Programmeren – Blok A

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

College 3

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

Trilogie van (programmeer)talen

- **Syntaxis** (vormleer): *Hoe ziet 't eruit, hoe schrijf je 't?*
- **Semantiek** (betekenisleer): *Wat betekent 't, hoe werkt 't?*
- **Pragmatiek** (gebuiksleer): *Hoe gebruik je 't in de praktijk?*

  conventies, methodes, technieken

Trilogie-voorbeeld: Verkeerslichten

Syntax (verschijningsvormen):

auto voetganger trein bus

Semantiek (bevelen): Stop, Pas op, Ga, . . .

Pragmatiek (gewoontes): door rood rijden, . . .

Pascal: Structuur van programma

program ⟨*prognaam*⟩ ;
{ ⟨*auteur*⟩, ⟨*toelichting*⟩ }

⟨*lijst definities*⟩ { const, type }

var { toestandsruimte }
⟨*varnaam*⟩ : ⟨*type*⟩ ; { ⟨*toelichting*⟩ }
. . .

begin { toestandsveranderingen }
⟨*lijst opdrachten*⟩ { aaneengeregen met ; }
end.

Pascal: Opdrachten

toekenning: ⟨*var*⟩ := ⟨*formule*⟩

invoer: *readln* (⟨*var*⟩)

uitvoer: *writeln* (⟨*formule*⟩)

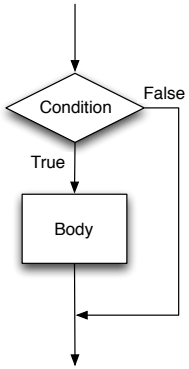
selectie: **if** ⟨*conditie*⟩ **then** ⟨*opdracht*⟩ **else** ⟨*opdracht*⟩

herhaling: **while** ⟨*conditie*⟩ **do** ⟨*opdracht*⟩ {0 of meer keer}

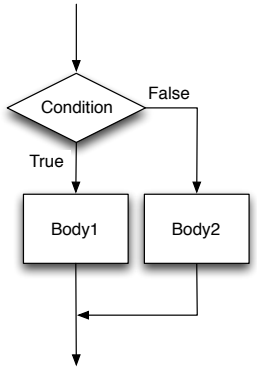
samenstelling: **begin** ⟨*lijst opdrachten*⟩ **end**

Operationele betekenis

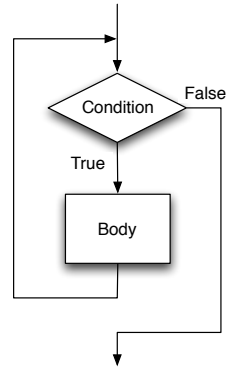
if Condition **then** Body



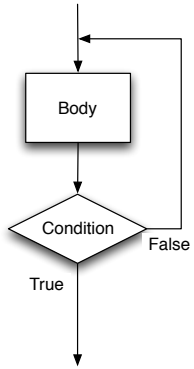
if Condition **then** Body1 **else** Body2



while Condition **do** Body



repeat Body **until** Condition



Meer Pascal

- Constante benoemen: **const** ⟨*constnaam*⟩ = ⟨*constexpr*⟩ ;
- Nieuw type construeren
  - Interval-type: ⟨*constexpr*⟩ .. ⟨*constexpr*⟩
- Type benoemen: **type** ⟨*typenaam*⟩ = ⟨*type*⟩ ;

Nog meer Pascal

- Array-type: **array** [⟨*indextype*⟩] **of** ⟨*type*⟩

  array-element: ⟨*arrayvarnaam*⟩ [⟨*indexformule*⟩]
- Vooraf bepaald aantal keer herhalen:

  for ⟨*varnaam*⟩ := ⟨*formule*⟩ **to** ⟨*formule*⟩ **do** ⟨*opdracht*⟩

  for ⟨*varnaam*⟩ := ⟨*formule*⟩ **downto** ⟨*formule*⟩ **do** ⟨*opdracht*⟩
- Eén of meer keer herhalen:

  repeat ⟨*lijst opdrachten*⟩ **until** ⟨*conditie*⟩

Nesten

Geneste haakjes: ((. . .) . . . (. . .))

