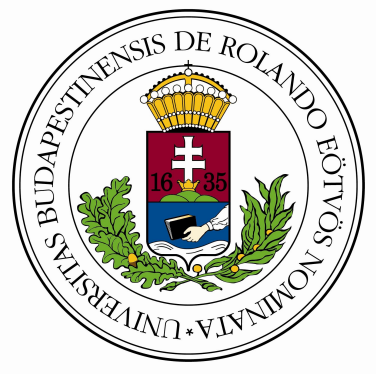
Imperatív programozás

Kifejezések



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

Példák

```
n + 1
```

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

```
x < 3.14
```

```
3.14 * r * r
```

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

Outline

Lexikális elemek

- literálok
- operátorok
- azonosítók
- zárójelek
- egyéb jelek, pl. vessző

Outline

Függvényhívás szintaktikája

aktuális paraméterlista

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

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

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

Operátorok arítása

- unáris, pl. `-x`, `c++`
- bináris, pl. `x-y`
- ternáris, pl. `x < 0 ? 0 : x` vagy `0 if x < 0 else x`

Operátorok fixitása

- prefix, pl. ++c
- postfix, pl. c++
- infix, pl. x+y
- mixfix, pl. `x < 0 ? 0 : x` vagy `0 if x < 0 else x`

Outline

Értelmes kifejezések

- A benne szereplő azonosítók deklaráltak
- Jól típusozott
- Nullával való osztás stb. nincs benne

Értelmes kifejezések

- A benne szereplő azonosítók deklaráltak
- Jól típusozott
- Nullával való osztás stb. nincs benne

Mi a jelentése?

Outline

Pure és impure nyelvek

- Impure (pl. C, Python)
  - Kifejezés értékének meghatározása
  - Mellékhatás
- Pure (pl. Haskell)
  - Kifejezés értékének meghatározása

    hivatkozási helyfüggetlenség (referential transparency)

Kiértékelés szabályai

- teljesen bezárójelezett kifejezés

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

Kiértékelés szabályai

- teljesen bezárójelezett kifejezés

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

- precedencia: a * erősebben köt, mint a +

  `12 - 3 * 4`

Kiértékelés szabályai

- teljesen bezárójelezett kifejezés

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

- precedencia: a `*` erősebben köt, mint a `+`

  ```
  12 - 3 * 4
  ```

- bal- és jobbasszociativitás
  - azonos precedenciaszintű operátorok esetén
  - `3 * n / 2` jelentése `(3 * n) / 2` (bal-asszociatív op.)
  - `n = m = 1` jelentése `n = (m = 1)` (jobb-asszociatív op. C-ben)
  - `2 ** 3 ** 2` jelentése `2 ** (3 ** 2)` (jobb-asszociatív op. Pythonban)

Értékadás C-ben

Értékadó utasítás

```
n = 1;
```

Értékadás C-ben

Értékadó utasítás

```
n = 1;
```

Mellékhatásos kifejezés

```
n = 1
```

Értékadás C-ben

Értékadó utasítás

```
n = 1;
```

Mellékhatásos kifejezés

```
n = 1
```

Mellékhatásos kifejezés értéke

(n = 1) értéke 1

Értékadás C-ben

Értékadó utasítás

```
n = 1;
```

Mellékhatásos kifejezés

```
n = 1
```

Mellékhatásos kifejezés értéke

`(n = 1)` értéke 1

Érték továbbgyűrűzése

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

Mellékhatás C-ben

```
printf("%d",n)
n = 1
i *= j
i++
++i
```

Eggyel növelő/csökkentő operátorok C-ben

```c
int n = 5;
int i;
i = ++n;
```

```c
int m = 5;
int j;
j = m++;
```

Eggyel növelő/csökkentő operátorok C-ben

```
int n = 5;
int i;
i = ++n;

n == 6
i == 6
```

```
int m = 5;
int j;
j = m++;

m == 6
j == 5
```

Kifejezés értéke

- „Normális'' érték
- Futási hiba, pl. `5/0` sok nyelvben
- Végtelen számítás

Lustaság, mohóság

- mohó: az `A + B` alakú kifejezés
- lusta: az `A && B` alakú kifejezés C-ben

Lusta/mohó logikai éselés értéke

Jelölje $\uparrow$, $\downarrow$, $\bot$ és ∞ a négy lehetséges eredményt egy logikai kifejezés kiértékeléséhez: igaz, hamis, kivétel, nem terminálló számítás. Az $\alpha \wedge \beta$ kifejezés értéke az α és β értékének függvényében:

$\alpha \wedge_{\text{lusta}} \beta$	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞

$\alpha \wedge_{\text{mohó}} \beta$	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\bot$	∞
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞

Példa

```
int i;
for( i=0; i<LENGTH && array[i] >= 0; ++i );
/* i is the index of first negative value
   or LENGTH if none exists */
```

Lusta/mohó kiértékelés hatása

Az „érték'' és a mellékhatás is különbözhet!

```
(n=1) + (m=1)
(n=1) || (m=1)
```

Lusta operátorok egy mohó nyelvben

`&&`	`and`
`||`	`or`
`?:`	`if else`

Operandusok, függvényparaméterek kiértékelési sorrendje

Legyen α, β, γ kifejezések.

$$\alpha + \beta$$

$$\mathtt{f}(\alpha, \beta, \gamma)$$

Don't do that!

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

Szekvenciapont C-ben

- Teljes kifejezés végén
- Függvényhívás aktuális paraméterlistájának kiértékelése végén
- Lusta operátorok első operandusának kiértékelése után
- Vessző operátornál

Vessző-operátor

```
<expression> ::= ...
               | <expression> ,  <expression>
```

- Az eredménye a jobboldali kifejezés eredménye
- Alacsony precedenciaszintű

```
int i = 1, v;
v = (i++, ++i);   /* nem ugyanaz, mint: v = i++, ++i; */
```

Vessző: operátor vagy elválasztójel

```
int i = 1, v;
if( v = f(i,i), v > i )
    v = f(v,v), i += v;

for( i = f(v,v), v = f(i,i); i < v; ++i, --v ){
    printf("%d %d\n",i,v);
}
```