Imperatív programozás

Alaptípusok

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. szeptember 19.

ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

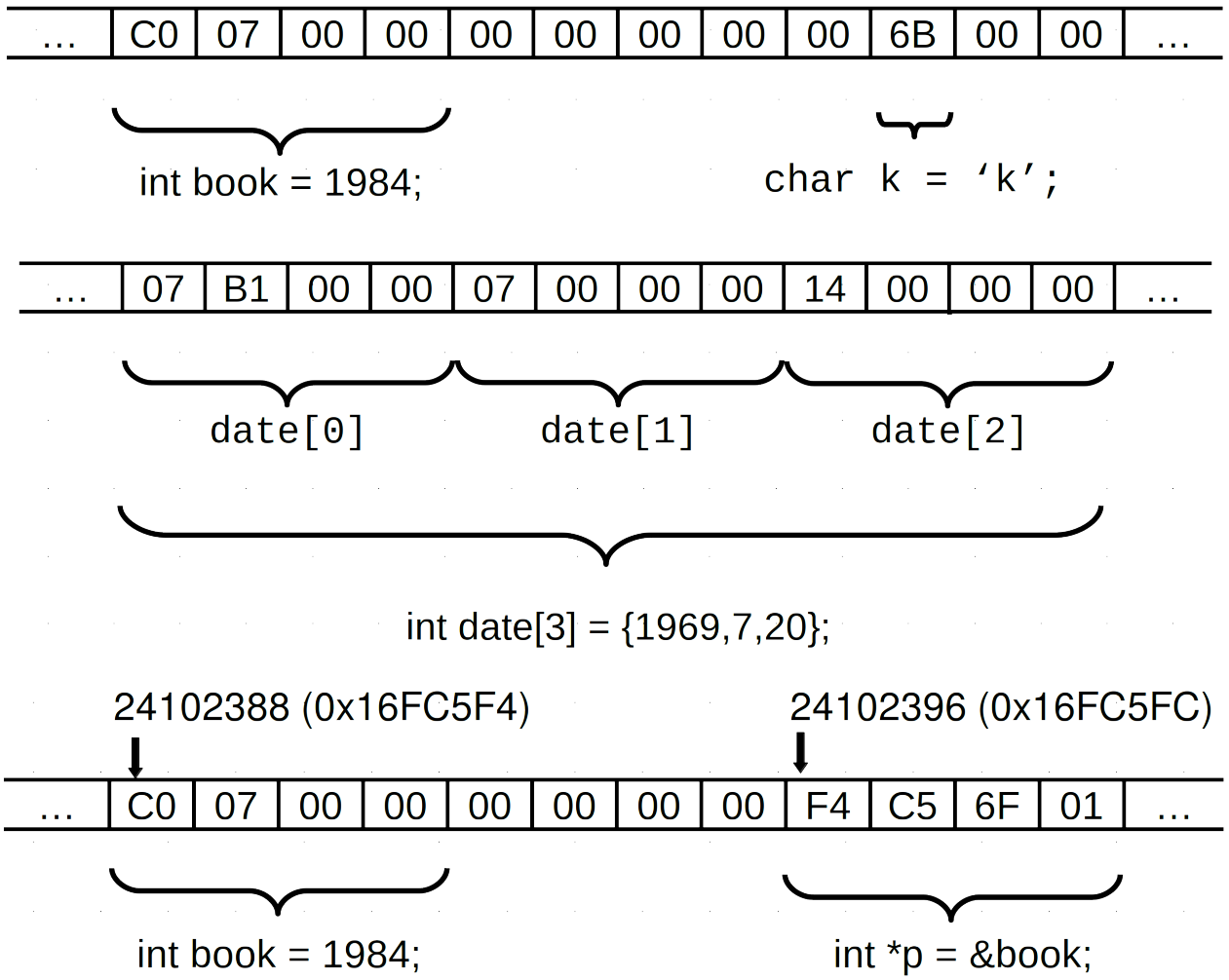
Tartalomjegyzék

A típus szerepe

- Védelem a programozói hibákkal szemben
- Kifejezik a programozók gondolatát
- Segítik az absztrakciók kialakítását
- Segítik a hatékony kód generálását

- Kifejezik egy bitsorozat értelmezési módját
- Meghatározzák, milyen értéket vehet fel egy változó
- Megkötik, hogy műveleteket milyen értékekkel végezhetünk el

Különböző típusú értékek a memóriában



p == 24102388
*p == 1984

Típusellenőrzés

- A változókat, függvényeket a típusuknak megfelelően használtuk-e
- A nem típushelyes programok értelmetlenek

Statikus és dinamikus típusrendszer

- A C fordító ellenőrzi *fordítási időben* a típushelyességet
- Egyes nyelvekben *futási időben* történik a típusellenőrzés

Erősen és gyengén típusos nyelv

- Gyengén típusos nyelvben automatikusan konvertálódnak értékek más típusúra, ha kell
  - Eleinte kényelmes
  - De könnyen írunk mást, mint amit szerettünk volna
- A C-ben viszonylag szigorúak a szabályok (erősen típusos)

