Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

L.J.M. Geurts

Cursus programmeren voor beginners
Een kennismaking met de programmeertaal B, Deel I

Department of Computer Science Notitie CS-N8407 Augustus

Programming course for beginners
An introduction to the programming language B, part I

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

CURSUS PROGRAMMEREN VOOR BEGINNERS
EEN KENNISMAKING MET DE PROGRAMMEERTAAL B, DEEL I

L.J.M. GEURTS
Centrum voor Wiskunde en Informatica, Amsterdam

Dit rapport bevat het eerste deel van een beginnerscursus programmeren, gebaseerd op de nieuwe programmeertaal B. B is speciaal ontworpen met het oog op beginners en andere "niet-professionele" gebruikers. De meeste elementaire programmeertechnieken en de meeste eigenschappen van B komen aan bod. De aandacht gaat vooral uit naar het ontwerpen en schrijven van programma's, niet zo zeer naar de praktische omgang met computers. Veel korte programma's worden als voorbeeld gegeven, of als oefening van de lezer gevraagd.
De tekst vereist geen voorkennis, en is zowel voor cursussen als voor zelfstudie geschikt.

1982 CR. CATEGORIES: D.3.3, D.1.0, K.3.2.
TREFWOORDEN: programming, programming languages, B
programmeren, programmeertalen, B

Notitie CS-N8407
Centrum voor Wiskunde en Informatica
Postbus 4079, 1009 AB Amsterdam

Inhoudsopgave

Voorwoord

Dit boek is een cursus programmeren voor beginners, gebaseerd op de nieuwe programmeertaal *B*. *B* is een programmeertaal die speciaal ontworpen is met het oog op beginners en andere niet-professionele computer-gebruikers. Hoewel de kunst van het programmeren niet geheel afhankelijk is van de keuze van een bepaalde programmeertaal, is het wel degelijk bevorderlijk een goede taal te kiezen om te *leren* programmeren. De meest in zwang zijnde programmeertalen zijn tussen de 15 en 25 jaar oud, en stammen uit de dagen dat gemak voor de programmeur minder telde dan computer-tijd. Deze talen zijn niet geschikt voor een programmeercursus voor beginners, omdat ze te moeilijk zijn om te leren en te dicht bij de computer staan om gemakkelijk gebruikt te worden. *B* daarentegen is ontworpen voor de nieuwe generatie van zeer krachtige personal computers die nu op de markt komen, en maakt het mogelijk optimaal van die machines gebruik te maken.
Maar het blijft waar dat programmeren niet gemakkelijk is. De moeilijkheid van programmeren berust op het feit dat computers machines zijn die in beginsel alles kunnen. Het valt nauwelijks te verwachten dat het gemakkelijk kan zijn zo'n machine te vertellen welke van de oneindig vele mogelijke taken uitgevoerd moet worden. *B* verkleint de problemen van het programmeren door de gebruiker in staat te stellen zich te concentreren op het probleem waarmee hij bezig is, i.p.v. op de wispelturigheden en beperkingen van de computer.

Opzet van het boek

Voor het volgen van de inhoud van dit boek is geen voorkennis vereist. Het kan zowel gebruikt worden voor cursussen als voor zelfstudie. De stof gaat vooral over het bedenken en schrijven van programma's, niet zozeer over het invoeren van programma's in de computer, het veranderen van die programma's, enz. In Deel 1 worden de meeste eenvoudige programmeertechnieken en de meeste elementen van *B* gepresenteerd. Veel kleine programma's worden behandeld, of als opgave van de lezer gevraagd. In Deel 2 worden geavanceerdere technieken behandeld, en de overige elementen van de taal. Daar worden ook meer realistische en interessantere voorbeelden gegeven, en wordt het ontwikkelen van grotere programma's bestudeerd.

Opgaven

De opgaven vormen een belangrijk deel van dit boek. Het is van het grootste belang de opgaven evenveel aandacht te geven als de rest van de tekst, omdat leren programmeren iets heel anders is dan lezen *over* programmeren en programma's. Het is als met het leren van elke vreemde taal: je kunt die niet leren door erover te lezen, en zelfs niet door teksten geschreven in die taal te lezen. Het beheersen ervan kan alleen bereikt worden door, met vallen en opstaan, te proberen die taal te spreken en te schrijven.

Het *B*-systeem

Bij het gebruiken van een *B*-systeem blijkt dat dit veel meer doet dan het uitvoeren van *B*-commando's. Het helpt bij het intoetsen van commando's, het veranderen van eerder gegeven commando's, e.d. Omdat een deel van die service nog aan verandering onderhevig is, en omdat het werken hiermee gemakkelijker in de praktijk wordt geleerd, worden deze aspecten niet in dit boek behandeld. Het boek is daardoor gelijkelijk geschikt voor wie wel en voor wie niet een *B*-systeem ter beschikking heeft.
De enige delen van een *B*-systeem die voor dit boek van belang zijn, zijn het *toetsenbord* en het *scherm*. De gebruiker toetst commando's in op het toetsenbord, en die verschijnen dan op het scherm. De computer schrijft de resultaten van de die commando's vervolgens ook op het scherm. Alle letters, cijfers en andere tekens die in dit boek gebruikt worden komen op het toetsenbord voor. Zeer lange commando's, of zeer lange resultaten, die niet op één regel van het scherm passen worden eenvoudig op de volgende regel voortgezet.

1. Rekenen: het WRITE-commando

Het is bekend dat computers goed zijn in rekenen. Later zullen we ze ook gebruiken voor interessantere dingen, maar we beginnen met eenvoudige berekeningen. We kunnen de computer iets laten uitrekenen door WRITE (Engels voor „schrijf") te typen, met daarachter de formule die we willen laten uitrekenen.

Als we bijvoorbeeld de oppervlakte willen weten van een rechthoek van 8 cm bij 12,5 cm, dan typen we:
en krijgen de uitkomst op de volgende regel:

```
WRITE 8*12.5
□ 100.0
```

Zoals we zien wordt de gewoonte uit Engelstalige gebieden gevolgd om een decimale punt te gebruiken in plaats van een decimale komma. Om verwarring te voorkomen zal ik die gewoonte ook in de tekst volgen. Het maalteken wordt als een * getypt. Een formule als 8*12.5 wordt een *expressie* genoemd. Het teken * is de *operator* van deze expressie, 8 en 12.5 zijn de *operanden*.

Om de omtrek van dezelfde rechthoek te weten te komen, kunnen we intoetsen:
Als uitkomst krijgen we dan op het scherm:

```
WRITE (8+12.5)*2
□ 41.0
```

De haakjes zijn nodig om aan te geven dat de optelling vóór de vermenigvuldiging moet worden uitgevoerd. Normaal gaat * vóór +. Laten we de haakjes weg, dan krijgen we dan ook:

```
WRITE 8+12.5*2
□ 33.0
```

Om het volume te berekenen van een doos met de genoemde rechthoek als grondvlak en met een hoogte van 1.75 cm, kunnen we typen:

```
WRITE 8*12.5*1.75
□ 175.000
```

☞ 1.1. Geef een commando om de totale lengte van alle ribben van dezelfde doos te laten berekenen.

Op de volgende manier kunnen we de oppervlakte van de drie verschillende zijvlakken van de doos berekenen:

```
WRITE 8*12.5, 12.5*1.75, 1.75*8
□ 100.0 21.875 14.00
```

We zien dat één WRITE-commando verscheidene expressies aankan, gescheiden door komma's. In het resultaat worden de uitkomsten dan niet door komma's, maar door spaties gescheiden.

Tot nu toe hebben we alleen + en * als operatoren gebruikt. Andere mogelijkheden zijn - en /. Hier worden die gebruikt om uit te rekenen hoeveel graden Celsius overeenkomen met 50 graden Fahrenheit:

```
WRITE "50 F =", (50-32)*5/9, "C"
□ 50 F = 10 C
```

We zien in dit voorbeeld ook dat WRITE met teksten kan werken, die we tussen " en " moeten typen, en die dan letterlijk in het resultaat verschijnen.

☞ 1.2. Verander het bovenstaande commando zo, dat het dezelfde berekening uitvoert voor 100 graden Fahrenheit.

De algemene regel is: * en / gaan vóór (hebben hogere prioriteit dan) + en -. Als + en - enkele keren na elkaar gebruikt worden, worden ze van links naar rechts uitgevoerd.

```
WRITE 7*3+10/3-24
□ 0.3333333333333333
WRITE 8-4+3, 8-(4+3)
□ 7 1
WRITE -1+3, -(1+3)
□ 2 -4
```

Als we een aantal malen * en/of / in een expressie willen gebruiken, kan de volgorde waarin ze worden uitgevoerd verschil maken:

```
WRITE (24/6)/2, 24/(6/2)
□ 2 8
WRITE (24/6)*2, 24/(6*2)
□ 8 2
```

In zulke gevallen moeten we haakjes gebruiken, maar die mogen we weglaten als de volgorde *geen* verschil maakt (zoals we al gezien hebben in de voorbeelden voor het volume en de graden):

```
WRITE (18*5)/10, 18*(5/10), 18*5/10
□ 9 9 9
```

Een precieze beschrijving van de prioriteiten zullen we later zien.
Als we de haakjes weglaten in een combinatie van * en/of / waarbij de volgorde van uitvoering *wel* verschil maakt, dan voert de computer dat commando niet uit, maar schrijft een boodschap over deze fout op het scherm.

Dat gebeurt altijd als een commando niet volgens de regels is. Als we bijvoorbeeld dit commando geven (met ✗ geef ik aan dat dit fout is):

```
✗ WRITE 7/(18-3-15)
```

dan krijgen we een foutmelding te zien die er ongeveer zo uitziet:

```
□ *** Deling door 0 niet toegestaan
```

Bij twijfel over wat er vóór wat gaat kan het nooit kwaad voor de zekerheid haakjes te gebruiken:

```
WRITE 2+3*4, 2+(3*4), ((2)+(((3)*4)))
□ 14 14 14
```

☞ 1.3. Wat wordt er op het scherm geschreven door deze drie commando's?

```
a. WRITE "1+1 is", 1+1
b. WRITE -1/4, -1/40, -1/400
c. WRITE 3+9/3, (3+9)/3
```

☞ 1.4. Geef een commando om de computer te laten schrijven:
a. het gemiddelde van de getallen −17, 14.157 and 500;
b. de dikte van een stapel van 142 vellen papier, als ieder vel 0.013 cm dik is;
c. de gemiddelde groei per jaar van een boom die in 1837 1.85 m hoog was, en in 1984 35.30 m;
d. 7% van 103.12;
e. 15% minder dan 157.

Er zijn nog meer operatoren die we in expressies kunnen gebruiken. Ik zal er hier enkele behandelen. De overige zullen we later tegenkomen.

De operator `round` (Engels voor „rond", „afronden") levert het dichtstbijzijnde gehele getal op:

```
WRITE round 2.4, round 2.5, round 2.6
□ 2 3 3
WRITE round (1-5/3), 2 * round 2
□ -1 4
```

De haakjes zijn hier nodig omdat `round 1-5/3` ook op een andere manier opgevat zou kunnen worden, nl. als `(round 1)-5/3`.

Als we willen afronden op een bepaald aantal cijfers achter de decimale punt, dan kunnen we `round` gebruiken met twee operanden i.p.v. één. Het linker-operand geeft aan op hoeveel cijfers achter de decimale punt er wordt afgerond:

```
WRITE 3 round (2/3), 6 round (800/16)
□ 0.667 50.000000
```

Er zijn nog twee manieren om een getal af te ronden: `floor` (Engels voor „vloer") om naar beneden af te ronden naar het dichtstbijzijnde gehele getal dat kleiner (of gelijk) is, `ceiling` (Engels voor „plafond") om af te ronden naar het eerst volgende gehele getal dat groter (of gelijk) is:

```
WRITE floor (-10/3), floor (91/7)
□ -4 13
WRITE ceiling (-10/3), ceiling (91/7)
□ -3 13
```

Om de rest van een deling te berekenen is er de operator `mod` (afkorting van de wiskundige term *modulo*), die twee operanden nodig heeft:

```
WRITE floor (7/3), 7 mod 3
□ 2 1
WRITE floor (5.3/1.1), 5.3 mod 1.1
□ 4 0.9
WRITE floor (60/20), 60 mod 20
□ 3 0
WRITE floor (3.1/8), 3.1 mod 8
□ 0 3.1
```

☞ 1.5. Bedenk voor elk van de volgende problemen een commando dat de computer het antwoord laat vinden.

a. Welk geheel getal ligt het dichtst bij het kwadraat van 12.7?
b. Hoeveel mensen kunnen hun *gehele* aandeel van 1.31 kg uit een voorraad van 6.7 kg chocola nemen?
c. Hoeveel chocola is er dan over?
d. Hoeveel mandjes zijn er minstens nodig om 141 eieren in te doen, als er in één mandje 8 eieren passen?
e. Als 7 kinderen 1351 knikkers gelijk onder elkaar willen verdelen, hoeveel knikkers blijven er dan over?
f. Hoeveel minuten en seconden komen overeen met 3859 seconden? Laat het resultaat er ongeveer zo uitzien:

```
150 seconden = 2 minuten 30 seconden
```

g. Hoeveel meter, afgerond naar de dichtstbijzijnde hele centimeter, is een zevende deel van 200 meter?
h. Er staan zeven kinderen in een kring. Zij hebben rugnummers, en ze staan in volgorde van die nummers, waarbij nummer 7 tussen nummer 6 en nummer 1 staat. Als we, te beginnen bij kind 1, tot 5 tellen, dan eindigen we dus bij kind 5. Als we tot 9 tellen, dan eindigen we bij kind 2. Waar zouden we eindigen als we tot 3528 tellen?

Regels

- De operatoren `*` en `/` gaan vóór `+` en `-`.
- Een serie operatoren `+` en/of `-` wordt van links naar rechts uitgewerkt.
- In een serie operatoren `*` en/of `/` zijn haakjes nodig, tenzij de volgorde van uitvoering voor het resultaat geen verschil maakt.

