Imperatív programozás

Kifejezések

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. augusztus 20.

Tartalomjegyzék

Példák

```
n + 1
```

```
3.14 * r * r
```

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

```
x < 3.14
```

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

Lexikális elemek

- Literálok
- Operátorok
- Azonosítók
- Zárójelek
- Egyéb jelek, pl.: vessző

Függvényhívás szintaxisa

```
<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`

Operátorok fixitása

- Prefix, pl.: `++c`
- Postfix, pl.: `c++`
- Infix, pl.: `x + y`
- Mixfix, pl.: `x < 0 ? 0 : x`

Értelmes kifejezések

- A benne szereplő azonosítók deklaráltak
  (Önmagában az `n + 1` nem értelmes)
- Jól típusozott
  (Van értelmük C-ben: `3 + 3.14`, `"hello" + 1`,
  nincs értelme C-ben: `"hello" * 42`)
- Típushelyes, de attól még nincs értelme
  (pl.: `1 / 0`)

Pure és impure nyelvek

- Impure (pl. C)
  - Kifejezés értékének meghatározása
  - Mellékhatás
- Pure (pl. Haskell)
  - Kifejezés értékének meghatározása
  - Hivatkozási helyfüggetlenség (referential transparency)

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.)

Értékadás

Értékadó utasítás

```
n = 1;
```

Mellékhatásos kifejezés

```
n = 1
```

Mellékhatásos kifejezés értéke

`(n = 1)` értéke `1`

Érték továbbgyűrűzése

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

Mellékhatás

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

Eggyel növelő/csökkentő operátorok

```
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 (továbbá: `||`, illetve `?:`)

```
int f() { printf("f"); return 1; }
int g() { printf("g"); return 2; }

int  i  = f() + g();
/* Kimenet: fg, Eredmény: i == 3 */

bool b1 = (f() == 0) && (g() == 2);
/* Kimenet: f, Eredmény: b1 == false */

bool b2 = (f() == 1) || (g() == 3);
/* Kimenet: f, Eredmény: b2 == true */
```

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:

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

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

Példa

Keressük meg egy tömb első negatív elemének indexét

```
/* Jó */
for (i = 0; i < LENGTH && array[i] >= 0; ++i);
/* Rossz */
for (i = 0; array[i] >= 0 && i < LENGTH; ++i);
```

Mellékhatás a lusta operátor operandusaiban

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

Operandusok, függvényparaméterek kiértékelési sorrendje

Kiértékelési sorrend nem specifikált

```
int f() { printf("f"); return 1; }
int g() { printf("g"); return 2; }

void printSum(int a, int b) { printf("%d\n", a + b); }

printf("%d\n", f() + g());      /* Kimenet: fg3 vagy gf3 */
printSum(f(), g());             /* Kimenet: fg3 vagy gf3 */
```

Ilyet ne csináljunk!

```
int i = 2;
int j = i-- - --i;
printf("%d\n", j);              /* Kimenet: 2 vagy 0     */
```

Szekvenciapont

- 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 f() { printf("f"); return 1; }
int g() { printf("g"); return 2; }
int h() { printf("h"); return 3; }

printf("%d\n", (f(), g(), h()));      /* Kimenet: fgh3 */
```