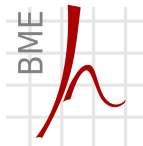


# A C nyelv aritmetikai típusai.

## A programozás alapjai I.



Hálózati Rendszerek és Szolgáltatások Tanszék  
Farkas Balázs, Fiala Péter, Vitéz András, Zsóka Zoltán

2020. szeptember 28.

# Tartalom

- 1 Gyakorló feladatok
- 2 A C nyelv aritmetikai típusai
  - Bevezetés
  - Egészek
  - Karakterek
  - Valósak

# 1. fejezet

## Gyakorló feladatok

# 1. Gyakorló feladat

Írjon C programot, mely egy egész számot ( $R$ ) olvas be a standard bemenetről, majd a standard kimeneten megjelenít egy  $10 \times 10$  mező méretű karakterábrát.

- Az ábra mezőit balról jobbra ( $x$ ) és fentről lefelé ( $y$ ), 1-től kezdve egyesével számozzuk.
- Azon mezőkbe, melyekre  $x^2 + y^2 < R^2$ , a program a '#' karaktert írja, a többi mezőt a '.' karakterrel jelölje.
- $R = 8$ -ra pl. az alábbi ábra jelenik meg:

```
#####...
#####...
#####...
#####...
#####...
#####...
#####...
###.....
.....
.....
.....
.....
```

## Megoldás

## 2. Gyakorló feladat

Írjon C programot, mely a standard bemenetére érkező egész számokat dolgozza fel.

- A program feladata, hogy képezze az összes bejövő szám abszolút értékét, majd kiírja a standard kimenetre a legkisebb és a legnagyobb érték különbségét.
- A számsor végét a 0 szám jelzi, melyet már nem kell feldolgoznia.
- Feltételezheti, hogy legalább egy feldolgozandó szám érkezik.

Megoldás

### 3. Gyakorló feladat

Írjon C programot, mely egy legfeljebb 100 valós számot tartalmazó végjeles sorozatot olvas be a standard bemenetről.

- A program feladata, hogy a standard kimeneten megadja, hogy hány olyan érték érkezett, mely nagyobb, mint a sorban tízzel korábban érkező érték.
- A sorozat végjele a 0.0 érték.

#### Megoldás

## 2. fejezet

### A C nyelv aritmetikai típusai

# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek



# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek

Valódi számítógép – véges értékkészlet

- Nem ábrázolhatunk tetszőlegesen nagy számokat

# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek

Valódi számítógép – véges értékkészlet

- Nem ábrázolhatunk tetszőlegesen nagy számokat
- Nem ábrázolhatunk tetszőlegesen pontos számokat

$$\pi \neq 3,141592654$$

# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek
- Ábrázolás

### Valódi számítógép – véges értékkészlet

- Nem ábrázolhatunk tetszőlegesen nagy számokat
- Nem ábrázolhatunk tetszőlegesen pontos számokat  
 $\pi \neq 3,141592654$
- Ismernünk kell az ábrázolható tartományokat, hogy adatainkat

# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek
- Ábrázolás

### Valódi számítógép – véges értékkészlet

- Nem ábrázolhatunk tetszőlegesen nagy számokat
- Nem ábrázolhatunk tetszőlegesen pontos számokat  
 $\pi \neq 3,141592654$
- Ismernünk kell az ábrázolható tartományokat, hogy adatainkat
  - információveszteség nélkül vagy

# Típusok – Bevezetés

## A típus

- Értékkészlet
- Műveletek
- Ábrázolás

### Valódi számítógép – véges értékkészlet

- Nem ábrázolhatunk tetszőlegesen nagy számokat
- Nem ábrázolhatunk tetszőlegesen pontos számokat  
 $\pi \neq 3,141592654$
- Ismernünk kell az ábrázolható tartományokat, hogy adatainkat
  - információvesztés nélkül vagy
  - elfogadható információvesztéssel de ne túl pazarlóan tárolhassuk

# A C nyelv típusai

- void
- skalár
  - aritmetikai
    - egész: integer, karakter, felsorolás
    - lebegőpontos
  - mutató
- függvény
- union
- összetett
  - tömb
  - struktúra

# A C nyelv típusai

- void
- skalár
  - aritmetikai
    - egész: **integer**, **karakter**, felsorolás
    - **lebegőpontos**
  - mutató
- függvény
- union
- összetett
  - tömb
  - **struktúra**
- Ma ezekről lesz szó

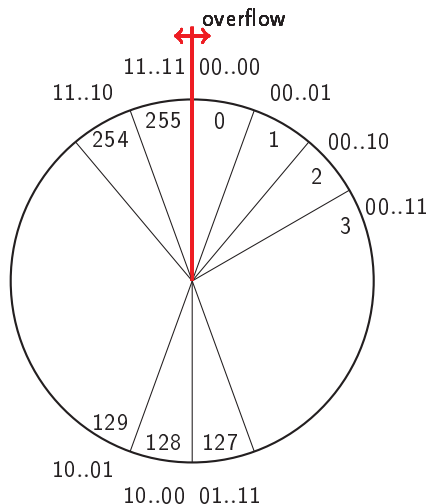
# Egészek bináris ábrázolása

## ■ 8 biten tárolt előjel nélküli egészek bináris ábrázolása

| dec | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | hex  |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|------|
| 0   | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00 |
| 1   | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0x01 |
| 2   | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0x02 |
| 3   | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 1     | 0x03 |
| ⋮   | ⋮     |       |       |       |       |       |       | ⋮     | ⋮    |
| 127 | 0     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0x7F |
| 128 | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x80 |
| 129 | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0x81 |
| ⋮   | ⋮     |       |       |       |       |       |       | ⋮     | ⋮    |
| 253 | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 1     | 0xFD |
| 254 | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 0xFE |
| 255 | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0xFF |

