Programmeren in Pascal: Voorbeelden

Tom Verhoeff
Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

Inhoudsopgave

1 Inleiding

We leggen de programmeertaal Pascal uit aan de hand van voorbeelden. Ieder voorbeeld is een volledig Pascal programma. Bij twijfel over een voorbeeld, raden we aan het op de computer uit te proberen. Tussendoor staan ook enkele oefeningen.

Niet alle details van Pascal worden hier uitgelegd. Raadpleeg daarvoor de documentatie of on-line help van het Pascal-systeem dat u gebruikt.

De Pascal programma's waar het hier om gaat hebben een zogenaamd console interface, in tegenstelling tot een grafisch user interface (GUI).

2 Tekst schrijven

2.1 Een eerste Pascal programma

Het volgende Pascal programma schrijft de tekst ‘`Ik schrijf dus ik ben!`’:

```
program Schrijf;
begin
  writeln ( 'Ik schrijf dus ik ben!' )
end.
```

Let er op dat alle symbolen worden ingetikt!

program Na het sleutelwoord **program** volgt de naam van het Pascal programma afgesloten door een puntkomma. In Pascal bestaat een naam (Eng.: identifier) uit een ⟨*naam*⟩ letter gevolgd door nul of meer letters of cijfers.

begin **end** De uit te voeren opdrachten van het programma zijn omsloten door de sleutelwoorden **begin** en **end**.

writeln De opdracht *writeln*, kort voor ‘write line’, schrijft naar standaard-uitvoer, veelal gekoppeld aan het scherm. De schrijfkop wordt erna aan het begin van de volgende regel gezet.

De te schrijven tekst (Eng.: string) is in het programma omsloten door apostrofs (enkele aanhalingstekens sluiten, niet te verwarren met aanhalingsteken openen). Als de plaatsing van spaties in zo’n tekst van belang is zullen we die ook wel met ‘␣’ aangeven: `'Ik␣schrijf␣dus␣ik␣ben!'`.

'

␣

Aan het einde van het programma staat een punt.

.

Wij geven in deze inleiding het programma als volgt weer:

program *Schrijf*;
begin
  writeln (`'Ik schrijf dus ik ben!'`)
end.

Voor de duidelijkheid zijn sleutelwoorden **vet**, namen *cursief* en letterlijke teksten in `schrijfmachinestijl` weergegeven.

2.2 Programma’s laten (ver)werken, fouten maken

De tekst van een Pascal programma wordt niet rechtstreeks uitgevoerd. Eerst leest het Pascal systeem de tekst en controleert of er geen taalfouten in zitten.

. . . run. . .

. . . foutmelingen. . .

2.3 Verdwijnend venster, opdrachten na elkaar

Bij het maken en uitvoeren van Pascal programma's uit deze inleiding zijn mogelijk nog wat details van belang die afhangen van het specifieke systeem waarmee gewerkt wordt. Zo is het bij Lazarus/Delphi nodig om aan te geven dat het gaat om een programma met console interface. Dit kan op verscheidene manieren, bijvoorbeeld door direct onder de eerste regel met **program** op te nemen:

```
{$APPTYPE CONSOLE}
```

Zulk soort toevoegingen zullen we hier niet telkens vermelden, maar ze kunnen wel nodig zijn om de voorbeelden uit te proberen. Voor verdere details verwijzen we naar aparte documentatie, met uitzondering van het volgende.

Bij uitvoering van zulke programma's in Lazarus, zal de tekst in een apart 'console' venster getoond worden en meteen 'verdwijnen'. Voortijdig sluiten kun je voorkomen door beëindiging van het programma uit te stellen tot de 'enter'-toets is aangeslagen:

```
program SchrijfEnWacht;
begin
  writeln('Ik schrijf dus ik ben!')
;
  write('Tik enter-toets: ')
;
  readln
end.
```

;

Een puntkomma tussen twee opdrachten betekent dat ze na elkaar worden uitgevoerd. We zullen de puntkomma's meestal niet op een aparte regel zetten:

```
program SchrijfEnWacht;
begin
  writeln('Ik schrijf dus ik ben!')
; write('Tik enter-toets: ')
; readln
end.
```

write

readln

De opdracht *write* schrijft ook naar standaard-uitvoer, maar laat de schrijfkop aan het einde van de regel staan. De opdracht *readln*, kort voor 'read line', leest van standaard-invoer —veelal gekoppeld aan het toetsenbord— tot en met een overgang naar een nieuwe regel.

Die laatste twee opdrachten zijn vaak handig om in je programma op te nemen, maar wij zullen ze hier niet elke keer toevoegen. Het tonen van de tekst 'Tik enter-toets: ' is niet nodig, maar wel 'gebruikersvriendelijk'.

