Programmeren – Blok B

http://www.win.tue.nl/~wstomv/edu/2ip05/

College 8

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

Onderwerpen

- Verdelen van oplossing over routines (om te kunnen heersen)
- **record**-type: 'verdeel en heers' toegepast op gegevens

Afwegingen bij specificeren van routines

- Wanneer en hoe oplossing verdelen over aparte routines?
- Oplossen met **procedure** of met **function**?
- Welke parameters introduceren, met welke namen, welke typen?
- Welke soort parameters: **const**, **out**, **var**, value?
- In welke volgorde parameters vermelden?
- Hoever een afgesplitst probleem generaliseren?

Wanneer en hoe oplossing verdelen over aparte routines

- Streef naar *blokken* met hooguit een (paar) dozijn opdrachten en beperkte *nesting*-diepte.

  Blok = hoofdprogramma of routine

  Dreigt het meer te worden, overweeg dan (verder) verdelen.

  Geneste repetities duiden er op dat verder opdelen gewenst is.

- Los één probleem per routine op.

  Niet meer en niet minder.

  Als specificatie van routine omslachtig is, dan vaak verkeerd.

Procedure versus function

procedure p (... ; **out** x: T); versus **function** f (...): T;

Aspect	procedure	function
Aantal resultaten	onbeperkt	één
Resultaat-type	onbeperkt	scalair
Aanroepbaar in formule	nee	ja
const-parameters	ja	ja
out/**var**-parameters	ja	afgeraden
value parameters	ja	ja

Als *duur* resultaat altijd wordt opgeslagen, dan procedure overwegen:

p (... , v) versus $v := f$ (...)

Welke parameters

- Streef naar routines met hooguit enkele parameters.

  Neem eventueel parameters samen in een **record** of een **set**.

```
1 type
2   Option = ( LeftAlign, RightAlign, Verbose, Newline );
3   OptionSet = set of Option;
4
5 procedure WriteStuff ( ...; const opt: OptionSet );
6   { pre: ...; post: ... }
7   begin ... end;
8
9 ... WriteStuff ( ..., [ LeftAlign, Newline ] ) ...
```


Welke soort parameters

- Gebruik *geen* **out**/**var**-parameters in functies.
- Gebruik **out**-parameters in procedures om resultaten te exporteren.
- Gebruik **var**-parameters in procedures om waardes te im- en exporteren, maar *niet* om *alleen* waardes te importeren (dan: **const** of value).
- Gebruik value parameters alleen als het bijkomende kopiëren naar lokale variabelen nuttig is (bijv. om aliasing te vermijden).
- Geef grotere structuren bij voorkeur *niet* door als value parameter, maar als **const**-parameter.

  Vaak kan volstaan worden met kopiëren van een *deel* ervan.

In welke volgorde parameters vermelden

Er zijn verscheidene stijlen:

- Volgorde van gebruik en in specificatie: Eerst in, dan uit

  procedure $abcFormule$ (**const** a, b, c: $Real$; **out** $x1, x2$: $Real$);
  { **pre**: $a > 0 \wedge b^2 - 4ac \geq 0$
  post: $(\forall\, x :: ax^2 + bx + c \approx a(x - x1)(x - x2)) \wedge x1 \leq x2$ }

- Volgorde à la toekenning $v := E$: Eerst uit, dan in
- Eerst primair, dan secundair

Hoever een afgesplitst probleem generaliseren

- Vermijd gebruik van globale variabelen in routines.

  Generaliseer m.b.v. parameters.

- Denk altijd even na over mogelijkheden voor generalisatie.

  Generalisatie vergroot hergebruiksmogelijkheid en inzicht.
  Kan ook oplossen vereenvoudigen.

- Generaliseer wanneer kosten tegen baten opwegen.