Geneste lussen: **while** B **do begin** . . . ; **while** C **do** S ; . . . **end**

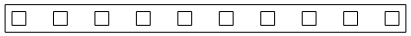
1-dimensionale arrays

Array-type: **array** [⟨*indextype*⟩] **of** ⟨*type*⟩

array-element: ⟨*arrayvarnaam*⟩ [⟨*indexformule*⟩]

var a: **array** [0 .. 9] **of** *Integer*;

a [0], a [1], . . . , a [9]



Kortschrift voor declaratie van 10 *Integer* variabelen $a[i]$

1-dimensionale arrays ingekleurd

var a: **array** [0 .. 9] **of** *Integer* ;

a [0] a [1] . . . a [9]

Inpakken versus uitpakken

Vergelijk met (wiskundige) functie f en beeld $f(x)$ bij origineel x:

Voor $f : D \to R$ en $x \in D$ is $f(x) \in R$

Toepassing van array bij VolleybalScore

```
1  type
2    Team   = 1 .. 2;
3    Score  = 0 .. MaxInt;
4
5  var
6    score1 : Score; { score van team 1 (uitvoer) }
7    score2 : Score; { score van team 2 (uitvoer) }
8    scoort : Team;  { team dat nu scoort (invoer) }
9  ...
10       if scoort = 1 then
11         score1 := score1 + 1
12       else {@ scoort = 2 }
13         score2 := score2 + 1
```

```
1  var
2    score: array [ Team ] of Score; { score[t] = score van team t (uitvoer) }
3  ...
4        score [ scoort ] := score [ scoort ] + 1
```

for-lus: vast aantal keer herhalen, interval doorlopen

for ⟨*varnaam*⟩ := ⟨*formule*⟩ **to** ⟨*formule*⟩ **do** ⟨*opdracht*⟩

for ⟨*varnaam*⟩ := ⟨*formule*⟩ **downto** ⟨*formule*⟩ **do** ⟨*opdracht*⟩

Voorbeelden:

```
1  writeln ( 1 ) ; writeln ( 2 ) ; writeln ( 3 )
2
3  for i := 1 to 3 do writeln ( i )
4
5  for i := n downto 1 do ... i ...
6
7  for c := 'a' to 'z' do ... c ...
```


for-lus: semantiek quiz

```
1    readln ( i )
2
3  ; for i := i + 1 to sqr ( i ) do writeln ( i )
4
5  ; writeln ( i )
```

Wat is de uitvoer bij invoer 0, 1, 2, 3?

Beter:

```
1    readln ( a )
2
3  ; for i := a + 1 to sqr ( a ) do  writeln ( i )
4
5  ; writeln ( sqr ( a ) )
```


for-lus: semantiek

for ⟨*varnaam*⟩ := ⟨*formule*⟩ **to** ⟨*formule*⟩ **do** ⟨*opdracht*⟩

for ⟨*varnaam*⟩ := ⟨*formule*⟩ **downto** ⟨*formule*⟩ **do** ⟨*opdracht*⟩

Boven- en ondergrens worden *vóóraf éénmalig* geëvalueerd.

Na afloop is de waarde van de stuurvariabele *ongedefinieerd*.

De herhaalde opdracht mag de stuurvariabele *niet* veranderen.

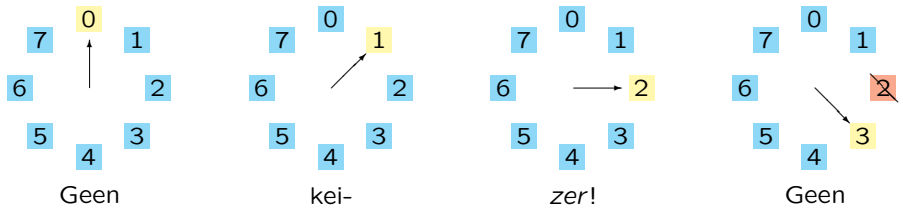
Verboden: **for** i := 1 **to** n **do begin** ... i := i + 1 ... **end**

Hoe een keizer te kiezen

Zet kandidaten 0 t/m $N-1$ in de kring.