2.4 Apostrof in tekst opnemen

''

Eén apostrof opnemen in een te schrijven tekst kan door *twee* apostrofs direct achterelkaar te zetten, niet te verwarren met het dubbele aanhalingsteken (").

```
program SchrijfProgramma;
begin
  writeln('program Schrijf;')
; writeln('begin')
; writeln('  writeln ( ''Ik schrijf dus ik ben!'' )')
; writeln('end.')
end.
```

Oefening. Pas dit toe in het programma *SchrijfEnWacht* om het woord 'enter' tussen aanhalingstekens (openen en sluiten) te zetten.

3 Berekeningen

3.1 Resultaat van berekening schrijven

Het resultaat van eenvoudige berekeningen kan direct worden getoond:

```
program Berekening;
begin
  writeln ( 12345679 * 63 )
end.
```

Een berekening wordt niet omsloten door apostrofs.

,

De opdrachten *write* en *writeln* kunnen meerdere dingen achterelkaar afdrukken door ze te scheiden met komma's:

```
program Berekening2;
begin
  writeln ('12345679 * 63 = ', 12345679 * 63 )
end.
```

$+ - */$

Er zijn vele operatoren om berekeningen mee uit te drukken, waaronder '$+$' (optellen), '$-$' (negatie, aftrekken), '$*$' (vermenigvuldigen) en '$/$' (delen). Vermenigvuldiging wordt altijd aangeduid met '$*$', en mag niet weggelaten worden. Operatoren hebben een zeker bindkracht (prioriteit), die doorbroken kan worden met haakjes '$(\ldots)$':

$(\ldots)$

```
program Berekening3;
begin
  writeln ( 8 - 1 * 8 + 1 )
; writeln ( (8 - 1) * (8 + 1) )
end.
```

Oefening. Doe de berekening $355/113$. Welke waarde wordt hiermee benaderd?

3.2 Resultaat opslaan in variabele

Het resultaat van een berekening kan worden opgeslagen in een variabele:

```
program Variabele;
var
  v: Integer;   { declaratie van variabele v }
begin
  v := 12345679 * 63   { variabele v krijgt een waarde }
; writeln ( v )   { waarde van variabele v wordt geschreven }
end.
```

{...} Tekst tussen accolades is commentaar en wordt door de computer genegeerd.

var Een variabele moet vóór gebruik worden gedeclareerd in een **var** clausule, waarin de naam en het type van de variabele worden vastgelegd:

⟨type⟩

```
var ⟨naam⟩: ⟨type⟩;
```

Via de naam wordt in het programma aan de variabele en bijbehorende waarde gerefereerd. Het type legt vast welke waardes de variabele kan aannemen. Zo

Integer staat het type *Integer* voor geheeltallige waardes, die in Object Pascal van -2^{31} t/m $2^{31} - 1$ reiken.

Op ieder moment tijdens de uitvoering van het programma heeft de variabele een waarde. In een toekenningsopdracht van de vorm

:=

```
⟨variabele⟩ := ⟨formule⟩
```

krijgt de variabele een ('nieuwe') waarde. De formule kan de variabele zelf bevatten, waarmee dan vewezen wordt naar de 'oude' waarde:

```
program Variabele2;
var
  v: Integer;
begin
  v := 12345679   { variabele krijgt een beginwaarde }
; writeln ( v )
; v := v * 63   { nieuwe waarde in termen van oude waarde }
; writeln ( v )
end.
```

N.B. Een variabele heeft in Pascal geen vóórgedefinieerde beginwaarde. De beginwaarde moet altijd expliciet in het programma worden toegekend. Dit staat ook bekend als het *initialiseren* van de variabele.

3.3 Meer variabelen

In de **var**-clausule kunnen vescheidene variabelen gedeclareerd worden. Variabelen van hetzelfde type mogen zelfs achterelkaar gedeclareerd worden:

```
program MeerVariabelen;
var
  a, b: Integer;   { teller en noemer }
  q: Real;   { quotient }
begin
  a := 2
; b := a
; a := a − 1   { b is onveranderd }
; q := a/b
; writeln ( a, ', ␣', b, ', ␣', q )
end.
```

Real **N.B.** Het resultaat van deling met / is altijd van het type *Real*, dat bestaat uit zogenaamde floating-point waardes; ook als de deling opgaat.

