Programozási nyelvek és paradigmák

Procedurális programozás Eiffelben

Kozsik Tamás (2020)

Alprogramok Eiffelben

- Feature egy osztálydefinícióban
- Objektum művelete
- Nincs „osztályszintű'' vagy „globális'' alprogram
- Rekurzió: megengedett
- Operátorok készíthetők

Paraméterátadás

- Kifejtett típusú: érték szerint (call-by-value)
- Referencia típusú: megosztás szerint (call-by-sharing)

Veszély

```
class DATUM
    ...
feature

    ...

    make_masnap( d: attached DATUM )
        -- váljunk d másnapjává
        do ...  end

    masnapra_lep
        -- lépjünk egy napot
        do
            make_masnap(Current)
        end -- masnapra_lep

end --class DATUM
```

Veszély

```
class FRACTION
create
   set
feature
   numerator, denominator: INTEGER
   ...
   divide_by( other: attached FRACTION )
      require
         other.numerator /= 0
      do
         numerator := numerator * other.denominator
         denominator := denominator * other.numerator
      end
invariant
   denominator /= 0
end
```

Veszély: f.divide_by(f)

```
class FRACTION
create
   set
feature
   numerator, denominator: INTEGER
   ...
   divide_by( other: attached FRACTION )
      require
         other.numerator /= 0
      do
         numerator := numerator * other.denominator
         denominator := denominator * other.numerator
      end
invariant
   denominator /= 0
end
```

Javítva – expanded + érték szerinti paraméterátadás

```
expanded class FRACTION
feature
   numerator: INTEGER
   denominator: INTEGER attribute Result := 1 end
   ...
   divide_by( other: FRACTION )
      require
         other.numerator /= 0
      do
         numerator := numerator * other.denominator
         denominator := denominator * other.numerator
      end
invariant
   denominator /= 0
end
```

Javítva – aliasing kizárása

```
class FRACTION
feature
   numerator: INTEGER
   denominator: INTEGER attribute Result := 1 end
   ...
   divide_by( other: attached FRACTION )
      require
         other.numerator /= 0
         other /= Current
      do
         numerator := numerator * other.denominator
         denominator := denominator * other.numerator
      end
invariant
   denominator /= 0
end
```

Javítva – aliasing ellenőrzése

```
divide_by( other: attached FRACTION )
    require
        other.numerator /= 0
    do
        if Current = other then
            numerator := 1
            denominator := 1
        else
            numerator := numerator * other.denominator
            denominator := denominator * other.numerator
        end
    end
```

Gond-e a Current.make_masnap(Current)?

```
make_masnap( d: attached DATUM ) -- váljunk d másnapjává
    do
        if d.nap = d.napok_szama_a_honapban then
            if d.honap = December then
                ev := d.ev + 1
                honap := Januar
                nap := 1
            else
                ev := d.ev
                honap := d.honap + 1
                nap := 1
            end
        else
            ev := d.ev
            honap := d.honap
            nap := d.nap + 1
        end
    end -- make_masnap
```

Imperatív és funkcionális stílusú rutin

```
class FRACTION
create set
feature
  numerator, denominator: INTEGER
  ...
  divide_by (other:attached FRACTION)
    ...
  divided_by(other:attached FRACTION):attached FRACTION
    require
      other.numerator /= 0
    do
      create Result.set(numerator * other.denominator,
                        denominator * other.numerator)
    end
  ...
end
```

Imperatív és funkcionális stílusú rutin hívása

```
local
    p, q, r: attached FRACTION
do
    create p.set(3,4)
    create q.set(5,3)
    r := p.divided_by(q)
    p.divide_by(q)
```

Imperatív és funkcionális stílusú rutin hívása

```
local
    p, q, r: attached FRACTION
do
    create p.set(3,4)
    create q.set(5,3)
    r := p.divided_by(q)
    p.divide_by(q)

    p := p.divided_by(q)
    p.divide_by(q)
```

Operátorok túlterhelése

```
class FRACTION
create
  set
feature
  numerator, denominator: INTEGER
  ...
  divided_by alias "/" ( other:attached FRACTION ) :
                                        attached FRACTION
    require
      other.numerator /= 0
    do
      create Result.set( numerator * other.denominator ,
                         denominator * other.numerator )
    end
  ...
end
```

Operátorok túlterhelése

```
class FRACTION
create
  set
feature
  numerator, denominator: INTEGER
  ...
  divided_by alias "/" ( other:attached FRACTION ) :
                                        attached FRACTION
    require
      other.numerator /= 0
    do
      create Result.set( numerator * other.denominator ,
                         denominator * other.numerator )
    end
  ...
end
```

Hívás: `p.divided_by(q)` vagy `p / q`

Operátorok túlterhelése: régi szintaxis

```
class FRACTION
create
  set
feature
  numerator, denominator: INTEGER
  ...
  infix "/" ( other:attached FRACTION ) :
                                        attached FRACTION
    require
      other.numerator /= 0
    do
      create Result.set( numerator * other.denominator ,
                         denominator * other.numerator )
    end
  ...
end
```

Hívás: `p / q`

Prefix operátor: régi szintaxis

```
prefix "-": attached FRACTION
    do
        create Result.set (-numerator, denominator)
    end
```

Hívás: `-p`

Prefix operátor: új szintaxis

```
negated alias "-": attached FRACTION
    do
        create Result.set (-numerator, denominator)
    end
```

Hívás: `p.negated` vagy `-p`

Operátorok

- Aritmetikai, pl.

```
i / j          i // j          i \\ j          i^j
```

Operátorok

- Aritmetikai, pl.

  `i / j` `i // j` `i \\ j` `i^j`

- Relációs

  `<` `<=` `>` `>=` `=` `/=` `~` `/~`

Operátorok

- Aritmetikai, pl.

  `i / j` `i // j` `i \\ j` `i^j`

- Relációs

  `<` `<=` `>` `>=` `=` `/=` `~` `/~`

- Logikai

  `not` `and` `or` `xor` `and then` `or else` `implies`

Operátorok

- Aritmetikai, pl.

  `i / j` `i // j` `i \\ j` `i^j`

- Relációs

  `<` `<=` `>` `>=` `=` `/=` `~` `/~`

- Logikai

  `not` `and` `or` `xor` `and then` `or else` `implies`

- Egyebek

  `.` `..` `old` `[,]`

Az `old` operátor

- Csak specifikációban (utófeltételben) szerepelhet
- Paramétertér

```
class ACCOUNT
...
feature
   balance: INTEGER

   deposit( amount: INTEGER )
      require
         amount > 0
      do
         balance := balance + amount
      ensure
         balance_updated: balance = old balance + amount
      end
...
```

Változás a művelet megkezdéséhez képest

$$A \;=\; \text{balance} : \mathbb{N}_0 \times \text{amount} : \mathbb{N}_0 \times ...$$

$$B \;=\; \text{balance}' : \mathbb{N}_0 \times \text{amount}' : \mathbb{N}_0$$

$$Q_{\text{balance}',\text{amount}'} \Rightarrow lf(\text{deposit}, R_{\text{balance}',\text{amount}'})$$

$$Q_{\text{balance}',\text{amount}'} \;=\; (\text{balance} = \text{balance}' \wedge \text{amount} = \text{amount}' > 0)$$

$$R_{\text{balance}',\text{amount}'} = (\text{balance} = \text{balance}' + \text{amount} \wedge \text{amount} = \text{amount}')$$

Nemváltozás

- Frame problem (McCarthy–Hayes, 1969)
- Frame rule $\in$ Separation logic (Reynolds és mások)
- Utófeltételben: mi nem változott meg
- Eiffelben?
  - Megadhatjuk
  - Kikövetkeztetés? (Meyer: Framing the frame problem, 2015)

Nemváltozás kifejezése

- frame condition, frame property

```
class ACCOUNT
...
feature
    balance: INTEGER
    id: INTEGER

    deposit( amount: INTEGER )
        require
            amount > 0
        do
            balance := balance + amount
        ensure
            balance_updated: balance = old balance + amount
            frame: id = old id
        end
...
```

Általában sok lekérdezés nem változik: only

- ECMA-szabvány, *estudio* nem támogatja

```
class ACCOUNT
...
feature
   balance: INTEGER
   id: INTEGER

   deposit( amount: INTEGER )
      require
         amount > 0
      do
         balance := balance + amount
      ensure
         balance_updated: balance = old balance + amount
         frame: only balance
      end
...
```

Általában sok lekérdezés nem változik: `old` és `strip`

- Régies jelölés, *estudio* támogatja

```
class ACCOUNT
...
feature
   balance: INTEGER
   id: INTEGER

   deposit( amount: INTEGER )
      require
         amount > 0
      do
         balance := balance + amount
      ensure
         balance_updated: balance = old balance + amount
         frame: strip(balance) ~ old strip(balance)
      end
...
```

Saját operátor definiálása

```
class POINT
create
  set
feature
  x,y : REAL_64
  ...
  distance alias "|-|" ( other: attached POINT ) :
                                        attached POINT
  ...
end
```

Hívás: `p |-| q` vagy p.distance(q)

Saját operátor definiálása: változatos karakterek

```
reciprocal alias "↕" : attached FRACTION
    require
      numerator /= 0
    do
      create Result.set( denominator, numerator )
    end
```

Hívás: ↕p

Szögletes zárójel – operátor

- Tömbök indexelése
- Rendezett n-esek (*manifest* tuple) megadása
- Mixfix, akárhány arítású lehet

Szögletes zárójel – operátor

- Tömbök indexelése
- Rendezett n-esek (*manifest* tuple) megadása
- Mixfix, akárhány arítású lehet

```
max_hely( arr: attached ARRAY[INTEGER] ): INTEGER
    require
        arr.count > 0
    ensure
        arr.lower <= Result
        Result <= arr.upper
        arr ~ old arr
        across arr as cur all cur.item <= arr[Result] end
    end -- max_hely
```

Maximumhely meghatározása

```
from
   Result := arr.lower ; m := arr.item(Result)
   i := Result
invariant
   arr.lower <= Result ; Result <= arr.upper
   arr.lower <= i; i <= arr.upper+1
   m = arr.item(Result)
   across (arr.lower |..| (i-1)) as c
                     all arr[c.item] <= m end
variant arr.upper - i + 1
until i > arr.upper
loop
   if arr.item(i) > m then
      Result := i
      m := arr.item(Result)
   end
   i := i + 1
end
```