

---

# PIPI

## RJEŠENJA VJEŽBI ZA BLITZ 01

Grupa 07,  
Z. Šimić, 2008.

# Teme za 1. blitz

---

- Prikaz pozitivnih cijelih brojeva
- Raspon cijelih brojeva
- Prikaz negativnih cijelih brojeva
- Raspon brojeva različitih tipova podataka
- Prikaz realnih brojeva (komponente)
- Prikaz realnih brojeva (vrijednosti)
- Brojevni sustavi (hexa, octal itd.)
- Tipovi konstanti i imena varijabli u C-u

# Prikaz pozitivnih cijelih brojeva

---

- Odrediti dekadsku vrijednost u registru (memoriji) za broj bez korištenja i uz korištenje dvojnog komplementa
  - Za registar od 3 bita i broj  $2+2$ :  
**bez 2k.:**  $2_{10} = 010_2 \rightarrow 2_{10} + 2_{10} = 100_2 \rightarrow 4_{10}$   
**sa 2k.:**  $2_{10} = 010_2 \rightarrow 2_{10} + 2_{10} = 100_2$  (dvojni komplement)  $\rightarrow$  komplement  $= 011_2 \rightarrow + 1 = 100_2 \rightarrow -4_{10}$
  - Za registar od 3 bita i broj  $3_{10}+3_{10}$  (rješenja  $6_{10}$  i  $-2_{10}$ ).
- Odrediti dekadsku vrijednost cijelog broja uz korištenje tehnike dvojnog komplementa
  - Primjer  $01011_2$  i  $11011_2$   
 $01011_2 = 11_{10}$   
 $11011_2 =$  (dvojni komplement)  $\rightarrow$   
 $\rightarrow$  komplement  $= 00100_2 \rightarrow + 1 = 00101_2 \rightarrow -5_{10}$

# Prikaz pozitivnih cijelih brojeva

---

- Nakon zadane računske operacije odrediti sadržaj registra
  - Za  $3+3+1$  u 3 bitnom registru:  $3+3+1 = 7_{10} = 111_2$ 
    - Ili isti broj u 2 bitnom registru:  $11_2 = 3_{10}$
- Odrediti maksimalni cijeli broj koji se može prikazati u registru s dvostrukim komplementom i bez 2k.:
  - Za 2 bitni i 4 bitni registar  
 $2^n - 1 \Rightarrow 2^2 - 1 = 3, \quad 2^4 - 1 = 15 \quad 2^{n-1} - 1 \Rightarrow 2^{2-1} - 1 = 1, \quad 2^{4-1} - 1 = 7$
- Odrediti broj različitih vrijednosti koje se može prikazati
  - Za prethodni primjer  
za 2 bita 4 vrijednosti i za 4 bita 16 vrijednosti; općenito  $= 2^n$

# Prikaz pozitivnih cijelih brojeva

---

- Kako izgleda zbroj dva binarna broja binarno i dekadski
  - $1101_2 + 1011_2$   
 $11000_2$                        $(24=13+11)_{10}$
- Koji broj je prikazan u prethodnom primjeru ukoliko se primjeni tehnika dvojnog komplementa?  
 $11000_2$  (dvojni kompl.)  $\rightarrow$  komplement  $= 00111_2 \rightarrow +1 \rightarrow 01000_2 = -8_{10}$
- Pretvoriti dekadski broj u binarni
  - Primjer  $199_{10}$   
 $11000111_2$
- Množenje razlomljenog binarnog broja
  - Primjer množenje s  $2_{10}$   
binarni zarez (točka) se pomiče 1 mjesto u desno

# Raspon cijelih brojeva

---

- Različiti pozitivni brojevi u binarnom prikazu
  - Recimo registar od 3 bita:  
000, 001, 011, 010 → 3 različita broja  
općenito se može prikazati  $2^{n-1} - 1$  brojeva
- Različiti negativni brojevi u binarnom prikazu
  - Recimo registar od 3 bita:  
100, 101, 111, 110 → 4 različita broja  
općenito se može prikazati  $2^{n-1}$  brojeva
- Raspon brojeva u binarnom prikazu
  - Recimo registar od 5 bita:  
bez predznaka                      s predznakom u tehnici dvojnog komplementa  
[0, 31]                                      [-16, 15]

# Raspon cijelih brojeva

---

- Raspon brojeva u binarnom prikazu

- char (oktet, byte) raspon:

s predznakom u tehnici dvojnog komplementa  
[-128, 127]

unsigned (bez predznaka)  
[0, 255] [0,  $2^8-1$ ]

- short int (16 bita) raspon:

unsigned (bez predznaka)  
[0,  $2^{16}-1$ ] ili [0, 65535]

s predznakom u tehnici dvojnog komplementa  
[- $2^{16-1}$ ,  $2^{16-1}-1$ ] ili [-32768, 32767]

# Prikaz negativnih brojeva

---

- Potreban broj bita za prikaz nekog negativnog broja
  - Broj  $-42_{10}$ :  
 $42_{10} = 101010_2 \rightarrow \text{komplement} = 010101_2$   
(dvojni komplement)  $+1 \rightarrow 010110_2$   
Potrebno je minimalno 7 bitova.
- Zapis negativnog broja tehnikom dvojnog komplementa
  - Broj  $-3$  u 16 bitnom registru:  
 $3_{10} = 0000\ 0000\ 0000\ 0011_2 \rightarrow \text{komplement} = 1111\ 1111\ 1111\ 1100_2$   
(dvojni komplement)  $+1 \rightarrow \mathbf{1111\ 1111\ 1111\ 1101_2}$



# Prikaz negativnih brojeva

---

- Prikazati dekadski broj heksadecimalno primjenom tehnike dvojnog komplementa
  - Broj  $-21_{10}$  u 8 bitnom registru :  
 $21_{10} = 0001\ 0101_2 \rightarrow \text{komplement} = 1110\ 1010_2$   
 $\rightarrow +1 \rightarrow (\text{dvojni komplement})\ \mathbf{1110\ 1011_2} = \mathbf{EB_{16}}$
  - Broj  $-121_{10}$  u 9 bitnom registru :  
 $121_{10} = 0\ 0111\ 1001_2 \rightarrow \text{komplement} = 1\ 1000\ 0110_2$   
 $\rightarrow +1 \rightarrow (\text{dvojni komplement})\ \mathbf{1\ 1000\ 0111_2} = \mathbf{187_{16}}$
- Sadržaj registra opisan hexadecimalno u dekadski
  - Kada je u 4 bitnom registru zapisano  $\mathbf{B_{16}}$   
 $\mathbf{B_{16}} = \mathbf{1011_2} \rightarrow \text{kompl.} = 0100_2 \rightarrow + 1 = 0101_2 \Rightarrow \mathbf{-5_{10}}$

# Vrijednost u varijabli nakon pridruživanja

---

## ■ char

```
char c = -8;
```

-8

(bin. ili hex.)

1111 1000 ili F8

```
char c = 121;
```

```
c = c + 7;
```

-128

1000 0000 ili 80

```
char c = 125;
```

```
c = c + 9;
```

-122

1000 0110 ili 86

```
char c = -125;
```

```
c = c - 6;
```

125

0111 1101 ili 7D

## ■ short

```
short i = 32767;
```

```
i = i + 1;
```

-32768

1000 0000 0000 0000 ili  
8 0 0 0

# Prikaz realnih brojeva

---

- Raspon binarnog eksponenta
  - [-126, 127] za jednostruku preciznost
    - $K = BE + 127$  (8 bita za karakteristiku)
    - $K = 255$  posebno značenje ( $M \neq 0 \rightarrow \text{NaN}$ ,  $M = 0 \rightarrow +\infty$ )
    - $K = 0$  posebno značenje ( $M = 0 \rightarrow 0$ ,  $M \neq 0 \rightarrow \text{denormalizirani broj}$ )
  - [-1022, 1023] za dvostruku preciznost
    - $K = BE + 1023$  (11 bita za karakteristiku)
    - $K = 2047$  posebno značenje ( $M \neq 0 \rightarrow \text{NaN}$ ,  $M = 0 \rightarrow +\infty$ )
- Duljina mantise – određuje preciznost
  - 23 bita za jednostruku preciznost (+ skriveni bit) - prec. 7 dek. znam.
  - 52 bita za dvostruku preciznost (+ skriveni bit) - prec. 15 dek. znam.
- Dodavanje jako malog broja jako velikom broju
  - Na primjer  $1,0e33 + 1,0e-33$   
Ne mijenja veliki broj jer mantisa ima ograničenu duljinu!