2. Het geheugen: het PUT-commando

Als we de oppervlakten willen weten van de drie verschillende zijkanten van een doos, en ook de totale buitenoppervlakte, dan moeten we enkele expressies tweemaal intoetsen:

```
WRITE 22*33.3, 33.3*44.44, 44.44*22
□ 732.6 1479.852 977.68
WRITE 2*(22*33.3+33.3*44.44+44.44*22)
□ 6380.264
```

Er is een manier om dit soort herhaling te voorkomen. Met behulp van een PUT-commando (Engels voor „zetten", „plaatsen") kunnen we aan de uitkomst van een berekening een naam geven, en vanaf dat moment die naam gebruiken in plaats van die uitkomst, zoals te zien is in dit *programma*, d.w.z. serie commando's.

```
PUT 22*33.3 IN a1
PUT 33.3*44.44 IN a2
PUT 44.44*22 IN a3
WRITE a1, a2, a3, (a1+a2+a3)*2
□ 732.6 1479.852 977.68 6380.264
```

Hoe gaat dit in zijn werk? Om het commando `PUT 22*33.3 IN a1` te gehoorzamen berekent de computer `732.6` als de uitkomst van de vermenigvuldiging, en bergt die uitkomst in zijn geheugen op „met een etiket `a1` erop geplakt". Het geheugen van de computer kan vergeleken worden met een groot vel papier waarop alleen de computer kan kijken en veranderingen kan aanbrengen. Het uitvoeren van het commando `PUT 22*33.3 IN a1` komt dan neer op het schrijven van `a1 = 732.6` in het geheugen. Als de computer later een commando als `WRITE a1+1` moet uitvoeren, zoekt hij `a1` op in zijn geheugen, vindt daar `732.6`, berekent `732.6+1` en schrijft `733.6` als *uitvoer*, d.w.z. op het scherm als resultaat van de `WRITE`-opdracht.
Laten we eens een reeks commando's op de voet volgen, en zien hoe het geheugen van de computer erdoor veranderd wordt. De gearceerde stukken hieronder stellen het geheugen voor.

Aanvankelijk is het geheugen leeg:

Na het commando:

```
PUT 3+4 IN p
```

bevat het geheugen één *lokatie*, met *etiket* `p` en *waarde* `7`:

```
p = 7
```

We kunnen een nieuwe lokatie maken door er eenvoudig met PUT iets in te zetten:

```
PUT 2*3 IN q
```

Nu bevat het geheugen twee lokaties:

```
p = 7   q = 6
```

De waarde van een lokatie is niet voor eeuwig onveranderbaar, maar kan door een later PUT-commando veranderd worden:

```
PUT 4 IN p
```

Hierdoor ontstaat niet een nieuwe lokatie, maar wordt de waarde van de bestaande lokatie `p` veranderd:

```
p = 4   q = 6
```

We kunnen de waarde van een lokatie op het scherm laten schrijven met een WRITE-commando:

```
WRITE p
□ 4
```

Hierbij blijft het geheugen onveranderd:

```
p = 4   q = 6
```

In een expressie mogen we, in plaats van voluit geschreven getallen, ook etiketten van lokaties gebruiken:

```
PUT p+q IN sum
```

Het resultaat is:

```
p = 4   q = 6   sum = 10
```

Om de waarde van `p` te kopiëren in `q` kunnen we dit commando gebruiken:

```
PUT p IN q
```

Hierdoor wordt de waarde van sum en p niet veranderd:

```
p = 4     q = 4     sum = 10
```

Om de waarde in lokatie p, wat die ook is, met 2 te verhogen, kunnen we dit commando geven:

```
PUT p+2 IN p
```

Nu bevat het geheugen:

```
p = 6     q = 4     sum = 10
```

Het is ook mogelijk met PUT verscheidene waarden tegelijk in verscheidene lokaties te plaatsen. De computer doet dat door eerst de expressies tussen PUT en IN uit te rekenen en dan elke uitkomst in de overeenkomstige lokatie te plaatsen:

```
PUT 9/3, 2*2 IN p, q
```

```
p = 3     q = 4     sum = 10
```

```
PUT p*q, p/q IN p, q
```

```
p = 12    q = 0.75  sum = 10
```

Die methode kan ook gebruikt worden om op een elegante manier de waarden van twee lokaties te verwisselen:

```
PUT p, q IN q, p
```

```
p = 0.75  q = 12    sum = 10
```

Regels

- Voor lokaties kan elk etiket gebruikt worden dat is samengesteld uit *kleine* letters, cijfers en het apostrof-teken ', maar het etiket moet beginnen met een letter.

Dit zijn dus correcte etiketten:

```
a  a1  a'plus'1  y'  verschil
```

en dit zijn foute etiketten:

```
✗ 1'plus'a  pH  'ja'
```

Omdat een etiket geen spaties kan bevatten, moeten etiketten die we uit meerdere woorden willen opbouwen ofwel gevormd worden door die woorden zonder spaties achter elkaar te plaatsen:

```
langekant
```

of liever door een ' tussen de woorden te zetten:

```
lange'kant
```

- Met PUT kan natuurlijk alleen iets in een *lokatie* geplaatst worden, niet in een getal. Dit is dus fout:

```
✗ PUT p-1 IN 5
```

- Ieder commando van een programma begint op een nieuwe regel.

☞ 2.1. Wat wordt door dit programmaatje als uitvoer op het scherm geschreven?

```
PUT 1.001, 19.513 IN v, x
WRITE x-v, x, x+v
```

☞ 2.2. Geef een commando om de waarde in lokatie a met 50% te vermeerderen.

☞ 2.3. Hoe kan aan de lokatie gemiddelde de waarde gegeven worden die halverwege ligt tussen de waarden van de lokaties p en q?

☞ 2.4. Als a en b lokaties zijn, gebruik dan het PUT-commando om de waarden van a en b zo te veranderen dat ze allebei de som van de twee oude waarden bevatten.

☞ 2.5. De lokatie afstand bevat het aantal kilometers dat een trein aflegt; de lokatie tijd bevat het aantal minuten dat hij daarvoor nodig heeft. Plaats de snelheid van de trein, uitgedrukt in km/uur, in de lokatie snelheid.

☞ 2.6. Gezien vanuit het getal 34 is 36 het dichtstbijzijnde veelvoud van 9. Wat is het dichtstbijzijnde veelvoud van 9 gezien vanuit de waarde in lokatie n?

☞ 2.7. Als 1 mijl = 1760 yard, 1 yard = 3 foot and 1 foot = 12 inch:
a. Geef de lokaties `mijl`, `yard` en `foot` een zodanige waarde dat `mijl` aangeeft hoeveel inch er in een mijl gaan, `yard` hoeveel inch een yard is, en `foot` hoeveel inch een foot is.
b. Gebruik die lokaties om te schrijven hoe lang een afstand van 1 mijl 3 yard 2 foot en 2.5 inch is, uitgedrukt in inches.
c. Gebruik die lokaties ook om te bepalen hoeveel mijl, yard, foot en inch overeenkomt met 1234567 inch. Laat de uitvoer er ongeveer zo uitzien:

```
9999999 inch = 13 ml 268 yrd 5 ft 3 inch
```

☞ 2.8. We hebben gezien dat de waarden van twee lokaties verwisseld kunnen worden met een „dubbel" PUT-commando als `PUT lok1, lok2 IN lok2, lok1`. Probeer hetzelfde te bereiken zonder een dubbel PUT-commando te gebruiken. (Aanwijzing: het staat vrij nieuwe lokaties met nieuwe etiketten in gebruik te nemen.)

☞ 2.9. Als `b` een lokatie is die een positief geheel getal bevat (b.v. `341`), geef dan een manier om een `4` aan het eind van dat getal toe te voegen (zodat het `3414` zou worden). Zorg ervoor dat het programma werkt voor elk positief getal dat `b` kan bevatten.

☞ 2.10. De lokatie `pg` bevat een positief getal (b.v. `4.13` of `2`). Laat die waarde zo veranderen dat alleen het deel na de decimale punt overblijft (in deze voorbeelden `0.13` of `0`). (Aanwijzing: gebruik `mod`.)

☞ 2.11. Geef een kortere manier om hetzelfde te doen als dit programma. Zorg ervoor dat het resultaat werkelijk hetzelfde is, wat de oorspronkelijke waarden van `a` en `b` ook zijn.

```
PUT a-b IN b
PUT a-b IN a
PUT a+b IN b
```

☞ 2.12. Doe hetzelfde voor dit programma:

```
PUT y, -x IN x, y
PUT y, -x IN x, y
PUT y, -x IN x, y
```

3. Herhaling: het WHILE-commando

Met de commando's WRITE en PUT kunnen alleen programma's voor vrij eenvoudige problemen worden geschreven. Door een of meer commando's verscheidene keren te laten herhalen is het mogelijk interessantere resultaten te bereiken.

Met een WHILE-commando (Engels voor „zolang") kan een stuk programma herhaald worden zolang aan een bepaalde voorwaarde voldaan is. Hier is een programmaatje dat alle veelvouden van het getal 142857 op het scherm schrijft die kleiner zijn dan 500000.

```
PUT 142857 IN veel
WHILE veel < 500000:
   WRITE veel
   PUT veel+142857 IN veel
□ 142857 285714 428571
```

Hoe werkt dit nu?
Zodra de computer bij de WHILE-regel komt, gebeurt er het volgende. De computer kijkt of veel kleiner is dan 500000. (Dat is de betekenis van <.) Zo niet, dan wordt het *hele* WHILE-commando, d.w.z. inclusief de twee ingesprongen regels, overgeslagen, en is dit programma dus afgelopen. Maar als veel wel kleiner is dan 500000, dan wordt eerst het ingesprongen gedeelte uitgevoerd, en gaat de computer daarna terug naar de WHILE-regel, en begint daar opnieuw te controleren of veel kleiner is dan 500000, enzovoort.
Het is dus moeilijk te voorspellen hoe vaak het ingesprongen gedeelte wordt uitgevoerd, maar we weten wel dat, als het gehele WHILE-commando eenmaal ten einde is gekomen, veel niet langer een waarde kleiner dan 500000 heeft.

☞ 3.1. Verander het bovenstaande programma om het de veelvouden van 7 te laten schrijven die kleiner zijn dan 100.

Uitdrukkingen als veel < 500000 heten *condities*. Hier volgt een lijst van de speciale tekens die gebruikt kunnen worden om condities op te bouwen:

a < b voor: a *is kleiner dan* b
a > b voor: a *is groter dan* b
a = b voor: a *is gelijk aan* b
a <= b voor: a *is kleiner dan of gelijk aan* b
a >= b voor: a *is groter dan of gelijk aan* b
a <> b voor: a *is niet gelijk aan* b

☞ 3.2. Gebruik de beschrijving hierboven om vast te stellen wat er door deze drie programma's geschreven wordt:

```
PUT 2 IN a
WHILE a < 3:
   WRITE a
   PUT a+2 IN a
```

```
PUT 2 IN a
WHILE a <= 6:
   WRITE a
   PUT a+2 IN a
```

```
PUT 2 IN a
WHILE a >= 3:
   WRITE a
   PUT a+2 IN a
```

☞ 3.3. Als n het getal 1703 bevat, wat zal n dan bevatten nadat dit programma is uitgevoerd?

```
WHILE n > 17:
   PUT n-17 IN n
```

☞ 3.4. Als n een positief getal bevat, wat zal dan de uitvoer zijn van dit programma? (Denk erom dat er maar één getal wordt geschreven, omdat het WRITE-commando niet ingesprongen is, zodat het niet behoort tot het gedeelte dat herhaald wordt.)

```
PUT 0 IN i
WHILE i<n:
   PUT i+1 IN i
WRITE i
```

We zullen nu het programma met 142857 zo veranderen dat het elk veelvoud op een nieuwe regel schrijft, voorafgegaan door het getal dat aangeeft welk veelvoud het is. (Dus de derde regel uitvoer moet luiden: 3 428571.) Het programma moet stoppen zodra een miljoen is bereikt. Het teken / aan het eind van het WRITE-commando geeft aan dat verdere uitvoer op de volgende regel moet beginnen.
Bestudeer dit voorbeeld zorgvuldig, totdat duidelijk wordt dat de getoonde uitvoer inderdaad geproduceerd wordt.

```
PUT 1, 142857 IN n, veel
WHILE veel < 1000000:
   WRITE n, veel /
   PUT n+1, veel+142857 IN n, veel
```

```
☐ 1 142857
☐ 2 285714
☐ 3 428571
☐ 4 571428
☐ 5 714285
☐ 6 857142
☐ 7 999999
```

Laten we eens precies nagaan wat de rol is van de lokaties die centraal staan in dit WHILE-commando: de lokaties n en veel. De lokatie n speelt de rol van het rangnummer van de volgende uitvoerregel, en veel is het veelvoud van 142857 dat aan de beurt is om geschreven te worden. Kortom, n is het regelnummer en veel = n*142857.
Het is altijd zeer belangrijk de rol van de centrale lokaties van een WHILE-commando goed in gedachten te houden. In dit verband is een lokatie *centraal* als de waarde ervan in het WHILE-commando veranderd kan worden, en die waarde nog van belang is voor de volgende keer dat het ingesprongen gedeelte uitgevoerd wordt. (In dit voorbeeld zijn alle lokaties centraal, maar later zullen we ook WHILE-commando's tegenkomen met andere lokaties.) De rol van een centrale lokatie kan het best geformuleerd worden als een beschrijving die elke keer klopt als het ingesprongen stuk van het WHILE-commando op het punt staat uitgevoerd te worden.
Hier zijn nog enkele versies van hetzelfde programma. Besteed speciale aandacht aan de beschrijving van de rol van de centrale lokaties.

Hier is een versie van hetzelfde programma, gebaseerd op dezelfde rollen van de centrale lokaties: n is weer het regelnummer van de volgende uitvoerregel, en veel = n * 142857.

```
PUT 1, 142857 IN n, veel
WHILE veel < 1000000:
   WRITE n, veel /
   PUT n+1 IN n
   PUT n*142857 IN veel
```

De eerste keer dat de computer bij het *begin van het ingesprongen stuk* komt is er nog geen uitvoer geschreven, en kloppen de waarden van de centrale lokaties met hun rolbeschrijving, want ze zijn:

```
n = 1   veel = 142857
```

De volgende keer dat de computer bij dit punt komt is er één uitvoerregel geschreven, en hebben de centrale lokaties deze waarden, die passen bij hun rol:

```
n = 2   veel = 285714
```

De volgende keer, na twee uitvoerregels, kloppen de waarden weer met de beschrijving, en dat blijft verder zo:

```
n = 3   veel = 428571
```

Van dit programma kan ook een versie geschreven worden die gebaseerd is op deze iets verschillende rol voor de centrale lokaties:
n is het aantal tot nu toe geschreven uitvoerregels, en veel = n*142857.

```
PUT 0, 0 IN n, veel
WHILE veel+142857 < 1000000:
   PUT n+1, veel+142857 IN n, veel
   WRITE n, veel /
```

Hier is een versie met maar één centrale lokatie: n, die het rangnummer van de volgende uitvoerregel aangeeft.

```
PUT 1 IN n
WHILE n*142857 < 1000000:
   WRITE n, n*142857 /
   PUT n+1 IN n
```

Het is een kwestie van smaak welke versie de beste is. Een belangrijke maatstaf is: welke versie is het gemakkelijkst te begrijpen, zodat het zo duidelijk mogelijk is dat er geen fouten in voorkomen. Daarvoor is het vaak goed weinig lokaties in het programma te hebben.

☞ 3.5. Ontwerp nog een versie van hetzelfde programma, nu gebaseerd op één centrale lokatie: `n`, die het aantal tot nu toe geschreven uitvoerregels aangeeft.

