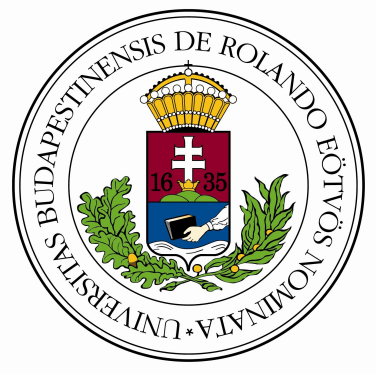
Imperatív programozás

Alaptípusok és -műveletek



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

Outline

Egész szám

- decimális alak: `42`, `1_000_000` (Python)
- oktális (C) és hexadecimális alak: `0123`, `0xCAFE`
- előjel nélküli ábrázolás (C): `34u`
- több biten ábrázolt (C): `99999999999999999L`
- „tetszőlegesen nagy'' értékű (Python): `99999999999999999999999999999999999999999999999`
- és kombinálva (C): `0xFEEL`

Lebegőpontos szám

- triviális: 3.141593
- exponenssel: 31415.93E-4
- több biten ábrázolt (C): 3.14159265358979L
- és kombinálva (C): 31415.9265358979E-4L

Karakter és szöveg

Szöveg Pythonban

- `'almafa'`
- `"almafa"`
- `'a'`
- `"a"`

Karakter és string C-ben

- karakterek: `'a'`, `'9'`, `'$'`
- stringek: `"a"`, `"almafa"`, `"1984"`

C és Python is

- escape-szekvenciák: `'\n'`, `'\t'`, `'\r'`, `"\n"`, `"\r\n"`
- több részből álló string: `"alma" "fa"`
- több sorba írt string:

```
"alma\
fa"
```

Outline

Operátorok

- aritmetikai
- értékadó (C)
- eggyel növelő/csökkentő (C)
- relációs
- logikai
- feltételes
- bitművelet
- `sizeof` (C)
- típuskényszerítő (C)

Aritmetikai operátorok

```
+ operand
- operand
left + right
left - right
left * right
left / right
left % right
```

„Valós” és egész osztás

C

```
5.0 / 2.0 == 2.5
5 / 2 == 2
```

Python

```
5 / 2 == 5.0 / 2.0
5 // 2 == 2
```

Egész osztás és osztási maradék

Python

- Egész osztás: lefelé kerekít

```
(left // right) * right + (left % right) == left
```

```
10 // (-3) == -4        10 % (-3) == -2
(-10) // 3 == -4        (-10) % 3 == +2
```

C

- Egész osztás: nulla felé kerekít
- Maradék előjele: `left` előjele

```
(left / right) * right + (left % right) == left
```

```
10 / (-3) == -3        10 % (-3) == +1
(-10) / 3 == -3        (-10) % 3 == -1
```

Aritmetikai operátorok – Python

```
left + right
left - right
left * right
left / right
left // right
left % right
left ** right
```

Hatványozás

Python

```
5.1 ** 2.1 == 30.612399320407246
```

C

```
#include <math.h>

    pow( 5.1, 2.1 )
```

Értékadó operátorok C-ben

```
n = 3
n += 3          n = (n + 3)
n -= 3          n = (n - 3)
n *= 3          n = (n * 3)
n /= 3          n = (n / 3)
n %= 3          n = (n % 3)
```

Eggyel növelő/csökkentő operátorok C-ben

Mellékhatás

```
c++;        c += 1;        c = (c + 1);
++c;        c += 1;        c = (c + 1);

c--;        c -= 1;        c = (c - 1);
--c;        c -= 1;        c = (c - 1);
```

Érték

c++	c
++c	c+1
c--	c
--c	c-1

Relációs operátorok

```
left == right
left != right
left <= right
left >= right
left < right
left > right
```

Logikai (boolean) értékűek

Összekapcsolás Pythonban

$3 < x < 7$

- C-ben ez egész mást jelent...

