Programiranje za elektrotehnike 1

Programiranje v programskem jeziku C

Vsebina poglavja

- Spremenljivke
- Konstante
- Pretvorbe med tipi podatkov
- Izpeljani tipi
- Podprogrami
- Lokalne in globalne spremenljivke
- Formatirano branje in izpis
- Ukazi prevajalniku – makro ukazi
- Dinamično dodeljevanje pomnilnika
- Delo z datotekami
- Programiranje vhodno/izhodnih vmesnikov
- Bitne operacije, maskiranje

Spremenljivke

- Spremenljivka je definirana:
  - ime spremenljivke
    - lahko ima poljubno število znakov
    - prevajalnik loči prvih x znakov
  - tip podatka (celo, realno, znak,...)
  - način hranjenja spremenljivke (kje in koliko časa)

Tipi spremenljivk

- lokalne
- globalne
- registrske
- konstante
- vektorji in polja
- kazalci

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}

Pomoč prevajalniku

- Pri deklaraciji spremenljivk lahko pomagamo prevajalniku (optimizatorju), tako, da mu napovemo, ali bo neka spremenljivka imela stalno ali spremenljivo vrednost:
  - Konstanta – spremenljivka ima konstantno vrednost:
    - const double e= 2.718281828;
  - Spremenljiva vrednost:
    - volatile char answer;
    - Ta tip se pogosto uporablja pri delu s prekinitvami!

Konstante

Konstanta	Tip	Pomen
123	int	decimalno število
033	int	osmiško število
0X1F	int	šestnajstiško število
112364L	long	dolgo celo število
63557u	unsigned	brez predznaka
3.14	double	dvojno realno število
1.22e-4	double	znanstvena notacija
3.14f	float	navadno realno število
‘A’	char	znak
‘\07’	char	znak z osmiško kodo 7
‘\n’	char	skok v novo vrsto (newline)
‘\t’	char	tabulator
‘\b’	char	backspace
‘\\’	char	backslash
“besedilo”	char[]	niz

Pretvorbe med tipi podatkov

- Implicitne pretvorbe - avtomatične:
  - V programu se pretvorbe med tipi izvajajo samodejno.

```
int a;
float b;
a=b;
```

  - Obstajajo pravila!
- Eksplicitne pretvorbe – zahtevane:
  - V programu določimo način pretvorbe (casting):

```
int a;
float b;
b = (float)a;
```

Implicitne pretvorbe - avtomatične

- Konverzija tipa izraza po standardu ANSI C:
  - 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.
- Hierarhija (manjši tipi se pretvorijo v večje):

  int < unsigned int < long < unsigned long < float < double < long double

Eksplicitne (zahtevane) pretvorbe

- Konverzija tipa je navedena v programu:
  - Uporabimo operator za pretvorbe (cast).
  - Primer:

```
#include <stdio.h>

int main(void)
{
int a;
float f,g;
    a=3; f=3.2; g=a+(int)f;
    printf("%f\n",g);
}
```

  - Rezultat je 6.

Oštevilčeni tipi (enum)

- Splošni zapis:
  - 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,..)

Izpeljani tipi

- *funkcije*, ki vrnejo objekt določenega tipa.
- *kazalci* na objekte nekega tipa,
- *polja* objektov nekega tipa,
- *strukture*, ki vsebujejo spisek objektov,
- *unije*, ki vsebujejo enega od objektov v spisku,

Podprogrami

- Razlogi za uporabo
- Lastnosti
- Modularnost
- Klic podprograma
- Vračanje iz podprograma

Razlogi za uporabo podprogramov

- Prednosti:
  - Večja preglednost kode.
  - Možnost večkratne uporabe delov programske kode.
    - Manjša dolžina kode – manjša poraba pomnilnika.
    - Krajši porabljen čas za pisanje kode.
- Slabosti:
  - Počasnejše izvajanje.
    - Sklad,
    - Trajanje skoka.
  - V primeru spremembe je treba upoštevati vse klice!

Modularnost

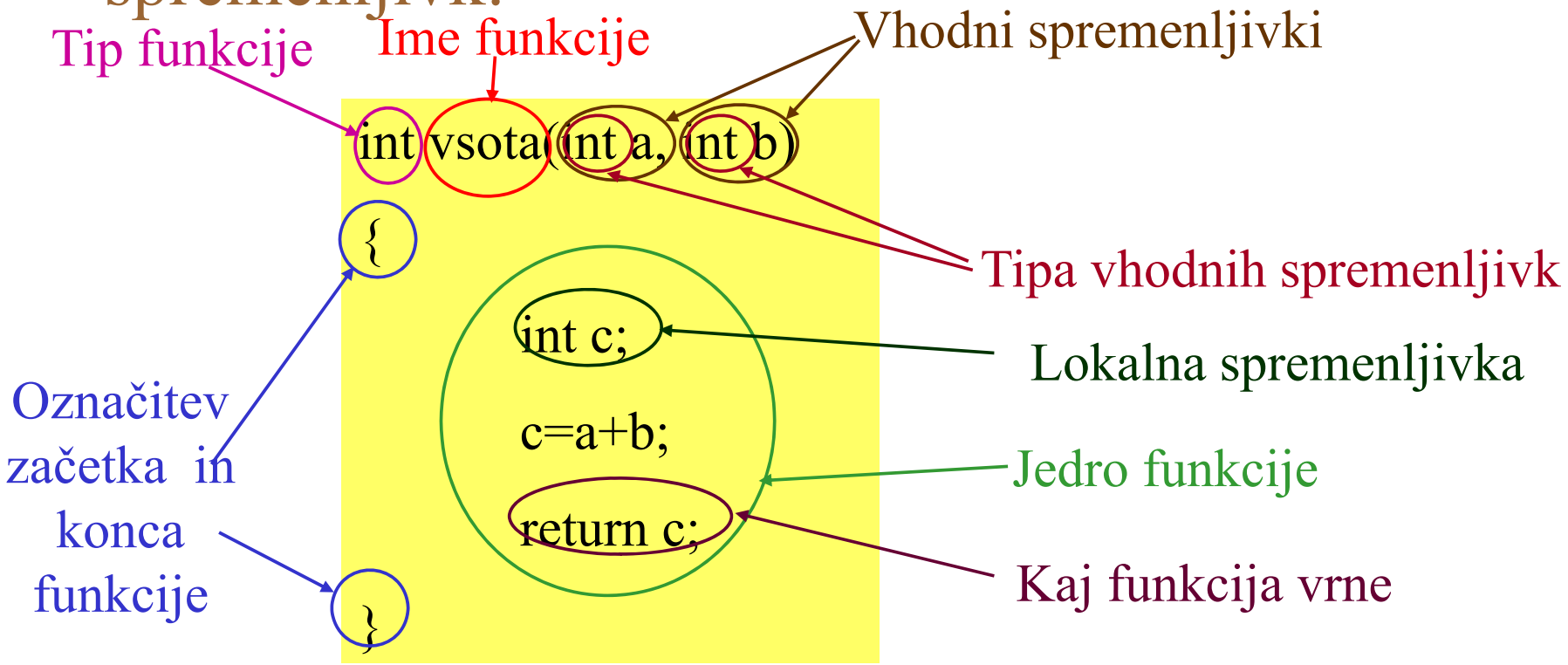
- Podprogram je zaključena celota:
  - Vhodni podatki,
  - Funkcija,
  - Izhodni podatki
- Čim manj globalnih spremenljivk – idealno nobene!
- Statične spremenljivke le v primeru potrebe.
- Najboljše so lokalne spremenljivke.

Funkcije

- Funkcije sestavljajo:
  - Ime funkcije
  - Tip funkcije oz. vrednosti, ki jih funkcija vrne
  - Vrednost, ki jo funkcija vrne
    - Funkcija v C vrne eno spremenljivko, če želimo da vrača več vrednosti, jih moramo vračati preko kazalcev oz. globalnih spremenljivk.
  - Jedro
- Tudi *main()* je funkcija!
  - V vsakem programu imamo samo eno funkcijo *main()*.