Een ingewikkelder voorbeeld om sommen voor de lagere school te genereren:

```
program MeerVariabelen2;
var
  a, b: Integer;   { gegevens, 0 < a < b }
  vkw, v: Integer;   { tussenresultaten }
begin
  a := 21 + Random ( 30 )   { 21 ≤ a ≤ 50 }
; b := 21 + Random ( 30 ) + a   { a + 21 ≤ b ≤ a + 50 }
; writeln ( '(', b, '^2 - ', a, '^2) / (', b, '-', a, ') = ?' )
; write ( 'Tik <return> om het antwoord te zien: ' )
; vkw := sqr ( b ) − sqr ( a )   { verschil van de kwadraten }
; v := b − a   { verschil }
; writeln ( 'quotient zonder rest: ', vkw div v )
; writeln ( 'rest na deling: ', vkw mod v )
end.
```

Random De standaard routine *Random* genereert pseudo-willekeurige getallen. Als n positief geheel is, dan is *Random* (n) ook geheel met

$$0 \leq \textit{Random}\ (\ n\ ) < n$$

Random is niet een pure functie, omdat bij herhaalde aanroep met hetzelfde argument de resultaten in het algemeen zullen verschillen.

sqr

De standaardfunctie *sqr* kwadrateert:

$$\mathit{sqr}\ (\ a\ )\ \ =\ \ a^2$$

Het resultaat heeft hetzelfde type als het argument.

div

mod

Voor gehele a en b met $b \neq 0$, geeft de uitdrukking $a\ \mathbf{div}\ b$ het gehele quotiënt zonder rest bij deling van a door b, terwijl $a\ \mathbf{mod}\ b$ de rest na deling van a door b geeft. De operatoren $a\ \mathbf{div}\ b$ en $a\ \mathbf{mod}\ b$ zijn alleen gedefinieerd op geheeltallige types.

Voor gehele getallen a en b met $0 \leq a$ en $0 < b$ geldt:

$$a\ \mathbf{div}\ b = q\ \wedge\ a\ \mathbf{mod}\ b = r\ \ \Leftrightarrow\ \ a = b * q + r\ \wedge\ 0 \leq r < b$$

Merk op dat in bovenstaand programma *MeerVariabelen2* de rest na deling altijd 0 is, en het quotiënt $a + b$, omdat *vkw* $= b^2 - a^2 = (b + a)(b - a)$.

Oefening. Wat 'doet' het volgende programma?

```
program Mysterie;
var
  n: Integer;   { invoer }
begin
  write ( 'Geef een natuurlijk getal: ' )
; readln ( n )
; writeln ( n mod 10, n div 10 )
end.
```

4 Waardes inlezen

Waardes kunnen worden ingelezen en toegekend aan variabelen:

```
program Formule;
  { Bereken het product van ingelezen gehele getallen a, b. }
var
  a, b: Integer;   { invoervariabelen }
begin
  write ( 'Geef a en b: ' )
; readln ( a, b )
; writeln ( a, ' x ', b, ' = ', a * b )
end.
```

De opdracht *readln* leest waardes in van standaard-invoer en kent ze toe aan de opgegeven variabelen. **N.B.** Bij invoer van getallen dienen deze gescheiden te worden door één of meer spaties (of Tabs of Enters, maar geen komma's e.d.).

Een voorbeeld met reële getallen:

```
program Cirkel;
  { Bereken omtrek en oppervlakte van een cirkel met ingelezen straal r. }
var
  r: Real;   { cirkelstraal (invoer) }
  h: Real;   { hulpvariabele }
begin
  write ( 'Geef de cirkelstraal: ' )
; readln ( r )
; h := r * 355/113
; writeln ( 'De omtrek is ongeveer ', 2 * h )
; writeln ( 'De oppervlakte is ongeveer ', h * r )
end.
```

.

N.B. Bij invoer en uitvoer van reële getallen wordt een punt gebruikt om het gehele deel van het fractionele deel te scheiden, zoals in '3.14159'. Ook is zogenaamde wetenschappelijke notatie met drijvende punt (Eng.: floating-point) mo-

e

gelijk: achtervoeging van de letter 'e' en een geheel getal n (eventueel negatief) betekent daarbij 'maal 10^n'. Bijv. $17.5\text{e}{-2}$ staat voor 0.175 en $1\text{e}9$ staat voor 10^9. Zie ook hieronder bij 'Uitvoer formatteren'.

Oefening. Evalueer de formules $(a+b)/2$ en $\sqrt{a^2+b^2}$ voor ingelezen reële a

sqrt

en b. De standaardfunctie *sqrt* trekt de vierkantswortel: $sqrt(x) \approx \sqrt{x}$ voor $x \geq 0$.

5 Constanten een naam geven

const Constanten kun je een naam geven in een **const** clausule:

```
program Product;
const
  Special = 12345679;   { speciaal getal, cijfer 8 overgeslagen }
var
  a: Integer;   { invoervariabele }
  b: Integer;   { hulpvariabele }
begin
  write ('Geef een getal in [ 1 .. 9 ]: ')
; readln ( a )
; b := a * 9   { magische factor }
; write ( Special, '  x ', b, '  = ', Special * b )
end.
```

Het kan ook met reële getallen:

```
program Bol;
  { Bereken de oppervlakte van een bol met ingelezen straal r. }
const
  Pi = 3.14159;   { Pi ≈ π }
var
  r: real;   { bolstraal (invoer) }
begin
  write ('Geef de bolstraal: ')
; readln ( r )
; writeln ('De oppervlakte is ', 4 * Pi * sqr ( r ) )
end.
```

Oefening. Toon ook de bolinhoud ($\frac{4}{3}\pi r^3$). Gebruik een hulpvariabele voor $4\pi r^2$.