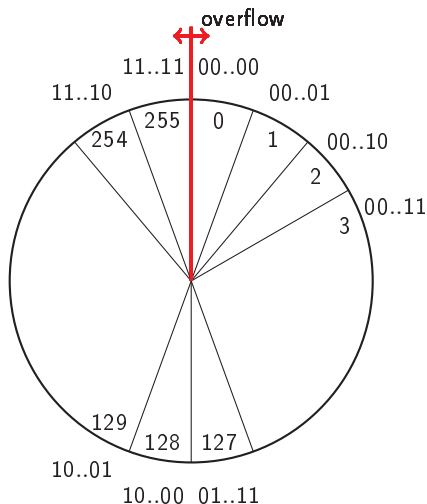


# A túlcsondulás (overflow)



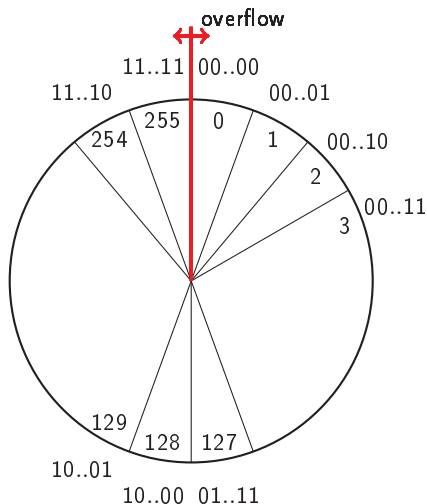
- 8 biten ábrázolt előjel nélküli számok esetén

# A túlcsordulás (overflow)



- 8 biten ábrázolt előjel nélküli számok esetén
- $255+1 = 0$

# A túlcsordulás (overflow)

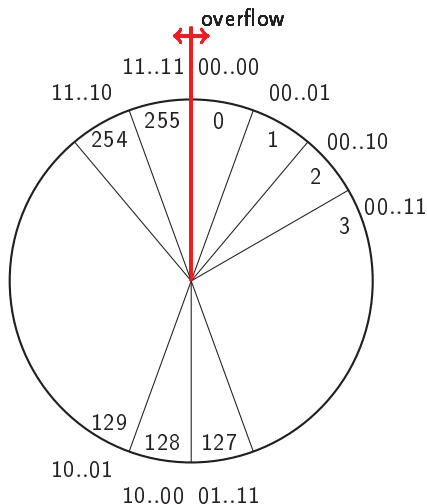


■ 8 biten ábrázolt előjel nélküli számok esetén

■  $255+1 = 0$

■  $255+2 = 1$

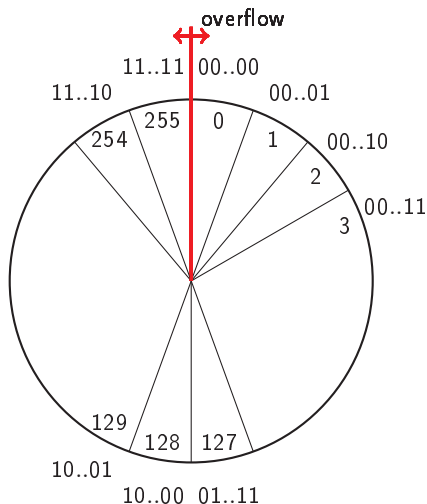
# A túlcsondulás (overflow)



■ 8 biten ábrázolt előjel nélküli számok esetén

- $255+1 = 0$
- $255+2 = 1$
- $2-3 = 255$

# A túlcsoordulás (overflow)



- 8 biten ábrázolt előjel nélküli számok esetén

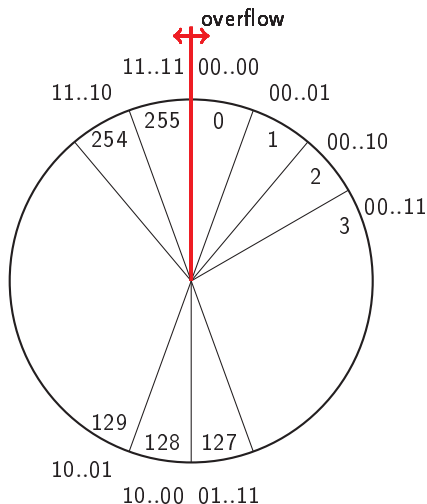
- $255 + 1 = 0$

- $255 + 2 = 1$

- $2 - 3 = 255$

- „modulo 256 aritmetika”

# A túlcsordulás (overflow)



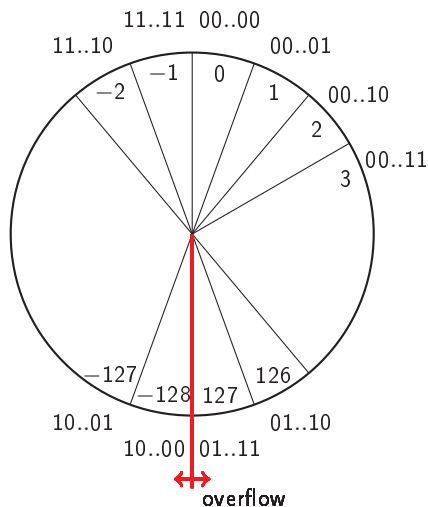
- 8 biten ábrázolt előjel nélküli számok esetén
  - $255+1 = 0$
  - $255+2 = 1$
  - $2-3 = 255$
- „modulo 256 aritmetika”
  - Mindig csak az eredmény 256-tal vett maradékát látom