Funkcija – struktura

- Primer – funkcija vrne vsoto dveh celoštevilčnih spremenljivk:



Funkcija tipa void

- Primer – funkcija ne vrne ničesar:
  - Izpis dveh celih števil:

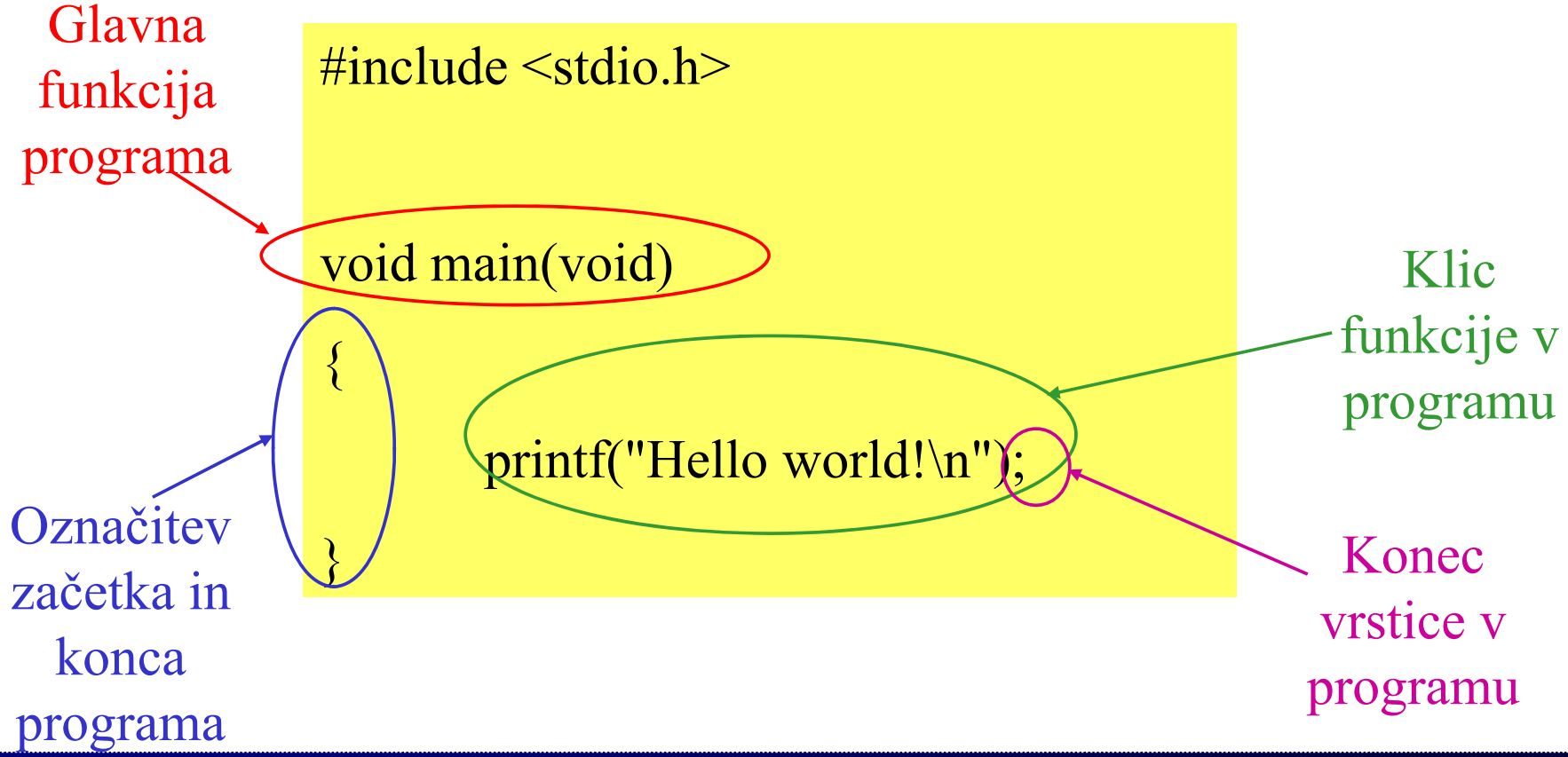
Tip funkcije →

```
void izpis(int a, int b)
{
        printf("a=%d, b=%d\n",a,b);
}
```

- Za takšne funkcije uporabljamo tip **void**!

Funkcija *main()*

- "Hello world!" - ponovitev:



Uporaba funkcij v programu

- Imamo dve možnosti:
  - Deklaracija funkcije je hkrati tudi definicija:
    - Vrstni red zapisa:
      - zapis naše funkcije,
      - zapis funkcije, ki uporabi našo funkcijo.
  - Deklaracija in definicija sta ločeni:
    - Vrstni red zapisa:
      - deklaracija naše funkcije,
      - zapis funkcije, ki uporabi našo funkcijo,
      - zapis naše funkcije.
  - Preddefinirane funkcije
    - So že napisane (deklaracije so v zaglavjih):
      - deklaracija funkcije,
      - zapis funkcije, ki jo uporabi.

Deklaracija funkcije je hkrati tudi definicija

```
#include <stdio.h>

void izpis(int a, int b)                    // definicija funkcije
{
        printf("a=%d, b=%d\n",a,b);
}

int main()
{
        int e=3;
        int f=4;
        izpis(e,f);                         // klic (uporaba) funkcije
}
```

Deklaracija in definicija sta ločeni

```
#include <stdio.h>

void izpis(int a, int b);                    // deklaracija funkcije

int main()
{
        int e=3;
        int f=4;
        izpis(e,f);                          // klic (uporaba) funkcije
}

void izpis(int a, int b)                     // definicija funkcije
{
        printf("a=%d, b=%d\n",a,b);
}
```

Funkcija je lahko definirana tudi v drugi datoteki!

Preddefinirane funkcije

- Funkcija je lahko definirana tudi v drugi datoteki!

```
#include <stdio.h>                          // zaglavje, v katerem je deklaracija funkcije

int main()
{
        int e=3;
        int f=4;
        printf("a=%d, b=%d\n",a,b);         // klic (uporaba) funkcije
}
```

- Funkcija *printf* je že izdelana, uporabimo *zaglavje stdio.h*, ki ga vključimo z ukazom prevajalniku *#include*
- Zaglavja si lahko izdelamo tudi sami!

Nekaj najpomembnejših zaglavij

- stdio.h
  - standardni vhod in izhod (printf, ...)
- math.h
  - Matematične funkcije (sin, cos, ...)
- stdlib.h
  - Standardne funkcije (abs, exit, system, ...)
- string.h
  - Delo z nizi znakov (strcmp, ...)
- ...

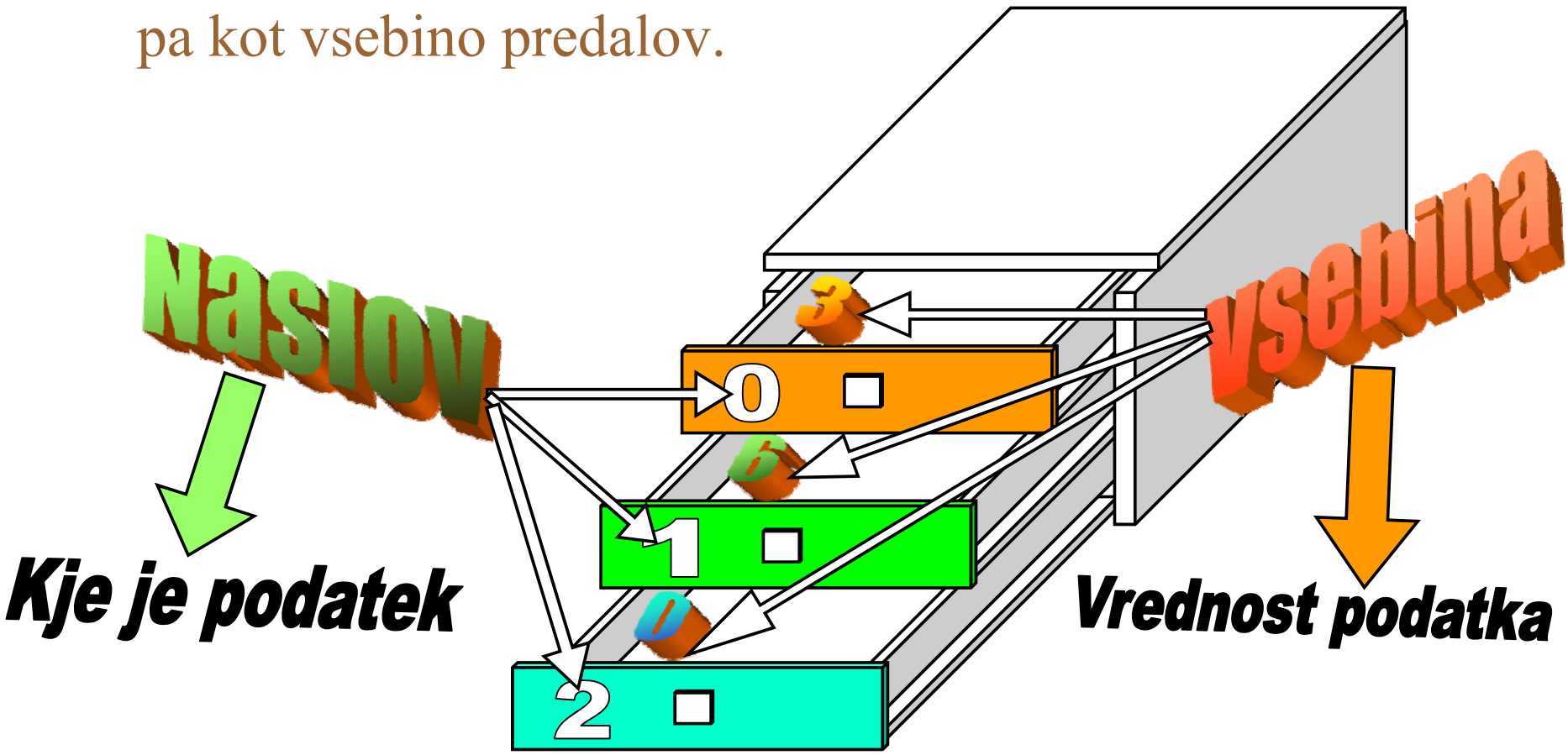
Angleško ime - predefined headers

Kazalci

- Kazalci so spremenljivke, ki vsebujejo naslov neke druge spremenljivke ali dela pomnilnika.
- Podobno kot v zvezi z običajnimi spremenljivkami tudi za kazalec poznamo:
  - ime,
  - tip (ki pove, na kakšne vrste podatkov kaže) in
  - način hranjenja.