  Pos: extra parameter met startindex voor zoeken in string

Dobbelen (3): Bepaal aantal keer dat worp voorkomt

```
1  procedure BepaalAantal ( const wl: WorpLijst; w: Waarde;
2                           out a: Integer );
3    { pre:  True
4      post: a = (# s: s in Speler: wl[s] = w) }
5    var
6      s: Speler; { doorloopt wl }
7    begin
8      a := 0  { a = (# i: i in [1..s): wl[i] = w) }
9
10   ; for s := 1 to NSpelers do begin
11         a := a + ord ( wl[s] = w )
12     end { for s }
13
14   end; { BepaalAantal }
```


Bepaal aantal keer dat worp voorkomt (met functie)

```
1  function Aantal ( const wl: WorpLijst; w: Waarde ): Integer;
2    { pre: True
3      ret: (# s: s in Speler: wl[s] = w) }
4    var
5      s: Speler;  { doorloopt wl }
6    begin
7      Result := 0  { Result = (# i: i in [1..s): wl[i] = w) }
8
9    ; for s := 1 to NSpelers do begin
10         Result := Result + ord ( wl[s] = w )
11     end { for s }
12
13   end; { Aantal }
```


Druk eventuele winnaar af (met twijfelachtige procedure)

```
1  procedure DrukWinnaarAf ( const wl: WorpLijst;
2                            w: Waarde; a: Integer );
3    { pre : w = max wl, a = (# s: s in Speler: wl[s] = w)
4      post: eventuele winnaar (werper unieke maximum) is afgedrukt
5    var
6      winnaar: Speler; { eventuele winnaar }
7    begin
8      if a = 1 then begin
9        BepaalWerper ( wl, w, winnaar )
10     ; writeln ( 'Speler ', winnaar, ' wint' )
11     end { then }
12     else
13       writeln ( 'Geen winnaar' )
14   end; { DrukWinnaarAf }
```

Verdeel en heers voor data: het record-type

Bundelt een stel variabelen, mogelijk met verschillende types:

record
⟨*veldnaam*⟩ : ⟨*type*⟩ ;
. . . ;
⟨*veldnaam*⟩ : ⟨*type*⟩
end

type
TLocatie = **record**
kolom: 'a' .. 'h';
rij: 1 .. 8;
end;

Gebruik van record-type

. . . ⟨*recordvarnaam*⟩ . ⟨*veldnaam*⟩ . . .

with ⟨*recordvarnaam*⟩ **do** . . . ⟨*veldnaam*⟩ . . .

var
locatie: *TLocatie*;

. . .
writeln (*locatie.kolom*, *locatie.rij*)

{ equivalent met }

with *locatie* **do begin**
writeln (*kolom*, *rij*)
end

Voorbeeld van record-type definitie bij KeizerKiezer

const
$MaxN = 10000$; { maximale aantal kandidaten, ≥ 1 }

type
$Kandidaat = 0 \,..\, MaxN - 1$; { de kandidaten, eigenlijk $0 \,..\, N - 1$ }

Kring = **record**
N: $1 \,..\, MaxN$; { aantal kandidaten bij aanvang }
inKring: **array** [*Kandidaat*] **of** *Boolean*;
{ *inKring*[i] = "kandidaat i staat nog in de kring" $(0 \leq i < N)$ }
rest: $1 \,..\, MaxN$; { aantal overgebleven kandidaten }
{ $rest = (\#\, i : 0 \leq i < N : inKring[i])$ }
aangewezen: *Kandidaat*; { doorloopt de kandidaten }
{ er geldt *inKring*[*aangewezen*] }
end; { *Kring* }

Specificatie van routines werkend op record-type

procedure *initKring* (**out** *k*: *Kring* ; **const** *a*: *Integer*);
{ **pre**: $1 \leq a \leq MaxN$
post: k is kring met a kandidaten, en 0 aangewezen }

procedure *wijsVolgendeAan* (**var** *k*: *Kring*);
{ **pre**: Zij $k = K$
post: $k = K$ met $k.aangewezen$ = volgende in kring na $K.aangewezen$ }

procedure *verwijderAangewezene* (**var** *k*: *Kring*);
{ **pre**: Zij $k = K$ met $k.rest \geq 2$
post: $k = K$ met $k.aangewezen$ = volgende in kring na $K.aangewezen$
$K.aangewezen$ is verwijderd uit de kring }