# Egészek kettes komplementes ábrázolása

## ■ 8 biten tárolt előjeles egészek kettes komplementes ábrázolása

| dec  | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | hex  |
|------|-------|-------|-------|-------|-------|-------|-------|-------|------|
| 0    | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x00 |
| 1    | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0x01 |
| 2    | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0x02 |
| 3    | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 1     | 0x03 |
| ⋮    | ⋮     |       |       |       |       |       |       | ⋮     | ⋮    |
| 127  | 0     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0x7F |
| -128 | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0x80 |
| -127 | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0x81 |
| ⋮    | ⋮     |       |       |       |       |       |       | ⋮     | ⋮    |
| -3   | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 1     | 0xFD |
| -2   | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 0xFE |
| -1   | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0xFF |

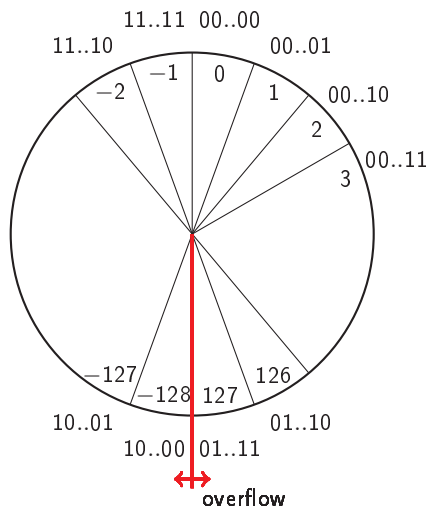
# A túlcscordulás (overflow)



- 8 biten ábrázolt előjeles számok esetén

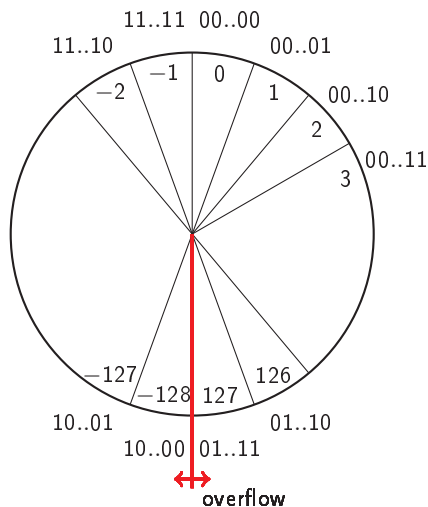


# A túlcscordulás (overflow)



- 8 biten ábrázolt előjeles számok esetén
- $127+1 = -128$

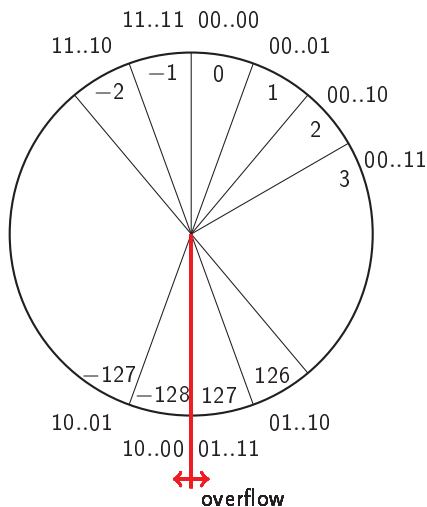
# A túlcsondulás (overflow)



■ 8 biten ábrázolt előjeles számok esetén

- $127+1 = -128$
- $127+2 = -127$

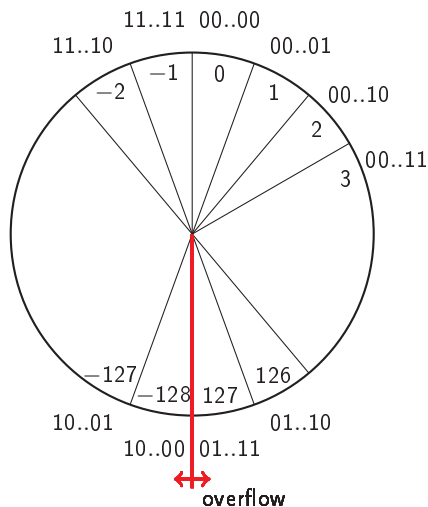
# A túlcsondulás (overflow)



