      /* od kod se steje odmik  */
```

Možne funkcije:

rewind (fp); Postavi odmik na začetek datoteke

fseek (fp, odmik, odkod);

Drugi parameter -odmik -pove, od kod dalje bo sledili branje ali zapis v datoteko. Tretji parameter ima lahko eno od vrednosti:

- SEEK_SET .... Začetek datoteke
- SEEK_CUR.....Trenutni položaj v datoteki
- SEEK_END......Konec datoteke

odmik = ftell (fp); Dobimo trenutni položaj (odmik) v datoteki

Preostale funkcije za delo z datotekami

ferror (fp) Vrne TRUE , če je indikator napake za dani (datotečni) tok (stream) setiran (zaradi neuspele predhodne vh/izh operacije)

feof (fp) Vrne TRUE, če želimo brati po koncu datoteke

clearerr (fp) Resetira indikatorja vh/izh napake in konca datoteke

fflush (fp) Sistem zapisuje podatke v datoteko preko medpomnilnika. Ta funkcija forsira prepis podatkov iz medpomnilnika v datoteko.

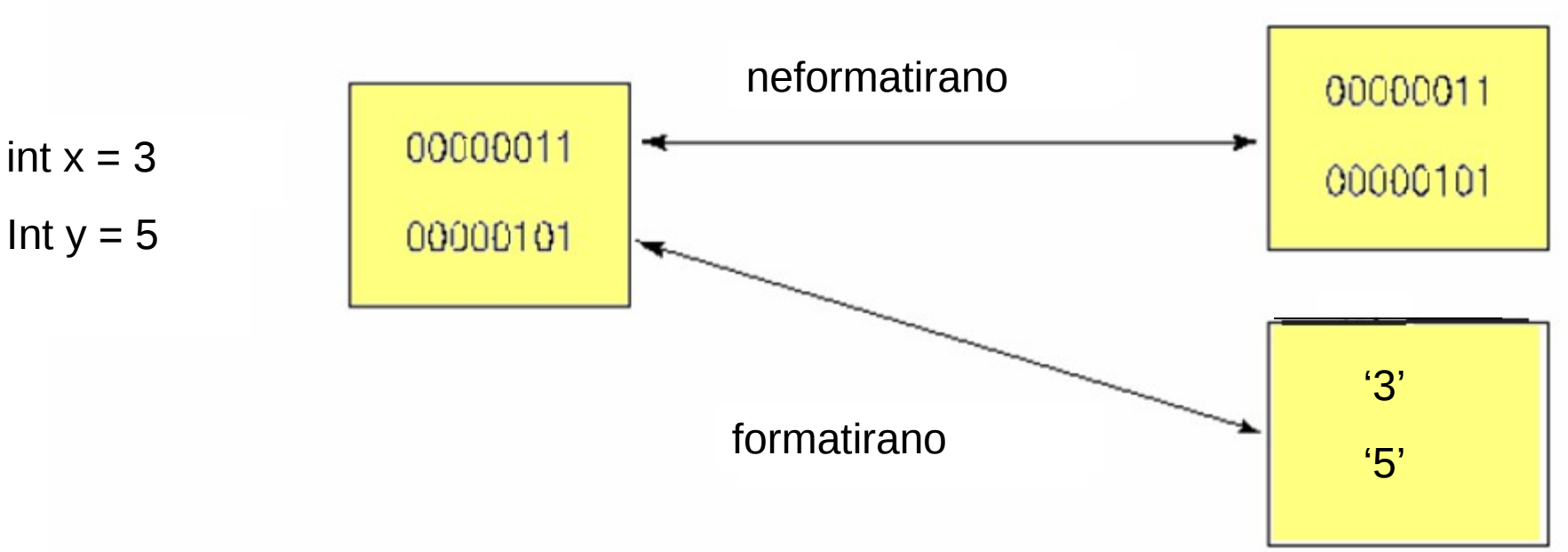
char buffer [BUFSIZ] setbuf (fp, bufer) Sami zagotovimo polje za medpomnilnik.BUFSIZ je definiran v stdio.h .

Če namesto naslova polja buffer navedemo (char *) NULL, bo do operacije brez medpomnilnika (unbuffered)

Formatiran, neformatiran vhod-izhod

Formatiran izhod
- •Podatki pretvorjeni v ASCII neodvisno od vhodnega formata

Neformatiran izhod
- •Zapis na disk brez konverzije (koda takšna, kot je)

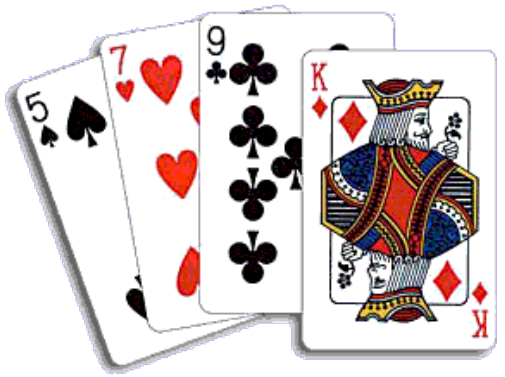


ASCII koda:

‘3’ = 00110011
‘5’ = 00110101

Neformatirano pisanje

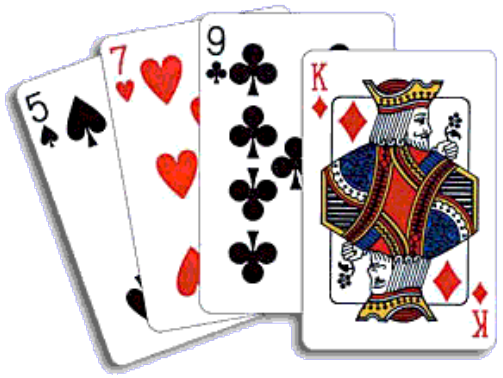
int fwrite(void *ptr, int size, int num_items, FILE *fp);