Pomnilnik

- Predstavljamo si ga lahko kot predalnik, podatke v njem pa kot vsebino predalov.

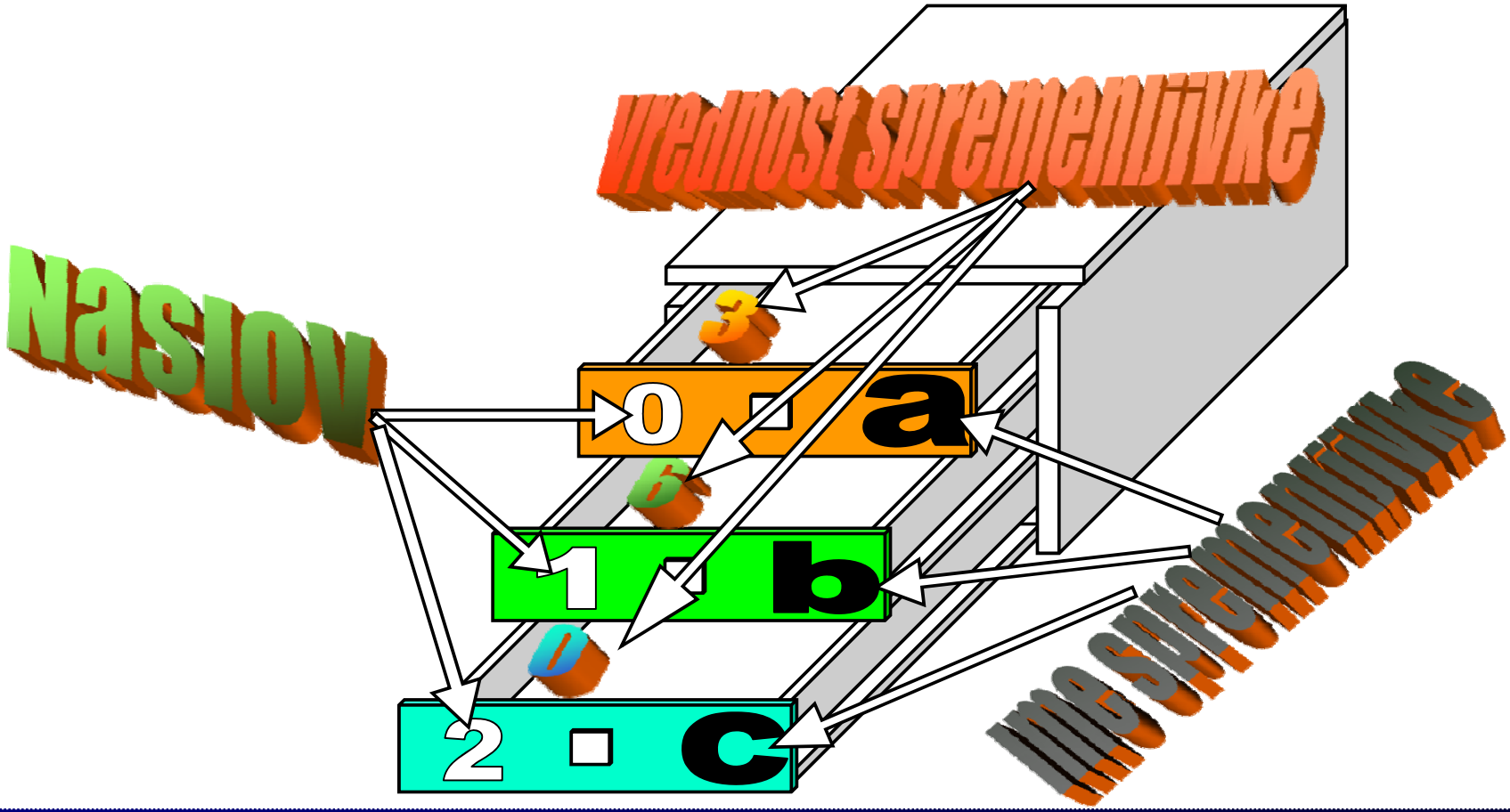


Pomnilnik (2)

- V resnici je boljši prikaz:

Podatki v pomnilniku

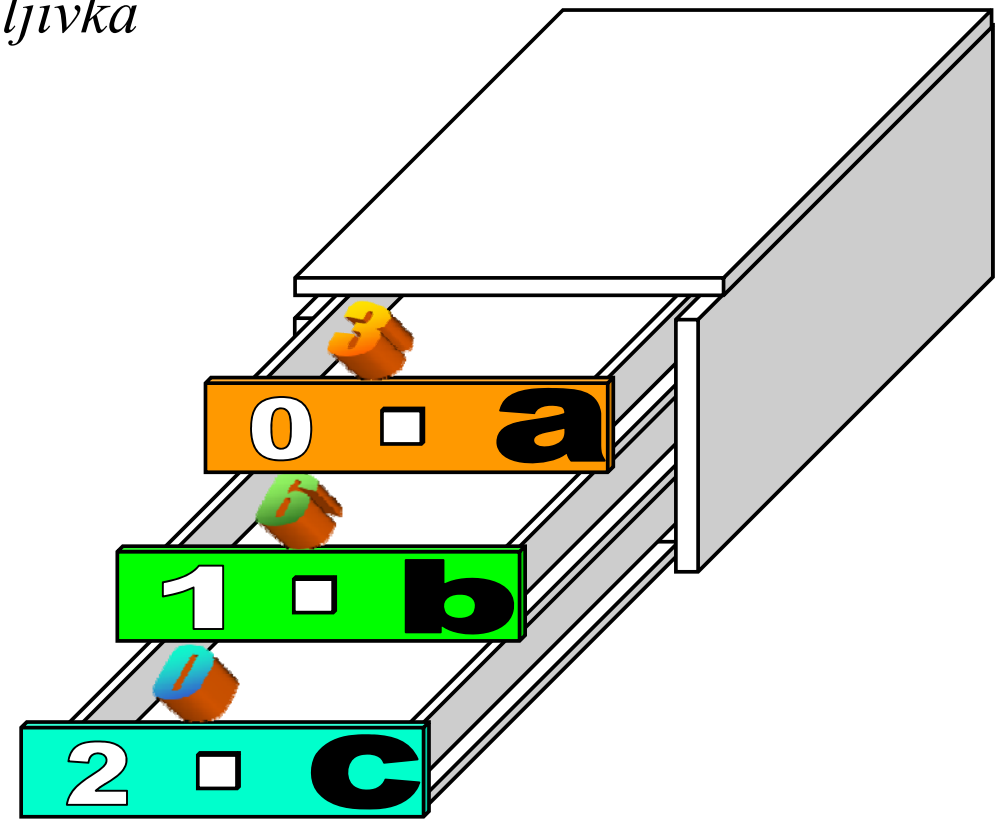
- Podatke v pomnilniku shranjujemo kot spremenljivke.



Kazalci – zapis

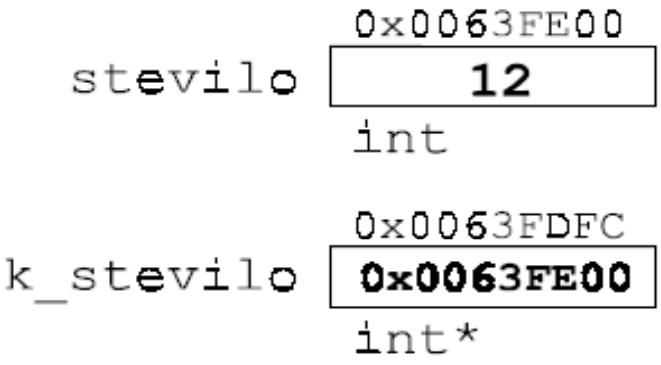
int a = 3; *// celostevilcna spremenljivka*
int *c; *// kazalec na int*
c = &a; *// kazalec dobi vrednost*

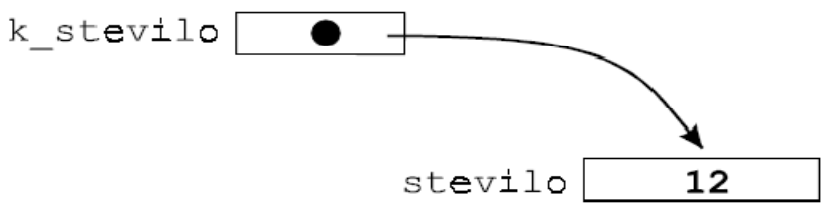
- Vrednost spremenljivke a:
  - a = 3
- Naslov spremenljivke a:
  - &a = 0
- Vrednost spremenljivke c:
  - c=0
- Kazalec na spremenljivko a:
  - *c=3
- Naslov spremenljivke c:
  - &c=2