Egész számok

- Decimális alak: `42`
- Oktális és hexadecimális alak: `0123`, `0xCAFE`
- Előjel nélküli ábrázolás: `34u`
- Több biten ábrázolt: `9999999999L`
- És kombinálva: `0xFEEL`

Lebegőpontos számok

- Triviális: `3.141593  5.  .3`
- Exponenssel: `31415.93E-4`
- Több biten ábrázolt: `3.14159265358979L`
- És kombinálva: `31415.9265358979E-4L`

Karakter és szöveg

- Karakterek: `'a'`, `'9'`, `'$'`
- Stringek: `"a"`, `"appletree"`, `"1984"`
- Escape-szekvenciák: `'\n'`, `'\t'`, `'\r'`, `"\n"`, `"\r\n"`
- Több részből álló string: `"apple" "tree"`
- Több sorba írt string:

```
"alma\
fa"
```

Karakterek

- 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`

Logikai típus?

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

```
                          #include <stdbool.h>
                          ...
_Bool v = 3 < 5;          bool v = true;
int one = (_Bool) 0.5;
int zero = (int) 0.5;
```

Komplex számok

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

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;
```

Operátorok

- Aritmetikai
- Értékadó
- Eggyel növelő/csökkentő (inkrementáló/dekrementáló)
- Relációs
- Logikai
- Feltételes
- Bitművelet
- `sizeof`
- Típuskényszerítő

Aritmetikai operátorok

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

„Valós” osztás

```
5.0 / 2.0 == 2.5

(C, Python2)
5 / 2 == 2

(Python3)
5 / 2 == 2.5
5 // 2 == 2
```

Egész osztás és osztási maradék

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

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

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

Hatványozás

```
#include <math.h>

pow(5.1, 2.1);
```

Értékadó operátorok

```
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

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
```

Mit csinál ez a kód?

```
if (x = 5)
{
  printf("Hello World!");
}
```

Mit csinál ez a kód?

```
if (x = 5)
{
  printf("Hello World!");
}
```

$3 < x < 7$

Mit csinál ez a kód?

```
if (x = 5)
{
  printf("Hello World!");
}
```

```
3 < x < 7
```

```
(3 < x) < 7
```

Bitműveletek

```
int two = 2;                // 00000010
int sixteen = 2 << 3;       // 00010000
int one = 2 >> 1;           // 00000001
int zero = 2 >> 2;          // 00000000

int three = two | one;      // 00000011
int thirteen = 13;          // 00001101
int seven = 7;              // 00000111
int five = 13 & 7;          // 00000101
int nine = 9;               // 00001001
int twelve = 9 ^ five;      // 00001100
int minusOne = ~zero;       // 11111111
```

Logikai operátorok

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

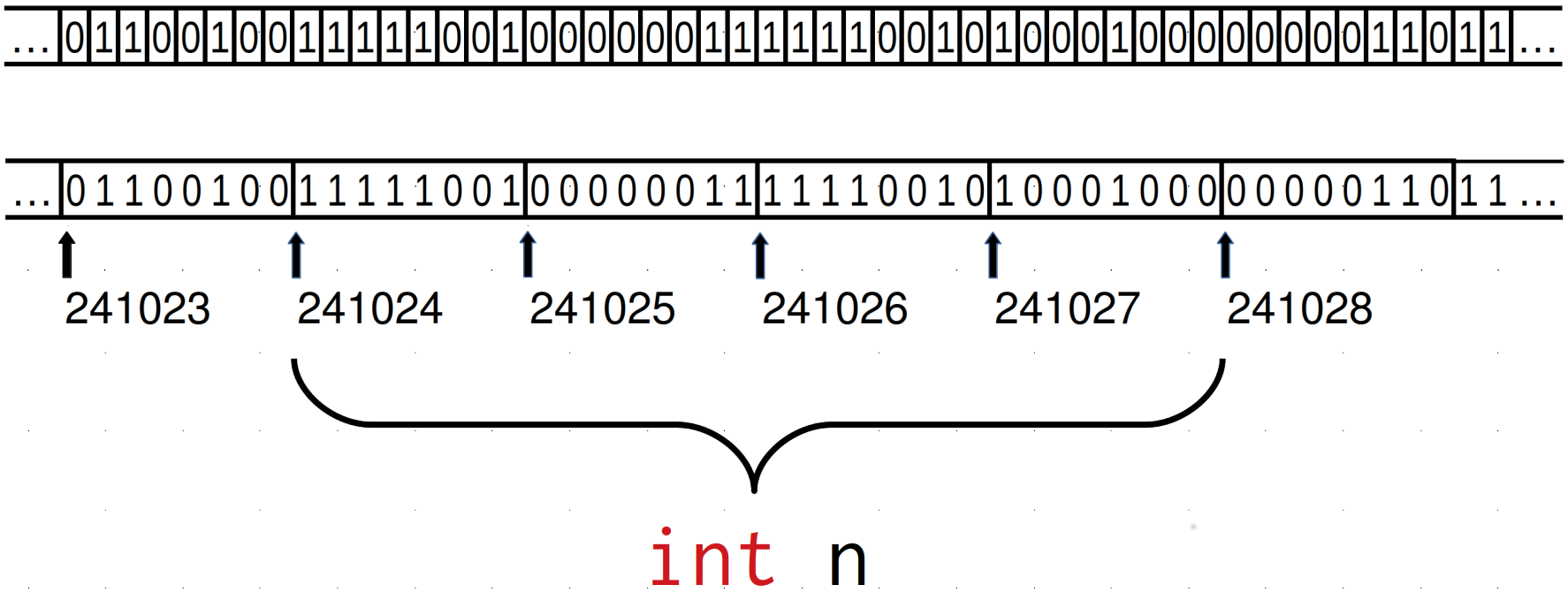
Feltételes operátor