Gebruik van routines werkend op record-type

```
var
   n: 1 .. MaxN; { aantal kandidaten (invoer) }
   deKring: Kring; { de kring met kandidaten }

begin
   readln ( n )
;  initKring ( deKring, n )

;  while deKring.rest ≠ 1 do begin
      wijsVolgendeAan ( deKring )
   ;  wijsVolgendeAan ( deKring )
   ;  verwijderAangewezene ( deKring )
   end

;  writeln('Kandidaat ', deKring.aangewezen, ' wordt keizer' )
```


Voorbeeld van record-type gebruik

```
procedure initKring ( out k: Kring; const a: Integer );
  { pre: 1 ≤ a ≤ MaxN
    post: k is kring met a kandidaten, en 0 aangewezen }
  var
    i: Kandidaat; { doorloopt kandidaten }
  begin
    with k do begin
      N := a
    ;  for i := 0 to N − 1 do begin
         inKring[i] := True
       end { for i }
    ;  rest := N
    ;  aangewezen := 0
    end { with k }
  end; { initKring }
```


Voorbeeld van record-type gebruik (2)

```
procedure wijsVolgendeAan ( var k: Kring );
  { pre: Zij k = K
    post: k = K met k.aangewezen = volgende in kring na K.aangewezen }

  begin
    with k do begin

      repeat
        aangewezen := (aangewezen + 1) mod N
      until inKring[aangewezen]

    end { with k }
  end; { wijsVolgendeAan }
```


Voorbeeld van record-type gebruik (3)

```
1  procedure verwijderAangewezene ( var k: Kring );
2    { pre:  Zij k = K, met $k.\id{rest} \ge 2$
3      post: k = K waarbij K.aangewezen is verwijderd en
4            diens opvolger is aangewezen }
5    begin
6      with k do begin
7        inKring [ aangewezen ] := False
8      ; rest := rest - 1
9      ; wijsVolgendeAan ( k )
10     end { with k }
11   end; { verwijderAangewezene }
```

Gevolgen van wijziging bij KeizerKiezer

```
type
  Kring = record
    N: 1 .. MaxN;   { aantal kandidaten bij aanvang }
    opvolger: array [ Kandidaat ] of Kandidaat;
      { opvolger[i] = "wie op kandidaat i volgt in de kring"  (0 ≤ i < N) }
    rest: 1 .. MaxN;   { aantal overgebleven kandidaten }
    aangewezen: Kandidaat; { doorloopt de kandidaten }
      { aangewezen] kandidaat staat in de kring }
  end; { Kring }
```

Hoofdprogramma ongewijzigd

Alleen definities van type *Kring* en de routines erop veranderen

Gevolgen van wijziging bij KeizerKiezer

```
procedure wijsVolgendeAan ( var k: Kring );
  { pre: Zij k = K
    post: k = K met k.aangewezen = volgende in kring na K.aangewezen }

  begin
    with k do begin
      aangewezen := opvolger [ aangewezen ]
    end { with k }
  end; { wijsVolgendeAan }
```


Huiswerkopgave: Energiepillen

Beschouw deze 3×4 matrix:

0	1	30	5
2	10	0	3
4	20	7	99

Wat is de maximale padsom bij enige wandeling van linksboven naar rechtsonder, waarbij alleen naar rechts en naar onder gestapt wordt?

(Hoeveel van dergelijke wandelingen zijn er?)

Energiepillen: analyse

Er zijn $\binom{3+4-2}{3-1} = \binom{5}{2} = 10$ wandelingen mogelijk.

Van de 5 stappen moet je er 2 naar onder kiezen en 3 naar rechts.

In 20×20 matrix: circa 35 miljard wandelingen!

0	1	30	5
2	10	0	3
4	20	7	99

De maximale padsom bedraagt hier 138.

Energiepillen: algoritme

Generaliseer het probleem:

Wat is de maximale padsom van enige monotone wandeling vanuit (i, j) naar rechtsonder?

Algoritme: reken terug vanaf eindpunt

invoermatrix

0	1	30	5
2	10	0	3
4	20	7	99

max. padsom vanuit (i, j)

138	138	137	107
138	136	106	102
130	126	106	99