6 Uitvoer formatteren

Int64

Het volgende programma toont een net uitgelijnde tabel met tweede, derde en vierde machten van de invoer. In Object Pascal reikt het type *Int64* van -2^{63} t/m $2^{63}-1$.

```
program Machten;
  { Bereken tweede, derde en vierde macht voor ingelezen n. }
var
  n: Int64;   { invoervariabele }
  n2: Int64;   { om kwadraat van n op te slaan }
begin
  write ('Geef n in [ 0 .. 55108 ]: ')
; readln ( n )
; n2 := sqr ( n )
; writeln ('n^2 = ', n2 : 20 )
; writeln ('n^3 = ', n * n2 : 20 )
; writeln ('n^4 = ', sqr ( n2 ) : 20 )
end.
```

$a:b$

De betekenis van $a:b$ in een *write* opdracht is dat de waarde van a met ten minste b tekens wordt getoond. De minimale veldbreedte b is een expressie met positieve gehele waarde. Als de gewone weergave van a minder dan b tekens vergt, dan wordt deze links aangevuld met spaties om precies breedte b te leveren (en anders wordt a gewoon weergegeven, mogelijk met meer dan b tekens).

Reële getallen worden standaard in floating-point notatie uitgevoerd. Fixed-point notatie wordt verkregen door als volgt een tweede formatteerparameter c op te geven: $a:b:c$. Hierin is c een expressie met niet-negatieve gehele waarde, die het aantal decimalen aangeeft waarop wordt afgerond.

$a{:}b{:}c$

```
program eMacht;
  { Bereken e^x op n decimalen voor ingelezen x, n. }
var
  x: Real;   { argument voor exp (invoer)}
  n: Integer;   { aantal decimalen (invoer) }
begin
  write ('Geef x en n (n>=0): ')
; readln ( x, n )
; writeln ('e^x = ', exp ( x ) : 1 : n )
end.
```

exp

De standaardfunctie *exp* levert e-machten. De uitvoer wordt hier getoond in n decimalen. Bij $n = 0$ wordt er ook geen punt geschreven.

Oefening. Toon de waarde van $1 + x + \frac{1}{2}x^2$ in n decimalen onder die van e^x. Vergelijk de waarden voor enkele x met $-1 \leq x \leq 1$.

7 Opdrachten selectief uitvoeren

7.1 Eén opdracht conditioneel uitvoeren

if then Eén opdracht conditioneel uitvoeren kan met **if**-**then**:

```
program Absoluut;
  { Bereken de absolute waarde van x voor ingelezen x. }
var
  x: Real;   { invoervariabele }
begin
  write ( 'Geef x: ' )
; readln ( x )
; if x < 0.0 then
    x := -x
; writeln ( 'De absolute waarde van x is ', x )
end.
```

abs De standaardfunctie *abs* kan ook gebruikt worden om de absolute waarde te bepalen.

7.2 Alternatieven onderscheiden

else Alternatieven onderscheiden gaat met **if**-**then**-**else**:

```
program Machtsverheffen;
  { Bereken x^y voor ingelezen x, y. }
var x, y: Real;   { invoervariabelen }
begin
  write ( 'Geef x en y: ' )
; readln ( x, y )
; write ( 'x^y is ' )
; if x ≤ 0.0 then
    writeln ( 'mogelijk niet in R' )
  else   { x > 0.0 }
    writeln ( exp ( y * ln(x) ) : 1 : 5 )
end.
```

≤ ≥ ≠ De relatie $\leq$ wordt ingetikt als <=. Analoog wordt $\geq$ ingetikt als >=, en $\neq$ als <>. Let er op dat de alternatieve opdrachten worden ingesprongen.

N.B. Pascal kent geen operator voor machtsverheffen. De standaardfunctie *ln* retourneert de natuurlijke logaritme (uit positieve getallen).

ln

Oefeningen. Toon van twee invoergetallen de grootste.

Bepaal voor gehele invoergetallen a, b met $b > 0$ of b een deler is van a. Denk aan de operator **mod** die de rest na deling oplevert.