■ 8 biten ábrázolt előjeles számok esetén

- $127+1 = -128$
- $127+2 = -127$
- $-127-3 = 126$

# A túlcsondulás (overflow)



■ 8 biten ábrázolt előjeles számok esetén

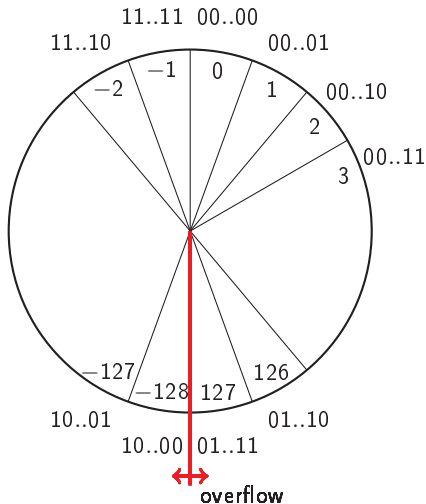
■  $127+1 = -128$

■  $127+2 = -127$

■  $-127-3 = 126$

■ viszont

# A túlcsondulás (overflow)



- 8 biten ábrázolt előjeles számok esetén

- $127+1 = -128$

- $127+2 = -127$

- $-127-3 = 126$

- viszont

- $2-3 = -1$

# Egész típusok C-ben

| típus                               | bit <sup>1</sup> | <limits.h> |            | printf            |
|-------------------------------------|------------------|------------|------------|-------------------|
| signed char                         | 8                | CHAR_MIN   | CHAR_MAX   | %hhd <sup>2</sup> |
| unsigned char                       | 8                | 0          | UCHAR_MAX  | %hhu <sup>2</sup> |
| signed short int                    | 16               | SHRT_MIN   | SHRT_MAX   | %hd               |
| unsigned short int                  | 16               | 0          | USHRT_MAX  | %hu               |
| signed int                          | 32               | INT_MIN    | INT_MAX    | %d                |
| unsigned int                        | 32               | 0          | UINT_MAX   | %u                |
| signed long int                     | 32               | LONG_MIN   | LONG_MAX   | %ld               |
| unsigned long int                   | 32               | 0          | ULONG_MAX  | %lu               |
| signed long long int <sup>2</sup>   | 64               | LLONG_MIN  | LLONG_MAX  | %lld              |
| unsigned long long int <sup>2</sup> | 64               | 0          | ULLONG_MAX | %llu              |

<sup>1</sup>Tipikus értékek, a szabvány csak a minimumot írja elő

<sup>2</sup>C99 szabvány óta

# Egészek deklarációja

## ■ Alapértelmezések

- A `signed` előjelmódosító elhagyható

```
1  int i;           /* signed int */
2  long int l;      /* signed long int */
```

# Egészek deklarációja

## ■ Alapértelmezések

### ■ A `signed` előjelmódosító elhagyható

```
1  int i;           /* signed int */
2  long int l;      /* signed long int */
```

### ■ Ha van előjel- vagy hosszmódosító, az `int` elhagyható

```
1  unsigned u;      /* unsigned int */
2  short s;         /* signed short int */
```



# Egész típusok

- Példa a táblázat használatához: egy igen sokáig futó program<sup>3</sup>

```
1 #include <limits.h> /* egész határokhoz */
2 #include <stdio.h> /* printf-hez */
3
4 int main(void)
5 { /* majdnem összes long long int */
6     long long i;
7
8     for (i = LLONG_MIN; i < LLONG_MAX; i = i+1)
9         printf("%lld\n", i);
10
11     return 0;
12 }
```

[link](#)

---

<sup>3</sup>feltéve, hogy `long long int` 64 bites, a program másodpercenként millió szám kiírásával 585 000 évig fut

# Egész számkonstansok

## ■ Egész számkonstansok megadási módjai

```
1 int i1=0, i2=123, i4=-33;           /* decimális */
2 int o1=012, o2=01234567;          /* oktális */
3 int h1=0x1a, h2=0x7fff, h3=0xAa1B  /* hexadecimális */
4
5 long l1=0x1a1, l2=-33L;            /* l vagy L */
6
7 unsigned u1=33u, u2=45U;           /* u vagy U */
8 unsigned long ul1=33uL, ul2=1231U; /* l és u */
```

# Egész számkonstansok

## ■ Egész számkonstansok megadási módjai

```
1 int i1=0, i2=123, i4=-33;           /* decimális */
2 int o1=012, o2=01234567;          /* oktális */
3 int h1=0x1a, h2=0x7fff, h3=0xAa1B /* hexadecimális */
4
5 long l1=0x1a1, l2=-33L;            /* l vagy L */
6
7 unsigned u1=33u, u2=45U;           /* u vagy U */
8 unsigned long ul1=33uL, ul2=1231U; /* l és u */
```

## ■ Ha nincs megadva u vagy l, akkor az első, amibe belefér:

- 1 int
- 2 unsigned int – hexa és oktális esetén
- 3 long
- 4 unsigned long

# Miért kell ismerni az ábrázolás korlátait?

Határozzuk meg a következő értéket!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15 - 12)!}$$

(Hányféleképpen választhatok ki 15 különböző csoki közül 12-t?)

# Miért kell ismerni az ábrázolás korlátait?

Határozzuk meg a következő értéket!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15 - 12)!}$$

(Hányféleképpen választhatok ki 15 különböző csoki közül 12-t?)

- A számláló értéke  $15! = 1\,307\,674\,368\,000$

# Miért kell ismerni az ábrázolás korlátait?

Határozzuk meg a következő értéket!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15 - 12)!}$$

(Hányféleképpen választhatok ki 15 különböző csoki közül 12-t?)

- A számláló értéke  $15! = 1\,307\,674\,368\,000$
- A nevező értéke  $12! \cdot 3! = 2\,874\,009\,600$

# Miért kell ismerni az ábrázolás korlátait?

Határozzuk meg a következő értéket!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15 - 12)!}$$

(Hányféleképpen választhatok ki 15 különböző csoki közül 12-t?)

- A számláló értéke  $15! = 1\,307\,674\,368\,000$
- A nevező értéke  $12! \cdot 3! = 2\,874\,009\,600$
- Egyik sem ábrázolható 32 bites `int`-tel!

# Miért kell ismerni az ábrázolás korlátait?

Határozzuk meg a következő értéket!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15 - 12)!}$$

(Hányféleképpen választhatok ki 15 különböző csoki közül 12-t?)