```
#include <stdio.h>
struct karta {
    int vrednost;
    char barva;
};

int main(void) {
    struct karta paket[ ] = { {2,'s'}, {7,‘k'}, {8,'c'}, {2,'s'}, {2,'h'} };

    FILE *fp= fopen("unformatted", "wb");
    printf("%d", sizeof(struct karta) );
    fwrite(paket, sizeof(struct karta), 5, fp);
    fwrite(paket, 1, sizeof(struct karta)*5, fp);
    fclose(fp);
}
```

Neformatirano branje

int fread (void *ptr, int size, int num_items, FILE *fp);



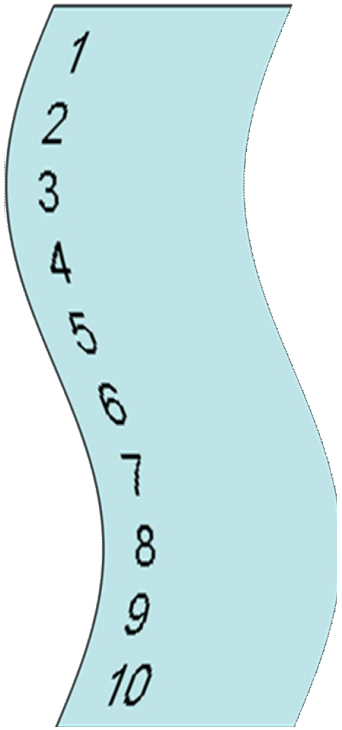
```
#include <stdio.h>
struct karta {
    int vrednost;
    char barva;
};

Int main(void){
    struct karta paket[10];
    FILE *fp= fopen("unformated", "rb");
    fread(paket, sizeof(struct karta), 10, fp);
    fclose(fp);
    printf(“vrednost= %d, barva = %c\n", paket[6].vrednost, paket[6].barva );
}
```

Formatirano pisanje datoteke

Zapis 10 števil v datoteko

```
#include <stdio.h>
#define MAX 10

int main() {
    FILE *f;
    int x;
    f=fopen(“stevila.txt","w");
    if (!f) return 1;
    for (x=1; x<=MAX; x++) fprintf (f,"%d\n",x);
    fclose(f);
    return 0;
}
```

stevila.txt



Formatirano branje datoteke

Branje vrstic v datoteki in izpis na zaslon

```
#include <stdio.h>

int main() {
    FILE *f;
    char s[1000];
    f=fopen(“pesmica.txt","r");
    if (!f) return 1;
    while (fgets(s,1000,f)!=NULL) printf("%s",s);
    fclose(f);
    return 0;
}
```

pesmica.txt

Standardna knjižnjica C

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.
int strcpy (s2, s1) Kopira niz s1 v niz s2.
int strncpy(s2, s1, n) Kopira niz s1 v niz s2, vendar največ n znakov.
char *strcmp (s2, s1) Primerja niza in vrne:
pozitivno vrednost če je s2>s1,
0..če je s2=s1
negativno vrednost sicer

char *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

Konverzija podatkov

Pri pregledu naslednjih funkcij za konverzijo podatkov imejmo definirano:

```
#include < stdio.h >
int base;
char *s, *end;
```

int atoi (s) Pretvorba števila v nizu s v format int.
long atol (s) Pretvorba števila v nizu s v format long int
double atof (s) Pretvorba števila v nizu s v format double
double strtod (s,end) Pretvorba števila v nizu s, pisanem v znanstveni notaciji. Funkcija nastavi kazalec end na znak, ki je zaključil pretvorbo

long strtol (s, end, base)
unsigned long strtoul (s, end, base)

Konverzija niza s v long oziroma unsigned long. Pri tem je base uporabljena osnova (mora biti med 2 in 36). Vodeče ničle so ignorirane.

Funkcije z znaki

*So deklarirane v datoteki **ctype.h**. Zato imejmo:*

```
#include < ctype.h >
int c;
```

Na voljo imamo naslednje funkcije:

```
int isalpha (c)
int isupper (c)
int islower(c)
int isalnum (c)
int isdigit (c)
int isprint (c)
int iscntrl (c)
int ispunct (c)
int isspace (c)
int toupper (c)
int tolower (c)
int toascii (c)
```

Primer: konverzija črk niza v velike črke

```
i = 0;
while (s[i] != 0) {
    if ( isalpha( s[i] ))  s[i]= toupper(s[i++ ]);
}

/* ali pa z uporabo kazalca... */
while (*s != 0) if ( isalpha(*s)) {
     *s = toupper(*s); s++;
}
```

Matematične funkcije

So deklarirane v datoteki **math.h**. Zato imejmo:

```
#include < math.h >
double x, y , *pd ;
long k;
int *pi , i;
```

Na voljo imamo:

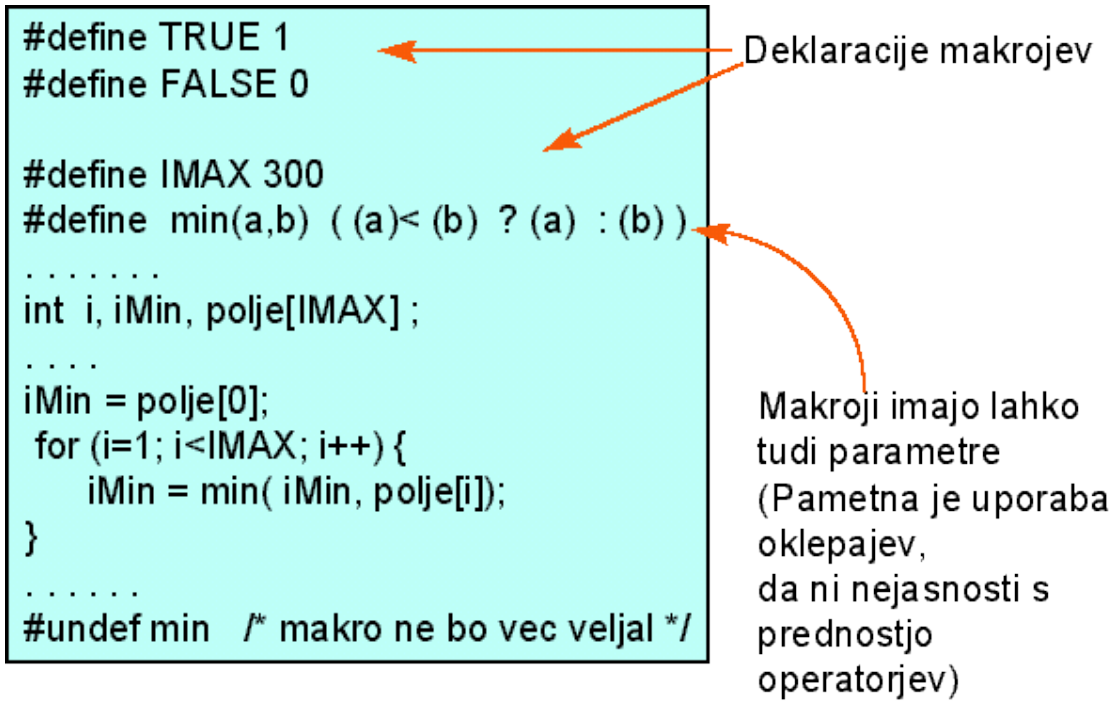
int abs (i)
ilong labs (k)
double fabs (x)
double fmod (x, y) *Vrne ostanek deljenja x / y*
double modf (x, pd)
double ldexp (x, i) *Vrne (x* (2 na i))*
double frexp (x, pi)
double floor (x) *Vrne največji integer, ki se gre v x*
double ceil (x) *Vrne najmanjši integer,ki ni manjši od x*
double sqrt (x)
double pow (x, y)

double sin (x) , double cos(x) , double tan(x), double asin (x), double acos (x)
double atan (x) .
double atan2 (x) *Vrne arctangens y/x. Uporabi predznaka argumentov za določitev kvadranta*
double exp (x) double log (x) double log10 (x)

C-jev predprocesor

Pred samim prevajanjem pregleda program v jeziku C predprocesor. Ta spozna navodila (direktive), za katera je značilno. da se začnejo z znakom #. Predprocesorju običajno povemo, katere datoteke naj vljuči v nas program (#include...), Deklariramo makroje (#define..), ki jih nato v našem programu razširi. Lahko tudi pogojimo, katere dele našega programa naj vključi v prevedeno kodo in katere ne.

Definicija in uporaba makrojev – primer:



Pogojno prevajanje

Primer