# Prikaz realnih brojeva

---

- Binarni rezultat množenja binarnog broja

- $101101 \times 2^{-5} =$

$$1.01101$$

- $101.101 \times 2^2 =$

$$10110.1$$

- Karakteristika za realni broj u jednostrukoj preciznosti

- Realni broj 33,257

Dovoljno je gledati samo cijelobrojni dio:  $33_d = 100001_b$

⇒ normaliziranu mantisu 1.00001 treba množiti sa  $2^5$  da se dobije isti broj  $100001 \Rightarrow BE=5$

⇒  $K=5+127=132 \Rightarrow 01000100$

# Broj bita mantise i dekadskih znamenaka

---

- Broj potrebnih bita **b** za dekadski broj s **d** znamenaka
  - $2^b = 10^d \rightarrow b = d/\log 2 = d/0,3$
  - Za dekadski broj s 6 znamenaka:  
 $b = 5/0,3 = 16,7 = 17$  bita  $\rightarrow$  16 bita mantisa (uz skriveni bit)
- Broj dekadskih znamenaka **d** uz zadani broj bita **b**
  - $2^b = 10^d \rightarrow d = b \cdot \log 2 = b \cdot 0,3$
  - Za mantisu sa 23 bita:  
 $d = 24 \cdot 0,3 = 7,2 = 7$  decimalnih znamenaka
- Imati na umu raspodjelu bita na **P + K + M**
  - Za preciznost imati na umu i skriveni bit  $b = M + 1$

# Određivanje mantise za IEEE 754 realni broj

**Prikaz u binarnom i heksadecimalnom obliku?**

**Primjer**  $-118.625_{10}$

Binarno:  $1110110.101_2 = 1.110110101 \cdot 2^6$

Binarno u standardnoj preciznosti:

$BE=6 \rightarrow K=6+127=133$

1	8	23	broj bitova
+--+-----+-----+-----+-----+			
S	Karakt.	Mantisa	
1	10000101	110110101000000000000000	
+--+-----+-----+-----+-----+			
31	30	23 22	redni broj bita (0 je desno) 0
BE +127			
1100 0010 1110 1101 0100 0000 0000 0000			
<b>C2ED4000<sub>16</sub></b>			

**Primjer**  $5.25_{10}$

Binarno:  $101.01_2 = 1.0101 \cdot 2^2$

Binarno u standardnoj preciznosti:

$BE=2 \rightarrow K=2+127=129$

0 100 0000 1 010 1000 0000 0000 0000 0000

**40A80000<sub>16</sub>**

**Primjer:** Prikazati  $-9.125$  u binarnoj standardnoj preciznosti te hexadecimalno i oktalno

■ rješenja:

**11000001000100100000000000000000**

**1100 0001 0001 0010 0000 0000 0000 0000**

**C 1 1 2 0 0 0 0<sub>16</sub>**

**11 000 001 000 100 100 000 000 000 000 000**

**3 0 1 0 4 4 0 0 0 0<sub>8</sub>**

# Još neka pitanja:

---

1. Koji se najmanji broj po apsolutnoj vrijednosti može pohraniti u registru korištenjem tehnike dvojnog komplementa?
  2. Kolika se vrijednost dobije nakon dodavanja broja 2 najvećem pozitivnom broju koji se može prikazati u jednostrukoj preciznosti?
  3. Koliko se puta poveća najveća vrijednost pozitivnog broja u registru sa **b** bita kada se broj bita poveća na **2\*b**?
  4. Koliko treba bita za prikaz pozitivnih brojeva do 6 znamenki u registru bez dvojnog komplementa?
  5. Koliko različitih cijelih brojeva se može prikazati korištenjem dvojnog komplementa u registru od 11 bita?
1. 0 – bez obzira na to koliki je registar!
  2. Vrijednost broja ostaje ista!
  3. Može se prikazati  **$2^b+1$**  puta veći broj.
  4. Treba  $6*3.33 = 20$  bita.
  5. Može se prikazati  **$2^{11}$**  brojeva.

## Tipovi konstanti i imena varijabli u C-u

---

- Koliko mjesta će u memoriji zauzeti konstanta `4.0f`  
`4` okteta – realni tip float jednostruka preciznost
- Identificirajte element izraza: `x=y+40.`  
`float` konstanta
- Konstanta `44U` prikazana hexadecimalno:  
`2c`



## Tipovi konstanti i imena varijabli u C-u

---

- Za izraz `a = 6U` što označava slovo U:
  - 6 je cjelobrojna konstanta bez predznaka
- Binarni prikaz konstante u C-u `0x66`  
`01100110`
- **Neispravno** ime za varijablu  
`int` – ključna riječ u C-u
- **Ispravno** ime za varijablu  
`a_9`

## Tipovi konstanti i imena varijabli u C-u

---

### Odrediti tip konstante i potreban broj okteta u memoriji:

3ed-2	3	21f	4u
krivo!	int: 4	krivo: nedostaje točka	unsigned int: 4
3e+2	9.1u	2.1E10u	0x77L
double: 8	krivo: nema realnog unsigned		hexadek. long int: 4
3e-2	34.1f	21.	3.24
double: 8	float: 4	double: 4	float: 4
07.7f	0x1A.1f	0xff	0101u
krivo: nema realnog okt. i hex.		hexadek.int: 4	okt. unsigned int: 4

## Tipovi konstanti i imena varijabli u C-u

---

- `'\x45'` predstavlja znak:
  - ASCII tablica predstavlja slovo 'A' brojem 65**E**
- Konstanta `0214` (int oktalno prikazan) predstavljena binarno **10001100**
- Ispravan zapis `signed long` hexadecimalne konstante **0x1011**

# PiPI

## RJEŠENJA VJEŽBI ZA BLITZ 02

Grupa 07

Z. Šimić, 2008.

## Teme za 2. blitz

- Jednostrana selekcija
- Apsolutne, relativne i numeričke pogreške
- Raspon realnih brojeva
- Cjelobrojne i realne konstante i varijable u C-u
- ASCII tablica i operacije sa znakovnim tipom podatka
- Redoslijed obavljanja aritmetičkih operacija i konverzija tipova podataka
- Konstantni niz znakova
- Logički tip podataka, relacijski operatori, logički izrazi i naredbe
- Logički izrazi s logičkim operatorima
- Dvostrana selekcija
- Jednostrana selekcija sa složenijim uvjetom
- Dvostrana selekcija sa složenijim uvjetima

# Izdvojeni detalji

- Prioritet izvođenja operatora - podsjetnik
- Razlikovati pridruživane (**i=1**) od ispitivanja identičnosti (**i==1**)
- Paziti na cjelobrojno dijeljenje
- Ponoviti dobro logičke iskaze:
  - **0 = laž**; **sve ostalo istina**
  - **laž = 0**; **istina = 1**
- Neke važne ASCII vrijednosti:
  - **'0'==48**;
  - **'A'==65**; **'a'==97**;

# Prioriteti operatora

## Prioritet operatora

	OPERATORI	PRIDRUŽIVANJE
← Viši prioritet	( )	L → D
	! ~ ++ -- sizeof & * unarni + -	D → L
	(cast)	D → L
	* / %	L → D
	+ -	L → D
	<< >>	L → D
	< <= > >=	L → D
	== !=	L → D
	&	L → D
	^	L → D
Niži prioritet →		L → D
	&&	L → D
		L → D
	? :	D → L
	= *= /= %= += -= &= ^=  = <<=	D → L
	>>=	
	,	L → D

# Raspon realnih brojeva

- Ovisi o broju bita karakteristike
- Jednostruka preciznost
  - najveći broj  $\pm 3.4\text{E}38$  ili  $\pm 2^{128}$   
najmanji broj  $\pm 1.4\text{E}-45$  ili  $\pm 2^{-149}$
- Dvostruka preciznost
  - najveći broj  $\pm 1.7\text{E}308$  ili  $\pm 2^{1024}$
  - najmanji broj  $\pm 4.9\text{E}-324$  ili  $\pm 2^{-1074}$



# Prikaz realnog broja u IEEE 754

- Odrediti sadržaj registra u heksadekadskom obliku koji za prikaz broja  $-1.25_{10}$  prema IEEE 754 standardu u **dvostrukoj preciznosti**.

**$BFF400000000000_{16}$**

- Za registar koji sadrži broj  $C029000000000000_{16}$  treba odrediti realnu dekadsku vrijednost podatka tipa double.

**$-12.5_{10}$**

- Napisati u heksadekadskom prikazu sadržaj registra koji prikazuje:  $-0$ ,  $+\infty$  i NaN

$8000000000000000_{16}$

$7FF0000000000000_{16}$

$7FF1000000000000_{16}$  – prva znamenka može biti i F, a 25 četvrta i ostale znamenke mogu biti bilo što, ali ne sve 0

# Preciznost realnog broja u IEEE 754

- Realni broj se pohranjuje u memoriju u jednostrukoj i dvostrukoj preciznosti te u prikaz koji ima 15 bita za karakteristiku i 65 bita za mantisu (uključujući skriveni bit). Treba prvo odrediti najveću moguću relativnu pogrešku, a potom najveću apsolutnu pogrešku za broj 1000.1 za sva tri slučaja.