Logikai értékek Pythonban

- `True` és `False` értékek
- elágazás és ciklus feltételében...
- nulla jellegű értékek – `False`

```
if x > 0:
  if 6-3*2:
    print("success")
```

Nulla jellegű értékek Pythonban

- `False`
- `0`, `0.0`, `0j`, `Decimal('0.0')`, `Fraction(0,3)`
- None
- `""`
- üres adatszerkezet, pl. `[]`

Logikai típus C-ben

ANSI C: nincsen

hamis: 0, igaz: minden más (de főleg 1)

```
int right = 3 < 5;
int wrong = 3 > 5;
printf("%d %d\n",right,wrong);
```

C99-től

```
_Bool v = 3 < 5;
int one = (_Bool) 0.5;
int zero = (int) 0.5;
```

```
#include <stdbool.h>
...
bool v = true;
```

Végtelen ciklus idiómája

```
while(1){
    ...
}
```

Mit csinál ez a kód?

```
while( x = 5 ){
    printf("%d\n", x);
    --x;
}
```

Mit jelent ez?

$3 < x < 7$

Logikai operátorok

Python

```
left and right
left or right
not operand
```

C

```
left && right
left || right
! operand
```

Feltételes operátor

Python

```
left if condition else right
```

C

```
condition ? left : right
```

Bitműveletek

```
int two = 2;
int sixteen = 2 << 3;
int one = 2 >> 1;
int zero = 2 >> 2;

int three = two | one;
int five = 13 & 7;
int twelve = 9 ^ five;
int minusOne = ~zero;
```

sizeof

- Adat vagy típus memóriabeli mérete
- Fordítási időben kiértékelhető

```
sizeof(char) == 1
sizeof(int)
sizeof(42)
sizeof(42L)

char str[7];
sizeof str
```

sizeof

- Adat vagy típus memóriabeli mérete
- Fordítási időben kiértékelhető

```
sizeof(char) == 1
sizeof(int)
sizeof(42)
sizeof(42L)

char str[7];
sizeof str
```

- size_t
- printf("%lu", sizeof 42L)

Konvertálás típusok között

```
float five = 5;     /* automatikus */
```

Konvertálás típusok között

```c
float five = 5;     /* automatikus */

float how_much = 5 / 2;
```

Konvertálás típusok között

```
float five = 5;     /* automatikus */

float how_much = 5 / 2;

float two_and_half = 5. / 2;
```

Típuskényszerítés

```
float pi = 3.141592;
int three = (int) pi;
```

Típuskényszerítés

```
float pi = 3.141592;
int three = (int) pi;

float one = three / 2;
float one_and_half = ((float)three) / 2;
```

Outline

Számok ábrázolása C-ben

- egész számok (integer) – egy intervallum $\mathbb{Z}$-ben
  - előjel nélküli (unsigned)
  - előjeles (signed)
- lebegőpontos számok (float) $\subsetneq \mathbb{Q}$

(különböző méretekben)

Előjel nélküli egész számok

Négy biten

$1011 = 2^3 + 2^1 + 2^0$

n biten

$$b_{n-1} \ldots b_2 \; b_1 \; b_0 = \sum_{i=0}^{n-1} b_i \; 2^i$$

C-ben

```
unsigned int big = 0xFFFFFFFF;
if( big > 0 ){ printf("it's big!"); }
```

Előjellel: „kettes komplemens'' ábrázolás

(two's complement)

- első bit: előjel
- többi bit: helyiértékek

Négy biten

0000	0			
0001	1	1111	-1	
0010	2	1110	-2	
0011	3	1101	-3	0011
0100	4	1100	-4	+1101
0101	5	1011	-5	-----
0110	6	1010	-6	10000
0111	7	1001	-7	
		1000	-8	

Előjeles egész C-ben

```c
int big = 0xFFFFFFFF;
if( big > 0 ){ printf("it's big!"); }
```

Aritmetika előjeles egészekre

- Aszimmetria: eggyel több negatív érték
- Természetellenes
  - „két nagy pozitív szám összege negatív lehet''
  - „negatív szám negáltja negatív lehet''
- Példa: két szám számtani közepe?

