Programmeren – Blok B

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

College 7

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

Drie aspecten van routines

- Specificeren van deelproblemen: verdeel
- Maken van deeloplossingen: en
- Gebruiken van deeloplossingen: heers

Deze week:

- Specificaties (contracten) *lezen* om routines te *maken*

Contract: zorg en nut

		Contract		
		Preconditie		**Postconditie**
Partij	**Gebruiker**	zorg		nut
		↓		↑
	Maker	nut	→	zorg

Gebruiker zorgt dat preconditie geldt.

Maker benut deze preconditie en zorgt dat postconditie geldt.

Gebruiker benut deze postconditie.

Pascal programma gezien als routine

program ⟨*prognaam*⟩ ;
⟨*blok*⟩ .

waarbij ⟨*blok*⟩ staat voor

{ ⟨*specificatie (contract) en toelichting*⟩ }

⟨*definities en variabelen*⟩

begin
⟨*lijst opdrachten*⟩
end

Syntax van routine-definities

function ⟨*funcnaam*⟩ ⟨*parameterlijst*⟩ : ⟨*typenaam*⟩ ;
⟨*blok*⟩ ;

waarbij ⟨*blok*⟩ opdracht(en) *Result* := ⟨*formule*⟩ bevat.

procedure ⟨*procnaam*⟩ ⟨*parameterlijst*⟩ ;
⟨*blok*⟩ ;

Het ⟨*blok*⟩ heet ook *body* of *implementatie* van de routine

Voorbeeld functie-definitie (1)

function *Afstand* (**const** x, y: *Real*): *Real*;
↑ ↑ *formele* parameters

{ **pre**: *True*
ret: afstand van (x, y) tot oorsprong }

begin

Result := *sqrt* ($sqr(x) + sqr(y)$)

end; { *Afstand* }

Voorbeeld functie-definitie (2)

function *Max* (**const** a, b: *Integer*): *Integer*;

{ **pre**: *True*
ret: maximum van a en b }

begin

if $a > b$ **then begin**
Result := a
end
else begin
Result := b
end

end; { *Max* }

Voorbeeld procedure-definitie (1)

procedure *WachtOpEnter*;

{ **pre**: *True*
post: prompt geschreven, overgang op nieuwe regel gelezen }

begin

write (`'Tik op ENTER om door te gaan: '`)
; *readln*

end; { *WachtOpEnter* }

Voorbeeld procedure-definitie (2)

procedure $BepaalMax$ (**out** m: $Integer$; **const** a, b: $Integer$);

{ **pre**: $True$
post: m = maximum van a en b }

begin

if $a > b$ **then begin**
$m := a$
end
else begin
$m := b$
end

end; { $BepaalMax$ }

Voorbeelden van routine-aanroepen

var
d: $Real$; { ... }
h, i, j, k, m: $Integer$; { ... }

...
↓ ↓ *actuele* parameters of *argumenten*
$d := Afstand$ (3, h)
; $m := Max$ ($i{+}j$, $Max(k, 100)$)

; $WachtOpEnter$

; $BepaalMax$ (h, k, 100)
; $BepaalMax$ (m, $i{+}j$, h)

Semantiek van procedures: correctheid van implementatie

procedure p (**const** c: $T1$; **out** v: $T2$; **var** x: $T3$; a: $T4$);
{ **pre:** $U(c, v, x, a)$
post: $V(c, v, x, a)$ }
⟨*body*⟩

met U, V predikaten over toestandruimte uitgebreid met parameters

Als **preconditie** U vóór body geldt, dan geldt erna **postconditie** V:

{@ $U(c, v, x, a) \wedge x = \text{old } x \wedge a = A$ }
⟨*body*⟩
{@ $V(c, v, x, A)$ } ⟵ a heeft waarde als in begintoestand

Vergelijk met procedure-aanroep

Lokale variabelen (binnen routine)

procedure $Verwissel$ (**var** x, y: $Integer$);

{ **pre**: $True$
post: $x = \text{old } y \wedge y = \text{old } x$ }

var
h: $Integer$; ⟵ *lokale* variabele, beperkt tot $Verwissel$

begin {@ $y = \text{old } y \wedge x = \text{old } x$ }
$h := x$ {@ $y = \text{old } y \wedge h = \text{old } x$ }
; $x := y$ {@ $x = \text{old } y \wedge h = \text{old } x$ }
; $y := h$ {@ $x = \text{old } y \wedge y = \text{old } x$ }
end; { $Verwissel$ }

De annotatie kan het best *van achter naar voren* gelezen worden.