Hier is een programma dat laat zien hoe een kapitaal van 1000 gulden groeit als er elk jaar 13 procent aan wordt toegevoegd. Het programma stopt zodra het kapitaal verdubbeld is. Ik gebruik de lokatie `som` om de som gelds te bevatten, en de lokatie `j` om het jaar bij te houden. Zoals we zien rondt de bank de som elk jaar af naar de dichtstbijzijnde cent.

```
PUT 1984, 1000 IN j, som
WHILE som < 2000:
   PUT j+1 IN j
   PUT 2 round (som*1.13) IN som
   WRITE j, som /
```

```
1985 1130.00
1986 1276.90
1987 1442.90
1988 1630.48
1989 1842.44
1990 2081.96
```

☞ 3.6. Verander het bovenstaande programma zo, dat de bank de som elk jaar precies berekent, en dat de afronding alleen in de uitvoer gebeurt.

Als we een `WHILE`-commando programmeren is het in de gaten houden van de rollen van de centrale lokaties niet onze enige zorg. We moeten er ook voor zorgen dat het programma ooit tot een einde komt.

Neem dit programmaatje voor het schrijven van oneven getallen. Het is gemakkelijk in te zien dat het programma braaf begint oneven getallen te schrijven, maar het probleem is dat het daar nooit mee ophoudt.

```
PUT 1 IN a
WHILE a > 0:
   WRITE a
   PUT a+2 IN a
```

En hier is nog een programma dat nooit stopt. (Wie dit soort fout in de praktijk maakt kan de speciale stop-toets gebruiken om de uitvoering van het programma af te breken.)

```
PUT 1 IN a
WHILE a <> 10:
   WRITE a
   PUT a+2 IN a
```

☞ 3.7. Hoewel de beide vorige programma's nauwelijks correct te noemen zijn, is het toch mogelijk de rol van de centrale lokatie `a` te beschrijven. Wat is die rol?

We kunnen een conditie opbouwen uit andere condities door die te combineren met behulp van `AND` (Engels voor „en"), zoals te zien is in dit programma voor het schrijven van de veelvouden van 7 onder de 1000, totdat het eerste veelvoud eindigend op 33 geschreven is. In dit voorbeeld met `AND` gaat de herhaling door zolang nog geldt dat `v+7 < 1000` *en* `v mod 100 <> 33`.

```
PUT 0, 0 IN n, v
WHILE v+7 < 1000 AND v mod 100 <> 33:
   PUT n+1 IN n
   PUT n*7 IN v
   WRITE n, v /
```

De uitvoer van het programma ziet er zo uit:

```
□ 1 7
□ 2 14
□ 3 21
□ 4 28
□ 5 35
□ 6 42
□ 7 49
□ 8 56
□ 9 63
□ 10 70
□ 11 77
□ 12 84
□ 13 91
□ 14 98
□ 15 105
□ 16 112
□ 17 119
□ 18 126
□ 19 133
```

In sommige combinaties is AND niet nodig. In plaats van `WHILE -3 < p AND p <= 7` kunnen we schrijven `WHILE -3 < p <= 7`. Evenzo betekent `a = b < 4` hetzelfde als `a = b AND b < 4`.

Condities kunnen ook gecombineerd worden met behulp van OR (Engels voor „of"). Hier is een voorbeeld waarin de waarde van a gehalveerd wordt zolang die groter is dan 10 *of* kleiner is dan −10:

```
WHILE a > 10 OR a < -10:
   PUT a/2 IN a
```

Als we een herhaling willen laten doorgaan zolang aan een bepaalde voorwaarde *niet* voldaan is, kunnen we NOT (Engels voor „niet") voor een conditie zetten, zoals te zien is in dit programma dat precies hetzelfde doet als het vorige:

```
WHILE NOT -10 <= a <= 10:
   PUT a/2 IN a
```

Regels

- De commando's van een WHILE-commando die herhaaldelijk moeten worden uitgevoerd, moeten ingesprongen getypt worden. Eventuele volgende commando's die niet herhaald moeten worden, moeten weer recht onder WHILE beginnen.

- Als AND, OR en NOT in dezelfde conditie voorkomen zijn er haakjes nodig:

```
WHILE (a > 1 AND b > 2*a) OR a = b:
```

- In een AND-combinatie stopt de computer met uitwerken van de afzonderlijke condities zodra hij er een is tegengekomen die niet waar is; in een OR-combinatie stopt hij bij de eerste ware conditie.

Advies

- Bij het schrijven van het ingesprongen deel van een WHILE-commando is het verstandig de situatie in gedachten te nemen ergens halverwege het proces, niet de situatie bij de eerste keer dat dit stuk wordt uitgevoerd, want die is meestal nogal afwijkend.

Bij het bedenken van b.v. een programma om de rij 2, 4, 8, 16 enz. te laten schrijven bestaat de verleiding om zo te beginnen:

```
PUT 1 IN n
WHILE n < 1000:
   WRITE 2
   enz.
```

Dit soort fout komt voort uit te veel aandacht voor de eerste keer, en te weinig voor het gemiddelde latere geval. Enig nadenken over zo'n later moment in het proces leert dat er een lokatie als `n` nodig is om het laatst geschreven getal te onthouden. Bij elke herhaling moet de waarde van `n` verdubbeld worden en moet die nieuwe waarde geschreven worden.

```
PUT 1 IN n
WHILE n < 1000:
   PUT 2*n IN n
   WRITE n
```

Het is goed ook na het schrijven van een `WHILE`-commando na te gaan of wat er bij de eerste en de laatste herhaling gebeurt wel correct is, want daar komen de meeste fouten voor.

• Een `WHILE`-commando moet ontworpen worden met de rollen van de centrale lokaties in gedachten. Die rollen moeten elke keer kloppen als de computer op het punt staat het ingesprongen gedeelte uit te voeren.

• In de voorbeelden hebben we een vast patroon gezien voor het gebruiken van de beschrijving van de rollen van de centrale lokaties. Hier wordt nog eens gedemonstreerd hoe dat patroon wordt toegepast op een van de programma's om veelvouden van 142857 te schrijven, nl. de versie waarin `n` het rangnummer van de volgende uitvoerregel aangeeft.

1. Voorafgaande aan het `WHILE`-commando geven we elke centrale lokatie een waarde die klopt met zijn rol:

```
PUT 1 IN n
```

2. In de `WHILE`-regel gebruiken we een of meer van de centrale lokaties om aan te geven hoe lang de herhaling moet doorgaan:

```
WHILE n*142857 < 1000000:
```

3. In het ingesprongen gedeelte doen we een stap in de richting van het doel van het programma. In dit geval is zo'n stap het schrijven van de volgende uitvoerregel:

```
   WRITE n, n*142857 /
```

4. Na zo'n stap zal de beschrijving van de rollen van de centrale lokaties in het algemeen niet meer opgaan, dus veranderen we nu de waarden van de centrale lokaties zo, dat ze weer voldoen aan de beschrijving:

```
   PUT n+1 IN n
```

☞ 3.8. Schrijf programma's die de getallen van de volgende rijen schrijven die kleiner zijn dan 1000. Beschrijf bij elk programma de rol van de centrale lokaties.

a. `1 2 4 8 ...`
b. `1 4 9 16 ...`
c. `1 2 6 24 ...` (Aanwijzing: `1*2*3*4*...`)
d. `111 111 222 222 333 333 444 ...`
e. `0 1 1 2 3 5 8 13 ...` (Aanwijzing: 13 = 5+8)

☞ 3.9. De lokatie `start` bevat een positief geheel getal. Schrijf een programma dat dit getal schrijft, dan `start` het dubbele van dat getal als waarde geeft en die waarde schrijft, die waarde weer verdubbelt, en zo verder totdat er een getal geschreven is dat eindigt op `00`, `08` of `80`.

☞ 3.10. De lokaties `lengte` en `breedte` bevatten waarden die de afmetingen aangeven van een rechthoekig vel papier, b.v. `lengte = 15`, `breedte = 10`. Schrijf een programma om de afmetingen van het vel te schrijven nadat het in tweeën is gevouwen (waarbij de lengte wordt gehalveerd), dan nadat het weer in tweeën is gevouwen haaks op de eerste vouw, dan weer haaks op de tweede vouw, enzovoort. De eerste twee uitvoerregels zouden er voor dit voorbeeld zo uitzien:

```
10 7.5
7.5 5
```

Laat het vouwen ophouden zodra de te vouwen zijde kleiner is dan `1`. Formuleer ook de rol van de centrale lokaties van het programma.

Ook in gevallen waar er maar een enkel stukje uitvoer verlangd wordt kan een WHILE-commando nuttig zijn. Als we b.v. het kleinste positieve gehele getal a willen weten waarvan a*a+a groter is dan 100, dan kunnen we dat doen door eerst a = 1 te proberen, dan, als het nog niet groter is, a = 2, enzovoort, totdat we bij een getal aanlanden waarvan het wel groter is.

```
PUT 1 IN a
WHILE NOT a*a+a > 100:
   PUT a+1 IN a
WRITE a
```

De rol van a is hier: voor alle positieve gehele getallen i kleiner dan a geldt: NOT i*i+i > 100. Het is gemakkelijk na te gaan dat dit in het begin klopt, en als het eenmaal klopt, blijft het kloppen totdat het WHILE-commando is afgelopen. Na afloop weten we dus dat a*a+a > 100 én dat dit voor kleinere positieve gehele getallen niet opgaat. Uit deze twee feiten kunnen we veilig afleiden dat a het gezochte getal bevat.

☞ 3.11. Gebruik dezelfde aanpak om het kleinste positieve even getal te vinden waarvan het kwadraat groter of gelijk is aan de waarde van s.

Een ander voorbeeld van een WHILE-commando dat vele malen wordt uitgevoerd en toch maar één getal als uitvoer levert is dit programma om de som te berekenen van de getallen 1, 2, 6, 24, ... onder de 1000. (Zoals we zien is het vierde getal 4 maal zo groot als het derde getal.) De rollen van de centrale lokaties zijn:
i is het volgende getal waarmee vermenigvuldigd moet worden,
p is het (i-1)-de getal van de rij, en
s is de som van de eerste i-2 getallen van de rij
(dus p is het volgende bij s te tellen getal als het kleiner is dan 1000).

```
PUT 0, 1, 2 IN s, p, i
WHILE p < 1000:
   PUT s+p, p*i, i+1 IN s, p, i
WRITE s
□ 873
```

☞ 3.12. Schrijf een nieuwe versie van het bovenstaande programma, gebaseerd op de volgende rollen voor de centrale lokaties:
s is de som van de eerste i getallen van de rij,
p is het i-de getal van de rij, dus
i is rangnummer van p.

☞ 3.13. Schrijf een programma dat het eerste getal groter dan 1000 vindt van de rij 0 1 1 2 3 5 8 13 ... .

☞ 3.14. Schrijf een programma dat de kleinste deler als uitvoer geeft van het gehele getal in lokatie n, dat groter is dan 1. Een *deler* is een geheel getal groter dan 1 waardoor n precies deelbaar is, dus n heeft minstens één deler: n zelf.

We hebben deze twee voordelen gezien van het gebruik van lokaties:
1. We kunnen er langdradige herhalingen van expressies mee voorkomen.
2. Ze spelen de centrale rollen in WHILE-commando's. Het is niet goed mogelijk een redelijk WHILE-commando te schrijven zonder lokaties.

4. Eigen commando's: HOW'TO

We hebben gezien hoe we van bepaalde soorten overbodige herhaling kunnen afkomen door lokaties te gebruiken. Maar er zijn vormen van herhaling die niet op die manier vermeden kunnen worden.

Als we b.v. alle veelvouden van 142857 en alle veelvouden van 123456 onder de 500000 willen zien, moeten we dit programma schrijven:

```
PUT 142857 IN veel
WHILE veel < 500000:
   WRITE veel
   PUT veel+142857 IN veel
□ 142857 285714 428571
PUT 123456 IN veel
WHILE veel < 500000:
   WRITE veel
   PUT veel+123456 IN veel
□ 123456 246912 370368 493824
```

Als *B* nu maar een speciaal commando VERVEELVOUDIG x TOT y had met de juiste betekenis, dan zou het programma verkort kunnen worden tot maar twee commando's:

```
VERVEELVOUDIG 142857 TOT 500000
VERVEELVOUDIG 123456 TOT 500000
```

Nu heeft *B* geen ingebouwd VERVEELVOUDIG-commando, maar er is wel een manier om zelf nieuwe commando's te definiëren. We hoeven de computer alleen maar te vertellen hoe zo'n commando uitgevoerd moet worden. Voor ons eigen VERVEELVOUDIG-commando kan dat als volgt. (HOW'TO is Engels voor „hoe te". WRITE / betekent: ga over op een nieuwe regel, zodat eventuele verdere uitvoer daar begint.)

```
HOW'TO VERVEELVOUDIG enkel TOT grens:
   PUT enkel IN veel
   WHILE veel < grens:
      WRITE veel
      PUT veel+enkel IN veel
   WRITE /
```

De woorden VERVEELVOUDIG en TOT zijn de *sleutelwoorden* van deze definitie, enkel en grens zijn de *voorlopige parameters*. Nu we deze definitie gegeven hebben, wordt hij niet uitgevoerd, maar opgeborgen in het *definitie-geheugen* van de computer. (Het andere deel is het *lokatie-geheugen*.) Van nu af aan kunnen we het nieuwe commando vrijelijk gebruiken, net als alle andere commando's van de taal.

Hier is een toepassing van het vers gedefinieerde VERVEELVOUDIG-commando.

```
VERVEELVOUDIG 142857 TOT 500000
□ 142857 285714 428571
```

In deze toepassing zijn VERVEELVOUDIG en TOT de *sleutelwoorden*, gelijk aan die van de commando-definitie; 142857 en 500000 zijn de *feitelijke parameters*, die corresponderen met de voorlopige parameters van de commando-definitie.

☞ 4.1. Gebruik het VERVEELVOUDIG-commando om de computer de positieve even getallen tot en met 50 te laten schrijven.