```
condition ? left : right
```

Például:

```
int x = 1 < 2 ? 10 : 20;
printf("%d\n", x);    // 10
int y = 2 < 1 ? 10 : 20;
printf("%d\n", y);    // 20
```

Objektumok mérete



sizeof

- Adat vagy típus memóriabeli mérete
- Fordítási időben kiértékelhető

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

char str[7];
sizeof(str) == 7
```

- sizet
- printf("\lu", sizeof(42L))

Konvertálás típusok között

```
float five = 5;                    /* 5.0 (automatikus) */
float how_much = 5 / 2;                              /* 2.0 */
float two_and_half = 5. / 2                          /* 5.2 */
```

Konvertálás típusok között

```
float five = 5;                    /* 5.0 (automatikus) */
float how_much = 5 / 2;                              /* 2.0 */
float two_and_half = 5. / 2                          /* 5.2 */

float pi = 3.141592;
int three = (int) pi;                                  /* 3 */
```

Konvertálás típusok között

```
float five = 5;                    /* 5.0 (automatikus) */
float how_much = 5 / 2;                              /* 2.0 */
float two_and_half = 5. / 2                          /* 5.2 */

float pi = 3.141592;
int three = (int) pi;                                  /* 3 */

float one = three / 2;                               /* 1.0 */
float one_and_half = ((float) three) / 2;            /* 1.5 */
```

Számok ábrázolása bitsorozatként a memóriában

- Egész számok (integer) – egy intervallum $\mathbb{Z}$-ben
  - Előjel nélküli (unsigned)
  - Előjeles (signed)
- Lebegőpontos számok (float) $\subsetneq \mathbb{Q}$

Egész típusok mérete

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

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

Előjel nélküli számok

Négy biten

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

n biten

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

C-ben

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

Előjeles számok („kettes komplemens”)

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	

C-ben

```
signed int small = 0xFFFFFFFF;
if (small < 0) { printf("It's small"); }
```

Előjeles és előjel nélküli char

```
  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`

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?

$$\frac{a+b}{2} \quad vs \quad \frac{a}{2}+\frac{b}{2}$$

Lebegőpontos számok

$$1423.3 = 1.4233 \cdot 10^{3}$$
$$13.233 = 1.4233 \cdot 10^{1}$$
$$0.14233 = 1.4233 \cdot 10^{-1}$$

Lebegőpontos típusok mérete

- `float`
- `double`
- `long double`

```
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

Bináris ábrázolás

$(-1)^s \cdot m \cdot 2^e$
(s: előjel, m: mantissza, e: exponens)

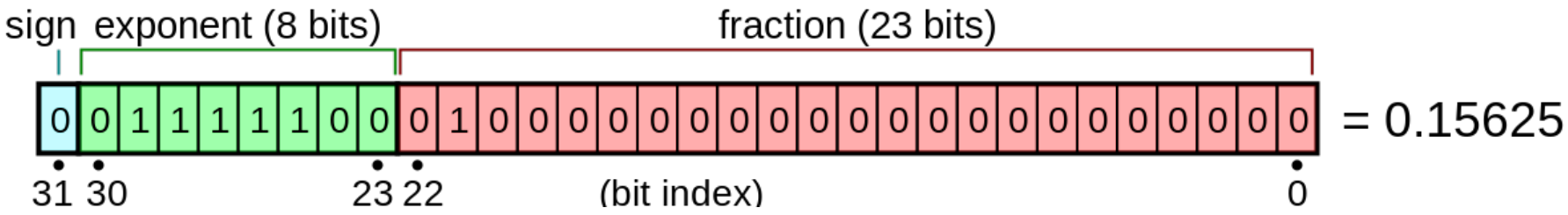
Rögzített számú biten reprezentálandó

- 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: 0111110, azaz 124
- Kitevő: Karakterisztika - 127 = -3
- Mantissza: 0.01000...0, azaz 1.25

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

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!