Werking van procedures: dubbele substitutie

procedure p (**const** c: $T1$; **out** v: $T2$; **var** x: $T3$; a: $T4$);
  var
    ℓ: $T5$;
  begin
    S
  end;

Aanroep $p\ (\ d, w, y, b\ )$ is te vervangen door:

begin { introduceer *verse* variabelen a, ℓ }
  $a := b$ ⟵ value parameter = geïnitialiseerde lokale variabele
; S' ⟵ verkregen uit S door c, v, x te vervangen door d, w, y
end { elimineer variabelen a, ℓ }

Dubbele substitutie bij aanroep $Verwissel(v, w)$

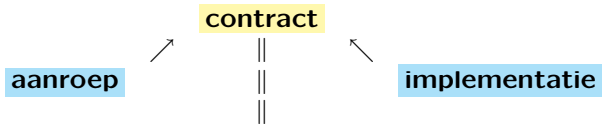
Aanroep $Verwissel\ (\ v, w\ )$ is te vervangen door

var
  ...
  h': $Integer$; { h' is een *verse* naam }

...

begin
  $h' := v$
; $v := w$ ⟵ body waarin x, y is vervangen door v, w
; $w := h'$
end

Specificatie speelt centrale rol



- Relateer aanroep aan contract en
  relateer implementatie aan contract.
- Relateer NOOIT aanroep en implementatie direct aan elkaar.
  Dan is er geen sprake van 'verdelen'.
  Dit is te complex en leidt tot fouten en dus niet tot 'heersen'.

Voorbeeld: werkt wel maar is dubbel fout

```
 1 procedure P ( var x: Integer );
 2   { pre:  x < 0
 3     post: x > - ( old x )
 4   }
 5   begin
 6     x := -x + 1   (* N.B. x := -x + 2 voldoet ook aan contract *)
 7   end; { P }
 8
 9 var
10   v: Integer;
11
12 begin
13   v := 0      (* preconditie geldt niet, maar P werkt toch wel *)
14 ; P ( v )
15   {@ v = 1 } (* volgt niet uit postconditie, maar P zorgt er wel voor *)
16 end.
```

Dobbelen (3): Aanhef en definities

```
1  program Dobbelen3;
2    { (c) 2001, Tom Verhoeff, Versie 3 }
3    { Lees 5 worpen en bepaal uitslag (werper van unieke maximum) }
4
5  const
6    NSpelers   =  5; { aantal spelers, 1 <= NSpelers }
7    MaxWaarde = 12; { maximale waarde van een worp, 1 <= MaxWaarde
8
9  type
10   Speler = 1 .. NSpelers;  { de verzameling spelers }
11   Waarde = 1 .. MaxWaarde; { de verzameling worpwaarden }
12   WorpLijst = array [ Speler ] of Waarde; { worpen van een ronde
```


Dobbelen (3): Worpen inlezen

```
1  procedure LeesIn ( out wl: WorpLijst );
2    { pre:  True
3      post: Worpen ingelezen in wl[1..NSpelers] }
4    var
5      s: Speler; { doorloopt wl }
6    begin
7      write ( 'Geef de ', NSpelers, ' worpen: ' )
8
9    ; for s := 1 to NSpelers do begin
10       read ( wl [ s ] )
11     end { for s }
12
13   ; readln
14   end; { LeesIn }
```


Dobbelen (3): Bepaal hoogste worp

```
1  procedure BepaalMaxWorp ( const wl: WorpLijst; out m: Waarde );
2    { pre:  True
3      post: m = maximum van wl[1..NSpelers] }
4    var
5      s: Speler; { doorloopt wl }
6    begin
7      m := wl [ 1 ]
8
9    ; for s := 2 to NSpelers do begin
10     if wl [ s ] > m then begin
11       m := wl [ s ]
12     end
13   end { for s }
14
15   end; { BepaalMaxWorp }
```