```
. . . . .
#ifdef min
   minimum = min(a,b);
#else
   if(a <b) minimum = a;
   else minimum = b;
#endif
. . . . . .
```

Direktivi sledi skupina stavkov (v novi vrsti)

Pregled direktiv za pogojno prevajanje

#if (kostantni izraz)

#if . . . defined (simbol)

#ifdef simbol

#ifndef simbol

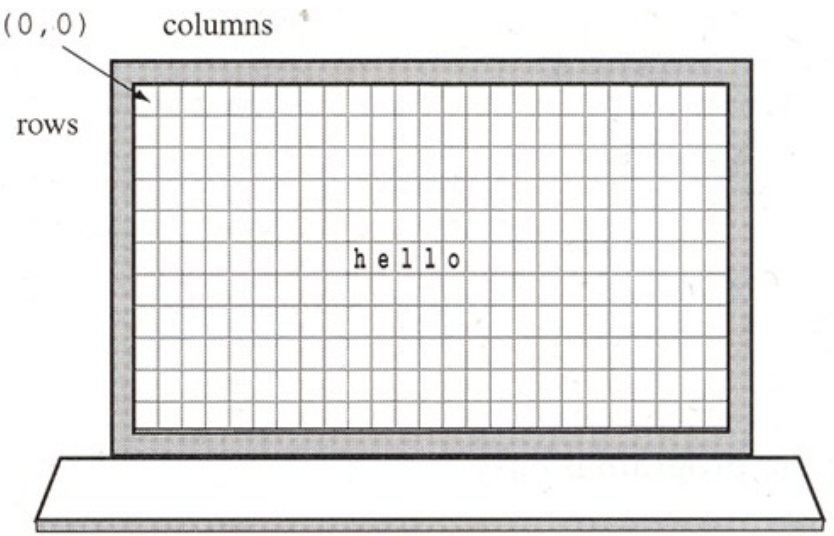
#else

#elif simbol

#endif

Krmiljenje zaslona

Pri nekaterih programih želimo "zaslonsko usmerjeno" zapisovanje in branje. V primeru operacijskega sistema LINUX si pomagamo s knjižnico funkcij z imenom "curses". Struktura programa, ki uporablja curses




```
#include < curses.h >
. . . . .
initscr();  /* iniciacija zaslona */
cbreak();   /* razlicne nastavitve */
nonl(); noecho(); . . . . .
while (!done) {
      /* nekaj primerov klicov za zapis na zaslon */
      move(row, col);
       addch(ch);
       printw (" Vrednost = %d \n ", vrednost);     . . . .
       refresh( );   /* azuriranje zaslona */     . . . .
}
endwin( );  /* pred izstopom pocisti za sabo */ exit (0);
```

Prevod in povezovanje programa: gcc <program file> -lncurses

Boljši primer