Kazalci – realni primer zapisa

int stevilo = 12; *// celostevilcna spremenljivka*
int *k_stevilo; *// kazalec na int*
k_stevilo = &stevilo; *// kazalec dobi vrednost*





Primer uporabe kazalcev (1)

- Zamenjava vednosti dveh spremenljivk z uporabo funkcije.
- Izvedba zamenjave vrednosti je preprosta:

```
...
zacas = a;      // shranimo vrednost a
a = b;          // prepišemo vrednost a z vrednostjo b
b = zacas;      // vrednost b prepišemo z vrednostjo
                // začasne spremenljivke
...
```

Primer uporabe kazalcev (2)

- Kako pa bi to izvedli s funkcijo?

```
...
zamenjaj(a,b); // primer, ki ne deluje
...
```

- Uporabimo kazalce!

```
...
zamenjaj(&a, &b);      // zamenjamo vsebino
                       // pomnilniških lokacij a in b
...
```

Primer uporabe kazalcev (3)

- Uporabljena funkcija:

```
void zamenjaj(int *s1, int *s2)
/* delamo s kazalci, saj lahko funkcija vrne le eno */
/* spremenljivko */
{
int zacas;                  /* začasna spremenljivka */
    zacas=*s1;              /* delamo s kazalci - */
    *s1=*s2;
    *s2=zacas;
}
/* ukaza return ne potrebujemo, ker je funkcija tipa */
/* void in ne vrne ničesar */
```

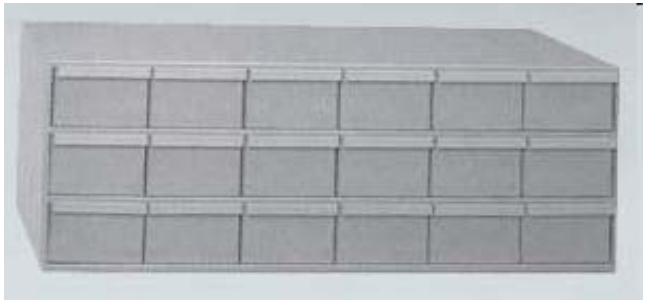
Vektorji in polja

- Vektorji:
  - Enodimenzionalne matrike
  - Primer:

$$v = [1, 5, 7, 9];$$

- Polja:
  - Dvo- ali večdimenzionalne matrike
  - Primer:

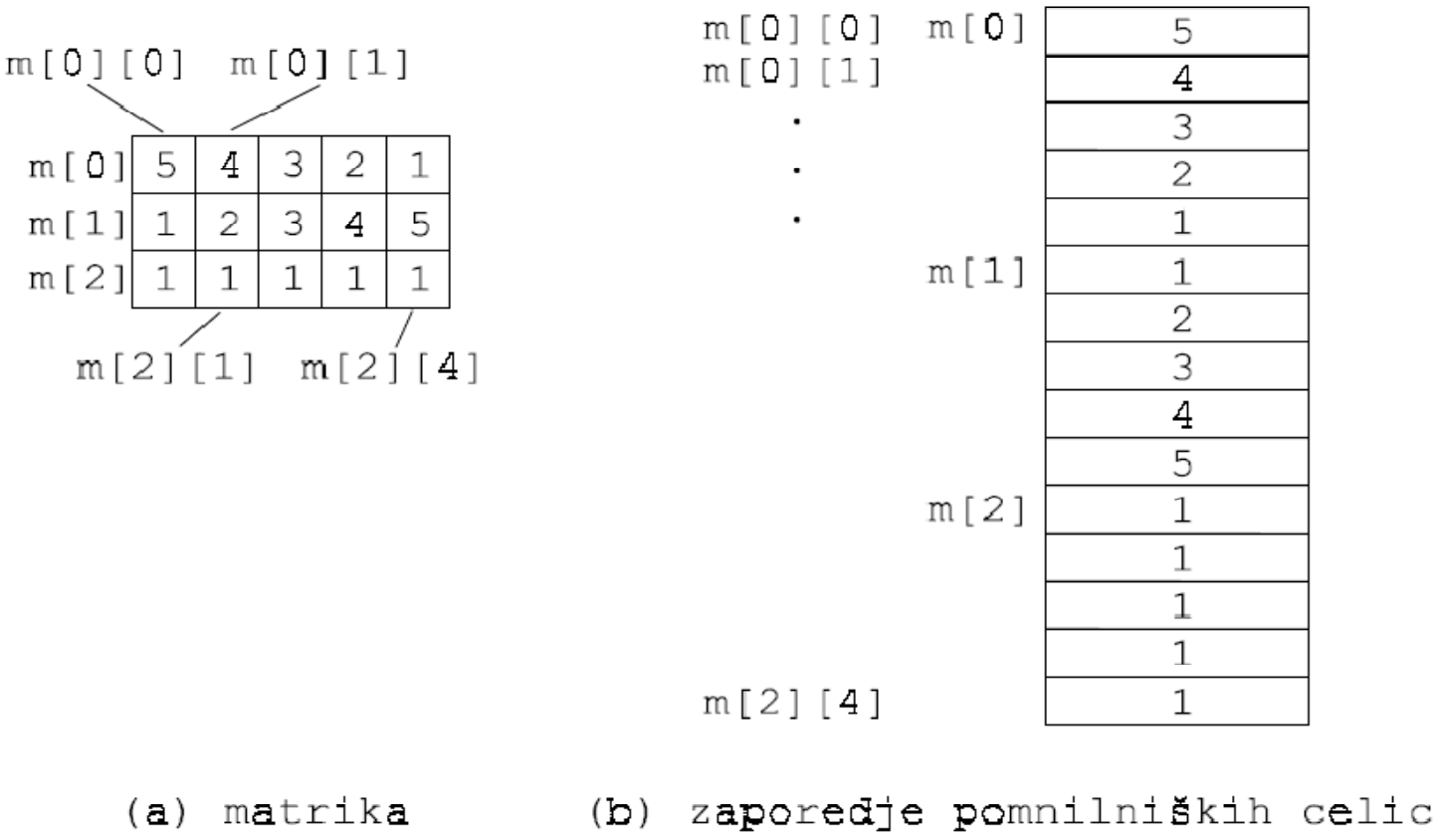
$$M = \begin{bmatrix} 5 & 6 & 88 & 4 \\ 7 & 54 & 0 & 1 \\ 6 & 102 & 1 & 4 \end{bmatrix}$$



- Vsi elementi vektorjev in polj so istega tipa!

Dvodimenzionalno polje

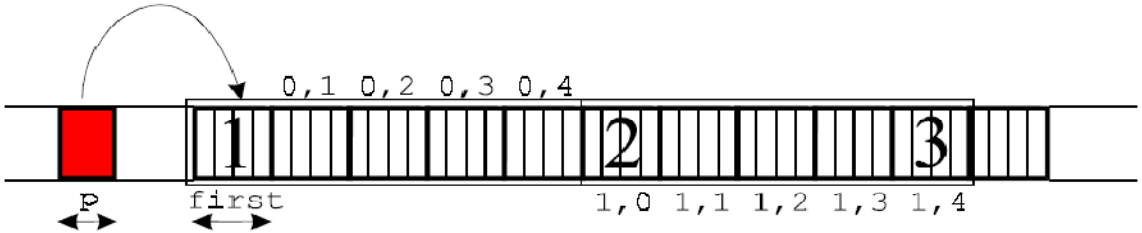
- Polje je v C izvedeno kot vektor več vektorjev.



