Massiivid

Massiivide BNF ja abstraktne süntaks:

⟨*exp*⟩ ::= ...

| `letarray` ⟨*arraydecls*⟩ `in` ⟨*exp*⟩

```
letarray (arraydecls body)
```

| ⟨*array-exp*⟩ `[`⟨*exp*⟩`]`

```
arrayref (array index)
```

| ⟨*array-exp*⟩ `[`⟨*exp*⟩`]` `:=` ⟨*exp*⟩

```
arrayassign (array index exp)
```

Massiivid

Massiivide BNF ja abstraktne süntaks (järg):

$$\langle \textit{array-exp} \rangle ::= \langle \textit{varref} \rangle \mid \texttt{(} \langle \textit{exp} \rangle \texttt{)}$$

$$\langle \textit{arraydecls} \rangle ::= \langle \textit{arraydecl} \rangle \{ \texttt{;} \langle \textit{arraydecl} \rangle \}^*$$

$$\langle \textit{arraydecl} \rangle ::= \langle \textit{var} \rangle \texttt{ [} \langle \textit{exp} \rangle \texttt{]} \qquad \texttt{decl (var exp)}$$

Näide:

```
letarray a[2]; b[1] in
   begin
     a[0] := 1; a[1] := 2; b[0] := 3;
     +(a[1],b[0])
   end
```

Massiivid

Massiivide kaudne (ingl. *indirect*) esitus:

$$\begin{aligned} \textit{Expressed Value} &= \textit{Number} + \textit{Procedure} + \textit{Array} \\ \textit{Denoted Value} &= \textit{Cell}(\textit{Expressed Value}) \\ \textit{Array} &= \{\textit{Cell}(\textit{Expressed Value})\}^* \end{aligned}$$

Näide:

```
--> "let p = proc (b) b[0] := 3 in
     letarray a[2] in
        begin
          a[0] := 1; a[1] := 2; p(a); a[0]
        end"
3
```

Massiividega interpretaator

Massiivide abstraktne andmetüüp:

```
(define make-array
  (lambda (dimension)
    (let ((array (make-vector (+ dimension 1))))
      (vector-set! array 0 '*array*)
      array)))

(define array?
  (lambda (x)
    (and (vector? x)
         (eq? (vector-ref x 0) '*array*))))
```

Massiividega interpretaator

Massiivide abstraktne andmetüüp (järg):

```
(define array-ref
  (lambda (array index)
    (vector-ref array (+ index 1))))

(define array-set!
  (lambda (array index value)
    (vector-set! array (+ index 1) value)))
```

Massiividega interpretaator

Massiivide abstraktne andmetüüp (järg):

```
(define array-whole-set!
  (lambda (dest-array source-array)
    (let ((source-len (vector-length source-array)))
      (letrec ((loop (lambda (n)
                       (if (< n source-len)
                           (begin
                             (vector-set! dest-array n
                                          (vector-ref source-array n))
                             (loop (+ n 1)))))))
        (loop 1)))))
```

Massiividega interpretaator

Massiivide abstraktne andmetüüp (järg):

```
(define array-copy
  (lambda (array)
    (let ((new-array (make-array
                       (- (vector-length array) 1))))
      (array-whole-set! new-array array)
      new-array)))
```

Massiividega interpretaator

Avaldiste väärtustamine:

```
(define eval-exp (lambda (exp env)
  (variant-case exp
    (varref (var) (denoted->expressed
                   (apply-env env var)))
    (app (rator rands)
      (apply-proc (eval-rator rator env)
                  (eval-rands rands env)))
    (varassign (var exp)
      (denoted-value-assign! (apply-env env var)
                             (eval-exp exp env)))
    ...
```

Massiividega interpretaator

Avaldiste väärtustamine (järg):

```
...
(letarray (arraydecls body)
  (eval-exp body
    (extend-env (map decl->var arraydecls)
      (map (lambda (decl)
             (do-letarray
                 (eval-exp (decl->exp decl) env)))
           arraydecls)
      env)))
...
```

Massiividega interpretaator

Avaldiste väärtustamine (järg):

```
...
(arrayref (array index)
  (array-ref (eval-array-exp array env)
             (eval-exp index env)))
(arrayassign (array index exp)
  (array-set! (eval-array-exp array env)
              (eval-exp index env)
              (eval-exp exp env)))
...
```

Massiividega interpretaator

Operaatori ja operandide väärtustamine:

```
(define eval-rator
  (lambda (rator env)
    (eval-exp rator env)))

(define eval-rands
  (lambda (rands env)
    (map (lambda (rand) (eval-rand rand env)) rands)))

(define eval-rand
  (lambda (exp env)
    (expressed->denoted (eval-exp exp env))))
```

Massiividega interpretaator

Abiprotseduurid:

```
(define denoted->expressed cell-ref)

(define denoted-value-assign! cell-set!)

(define do-letarray
  (lambda (x)
    (make-cell (make-array x))))

(define eval-array-exp eval-exp)

(define expressed->denoted make-cell)
```

Massiivid

Massiivide otsene (ingl. *direct*) esitus:

$$
\begin{aligned}
\mathit{Expressed\ Value} &= \mathit{Number} + \mathit{Procedure} + \mathit{Array} \\
\mathit{Denoted\ Value} &= \mathit{Cell}(\mathit{Number}) + \mathit{Cell}(\mathit{Procedure}) + \mathit{Array} \\
\mathit{Array} &= \{\mathit{Cell}(\mathit{Number} + \mathit{Procedure})\}^*
\end{aligned}
$$