Dobbelen (3): Bepaal hoogste worp (alternatief)

```
1  function MaxWorp ( const wl: WorpLijst ): Waarde;
2    { pre: True
3      ret: maximum van wl[1..NSpelers] }
4    var
5      s: Speler; { doorloopt wl }
6    begin
7      Result := wl [ 1 ]
8
9    ; for s := 2 to NSpelers do begin
10     if wl [ s ] > Result then begin
11       Result := wl [ s ]
12     end
13   end { for s }
14
15   end; { MaxWorp }
```

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 = aantal keer dat w voorkomt in wl }
5    var
6      s: Speler; { doorloopt wl }
7    begin
8      a := 0
9
10   ; for s := 1 to NSpelers do begin
11         Inc ( a, ord ( wl[s] = w ) )
12     end { for s }
13
14   end; { BepaalAantal }
```


Dobbelen (3): Bepaal wie worp deed

```
1  procedure BepaalWerper ( const wl: WorpLijst; w: Waarde;
2                           out r: Speler );
3    { pre:  w komt voor in wl
4      post: wl [ r ] = w }
5    var
6      s: Speler; { doorloopt wl }
7    begin
8      r := 1
9
10   ; while wl [ r ] <> w do begin
11       r := r + 1
12     end { while }
13
14   end; { BepaalWerper }
```


Dobbelen (3): Toestandsruimte

```
1  var
2    worp: WorpLijst;
3      { worp[i] = worp van speler i }
4    max: Waarde;
5      { maximum van worp[1..NSpelers] }
6    telMax: Integer;
7     { aantal keer dat max voorkomt in worp[1..NSpelers] }
8    winnaar: Speler;
9     { eventuele winnaar }
```


Dobbelen (3): Toestandsveranderingen