We zullen de stappen volgen die de computer doorloopt als hij dit programmaatje uitvoert:

```
PUT 50 IN hoog
VERVEELVOUDIG 7 TOT hoog-1
```

1. De computer geeft de lokatie hoog een waarde. Het lokatie-geheugen bevat daarna:

```
hoog = 50
```

2. De computer zoekt nu in het definitie-geheugen naar een definitie die begint met HOW'TO VERVEELVOUDIG, en controleert of het tweede sleutelwoord TOT is. Als die definitie er niet is, of als het tweede sleutelwoord niet klopt, geeft de computer een foutmelding; anders kopieert hij de definitie op een tijdelijk stuk extra geheugen:

```
HOW'TO VERVEELVOUDIG enkel TOT grens:
   PUT enkel IN veel
   WHILE veel < grens:
      WRITE veel
      PUT veel+enkel IN veel
   WRITE /
```

3. In deze kopie verandert de computer elk voorkomen van de voorlopige parameters `enkel` en `grens` door `(7)` en `(hoog-1)`. De reden voor de haakjes wordt duidelijk in hoofdstuk 5.3.

```
HOW'TO VERVEELVOUDIG (7) TOT (hoog-1):
   PUT (7) IN veel
   WHILE veel < (hoog-1):
      WRITE veel
      PUT veel+(7) IN veel
   WRITE /
```

4. De computer veegt de `HOW'TO`-regel van de kopie uit, en voert het overblijvende programma uit.

```
PUT (7) IN veel
WHILE veel < (hoog-1):
   WRITE veel
   PUT veel+(7) IN veel
WRITE /
```

Dit geeft als uitvoer op het scherm:

```
□ 7 14 21 28 35 42
```

5. Tenslotte gooit de computer het tijdelijke stuk extra geheugen weg, maar bewaart wel de oorspronkelijke commando-definitie in het definitie-geheugen. Het lokatie-geheugen ziet er na afloop als volgt uit. (Zoals we zien blijft de lokatie `veel` niet bewaard. Dat zal later duidelijk worden.)

```
hoog = 50
```

De computer werkt deze stappen af zonder daarvan iets te laten zien, behalve natuurlijk de resultaten van eventuele `WRITE`-commando's die hij tegenkomt. Dus noch het lokatie-geheugen, noch het tijdelijke stuk extra geheugen is op het scherm te zien.

☞ 4.2. Wat is de uitvoer van dit commando?

```
VERVEELVOUDIG -3 TOT 10
```

We kunnen van elk programma dat we schrijven een commando-definitie maken. Zo'n programma kan dan met heel weinig moeite steeds opnieuw gebruikt worden.

Als we een programma nodig hebben om het meest rechtse cijfer van het getal in `a` af te hakken, en dit cijfer in de lokatie `rechts` te bewaren, zullen we gewoonlijk een programmaatje als dit schrijven. (Zoals we eerder gezien hebben berekent `mod` de rest na deling.)

```
PUT a mod 10 IN rechts
PUT (a-rechts)/10 IN a
```

We kunnen dat programma de vorm geven van een commando-definitie:

Dan kunnen we hetzelfde resultaat als boven krijgen door te typen:

```
HAK'RECHTS a NAAR rechts
```

Verder kunnen we dit nieuwe eigen commando elke keer gebruiken als we het meest rechtse cijfer van een getal apart willen zetten, ook in de definitie van een ander commando, zoals dit voor het vinden van de som van de cijfers van een getal. (In het `WHILE`-commando bevat `som` de som van die cijfers van getal die niet meer in `g` staan.)

```
HOW'TO TEL'CIJFERS'OP getal:
   PUT getal IN g
   PUT 0 IN som
   WHILE g > 0:
      HAK'RECHTS g NAAR rechts
      PUT som+rechts IN som
   WRITE som
```

Als we het nieuwe commando zo gebruiken:

```
TEL'CIJFERS'OP 13*13
```

dan krijgen we als uitvoer:

```
□ 16
```

Dat is correct omdat 13*13=169, en de som van de cijfers van 169 is 16. Om duidelijk te maken wat de computer precies doet om dit antwoord te vinden, is hier het resultaat van vervanging van de voorlopige parameters in de commando-definitie van TEL'CIJFERS'OP door de feitelijke parameters:

```
PUT (13*13) IN g
PUT 0 IN som
WHILE g > 0:
   HAK'RECHTS g NAAR rechts
   PUT som+rechts IN som
WRITE som
```

Om door een nog sterker vergrootglas te zien wat er gebeurt is hier dezelfde tekst, maar nu met het HAK'RECHTS-commando ook uitgeschreven:

```
PUT (13*13) IN g
PUT 0 IN som
WHILE g > 0:
   PUT (g) mod 10 IN (rechts)
   PUT ((g)-(rechts))/10 IN (g)
   PUT som+rechts IN som
WRITE som
```

Deze uitgebreide demonstratie van de werking van eigen commando's is alleen bedoeld als kennismaking met dit mechanisme, niet als aanmoediging om elke toepassing van een eigen commando met de hand uit te schrijven, ook al kan dat af en toe nuttig zijn als de computer onverwachte resultaten geeft.

Regels

- Commando-definities worden opgeborgen in het definitie-geheugen van de computer en blijven daar totdat we de moeite nemen ze weg te gooien, met behulp van speciale voorzieningen die het *B*-systeem biedt. Het is ook steeds mogelijk deze definities te bekijken, te veranderen of een andere naam te geven.

- Afgezien van de eerste regel moet een commando-definitie ingesprongen getypt worden. Dat kan tot dubbel inspringen leiden als er een WHILE-commando in de definitie voorkomt.

- Sleutelwoorden van eigen commando's zien er hetzelfde uit als etiketten van lokaties, maar dan met *hoofdletters* i.p.v. kleine letters. Voorbeelden van correcte sleutelwoorden zijn dus: PLAATS, BIJ, VOEG', GEEF'2DE'GETAL.

- Voorlopige parameters zien er hetzelfde uit als etiketten van lokaties.

- Er mogen in een commando ook meer dan twee parameters voorkomen, steeds van elkaar gescheiden door een sleutelwoord. Een commando kan ook bestaan uit een enkel sleutelwoord, zonder parameter.

- Commando-definities moeten met verschillende sleutelwoorden beginnen. Dus als we deze definitie al hebben:

```
HOW'TO TEL v BIJ a:
   PUT a+v IN a
```

dan kunnen we deze definitie niet toevoegen:

```
✗ HOW'TO TEL a PLUS b:
      WRITE a, "+", b, "=", a+b /
```

- Een feitelijke parameter in een toepassing van een eigen commando kan elke expressie zijn, dus ook een enkele lokatie (zoals p) of een enkel getal (zoals 3.14).

Advies

- Bij het definiëren van een nieuw commando (b.v. voor het schrijven van de som en het produkt van twee getallen), is het verstandig eerst te besluiten welke gegevens we van keer tot keer verschillend willen kunnen opgeven (in dit geval de twee getallen waarvan we som en produkt willen weten). Vervolgens kunnen we, door sleutelwoorden te kiezen, bepalen hoe we willen dat een toepassing van het nieuwe commando eruitziet (b.v. als PLUS'EN'MAAL 17 EN 36). Als dat beslist is, is het gemakkelijk de eerste regel van de definitie op te schrijven (HOW'TO PLUS'EN'MAAL a EN b:).

• Het is verstandig te proberen de stam van een werkwoord als eerste sleutelwoord te kiezen, zodat een toepassing van het commando er werkelijk als een commando uitziet. In het algemeen zijn programma's gemakkelijker te begrijpen als de sleutelwoorden, parameters en etiketten zo gekozen zijn dat ze hun functie aan de menselijke lezer verduidelijken.

☞ 4.3. Geef een beschrijving van de betekenis van deze commando-definitie, en kies betere woorden.

```
HOW'TO EPIBREER woord:
   PUT woord*woord IN woord
```

☞ 4.4. Doe hetzelfde voor:

```
HOW'TO QRZ pppp M ppp:
   PUT ppp, pppp IN pppp, ppp
```

☞ 4.5. Definieer een commando om zowel het kwadraat als de derde macht van een getal te schrijven. Het commando moet zo gebruikt kunnen worden:

```
GEEF'MACHTEN'VAN 9
□ 9 81 729
```

☞ 4.6. Als iemand niet gelukkig is met het PUT-commando van *B*, en liever GEEF a WAARDE 3 schrijft dan PUT 3 IN a, welke definitie moet dan gegeven worden om dat mogelijk te maken.

☞ 4.7. Hier is de definitie van een commando dat laat zien hoe er rente wordt bijgeschreven bij een som geld totdat die verdubbeld is. Verander deze definitie zo, dat het rentepercentage niet vastligt op 15, maar elke keer moet worden opgegeven als het commando gebruikt wordt.

```
HOW'TO VERDUBBEL som:
   PUT 1984, som IN jaar, begin
   WHILE som < 2*begin:
      PUT jaar+1 IN jaar
      PUT 2 round (som*1.15) IN som
      WRITE jaar, som /
```

☞ 4.8. Met behulp van de volgende commando-definitie is het mogelijk de rij
1, 1, 2, 3, 5, 8, 13 ... (13 = 5 + 8)
voort te brengen zolang deze onder de 1000 blijft.
a. Hoe kan dit commando gebruikt worden om de rij
2, 5, 7, 12, 19 ... (19 = 7 + 12)
voort te brengen zolang die onder de 500 blijft?
b. Hoe kan de definitie veranderd worden om ook rijen als
1, 1, 5, 13, 41, 121 ... (121 = 3*13 + 2*41) en
1, 3, 8, 21, 55, 144 ... (144 = −21 + 3*55)
voort te brengen?
c. Geef toepassingen van dat nieuwe commando om elk van de vier genoemde rijen voort te brengen.

```
HOW'TO TEL'OP p EN q TOT grens:
   PUT p, q IN a, b
   WRITE a
   WHILE b < grens:
      WRITE b
      PUT b, a+b IN a, b
```

5. HOW'TO nader bestudeerd

Als we wat nauwkeuriger naar het definiëren en gebruiken van eigen commando's kijken, zien we dat er zich enkele complicaties kunnen voordoen.

5.1. Expressies of lokaties als feitelijke parameters

Hier is een definitie van een commando om de waarde van een lokatie met 1 te verhogen:

```
HOW'TO VERHOOG a:
   PUT a+1 IN a
```

Dit commando kan zo gebruikt worden:

```
PUT 3 IN aantal'keren'tot'nu'toe
VERHOOG aantal'keren'tot'nu'toe
WRITE aantal'keren'tot'nu'toe
□ 4
```

maar niet zo:

```
✗ VERHOOG 3
```

Waarom is dat fout? Je zou kunnen zeggen, omdat het zinloos is de waarde van het getal 3 in 4 te veranderen. Als we preciezer kijken, zien we dat VERHOOG 3 neerkomt op PUT (3)+1 IN (3), hetgeen duidelijk fout is, omdat de tweede parameter van een PUT-commando een lokatie moet zijn.

Een ander voorbeeld van hetzelfde probleem is dit programma dat een beginwaarde blijft verdubbelen zolang die onder een bepaalde grens blijft.

```
HOW'TO VERDUBBEL g TOT grens:
   WHILE g < grens:
      WRITE g
      PUT g+g IN g
```

Dit commando kan zo gebruikt worden:

```
PUT 3 IN p
VERDUBBEL p TOT 100
□ 3 6 12 24 48 96
```

maar niet zo:

```
✗ VERDUBBEL 3 TOT 100
```

Dit is fout omdat vervanging van de voorlopige parameters door de feitelijke parameters tot een fout PUT-commando blijkt te leiden:

```
WHILE (3) < (100):
   WRITE (3)
 ✗ PUT (3)+(3) IN (3)
```

Als we VERDUBBEL 3 TOT 100 toch willen toestaan, moeten we een extra lokatie in de definitie gebruiken. Deze extra lokatie, die alleen een rol speelt binnen het VERDUBBEL-commando, is een *interne* lokatie, in tegenstelling tot de gebruikelijke lokaties, die *extern* zijn. (We hebben al eerder interne lokaties gezien; de laatste was jaar in de definitie van VERDUBBEL in opgave 4.7.)

```
HOW'TO VERDUBBEL g TOT grens:
   PUT g IN a
   WHILE a < grens:
      WRITE a
      PUT a+a IN a
```

☞ 5.1. Hoe ziet bovenstaande definitie eruit als de feitelijke parameters van deze toepassing erin verwerkt zijn?

```
VERDUBBEL 3 TOT 100
```

☞ 5.2. Waarom is het niet mogelijk de definitie van VERHOOG zo te veranderen dat het niet langer verplicht is een lokatie als feitelijke parameter te gebruiken?

Door binnen in een commando-definitie te kijken is het altijd mogelijk vast te stellen of voor een bepaalde voorlopige parameter een *lokatie* als feitelijke parameter moet worden opgegeven of dat *elke expressie* goed is.

☞ 5.3. Stel dit vast voor de parameters van:

```
HOW'TO VERLAAG k ONDER n:
   WHILE k >= n:
      PUT k/2 IN k
   WRITE k
```

5.2. Interne lokaties en externe lokaties

Een van de functies van eigen commando's is dat ze ons in staat stellen programma's begrijpelijker te maken door ze in zinvolle brokken te verdelen en die brokken tot aparte commando's te maken. In de praktijk zijn programma's meestal groter dan de voorbeelden die we gezien hebben. In zulke grote programma's is het moeilijk fouten te vermijden en het is daarom belangrijk programma's zo leesbaar mogelijk te houden. Het voordeel van eigen commando's voor dat doel is dat we de details van zo'n commando maar één keer hoeven te begrijpen, nl. wanneer we de definitie schrijven. Later is het voldoende te onthouden wat de sleutelwoorden van het commando zijn en *wat* het commando doet, niet *hoe* het dat doet. We willen vooral niet de namen van de voorlopige parameters en de etiketten van de interne lokaties hoeven te onthouden.

Neem deze definitie van een commando dat, zij het wat onhandig, de waarden van twee lokaties verwisselt. In deze definitie is `z` een interne lokatie, en zijn `a` en `b` voorlopige parameters.

```
HOW'TO VERWISSEL a EN b:
   PUT a IN z
   PUT b IN a
   PUT z IN b
```

We willen niet dat gebruik van dit commando leidt tot het verknoeien van de externe lokatie `z` die we misschien hebben. We willen dus dat dit werkt:

```
PUT 4 IN z
PUT 8, 5 IN p, q
VERWISSEL p EN q
WRITE p, q, z
□ 5 8 4
```

Ook moeten bij gebruik van het VERWISSEL-commando de waarden van eventuele externe lokaties `a` en `b` onveranderd blijven, tenzij ze als feitelijke parameter worden opgegeven:

```
PUT 3, 1, 5 IN a, b, z
VERWISSEL a EN z
WRITE a, b, z
□ 5 1 3
```

Om ervoor te zorgen dat deze wensen vervuld worden geeft de computer elke keer aan iedere interne lokatie een uniek etiket door er `'`-tekens achter te zetten totdat het verschilt van elk ander etiket. Aan het eind van dit hoofdstuk zullen we een volledige beschrijving van het kopieer- en vervangmechanisme zien.

5.3. Zorgvuldige vervanging

Als we deze definitie hebben van een commando om de waarde van een lokatie te delen door een gegeven getal:

```
HOW'TO DEEL a DOOR b:
   PUT a/b IN a
```

dan mag dat zo gebruikt worden:

```
PUT 12 IN s
DEEL s DOOR 3
WRITE s
□ 4
```

omdat de definitie er na vervanging van de voorlopige parameters zo uitziet:

```
PUT (s)/(3) IN (s)
```

Natuurlijk is het resultaat gelijk als DEEL op deze iets verschillende manier gebruikt wordt:

```
PUT 12 IN s
DEEL s DOOR 2+1
WRITE s
□ 4
```

omdat vervanging tussen haakjes dit oplevert:

```
PUT (s)/(2+1) IN (s)
```

Zou vervanging zonder haakjes gebeuren, dan zouden we een verkeerd resultaat krijgen, omdat `s` dan eerst door `2` gedeeld zou worden, voordat `1` zou worden bijgeteld:

```
PUT s/2+1 IN s
```

In gevallen waar deze haakjes niet nodig zijn, doen ze ook geen kwaad.