- Najveća moguća relativna pogreška ovisi o raspoloživom broju bita mantise (uključujući skriveni bit)  $m$ : općenito  $= 2^{-m}$

$$2^{-24} \approx 6 \cdot 10^{-8} \text{ za jednostruku preciznost}$$

$$2^{-53} \approx 1 \cdot 10^{-16} \text{ za jednostruku preciznost}$$

$$2^{-65} \approx 3 \cdot 10^{-20} \text{ za prikaz prema zadatku}$$

- Najveća moguća apsolutna pogreška ovisi o konkretnom broju  $x$  i najvećoj mogućoj relativnoj pogreški: općenito  $= x \cdot 2^{-m}$

$$1000.1 \cdot 6 \cdot 10^{-8} \approx 6 \cdot 10^{-5} \text{ za jednostruku preciznost i broj 1000.1}$$

$$1000.1 \cdot 1 \cdot 10^{-16} \approx 1 \cdot 10^{-13} \text{ za jednostruku preciznost i broj 1000.1}$$

$$26 \quad 1000.1 \cdot 3 \cdot 10^{-20} \approx 3 \cdot 10^{-17} \text{ za prikaz prema zadatku i broj 1000.1}$$

# Znakovni tip, ASCII tablica, nizovi znakova

ASCII	Znak
48	0
65	A
97	a

```
#include <stdio.h>
int main( ) {
char z='c';
printf(" = %c ", z/2);
printf(" = %d ", z/2);
printf("z= %c ", z-1);
printf(" = %c ", z-32);
printf(" = %d ", z - 'a');
z = z + 1;
printf("z= %c i %d", z, z);
return 0;
}
```

Ispis:

= 1

= 49

z= b

= C

= 2

z= d i 100

# Znakovni tip, ASCII tablica, nizovi znakova

---

```
char z = '\\x41';  
  
printf("z= %c\\n", z);  
  
z = '1';  
printf("z= %c\\n", z+1);  
z = (z+1)/'1';  
printf("z= %d\\n", z);  
z = '1'+'1';  
printf("z= %c\\n", z);
```

Ispis:

z= A

z= 2

z= 1

z= b

ASCII	Znak
48	0
65	A
97	a

# Aritmetički operatori i izrazi

---

- Koliko iznose varijable nakon izvršavanja?

```
int main() {  
    int a=1, b=1;  
    b = (a = a - b) + b;  
    a = a + b;  
}
```

```
/* rezultat:  a = 1    b = 1 */
```

# Aritmetički operatori i izrazi

---

- Koliko iznosi varijabla **x** nakon izvršavanja?

```
int  y, z;  
float x;  
y=4; z=14;  
x = z / y * y + z % y;  
printf("x = %f ", x);
```

**x = 14.000000**

# Aritmetički operatori i izrazi

---

- Što ispisuje program?

```
int x=5, y;  
y = 2*x + x * 5 % x;  
printf("x=%d y=%d", x, y);
```

**x=5 y=10**

Što je rezultat operacije  $a \bmod b$  nad integerima u C-u ( $a \% b$ )!?

# Aritmetički operatori i izrazi

---

- Kolika je vrijednost varijabli nakon izvođenja?

```
int x, y;  
float w;  
x = 5;  
y = x;  
w = x / 4 * y - x * 5 * y;
```

```
// rezultat:  x=5 y=5 w=-120.000000
```



# Aritmetički operatori i izrazi

---

- Kolika je vrijednost varijabli nakon izvođenja?

```
int  a = 9, b = 10;  
a = (b = b - a) + a;  
b = b + a;
```

```
// rezultat  a = 10  b = 11
```

# Aritmetički operatori i izrazi

---

- Odrediti vrijedosti za varijable d i j:

```
int b = 91, d, j;
```

```
d = b / 10;  /* d==9 -broj desetica */
```

```
j = b % 10; /* j==1 -broj jedinica */
```

- To su korisni izrazi za razlaganje dvoznamenkastog broja na jedinice i desetice!

# Aritmetički operatori i izrazi

---

- Dodavanje cijelom broju realnog broja:

```
int i;  
float k;  
k = 6.;  
i = k + 0.5;
```

```
/* rezultat: i = 6 */
```

- Dijeljenje dva cijela broja:

```
int i = 8;  
float r = 2*i/3%2;
```

```
/* rezultat u r = 1.0 */
```

# Vrijednost izraza

```
int x=1, y=-1, z=0;
```

```
z = !z && !y || x;
```

```
z = (x >= y) && (x != y);
```

```
z = x >= y && x != y;
```

```
x = y+x>-1;
```

```
x = y+x>1;
```

```
z=0, x=0; x = !x || y && z;
```

```
x = (!x || y) && z;
```

```
z=5, z = !z + 1;
```

## Rezultat:

➤ z = 1

➤ z = 1

➤ z = 1

➤ x = 1

➤ x = 0

➤ x = 1

➤ x = 0

➤ z = -1

# Vrijednost izraza

---

```
int x=10, y=20, z=0;
```

**Rezultat:**

```
z = x>y || x==10 && y==2;
```

➤ z = 0

```
z = x >= x * x - 90;
```

➤ z = 1

```
z = y <= y / y + 10;
```

➤ z = 0

# Aritmetički operatori i izrazi

- Koliko iznosi varijabla **a** nakon izvršavanja?

```
int main() {  
    int a=1, b=0;  
    if (b!=a) a=!b;  
    a = 1 + a&&1;  
}
```

*/\* rezultat: a = 1*

*stoga što je a = (1 + a) && 1 \*/*

# Kontrolna naredba selekcije

Rezultat u varijablama:

```
int x = 10, y = 0;  
if (!x - y) x = 1+y;  
else y = x-1;
```

x = 10          y = 9

```
x = y = 0;  
if (x - !y) x = y+1;  
else y = x-1;
```

x = 1          y = 0

# Kontrolna naredba selekcije

```
int x = 1, y = 0;  
if (x == !y)  
    x = x+y+1;  
else  
    x = x+y-1;
```

Rezultat u varijablama:

x = 2 y = 0

```
x = 2, y = 1;  
if (x == !y)  
    x = x+y+1;  
else  
    x = x+y-1;
```

x = 2 y = 1



# Kontrolna naredba selekcije

```
int x=1;  
if (x%2) x=11;  
else if (x) x=22;
```

Rezultat u varijabli:

**x = 11**

```
x = 2;  
if (x%2) x=11;  
else if (x) x=22;
```

**x = 22**

# Kontrolna naredba selekcije

```
int x=-1, y=-1, z=1;  
if(x - z == 0) x += x - 1;  
    else z = x && 0;  
        y += z + 1;
```

Rezultat u varijablama:

x=-1 y=0 z=0

```
x = z = -1;  
y = 0;  
if(x - z == 0) x += x - 1;  
    else z = x && 0;  
        y += z + 1;
```

x=-2 y=0 z=-1

# Kontrolna naredba selekcije

Rezultat u varijablama:

```
int x=-1, y=1;  
if(x && y) x=x+1;  
else y=y+1;
```

x=0 y=1

```
x=-1, y=1  
if(x > 0 && y > 0) x=x+1;  
else y=y+1;
```

x=-1 y=2

# Kontrolna naredba selekcije

```
int x=1, y=1, z=1;
char c;
if(x < y){
    if(y < z)
        c = 'A';
    else
        c = 'B';
} else {
    if(y > z)
        c = 'C';
    else
        c = 'D';
}
// Rezultat:  c = 'D'
```

```
int x = -1, y = 1;
if(x+1 > y-1) x=x+1;
else y=y+1;

// Rezultat: x=-1  y=2

x = -1, y = 1;
if(x = y-1) x=x+1;
else y=y+1;

// Rezultat:  x=0  y=2
```

# Istinitost izraza

```
int x=0, y=1;
```

```
if (x==y) ➤ Laž
```

```
if (x=y) ➤ Istina
```

```
x=0; if (! (y=x)) ➤ Istina
```

```
y=1; if (! (y==x)) ➤ Istina
```

```
if (x+1) ➤ Istina
```

```
x=0; if (1+x) ➤ Istina
```

```
if (y-1) ➤ Laž
```

```
y=1; if (1-y) ➤ Laž
```

# Zamke selekcije

Sadržaj varijabli nakon izvođenja,  
slijedom:

Vrijednost:

```
int x=1, y=0, i=1;  
if (x=y+1) x=1; y=5;  
if (!y) x=5; y=0;  
  
if (i%2) i=11;  
else if (i) i=22;  
  
if (x=i<y) x=10;
```

**x=1**   **y=0**

**i=11**

**x=0**

# Zamke selekcije

```
int x, y;  
x = -1; y = 6;  
if (x = y)  
    printf (>%d%d", x, y);  
else  
    y = -1;  
    printf ("%d%d<", x, y);
```

Ispis:

>6666<

Rezultat:

---

```
int x=1, y=-1, z=0;  
if (!(x + y>z))  
    x = x-y+2;  
    y = y-x-1;  
    z = z * (x - y);
```

x=1 y=-1 z=0

# Još neka pitanja:

1. Koliko se mjesta u memoriji zauzme kada se u C-u želi prikazati logički tip podataka (*boolean*)?
  2. Koliko puta je veća najveća moguća apsolutna pogreška realnog broja jednostruke preciznosti kada bi imao mantisu od 16 bita (uključujući skriveni bit)?
  3. Koliko puta je manja najveća moguća relativna pogreška realnog broja dvostruke preciznosti od realnog broja jednostruke preciznosti?
  4. Koliko se smanji najveća moguća relativna pogreška u IEEE 754 formatu jednostruke preciznosti, ako se za karakteristiku koristi 10 bitova uz nepromijenjenu mantisu?
1. Programski jezik C nema logički tip *boolean*! Istina je da u standardu za C99 *boolean* kao tip podatka postoji eksplicitno, ali mi radimo prema standardu za C89.
  2.  $x \cdot 2^{-16} / (x \cdot 2^{-24}) = 2^8$  puta veća najveća apsolutna pogreška
  3.  $2^{-24} / 2^{-52} = 2^{28}$  puta manja najveća relativna pogreška
  4. Veličina karakteristike ne utječe na preciznost i najveća moguća relativna pogreška se ne mijenja.