- A számláló értéke  $15! = 1\,307\,674\,368\,000$
- A nevező értéke  $12! \cdot 3! = 2\,874\,009\,600$
- Egyik sem ábrázolható 32 bites `int`-tel!
- A kifejezést egyszerűsítve

$$\frac{15 \cdot 14 \cdot 13}{3 \cdot 2 \cdot 1} = \frac{2730}{6} = 455$$

minden részletszámítás gond nélkül elvégezhető már akár 12 biten is



# Karakterek ábrázolása – Az ASCII karaktertáblázat

- 128 karakter, melyeket a 0x00–0x7f számokkal indexelhetünk

| Kód | 00  | 10  | 20 | 30 | 40 | 50 | 60 | 70  |
|-----|-----|-----|----|----|----|----|----|-----|
| +00 | NUL | DLE | □  | 0  | @  | P  | '  | p   |
| +01 | SOH | DC1 | !  | 1  | A  | Q  | a  | q   |
| +02 | STX | DC2 | "  | 2  | B  | R  | b  | r   |
| +03 | ETX | DC3 | #  | 3  | C  | S  | c  | s   |
| +04 | EOT | DC4 | \$ | 4  | D  | T  | d  | t   |
| +05 | ENQ | NAK | %  | 5  | E  | U  | e  | u   |
| +06 | ACK | SYN | &  | 6  | F  | V  | f  | v   |
| +07 | BEL | ETB | ,  | 7  | G  | W  | g  | w   |
| +08 | BS  | CAN | (  | 8  | H  | X  | h  | x   |
| +09 | HT  | EM  | )  | 9  | I  | Y  | i  | y   |
| +0a | LF  | SUB | *  | :  | J  | Z  | j  | z   |
| +0b | VT  | ESC | +  | ;  | K  | [  | k  | {   |
| +0c | FF  | FS  | ,  | <  | L  | \  | l  |     |
| +0d | CR  | GS  | -  | =  | M  | ]  | m  | }   |
| +0e | SO  | RS  | .  | >  | N  | ^  | n  | ~   |
| +0f | SI  | US  | /  | ?  | O  | _  | o  | DEL |

# Karakterek tárolása, kiírása, beolvasása

- Karaktereket (az ASCII tábla indexeit) a `char` típusban tárolunk
- kiírás/beolvasás `%c` formátumkóddal

```
1 char ch = 0x61; /* hex 61 = dec 97 */  
2 printf("%d: %c\n", ch, ch);  
3 ch = ch+1; /* értéke hex 62 = 98 lesz */  
4 printf("%d: %c\n", ch, ch);
```

# Karakterek tárolása, kiírása, beolvasása

- Karaktereket (az ASCII tábla indexeit) a `char` típusban tárolunk
- kiírás/beolvasás `%c` formátumkóddal

```
1 char ch = 0x61; /* hex 61 = dec 97 */  
2 printf("%d: %c\n", ch, ch);  
3 ch = ch+1; /* értéke hex 62 = 98 lesz */  
4 printf("%d: %c\n", ch, ch);
```

- A program kimenete

```
97:  a  
98:  b
```

# Karakterek tárolása, kiírása, beolvasása

- Karaktereket (az ASCII tábla indexeit) a `char` típusban tárolunk
- kiírás/beolvasás `%c` formátumkóddal

```
1 char ch = 0x61; /* hex 61 = dec 97 */  
2 printf("%d: %c\n", ch, ch);  
3 ch = ch+1; /* értéke hex 62 = 98 lesz */  
4 printf("%d: %c\n", ch, ch);
```

- A program kimenete

```
97:  a  
98:  b
```

- Ezek szerint karakterek kiírásához meg kell tanulnunk az ASCII-kódokat?

# Karakterkonstansok

## ■ Aposztrófok közé írt karakter ekvivalens az ASCII-kóddal

```
1 char ch = 'a'; /* ch-ba a 0x61 ASCII-kód kerül */
2 printf("%d: %c\n", ch, ch);
3 ch = ch+1;
4 printf("%d: %c\n", ch, ch);
```

# Karakterkonstansok

## ■ Aposztrófok közé írt karakter ekvivalens az ASCII-kóddal

```
1 char ch = 'a'; /* ch-ba a 0x61 ASCII-kód kerül */  
2 printf("%d: %c\n", ch, ch);  
3 ch = ch+1;  
4 printf("%d: %c\n", ch, ch);
```

```
97: a  
98: b
```

# Karakterkonstansok

## ■ Aposztrófok közé írt karakter ekvivalens az ASCII-kóddal

```
1 char ch = 'a'; /* ch-ba a 0x61 ASCII-kód kerül */
2 printf("%d: %c\n", ch, ch);
3 ch = ch+1;
4 printf("%d: %c\n", ch, ch);
```

```
97: a
```

```
98: b
```

## ■ Vigyázat! '0' $\neq$ 0 !

```
1 char n = '0'; /* ch-ba a 0x30 ASCII-kód kerül !!! */
2 printf("%d: %c\n", n, n);
```

# Karakterkonstansok

## ■ Aposztrófok közé írt karakter ekvivalens az ASCII-kóddal

```
1 char ch = 'a'; /* ch-ba a 0x61 ASCII-kód kerül */
2 printf("%d: %c\n", ch, ch);
3 ch = ch+1;
4 printf("%d: %c\n", ch, ch);
```

```
97:  a
```

```
98:  b
```

## ■ Vigyázat! '0' $\neq$ 0 !

```
1 char n = '0'; /* ch-ba a 0x30 ASCII-kód kerül !!! */
2 printf("%d: %c\n", n, n);
```

```
48:  0
```



# Karakterkonstansok

- Speciális karakterkonstansok – amiket egyébként nehéz lenne beírni

|      |    |                      |                      |
|------|----|----------------------|----------------------|
| 0x00 | \0 | nullkarakter         | null character (NUL) |
| 0x07 | \a | hangjelzés           | bell (BEL)           |
| 0x08 | \b | visszatörlés         | backspace (BS)       |
| 0x09 | \t | tabulátor            | tabulator (HT)       |
| 0x0a | \n | soremelés            | line feed (LF)       |
| 0x0b | \v | függőleges tabulátor | vertical tab (VT)    |
| 0x0c | \f | lapdobás             | form feed (FF)       |
| 0x0d | \r | kocsi vissza         | carriage return (CR) |
| 0x22 | \" | idézőjel             | quotation mark       |
| 0x27 | \' | aposztróf            | apostrophe           |
| 0x5c | \\ | visszaper            | backslash            |