5.4. Gezamenlijk gebruikte lokaties

Tot nu toe waren de parameters het enige kanaal om informatie in en uit een commando te krijgen, omdat alle andere etiketten in een commando-definitie interne lokaties aanduiden. Soms is dat hinderlijk.

B.v. als we een lijst als deze willen behandelen:

```
5 flessen wijn van 1.95
2 hammen van 30.50
1 stuk zeep van 1.35
```

Om de totale kosten en het totale aantal stuks te berekenen, kunnen we zo te werk gaan:

```
PUT 0, 0 IN stuks, fl
PUT stuks+5, fl+5*1.95 IN stuks, fl
PUT stuks+2, fl+2*30.50 IN stuks, fl
PUT stuks+1, fl+1.35 IN stuks, fl
WRITE stuks, fl
□ 8 72.10
```

Voor het verwerken van een lange lijst zouden we graag een speciaal commando hebben:

```
HOW'TO TEL ex A prijs BIJ tex EN tp:
   PUT ex*prijs IN p
   PUT tex+ex, tp+p IN tex, tp
```

Maar het gebruiken van dit commando is net zo langdradig als de oorspronkelijke aanpak, doordat er zoveel parameters nodig zijn.

```
PUT 0, 0 IN stuks, fl
TEL 5 A 1.95 BIJ stuks EN fl
TEL 2 A 30.50 BIJ stuks EN fl
TEL 1 A 1.35 BIJ stuks EN fl
WRITE stuks, fl
□ 8 72.10
```

Het is echter mogelijk de voorlopige parameters `tex` en `tp` weg te werken.

Daarvoor is een SHARE-commando nodig (Engels voor „delen", „gezamenlijk gebruiken"), waarin wordt aangegeven dat de *externe* lokaties `stuks` en `fl` steeds gebruikt moeten worden door het commando:

```
HOW'TO TEL ex A prijs:
   SHARE stuks, fl
   PUT ex*prijs IN p
   PUT stuks+ex, fl+p IN stuks, fl
```

Het gebruiken van de nieuwe versie is veel aantrekkelijker:

```
PUT 0, 0 IN stuks, fl
TEL 5 A 1.95
TEL 2 A 30.50
TEL 1 A 1.35
WRITE stuks, fl
□ 8 72.10
```

☞ 5.4. Definieer een commando dat kan helpen bij het optellen van een lijst getallen. Het commando moet steeds als een getal is bijgeteld het subtotaal laten zien. Het moet zo gebruikt kunnen worden:

```
T 45
□ 45
T 13
□ 58
T -4/2
□ 56
```

Regels

- SHARE-commando's moeten direct na de HOW'TO-regel volgen, vóór de andere commando's.
- Hier volgt een volledige beschrijving hoe een eigen commando door de computer uitgevoerd wordt.

Om de werking begrijpelijker te maken neem ik een eenvoudig voorbeeld, waarbij het lokatiegeheugen er aanvankelijk zo uitziet:

```
fl = 9.75  stuks = 5  ex = 4
p = 3  p' = 1
```

Het commando dat we bekijken is deze toepassing van de definitie die we zojuist gezien hebben:

```
TEL 2 A 30.50
```

1. Het geheugen wordt tijdelijk uitgebreid (er wordt een extra stuk papier aan geplakt), en de commando-definitie wordt daar gekopieerd.

```
fl = 9.75  stuks = 5  ex = 4
p = 3  p' = 1

HOW'TO TEL ex A prijs:
   SHARE stuks, fl
   PUT ex*prijs IN p
   PUT stuks+ex, fl+p IN stuks, fl
```

2. Elk etiket van een interne lokatie (d.w.z. elk etiket in de kopie dat geen voorlopige parameter is en ook niet in een SHARE-commando van de definitie voorkomt) wordt uniek gemaakt, d.w.z. systematisch veranderd door toevoeging van zoveel '-tekens aan het eind als nodig om ze verschillend te maken van de andere etiketten.

```
fl = 9.75  stuks = 5  ex = 4
p = 3  p' = 1

HOW'TO TEL ex A prijs:
   SHARE stuks, fl
   PUT ex*prijs IN p''
   PUT stuks+ex, fl+p'' IN stuks, fl
```

3. De feitelijke parameters worden tussen haakjes in de plaats van de voorlopige parameters geschreven.

```
fl = 9.75  stuks = 5  ex = 4
p = 3  p' = 1

HOW'TO TEL (2) A (30.50):
   SHARE stuks, fl
   PUT (2)*(30.50) IN p''
   PUT stuks+(2), fl+p'' IN stuks, fl
```

4. De eerste regel van de kopie, en eventuele SHARE-regels, worden verwijderd.

```
fl = 9.75  stuks = 5  ex = 4
p = 3  p' = 1

PUT (2)*(30.50) IN p''
PUT stuks+(2), fl+p'' IN stuks, fl
```

5. De aangepaste versie wordt nu uitgevoerd, waarbij het extra geheugen gebruikt wordt voor de interne lokaties. We zien dat de externe lokaties p en p' niet verknoeid worden doordat er toevallig ook een *interne* lokatie p in de commando-definitie voorkomt. Evenmin wordt de externe lokatie ex aangetast, ook al heeft die een etiket dat gelijk is aan een van de voorlopige parameters. Ook zien we dat de SHARE-lokaties stuks en fl naar behoren worden veranderd.

```
fl = 70.75  stuks = 7  ex = 4
p = 3  p' = 1

PUT (2)*(30.50) IN p''
PUT stuks+(2), fl+p'' IN stuks, fl

p'' = 71
```

6. Tenslotte wordt het extra stuk geheugen weggegooid.

```
fl = 70.75  stuks = 7  ex = 4
p = 3  p' = 1
```

● Externe lokaties zijn *permanent*, d.w.z. ze blijven bestaan totdat we ze opruimen. (Hoe dat gaat zullen we later zien.) Interne lokaties bestaan daarentegen alleen gedurende de uitvoering van het commando waarin ze voorkomen. Als hetzelfde etiket ook in een andere definitie voorkomt, dan duidt het daar een andere lokatie aan.

☞ 5.5. Gegeven zijn deze definitie:

```
HOW'TO MAAL g:
   SHARE pp, f
   PUT g*f IN p
   WRITE p
   PUT pp*p IN pp
```

en deze geheugen-situatie:

```
g = 4  p = 8  k = 6  pp = 2  f = 1.5
```

a. Voorspel hoe de uitvoer en het geheugen eruitzien nadat het commando MAAL k+1 is uitgevoerd. 10.5 4 8 6 21.0 1.5

b. Ga na of die voorspelling klopt door de boven gegeven stappen te doorlopen.

c. Beschrijf in gewoon Nederlands wat het effect is van een toepassing van het MAAL-commando.

6. Teksten

We hebben het gebruik van teksten al gezien in WRITE-commando's:

```
PUT 3, 4 IN g, d
WRITE g, "gedeeld door", d, "=", g/d
□ 3 gedeeld door 4 = 0.75
```

Met behulp van die mogelijkheid kunnen we een commando definiëren om de tekst Goede morgen te laten schrijven omlijst door een rechthoek van sterretjes. (Dit commmando heeft maar één sleutelwoord en geen parameters; daar is geen bezwaar tegen.)

```
HOW'TO WENS:
   WRITE "****************" /
   WRITE "*              *" /
   WRITE "* Goede morgen *" /
   WRITE "*              *" /
   WRITE "****************"
```

Dit commando kunnen we zo gebruiken:

```
WENS
□ ****************
□ *              *
□ * Goede morgen *
□ *              *
□ ****************
```

Een nieuwe mogelijkheid van teksten is dat ze in lokaties bewaard kunnen worden, net als getallen. Daardoor kunnen we WENS ook zo definiëren:

```
HOW'TO WENS:
   PUT "****************" IN sterren
   PUT "*              *" IN spaties
   WRITE sterren /
   WRITE spaties /
   WRITE "* Goede morgen *" /
   WRITE spaties /
   WRITE sterren
```

Natuurlijk zijn de operatoren voor getallen die we gezien hebben niet geschikt om voor teksten gebruikt te worden, omdat het niet duidelijk is wat een expressie als deze zou moeten betekenen:

```
✗ ("Goede morgen"*"ja") mod 3
```

Maar er zijn speciale operatoren voor teksten. Om teksten aan elkaar te plakken tot één tekst wordt de operator ^ gebruikt.

```
PUT "bas", "alt" IN laag, hoog
WRITE laag^hoog
□ basalt
```

Om een tekst een aantal malen achter elkaar te plakken kan de operator ^^ gebruikt worden:

```
WRITE "Nee!"^^3
□ Nee!Nee!Nee!
```

Om het aantal *karakters* van een tekst te bepalen is er de operator # (spreek uit: „lengte"). Een karakter is een letter of een cijfer of een van de andere tekens waaruit teksten bestaan, waaronder de spatie. (Let op het verschil tussen de tekst "laag" en het etiket laag.)

```
WRITE #laag, #"laag", #"te recht"
□ 3 4 8
WRITE #("Nee!"^^3)
□ 12
```

☞ 6.1. Geef een definitie van het commando ONDERSTREEP dat een gegeven tekst gevolgd door een uitroepteken op het scherm schrijft, met op de volgende regel een rij mintekens van dezelfde lengte. Het commando moet zo gebruikt kunnen worden:

```
ONDERSTREEP "Au"
□ Au!
□ ---
```

☞ 6.2. De lokatie antwoord bevat een tekst van hoogstens 5 karakters. Schrijf een programma om die tekst aan de rechterkant uit te breiden met zoveel uitroeptekens dat hij precies 10 karakters lang wordt.

De nieuwe operatoren kunnen gebruikt worden voor weer een andere versie van het WENS-commando. (Een WRITE-commando geeft geen spatie tussen twee teksten.)

```
HOW'TO WENS:
   PUT "Goede morgen" IN woorden
   PUT "*"^^#woorden IN sterren
   PUT " "^^#woorden IN spaties
   WRITE "**", sterren, "**" /
   WRITE "* ", spaties, " *" /
   WRITE "* ", woorden, " *" /
   WRITE "* ", spaties, " *" /
   WRITE "**", sterren, "**"
```

De nieuwe versie ziet er niet bijzonder aantrekkelijk uit, maar heeft het voordeel dat hij gemakkelijk geschikt gemaakt kan worden voor het inlijsten van *elke* tekst:

```
HOW'TO OMLIJST woorden:
   PUT "*"^^#woorden IN sterren
   PUT " "^^#woorden IN spaties
   WRITE "**", sterren, "**" /
   WRITE "* ", spaties, " *" /
   WRITE "* ", woorden, " *" /
   WRITE "* ", spaties, " *" /
   WRITE "**", sterren, "**"
```

Nu kan deze tekst omlijst worden:

```
OMLIJST "PAS OP!"
□ ***********
□ *         *
□ * PAS OP! *
□ *         *
□ ***********
```

en zelfs de *lege* tekst:

```
OMLIJST ""
□ ****
□ *  *
□ *  *
□ *  *
□ ****
```

☞ 6.3. Gebruik het OMLIJST-commando voor het definiëren van een nieuwe versie van het WENS-commando.

Twee andere operatoren voor teksten zijn | (spreek uit: „op lengte") om het eerste deel van een tekst te krijgen, en @ (spreek uit: „vanaf") om het laatste deel te krijgen:

```
WRITE "kwartslagen"|6
□ kwarts
WRITE "kwartslagen"@6
□ slagen
```

Door die twee operatoren te combineren kunnen we elk deel van een tekst krijgen:

```
WRITE "kwartslagen"@3|4
□ arts
WRITE "kwartslagen"|9@7
□ lag
```

De operatoren | en @ moeten gebruikt worden met gehele getallen die niet te klein of te groot zijn; deze vijf commando's leiden daardoor allemaal tot een foutmelding:

```
✗ WRITE "kwartslagen"|2.5
✗ WRITE "kwartslagen"|14
✗ WRITE "kwartslagen"@14
✗ WRITE "kwartslagen"|-2
✗ WRITE "kwartslagen"@-2
```

Dit zijn de kleinste en de grootste getallen die nog goed gaan. Het eerste en het laatste commando geven een lege tekst als uitvoer.

```
WRITE "kwartslagen"|0
□
WRITE "kwartslagen"|11
□ kwartslagen
WRITE "kwartslagen"@1
□ kwartslagen
WRITE "kwartslagen"@12
□
```

☞ 6.4. De lokatie `t` bevat een niet-lege tekst. Geef commando's om de volgende situaties te bereiken:
a. alleen het eerste karakter staat nog in `t`;
b. alles behalve het eerste karakter staat nog in `t`;
c. alleen het laatste karakter staat nog in `t`;
d. alles behalve het laatste karakter staat nog in `t`;
e. geen van de karakters staat nog in `t`.

De operatoren `<`, `<=`, `=`, `>=`, `>` en `<>`, die we gezien hebben voor getallen, hebben ook betekenis voor teksten. Een tekst is kleiner dan een andere als hij eerder zou komen in een alfabetisch woordenboek, dus `"nee" < "neen" < "s"`. De lege tekst `""` is kleiner dan elke andere tekst.

☞ 6.5. Definieer een commando HAK dat alle letters `e` verwijdert die aan het begin en het eind van een tekst in een lokatie staan. Neem aan dat die tekst niet leeg is. HAK moet dit effect hebben:

```
PUT "eerdere" IN woord
HAK woord
WRITE woord
□ rder
```

☞ 6.6. Hier is een programma om alle uitgangen van het woord `lego` te laten zien. Maak van dit programma een definitie van een commando ELK'EINDE waarmee hetzelfde met *elke* tekst gedaan kan worden.

```
PUT "lego" IN w
WHILE w > "":
   WRITE w /
   PUT w@2 IN w
□ lego
□ ego
□ go
□ o
```

☞ 6.7. Definieer een soortgelijk commando ELK'BEGIN.