Beginnend bij 0 valt elke 3e af.

Laatstoverblijvende wordt keizer.



Ontwerp voor KeizerKiezer

Houdt bij welke kandidaten in kring staan (*Boolean* per kandidaat).

1. Initieel alle kandidaten in kring.
2. Herhaaldelijk één kandidaat laten afvallen (aanwijzen en aftellen).
3. Overgebleven kandidaat wordt keizer.

Programma voor KeizerKiezer: Definities en declaraties

```
1  program KeizerKiezer;
2    { Kies keizer uit kandidaten 0 t/m N-1.
3      Beginnend bij kandidaat 0 valt elke 3e kandidaat af. }
4
5  const
6    N = 8; { aantal kandidaten }
7
8  type
9    Kandidaat = 0 .. N - 1; { de kandidaten }
10
11 var
12   inKring: array [ Kandidaat ] of Boolean;  { de kring met kandidaten }
13     { inKring[i] = "kandidaat i staat nog in de kring" }
14   rest: 1 .. N;  { aantal overgebleven kandidaten }
15     { rest = (# i: i in Kandidaat: inKring[i] ) }
16   aangewezen: Kandidaat;  { doorloopt de kandidaten }
17     { er geldt inKring [ aangewezen ] }
```


Programma voor KeizerKiezer: Initialisatie

```
1  begin
2
3    { 1. Initialiseer volle kring, wijs eerste kandidaat aan }
4
5  ; for aangewezen := 0 to N - 1 do begin
6      inKring [ aangewezen ] := True
7    end { for aangewezen }
8
9  ; rest := N        { alle kandidaten doen (nog) mee }
10 ; aangewezen := 0 { kandidaat 0 als eerste aangewezen }
```


Programma voor KeizerKiezer: Volgende aanwijzen

```
1      ...
2
3      {@ lettergreep g valt op aangewezen kandidaat }
4
5      repeat
6
7        aangewezen := ( aangewezen + 1 ) mod N
8
9      until inKring [ aangewezen ]
10
11     {@ lettergreep g+1 valt op aangewezen kandidaat }
12
13     ...
```

Programma voor KeizerKiezer: Herhaald afvallen

```
  { 2. Dun kring uit tot een kandidaat resteert, die keizer wordt }

  {@ Invariant: 1e lettergreep valt op aangewezen kandidaat }
; while rest <> 1 do begin
    { 2.1.  Wijs afvaller aan }
    repeat ... until inKring [ aangewezen ]  {@ 2e lettergreep ... }
  ; repeat ... until inKring [ aangewezen ]
    {@ 3e lettergreep valt op aangewezen kandidaat }

    { 2.2. Verwijder aangewezen kandidaat uit kring }
  ; inKring [ aangewezen ] := False
  ; rest := rest - 1

    { 2.3. Wijs volgende kandidaat in kring aan voor 1e lettergreep }
  ; repeat ... until inKring [ aangewezen ]  {@ 1e lettergreep aangewezen }
  end { while }
  {@ rest = 1 }
```


Programma voor KeizerKiezer: Herhaald afvallen

```
  { 3. Schrijf het antwoord }

; writeln ( 'Kandidaat ', aangewezen, ' wordt keizer' )

end.
```


Programma voor KeizerKiezer: Aantal kandidaten inlezen

```
program KeizerKiezer;
  { ... }

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

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

var
  N: 1 .. MaxN;  { aantal kandidaten (invoer) }
  inKring: array [ Kandidaat ] of Boolean; { wie er nog meedoen }
  rest: 1 .. MaxN;  { aantal overgebleven kandidaten }
  aangewezen: Kandidaat;  { doorloopt kring }

begin
  ...
  readln ( N )
```