# PiPI

RJEŠENJA VJEŽBI ZA BLITZ 03

Grupa 07,  
Z. Šimić, 2008.

## Teme za 3. *blitz*

- Višestruko pridruživanje i skraćeno pridruživanje
- Odvajanje naredbi, uvjetno pridruživanje
- Programske petlje:
  - `while` (kratki i dugi oblik naredbe)
  - `do-while` (kratki i dugi oblik naredbe)
  - `for`
  - kombinacije više tipova petlji
  - u kombinaciji s `break`, `continue`
  - kombinacije selekcija i programskih petlji
  - degenerirane (beskonačne, one koje se ne izvršavaju)
- Korištenje bitovnih operatora
- Korištenje `goto`
- Naredba `switch`

# Izdvojeni detalji

- Prioritet izvođenja operatora – koristiti podsjetnik
- Razlikovati pridruživane (**i=1**) od ispitivanja identičnosti (**i==1**)
- Paziti na:
  - opseg petlje i uvjeta
  - naredba prema bloku naredbi
  - cjelobrojno dijeljenje
- Ponoviti logičke iskaze:
  - **0 == laž**; **sve ostalo !=0 istina**
  - **laž == 0**; **istina == 1**
- **continue** završava izvođenje koraka petlje i nastavlja od kraja petlje
- **break** prekida izvođenje petlje i **switch**-a
- Neke korisne ASCII vrijednosti koristiti podsjetnik
  - **'0' == 48**;
  - **'A' == 65**; **'a' == 97**;

## Operatori – rezultati slijedom

<u>ASCII</u>	<u>Znak</u>
48	0
65	A
97	a

```
char z='5' ;
printf("%d", z & 0x0f) ;
printf("%d", z^z+2) ;
printf("%d", ~(z-53)) ;
int x=1, y=2;
x *= y/2 + 4;
y /= x/3 + 1;
printf("x=%d y=%d", x, y) ;
printf("x=%d", x%=(y+=2)) ;
printf("y=%d", y) ;
printf("%d", z>>=y/x) ;
```

Ispis:

5

2

-1

x=5 y=1

x=2

y=3

26

# Uvjetni operator

- rješenja:

## 1. a, b, c?

```
int a=0, b=-2, c=-1;  
a= (c && a) ? c += a : b == a;
```

a=0   b=-2   c=-1

## 2. a, c?

```
int a=1, b=-1, c=1;  
a= (b<c)<<1 ? c+1 : c-1;
```

a=2   c=1

## 3. z?

```
char z, c='5';  
z= c<48 || c>57 ? 'Z' : 'B';
```

z= 'B'

# Uvjetni operator

## 4. Ispis?

```
int n=1, uvjet=0;
printf("%d. odgovor je %s!\n", n,
      uvjet ? "DA" : "NE");
```

## • rješenja:

1. odgovor je NE!

## 5. Ispis?

```
int a=5, b=-1, c=1;
c = (a=c&&b) ? a = b : ++c;
printf("a=%d b=%d c=%d", a, b, c);
```

a=-1, b=-1, c=-1

## 6. x, y?

```
int x=-1, y=1;
x= ++x >= y-- ? x && y++ : x-(++y);
```

x=-1 y=1

## Programska petlja s ispitivanjem uvjeta na početku: kod za izraz:

$$\prod_{n=1}^5 \frac{1}{(n+3)}$$

```
float produkt = 1.;  
int n = 1;  
while (n <= 5) {  
    produkt = produkt / (n+3);  
    n=n+1; }
```

$$\sum_{n=1}^5 \frac{1}{(n+3)}$$

```
float zbroj = 0.;  
int n = 1;  
while(n<=5) {zbroj+=1./ (n+3); n++;}
```

$$\sum_{n=1}^5 \frac{1}{n \cdot 3}$$

```
float zbroj = 0.;  
int n = 1;  
while(n<=5) zbroj += 1./ (3 * n++);
```

# Programska petlja s ispitivanjem uvjeta na početku

Koliko puta se obavlja petlja?

```
char c = 1;  
c = 1;  
while(c>0) c=c+1;
```

/\* Petlja se obavlja 127 puta

/\* Sličan problem sa **short int**  
varijablom!

Koliko puta se obavlja petlja?

```
int n = 33;  
while (n > 9)  
    n -= 3;
```

/\* Petlja se ponavlja 8 puta



# Programska petlja s ispitivanjem uvjeta na početku

Što je rezultat?

```
int n=0, m=10;
while( !(n==3) ) {
    m = m - n;
    n++;
}
```

/\* Na kraju: n=3 m=7 \*/

Što je rezultat?

```
int n, m;
n=m=10;
while (m>0 && n/m) m--;
```

/\* Na kraju: n=10 m=0 \*/

# Do while petlja

```
int x=1, y=1;
do {
    ++x;
    y *= x / 2;
} while (y%x);
```

Prolaz	x	y	y%x
1.	2	1	2
2.	3	1	6
3.	4	2	2
4.	5	4	4
5.	6	12	0

Rezultat: x=6 y=12

```
int x = -1;
float r = 2.;
do {
    x++;
    r += x;
    printf("%f %d ", r, x);
} while (r<4 && x);
printf("DNO");
```

2.000000 0 DNO

# Do while petlja

```
char z='1';
do {
    z+=2;
    printf("%c%d ", z, z);
} while ('5'-z);
```

Ispis: 351553

---

```
int x=9;
do {
    x-=2;
} while (++x>5);
```

Rezultat:

Nakon 4 prolaza x = 5

```
int a=10, b=10, c=10;
do {
    a = --b;
    do {
        b = c--;
    }while(b>9);
}while(!(a<10));
```

Prolaz	a	b	c
1.v	9	9	10
1.u		10	9
2.u		9	8
Kraj	9	9	8

v. i u. – vanjska i unutrašnja petlja

# For petlja

```
int i, j, x=10;
for (i=0; i<2; i++)
    for (j=1; j<3; ++j)
        x -= i + j;
```

Prolaz	i	j	x
1.v-1.u	0	1	9
1.v-2.u	0	2	7
2.v-2.u	1	1	5
2.v-2.u	1	2	2

Na kraju: i=2 j=3 x=2

v.i u. – vanjska i unutrašnja petlja

```
int j, ukupno=0;
for (j=1; ukupno<10; j+=3)
    ukupno += j;
    ukupno /= j;
```

Rezultat: j=10 ukupno=1

---

```
int j, ukupno;
for (j=15, ukupno=0; j>10; j--)
    if (ukupno%j) ukupno += j;
```

Rezultat: j=10 ukupno=0

# For petlja

```
int i, j, x=0;
for (i=10; i>6; i-=2)
    for (j=8; j>6; --j)
        x += i - j;
```

Prolaz	i	j	x
1.v-1.u	10	8	2
1.v-2.u	10	7	5
2.v-2.u	8	8	5
2.v-2.u	8	7	6

Na kraju: i=6 j=6 x=6

v. i u. – vanjska i unutrašnja petlja

```
int i, j;
for (i=1,j=0; i<4||j<9; i++,j+=3)
    printf("%d %d ", i, j);
```

Ispis: 1 0 2 3 3 6

Na kraju: i=4 j=9

---

```
int i, x;
for (i=0, x=0; i<10; x+=i, i++){
    printf ("%d %d ", i, x);
    if (i&&x&&i%3!=1) break;
}
```

Ispis: 0 0 1 0 2 1

Na kraju: i=2 x=1

# Naredba `switch`

```
int x=0;
switch(x=1){
    case 0: x*=100;
    case 1: x+=10;
    case 2: x+=10;
    default: x/=10;
}
```

Rezultat:        `x=2`

```
switch('c'-'a'){
    case 1: printf("b");
    case 2: printf("c");
    case 3: printf("d");
    default: printf("*");
}
```

Ispis:            `cd*`

# Naredba `switch`

```
int x=1, y=5;
++x;
switch(y%x){
    case 0: y+=x;
    case 1: y-=x;
    case 2: y--; break;
    default: y/=5;
}
```

Rezultat:        x=2     y=2

```
char z='C' ;
switch(z-'A'){
    case 1:
        printf("%d ", z++);
    case 2:
        printf("%c=%d", z, z);
    case 3:
        printf(",z=%d", z+=2);
        break;
    default: printf("*");
}
```

Ispis:            C=67, z=69

# Petlje za kraj

```
i=0;
while(++i){
    if (i%3==0) continue;
    if (!(i%10)) break;
    printf("%d ",i);
}
```

Za 10. prolaza izvršava se **break**, a 3 puta se izvršava **continue**.

Ispis: 1 2 4 5 7 8

Na kraju: i=10

```
for(i=j=1; i!=0; i++){
    if (i%2) continue;
    if (i+j>10) break;
    j+=2;
    printf("%d %d ",i, j);
}
```

Za 10. prolaza izvršava se **break**, a 4 puta se izvršava **continue**.