7.3 Samengestelde opdracht

begin
end

Na **then** en na **else** volgt één opdracht. Een lijst opdrachten met puntkomma's ertussen kan samengesteld worden tot één opdracht door er **begin** en **end** omheen te zetten:

```
program StelLinVgl;
  { Los stelsel van 2 lineaire vergelijkingen met 2 onbekenden op. }

var
  a, b, c, d, e, f: Integer;   { coëfficiënten (invoer) }
  det: Integer;   { determinant }
  x, y: Real;   { oplossing }

begin
  write ('Geef a, b en c: ')
; readln (a, b, c)
; write ('Geef d, e en f: ')
; readln (d, e, f)
; writeln (Het lineaire stelsel)
; writeln ('  ', a, ' x + ', b, ' y = ', c)
; writeln ('  ', d, ' x + ', e, ' y = ', f)
; det := a * e - b * d
; if det = 0 then
    writeln ('vergt nadere analyse')
  else begin   { det ≠ 0 }
    x := (c * e - b * f)/det
  ; y := (a * f - c * d)/det
  ; writeln ('heeft oplossing')
  ; writeln ('  x = ', x : 10 : 5)
  ; writeln ('  y = ', y : 10 : 5)
  end   { else det ≠ 0 }
end.
```

7.4 Meer dan twee gevallen: geneste selectie

Meer dan twee gevallen onderscheiden kan door "geneste" selectie:

```
program abcFormule;
  { Los kwadratische vergelijking op met ingelezen coëfficiënten a, b, c. }

var
  a, b, c: Real;   { coëfficiënten (invoer) }
  discr: Real;   { discriminant }
  x1, x2: Real;   { oplossingen }

begin
  write ( 'Geef a, b en c: ' )
; readln ( a, b, c )
; writeln ( a, ' x^2 + ', b, 'x + ', c, ' = 0' )
; if a = 0.0 then begin
    if b = 0.0 then begin
      if c = 0.0 then   { a = 0.0 en b = 0.0 en c = 0.0 }
        writeln ( 'voor alle x' )
      else   { a = 0.0 en b = 0.0 en c ≠ 0.0 }
        writeln ( 'voor geen enkele x' )
    end   { if b = 0.0 then }
    else   { a = 0.0 en b ≠ 0.0 }
      writeln ( 'voor x = ', -c/b )
  end   { if a = 0.0 then }
  else begin   { a ≠ 0.0 }
    discr := sqr ( b ) - 4 * a * c
  ; if discr < 0.0 then
      writeln ( 'voor geen enkele x in R' )
    else begin   { a ≠ 0.0 en discr ≥ 0.0 }
      x1 := ( -b - sqrt ( discr ) ) / (2 * a)
    ; x2 := ( -b + sqrt ( discr ) ) / (2 * a)
    ; writeln ( 'voor x = ', x1 )
    ; writeln ( '   en x = ', x2 )
    end   { else discr < 0.0 }
  end   { else a ≠ 0.0 }
end.
```

The **begin**-**end** paren bij **if** $a = 0.0$ **then** en bij **if** $b = 0.0$ **then** zijn strictgenomen niet nodig, maar wel aan te bevelen. Merk op dat de inspringdiepte varieert met de nestingsdiepte.

Overigens is het beter om *x2* te berekenen als $-2 * c \ / \ (\ b + \mathit{sqrt}\ (\ \mathit{discr}\ )\ )$, met name als $4ac$ klein is t.o.v. b^2.

Oefening. Bepaal van drie ingelezen getallen de middelste.

7.5 Meer dan twee gevallen: geschakelde selectie

Bij geneste selectie wordt er steeds verder ingesprongen. Soms wil je een aantal gelijkwaardige gevallen onderscheiden. Dit kan dan beter door het op te vatten als een geschakelde selectie:

```
program Classificatie;
  { Bepaal teken van ingelezen getal. }
var
  n: Integer;   { invoervariabele }
begin
  write ( 'Geef n: ' )
; readln ( n )
; write ( 'n is ' )
; if n < 0 then
    writeln ( 'negatief' )
  else if n = 0 then
    writeln ( 'nul' )
  else   { if n > 0 then }
    writeln ( 'positief' )
end.
```

Merk op dat we bij zo'n meerwegkeuze na de **else** *niet* verder inspringen:

```
; if n < 0 then
    writeln ( 'negatief' )
  else
    if n = 0 then
      writeln ( 'nul' )
    else   { if n > 0 then }
      writeln ( 'positief' )
```

Bij meer gevallen 'loopt' de tekst anders steeds verder weg naar rechts. Dit is overigens anders dan het gevalsonderscheid in het vorige programa *abcFormle*.