# Karakter vagy egész szám?

- C-ben a karakterek egész számokkal ekvivalensek

# Karakter vagy egész szám?

- C-ben a karakterek egész számokkal ekvivalensek
- Csak megjelenítéskor dől el, hogy egy egész értéket számként vagy karakterként (%d vagy %c) ábrázolunk

# Karakter vagy egész szám?

- C-ben a karakterek egész számokkal ekvivalensek
- Csak megjelenítéskor dől el, hogy egy egész értéket számként vagy karakterként (`%d` vagy `%c`) ábrázolunk
- Karaktereken ugyanolyan műveleteket végezhetünk, mint egészekben (összeadás, kivonás stb. . .)

# Karakter vagy egész szám?

- C-ben a karakterek egész számokkal ekvivalensek
- Csak megjelenítéskor dől el, hogy egy egész értéket számként vagy karakterként (`%d` vagy `%c`) ábrázolunk
- Karaktereken ugyanolyan műveleteket végezhetünk, mint egészekben (összeadás, kivonás stb. . .)
- De mi értelme lehet karaktereket összeadni-kivonni?



# Műveletek karakterekkel

Írjunk programot, mely karaktereket olvas be mindaddig, míg az újsor karakter nem érkezik. Ezután a program írja ki a beolvasott számjegyek összegét.

# Műveletek karakterekkel

Írjunk programot, mely karaktereket olvas be mindaddig, míg az újsor karakter nem érkezik. Ezután a program írja ki a beolvasott számjegyek összegét.

```
1 char c;  
2 int sum = 0;  
3 do  
4 {  
5     scanf("%c", &c);                /* beolvasás */  
6     if (c >= '0' && c <= '9')      /* ha számjegy */  
7         sum = sum + (c - '0');      /* összegzés */  
8 }  
9 while (c != '\n');                 /* leállási feltétel */  
10 printf("Az összeg: %d\n", sum);
```

# Műveletek karakterekkel

Írjunk programot, mely karaktereket olvas be mindaddig, míg az újsor karakter nem érkezik. Ezután a program írja ki a beolvasott számjegyek összegét.

```
1 char c;  
2 int sum = 0;  
3 do  
4 {  
5     scanf("%c", &c);                /* beolvasás */  
6     if (c >= '0' && c <= '9')      /* ha számjegy */  
7         sum = sum + (c - '0');    /* összegzés */  
8 }  
9 while (c != '\n');                /* leállási feltétel */  
10 printf("Az összeg: %d\n", sum);
```

Karambolozott a 12:35-ös gyors

Az összeg: 11





# Műveletek karakterekkel

Írjunk programot, mely az angol ábécé kisbetűs karaktereit nagybetűssé alakítja, a többi karaktert változatlanul hagyja.

# Műveletek karakterekkel

Írjunk programot, mely az angol ábécé kisbetűs karaktereit nagybetűssé alakítja, a többi karaktert változatlanul hagyja.