☞ 6.8. Definieer een commando SCHUIF1 dat het volgende kan doen met elke tekst van tenminste twee karakters.

```
PUT "steken" IN w
SCHUIF1 w
WRITE w
□ tekens
```

☞ 6.9. Definieer een commando SCHUIF dat hetzelfde herhaaldelijk doet en het resultaat van elke verschuiving laat zien:

```
SCHUIF "ene"
□ ene
□ nee
□ een
SCHUIF "inging"
□ inging
□ ngingi
□ gingin
```

☞ 6.10. Palindromen zijn woorden die achterstevoren hetzelfde zijn, zoals `kook` en `lepel`. Definieer een commando dat helpt bij het vaststellen of een woord een palindroom is. Het commando moet laten zien wat er overblijft nadat gelijke letters aan het begin en het eind van het woord paarsgewijze zijn verwijderd. Het moet dus zo werken:

```
TOON'REST "negeren"
□ De rest is: ger
TOON'REST "lepel"
□ De rest is: p
TOON'REST "kook"
□ De rest is:
```

Regels

• Om van twee verschillende teksten vast te stellen welke de kleinste is (in de zin van `<`), kijkt de computer eerst of een van de teksten korter is *en* gelijk aan het begin van de andere tekst. Zo ja, dan is de kortste tekst ook de kleinste (b.v. `"een" < "eender"`). Anders vergelijkt de computer het eerste karakter van beide teksten, dan het tweede, enzovoort. Het eerste verschillende karakter bepaalt welke tekst de kleinste is (dus `"eender" < "eendracht"` omdat de `e` vóór de `r` komt in het alfabet). Omdat niet alle karakters in teksten letters zijn, moeten we ook de alfabetische volgorde van de andere karakters weten.

De alfabetische volgorde van alle karakters die een tekst kan bevatten staat in deze lijst. Het eerste karakter is de spatie, die gemakkelijk over het hoofd gezien wordt. Het is niet erg zinvol deze lijst uit het hoofd te leren, maar we zien dat de cijfers in hun gebruikelijke volgorde staan, net als de kleine letters en de hoofdletters.

```
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
```

● De gebruikelijke spatie die het WRITE-commando tussen twee stukken uitvoer voegt blijft achterwege tussen twee teksten.

☞ 6.11. Wat is de volgorde (in de zin van <) van deze zeven teksten:

```
nr1   nr2   nr12   NRS   nrs   "#!<   ~@#?
```

7. Types

We hebben tot nu toe twee types waarden gebruikt in onze programma's: getallen en teksten. Later zullen we meer types tegenkomen, maar dit is een goed moment om eens te kijken naar de gevolgen van het bestaan van meer dan één type waarde. Allereerst heeft ieder type zijn eigen operatoren, die niet voor andere types gebruikt kunnen worden. Zo werkt + allen voor twee getallen, ^ voor twee teksten en ^^ voor één tekst en één getal.

De computer zou het volgende commando dus niet kunnen uitvoeren, en zou onmiddellijk de fout melden dat + verkeerd gebruikt wordt:

```
✗ WRITE "Ho"+"og"
```

Als we zo'n fout maken in de definitie van een eigen commando, dan zou het prettig zijn meteen gewaarschuwd te worden op het moment dat we de fout maken, i.p.v. later als de computer het nieuwe commando ook *uitvoert.*

```
HOW'TO VERLENG getal MET cijfer:
 ✗ PUT (10*getal)^cijfer IN getal
```

Gelukkig waarschuwt de computer ons inderdaad onmiddellijk. Zodra we klaar zijn met het typen van de foute regel `PUT (10*getal)^cijfer IN getal`, waarschuwt de computer ons dat we proberen ^ te gebruiken voor getallen. Dit heeft het voordeel dat we de fout meteen kunnen corrigeren, en niet later naar dit probleem hoeven terug te keren, als we misschien al half vergeten zijn waar die regel voor bedoeld was.

Een ander soort fout dat de computer snel bemerkt is het gebruiken van een lokatie voor een ander type waarde dan tevoren.

Dat maakt het mogelijk het per ongeluk gebruiken van de verkeerde lokatie onmiddellijk te verbeteren, zoals in deze definitie van een commando dat telt met hoeveel gelijke karakters een tekst begint.

```
HOW'TO TEL'GELIJKE tekst:
   PUT tekst|1, tekst IN e, rest
   PUT 0 IN t
   WHILE rest|1 = e:
    ✗ PUT e+1, rest@2 IN e, rest
   WRITE t, "letters ", e
```

Zodra we `PUT e+1, rest@2 IN e, rest` getypt hebben, meldt de computer ons dat we nu ten onrechte de lokatie `e` voor een getal gebruiken, terwijl we `e` in de vorige regel nog voor een tekst gebruikten. Natuurlijk had de definitie zo moeten luiden:

```
HOW'TO TEL'GELIJKE tekst:
   PUT tekst|1, tekst IN e, rest
   PUT 0 IN t
   WHILE rest|1 = e:
      PUT t+1, rest@2 IN t, rest
   WRITE t, "letters ", e
```

Soms willen we een lokatie misschien met opzet voor een ander type waarde gebruiken. We zouden b.v. op deze resultaten kunnen hopen:

```
56789 3224990521 183143986697069
PUT 56789 IN a
WRITE a, a*a, a*a*a
☐ 56789 3224990521 183143986697069
✗ PUT "ten" IN a
WRITE a, a^a, a^a^a
☐ ten tenten tententen
```

Maar ook hier is het niet toegestaan de lokatie `a` eerst voor het bewaren van een getal te gebruiken en later voor een tekst. Dat betekent dat we een ander etiket moeten kiezen voor het bewaren van `"ten"`.

De regel dat een lokatie alleen waarden kan bevatten van hetzelfde type als de eerste waarde die erin gestopt werd, geldt dus zowel voor interne als voor externe lokaties. Dat zou betekenen dat, als we de externe lokatie `a` eenmaal een getal als waarde hebben gegeven, we `a` nooit meer voor een tekst zullen kunnen gebruiken! Om ons van zo'n eeuwigdurende verplichting te ontslaan, en ook om de voortdurende groei van het aantal in gebruik zijnde lokaties te beperken, kunnen we het commando `DELETE` (Engels voor „vernietig") gebruiken om een lokatie op te ruimen, zowel de waarde als het etiket.

Als het lokatie-geheugen er zo uitziet:

```
a = 46   w = "kook"   p = "ja"   nr = 3
```

dan kunnen de lokaties w en nr verwijderd worden met het commando:

```
DELETE w, nr
```

Als gevolg hiervan zal het geheugen er zo uitzien:

```
a = 46   p = "ja"
```

Het is een goede gewoonte de externe lokaties die geen nut meer hebben op te ruimen, omdat het dan gemakkelijker is de functie van de overige lokaties te onthouden die wel nog zin hebben.

De regel dat een lokatie maar voor één type waarde gebruikt kan worden, verbiedt ons niet dit commando om de waarde van twee lokaties te verwisselen nu eens te gebruiken voor getallen en dan weer voor teksten:

```
HOW'TO WISSEL a EN b:
   PUT a, b IN b, a
   WRITE a /
   WRITE b /

PUT 1, 2, "j", "n" IN p, q, pos, neg
WISSEL p EN q
□ 2
□ 1
WISSEL pos EN neg
□ n
□ j
```

We kunnen WISSEL natuurlijk niet gebruiken om de waarde van een getal-lokatie en een tekst-lokatie te verwisselen:

```
✗ WISSEL pos EN q
```

Het probleem is dat dat commando leidt tot dit commando dat duidelijk tegen de regels is:

```
✗ PUT pos, q IN q, pos
```

Regels

- Een bestaande externe lokatie mag alleen een nieuwe waarde gegeven worden van hetzelfde type als de vorige waarde.

- Elke keer dat een eigen commando gebruikt wordt moeten alle waarden die aan een bepaalde interne lokatie gegeven worden van hetzelfde type zijn. De computer controleert dat als we de definitie geven.

☞ 7.1. Bepaal van alle voorlopige parameters in de volgende commando-definities of er een feitelijke parameter bij hoort van het type getal of het type tekst, of dat allebei goed zijn. Bepaal ook of de feitelijke parameter een lokatie moet zijn, of elke expressie van het goede type mag zijn.

```
HOW'TO TEL l IN t UITKOMST f:
   PUT 0 IN f
   WHILE t > "" AND t|1 = l:
      PUT t@2 IN t
      PUT f+1 IN f

HOW'TO OMGEEF woord MET teken:
   WRITE teken, woord, teken

HOW'TO YAF p SO q AH r DU s:
   PUT r+1, s, p, q IN p, q, r, s
```

8. Invoer: het READ-commando

In een vorig hoofdstuk hebben we deze definitie gezien van een commando dat ons helpt een lijst getallen op te tellen doordat het na ieder nieuw getal een subtotaal laat zien:

```
HOW'TO T n:
   SHARE totaal
   PUT totaal+n IN totaal
   WRITE totaal
```

Dat commando kan zo gebruikt worden:

```
PUT 0 IN totaal
T 20.80
□ 20.8
T 7.75
□ 28.55
T 9/2
□ 33.05
```

Hoewel het SHARE-commando in de definitie van het T-commando ons ontslaat van de plicht elke keer de externe lokatie totaal op te geven, en hoewel de naam T van het commando toch moeilijk korter kan, willen we misschien nog minder hoeven in te toetsen, nl. alleen de getallen die opgeteld moeten worden.

Dat is mogelijk als we READ-commando's in de definitie gebruiken. (READ is Engels voor „lees", EG voor „b.v.".)

```
HOW'TO TEL'IN t:
   PUT 0 IN t
   READ g EG 0
   WHILE g <> 0:
      PUT t+g IN t
      WRITE t /
      READ g EG 0
```

Als de computer een commando uitvoert als TEL'IN totaal, dan geeft hij eerst totaal de waarde 0. Dan komt hij bij het commando READ g EG 0 en waarschuwt de gebruiker aan het scherm door een ? te schrijven, en wacht tot de gebruiker een getal heeft ingetoetst (of een expressie waar een getal uitkomt). Zodra dat gebeurd is geeft de computer aan de lokatie g dat getal als waarde en gaat verder met het volgende commando. Het stuk EG 0 van dit READ-commando geeft aan dat de computer moet controleren of er een expressie van het type getal ingetoetst wordt, niet van het type tekst.

Dus als we dit commando geven:

```
TEL'IN totaal
```

dan gebeurt er dit op het scherm. Het enige dat we zelf hoeven te typen zijn de op te tellen getallen. Omdat in de WHILE-regel van de definitie de conditie g <> 0 staat, stopt het proces zodra we 0 typen als *invoer*, d.w.z. als antwoord op het vraagteken. Ik heb het getal 0 gekozen als teken dat ik wil ophouden, omdat het onzin is 0 te willen bijtellen.

```
□ ?
250
□ 250
□ ?
5*5
□ 275
□ ?
-15
□ 260
□ ?
0
```

☞ 8.1. Elke keer dat we het TEL'IN-commando gebruiken begint dat opnieuw vanaf 0 te tellen. Soms willen we dat niet. We willen b.v. een lokatie uitgaven bijhouden door er elke dag de uitgaven van die dag in bij te tellen. In de tussentijd willen we de computer voor andere dingen kunnen gebruiken. Hoe zou de definitie van TEL'IN veranderd kunnen worden om dat mogelijk te maken? Hoe moet het nieuwe commando gebruikt worden?

Hier is nog een voorbeeld van een commando-definitie met een READ-commando. Dit commando TOON'RESTEN blijft de gebruiker om woorden vragen om vast te stellen of het palindromen zijn. We maken natuurlijk gebruik van het TOON'REST-commando uit opgave 6.11. Omdat hier teksten gelezen moeten worden i.p.v. getallen, gebruiken we hier EG "" i.p.v. EG 0 in de READ-commando's.

```
HOW'TO TOON'RESTEN:
   READ w EG ""
   WHILE w <> "stop":
      TOON'REST w
      WRITE /
      READ w EG ""
```

Hier is een voorbeeld van een toepassing van het commando TOON'RESTEN. Nu gebruiken we de tekst "stop" om aan te geven dat we willen ophouden (stop is duidelijk geen palindroom). We zien dat de invoer naast letterlijke teksten ook expressies als "tet"^^3 mag bevatten.

```
TOON'RESTEN
□ ?
"negeren"
□ De rest is: ger
□ ?
"kook"
□ De rest is:
□ ?
"tet"^^3
□ De rest is: e
□ ?
"stop"
```

Het is vaak vervelend twee keer " te moeten typen bij elke invoer-tekst. In zulke gevallen kunnen we het sleutelwoord RAW (Engels voor „rauw") in de definitie gebruiken i.p.v. EG "". Dat betekent dat de invoer-regel, als tekst tussen aanhalingstekens geplaatst, de waarde moet worden van de lokatie van het READ-commando.

```
HOW'TO TOON'RESTEN:
   READ w RAW
   WHILE w <> "stop":
      TOON'REST w
      WRITE /
      READ w RAW
```

Als we invoer typen voor een READ-commando kunnen we geen expressies als "tet"^^3 gebruiken, omdat die zou worden opgevat als de letterlijke tekst "tet"^^3:

```
TOON'RESTEN
□ ?
negeren
□ De rest is: ger
□ ?
"tet"^^3
□ De rest is: "tet"^^3
□ ?
"stop"
□ De rest is: stop
□ ?
stop
```

Hier volgt een programma waarin de meeste mogelijkheden van *B* voorkomen die we tot nu toe gezien hebben. Het is de definitie van een commando waarmee een woordraadsel gespeeld kan worden. De speler krijgt steeds de beurt om een geheim woord te raden. Bij iedere beurt laat de computer het woord zien zover als het geraden is, plus nog een extra letter. Het programma is misschien wat ingewikkelder dan de lezer nu zelf zou kunnen schrijven. In feite zou het programma overzichtelijker geschreven kunnen worden met behulp van constructies die in Deel 2 aan bod zullen komen. Het is niet zo interessant dit spelletje zelf te spelen, omdat men zelf het geheime woord in het programma moet zetten, maar iemand anders wil misschien eens proberen het woord in zo weinig mogelijk beurten te raden.

```
HOW'TO WOORDRAADSEL:
   PUT "enigma" IN x
   WRITE "Probeer het geheime woord te raden." /
   PUT 1, 1 IN beurt, bekend
   WRITE "Het begint met: ", x|bekend /
   READ poging RAW
   WHILE #poging < #x OR poging|#x <> x:
      PUT beurt+1, bekend+1 IN beurt, bekend
      WHILE #poging >= bekend AND #x >= bekend AND poging|bekend = x|bekend:
         PUT bekend+1 IN bekend
      WRITE "Het begint met: ", x|bekend /
      READ poging RAW
   WRITE "Het woord was: ", x /
   WRITE beurt, "beurten om", #x-1, "letters te raden" /
```

In de bovenstaande definitie stelt bekend het aantal letters voor dat aan de speler getoond kan worden; beurt bevat het aantal keren dat de speler geraden heeft en poging is het laatste woord dat de speler geprobeerd heeft. Een spelletje WOORDRAADSEL zou er zo kunnen uitzien:

```
WOORDRAADSEL
□ Probeer het geheime woord te raden.
□ Het begint met: e
□ ?
evolutie
□ Het begint met: en
□ ?
enigszins
□ Het begint met: enigm
□ ?
enigmatisch
□ Het woord was: enigma
□ (3 beurten om 5 letters te raden)
```

Regels

- Elke keer als de gebruiker aan een READ-commando invoer geeft van het verkeerde type, vraagt de computer alsnog om het goede type invoer.