Je moet wel oppassen dat de volgende twee programmafragmenten in het algemeen niet equivalent hoeven te zijn, al lijken ze sterk op elkaar:

if A **then** S **else if** B **then** T **else** U	**if** B **then** T **else if** A **then** S **else** U

In het linkergeval wordt T uitgevoerd onder de conditie $\neg A \ \wedge \ B$ en in het rechtergeval onder de conditie B. Beschouw bijvoorbeeld:

```
program Classificatie2;
  { Bepaal aantal cijfers van ingelezen natuurlijk getal < 1000. }
var
  n: Integer;   { invoervariabele }
begin
  write ('Geef n in [ 0 .. 999 ]: ')
; readln ( n )
; write ('n heeft ')
; if n < 10 then   { 0 ≤ n < 10 }
    writeln ('een cijfer')
  else if n < 100 then   { 10 ≤ n < 100 }
    writeln ('twee cijfers')
  else   { 100 ≤ n < 1000 }
    writeln ('drie cijfers')
end.
```

Het volgende is *niet* goed:

```
; if n < 100 then   { 10 ≤ n < 100 }
    writeln ('twee cijfers')
  else if n < 10 then   { 0 ≤ n < 10 }
    writeln ('een cijfer')
  else   { 100 ≤ n < 1000 }
    writeln ('drie cijfers')
```

Door het commentaar achter **then** valt dat meteen op. *Wel* hetzelfde zijn:

```
; if 0 ≤ n < 10 then
    writeln ('een cijfer')
  else if 10 ≤ n < 100 then
    writeln ('twee cijfers')
  else   { 100 ≤ n < 1000 }
    writeln ('drie cijfers')
```

en

```
; if 10 ≤ n < 100 then
    writeln ('twee cijfers')
  else if 0 ≤ n < 10 then
    writeln ('een cijfer')
  else   { 100 ≤ n < 1000 }
    writeln ('drie cijfers')
```

Maar het tweede oogt minder systematisch.

7.6 Boolse uitdrukkingen en variabelen

not
and
or

Ingewikkeldere condities kunnen worden opgebouwd met de Boolse operatoren **not**, **and**, **or** en haakjes.

```
program VolleybalStand;
  { Beoordeel een ingelezen volleybalstand a, b. }

var
  a, b: Integer;   { stand (invoer) }

begin
  write ( 'Geef stand a, b >= 0: ' )
; readln ( a, b )
; if ( (a < 25) and (b < 25) ) or ( abs(a - b) < 2 ) then
    writeln ( 'Doorspelen' )
  else begin   { klaar }
    if a < b then
      writeln ( 'Team 1 heeft gewonnen' )
    else   { a > b }
      writeln ( 'Team 2 heeft gewonnen' )
  end   { else klaar }
end.
```

N.B. De haakjes in de conditie zijn nodig, want $a < 25$ **and** $b < 25$ wordt geïnterpreteerd als $a < (25$ **and** $b) < 25$, hetgeen onzin is.

Boolean

False
True

Condities hebben het type *Boolean*, dat bestaat uit de twee Boolse waardes *False* en *True*, en kunnen opgeslagen worden in variabelen van het type *Boolean*. Het volgende programma heeft dezelfde functionaliteit als *VolleybalStand*, maar gebruikt een *Boolean* variabele:

```
program VolleybalStand2;
  { Beoordeel een ingelezen volleybalstand a, b. }

var
  a, b: Integer;   { stand (invoer) }
  klaar: Boolean;   { berekende eindconditie }

begin
  write ( 'Geef stand a, b >= 0: ' )
; readln ( a, b )
```

```
; klaar := ( (a ≥ 25) or (b ≥ 25) ) and (abs(a − b) ≥ 2)
; if not klaar then
    writeln ( 'Doorspelen' )
  else begin   { klaar, dus a ≠ b }
    if a < b then
      writeln ( 'Team 1 heeft gewonnen' )
    else   { a > b }
      writeln ( 'Team 2 heeft gewonnen' )
  end   { else klaar }
end.
```

7.7 Ambiguïteit bij ‘if’ zonder ‘else’

Wat betekent: **if** A **then if** B **then** T **else** U?

if A **then**
  if B **then** T
else U
	of	**if** A **then**
		  if B **then** S
		  else U

De programmeertaal Pascal houdt het op de rechtse interpretatie: **else** wordt bij de dichtsbijzijnde voorafgaande **if-then** getrokken. Het inspringen in de linkerversie vermijdt dat niet en is daarom misleidend. Gebruik in zo’n geval altijd een **begin-end** paar:

if A **then begin**
  if B **then** T
end
else $\{$ not A $\}$ U
	of	**if** A **then begin**
		  if B **then** S
		  else $\{$ not B $\}$ U
		end

ook al is het rechts strictgenomen niet nodig.