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char c;
5     while (scanf("%c", &c) != EOF)
6     {
7         if (c >= 'a' && c <= 'z')
8             c = c + 'A' - 'a';
9         printf("%c", c);
10    }
11    return c;
12 }
```

# Lebegőpontos típusok

## ■ Normálalak

$$23,2457 = (-1)^0 \cdot 2,3245700 \cdot 10^{+001}$$

$$-0,001822326 = (-1)^1 \cdot 1,8223260 \cdot 10^{-003}$$

# Lebegőpontos típusok

## ■ Normálalak

$$23,2457 = (-1)^0 \cdot 2,3245700 \cdot 10^{+001}$$

$$-0,001822326 = (-1)^1 \cdot 1,8223260 \cdot 10^{-003}$$

## Normálalak ábrázolása

■ Lebegőpontos tört = előjelbit + mantissza + exponens

# Lebegőpontos típusok

## ■ Normálalak

$$23,2457 = (-1)^0 \cdot 2,3245700 \cdot 10^{+001}$$

$$-0,001822326 = (-1)^1 \cdot 1,8223260 \cdot 10^{-003}$$

## Normálalak ábrázolása

- Lebegőpontos tört = előjelbit + mantissza + exponens

**1** előjelbit: 0–pozitív, 1–negatív

# Lebegőpontos típusok

## ■ Normálalak

$$23,2457 = (-1)^0 \cdot 2,3245700 \cdot 10^{+001}$$

$$-0,001822326 = (-1)^1 \cdot 1,8223260 \cdot 10^{-003}$$

## Normálalak ábrázolása

- Lebegőpontos tört = előjelbit + mantissza + exponens

1 előjelbit: 0–pozitív, 1–negatív

2 mantissza: előjel nélküli egész (a tizedesvesszőt elhagyva),  
normalizálás miatt az első számjegy  $\geq 1$

# Lebegőpontos típusok

## ■ Normálalak

$$\begin{aligned} 23,2457 &= (-1)^0 \cdot 2,3245700 \cdot 10^{+001} \\ -0,001822326 &= (-1)^1 \cdot 1,8223260 \cdot 10^{-003} \end{aligned}$$

## Normálalak ábrázolása

- Lebegőpontos tört = előjelbit + mantissza + exponens
- 1 előjelbit: 0–pozitív, 1–negatív
- 2 mantissza: előjel nélküli egész (a tizedes vesszőt elhagyva), normalizálás miatt az első számjegy  $\geq 1$
- 3 exponens (másként karakterisztika): előjeles egész

# Lebegőpontos típusok

## ■ Bináris normálalak

$$5,0 = 1,25 \cdot 2^{+2} = (-1)^0 \cdot 1,0100_b \cdot 2^{010_b}$$

|   |      |     |
|---|------|-----|
| 0 | 0100 | 010 |
|---|------|-----|

---

<sup>4</sup>implicit bites ábrázolás



# Lebegőpontos típusok

## ■ Bináris normálalak

$$5,0 = 1,25 \cdot 2^{+2} = (-1)^0 \cdot 1,0100_b \cdot 2^{010_b}$$

|   |      |     |
|---|------|-----|
| 0 | 0100 | 010 |
|---|------|-----|

## Bináris normálalak ábrázolása

■ Lebegőpontos tört = előjelbit + mantissza + exponens

---

<sup>4</sup>implicit bites ábrázolás

# Lebegőpontos típusok

## ■ Bináris normálalak

$$5,0 = 1,25 \cdot 2^{+2} = (-1)^0 \cdot 1,0100_b \cdot 2^{010_b}$$

|   |      |     |
|---|------|-----|
| 0 | 0100 | 010 |
|---|------|-----|

## Bináris normálalak ábrázolása

■ Lebegőpontos tört = előjelbit + mantissza + exponens

**1** előjelbit: 0–pozitív, 1–negatív

---

<sup>4</sup>implicit bites ábrázolás

# Lebegőpontos típusok

## ■ Bináris normálalak

$$5,0 = 1,25 \cdot 2^{+2} = (-1)^0 \cdot 1,0100_b \cdot 2^{010_b}$$

|   |      |     |
|---|------|-----|
| 0 | 0100 | 010 |
|---|------|-----|

## Bináris normálalak ábrázolása

### ■ Lebegőpontos tört = előjelbit + mantissza + exponens

1 előjelbit: 0–pozitív, 1–negatív

2 mantissza: előjel nélküli egész (a **kettedesvesszőt** elhagyva), normalizálás miatt az első számjegy = 1, nem is tároljuk<sup>4</sup>.

---

<sup>4</sup>implicit bites ábrázolás

# Lebegőpontos típusok

## ■ Bináris normálalak

$$5,0 = 1,25 \cdot 2^{+2} = (-1)^0 \cdot 1,0100_b \cdot 2^{010_b}$$

|   |      |     |
|---|------|-----|
| 0 | 0100 | 010 |
|---|------|-----|

## Bináris normálalak ábrázolása

### ■ Lebegőpontos tört = előjelbit + mantissza + exponens

- 1 **előjelbit**: 0–pozitív, 1–negatív
- 2 **mantissza**: előjel nélküli egész (a **kettedesvesszőt** elhagyva), normalizálás miatt az első számjegy = 1, nem is tároljuk<sup>4</sup>.
- 3 **exponens**: előjeles egész

---

<sup>4</sup>implicit bites ábrázolás

# Lebegőpontos típusok C-ben

## ■ A C nyelv lebegőpontos típusai

| típus       | tipikus értékek |           |          | printf/scanf |
|-------------|-----------------|-----------|----------|--------------|
|             | bitszám         | mantissza | exponens |              |
| float       | 32 bit          | 23 bit    | 8 bit    | %f           |
| double      | 64 bit          | 52 bit    | 11 bit   | %f/%lf       |
| long double | 128 bit         | 112 bit   | 15 bit   | %Lf          |

# Lebegőpontos típusok C-ben

## ■ A C nyelv lebegőpontos típusai

| típus                    | tipikus értékek |           |          | printf/scanf        |
|--------------------------|-----------------|-----------|----------|---------------------|
|                          | bitszám         | mantissza | exponens |                     |
| <code>float</code>       | 32 bit          | 23 bit    | 8 bit    | <code>%f</code>     |
| <code>double</code>      | 64 bit          | 52 bit    | 11 bit   | <code>%f/%lf</code> |
| <code>long double</code> | 128 bit         | 112 bit   | 15 bit   | <code>%Lf</code>    |

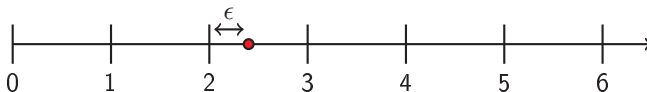
## ■ Lebegőpontos számkonstansok (tizedespont)