- Als een commando waarin een READ-commando voorkomt om invoer vraagt, kan het proces gestopt worden met de stop-toets.

Advies

- In de programma-voorbeelden in dit hoofdstuk is veel zorg besteed aan het netjes eindigen van programma's. Hoewel de stop-toets in geval van nood altijd gebruikt kan worden, is het beter als commando's netjes eindigen wegens de mogelijkheid dat we zo'n commando later in de definitie van een ander commando willen gebruiken. Het zou dan vervelend zijn als een van de commando's in die definitie niet gewoon zou eindigen, omdat dan de rest van de definitie nooit aan de beurt zou komen.

☞ 8.2. Definieer een commando dat de gebruiker om de drie afmetingen van een doos vraagt, en dat dan de uitkomst van enkele berekeningen laat zien:

```
DOOS
□ Geef 3 afmetingen in cm.
□ ?
2
□ ?
5
□ ?
10
□ Zij-oppervlakken:    10 20 50 cm2
□ Totaal oppervlak:    160 cm2
□ Volume:              100 cm3
□ Oppervlak per cm3:   1.6 cm2
```

9. Voorwaardelijke uitvoering: het IF- en het SELECT-commando

Een oppervlakkig programma om het verkleinwoord van Nederlandse zelfstandige naamwoorden to produceren is het volgende. (Denk eraan dat WRITE geen spatie plaatst tussen twee teksten.)

```
HOW'TO VERKLEIN woord:
   WRITE woord, "je"
```

Het VERKLEIN-commando geeft in vele gevallen het goede resultaat:

```
VERKLEIN "plank"
□ plankje
VERKLEIN "kist"
□ kistje
```

Het gaat b.v. mis als het woord op een r eindigt:

```
VERKLEIN "pilaar"
□ pilaarje
VERKLEIN "pijler"
□ pijlerje
```

Om het commando ook voor zulke woorden goed te laten werken zou het prettig zijn als we in staat zouden zijn alleen een t te laten schrijven als het woord op een r eindigt, en daarna in alle gevallen met je te laten doorgaan. Het IF-commando (Engels voor „indien", „als") is daarvoor bedoeld:

```
HOW'TO VERKLEIN woord:
   WRITE woord
   IF woord@#woord = "r":
      WRITE "t"
   WRITE "je"
```

De meeste woorden die op een r eindigen zullen nu ook goed gaan:

```
VERKLEIN "uur"
□ uurtje
```

Het uiterlijk en de werking van een IF-commando lijken veel op die van een WHILE-commando, maar bij een IF-commando wordt het ingesprongen deel hoogstens één keer uitgevoerd. Een IF-commando kunnen we gebruiken als we een groep van één of meer commando's alleen willen laten uitvoeren als een bepaalde conditie geldt.

Soms willen we de computer uit twee of meer mogelijkheden laten kiezen. Als we het VERKLEIN-commando ook goed willen laten werken voor woorden die op een m eindigen, zoals bodem en droom, dan willen we de computer uit drie verschillende behandelingen er één laten uitzoeken, afhankelijk van de laatste letter van het zelfstandig naamwoord.

Dat kunnen we doen door gebruik te maken van het SELECT-commando (Engels voor „uitzoeken"):

```
HOW'TO VERKLEIN woord:
   SELECT:
      woord@#woord = "r":
         WRITE woord, "tje"
      woord@#woord = "m":
         WRITE woord, "pje"
      ELSE:
         WRITE woord, "je"
```

Een SELECT-commando bevat gewoonlijk twee of meer *alternatieven*, elk ingesprongen. Een alternatief begint met een conditie en een :, gevolgd door een aantal (verder) ingesprongen commando's. Als de computer bij een SELECT-commando komt, kijkt hij naar de condities die erop volgen, te beginnen bij de eerste. Zodra hij een conditie tegenkomt die klopt, voert hij de ingesprongen commando's uit die erop volgen. De verdere alternatieven van het SELECT-commando worden dan overgeslagen. In het laatste alternatief mogen we ELSE (Engels voor „anders") gebruiken i.p.v. een conditie. Dat betekent dat de computer dit laatste alternatief moet kiezen als geen van de voorgaande condities klopte. Als we geen ELSE gebruiken, moeten we ervoor zorgen dat minstens één van de condities klopt. Anders zal de computer een foutmelding geven.

Hier is een versie van het VERKLEIN-commando die een aantal andere speciale gevallen ook herkent. (Later zullen we een handige manier zien om ook woorden als ster en kan correct te behandelen.)

```
HOW'TO VERKLEIN w:
   PUT w@#w IN z
   SELECT:
      z = "n" OR z = "r" OR z = "l":
         WRITE w, "tje"
      z = "m":
         WRITE w, "pje"
      z = "e" OR z = "i" OR z = "u":
         WRITE w, "tje"
      #w > 5 AND w@(#w-2) = "ing":
         WRITE w|(#w-1), "kje"
      #w > 2 AND w@(#w-2) = "ing":
         WRITE w, "etje"
      ELSE:
         WRITE w, "je"
```

☞ 9.1. Zou het resultaat van het VERKLEIN-commando verschillend kunnen zijn als we de eerste twee alternatieven van het SELECT-commando zouden verwisselen (zowel de conditie als het bijbehorende commando)? En als we het vierde en het vijfde alternatief zouden verwisselen?

☞ 9.2. Verander het VERKLEIN-commando zo dat het ook:
a. woorden op j (b.v. rij) en w (b.v. duw) de uitgang tje geeft;
b. bij woorden op een a of o de eindletter verdubbelt voor de uitgang tje.

```
VERKLEIN "vrouw"
□ vrouwtje
VERKLEIN "la"
□ laatje
```

We moeten dus een IF-commando gebruiken als er een groep commando's is die soms wel en soms niet uitgevoerd moet worden. Een SELECT-commando wordt gebruikt als van een aantal groepen commando's er precies één moet worden uitgevoerd. Voor sommige problemen kunnen we de oplossing op beide manieren formuleren.

Als we een commando willen om de waarden van zijn twee parameters in volgorde van hun grootte op het scherm te schrijven, kunnen we IF gebruiken om ze te verwisselen als ze aanvankelijk in de verkeerde volgorde staan:

```
HOW'TO SCHRIJF'OP a EN b:
   PUT a, b IN p, q
   IF p > q:
      PUT p, q IN q, p
   WRITE p, q /
```

Met SELECT kunnen we deze nette oplossing schrijven:

```
HOW'TO SCHRIJF'OP a EN b:
   SELECT:
      a < b:
         WRITE a, b /
      a >= b:
         WRITE b, a /
```

of met ELSE in het laatste alternatief:

```
HOW'TO SCHRIJF'OP a EN b:
   SELECT:
      a < b:
         WRITE a, b /
      ELSE:
         WRITE b, a /
```

☞ 9.3. Wat zal er gebeuren met deze vijf toepassingen van SCHRIJF'OP?

```
PUT 5, 3 IN s, t
PUT "zaal", "sport" IN x, y
SCHRIJF'OP s EN t
SCHRIJF'OP 3 EN 5
SCHRIJF'OP x EN t
SCHRIJF'OP x EN y
SCHRIJF'OP "sport" EN "zaal"
```

☞ 9.4. Verander het TOON'REST-commando uit opgave 6.10 zo, dat het op deze manier antwoordt:

```
TOON'REST "negeren"
☐ Nee, negeren is geen palindroom.
TOON'REST "lepel"
☐ Ja, lepel is een palindroom.
```

☞ 9.5. Schrijf de volgende variatie op het WOORDRAADSEL-commando uit hoofdstuk 8. De speler wordt uitgenodigd een geheim getal te raden. Bij iedere foute gissing maakt de computer bekend of die te laag is of te hoog. De conversatie moet ongeveer zo verlopen:

```
GETALRAADSEL
☐ Probeer het geheime getal te raden.
☐ ?
100
☐ lager
☐ ?
30
☐ hoger
☐ ?
70
☐ lager
☐ ?
50
☐ lager
☐ ?
40
☐ hoger
☐ ?
44
☐ Goed!
```

Regels

- Het SELECT-commando wordt gebruikt om uit een aantal alternatieven er één te kiezen. Het IF-commando wordt gebruikt als er maar één alternatief is, dat al of niet moet worden uitgevoerd.

- Minstens één van de alternatieven van een SELECT-commando moet opgaan. Als er een ELSE voorkomt, moet het bij het laatste alternatief zijn.

- Soms is het nodig een loos alternatief toe te voegen aan een SELECT-commando om aan de regel te voldoen dat minstens één van de alternatieven klopt:

```
HOW'TO TURF g:
   SHARE pos, neg, apos, aneg
   SELECT:
      g < 0:
         PUT neg+g IN neg
         PUT aneg+1 IN aneg
      g = 0:
         \nul hoeft niet geteld
      g > 0:
         PUT pos+g IN pos
         PUT apos+1 IN apos
```

In dit geval bevat het tweede alternatief geen commando's. In plaats daarvan heb ik er een *commentaar* geplaatst dat de situatie verduidelijkt voor de menselijke lezer.

- Commentaar is te herkennen aan het teken \. Alles dat in een regel volgt na een \ wordt door de computer genegeerd.

10. Lijsten

Tot nu toe hebben we alleen programma's gezien die maar weinig in het geheugen van de computer opborgen. Met de tot nu toe behandelde middelen kunnen we een programma alleen met veel gegevens laten werken als het programma ook veel lokaties bevat, elk met zijn eigen etiket. Willen we de computer elk palindroom laten onthouden dat we hem gegeven hebben, dan moeten we voor elk palindroom een aparte lokatie gebruiken. Om die palindromen dan op het scherm te laten schrijven zouden we weer een programma nodig hebben dat dat (groeiende) aantal lokaties zou gebruiken. In plaats van zoveel verschillende lokaties te gebruiken, kunnen we gebruik maken van één lokatie van het nieuwe type *lijst*. Een lijst is een verzameling van elementen, b.v. teksten.

Hier is een commando om een lijst van twee woorden in een lokatie op te bergen:

```
PUT {"tot"; "kaak"} IN dromen
```

Als gevolg van dit commando bevat de lokatie dromen één waarde van het nieuwe type lijst. De lijst bevat twee *elementen* (de teksten "kaak" en "tot"), en wel in alfabetische volgorde:

```
dromen = {"kaak"; "tot"}
```

Wat kunnen we nu zoal doen met dit nieuwe type van waarden en lokaties? Ten eerste kunnen we er alles mee doen dat ook met de beide andere typen (getallen en teksten) kan, zoals PUT en WRITE.

Hier wordt de waarde van de lokatie dromen gekopieerd in de lokatie d, en wordt de waarde van d vervolgens op het scherm geschreven:

```
PUT dromen IN d
WRITE d
□ {"kaak"; "tot"}
```

Als we een eigen commando gebruiken met voorlopige parameters die van elk type mogen zijn, dan kunnen we die dus voortaan ook met feitelijke parameters van type lijst gebruiken.

```
PUT {"de"; "het"; "een"} IN woordjes
WISSEL d EN woordjes
WRITE d
□ {"de"; "een"; "het"}
```

Voor lijsten van opeenvolgende gehele getallen of karakters kunnen we een verkorte notatie gebruiken met .. (spreek uit: „tot en met"):

```
PUT {"n".."q"} IN letters
WRITE letters
□ {"n"; "o"; "p"; "q"}
PUT {2*2..3*3} IN cijfers
WRITE cijfers
□ {4; 5; 6; 7; 8; 9}
```

Zoals er speciale operatoren zijn voor getallen en teksten, zo zij er ook speciale operatoren *en* commando's voor lijsten.

Als we een nieuw woord aan de lijst dromen willen toevoegen, kunnen we gebruik maken van het speciale commando INSERT (Engels voor „tussenvoegen"):

```
INSERT "pop" IN dromen
```

Als we nu de nieuwe waarde van dromen op het scherm laten schrijven, zien we dat de volgorde van de elementen weer alfabetisch is:

```
WRITE dromen
□ {"kaak"; "pop"; "tot"}
```

De lijst blijft gesorteerd als we met INSERT nieuwe woorden toevoegen. (Denk eraan dat hoofdletters alfabetisch vóór kleine letters komen.)

```
INSERT "U" IN dromen
WRITE dromen
□ {"U"; "kaak"; "pop"; "tot"}
```

Met getallen zien we hetzelfde gebeuren:

```
WRITE {8; 3*5; 4; 1; 2*5}
□ {1; 4; 8; 10; 15}
```

Het is ook toegestaan een element toe te voegen dat al in de lijst voorkomt:

```
INSERT "pop" IN dromen
WRITE dromen
□ {"U"; "kaak"; "pop"; "pop"; "tot"}
```

We kunnen ook een element uit een lijst verwijderen met REMOVE (Engels voor „verwijderen"). Als dat element meer dan één keer in de lijst voorkomt, wordt er maar één element verwijderd.

```
REMOVE "pop" FROM dromen
WRITE dromen
□ {"U"; "kaak"; "pop"; "tot"}
```

Het is niet toegestaan te proberen een element te verwijderen dat niet voorkomt in de lijst. Dit zou dus tot een foutmelding leiden:

```
✗ REMOVE "moorddroom" FROM dromen
```

Er is een manier om er achter te komen of een bepaald element in een lijst voorkomt:

```
IF "moorddroom" in dromen:
   REMOVE "moorddroom" FROM dromen
```

De operator `in` wordt gebruikt om condities op te bouwen, net als < en =. De operator `not'in` geeft het tegengestelde resultaat, zoals te zien is in dit commando om een element alleen toe te voegen als het nog niet voorkomt:

```
HOW'TO INSERT1 element IN lijst:
   IF element not'in lijst:
      INSERT element IN lijst
```

☞ 10.1. Definieer een commando HAAL'ELKE dat een element zonodig meermalen uit een lijst verwijdert, zodat het na afloop niet meer voorkomt, hoe vaak het aanvankelijk ook voorkwam. Het commando moet op de volgende manier gebruikt kunnen worden. (De uitvoer {} geeft een lege lijst aan.)

```
PUT {5; 5; 8} IN getallen
HAAL'ELKE 5 UIT getallen
WRITE getallen
□ {8}
HAAL'ELKE 4 UIT getallen
WRITE getallen
□ {8}
HAAL'ELKE 8 UIT getallen
WRITE getallen
□ {}
```

☞ 10.2. Bedenk een kortere schrijfwijze voor de conditie in dit programma:

```
IF z = "n" OR z = "r" OR z = "l":
   WRITE woord, "tje"
```

De operatoren `in` en `not'in` kunnen ook gebruikt worden om te bepalen of een bepaald karakter in een tekst voorkomt:

```
PUT "ieoau" IN klinkers
IF letter not'in klinkers:
   WRITE letter, " is geen klinker"
```

Hier volgen nog enkele operatoren die zowel voor lijsten als voor teksten gebruikt kunnen worden.