Programma voor KeizerKiezer: Alternatief ontwerp

```
var
  inKring: array [ Kandidaat ] of Boolean; { wie er nog meedoen }

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

veranderen in

```
var
  opvolger: array [ Kandidaat ] of Kandidaat;
    { voor alle i: 0 <= i < NKandidaten:
      opvolger[i] = "opvolger van i in de kring" }

...
      aangewezen := opvolger [ aangewezen ]
```

Codeerstandaard

Hoe schrijf je een programma op?

Beperkingen zorgen voor

- leesbaarheid
- uniformiteit
- minder fouten

Onze codeerstandaard: 'doorsnede' van aantal codeerstandaarden

Practium week 3

- Letters turven
- Keizer kiezer met opvolger array
- Simulatie van dobbelspel

Letters Turven: aanhef en definities

```
 1 program LettersTurvenOrig;
 2   { Tom Verhoeff, TU/e, 8 september 2008 }
 3   { Lees 20 letters (uit 'a' t/m 'z') van een regel, zonder spaties ertussen.
 4     Bepaal het aantal voorkomens van elke letter.
 5     Schrijf een tabel met per letter van 'a' t/m 'z' een regel,
 6     waarop de letter en het aantal voorkomens ervan. }
 7
 8   { (c) 1981, Tom Verhoeff, TUE.  Version 1.0 }
 9
10 const
11   MinLetter = 'a'; { eerste te tellen letter }
12   MaxLetter = 'z'; { laatste te tellen letter }
13   Aantal    = 20;  { aantal letters in invoer }
14
15 type
16   Letter = MinLetter .. MaxLetter; { te tellen letters }
```


Letters Turven: variabelen en initialisatie

```
18 var
19   a: Letter; { om de te tellen letters te doorlopen of in te lezen }
20   i: 1 .. Aantal; { om de letters af te tellen bij het inlezen }
21   freq: array [ Letter ] of Cardinal; { frequentietelling van letters }
22     { freq [ a ] = aantal voorkomens van letter a }
23
24 begin
25   { 1. Initialiseer de tellingen op nul }
26
27   for a := MinLetter to MaxLetter do begin
28     freq [ a ] := 0
29   end { for a }
30
31   {@ freq [ a ] = 0, voor a in Letter }
```

Letters Turven: frequenties tellen en afdrukken

```
33    { 2. Lees de letters en tel hun voorkomens }
34
35  ; for i := 1 to Aantal do begin
36      read ( a )
37    ; freq [ a ] := freq [ a ] + 1
38    end { while }
39
40  ; readln
41    {@ freq [ a ] = aantal keer dat a gelezen is }
42
43    { 3. Schrijf de tabel }
44  ; writeln ( 'Ltr   #' )
45  ; writeln ( '---  --' )
46
47  ; for a := MinLetter to MaxLetter do begin
48      writeln ( ' ', a, ' ', '  ', freq [ a ] )
49    end { for a }
51    {@ freq is geschreven als tabel }
```


Groter programma 'aan de praat krijgen'

- In stapjes opbouwen.
- Telkens code nalezen en testen.

Letters Turven:

1. Eerst alleen stappen 1 (initialisatie) en 3 (afdrukken).
2. Dan pas stap 2 (tellen) tussenvoegen.

Dobbelsimulatie

1. In Dobbelronde, de invoer vervangen door *Random* gegenereerde waarden. Druk daarbij (tijdelijk) de worpwaarden af.
2. Een for-lus eromheen plaatsen, om een ingelezen aantal ronden te simuleren.
3. Variabele voor frequentietelling toevoegen, initialiseren, en na for-lus afdrukken.
4. In de for-lus de frequentie voor de winnaar bijwerken.
5. Het afdrukken van de worpwaarden verwijderen.

Huiswerkopgaven

Achterstand wegwerken

Zie FAQ op studeerwijzer.