(a) matrika (b) zaporedje pomnilniških celic

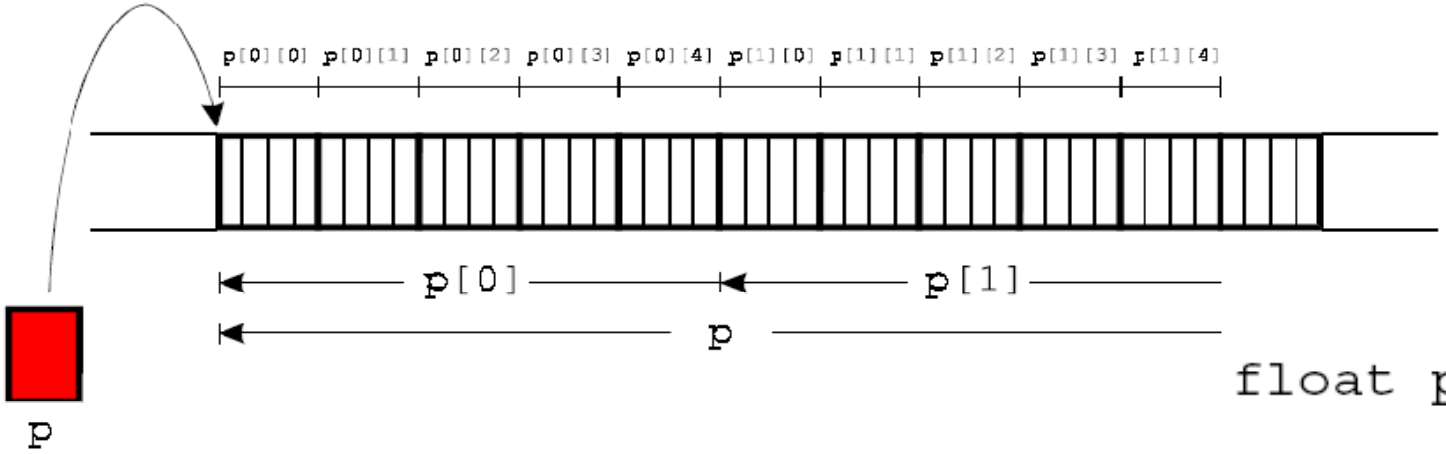
Vektorji in polja - zapis

- Vektorji:




```
float p[10];
p[0] = 1;
p[1] = 2;
p[2] = 3;
```

- Polja:




```
float p[2][5];
```

Vektorji in polja – primer

- Množimo matriko z vektorjem:

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

int main()
{
    int M[2][2]={{1,5},{2,3}};          // matrika – določimo vrednosti
    int v[2]={1,2};                     // vektor – določimo vrednosti
    int r[2];                           // rezultat

    r[0]=M[0][0]*v[0]+M[0][1]*v[1];
    r[1]=M[1][0]*v[0]+M[1][1]*v[1];
    printf("r[0]=%d, r[1]=%d\n",r[0],r[1]);

    return 0;
}
```

$$M = \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$r = M * v = \begin{bmatrix} 11 \\ 8 \end{bmatrix}$$

Vektorji in polja – primer

- Množimo matriko z vektorjem – uporaba funkcije:

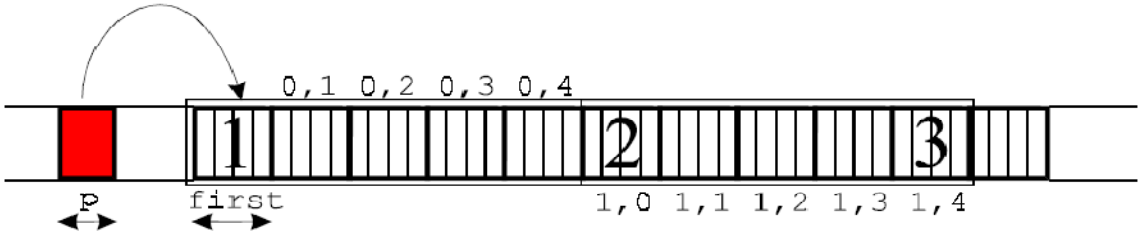
```
#include <stdio.h>

void mnozi(int Mat[2][2],int vect[2],int rez_vect[2])
{
   rez_vect[0]=Mat[0][0]*vect[0]+Mat[0][1]*vect[1];
   rez_vect[1]=Mat[1][0]*vect[0]+Mat[1][1]*vect[1];
}

int main()
{
      int v[2]={1,2};
      int r[2];
      int M[2][2]={{1,5},{2,3}};
      mnozi(M,v,r);
      printf("r[0]=%d, r[1]=%d\n",r[0],r[1]);
      return 0;
}
```

$$M = \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$r = M * v = \begin{bmatrix} 11 \\ 8 \end{bmatrix}$$

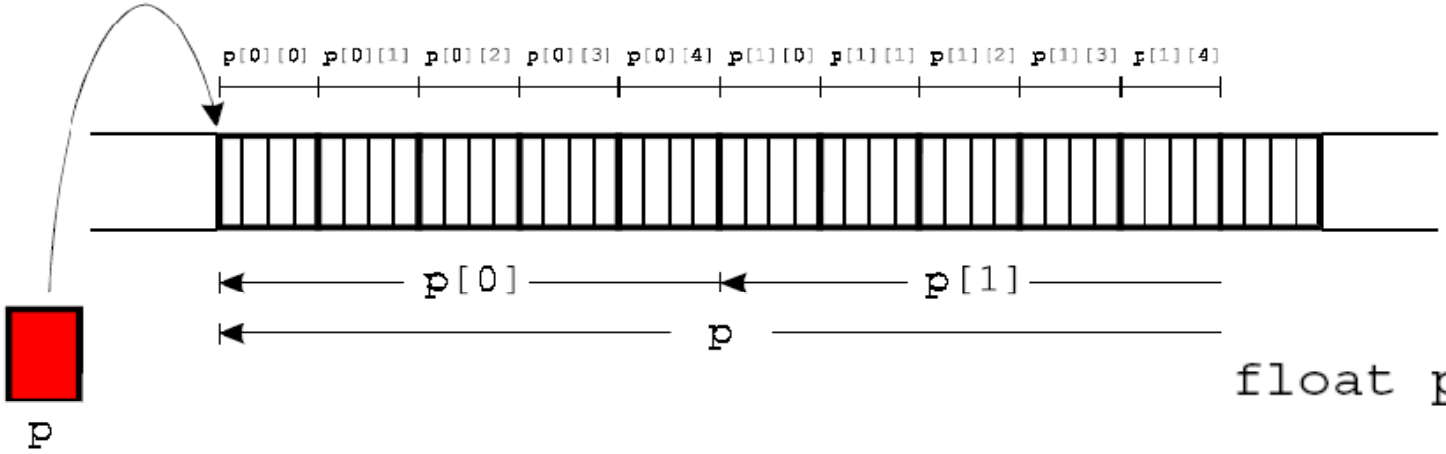
Vektorji in polja - zapis

- Vektorji:




```
float p[10];
p[0] = 1;
p[1] = 2;
p[2] = 3;
```

- Polja:




```
float p[2][5];
```

Vektorji in polja – primer

- Zamenjamo vsebino vektorjev:

```
#include <stdio.h>

void zamenjaj_vect(int vect1[2],int vect2[2])
{
int vectZac[2];
      vectZac[0]=vect1[0];vectZac[1]=vect1[1];
      vect1[0]=vect2[0];vect1[1]=vect2[1];
      vect2[0]=vectZac[0];vect2[1]=vectZac[1];
}