Ispis: 2 3 4 5

Na kraju: i=6 j=5



# Rezultat izvršavanja koda

Vrijednosti varijabli na kraju?

```
int a, b=0, c=2, d=4;
a = ++b > c ? d : --c;
if (d = 4) a += b;
if (d == 4) a -= b;
printf("a=%d b=%d c=%d
d=%d\n", a, b, c, d);
```

```
/* a=1 b=1 c=1 d=4 */
```

Što se i koliko ispisuje?

```
int i = 1;
for(;;i++) {
    if(i % 5 == 0) {
        printf("%d, ", i);
        continue;
    }
    if(i % 25 == 0) {
        i=0;
        break;
    }
}
```

```
/* 5, 10, 15, 20, 25, 30, ...
bez prestanka */
```

# goto

```
int x=1, y=1;
do {
    if (x>3) goto dosta;
    ++x;
    y *= x + 2;
} while (y%x);
dosta:
```

Prolaz	x	y	y%x
1.	2	4	0

Rezultat: x=2 y=4

```
int x = 1, y = -2;
do {
    if (x==y) goto van;
    x++;
    y += x;
    printf("%d %d ", x, y);
} while (x<4 || y);
van:
```

2 0 3 3

# goto

```
char z='1';
do {
    if (!('5'-z)) goto dno;
    z+=2;
    printf("%c %d ", z, z);
} while (1);
```

dno:

Ispis:            3 51 5 53

---

```
int x=9;
gore:
do {
    if (x/3) goto gore;
    x-=2;
} while (x>5);
```

Beskonačna petlja!

```
int a=10, b=5, c=5;
prvo:
if (a-b-c) goto zadnje;
b -= c-a;
if (a-b) goto drugo;
c += a+b;
drugo:
if (b) goto prvo;
a -= b+c;
goto prvo;
zadnje:
```

Nakon svega:

a=10 b=10 c=25

# goto

```
i=0;
while(++i){
    if (i%3==0) goto dalje;
    if (!(i%10)) goto stani;
    printf("%d ",i);
    dalje:
}
stani:
```

Za 10. prolaza izvršava se `goto stani`, a  
3 puta se izvršava `goto dalje`.

Ispis: 1 2 4 5 7 8

Na kraju: i=10

```
for(i=j=1; i!=0; i++){
    if (i%2) goto ponovi;
    if (i+j>10) goto prekini;
    j+=2;
    printf("%d %d ", i, j);
    ponovi:
}
prekini:
```

Za 10. prolaza izvršava se `break`, a  
4 puta se izvršava `continue`.

Ispis: 2 3 4 5

Na kraju: i=6 j=5

# Na kraju

## Vrijednosti varijabli?

```
int a=0,b=0;  
if(a=b) a-=1; b+=1;  
if(!b) b-=1; a+=1;
```

```
/* a=1 b=1 */
```

## Vrijednosti varijabli?

```
int a=1, b=1, c=1;  
while (a) {  
    for (; b<11; b++) {  
        if (b%2)  
            break;  
        else  
            continue;  
        a--;  
        c+=10;  
    }  
    c++;  
    if (c>10) break;  
}  
/* a=1 b=1 c=11 */
```

# PiPI

RJEŠENJA VJEŽBI ZA BLITZ 04

Grupa 07,  
Z. Šimić, 2008.

# Teme za 4. *blitz*

- Jednodimenzionalnih polja
  - Deklaracija, dodjeljivanje početnih vrijednosti (bez znakovnih polja)
  - Korištenje (pristupanje članovima polja, indeksni izrazi)
  - Znakovna polja, dodjeljivanje početnih vrijednosti nizu znakova
  - Algoritmi
    - s numeričkim poljima
    - sa znakovnim poljima
    - složeniji
- Deklaracija dvodimenzionalnih i višedimenzionalnih polja i dodjeljivanje početnih vrijednosti dvodimenzionalnim poljima
- Jednostavniji algoritmi s dvodimenzionalnim poljima
- Zauzeće memorije varijablama i poljima (sizeof, ručno brojanje, procjena)
- Učitavanje polja i ispis polja (jednostavni formati kao npr. %5d, %15.7f, %s, %c)
- Pokazivači
  - Definiranje, tipovi i inicijalizacija
  - Korištenje (pristup podatku, izmjena podatka, aritmetika)

# Polja – kratko ponavljanje

- Broj elemenata u polju nije ograničen
- Indeks prvog ili početnog elementa polja je uvijek **0**
- Upisivanje i čitanje elementa polja izvan polja je moguće, ali krivo i nepredvidivih posljedica:
  - takvo upisivanje može ‘srušiti’ program
  - takvo čitanje daje vrijednost ovisnu o stanju memorije
- Indeks zadnjeg elementa polja za jedan je manji od broja elemenata polja
- Polje može sadržavati elemente bilo kojeg tipa podataka (char, int, float), ali samo jednog tipa!
- Polje je moguće inicijalizirati kod definiranja, kompletno ili dijelom (kod djelomične inicijalizacije svim ne inicijaliziranim elementima polja pridružuje se **0** )



## Definiranje, inicijaliziranje i indeksiranje polja

<code>int a[5] = {4}, i;</code>	<b>Ispis/posljedica:</b>
<code>printf("%d\n", a[4]);</code>	0
<code>printf("%d\n", a[5]);</code>	<i>indeksirano izvan polja: ispis nepredvidiv</i>
<code>a[5]=1; a[-1]=1;</code>	<i>1 se upisuje na krivo mjesto u memoriji - opasno!</i>
<code>for (i=0; i&lt;5; i++)     printf("%d ", a[i]);</code>	4 0 0 0 0
<code>for (i=0; i&lt;5; i++) {     a[i]=(4-i)*10;     printf(" %d", a[i]); }</code>	40 30 20 10 0
<code>for (i=0; i&lt;5; i++) {     a[i]/=10;     printf(" %d %d", a[i], a[i/2]); }</code>	4 4 3 4 2 3 1 3 0 2

## Definiranje, inicijaliziranje i indeksiranje polja

Ispis:

---

```
char z[] = { '0', 'a' };  
int c[2][3]={0};  
printf("%c%c%d\n", z[0], z[1], c[1][2]);
```

---

0a0

```
int p[3][2]={9, 8, 7, 6, 5};  
printf("%d %d\n", p[0][1], p[2][1]);
```

---

8 0

```
int m[4][3]={8,9,0,9,8,7,6,5,4,3,2,1};  
printf("%d %d\n", m[3][1], m[1][2]);
```

---

2 7

```
int n[4][3]={ {0}, {2}, {6,5} };  
printf("%d %d\n", n[0][2], n[2][1]);
```

---

0 5

```
int a[3][4]={9,8,7,6,5,4,3,2,1};  
printf("%d %d\n", m[0][3], m[2][0]);
```

---

6 1

## Definiranje, inicijaliziranje i indeksiranje polja

### Ispravno ili ne:

```
char z[4] = { 'a', 'b', 99, 100};
```

Ispravno

```
unsigned int i[4][0][1][2][3];
```

*0 nedopuštena dimenzija*

```
int i[6]={1, 2, 3, 4, 5, 6};
```

Ispravno

```
char c[] = {};
```

*Kriva sintaksa.*

```
float f[4]={5, 3., 1.};
```

Ispravno

```
int i_1[1][2]={1, 2, 3, 4};
```

*Previše inicijal. elemenata.*

```
float _f[4]={2.f, 4.f, 6.f, 8.f};
```

Ispravno

```
int [2,2] = {1};
```

*Kriva sintaksa.*

```
int i=0, e;
```

```
float a[4]={0}, b[4], c[e]={i};
```

```
for(i=1;i<=4;i++) b[i-1]=0;
```

- polja **a** i **b** su ispunjena 0-ma,  
polje **c** ima sintaksne pogreške

Koliko polje zauzima memorije?

Odgovor:

---

```
char z[7] = { 'a', 'b' };
```

7 bytes

```
long int i[10];
```

40 bytes

---

```
int i[]={1, 2, 3, 4, 5};
```

20 bytes

```
char c[5+1];
```

6 bytes

---

```
float f[50][2];
```

400 bytes

```
long a[10][5][2];
```

400 bytes

---

```
double d[20][50];
```

8000 bytes

```
short s[10][10] = {1, 2};
```

200 bytes

# 1D polja u programu

## Dio programa:

```
int a[]={1, 2, 3, 4, 5}, i=3;
```

```
do {
```

```
    a[i] = a[i-1]-a[i+1];
```

```
    printf(" %d", a[i]);
```

-2 4 -3

```
} while(--i>0);
```

```
/* program koji slijedi nastavak je programa iznad! */
```

```
for (i=4; i>=1; i--)
```

```
    printf("%d", a[i]=i*i);
```

```
printf("a[%d]=%d", i, a[i]);
```

16941a[0]=1

```
for (i=4; i>2&& i!=1; i/=2)
```

```
    scanf("%d", &a[i]);
```

Učitava jednu  
vrijednost; za a[4]

# 2D polja u programu

Dio programa:

```
int a[2][10], x=0, y=0, i;  
for (i=9; i!=0; i--) {  
    a[0][i] = i;  
    a[1][i] = i*2;  
    y += a[0][i];  
    x += a[1][i] - a[0][i];  
}
```

Vrijednost/ispis:

**Nakon petlje:**

**y==45**

**x==45**

---

```
int n[4][4]={ {1}, {1,2}, {3,4,5} };  
int b=0, c=0, i, j;  
for (i=0; i<4; i++){  
    b += n[i][i];  
    for (j=0; j<4; ++j)  
        c += n[i][j];  
}
```

**Nakon petlji:**

**b == 8**

**c == 16**

# 2D polja u programu

Dio programa:

```
int m[9][9], z=0, i;  
for (i=0; i<5; i++)  
    m[i][8-i] = m[8-i][i] = 1;  
for (i=0; i<9; i++) z+=m[i][8-i];
```

Rezultat:

**Nakon petlje:**  
Postavlja elemente na glavnoj  
dijagonali na 1.  
**z == 9**

---

```
int n[6][6], s=0, i, j;  
for (i=0; i<6; i++)  
    for (j=5-i; j<6; ++j){  
        n[i][j]= 1;  
        s += n[i][j];  
    }  
}
```

**Nakon petlji:**  
Postavlja elemente polja ispod  
sporedne dijagonale na 1.  
**s == 15**

# Pokazivači

## Pokazivači

- varijable koje sadrže memorijsku adresu
- pored adrese deklaracijom je određen tip podatka na koji pokazivač pokazuje
- operator `*` omogućava čitanje i spremanje vrijednosti na pokazivanu adresu preko pokazivača
- operator `&` omogućava pridruživanje adrese bilo koje (uključujući pokazivače) varijable pokazivaču
- svaki pokazivač zauzima 4 bajta
- aritmetika nad pokazivačima mijenja vrijednost (pokazivača) u kvantima koje određuje tip podatka na koji pokazivač pokazuje

```
char *pc, c='Y' ;  
int *pi, *pk, i=1, j;  
double *pd, d=0;  
pc = &c;  
pi = &i;  
pk = pi;  
pd = &d;  
*pc = 'N' ;  
j = *pi;  
*pk = 0;  
*pd = 2;
```

### Stanje na kraju:

`c=='N'    i==0    j==1    d==2.0`



# Aritmetika pokazivača

## **Dozvoljeno je i smisleno**

- Zbrajati i oduzimati cijeli broj od pokazivača
- Za dva pokazivača na isto polje:
  - oduzimati ih
  - uspoređivati ih ( $\leq$  i  $\geq$ )

## **Nema smisla pridruživati pokazivaču rezultat:**

- množenja, dijeljenja ili % (mod) operacija dvaju cijelih brojeva

## **Nije dozvoljeno**

- Zbrajanje dvaju pokazivača
- Množenje, dijeljenje i % (mod) među pokazivačima

# Pokazivači

```
short n=1, m=2, k;  
short *pi = &n, *pj = &m, *pp;  
pp = &k;  
*pp = *pi;  
*pi = m;  
m = *pp;  
*pp = k * m + n;  
pp = pi;  
printf("%d %d %d\n", n, m, k);  
printf("%d %d %d", *pi, *pj, *pp);  
printf("\n%d ", *pi + *pj);  
printf("\n%d ", sizeof(n));  
printf("\n%d \n", sizeof(pi));  
n = sizeof(n) + sizeof(pj);  
printf("\n%d %d", n, n + *pi);
```

Ispis:

2 1 3

2 1 2

3

2

4

6 12

# Pokazivači

```
double x=3., y=4., z;  
double *pa=&x, *pb=&y, *pd;  
pd = &z;  
*pa += *pb;  
*pb = *pa - y;  
x = *pa - *pb;  
*pd = x * y ;  
pd = pb;  
printf("%f %f %f\n", x, y, z);  
printf("%f %f %f", *pa, *pb, *pd);  
printf("\n%f ", *pb + *pd);  
printf("\n%d ", sizeof(x));  
printf("\n%d \n", sizeof(pa));  
z = sizeof(x) + sizeof(pb);  
printf("\n%f %f", z, z + *pd);
```

Ispis:

4.0 3.0 12.0  
4.0 3.0 3.0  
6.0  
8  
4  
12.0 15.0

# Pokazivači – radi, ali nema smisla:

Ispis:

```
int n;  
int *pi = &n, *pj;  
double x = 5.;  
double *pa = &x, *pb;
```

Neka je početna adresa varijable **x**  
4377768 (0042CCA8<sub>16</sub> )

```
pj = (int *)x;  
printf("%lu %p\n", pj, pj);  
*pa = 100 * (int)(&x);  
printf("%p %p\n", pa, &x);  
printf("%lu %lu\n", pa, &x);  
printf("%f %f\n", *pa, x);  
printf("%lu", 10 * (int)(&x));
```

5 00000005

0042CCA8 0042CCA8

4377768 4377768

437776800.0 437776800.0

43777680

---

**%p** – format za heksadski ispis adrese  
**%lu** – format unsigned long  
za cjelobrojni ispis bez predznaka

Adresa varijable **x** je proizvoljna i  
rezultat se može mijenjati kod  
ponovnog izvođenja!

# Pokazivači – aritmetika

Neka je početna  
adresa polja **ar**  
1000016!

Ispis:

```
float ar[3]={1.1, 2.2, 3.3};
float *pa, *pb;
char az[40], *pz=&az[0];
pa = pb = ar;
printf("\n%lu %f\n", pa, *pa);      1000016 1.100000
pb += 2;
printf("%lu %f\n", pb, *pb);        1000024 3.300000
printf("%lu %f\n", ar+2, *(ar+2));  1000024 3.300000
printf("%lu %f\n", &ar[2], ar[2]);  1000024 3.300000
printf("%d\n", pb-pa);              2
printf("%d\n", (int)pb-(int)pa);     8
printf("%lu %f\n", pb-1, *(pb-1));  1000020 2.200000
pz += 4;
printf("%d\n", pz-az);              4
printf("%d\n", (int)pz-(int)az);     4
pz = (char *) ((int *) az +4);
printf("%d\n", pz-az);              16
printf("%d\n", (int)pz-(int)az);     16
```

# PiPI

## RJEŠENJA VJEŽBI ZA BLITZ 05

Grupa 07,  
Z. Šimić, 2008.

# Teme za 5. blitz

- Definicija funkcije
- Naredba `return`
- `void` funkcije i funkcije bez argumenata
- Prijenos kopija vrijednosti (bez polja)
- Prijenos referencija-adresa (bez polja)
- Formalni i stvarni argumenti (izrazi, redoslijed, tipovi pri pozivu funkcije) Ne spominjati stog!
- Prototipovi funkcija, organizacija složenijih programa
- Smještajni razredi (postojanost, područje važenja varijabli)  
Samo elementarni pojmovi, po mogućnosti bez register i external!
- Jednodimenzionalna polja kao argumenti funkcije
  - rad s indeksnim izrazima
  - rad s pokazivačima
- Dvodimenzionalna polja kao argumenti funkcije
  - rad s indeksnim izrazima
  - rad s pokazivačima

# Funkcije

- Funkcije su korisne
  - bolja preglednost koda
  - mogu smanjiti kod
  - neograničeno pozivanje
  - lakše ispravljanje koda
- Ne ubrzavaju uvijek program
- *Call by value*
  - može mijenjati vrijednosti argumenata koji **nisu** vidljivi nakon izlaska iz funkcije
- *Call by reference*
  - može mijenjati vrijednosti argumenata koji su vidljivi nakon izlaska iz funkcije
- *Default* tip funkcije je **int**
- Prototip funkcije
  - pomaže otkrivanju krivog poziva funkcije
  - smješta se na vrh datoteke ili zajedno s drugim prototipovima u posebnu datoteku koja se referencira s `#include <ime.h>`
  - nije nužan ako se funkcije poziva nakon što je opisana



# *void* funkcija

- *void* funkcija
  - ne vraća vrijednost
  - može ali ne mora imati argumente
- sintaktički je ispravno:  
**`void nako() { ; }`**
  - najjednostavniji ispravni oblik
  - može se koristiti **`return`** bez argumenta unutar *void* funkcije

```
void neka(int i) {  
    printf("2i=%d", 2*i);  
}
```

- sintaktički je pogrešno:
  - pozivati *void* funkciju očekujući rezultat:  
**`neka(neka(10)) ;`**
  - koristiti **`return`** s argumentom unutar *void* funkcije

# *void* funkcija

Dio programa:

```
void fn2(int n){
    printf("fn2- %d ", 2*n);
}
void fn1(int n){
    printf("fn1- %d ", 2*n);
    fn2(--n);
}
void main(){
    fn1(2);
}
```

Rezultat/ispis:

**fn1- 4 fn2- 2**

---

```
void uradi(int m) {
    int i, zbroj=0;
    for(i=0; i<m; i++)
        if(i%3) zbroj+=i;
    printf("%d ", zbroj);
}
```

*void* funkcija **uradi**:

- sumira sve brojeve manje od **m** koji nisu djeljivi sa **3** i
- na kraju ispisuje rezultat

# Formalni i stvarni argumenti funkcija

Definicije (tri reda na početku) vrijede za sve pozive poslije

```
int n, m;
```

```
char z;
```

```
double r, u;
```

## Prototip funkcije:

---

```
m=uf(z,n,3*r,2e-4);  int uf(char, int, double, double);
```

---

```
uh(m,u,z-32);
```

```
void uh(int i, double d, char c);
```

```
void uh(int, double, char);
```