```
#include <ncurses.h>

int main() {
   int ch;

    initscr();                          /* Start curses mode                    */
    raw();                                      /* Line buffering disabled  */
    keypad(stdscr, TRUE);               /* We get F1, F2 etc..                  */
    noecho();                           /* Don't echo() while we do getch */

     printw("Type any character to see it in bold\n");
     ch = getch();                                  /* If raw() hadn't been called
                                                     * we have to press enter before it
                                                     * gets to the program                  */
     if(ch == KEY_F(1))                 /* Without keypad enabled this will */
          printw("F1 Key pressed");/*  not get to us either */
                                                    /* Without noecho() some ugly escape
                                                     * charachters might have been printed
                                                     * on screen                            */
     else {
         printw("The pressed key is ");
         attron(A_BOLD);
         printw("%c", ch);
         attroff(A_BOLD);
     }
     refresh();                         /* Print it on to the real screen */
     getch();                           /* Wait for user input */
     endwin();                          /* End curses mode                      */

     return 0;
}
```

Pisanje obsežnih programov

Zaglavne datoteke (header files)
Zunanje spremenljivke in funkcije
Prednosti uporabe več datotek
Kako razbijemo program na več datotek
Organizacija podatkov v vsaki datoteki
Orodje Make
Napravimo datoteko makefile
Makroji Make
Kompleten primer datoteke Makefile

Pisanje bolj obsežnih programov terja delitev programov na module oziroma ločene datoteke. Zato bo funkcija main( ) v eni datoteki, druge funkcije pa verjetno v kateri drugi.

Tvorimo lahko knjižnico funkcij, ki so lahko skupinsko pomnjene v takih modulih. To omogoča uporabo pripravljenih modulov v različnih programih, v katere jih dodamo v času samega prevajanja.

Kaj pa grafične aplikacije

Ni grafičnega vmesnika (API)

C so razvili precej, preden so se pojavili različni grafični vmesniki oziroma grafično podprti programi. Tako nimamo enotnega grafičnega programskega vmesnika (API, Application Programming Interface), kot je to sicer značilnoza Javo). Obstajajo sicer grafični programski vmesniki za operacijske sisteme Unix (X-Windows) in za MSWindows, ki pa se med seboj zelo razlikujejo.

Prednosti uporabe več datotek

Program lahko razvija več programerjev. Vsak dela svojo datoteko. Uporabimo lahko objektno usmerjen pristop. Posamezne datoteke lahko predstavljajo posamezne objekte in vsebujejo ustrezne podatke in operacije nad njimi. Vzdrževanje tako strukturiranih programov je lažje.

Datoteke lahko vsebujejo funkcije določene skupine, na primer vse matrične operacije. Imamo tako lahko knjižnice funkcij.

Dobro zasnovane funkcije lahko uporabimo v drugih programih in tako skrajšamo čas razvoja.

Pri zelo obsežnih programih morda celotno datoteko zaseda ena pomembna funkcija, ob njej pa je še več nizkonivojskih funkcij.

Če spremenimo neko datoteko, je potrebno za obnovitev izvedljivega programa ponovno prevajanje le te datoteke. Na sistemih Linux si pri tem lahko pomagamo z orodjem make.

Zunanje definicije

```
#include <stdio.h>

extern char user2line [20];   /* globalna sprem., definirana v drugi datoteki */
char user1line[30];           /* globalna v tej datoteki*/
void dummy(void);

void main(void) {
 char user1line[20];       /* različna od prejšnje user1line[30] */
 . . .                     /* omejena le na to funkcijo */
}

void dummy(){
 extern char user1line[ ];  /* globalna user1line[30] */
 . . .
}
```

Zaglavne datoteke (header files)

Pri uvedbi takega, modularnega programiranja želimo obdržati definicije spremenljivk, prototipe funkcij ipd. znotraj posameznega modula. Kaj pa,če take definicije souporablja več modulov? Bolje je, če centraliziramo take definicije v eni datoteki in souporabljamo to datoteko v drugih modulih. Taki datoteki pravimo zaglavna datoteka (header file) in ima podaljšek .h

Standardne zaglavne datoteke upoštevamo v našem programu na naslednji znan način:

```
#include <stdio.h>
```

Lahko pa napišemo tudi lastne zaglavne datoteke, ki jih v našem programu upoštevamo tako:

```
#include "mojaDatoteka.h"
```

Primer

V programu main.c imamo funkcijo main( ), ki kliče funkcijo WriteMyString( ), ta pa se nahaja v modulu WriteMyString.c Prototip te funkcije pa se nahaja v zaglavni datoteki Header.h

Datoteka main.c