```

1 float      f1=12.3f , f2=12.F , f3=.5f , f4=1.2e-3F ;
2 double     d1=12.3 , d2=12. , d3=.5 , d4=1.2e-3 ;
3 long double l1=12.31 , l2=12.L , l3=.51 , l4=1.2e-3L ;

```

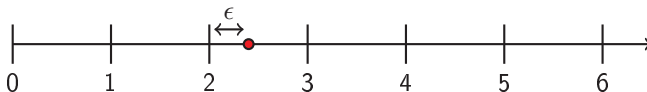
# Egész típusok ábrázolási pontossága



## Abszolút számábrázolási pontosság

A maximális  $\epsilon$  hiba, ha egy tetszőleges valós számot a hozzá legközelebbi ábrázolt értékkel közelítünk

# Egész típusok ábrázolási pontossága



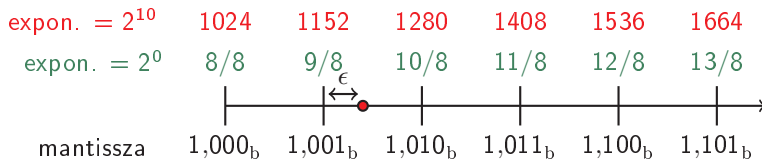
## Abszolút számábrázolási pontosság

A maximális  $\epsilon$  hiba, ha egy tetszőleges valós számot a hozzá legközelebbi ábrázolt értékkel közelítünk

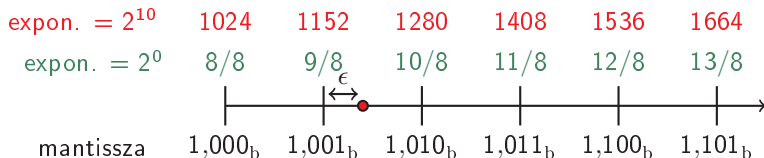
- Az egész típusok abszolút ábrázolási pontossága 0,5



# Lebegőpontos típusok ábrázolási pontossága



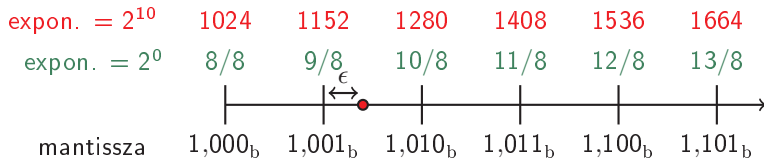
# Lebegőpontos típusok ábrázolási pontossága



■ jelen példában

■ A mantissa (abszolút) ábrázolási pontossága  $1/16$

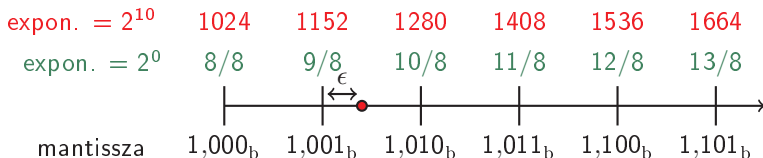
# Lebegőpontos típusok ábrázolási pontossága



## ■ jelen példában

- A mantissza (abszolút) ábrázolási pontossága  $1/16$
- $2^0$  exponens mellett az ábrázolási pontosság  $1/16$

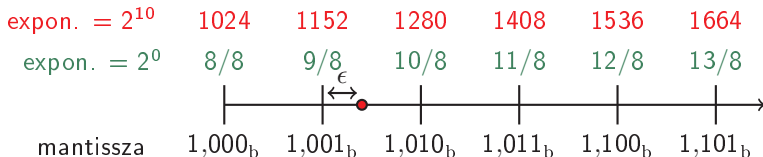
# Lebegőpontos típusok ábrázolási pontossága



## ■ jelen példában

- A mantissza (abszolút) ábrázolási pontossága  $1/16$
- $2^0$  exponens mellett az ábrázolási pontosság  $1/16$
- $2^{10}$  exponens mellett az ábrázolási pontosság  $2^{10}/16 = 64$

# Lebegőpontos típusok ábrázolási pontossága



## ■ jelen példában

- A mantissza (abszolút) ábrázolási pontossága  $1/16$
- $2^0$  exponens mellett az ábrázolási pontosság  $1/16$
- $2^{10}$  exponens mellett az ábrázolási pontosság  $2^{10}/16 = 64$
- Nem beszélhetünk abszolút, csak relatív ábrázolási pontosságról, ami jelen esetben 3 bit.

# A véges számábrázolás következménye

- Mivel a lebegőpontos számábrázolás pontatlan, műveletek eredményét **nem szabad** egyenlőségre összehasonlítani!

$$\frac{22}{7} + \frac{3}{7} \neq \frac{25}{7}$$

helyette

$$\left| \frac{22}{7} + \frac{3}{7} - \frac{25}{7} \right| < \varepsilon$$

# A véges számábrázolás következménye

- Mivel a lebegőpontos számábrázolás pontatlan, műveletek eredményét **nem szabad** egyenlőségre összehasonlítani!

$$\frac{22}{7} + \frac{3}{7} \neq \frac{25}{7}$$

helyette

$$\left| \frac{22}{7} + \frac{3}{7} - \frac{25}{7} \right| < \varepsilon$$

- A nagy számok sokkal pontatlanabbak, mint a kis számok. A nagy számok hibája „megeheti” a kicsiket:

$$A + a - A \neq a$$

# A bináris számábrázolás következménye

- Ami decimálisan véges, binárisan nem biztos, hogy az. pl:

$$0,1_d = 0,0\overline{0011}_b$$



# A bináris számábrázolás következménye

- Ami decimálisan véges, binárisan nem biztos, hogy az. pl:

$$0,1_d = 0,000\overline{11}_b$$

- Hányszor fut le az alábbi ciklus?

```
1 double d;  
2 for (d = 0.0; d < 1.0; d = d+0.1) /* 10? 11? */  
3 {  
4     ...  
5 }
```

# A bináris számábrázolás következménye

- Ami decimálisan véges, binárisan nem biztos, hogy az. pl:

$$0,1_d = 0,000\overline{11}_b$$

- Hányszor fut le az alábbi ciklus?

```
1 double d;  
2 for (d = 0.0; d < 1.0; d = d+0.1) /* 10? 11? */  
3 {  
4     ...  
5 }
```

- Helyesen:

```
1 double d, eps = 1e-3; /* mekkora eps jó ide? */  
2 for (d = 0.0; d < 1.0-eps; d = d+0.1) /* 10-szer */  
3 {  
4     ...  
5 }
```

# Keressük meg a hibát!

hibás:

```
1 double d = 3 / 2;  
2 long long int c = 500000 * 500000;
```

# Keressük meg a hibát!

hibás:

```
1 double d = 3 / 2;  
2 long long int c = 500000 * 500000;
```

helyesen:

```
1 double d = 3.0 / 2.0;  
2 long long int c = 500000LL * 500000LL;
```

Köszönöm a figyelmet.