---

```
oj(10.f, 3., 45);
```

```
void oj(float f, double d, char c);
```

```
void oj(float, double, int);
```

---

```
u = ah(n,&r,&z);
```

```
double ah(int i, double *d, char *c);
```

---

```
printf("\n%d", eh(u,z,m));
```

```
int eh(double, char, int);
```

## Što nam znači prototip funkcije?

---

```
void fu(char c, int *n);
```

-može vratiti rezultat preko pokazivača **n**

---

```
int *bu(char *z);
```

-funkcija vraća pokazivač na cijeli broj i ima pokazivač na niz znakova kao ulazni argument

---

```
mu(int *n);
```

- funkcija vraća cijeli broj (*default* tip) i prima ulazni pokazivač na cijeli broj kao ulazni argument

---

**Kako glasi prototip funkcije koja treba izračunati sumu cijelih brojeva u nekom intervalu?**

```
long su(int d, int g); ili  
void su(long *s, int d, int g);
```

---

# Call by value

Program:

```
char gore(int n) {  
    n+10;  
    return n;  
}  
  
void ravno(int n) {  
    n-=10;  
}  
  
void main() {  
    int n=10;  
    n = gore(n) ;  
    ravno(n) ;  
    printf("n=%d", n);  
}
```

Rezultat/ispis:

-funkcija **gore** zapravo ne mijenja **n**, ali ima problem za veliki ulazi argument, jer je povratna vrijednost tipa `char`, a primljena vrijednost tipa `int`!

-funkcija **ravno** zapravo ne radi ništa jer je `void` i prima vrijednost varijable!

**n=10** nakon izvršavanja **gore(n)**

**n** opet nepromijenjen nakon **ravno(n)**

**n=10** ispis na kraju

# Call by value

Program:

```
void ispisi(int n){
    n--;
    printf("ni=%d ", n);
}

void main(){
    int n=10;
    ispisi(n);
    ispisi(++n);
    ispisi(n++);
    n-=2;
    printf("nm=%d ", n);
}
```

Rezultat/ispis:

-funkcija `ispisi` ne mijenja `n`, jer prima vrijednost od `n` te istu umanjenu za 1 ispisuje!

`ni=9 ni=10 ni=10 nm=10`

# Call by value

## Program:

```
int fdp(int n) {
    n+=5;
    return n;
}

void main() {
    int n=0, m, k;
    n = fdp(n);
    fdp(m=0);
    k=fdp(fdp(m-n));
    printf("%d%d%d", n, m, k);
}
```

## Rezultat/ispis:

-funkcija **fdp** vraća vrijednost **n**,  
uvećanu za 5

**n=5** nakon izvršavanja **fdp(0)**

-ovdje se ne mijenja ništa, **fdp(0)**  
opet vraća 5, ali se to ne sprema  
nigdje

**k=5** nakon izvršavanja

**fdp(fdp(-5)) = fdp(0) = 5**

505

# Jednostavne funkcije

## Program:

```
int fp(int n, int m){
    return n*m;
}

int fm(int n, int m){
    return n%m;
}

void main(){
    int n=5, m=2, k=6;
    n = fp(fm(n,m),k);
    m = fm(5,fp(m,1));
    k = fp(fp(2,3),fm(3,2));
}
```

## Rezultat:

-funkcija **fp** vraća produkt  
ulaznih vrijednosti

-funkcija **fm** vraća produkt  
ulaznih vrijednosti

$n = fp(1, 6) = 6$

$m = fm(5, 2) = 1$

$k = fp(6, 1) = 6$



# Jednostavne funkcije

Program:

```
int lim(int m){  
    if (m<10) return 10*m;  
    return -10*m;  
}
```

```
int mil(int n){  
    return n%12;  
}
```

```
double rec(int m){  
    return 10./m;  
}
```

```
void main(){  
    int n=0, m=10;  
    n = lim(n+m);  
    m = mil(m+8);  
    printf("%f",rec(n));  
}
```

Rezultat:

**Za početno kao u programu  
(n=0 i m=10) rezultat je:**

n = -100

m = 6

ispis: -0.100000

**Za početno n = m = 1 rezultat je:**

n = 20

m = 9

ispis: 0.500000

# Stog i varijable

- Stog sadrži
  - povratnu adresu (4 bajta)
  - argumente funkcije i sve lokalne automatske varijable (prema tipovima)
  - okvir stoga (ne broji se)
- Stog ne sadrži
  - lokalne konstante
  - statičke lokalne varijable
  - globalne varijable

## Primjer:

```
void nema() {}  
void main() {  
    nema();  
}
```

- poziv funkcije **nema()** zauzima na stogu memoriju potrebnu za povratnu adresu

# Stog – zauzeće memorije

## Program:

```
int fdr(long n, char z){
    short k=5*n/z;
    return k;
}

double fpr(int m){
    return 1.*m/fdr(300,'*');
}

void main(){
    double dupla;
    dupla = fpr(10000);
}
```

## Rezultat:

- Prvo se poziva funkcija fpr iz main-a
  1. <na stog ide povratna adresa i int vrijednost>,
- potom se iz fpr poziva fdr
  2. <na stog se dodaje još jedna povratna adresa te vrijednost za long i char>,
  3. unutar fdr na stog se dodaje short varijabla k.
- Tu je sve gotovo i maksimalno zauzeće stoga, bez tzv. okvira, je u bajtima:
  - $4 + 4 + 4 + 4 + 1 + 2 = 19$  bajta

# *Call by reference – bez polja*

Program:

```
void pomakni(char *c) {  
    *c += 2;  
}  
  
void main () {  
    char c = 'a', z = 'b';  
    pomakni(&c);  
    pomakni(&z);  
    printf ("c='%c'\n", c);  
    printf ("z='%c'", z);  
}
```

Rezultat/ispis:

```
c='c'  
z='d'
```

# Call by reference – bez polja

Program:

Rezultat/ispis:

```
int fopa(char *z, short t) {
    (*z)--;
    t-=2;
    return (*z) * t;
}

main () {
    char c = '3';
    int i;
    short t = 3;

    i = fopa (&c, t);
    printf ("%d %d\n", i, t);          50 3
    printf ("%d %c\n", c, c);        50 2
}
```

## Call by reference – bez polja

Program:

Rezultat/ispis:

```
int puk(int *i, int j) {  
    int k;  
    k=*i * j;  
    (*i)++;  
    j--;  
    *i *= j;  
    printf ("f: %d ", *i);  
    printf ("%d %d", j, k);  
    return *i;  
}
```

f: 8 4 5

```
main () {  
    int i, j=5, k=1;  
    i = puk (&k, j);  
    printf ("\nm: %d ", i);  
    printf ("%d %d", j, k);  
}
```

m: 8 5 8

# Call by reference – 1D polje

Program:

Rezultat/ispis:

```
int puni(int *ari, int n) {  
    int i=0, p=5;  
    for(;i<n;i++) ari[i]=p;  
    return p;  
}
```

```
main () {  
    int ari[20];  
    int i=20, j;  
    j = puni(ari, i);  
    printf ("%d %d", *ari, ari[0]);  
    printf ("\n%d", j);  
}
```

5 5  
5

# Call by reference – 2D polje

Program:

Rezultat/ispis:

```
void fda(int *ari, int n, int m, int stupmax) {  
    printf("%d ", *(ari+n*stupmax+m));  
    printf("%d \n", ari[n*stupmax+m]);  
}
```

9 9

```
void fdb(int *ari, int n, int m, int stupmax) {  
    int i=0, j;  
    for (;i<n;i++)  
        for (j=0;j<m;j++)  
            printf("%d ", ari[i*stupmax+j]);  
}
```

1 2 3 4 5 6 7 8 9

```
main () {  
    int ari[5][10]={{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  
    int i=3, j=3, k=10;  
    fda(&ari[0][0], i-1, j-1, k);  
    fdb(ari[0], i, j, k);  
}
```



# PiPI

RJEŠENJA VJEŽBI ZA BLITZ 06

Grupa 07,  
Z. Šimić, 2008.