Indien **else** even vaak voorkomt als **if**, dan is er geen twijfel:

if A **then**
  if B **then** T
  else $\{$ not B $\}$ U
else $\{$ not A $\}$ V

8 Opdrachten herhalen

8.1 Bepaald aantal keer herhalen

for
to
do

Een vooraf bepaald aantal keer iets herhalen kan met de **for**-opdracht:

```
program SterrenOmlijsting;
  { Schrijf een kader van sterretjes om een m × n rechthoek van spaties. }

var
  m, n: Integer;   { invoervariabelen }
  i: Integer;   { om sterretjes af te tellen }

begin
  write ( 'Geef m en n tenminste 0: ' )
; readln ( m, n )
; for i := 1 to n + 2 do write ( '*' )
; writeln
; for i := 1 to m do write ( '*', '*' : n+1 )
; writeln
; for i := 1 to n + 2 do write ( '*' )
; writeln
end.
```

Merk op dat de n spaties op een regel worden afgedrukt door in de tweede **for**-opdracht handig gebruik te maken van uitvoerformattering:

```
write ( '*' : n+1 )
```

drukt n spaties voor het sterretje.

N.B.

- Na afloop van de opdracht **for** $i := \ldots$ is de waarde van i ongedefinieerd.
- Het is *niet* gestaan om in de herhaalde opdracht de stuurvariabele i van waarde te veranderen: ($i := \ldots$, *readln*(i), etc.
- De uitdrukkingen voor onder- en bovengrens, direct vóór en na **to**, worden eenmalig bij aanvang geëvalueerd.
- Als de ondergrens groter is dan de bovengrens wordt de herhaalde opdracht nul keer (dus niet) uitgevoerd.

8.2 Meer dan één opdracht herhalen

Bij herhaling van meer dan één opdracht is weer **begin**-**end** nodig:

```
program RijSommeren;
  { Sommeer een ingelezen rij van N gehele getallen. }

const
  N = 5;   { aantal te sommeren getallen }

var
  a: Integer;   { invoervariabele }
  s: Integer;   { som van gelezen getallen }
  i: Integer;   { om getallen af te tellen }

begin
  s := 0
; write ('Geef ', N : 1, ' getallen: ')

; for i := 1 to N do begin
    read ( a )
  ; s := s + a
  end   { for i }

; readln
; writeln ('De som is ', s )
end.
```

readln

De invoergetallen staan op één regel, vandaar dat *read* (a) is gebruikt i.p.v. *readln* (a). De opdracht *readln* leest tot en met de eerstvolgende overgang op een nieuwe regel. In dit geval leest *readln* dus alleen de ingetikte 'return' na de N getallen (mits de gebruiker doet wat gevraagd wordt).

Oefening. Toon het maximum van de invoerrij.

8.3 Terugtellen

downto Achteruit tellen kan met **downto** i.p.v. **to**:

```
program Aftellen;
   Telt af voor lancering.

var
  i: Integer; { afteller }

begin

  for i := 10 downto 0 do write ( i : 1, ', ' )

; writeln ( 'Lift off' )
end.
```

Het is overigens beter om de herhaalde opdracht(en) altijd tussen **begin**-**end** te plaatsen:

```
  for i := 10 downto 0 do begin
    write ( i : 1, ', ' )
  end  for i
```

8.4 Geneste herhaling

Een herhaling binnen een herhaling heet geneste herhaling:

```
program SterrenDriehoek;
  { Schrijf een gevulde driehoek van sterretjes. }

var
  n: Integer;   { invoervariabele }
  i, j: Integer;   { om rijen en kolommen te doorlopen }

begin
  write ( 'Geef n: ' )
; readln ( n )

; for i := 1 to n do begin

    for j := 1 to i do begin
      write ( '*' )
    end { for j }

  ; writeln
  end { for i }

end.
```

In de praktijk van het programmeren blijkt telkens weer dat herhaalde opdrachten meer aanleiding geven tot fouten dan andere opdrachten. Om herhaalde opdrachten beter op te laten vallen worden ze vaak omgeven door lege regels.

8.5 Onbepaald aantal keer herhalen: while

while do

Indien niet vooraf bekend is hoe vaak herhaald moet worden, dan kan dat met de **while**-opdracht:

```
program GGDtraag;
  { Bepaal de grootste gemene deler van twee ingelezen getallen. }

var
  a, b: Integer; { invoervariabelen }

begin
  write ('Geef a en b, beide groter 0: ')
; readln (a, b)
; write ('ggd (', a : 1, ', ', b : 1,') = ')

; while a ≠ b do
    if a > b then a := a − b
    else { a < b } b := b − a

; writeln (a : 1)
end.
```

De herhalingsconditie vermeld na **while** wordt in iedere cykel gecontroleerd vóór uitvoering van de opdracht na **do**. Deze opdracht wordt slechts uitgevoerd als de conditie geldt, en anders wordt de herhaling direct afgebroken. De opdracht wordt dus nul of meer keer uitgevoerd.