int main()
{
int v[2]={1,2};
int r[2]={3,4};
      zamenjaj_vect(v,r);
      printf("v[0]=%d, v[1]=%d\n",v[0],v[1]);
      printf("r[0]=%d, r[1]=%d\n",r[0],r[1]);
      return 0;
}
```

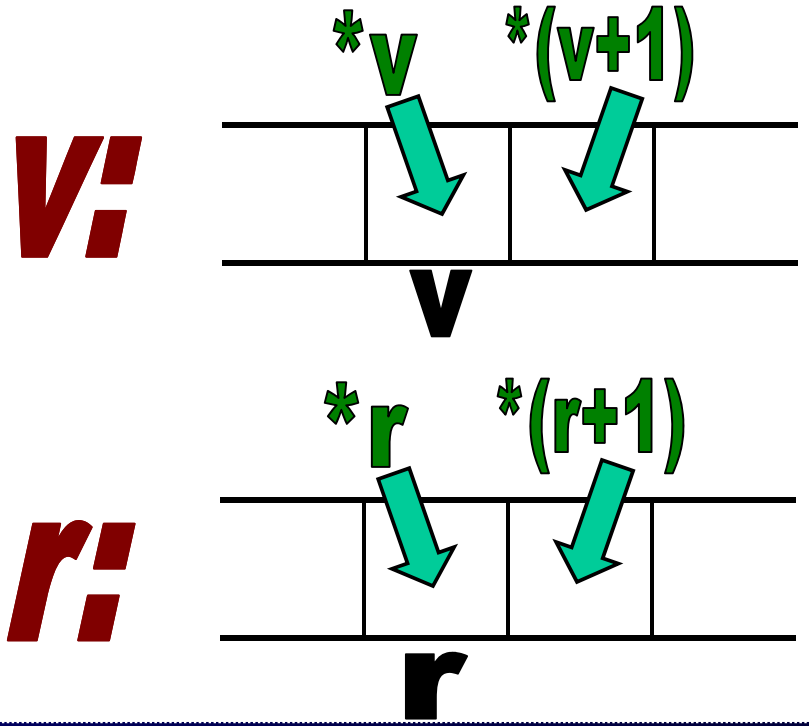
Kazalca!!!

Vektorji – primer s kazalci

- Zamenjamo vsebino vektorjev:

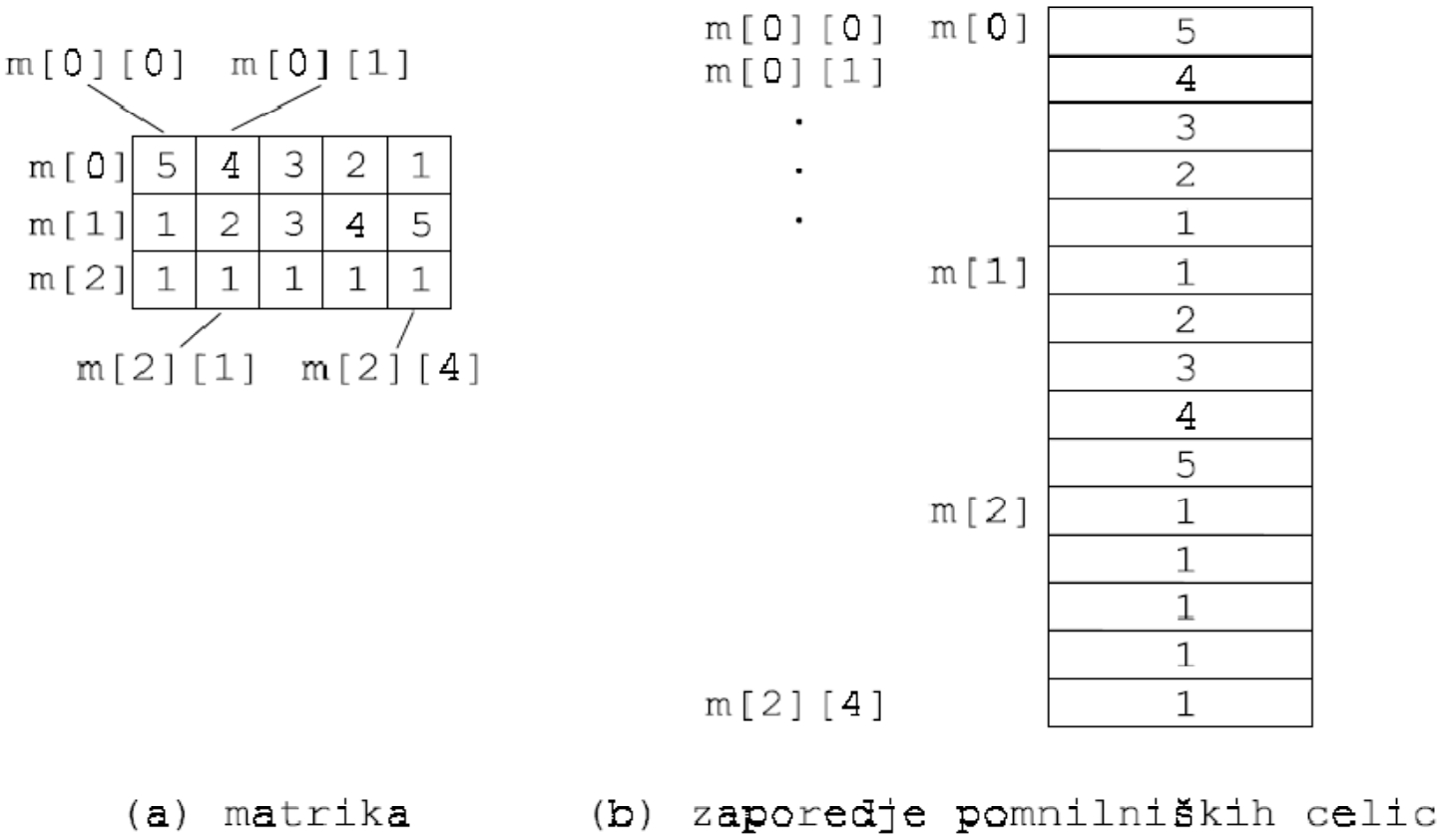
```
#include <stdio.h>

void zamenjaj_vect(int *vect1,int *vect2)
{
int vectZac[2];
   vectZac[0]=*vect1;vectZac[1]=*(vect1+1);
   *vect1=*vect2;*(vect1+1)=*(vect2+1);
   *vect2=vectZac[0];*(vect2+1)=vectZac[1];
}

int main()
{
int v[2]={1,2};
int r[2]={3,4};
      zamenjaj_vect(v,r);
      printf("v[0]=%d, v[1]=%d\n",v[0],v[1]);
      printf("r[0]=%d, r[1]=%d\n",r[0],r[1]);
      return 0;
}
```

Kazalci!!!



Dvodimenzionalno polje

- Polje je v C izvedeno kot vektor več vektorjev.



(a) matrika (b) zaporedje pomnilniških celic

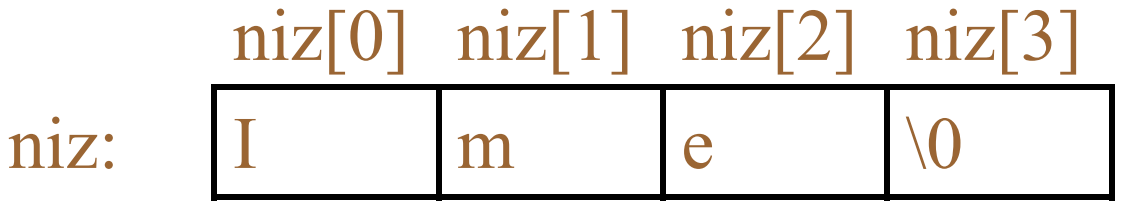
Polja – primer s kazalci

- Zamenjamo vsebino polj:

```
#include <stdio.h>

void zamenjaj_pol(int *pol1,int *pol2)
{
int PolZac[4];
   PolZac[0]=*pol1;PolZac[1]=*(pol1+1);PolZac[2]=*(pol1+2);PolZac[3]=*(pol1+3);
   *pol1=*pol2;*(pol1+1)=*(pol2+1);*(pol1+2)=*(pol2+2);*(pol1+3)=*(pol2+3);
   *pol2=PolZac[0];*(pol2+1)=PolZac[1];*(pol2+2)=PolZac[2];*(pol2+3)=PolZac[3];
}

int main()
{
int P1[2][2]={{1,2},{3,4}};
int P2[2][2]={{5,6},{7,8}};
      zamenjaj_pol(P1,P2);
      printf("P1[0][0]=%d, P1[0][1]=%d\n",P1[0][0],P1[0][1]);printf("P1[1][0]=%d, P1[1][1]=%d\n",P1[1][0],P1[1][1]);
      printf("P2[0][0]=%d, P2[0][1]=%d\n",P2[0][0],P2[0][1]);printf("P2[1][0]=%d, P2[1][1]=%d\n",P2[1][0],P2[1][1]);
 return 0;
}
```

Polja znakov

- Pogosto uporabljamo za izpise.
- Imajo posebno obliko:
  - Definicija:
    - char niz[4]=”Ime”;
  - Zadnji znak v polju je vedno ´\0´, kar pomeni konec niza.
  - Niz je vedno vsaj za eno mesto večji od števila znakov zapisa!



- Z zaglavjem *string.h* imamo na voljo več funkcij za delo z nizi znakov.

Polja znakov - primer

- Zamenjamo vsebino niza znakov:

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

int main()
{
char ime[20]="Joze";

    strcpy(ime,"Janez");

    printf("%s\n",ime);

     return 0;
}
```

Zaglavje *string.h*

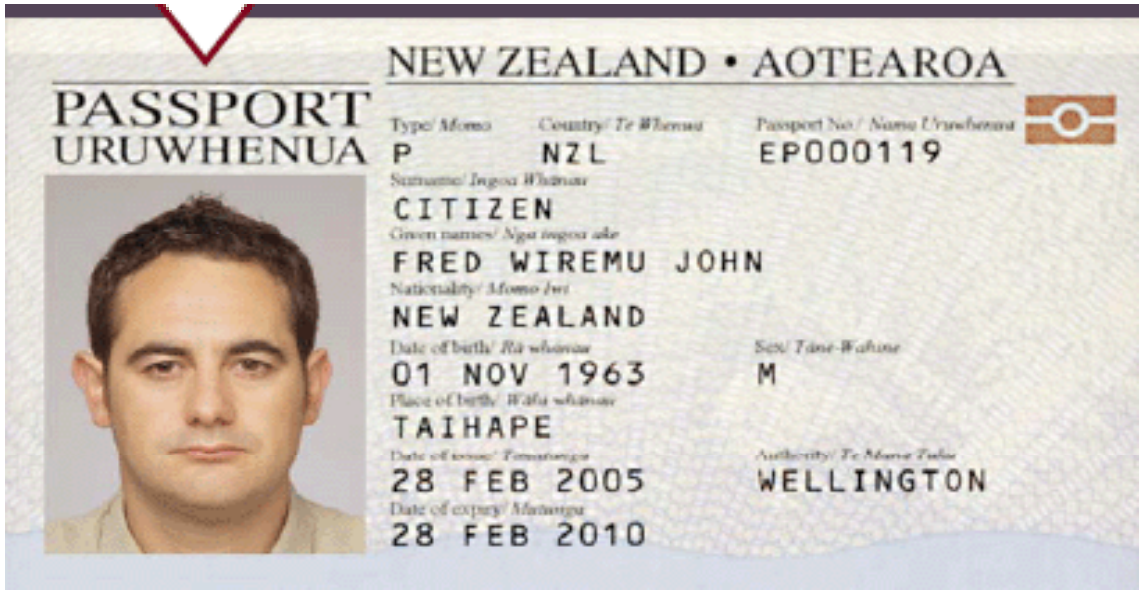
Začetni niz znakov

Sprememba niza

Izpis niza

Strukture

- Strukture so v osnovi zelo podobne vektorjem, vendar jih za razliko od le-teh sestavljajo elementi različnih tipov.
- Primer strukture je vsebina osebne izkaznice:



Primer strukture

```
struct Osebni_podatki                /* deklaracija strukture */
{
   char ime[20];              /* ime ima lahko največ 20 znakov */
   char priimek[20];          /* priimek ima lahko največ 20 znakov */
   int leto_rojstva;          /* letnica rojstva */
   unsigned spol:1;           /* spol - 0:moški*/
};

struct Osebni_podatki osebni_podatki;  /* definicija strukture */
osebni_podatki.leto_rojstva=1900; /* prireditev vrednosti elementu
   strukture */
```

Primer strukture

- Izpis osebnih podatkov:

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

struct Osebni_podatki
{
      char ime[20];
      char priimek[20];
      int dan_rojstva;
      int mesec_rojstva;
      int leto_rojstva;
      /* dodamo lahko še več */
};
```

```
int main()
{
struct Osebni_podatki joze;
      strcpy(joze.ime,"Joze");
      strcpy(joze.priimek,"Marjetica");
      joze.dan_rojstva=20;
      joze.mesec_rojstva=8;
      joze.leto_rojstva=2007;
      printf("\n%s %s\n",joze.ime,joze.priimek);
      printf("rojen: %d.%d.",joze.dan_rojstva,joze.mesec_rojstva);
      printf("%d\n",joze.leto_rojstva);
}
```

Primer strukture – s funkcijo

- Izpis osebnih podatkov:
  - Uporabimo funkcijo

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

struct Osebni_podatki
{
      char ime[20];
      char priimek[20];
      int dan_rojstva;
      int mesec_rojstva;
      int leto_rojstva;
      /* dodamo lahko še več */
};
```

```
void izpis_op(struct Osebni_podatki op)
{
    printf("\n%s %s\n",op.ime,op.priimek);
    printf("rojen: %d.%d.",op.dan_rojstva,op.mesec_rojstva);
    printf("%d.\n",op.leto_rojstva);
}

int main()
{
struct Osebni_podatki joze = {”Joze”, ”Marjetica”,20,8,2007};
      izpis_op(joze);
}
```

Strukture – s kazalci

- Tudi pri strukturah lahko uporabimo kazalce.
- Zapis:
  - Original:
    - Struktura.element
  - Kazalec:
    - Struktura->element

Primer strukture – s kazalci

- Zamenjava osebnih podatkov:
  - Uporabimo funkcijo *zamenjava_op*

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

struct Osebni_podatki
{
      char ime[20];
      char priimek[20];
      int dan_rojstva;
      int mesec_rojstva;
      int leto_rojstva;
      /* dodamo lahko še več */
};
void zamenjava_op(struct Osebni_podatki *op)
{
    strcpy(op->ime,"Janez");
}
```

```
void izpis_op(struct Osebni_podatki op)
{
    printf("\n%s %s\n",op.ime,op.priimek);
    printf("rojen: %d.%d.",op.dan_rojstva,op.mesec_rojstva);
    printf("%d.\n",op.leto_rojstva);
}

int main()
{
struct Osebni_podatki joze;
      strcpy(joze.ime,"Joze"); strcpy(joze.priimek,"Marjetica");
      joze.dan_rojstva=20; joze.mesec_rojstva=8;
      joze.leto_rojstva=2007;
      zamenjava_op(&joze);
      izpis_op(joze);
}
```

Kazalci na funkcije

- Definiramo lahko tudi kazalce na funkcije. Tak kazalec lahko kaže na različne funkcije (istega tipa).
- Kazalec na funkcijo je spremenljivka!!
- Lahko je tudi element nekega polja ali strukture.

Kazalec na funkcijo - primer

- Različne računske operacije:

```
#include <stdio.h>
#include <string.h>
float (* operacija)(float,float);

float vsota (float a, float b)
{
   return (a+b);
}

float razlika (float a, float b)
{
     return(a-b);
}
```

```
void main()
{
float x= 100.0, y=200.0, c;

   operacija = vsota;
   c = operacija(y, x);
   printf("vsota = %f \n", c);

   operacija = razlika;
   c = operacija(y, x);
   printf("razlika = %f \n", c);
}
```

Strukture z bitnimi polji

- Elementi v strukturi so lahko tudi zlogi, dolgi le nekaj bitov. To je uporabno predvsem pri sistemskem programiranju.
- Prihranimo prostor v pomnilniku.
- Primer:
  - Znak za prikaz na zaslonu:

```
struct znak
{
   char koda;          /* koda znaka */
   int barva:4;        /* barva izpisa: zasede 4 bite */
   int poudarjen:1;  /* 0-ni poudarjen, 1-poudarjen – zasede 1 bit */
   int font:5;         /* tip fonta: zasede 5 bitov */
   ...
}
```

Unije

- Deklaracija unije je enaka deklaraciji strukture.
- Unija omogoča shranjevanje podatkov različnih tipov v isto pomnilno področje.
- Prihranimo prostor v pomnilniku.
- Primer:

```
union podatek
{
   char ch;
   int num;
   float fnum;
}
```

```
union podatek a,b;
...
   a.ch = "1";
...
   a.num =1;
...
```

Lokalne in globalne spremenljivke

- Lokalne spremenljivke
  - So veljavne znotraj funkcij.
  - Imamo tudi statične spremenljivke (obdržijo vrednost tudi po izhodu iz funkcije).
- Globalne spremenljivke
  - Veljajo za celoten program.
  - Ponavadi so definirane v *main* funkciji, vendar se jih izogibamo
    - Ko imamo večje programe ne vemo katera funkcija se je spremenila, ko je kaj narobe.
    - Paziti moramo na začetno vrednost! ! Vgrajeni sistemi brez operacijskega sistema imajo poljubno vrednost ob vklopu.

Lokalne in globalne spremenljivke - primer

```
#include <stdio.h>
int a=10;
int b=20;
int c=30;

void izpis_lok()
{
    int a=1;
    printf("Lokalna spremenljivka a = %d\n",a);
    printf("Globalna spremenljivka b = %d\n",b);
    printf("Globalna spremenljivka c = %d\n",c);
}

int main(void)
{
int b;
   b=2;
   printf("Globalna spremenljivka a = %d\n",a);
   printf("Lokalna spremenljivka b = %d\n",b);
   printf("Globalna spremenljivka c = %d\n",c);
   izpis_lok();
}
```

Globalne spremenljivke

Lokalna spremenljivka

Lokalna spremenljivka

Izpis:

Globalna spremenljivka a = 10

Lokalna spremenljivka b = 2

Globalna spremenljivka c = 30

Lokalna spremenljivka a = 1

Globalna spremenljivka b = 20

Globalna spremenljivka c = 30

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 (obdržijo vrednost) tudi po izstopu iz njihove funkcije oziroma bloka. Eksterne spremenljivke, ki so dostopne le funkcijam za njihovo (eksterno) deklaracijo.

Spremenljivke tipa static

- Spremenljivka naj obdrži vrednost tudi po izhodu iz funkcije.
- Primerno npr. za štetje, kolikokrat je bila funkcija izvedena.
- Primer:

```
#include <stdio.h>

void stetje()
{
    static int a=0;
    printf("Funkcija se je izvedla %d krat\n",++a);
}