Egész típusok mérete

- `short`: legalább 16 bit
- `int`: legalább 16 bit
- `long`: legalább 32 bit
- `long long`: legalább 64 bit (C99)

```
sizeof(short) <= sizeof(int) <= sizeof(long)
```

Egész számok Pythonban

„Tetszőlegesen nagy'' abszolút értékű számok – mint Haskell Integer

- Gépközeli, hatékony ábrázolás
- Automatikusan átvált

Lebegőpontos számok

$$1423.3 = 1.4233 \cdot 10^{3}$$

$$14.233 = 1.4233 \cdot 10^{1}$$

$$0.14233 = 1.4233 \cdot 10^{-1}$$

Bináris ábrázolás

$(-1)^s \cdot m \cdot 2^e$

(s: előjel; m: mantissza; e: exponens)

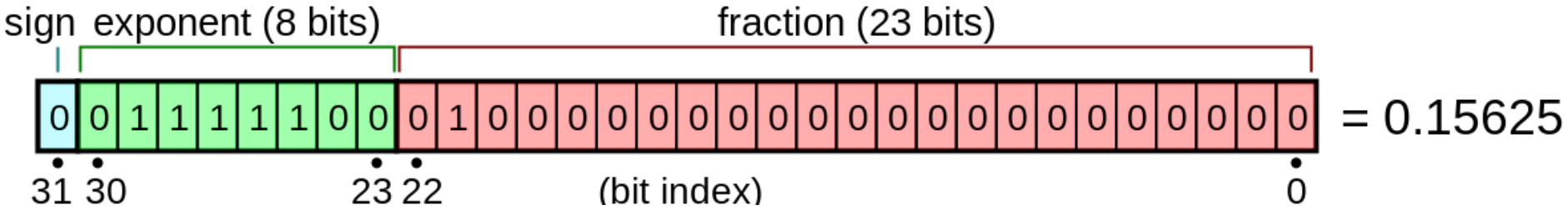
Rögzített számú biten reprezentálandó

- Előjel
- Kitevő
- Értékes számjegyek

IEEE 754

- Bináris rendszer
- A legtöbb számítógépes rendszerben
- Különböző méretű számok
  - egyszeres (32 bites: $1 + 23 + 8$)
  - dupla (64 bites: $1 + 52 + 11$)
  - kiterjesztett (80 bites: $1 + 64 + 15$)
  - négyszeres (128 bites: $1 + 112 + 15$)
- Mantissza 1 és 2 közé esik (pl. `1.01101000000000000000000`)
  - implicit első bit

32 bites példa



- előjel: 0 (nem negatív szám)
- „karakterisztika'': `01111100`, azaz 124
  - kitevő = karakterisztika - 127 = -3
- mantissza: 1.01000...0, azaz 1.25

Jelentés: $(-1)^0 \cdot 1.25 \cdot 2^{-3}$, azaz $1.25/8$

Lebegőpontos számok tulajdonságai

- Széles értéktartomány
  - Nagyon nagy és nagyon kicsi számok
- Nem egyenletes eloszlású
- Alul- és túlcsordulás
  - Pozitív és negatív nullák
  - Végtelenek
  - NaN
  - Denormalizált számok

Lebegőpontos aritmetika

```
2.0 == 1.1 + 0.9
2.0 - 1.1 != 0.9
2.0 - 0.9 == 1.1
```

Pénzt például sosem ábrázolunk lebegőpontos számokkal!