Näide:

```
--> "let p = proc (b) b[0] := 3 in
     letarray a[2] in
        begin
          a[0] := 1; a[1] := 2; p(a); a[0]
        end"
1
```

Massiivide otsese esitusega interpretaator

```
(define denoted->expressed
  (lambda (den-val)
    (if (array? den-val)
        den-val
        (cell-ref den-val))))

(define expressed->denoted
  (lambda (exp-val)
    (if (array? exp-val)
        (array-copy exp-val)
        (make-cell exp-val))))
```

Massiivide otsese esitusega interpretaator

```
(define denoted-value-assign!
  (lambda (den-val exp-val)
    (cond
      ((not (array? den-val))
          (cell-set! den-val exp-val))
      ((array? exp-val)
          (array-whole-set! den-val exp-val))
      (else (error "Must assign array:" den-val)))))

(define do-letarray make-array)

(define eval-array-exp eval-exp)
```

Massiivide otsese esitusega interpretaator

```
(define array-set!
  (lambda (array index value)
    (if (array? value)
        (error "Cannot assign array to element" value)
        (vector-set! array (+ index 1) value))))
```

Parameetrite edastamise viisid

- Call-by-value — parameetrid edastatakse väärtuse kaudu; s.o. protseduuri parameetriga seotakse tegeliku argumendi väärtuse koopia
  - kõik seni vaadeldud interpretaatorid realiseerisid parameetrite edastamise “call-by-value”
- Call-by-reference — parameetrid edastatakse viida kaudu

```
let a = 3;  b = 4;
    p = proc (x) x := 5
in begin p(a); p(b);
         +(a,b)
   end
```

Viida kaudu edastamine

Muutujate vahetamine:

```
--> "let swap = proc (x,y)
                  let temp = 0 in
                  begin temp := x;
                        x    := y;
                        y    := temp
                  end;
        b = 4;
        c = 3
     in begin swap(b,c); b end"
3
```

Viida kaudu edastamine

Massiivi elementide edastamine viida kaudu:

```
--> "let b = 2 in
     letarray a[3] in
     begin
         a[1] := 5;
         swap(a[1], b);
         a[1]
     end"
2
```

Viida kaudu edastamine

Üldiste avaldiste edastamine viida kaudu:

```
--> "let c = 3;
         p = proc (x) x := 5
     in begin
          p(add1(c));
          c
        end"
3
```

Viida kaudu edastamine

Aliased — erineva nimega, kuid samale väärtusele viitavad muutujad:

```
--> "let b = 3;
         p = proc(x, y)
               begin
                x := 4;
                y
               end
     in p(b, b)"
4
```

Viida kaudu edastamine

Aliased raskendavad programmidest arusaamist:

```
swap2 = proc (x, y)
         begin
           x := +(x, y); y := -(x, y); x := -(x, y)
         end
---> "let b = 1; c = 2 in
      begin swap2(b, c); c end"
1
--> "let b = 1
     in begin swap2(b, b); b end"
0
```

"Call-by-reference" interpretaator

Operandide BNF ja abstraktne süntaks:

$$
\begin{array}{lcll}
\langle \textit{operand} \rangle & ::= & \langle \textit{varref} \rangle & \\
 & | & \langle \textit{array-exp} \rangle \texttt{[}\langle \textit{exp} \rangle\texttt{]} & \texttt{arrayref (array index)} \\
 & | & \langle \textit{exp} \rangle &
\end{array}
$$

Andmemudel:

$$
\begin{array}{lcl}
\textit{Denoted Value} & = & \textit{L-Value} \\
\textit{L-Value} & = & \textit{Cell}(\textit{Expressed Value}) \\
 & + & \textit{Array Element}(\textit{Expressed Value})
\end{array}
$$

“Call-by-reference” interpretaator

Operandide väärtustamine:

```
(define-record ae (array index))


(define eval-rand
  (lambda (rand env)
    (variant-case rand
      (varref (var) (apply-env env var))
      (arrayref (array index)
         (make-ae (eval-array-exp array env)
                  (eval-exp index env)))
      (else (make-cell (eval-exp rand env))))))
```

“Call-by-reference” interpretaator

```
(define denoted->expressed
  (lambda (den-val)
    (cond
      ((cell? den-val) (cell-ref den-val))
      ((ae? den-val)
         (array-ref (ae->array den-val)
                    (ae->index den-val)))
      (else (error "Can't dereference"
                   "denoted value:" den-val)))))
```

“Call-by-reference” interpretaator

```
(define denoted-value-assign!
  (lambda (den-val val)
    (cond
      ((cell? den-val) (cell-set! den-val val))
      ((ae? den-val)
         (array-set! (ae->array den-val)
                     (ae->index den-val) val))
      (else (error "Can't assign"
                   "to denoted value:" den-val)))))
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 6.1 – 6.3
  - NB! EOPL 6.3 — “call-by-value-result” ja “call-by-result”
- “Mängida” interpretaatoritega
  - `loeng13-1.ss` — “call-by-value” kaudsete massiividega;
  - `loeng13-2.ss` — “call-by-value” otseste massiividega;
  - `loeng13-3.ss` — “call-by-reference” (kaudsete massiividega).