```
#include "header.h"
#include <stdio.h>
char *drugNiz = "Pozdravljeni";
main(){
    WriteMyString(MOJ_NIZ); /* klic funkcije iz drugega modula */
}
```

Datoteka *WriteMyString.c*

```
extern char *drugNiz; void WriteMyString(char *taNiz) {
        printf("%s\n", taNiz);
        printf("Globalna spremenljivka = %s\n", drugNiz);
}
```

Datoteka header.h

```
#define MOJ_NIZ "Dober dan" void WriteMyString();
```

Kak razdelimo program na več datotek

Programerji običajno začnejo snovanje programa tako, da razbijejo problem na več sekcij. Vsako od teh sekcij lahko realiziramo z eno ali več funkcijami. Vse funkcije iste sekcije pomnimo v isti datoteki.

Podobno velja, če implementiramo objekte kot podatkovne strukture. Vse funkcije, ki delajo s tako strukturo, pomnimo v isti datoteki. Kasnejša sprememba "objekta" terja spremembo le v njemu ustrezni datoteki.

Najbolje je,če za vsako od teh izvornih datotek (s podaljškom .c) napišemo še ustrazno zaglavno datoteko (z enakim imenom, toda s podaljškom .h). Zaglavna datoteka vsebuje definicije vseh funkcij, ki jih "njena" izvorna datoteka uporablja.

Druge datoteke,ki uporabljajo te funkcije morajo na začetku imeti ustrezen stavek #include z navedbo datoteke .h

Organizacija podatkov v datotekah

Tipičen vrstni red je naslednji:

Definicija vseh konstant s stavki #define,
Definicija vseh potrebnih zaglavnih datotek s stavki #include
Definicija vseh važnejših podatkovnih tipov s stavki typedef
Deklaracija globalnih in eksternih spremenljivk. Globalne spremenljivke lahko sočasno tudi inicializiramo.
Ena ali več funkcij.

Kako naj ne programiramo v C

```
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,
/#\ ;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \ q#'+d'K#!/+k#;q#'r}
eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw'
i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \ iwk{KK{nl]!/w{%'l##w#' i; :
{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \ ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}
{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1); }
```

In rezultat tega

On the first day of Christmas my true love gaaaaave to me
a partridge in a peaaaaar tree.

On the second day of Christmas my true love gaaaaave to me
two turtle doves
and a partridge in a peaaaaar tree.

On the third day of Christmas my true love gaaaaave to me
three french hens, two turtle doves
and a partridge in a peaaaaar tree.

On the fourth day of Christmas my true love gaaaaave to me
four caaaaalling birds, three french hens, two turtle doves
and a partridge in a peaaaaar tree.

On the fifth day of Christmas my true love gaaaaave to me
five gold rings;
four caaaaalling birds, three french hens, two turtle doves
and a partridge in a peaaaaar tree.

..... Se nadaljuje do dvanajstega dne

Še nekaj napotkov….

- Vedno vse inicializirajte pred uporabo (posebno kazalce)
- Ne uporabljajte kazalcev, ko jih sprostite
- Ne vračajmo lokalnih spremenljivk neke funkcije po referenci
- Ni izjem– torej sami preverjajte možnost napak
- Polje je vedno kazalec, katerega vrednosti pa ne moremo spreminjati (je naslov).

- Pa še kaj bi se našlo.

C v primerjavi z Javo

- Java je objektno usmerjena, C je funkcijsko usmerjen
- Java se strogo drži podatkovnih tipov, C is je pri tem površen
- Java podpira polimorfizem (na primer. + ==), C nima polimorfizma
- Javanski razredi (classes) pomagajo pri nadzoru imenskega prostora (name space), C ima en sam imenski prostor
- Spremenljivke C lahko definiramo le na začetku bloka
- Programe C prevajalnik pred-procesira, za Javo to ne velja
- Java ima kompleksen večplastni vhodno-izhodni model, C ima le preprost vhodni oziroma izhodni tok bajtov
- Java ima avtomatsko upravljanje s pomnilnikomt, pri C moramo za to sami skrbeti
- Java nima eksplicitnih kazalcev, C uporablja kazalce pogosto
- Java has by-reference and by-value params , C has by-value only
- Java has exceptions and exception handling, C has signals
- Java has concurrency primitives, C concurrency is via library functions
- C arrays have no associated length attribute (remember it yourself)
- Nizi v Javi (Strings) so dobro definirani objekti, nizi v C so polja char[]