Python: fixpontos számok

```
from decimal import Decimal
...
Decimal('2.0') - Decimal('1.1') == Decimal('0.9')
```

Python: racionális számok

```
from fractions import Fraction
...
Fraction(1,3)
```

Komplex számok

Valós és képzetes részből, pl.: $3.14 + 2.72i$ (ahol $i^2 = -1$)

Python

```
v = 3.14 + 2.72j
v * v
```

C99

```
float _Complex fc;
double _Complex dc;
long double _Complex ldc;
```

Komplex számok C99-től

```
#include <complex.h>
...
double complex dc = 3.14 + 2*I;
```

Értékek

- szám
  - egész (`144L`, `-23`, `0xFFFF`)
  - valós (`123.4`, `314.1592E-2`)
  - komplex (`3.14j`)
- logikai érték (`0` vagy `False`)
- karakter (`'a'`, `'\n'`)
- sorozat
  - szöveg
  - számsorozat

Karakterek

Python

Nincs külön ilyen

C

Valójában egy egész szám!

- Egy bájtos karakterkód, pl. ASCII

```
char c = 'A';       /* ASCII: 65 */
```

Escape-szekvenciák

- Speciális karakter: `\n`, `\r`, `\f`, `\t`, `\v`, `\b`, `\a`, `\\`, `\'`, `\"`, `\?`
- Oktális kód: `\0` – `\377`
- Hexadecimális kód, pl. `\x41`

Előjeles és előjel nélküli char (C-ben)

```
  signed char a = '\xFF';    /* a < 0          */
unsigned char b = '\xFF';    /* b > 0          */
         char c = '\xFF';    /* platformfüggő */
```

Szélesebb ábrázolás

```
wchar_t w = L'é';
```

- Implementációfüggő!
  - Windows: UTF-16
  - Unix: általában UTF-32
- C99-től „univerzális kód”:, pl. `\uC0A1` és `\U00ABCDEF`

Outline

Sorozatok típusa

- C: tömb
- Python: lista és rendezett n-es

C tömb

```c
double point[3];      /* a méret legyen konstans */
point[0] = 3.14;      /* nullától indexelünk */
point[1] = 2.72;
point[2] = 1.0;
```

C tömb inicializációja

```c
double point[] = {3.14, 2.72, 1. + .1};
                /* az elemek legyenek "konstansok", ha globális */


point[2] = 1.0;    /* módosítható */
```

Feldolgozás

```
#define DIMENSION 3

double sum( double point[] ){
    double result = 0.0;
    int i;
    for( i=0; i<DIMENSION; ++i ){
        result += point[i];
    }
    return result;
}

int main(){
    double point[DIMENSION] = {3.14, 2.72, 1.0};
    printf("%f\n", sum(point));
    return 0;
}
```

Általánosítás

```
double sum( double nums[], int length )
{
    double result = 0.0;
    int i;
    for( i=0; i<length; ++i ){
        result += nums[i];
    }
    return result;
}

int main()
{
    double point[] = {3.14, 2.72, 1.0};
    printf("%f\n", sum(point,3));
    return 0;
}
```

Veszélyforrások

Fordítási figyelmeztetés

```
double point[3] = {3.14, 2.72, 1.0, 2.0};
```

Inicializálatlan elemek: automatikusan 0.0

```
double point[3] = {3.14, 2.72};
```

Túlindexelés, illegális memóriaolvasás

```
printf("%f\n", point[1024]);
```

Túlindexelés, illegális memóriaírás (buffer overflow)

```
point[31024] = 1.0;   /*  Segmentation fault? */
```

Rendezett n-es Pythonban

Tuple

```
(3.14, 2.72)                    # rendezett pár
(100, 'bolha')                  # különböző típusúak is lehetnek
(2018, 'October', 2, 11, 0)     # rendezett 5-ös
(3.14,)                         # rendezett 1-es
()                              # üres tuple
(2018, 'October', 2) + (11, 0)  # összefűzés
(3,) * 10                       # multiplikálás
```

A zárójelek elhagyhatók!

