Kifejezések

Kozsik Tamás

December 11, 2016

Kifejezések

- Lexika
- Szintaktika
- Szemantika

Lexika

- azonosítók (változó-, metódus-, típus- és csomagnevek)
- literálok
- operátorok, pl. +
- zárójelek: (), [], {}, <>
- vessző, pont

Lexika: literálok

- karakter, sztring, egész szám, lebegőpontos szám,

```
'c', '\ubabe' // char típusú
"hello"       // java.lang.String típusú
12, 12L, 12l  // int és long típusú
12.3, -12.3E5, -12.3E-5D, -12.3E-5d  // double
12.3F, -12.3E-5F, 12.3f  // float
1_000_000  // tagolás az olvashatóság érdekében
```

- speciális: `null`, `this`, `super`, `true`, `false`

Szintaktika

- helyes zárójelezés
- operátorok arítása
  - unáris, pl. `-`
  - bináris, pl. `*`
  - ternáris: `x>0 ? 2 : 3`
  - *n*-áris: `()` (C++ függvényalkalmazás operátor)
- operátorok fixitása
  - prefix, pl. `!a`, `++a`
  - postfix, pl. `a++`
  - infix, pl. `a * b`
  - mixfix, pl. `a ? b : c`

```
n = 4;
n++;    // értéke 4, mellékhatás: n vált. 5-re
++n;    // értéke 6, mellékhatás: n vált. 6-ra
```

- tömb kifejezés: `{1,3,5,7}`

Szemantika

- precedencia
  - pl. `1+2*3` $\equiv$ `1+(2*3)` $\rightarrow$ 7, míg `(1+2)*3 == 9`
  - redundáns zárójelezés segítheti a kód olvasását
- asszociativitás
  - azonos precedenciaszintű operátorok zárójelezése
  - pl. `4/2*5` $\equiv$ `(4/2)*5` $\rightarrow$ 10 (balasszociatív operátorok)
  - pl. `x = y = 5` jelentése: `x = (y = 5)`
    - azaz `y`-ba 5 kerül, az `y=5` kifejezés értéke 5, `x`-be 5 kerül
    - jobbasszociatív operátor
- operandusok kiértékelési sorrendje
- tisztaság, mellékhatások
- lustaság, mohóság

Kiértékelés problémái

- véges értéktartomány
  - számábrázolás: véges sok biten
  - túl- és alulcsordulás
  - "nagy számok", pl. `java.lang.BigInteger` is valójában véges (amennyi memóriánk van)
- nem terminálló számítások

A matematikusokhoz képest nehéz dolga van a programozóknak!

Két egész szám átlaga?

Túlcsordulás nélkül számítsuk ki: (a+b)/2

```
int avg( int a, int b ){
    int halfA = a/2, halfB = b/2;
    if( a >= 0 && b >= 0 ){          // a+b may be too large
        if( halfA + halfA < a && halfB + halfB < b )
            return halfA + halfB + 1;
        else
            return halfA + halfB;
    } else if( a < 0 && b < 0 ){  // a+b may be too small
        if( halfA + halfA > a && halfB + halfB > b )
            return halfA + halfB - 1;
        else
            return halfA + halfB;
    } else return (a+b)/2;    // different signums, safe!
}
```

Számábrázolás

- Kettes komplemens egészek ábrázolására
  - értéktartomány n bit esetén: -2^{n-1} .. $2^{n-1}-1$
  - pl. $n = 8$ (`byte`) esetén `-128..127`
  - több negatív érték, mint pozitív, tehát az unáris "-" művelet is hibás eredményt adhat: `b == -128` esetén `-b == -128`

```
for( byte b = 0; b<=127; ++b ){
    System.out.println(b);
}  // végtelen ciklus
```

- Lebegőpontos ábrázolás valós számokhoz
  - $\approx$ racionális számok egy részhalmaza
  - leggyakrabban IEEE 754 szabvány szerint `https://en.wikipedia.org/wiki/IEEE_floating_point`

Mellékhatásos kifejezések versus “tisztaság”

- írhatók mellékhatásos függvények
- vannak mellékhatásos operátorok
  - pl. =, +=, ++ (prefix és postfix)
  - szinte mindig a mellékhatása miatt használjuk, pl. `n = 1` kifejezés
    - értéke: 1
    - mellékhatása: `n`-nek értékül adja az 1-et
- vannak idiómák, melyek a mellékhatásos operátorra építenek

```
BufferedReader in = ...
String str;
while( (str=in.readLine()) != null ){ ... }
```

- ökölszabály: szeretjük a “tiszta” (mellékhatásmentes) függvényeket!

Kifejezés versus utasítás

- kifejezés plusz pontosvessző: utasítás
- kiértékeli a kifejezést
- jellemzően akkor használjuk, amikor mellékhatása is van
- például: értékadás

```
n = 5;
n++;
```

Operandusok kiértékelési sorrendje

- Egyes nyelvekben (pl. Java) kötött: balról jobbra
  - pl. α op β kiértékelése:
    - először α
    - azután β
    - végül az op művelet a két kiértékelt operanduson
  - hasonlóan $f(\alpha, \beta, \gamma)$ kiértékelése
  - ez más, mint a precedencia és asszociativitás
- Sok nyelvben (pl. C++) a fordítóprogram dönti el, azaz nem feltétlenül egyértelmű, milyen eredményt kapunk
  - több lehetséges eredmény a mellékhatások miatt
  - a hatékonyságot befolyásolja, ezért ez egy optimalizációs módszer a fordító számára!
- Jobb, ha kerüljük az olyan kifejezéseket, ahol ez számít!

```
c++ + ++c
```

Lustaság versus mohóság

- Mohó kiértékelés: először argumentum/operandus, utána művelet
  - mohó operátorok, pl. +
  - függvényhívások
- Lusta operátorok kiértékelése:
  - Nem feltétlenül értékel ki minden operandust
    - `&&` és || rövidzár logikai operátorok
    - pl. α `&&` β kiértékeléséhez nem értékeli ki β-t, ha α hamisra értékelődött ki
  - Nem értékel ki minden operandust
    - `?:` operátor legfeljebb két operandust a háromból
  - Van logikai operátor mohó és lusta változatban is
    - `&` és | versus `&&` és ||

Lehet más a lusta és a mohó?

- A logikában nem. De a programozóknak nehezebb dolguk van, mint a matematikusoknak.
- A különbség lehetséges okai, pl. $\alpha \wedge \beta$ esetben:
  - lehet mellékhatás a β részkifejezésben;
  - lehet, hogy β kiértékelése nem mindig terminál;
  - lehet, hogy β kiértékelése néha kivételt vált ki.
    - idióma:

```
while( i < t.length && t[i] != 0 ){
    ++i;
}
```

Lusta és mohó művelettábla

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 (a mellékhatásoktól itt eltekintünk):

α && β	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞

α & β	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\bot$	∞
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