## Teme za 6. blitz

- Macro s parametrima
- Matematičke ugrađene funkcije
- Vlastite funkcije za rad s nizovima
  - deklariranim kao polje
  - deklariranim kao pokazivač
- Ugrađene funkcije iz `string.h`
- Ugrađene funkcije iz `ctype.h`
- Ulaz/izlaz (`gets`, `puts`)
- Formatni za ispis (`printf`)
- Formatni za unos (`scanf`)
- Ulaz/izlaz (`getchar`, `putchar`)
- `typedef` (bez strukture)

# Macro s parametrima

Program:

```
#define ta(a,b,c) a*b*c
#define tb(a,b,c) (a)*(b)*(c)
#define tc(a,b,c) ((a)*(b)*(c))
void main(){
    int x, y, z;
    int n=0, m=2, k=1;
    x = !ta(n,m,k);
    y = !tb(n,m,k);
    z = !tc(n,m,k);
    x = ta(n,m,k);
    y = tb(n,m,k);
    z = tc(n,m,k);
}
```

Rezultat:

Zagrade!!!

x = 2

y = 2

z = 1

x = 0

y = 0

z = 0

# typedef (bez strukture)

## typedef

- služi za definiranje korisničkih tipova podataka
- za ime korisničkog tipa nije dozvoljeno koristiti postojeće tipove podataka, ključne riječi niti velika slova
- novi tip se može definirati korištenjem prije definiranog tipa
- korisnički tip ne može biti novi tip podataka koji nije podržan u C-u
- zauzeće memorije za varijable definirane korisničkim tipom određeno je prema izvornom tipu podatka, npr.  

```
typedef long red;  
red a1[10];
```

**polje a1 zauzima 40 bajta**
- korisnički tip može biti i pokazivač, npr.:  

```
typedef double *p2d;
```

# Pregled obrađenih ugrađenih funkcija

## Matematičke funkcije <math.h>

```
double fabs(double arg);  
double pow(double arg, double exp);
```

## Posebne funkcije <stdlib.h>

```
void exit(int stanje);  
void srand(unsigned int sjeme);  
int rand(void);
```

## Standardne funkcije <stdio.h>

```
int getchar(void);  
int putchar(int ch);  
char * gets(char * str);  
int puts(const char * z);  
int printf(const char * format, arg1, ..., arg n);  
int scanf(const char * format, arg1, ..., arg n);
```

## Funkcije znak. niza <string.h>

```
int strlen(const char * str);  
char * strcpy(char * str1, const char * str2);  
char * strncpy(char * s1, const char * s2, size_t makslen);  
char * strcat(char * str1, const char * str2);  
int strcmp(const char * str1, const char * str2);  
int strncmp(const char * s1, const char * s2, size_t makslen);  
char * strchr(const char * str, int ch);  
char * strstr(const char * str1, const char * str2);  
int toupper(int ch);  
int isalpha(int ch);  
...
```

## Znakovne funkcije <ctype.h>

```
int isdigit(int ch);    int isalpha(int ch);  
int isalnum(int ch);  
int islower(int ch);    int isupper(int ch);  
int tolower(int ch);    int toupper(int ch);
```

# Formati za unos (`scanf`)

- **%s**
  - učitavanje u polje znakova sve do unosa praznine tab-a ili enter-a
- **%6s**
  - učitavanje u polje znakova najviše 6 znakova sve do unosa praznine tab-a ili enter-a
- **%[ samo]**
  - učitavanje u polje znakova sve do unosa znaka koji nije ' ', 's', 'a', 'm' 'o'
  - upisivanje tab-a i enter-a također znači kraj učitavanja
- **%[^NeTo]** ili **%[^N^e^T^o]**
  - učitavanje u polje znakova sve do unosa nekog od znakova 'N', 'e', 'T' ili 'o'
  - upisivanje praznine, tab-a i enter-a također znači kraj učitavanja
- **%c**
  - učitavanje jednog znaka
- **%d**
  - učitavanje cijelog broja sve do unosa praznine tab-a ili enter-a
- **%2d%3d**
  - učitavanje dva cijela broja jednog s dvije znamenke i drugog s tri znamenke, bez znaka između ili do unosa praznine tab-a ili enter-a
- **%o**
  - učitavanje oktalnog broja sve do unosa praznine tab-a ili enter-a
- **%x**
  - učitavanje heksadecimalnog broja sve do unosa praznine tab-a ili enter-a

# Formati za unos (scanf)

```
char slova[10];
```

```
scanf("%6s", slova);
```

Za unos: Primjer za  
slova="Primje"

```
scanf("%[ samo]", slova);
```

Za unos: samo se  
slova="samo s"

```
scanf("%[^NeTo]", slova);
```

Za unos: nEtONE  
slova="nEtO"

```
char slova[10];
```

```
int i, j;
```

```
scanf("%2d%3d", &i, &j);
```

Za unos: 987653  
i=98, j=765

```
scanf("%2d %s", &i, slova);
```

Za unos: 34 Neki unos  
i=34, slova="Neki"

```
scanf("%o", &i);
```

```
printf("x%X d%d o%o", i, i, i);
```

Za unos: 17  
x0F d15 o17

# Formati za ispis (printf)

## Ispis:

```
char at[] = "\n.....\n";
int m = 8, n=100, d = 0xa;
float x = 1.61803, y=-3.14159, q = 2.718;
char ac[20] = "NEKI TEKST", az[]="Znakovi";
printf("%s|%-3d %05.3f %.3s", at, m, x, &ac[5]);
printf("%s|n=%05d, y=%07.3f", at, n, y);
printf("%s|%4.1f %4.2f %4.0f", at, q, q, q);
printf("%s|Broj %.0f.", &at[0], q);
printf("%s|%-10s%-3X-%+3x%3d", at, az, d, d, d);
printf("%s|%-10s %04d %6.3f", at, "DA", 987, -y);
printf("%s|%5.2f%-6.4s", at, 10*x, "zlatni");
printf("%s|%-3d %03d", at, m, -m);
printf("%s|%03d%-5.1f", at, 2, 3.14);
printf("%s|%04d%4d%02d%2d", at, 4, 4, 20, 20);
printf("%s|%05.2f%5.3s\n", at, 1.2345, "Simbol");
```

.....  
|8 1.618 TEK  
.....  
|n=00100, y=-03.142  
.....  
| 2.7 2.72 3  
.....  
|Broj 3.  
.....  
|Znakovi A - a 10  
.....  
|DA 0987 3.142  
.....  
|16.18zlat  
.....  
|8 -08  
.....  
|0023.1  
.....  
|0004 42020  
.....  
|01.23 Sim



# Ugrađene funkcije iz string.h

```
#include <string.h>
...
char st[] = "neki tekst", *nz="znakovi-";
char slova[20];
strncpy(slova, st, 4);
printf("%s\n", slova);
slova[4]='\0';
printf("%s\n", slova);
printf("%s\n", strcpy(slova, nz));
printf("%s %d\n", slova+4, strlen(&slova[4]));
strcpy(slova, nz);
printf("%s\n", strstr(st, "tek"));
printf("%s\n", strstr(st, "t")+1);
printf("%c\n", *(strstr(st, "t")+2));
printf("%s\n", strcat(slova, st));
printf("%s\n", strchr(st, 'i'));
printf("%d\n", strcmp(st, nz));
printf("%d\n", strcmp("abc", "ABC"));
```

## Ispis:

neki??????? Nedefiniran  
ispis jer nema oznake kraja  
niza

neki

znakovi-

ovi- 4

tekst

ekst

k

znakovi-neki tekst

i tekst

-1

1

zš)

# Ugrađene funkcije iz ctype.h

Ispis:

```
#include <ctype.h>
```

```
int i=0, v=0, m=0;
```

```
char c='c', z='C', s='1';
```

```
char niz[]="Testiranje 123 ABCD";
```

```
printf("%d %d\n", isupper(z), isalpha(s));
```

1 0

```
printf("%d %c\n", isupper(c), toupper(c));
```

0 C

```
for(; niz[i]!='\0'; i++){  
    if (islower(niz[i]) m++;  
    v += isupper(niz[i]);  
    niz[i] = toupper(niz[i]);  
}
```

```
printf("%d %d\n", v, m);
```

5 9

```
printf("%s\n", niz);
```

TESTIRANJE 123 ABCD

```
printf("%d %c\n", tolower(z), tolower(z));
```

99 c

```
printf("%c\n", z);
```

C

# Ulaz/izlaz (getchar, putchar)

Ispis:

```
int i;
char niz[]="123 abc ABC";
char *ps = "pointer na string";

i=strlen(niz);
while (isupper(niz[--i])) {
    putchar(niz[i]+32);
    if (niz[i]=='A') niz[i]='a';
    if (islower(niz[i])) break;
}
putchar('\n');
for(i=0; i<strlen(niz); i++)
    putchar(*(niz+i));
putchar('\n');
for (; *ps; ps++)
    if (*ps >= 'd' && *ps <= 'r')
        putchar(*(++ ps));
printf("\n%s", ps-4);
```

cba

123 abc aBC

onraig

ring

za blitze

## Ulaz/izlaz (getchar, putchar)

```
void fprva(int i, char str[]) {  
    for(; str[i]; i++)  
        if (isdigit(str[i])) putchar(str[i]);  
}
```

Ispis:

123

```
void fdruga (int i, char *str) {  
    for (i=strlen(str)-1; i>=0; i--)  
        putchar(*(str+i));  
}
```

321 CBA :T

```
char *ftreca(int i, char *str) {  
    do {  
        str[++i] = getchar();  
    } while( str[i] != 'X');  
    str[i] = 0;  
    return str;  
}
```

Unos s tastature:

test 1X

```
void main () {  
    char niz[]="\nT: ABC 123\n", as[40];  
    fprva(0, niz);  
    fdruga(0, niz);  
    strcpy(as, ftreca(-1, niz));  
    printf("%s %s\n", niz, as);  
}
```

test 1 test 1

# Ulaz/izlaz (gets, puts)

## Rezultat/Ispis:

```
char niz[80], z;  
char str[] = "Neki tekst", *ps;
```

```
puts(str);
```

Neki tekst

```
str[6]='\0';
```

```
puts(str);
```

Neki t

```
puts("Unesi tekst: ");
```

```
z = getchar();
```

```
ps = gets(niz)+5;
```

Unesi tekst:

jos jedan test

```
printf("%c%s\n", z, niz);
```

```
printf("%s %d", ps, strlen(niz));
```

jos jedan test

dan test 13

Uneseni  
tekst: