Programozási nyelvek és paradigmák

Imperatív programozás Eiffelben

Kozsik Tamás (2020)

Változók

- Adattag
- Lokális változó
- Formális paraméter

```
class PERSON
create set_name
feature
    name: STRING
    set_name( new_name: STRING )
        local
            tmp: STRING
        do
            tmp := new_name.mirrored
            name := tmp.mirrored
        end
end
```

Változók tárolása

- Heap (szemétgyűjtés)
- Végrehajtási verem

Objektumok ábrázolása

Referencia mögött

```
class PERSON
    ...
end
```

Kifejtett módon

```
expanded class INTEGER
    ...
end
```

Objektum létrehozása

```
john: PERSON      -- Void értékre inicializálódik
id: INTEGER       -- 0 értékre inicializálódik
...
create john.set_name("John")                    -- utasítás
create id
...
... create{PERSON}.set_name("Peter") ...        -- kifejezés
```

Objektum létrehozása

```
john: PERSON      -- Void értékre inicializálódik
id: INTEGER       -- 0 értékre inicializálódik
...
create john.set_name("John")                    -- utasítás
create id
...
... create{PERSON}.set_name("Peter") ...        -- kifejezés

...
jack: ANY
...
create{PERSON} jack.set_name("Jack")
```

Objektum létrehozása

```
john: PERSON      -- Void értékre inicializálódik
id: INTEGER       -- 0 értékre inicializálódik
...
create john.set_name("John")                    -- utasítás
create id
...
... create{PERSON}.set_name("Peter") ...    -- kifejezés

...
jack: ANY
...
create{PERSON} jack.set_name("Jack")

!PERSON! jack.set_name("Jack")    -- régi szintaxis
```

Referenciák és aliasing

```
class PERSON
create set_name
feature
   name: STRING
   set_name( new_name: STRING )
      do
         name := new_name    -- jobb: := new_name.twin
      end
end
```

Változók írása

- Objektum a saját mezőjét
- Lokális változót

Hibás értékadások

```
class PERSON
create set_name
feature
    name: STRING                          -- nyilvános
    set_name( new_name: STRING )
       do
           new_name := name               -- fordítási hiba
       end
    set_friends_name( friend: PERSON )
       do
           friend.name := name            -- fordítási hiba
       end
end
```

Objektum megváltoztatása

- Megkérjük az objektumot, hogy változzon meg
- Rutint hívunk rá

Objektum megváltoztatása

- Megkérjük az objektumot, hogy változzon meg
- Rutint hívunk rá

```
class PERSON
create set_name
feature
    name: STRING
    set_name( new_name: STRING )
       do
           name := new_name.twin
       end
    set_friends_name( friend: PERSON )
       do
           friend.set_name( name )
       end
end
```

Syntax sugar: assign procedure

```
class
    PERSON
create
    set_name
feature
    name: STRING assign set_name
    set_name( new_name: STRING )
       do
           name := new_name.twin
       end
    set_friends_name( friend: PERSON )
       do
           friend.name := name   -- friend.set_name(name)
       end
end
```

Formális paraméter nem írható

```
gcd( a, b: INTEGER ): INTEGER      -- függvény
do
   from
   until
      a = b
   loop
      if a > b
      then a := a - b     -- fordítási hiba
      else b := b - a     -- fordítási hiba
      end
   end
   Result := a     -- visszatérési érték megadása
end -- gcd
```

Kitérő

- Kis- és nagybetűk nem számítanak
  - kulcsszavakban
  - azonosítókban
- Konvenció
  - kulcsszavak kisbetűvel
  - egyéb foglalt szavak: első betű nagy
    (`Result`, `Current`, `Void`, `True`, `False`, `Precursor` …)
  - konstans tagok: -“-

    ```
    Line_width: INTEGER = 256
    ```
  - osztálynevek végig naggyal
  - többi azonosító: végig kicsi, aláhúzással tagolt

Formális paraméter nem írható: segédváltozó kell

```
gcd( a, b: INTEGER ): INTEGER
local
   number: INTEGER
do
   from
      Result := a
      number := b
   until
      Result = number
   loop
      if Result > number
      then Result := Result - number
      else number := number - Result
      end
   end
end -- gcd
```

Utasítások

- Értékadás
- Elágazások
  - `if-then-elseif-else-end`
  - `inspect`
- Ciklusok
  - `from-until-loop-end`
  - `across`
- `check`
- `debug`

Nincsenek nem strukturált utasítások!

elseif-példa

```
szokoev: BOOLEAN
    do
        if ev \\ 400 = 0 then Result := True
        elseif ev \\ 100 = 0 then Result := False
        elseif ev \\ 4 = 0 then Result := True
        else Result := False
        end
    end
```

- Egész osztás: //
- Osztási maradék: \\

inspect-példa

```
napok_szama_a_honapban: INTEGER
    do
        inspect honap
            when 1,3,5,7,8,10,12 then Result := 31
            when 4,6,9,11 then Result := 30
            when 2 then
                if szokoev then Result := 29
                else Result := 28
                end
        end
    end -- napok_szama_a_honapban
```

inspect

```
Multi_branch     = "inspect" Expression [When_part_list]
                   [Else_part] "end"
When_part_list   = "when" {When_part "when" ...}+
When_part        = Choices "then" Compound
Choices          = {Choice "," ...}
Choice           = Constant | Interval
Interval         = Integer_interval | Character_interval
Integer_interval = Integer_constant ".." Integer_constant
Else_part        = "else" Compound
```

Iteráló ciklus

```
across
    <<1969, 7, 20, 20, 17, 40>> as i
loop
    print (i.item.out)
    print ("%N")
end
```

Iteráló kifejezés

Univerzális kvantálás

```
across <<7, 20, 20, 17, 40>> as i all i.item > 0 end
```

Egzisztenciális kvantálás

```
across <<7, 20, 20, 17, 40>> as i some i.item = 17 end
```

debug-utasítás

`debug` (*Debug_key*, ...) *instructions* `end`

`check`-utasítás

- Design by Contract

`check` *assertion* `end`

check-utasítás

- Design by Contract

`check` *assertion* `end`

```
check
   size_is_not_too_large: size <= capacity
   -- Size is not too large because ...
end
```