int main(void)
{
    stetje();
    stetje();
}
```

Knjižnica vhodno/izhodnih funkcij

- 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

Formatirano branje in izpis

- Uporabljamo funkciji *printf* in *scanf*:
- Branje:
  - Splošna oblika:
    - scanf(format, seznam naslovov spremenljivk);
  - Primer:
    - scanf("%d“,&a);
  - Znak & pred imenom spremenljivke: za vhodni parameter mora biti podan naslov in ne vrednost spremenljivke
- Izpis:
  - Splošna oblika:
    - printf (format, seznam spremenljivk ali konstant);
  - Primer:
    - printf("a=%d“,a);

Formatirano branje in izpis – specifikacije (za %)

d	Desetiška cela števila
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 (crka e)
E	Realna števila,znanstvena notacija (crka 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

Ukazi prevajalniku – makro ukazi

- Pred samim prevajanjem pregleda program v jeziku C predprocesor. Ta spozna navodila (direktive), za katera je značilno, da se začnejo z znakom #.
- Predprocesor izvede operacije:
  - Vključevanje datotek v program
    - #include
  - Deklaracija makrojev, ki jih nato v programu razširi.
    - #define
    - #undefine
  - Pogojevanje, katere dele našega programa naj vključi v prevedeno kodo in katere ne.
    - #if
    - #ifdef simbol
    - #ifndef simbol
    - #else
    - #endif
  - Javlja napake
    - #error
- Z direktivami pragma nastavljamo informacije, ki se v splošnem od sistema do sistema razlikujejo.
  - #pragma OPTIMIZE ON

Ukazi prevajalniku – definicija

- #define
  - #define NIZ_ZANKOV niz_znakov
    - Prevajalnik bo zapis NIZ_ZNAKOV povsod v datoteki nadomestil z zapisom niz_znakov.
  - #define ST_ZANKOV 20
    - Prevajalnik bo zapis NIZ_ZNAKOV povsod v datoteki nadomestil s številom 20.
  - #define vsota(a,b) ((a)+(b)).
    - Prevajalnik bo zapis vsota(a,b) povsod v datoteki nadomestil z zapisom ((a)+(b)).
- #undefine
  - #undefine NIZ_ZNAKOV
    - Prevajalnik ne pozna več zapisa NIZ_ZNAKOV.

Ukazi prevajalniku – vključevanje

- *#include*
  - *#include "header.h“*
    - Prevajalnik bo na mesto zapisa #include”header.h” vstavil vsebino datoteke header.h.
    - header.h je v istem direktoriju kot datoteka.
  - *#include <header.h>*
    - Prevajalnik bo na mesto zapisa #include”header.h” vstavil vsebino datoteke header.h.
    - header.h je v globalnem direktoriju, ki je definiran za vključevanje zaglavij.

Ukazi prevajalniku – pogojno prevajanje

- *#if, #else in #endif*

```
#if T_ALI_F
    blok_programa1
#else
    blok_programa2
#endif
```

Če je T_ALI_F različen od 0, se prevede blok_programa1, sicer se prevede blok_programa2

- *#ifdef, #else in #endif*

```
#ifdef IME1
    blok_programa3
#else
    blok_programa4
#endif
```

Če je IME1 definiran, se prevede blok_programa3, sicer se prevede blok_programa4

Dinamično dodeljevanje pomnilnika

- Za zasedbo pomnilnika uporabljamo funkcije:
  - malloc
    - Funkcija malloc ima en sam argument, ki pove, koliko bytov pomnilnika potrebujemo.
  - calloc
    - Funkcija calloc ima dva argumenta: prvi pove število elementov, drugi pove velikost posameznega elementa (v bytih).
  - Oba klica vrneta kazalec tipa "void". Zato praviloma kazalec pretvorimo v pravi tip.
  - realloc
    - Funkcija realloc spremeni polje, naslovljeno s kazalcem, za definirano število bytov.
  - Če spomina ni dovolj, vrnejo vse tri funkcije vrednost 0.
- Za sprostitev pomnilnika uporabljamo funkcijo:
  - free
    - Funkcija sprosti pomnilnik, ki ga je zasedala spremenljivka.

Dinamična alokacija pomnilnika - malloc

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>

int main(void)
{
   char *str;
   /* zasedi pomnilnik za polje str */
   if ((str = (char *) malloc(10)) == NULL)
   {
     printf("Pomnilnika ni dovolj!\n");
     exit(1);  /* Če je pomnilnika premalo, zaključi program */
   }
   strcpy(str, "Hello");
   printf("Niz znakov je %s\n", str);
   free(str);     /* sprosti pomnilnik */
}
```

Pretvorba v pravi tip

Dinamična alokacija pomnilnika - calloc

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

int main(void)
{
   char *str = NULL;

   /* zasedi pomnilnik za polje str */
   str = (char *) calloc(10, sizeof(char));

   strcpy(str, "Hello");
   printf("Niz znakov je %s\n", str);

   free(str);     /* sprosti pomnilnik */
}
```

Pretvorba v pravi tip

Velikost elementa

Delo z datotekami

- Za delo z datotekami uporabljamo funkcije:
  - fopen
    - Odpiranje datoteke. V argumentih določimo ime in lastnosti datoteke.
  - fclose
    - Zapiranje datoteke. V argumentih določimo ime in lastnosti datoteke.
  - fprintf
    - Formatirano pisanje v datoteko, primerno za pisanje v tekstovne datoteke.
  - fscanf
    - Formatirano branje iz datoteke, primerno za branje iz tekstovnih datotek.
  - feof
    - Funkcija preverja, če je bil dosežen konec datoteke.
  - fwrite
  - fread
  - fgetc
  - ...

Delo z datotekami – primer

```
#include <stdio.h>

int main(void)
{
  FILE *in, *out;                                   // definicija datotek
  int i;

  if ((in = fopen("vhod.txt", "rt")) == NULL)       // odpiranje vhodne datoteke
  {
    fprintf(stderr, "Ne morem odpreti vhodne datoteke.\n");
    return 1;
  }
  if ((out = fopen("izhod.txt", "wt")) == NULL)     // odpiranje izhodne datoteke
  {
    fprintf(stderr, "Ne morem odpreti izhodne datoteke.\n");
    return 1;
  }
  while (!feof(in))             // do konca datoteke vhod.txt
  {
    fscanf(in,"%d",&i);                 // beri stevilko iz vhod.txt
    fprintf(out,"%d\n",i);              // zapisi stevilko v novo vrstico izhod.txt
  }
  fclose(in);
  fclose(out);
  return 0;
}
```

vhod.txt

```
1 2 3 4 5
6 7 8 9
```

izhod.txt

```
1
2
3
4
5
6
7
8
9
```

Programiranje vhodno/izhodnih vmesnikov

- Uporabljamo maskiranje bitov!
- Posamezni bit lahko pomeni:
  - Informacijo o:
    - Stanju naprave
    - Stanju vhoda/izhoda
  - Ukaz napravi
- Primer:
  - Register definiramo:

```
#define RESULT1          *((volatile int *)0x70A9)
```

  - Pozneje mu spreminjamo vrednosti:

```
RESULT1 = 0x01;
```

  - Ali preberemo vrednost:

```
Spr = RESULT;
```

Bitne operacije, maskiranje

- Omogočajo postavljanje, brisanje ali branje določenih bitov znotraj spremenljivke (besede) brez vpliva na ostale bite.
- Uporabljamo ga za spreminjanje in branje vsebine registrov.
- Maskiranje je dejansko le uporaba bitnih operacij:
  - Dva argumenta:
    - Spremenljivka
    - Maska
  - & (bitni IN) uporabljamo za branje in brisanje.
  - | (bitni ALI) uporabljamo za postavljanje bitov.

Maskiranje, Branje

- Primer:
  - Prebrati želimo bit 3 v besedi. Uporabimo bitni IN:

	1	0	1	1	0	1	0	0	**Spremenljivka**
&	0	0	0	0	0	1	0	0	**Maska**
	0	0	0	0	0	1	0	0	**Rezultat**

- Rezultat je maskirana vrednost spremenljivke, ki vsebuje le informacijo o 3. bitu.
- Zapis v C:

```
unsigned int Rezultat;
unsigned int Spremenljivka = 0xb4;
Rezultat = Spremenljivka & 0x04;
```

Maskiranje, Postavljanje

- Primer:
  - Bit 4 v besedi želimo postaviti na 1, ostalih bitov ne želimo spreminjati. Uporabimo bitni ALI.

	1	0	1	1	0	1	0	0	**Spremenljivka**
|	0	0	0	0	1	0	0	0	**Maska**
	1	0	1	1	1	1	0	0	**Rezultat**

- Bit 4 je 1, ostali biti se niso spremenili.
- Zapis v C:

```
unsigned int Rezultat;
unsigned int Spremenljivka = 0xb4;
Rezultat = Spremenljivka | 0x08;
```

Maskiranje, Brisanje

- Primer:
  - Izbrisati (postaviti na 0) želimo bit 6, ostale vrednosti naj ostanejo nespremenjene. Uporabimo bitni IN.

	1	0	1	1	0	1	0	0	**Spremenljivka**
&	1	1	0	1	1	1	1	1	**Maska**
	1	0	0	1	0	1	0	0	**Rezultat**

- Maska vsebuje eniški komplement (bitno negacijo). Bit 6 rezultata je na 0, ostali se niso spremenili.
- Zapis v C:

```
unsigned int Rezultat;
unsigned int Spremenljivka = 0xb4;
Rezultat = Spremenljivka & (~0x20);
```

Domača naloga

Možna vprašanja na izpitu

- Opiši definicijo spremenljivk!
- Naštej tipe spremenljivk!
- Podaj razliko med lokalnimi in globalnimi spremenljivkami!
- Opiši avtomatične pretvorbe med tipi!
- Opiši zahtevane pretvorbe med tipi!
- Podaj prednosti in slabosti uporabe podprogramov (funkcij)!
- Opiši strukturo funkcije!
- Opiši deklaracijo in definicijo funkcije!
- Kazalci – podaj opis!
- Opiši vektorje in polja v programskem jeziku C!
- Opiši strukture v programskem jeziku C!
- Bitne operacije – kako je izvedeno branje enega bita?
- Bitne operacije – kako je izvedeno brisanje enega bita?
- Bitne operacije – kako je izvedeno postavljanje enega bita?