```
1  begin
2    LeesIn ( worp )
3  ; BepaalMaxWorp ( worp, max )
4  ; BepaalAantal ( worp, max, telMax )
5  ; if telMax = 1 then begin
6      BepaalWerper ( worp, max, winnaar )
7    ; writeln ( 'Speler ', winnaar, ' wint' )
8    end
9    else begin
10     writeln ( 'Geen winnaar' )
11   end
12   { de uitslag is afgedrukt }
13 end.
```

Value parameters in postconditities

Conventie: Value parameter in postconditie refereert aan beginwaarde.

procedure *LaatSterretjesZien* (n: *Integer*);
{ **pre**: $0 \leq n$
post: n sterretjes zijn geschreven }

begin
$n := 0$ ⟵ zou niet kunnen met **const** n: *Integer*
end; { *LaatSterretjesZien* }

Deze implementatie voldoet i.h.a. niet aan contract, maar realiseert wel 'n sterretjes zijn geschreven'.

LaatSterretjesZien (3) moet 3 sterretjes schrijven.

Beginwaarde van value parameter benoemen in contract

Alternatieve contractvorm:

procedure *LaatSterretjesZien* (n: *Integer*);
{ **pre**: Zij $n = N$ en $0 \leq N$
post: N sterretjes zijn geschreven }

begin

{@ invariant: nog n sterretjes te schrijven }
while $n \neq 0$ **do begin**
write ('$*$')
; $n := n - 1$ ⟵ zou niet kunnen met **const** n: *Integer*
end

end; { *LaatSterretjesZien* }

Value parameters: *MultiAssign*

procedure *MultiAssign* (**var** x, y: *Integer*; e, f: *Integer*);

{ **pre**: Zij $e = E$ en $f = F$
post: $x = E$ en $y = F$ }

begin

$x := e$
; $y := f$

end; { *MultiAssign* }

N.B. Implementatie realiseert i.h.a. niet "$x = e \wedge y = f$"!

Bijvoorbeeld: *MultiAssign* (v, w, $w + 1$, $v + 1$)

Goed gebruik van *MultiAssign*

var
a, b: *Integer*; { . . . }

begin
. . .

{@ $a = A$ en $b = B$ }

MultiAssign (a, b, b, a)

{@ $a = B$ en $b = A$ }

Dubbele substitutie bij aanroep $MultiAssign(a, b, b, a)$

var
  ...
  e', f': *Integer*; { e', f' zijn *verse* namen }

...

begin
  $e' := b$
; $f' := a$

; $a := e'$
; $b := f'$
end

Vergelijk dit met aanroep $Verwissel(a, b)$.

Contractbreuk bij $MultiAssign$: Aliasing

var
  a: *Integer*;

begin

  {@ $a = A$ }

  $MultiAssign$ ($a, a, 0, 1$) ⟵ aliasing: NIET DOEN

  {@ ??? $a = 0$ en $a = 1$??? }

aliasing = verschillende namen voor hetzelfde object

In $MultiAssign$ verwijzen x en y beide naar a.

$MultiAssign$ met const parameters

procedure $MultiAssignConst$ (**var** x, y: *Integer*; **const** e, f: *Integer*);

{ **pre**: *True*
  post: $x = e$ en $y = f$ }

begin

  $x := e$
; $y := f$

end; { $MultiAssignConst$ }

Dubbele substitutie bij aanroep $MultiAssignConst(a, b, b, a)$

Aanroep $MultiAssignConst(a, b, b, a)$ is af te raden!

Deze aanroep is te vervangen door:

begin
  $a := b$
; $b := a$
end

N.B. Bij **const**-parameter wordt waarde niet gekopieerd (efficiënter).

Aliasing gooit roet in het eten

var
a, b: *Integer*;

begin

{@ $a = A$ en $b = B$ }

MultiAssignConst (a, b, b, a) ⟵ aliasing: NIET DOEN

{@ ??? $a = b$ en $b = a$??? }

In *MultiAssignConst* verwijzen x en f beide naar a.

In *MultiAssignConst* verwijzen y en e beide naar b.

Globale variabelen (buiten routine)

var
g, c: *Integer*; ⟵ globale variabelen t.o.v. p

procedure p; ⟵ gebruik van g, c "onzichtbaar" in contract
begin
. . .
$g := g + c$ ⟵ gebruik van globalen g, c in implementatie
. . .
end; { p }

begin

p ⟵ deze aanroep kan g veranderen

Globale variabelen: documenteren

var
g, c: *Integer*; ⟵ globale variabelen t.o.v. p

procedure p { **glob var** g; **glob const** c };
{ **pre**: Zij $c = C$ ↑
post: $g =$ old $g + C$ } ⟵ g, c zichtbaar in contract
begin
. . .
$g := g + c$
. . .
end; { p }

begin

p

Globale variabelen: vermijden via extra parameters

procedure p (**var** x: *Integer*; **const** y: *Integer*);
{ **pre**: Zij $y = Y$
post: $x =$ old $x + Y$ }
begin
. . .
$x := x + y$ ⟵ gebruik van parameters x, y in implementatie
. . .
end; { p }

var
g, c: *Integer*; ⟵ niet direct bruikbaar in body van p
vanwege de eis *'define before use'*
begin
p (g, c) ⟵ gebruik van g, c zichtbaar in aanroep

Richtlijnen bij aanroepen van routines

- Gebruik een routine volgens diens specificatie.
- Vermijd aliasing in routine aanroepen.

  Een *actuele* **out**/**var**-parameter mag niet in dezelfde aanroep nogmaals als actuele **out**-, **var**- of **const**-parameter vóórkomen:

  ↓ als dit een actuele **out**/**var**-parameter is

  $routine$ (. . . , v, . . . , v, . . .)

  ↑ dan moet dit een actuele value-parameter zijn

Aanwijzingen bij implementeren van routines

- Implementeer de specificatie, niet meer en niet minder.
- **Const** parameters kunnen in body *niet* gewijzigd worden.

  Value parameters mogen *wel* in body gewijzigd worden.

  Value parameter = geïnitialiseerde *lokale* variabele
- Introduceer geschikte lokale variabelen.
- Globale variabelen niet in body gebruiken, tenzij contract het vereist.

  Gebruik van globale *constantes* en *types* in body is wel OK.

  Definieer routines zoveel mogelijk *vóór* de globale variabelen.

Waarschuwing

Veel details rond routines zijn nog steeds niet aangeroerd

en zijn ook niet van belang bij Programmeren – Blok B.

Gebruik alleen mogelijkheden die behandeld zijn en die je begrijpt.