Indexelés, szelet kiválasztása

```
date = (2018, 'October', 2, 11, 0)

date[2] == 2      # nullától indexelünk
date[7]           # hibás
date[-2] == 11
date[-7]          # hibás

date[1:3] == ('October', 2)
date[-2:5] == (11,0)
date[3:] == (11, 0)
date[:3] == (2018, 'October', 2)

date[4] = 30      # hibás
```

Módosíthatatlan / Immutable

Listák Pythonban

Módosítható, heterogén sorozat

```
date = [2018, 'October', 2, 11, 0]
date[4] = 15    # módosítható (mutable)
```

- indexelés, szeletkiválasztás, összefűzés, multiplikálás
- törlő, beszúró stb. műveletek
- list comprehension

```
[ x**2 for x in range(10) ]
```

```
[ (x,y) for x in [1,2,3] for y in [3,1,4] if x != y ]
```

Halmazok és leképezések

```
{1,3,4,4,5} == {4,5,3,1}
{1,2,3} | {2,3,4} == {1,2,3,4}
```

```
store = {"bread": 230, "butter": 300, "milk": 180}
store["butter"] == 300
```

Outline

Szövegek Pythonban

Módosíthatatlan karaktersorozat

```
name = "John Doe"
name[0] = 'j'    # hibás
```

- indexelés, szeletkiválasztás, összefűzés, multiplikálás

Szövegbeli betűk Pythonban

- Szövegek UTF-8-ban ábrázolva
  - Minden Unicode karakter ábrázolható
- Forráskód alapból UTF-8 kódolású

  `'körte'`
  - Megváltoztatható

    ```
    'k\u00F6rte'       # UTF-16
    'k\U000000F6rte'   # UTF-32
    'k\N{LATIN SMALL LETTER O WITH DIAERESIS}rte'
    ```
  - Azonosítókban is használhatunk Unicode karaktereket

    ```
    körte = 3
    ```

Szövegek C-ben

- Nem `string`!
- Karakterek tömbje, `'\0'`-val terminálva
  - Nullától indexelünk

```
char word[] = "apple";
printf("%lu\n", sizeof(word));    /* 6 */

char a = word[0];
word[0] = 'A';

wchar_t wide[] = L"körte";
```

Ékezetes betűk a szövegben (C-ben)

- Platformfüggő ábrázolás
- Egy karaktert több bájton is ábrázolhat
  - pl. UTF-8

```
char word[] = "körte";
printf("%lu\n", sizeof(word));   /* 7 */
```

Karaktertömb lefoglalása

```c
char w1[] = "alma";
char w2[8] = "alma";
printf("%lu %s\n", sizeof(w1), w1);    /* 5 alma */
printf("%lu %s\n", sizeof(w2), w2);    /* 8 alma */
```

Veszély: túl kis tömb foglalása

```
char w1[] = "lakoma";
char w2[4] = "alma";
printf("%lu %s\n", sizeof(w1), w1);    /* 7 lakoma */
printf("%lu %s\n", sizeof(w2), w2);    /* 4 almalakoma */
```

Szövegen belül nulla

```
char word[] = "lak\0ma";
printf("%lu %s\n", sizeof(word), word);    /* 7 lak */
printf("%c\n", word[4]);                   /* m */
```

Szövegek manipulálása

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

int main()
{
    char word[100];
    strcpy(word,"alma");
    strcat(word,"lakoma");
    printf("%lu %s\n", sizeof(word), word);  /* 100 almalakoma */
    printf("%lu\n", strlen(word));           /* 10 */
    return 0;
}
```

Kitekintés

```
char w1[]  = "alma";   /* a szöveg benne lesz */
char w2[6] = "alma";   /* a szöveg benne lesz */
char * w3  = "alma";   /* szövegre mutat, nem kellene módosítani */
printf("%lu %s\n", sizeof(w1), w1);   /* 5 alma */
printf("%lu %s\n", sizeof(w2), w2);   /* 6 alma */
printf("%lu %s\n", sizeof(w3), w3);   /* 8 alma */

w1[0] = 'A';
w2[0] = 'A';
w3[0] = 'A';     /* problémás - Segmentation Fault? */
```

Szövegek megadása tömbként


```
char good[] = "good";
char  bad[] = {'b', 'a', 'd'};
char ugly[] = {'u','g','l','y','\0'};
printf("%s %s %s\n", good, bad, ugly);
```