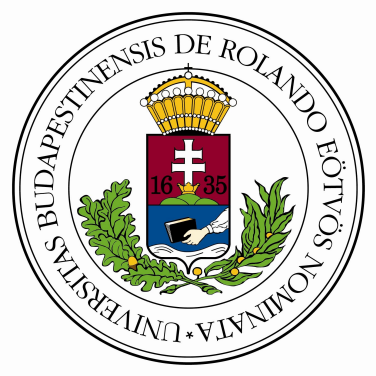
Imperative programming

3rd Lecture



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Examples

```
n + 1
```

```
3.14 * r * r
```

```
3 * v[0]
```

```
x < 3.14
```

```
3 * (r1 + r2) == factorial(x)
```

Lexica

- Literals
- Operators
- Identifiers
- Parentheses
- other, e.g. comma

Number Representation in C

- Whole Numbers (integers) – an interval in $\mathbb{Z}$
  - Unsigned
  - Signed
- Floating-point Numbers (floats) $\subsetneq \mathbb{Q}$

(in various sizes)

Unsigned Integers

On 4 bits

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

On n bits

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

in C

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

with Sign: „two's complement'' representation

- 1st bit: Sign
- rest of bits: Binary Digits

On 4 bits

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	

Signed Integer in C

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

Signed Integer Arithmetic

- Asymmetry: one more of negative values
- Unnatural
  - „two big positives added give a negative''
  - „negating a negative can be negative''
- Example: Arithmetic Mean (average) of two numbers?

Size of Integers

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

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

Floating-point Numbers

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

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

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

Binary Representation

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

(s: sign; m: mantissa / significand; e: exponent)

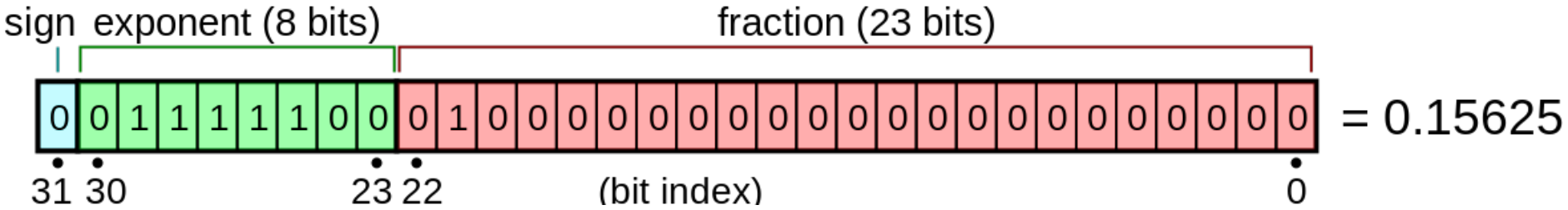
Represented in Fixed number of Bits

- Sign
- Exponent / Characteristic
- Significand / Mantissa / Fraction

IEEE 754

- Binary System
- Present in most computer systems
- Variable Sizes (Precisions)
  - Single (32 bits: $1 + 23 + 8$)
  - Double (64 bits: $1 + 52 + 11$)
  - Extended (80 bits: $1 + 64 + 15$)
  - Quadruple (128 bites: $1 + 112 + 15$)
- Mantissa is between 1 and 2 (e.g. `1.01101000000000000000000`)
  - implicit 1st bit

32 bits Example



- sign: 0 (non-negative)
- „characteristic'': `01111100`, so 124
  - exponent = characteristic - 127 = -3
- mantissa: 1.01000...0, so 1.25

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

Features of Floating-point Numbers

- Wide Range of Values
  - Very big and very small numbers
- Not Even Distribution of Numbers
- Over- and Underflow
  - Positive and Negative Zeros
  - Infinities
  - NaNs (Not a Number)
  - Denormal Values

Floating-point Arithmetic

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

No money in floating-point please!

Complex Numbers

Real and Imaginary parts, e.g. $3.14 + 2.72i$ (where $i^2 = -1$)

C99

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

Complex Numbers from C99

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

Conversion Between Types

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

Operators

- Arithmetic
- Assignment (C)
- Increase/Decrease (C)
- Relational
- Logical
- Conditional
- Bit Operations
- `sizeof` (C)

Arithmetic Operators

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

„Real'' and Integer Division

C

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

Exponentiation (Power of)

C

```c
#include <math.h>

    pow( 5.1, 2.1 )
```

Assignment in C

Assignment Statement

```
n = 1;
```

Expression with Side-effect

```
n = 1
```

Value of Expression with Side-effect

```
(n = 1) == 1
```

Value Propagation

```
m = (n = 1)
```

Assignment Operators in C

```
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)
```

Increase/Decrease Operators in C

Side-effect

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

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

Results of Evaluation

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

Relational Operators

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

Logical (boolean) Values

Chaining in Python

```
3 < x < 7
```

- In C this means something totally different...

Logical Type in C

ANSI C: none

false: 0, true: anything else (1 first of all)

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

from C99

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

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

Infinite Loop Idiom

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

What is this code doing?

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

Logical Operators

C

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

Conditional Operator

C

```
condition ? left : right
```

Bitoperations

```
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;
```

Syntax of Function Calls

Actual Parameters

```
<function-call> ::= <identifier> ( )
                  | <identifier> (<argument-list>)

<argument-list> ::= <expression>
                  | <expression> , <argument-list>
```

`pi()`, `factorial(n+m)`, `min(0,x*y)`

Usage of Operators

- Arity
  - Unary, e.g. `-x`, `c++`
  - Binary, e.g. `x-y`
  - Ternary, e.g. `x < 0 ? 0 : x`
- Fixity
  - Prefix, e.g. `++c`
  - Postfix, e.g. `c++`
  - Infix, e.g. `x+y`
  - Mixfix, e.g. `x < 0 ? 0 : x`

Evaluation of Expressions

- Fully Parenthesized Expression

  `3 + ((12 - 3) * 4)`

- Precedence: operator `*` bonds stronger than `+`

  `12 - 3 * 4`

- Left- and Right Associativity
  - between operators on the same precedence level
  - `3 * n / 2` means `(3 * n) / 2` (left-associative op.)
  - `n = m = 1` means `n = (m = 1)` (right-associative op.)

Evaluation of Expressions (cont.)

- Lazyness, Greedyness
  - Greedy: like expression `A + B`
  - Lazy: like expression `A && B`
- Side-effect

```
n = 1
i++
++i
i *= j
```

- Evaluation Order of Operands, Function Parameters

```
int i = 2;
int j = i -- - -- i;
```

- Sequence Point