N.B. Bij herhaling van meer dan één opdracht is weer **begin**-**end** nodig. Ook de **while**-opdracht in het voorgaande voorbeeld kan beter omgeven worden door **begin**-**end** om explicieter te laten blijken wat precies herhaald wordt:

```
; while a ≠ b do begin
    if a > b then a := a − b
    else { a < b } b := b − a
  end { while }
```

8.6 Onbepaald aantal keer herhalen: repeat

repeat

Als er ten minste één keer herhaald dient te worden, dan kan dat ook met de **repeat**-opdracht. De herhalingsconditie vermeld na **until** wordt bij iedere cykel gecontroleerd na uitvoering van de opdrachten ervoor. De herhaling wordt afgebroken zodra de conditie geldt.

```
program GGDsneller;
  { Bepaal de grootste gemene deler van twee ingelezen getallen. }

var
  a, b: Integer; { invoervariabelen }

begin
  write ('Geef a en b, beide groter 0: ')
; readln (a, b)
; write ('ggd␣(␣', a : 1, ',␣', b : 1,'␣)␣=␣' )

; repeat
    if a > b then a := a mod b
    else { a ≤ b } b := b mod a
  until (a = 0) or (b = 0)

; writeln ( a + b : 1 )
end.
```

8.7 Interactief bepaald aantal keer herhalen

Een voorbeeld met interactieve onderbreking van de herhaling:

```
program ComplexeProducten;
  { Bepaal product van ingelezen complexe getallen, interactief herhaald. }

var
  a, b, p, q: Real; { invoergetallen }
  c: Char; { antwoord (invoer) }
  im: Real; { imaginair deel }

begin

  repeat
    { Vraag twee complexe getallen }
    write ('Geef a, b, p en q: ')
  ; readln (a, b, p, q)

    { Toon het complexe product }
  ; write ('(a + b*i)(p + q*i) = ')
  ; write (a * p - b * q : 1 : 2)
  ; im := a * q + b * p
  ; if im >= 0 then write ('+')
  ; writeln (im : 1 : 2, '*i')

    { Vraag om herhaling }
  ; write ('Nog eens (j/n)? ')
  ; readln (c)
  until c = 'n'

end.
```

char Losse karakters hebben type *Char*.

N.B. Bij herhaling van meer dan één opdracht met **repeat**-**until** zijn **begin**-**end** *niet* nodig.

9 Rijen, lijsten, tabellen opslaan

9.1 Eéndimensionale tabellen

array

Een rij, lijst of tabel van waardes met hetzelfde type kan opgeslagen worden in een **array**. Het element met 'index' i in array a wordt geselecteerd door $a[i]$:

$a[i]$

```
program RijOmkeren;
  { Druk ingelezen rij letters omgekeerd af. }

const
  N = 10;

var
  a: array [ 1 .. N ] of Char; { invoer }
  i: Integer; { doorloopt a }

begin
  write ( 'Geef ', N : 1, ' letters: ' )
; for i := 1 to N do read ( a [ i ] )
; readln
; for i := N downto 1 do write ( a [ i ] )
; writeln
end.
```

Nog een voorbeeld:

```
program Cijfers;
  { Bepaal decimale cijfers van een ingelezen getal. }

const
  Radix = 10;   { grondtal }

var
  n: Integer; { invoervariabele }
  k: Integer; { doorloopt c }
  c: array [ 0 .. 9 ] of Integer; { resultaat }

begin
  write ( 'Geef n (niet negatief): ' )
; readln ( n ) { n = N ∧ 0 ≤ N }
; k := 0
```

; **while** $n \neq 0$ **do begin**
$c[k] := n \bmod$ *Radix*
; $n := n \operatorname{\mathbf{div}}$ *Radix*
; $k := k + 1$
end { while }
{ $N = \Sigma_{i=0}^{k-1} c[i] * \mathit{Radix}^i$ en $0 \leq c[i] < \mathit{Radix}$ }

end.

9.2 Tweedimensionale tabellen

```
program Tafels;
  { Genereer de tafels van 1 t/m 10 en sla ze op. }

const
  N = 10;   { Aantal regels per tafel }
  T = 10;   { Aantal tafels }

var
  i: Integer; { doorloopt de T tafels }
  ij: Integer; { doorloopt de N regels van een tafel }
  tafel: array [ 1 .. N, 1 .. N ] of Integer;    { om tafels op te slaan }

begin

  for i := 1 to T do begin

    for j := 1 to N do
      tafel [ i, j ] := j * i
    end   { for j }

  end   { for i }

end.
```

Oefening. Pas bovenstaande programma aan om de machten i^j op te slaan.

N.B. Machtsverheffen is geen voorgedefinieerde operatie. Gebruik dat $i^1 = i$ en $i^{k+1} = i^k \times i$.

Wat moet je verder veranderen om de machten j^i op te slaan?