De operator #, die we voor teksten gebruikt hebben om het aantal karakters te weten te komen, kan ook gebruikt worden om het aantal elementen van een lijst te achterhalen:

```
WRITE #{8; 4; 7}
□ 3
PUT 7 IN n
WRITE #{2..n}
□ 6
```

Er is ook een versie van # met twee operanden, om vast te stellen hoeveel keer een bepaald element in een lijst voorkomt:

```
PUT {"de"; "het"; "een"} IN woordjes
WRITE "het"#woordjes
□ 1
WRITE "dat"#woordjes
□ 0
WRITE 2#{1; 2; 2; 2; 3; 4; 4}
□ 3
```

Als we # gebruiken met een tekst als rechter operand, dan moet het linker operand één enkel karakter zijn:

```
PUT "mississippi" IN rivier
WRITE "i"#rivier
□ 4
```

Het kleinste element van een lijst kunnen we bepalen met de operator `min`:

```
WRITE min {1..7}
□ 1
WRITE min {"kou"; "kauw"; "kouw"}
□ kauw
```

Als we `min` gebruiken voor een tekst, dan krijgen we het alfabetisch eerste karakter:

```
WRITE min "pirouette"
□ e
```

Wat `max` doet is gemakkelijk te raden:

```
PUT {13; 5; 8} IN getallen
WRITE min getallen, max getallen
□ 5 13
```

Om het vierde karakter van een tekst te krijgen, of het derde element van een lijst kunnen we gebruik maken van de operator `th'of` (Engels voor „-ste van", „-de van"):

```
WRITE 4 th'of rivier
□ s
WRITE 3 th'of woordjes
□ het
```

☞ 10.3. Geef een andere schrijfwijze voor:

```
a. WRITE 1 th'of dromen
b. WRITE 1 th'of rivier
c. WRITE #dromen th'of dromen
```

☞ 10.4. Wat zal de uitvoer zijn van deze commando's?

```
a. WRITE 2 th'of {"zeven"; "acht"}
b. WRITE 2 th'of "zeven acht"
c. WRITE 2 th'of 78
d. WRITE "twee" th'of {"zeven"; "acht"}
```

Regels

• De operatoren `<` enz. werken voor elk type, ook voor lijsten. Om na te gaan welke van twee lijsten de kleinste is (in de zin van `<`) kijkt de computer eerst welke lijst met het kleinste element begint. Als beide lijsten met hetzelfde element beginnen, kijkt de computer naar het tweede element van beide lijsten, enz.

```
{1; 2; 9} < {1; 3; 4} < {2}
```

Als een lijst gelijk is aan het eerste stuk van een andere lijst, dan is die andere lijst groter in de zin van `>`:

```
{2; 3; 5} < {2; 3; 5; 8}
{} < {-1}
```

• Alle elementen van een lijst moeten van hetzelfde type zijn. Dat betekent dat dit fout is:

```
PUT {2..5} IN getallen
✗ INSERT "zes" IN getallen
```

• We kunnen *elk* type kiezen voor de elementen van een lijst. De elementen van een lijst mogen dus zelf ook weer lijsten zijn:

```
PUT {} IN s3
INSERT {1; 2} IN s3
INSERT {1; 1; 1} IN s3
INSERT {3} IN s3
WRITE s3
□ {{1; 1; 1}; {1; 2}; {3}}
WRITE 2 th'of s3
□ {1; 2}
```

Er bestaan in feite vele verschillende types lijst. Zo heeft de lijst `{"s"; "z"; "x"}` hetzelfde type als `{"ja"; "nee"}` (lijst van teksten), maar `{9; 11}` heeft een ander type (lijst van getallen). Dat betekent weer dat deze verschillende types lijst niet als elementen van één lijst mogen voorkomen.

Dit is dus fout:

```
✗ PUT {{0; 0}; {"een"; "een"}} IN nn
```

• De operatoren `in`, `not'in`, `#`, `min`, `max` en `th'of` kunnen zowel voor teksten als voor lijsten gebruikt worden. Zij doen hetzelfde met de *karakters* van een tekst als met de *elementen* van een lijst. De commando's `INSERT` en `REMOVE` werken alleen voor lijsten.

• De operatoren `min`, `max` en `th'of` mogen (natuurlijk) niet gebruikt worden voor de lege lijst `{}` of de lege tekst `""`.

• Evenzo moet `th'of` gebruikt worden met gehele getallen van de juiste grootte, zodat deze toepassingen van `th'of` fout zijn:

```
✗ PUT 2.5 th'of {2; 5; 7; 7; 9} IN s
✗ WRITE 3 th'of {"ja"; "nee"}
```

☞ 10.5. Wat is het resultaat van de volgende commando's?

```
a. WRITE min {"3"; "22"; "111"}
b. WRITE min {{}}
c. WRITE min min {{5; 2}; {5; 2; 1}}
d. WRITE #{}
```

☞ 10.6. De lokaties `a`, `b` en `c` bevatten elk een verschillend getal. Geef een commando dat het middelste van die drie getallen als uitvoer geeft, d.w.z. niet het grootste en niet het kleinste.

☞ 10.7. Definieer een commando dat de gebruiker regels tekst als invoer laat intoetsen totdat eenzelfde regel voor de tweede keer wordt ingevoerd. Het commando moet er dan voor zorgen dat het aantal verschillende regels als uitvoer wordt gegeven.

```
VIND'DUBBELE
☐ ?
ja
☐ ?
best
☐ ?
inderdaad
☐ ?
zeker
☐ ?
wel degelijk
☐ ?
best
☐ Dat waren 5 verschillende regels.
```

☞ 10.8. Als `k` een kleine letter bevat, hoe kan dan de waarde van `h` beschreven worden na afloop van dit commando:

```
PUT (#{"a"..k}) th'of {"A".."Z"} IN h
```

11. Een andere vorm van herhaling: het FOR-commando

Bij het werken met lijsten willen we vaak alle elementen van een lijst op dezelfde manier behandelen, b.v. ze op aparte regels op het scherm laten zien.

Het is mogelijk om dat met een WHILE-commando te doen:

```
HOW'TO TOON'LIJST l:
   PUT l IN lijst
   WHILE lijst > {}:
      PUT min lijst IN element
      REMOVE element FROM lijst
      WRITE element /
```

Omdat het zo vaak voorkomt dat we een lijst element voor element willen afgaan, is er een speciaal commando voor:

```
PUT {"U"; "netten"; "pap"} IN dromen
FOR w IN dromen:
   WRITE w, " is een palindroom." /
□ U is een palindroom.
□ netten is een palindroom.
□ pap is een palindroom.
```

Het is gemakkelijk te bepalen hoeveel herhalingen dit FOR-commando zal opleveren: precies evenveel als er elementen in de lijst dromen voorkomen. De eerste keer dat het ingesprongen stuk wordt uitgevoerd bevat de lokatie w het eerste (= kleinste) element van de lijst; de tweede keer bevat w het tweede element, enzovoort.

In de volgende commando-definitie wordt een FOR-commando gebruikt om alle elementen van een lijst toe te voegen aan een andere lijst:

```
HOW'TO VOEG a BIJ b:
   FOR element IN a:
      INSERT element IN b
```

Dit VOEG-commando kan zo gebruikt worden:

```
PUT {-3..3} IN getallen
VOEG {5; 2} BIJ getallen
WRITE getallen
□ {-3; -2; -1; 0; 1; 2; 2; 3; 5}
```

☞ 11.1. Herschrijf de boven gegeven definitie van TOON'LIJST met behulp van een FOR-commando.

☞ 11.2. De lokaties a, b, c en d bevatten elk een getal. Gebruik het feit dat de elementen van een lijst gesorteerd worden bewaard om de waarden van deze vier lokaties in stijgende volgorde op het scherm te laten verschijnen.

Hier volgt een programma dat de lengte bepaalt van de langste tekst in de lijst dromen. De centrale lokatie langste'tot'nu bevat de lengte van de langste tekst woord die tot nu toe gezien is, de huidige waarde van woord niet meegerekend.

```
PUT 0 IN langste'tot'nu
FOR woord IN dromen:
   IF #woord > langste'tot'nu:
      PUT #woord IN langste'tot'nu
WRITE "Langste was:", langste'tot'nu
□ Langste was: 6
```

☞ 11.3. Schrijf een programma dat de totale lengte berekent en als uitvoer geeft van alle teksten die element zijn van de lijst dromen. Formuleer de rol van de centrale lokaties die in het programma voorkomen.

☞ 11.4. Schrijf een commando FAHRENHEIT dat laat zien hoeveel graden Celsius overeenkomen met een temperatuur in graden Fahrenheit die als parameter wordt opgegeven. De formule is te vinden in hoofdstuk 1. Laat de uitvoer er zo uitzien:

```
FAHRENHEIT 451
□ 451 F = 232.78 C
```

☞ 11.5. Gebruik het commando FAHRENHEIT om een commando te definiëren dat zo gebruikt kan worden:

```
ELKE'FAHRENHEIT {40; 45}
□ 40 F = 4.44 C
□ 45 F = 7.22 C
ELKE'FAHRENHEIT {40..45}
□ 40 F = 4.44 C
□ 41 F = 5.00 C
□ 42 F = 5.56 C
□ 43 F = 6.11 C
□ 44 F = 6.67 C
□ 45 F = 7.22 C
```

FOR-commando's zijn geschikt om een tevoren vaststaand aantal herhalingen te laten uitvoeren, ook als er overigens helemaal geen lijst in het spel is. Hier is een manier om de tekst ja op vier regels steeds vaker te laten zien:

```
FOR regel IN {1..4}:
   WRITE "ja"^^regel /
□ ja
□ jaja
□ jajaja
□ jajajaja
```

Dezelfde methode kunnen we altijd gebruiken als we een lokatie nodig hebben waarvan de waarde bij elke herhaling 1 groter wordt.

In situaties waarin we een lokatie willen waarvan de waarde bij iedere herhaling 1 kleiner wordt, kunnen we de volgende methode gebruiken:

```
FOR v IN {44..51}:
   WRITE 44+51-v
□ 51 50 49 48 47 46 45 44
```

☞ 11.6. Definieer een commando dat een pijlvormige figuur vertoont die bestaat uit sterretjes. In de volgende toepassing van dat commando bestaat de langste regel uit 4 keer 3 sterretjes:

```
PIJL 4
□ ***
□ ******
□ *********
□ ************
□ *********
□ ******
□ ***
```

Regels

• Als de computer een FOR-commando gaat uitvoeren berekent hij eerst de lijst die in de FOR-regel voorkomt, en bergt het resultaat op in een tijdelijk stuk extra geheugen, dat hij weer weggooit zodra het gehele FOR-commando afgelopen is. Deze kopie gebruikt de computer om te bepalen welke serie waarden hij aan de lokatie in de FOR-regel moet geven. Die lokatie wordt zelf ook in dat tijdelijke stuk geheugen bewaard.

Daardoor kan het volgende programma gebruikt worden om een extra uitroepteken te plaatsen achter die teksten in de lijst ww die al op een uitroepteken eindigen.

```
PUT {"ho!"; "hoe?"; "hoera!!"} IN ww
FOR woord IN ww:
   IF woord@#woord = "!":
      REMOVE woord FROM ww
      INSERT woord^"!" IN ww
WRITE ww
□ {"ho!!"; "hoe?"; "hoera!!!"}
```

• Een FOR-commando kan ook gebruikt worden om de karakters van een tekst te doorlopen. De karakters woorden dan doorlopen in de volgorde waarin ze in de tekst staan, *niet* in alfabetische volgorde.

```
FOR kar IN "nee":
   WRITE kar /
□ n
□ e
□ e
```

- Als de lijst of de tekst in een FOR-regel leeg is wordt het ingesprongen deel helemaal niet uitgevoerd. Dus deze commando's geven geen uitvoer:

```
PUT {""} IN teksten
FOR c IN min teksten:
   WRITE "Hallo!" /
REMOVE max teksten FROM teksten
FOR tekst IN teksten:
   WRITE "Hoera!" /
```

Advies

- Zowel het FOR- als het WHILE-commando zijn herhalingscommando's. Een FOR-commando is het handigst in gevallen waarin het aantal herhalingen van te voren bekend is; in andere gevallen is het beter een WHILE-commando te gebruiken. In geval van twijfel is FOR beter, omdat het dan zeker is dat er een eind komt aan de herhaling.

- Net als bij WHILE-commando's is het bij FOR-commando's van belang de rol van de centrale lokaties in de gaten te houden. In beide soorten herhaling is het ook belangrijk bij het programmeren niet de eerste herhaling in gedachten te nemen, maar een moment ergens halverwege het proces.

☞ 11.7. Definieer een commando KEER'OM dat de tekst-parameter die we eraan opgeven omgekeerd als uitvoer geeft. Gebruik een FOR-commando.

```
KEER'OM "droom"
□ moord
```

☞ 11.8. Schrijf een programma dat alle getallen onder de 100 laat zien die een veelvoud van 11 of van 13 zijn. Die getallen moeten op volgorde van klein naar groot op het scherm verschijnen.

12. Tabellen

We hebben gezien hoe een lijst gebruikt kan worden voor het opbergen van een verzameling elementen, b.v. Nederlandse zelfstandige naamwoorden. Helaas zijn lijsten niet geschikt voor het opslaan van iets als een telefoonlijst in het geheugen van de computer, waarin we het telefoonnummer zouden kunnen laten opzoeken dat bij een bepaalde naam hoort.

Of we zouden misschien een woordenlijst willen opbouwen waarin een aantal zelfstandige naamwoorden bewaard worden gekoppeld aan hun meervoud, om zo het bepalen van het meervoud van die woorden gemakkelijk te maken. Dat kunnen we doen door gebruik te maken van het vierde type van *B*: een tabel.

```
PUT {["are"]: "aren"} IN mv
```

Hierdoor wordt een nieuwe lokatie `mv` van het type tabel in het leven geroepen. De waarde van `mv` wordt zo in het geheugen bewaard:

```
mv = {["are"]: "aren"}
```

Als we nu dit commando geven:

```
WRITE mv["are"]
```

krijgen we als resultaat:

```
□ aren
```

We kunnen dit korte tabelletje uitbreiden:

```
PUT "azen" IN mv["aas"]
```

Als we nu de nieuwe waarde van `mv` op het scherm laten schrijven, zien we:

```
WRITE mv
□ {["aas"]: "azen"; ["are"]: "aren"}
```

De enkelvouden `"aas"` en `"are"` zijn de *sleutels* van de tabel; `"azen"` en `"aren"` zijn de *ladingen* van de tabel. Zo'n tweetal bestaande uit een sleutel en een lading heet een *post* van de tabel.

We kunnen nieuwe posten aan de tabel toevoegen:

```
PUT "eieren" IN mv["ei"]
PUT "bladen" IN mv["blad"]
```

We kunnen een van de ladingen veranderen:

```
PUT "bladeren" IN mv["blad"]
WRITE mv["blad"]
□ bladeren
```

Een tabel is te beschouwen als een verzameling lokaties, waarbij iedere post als lokatie fungeert: `mv["are"]`, `mv["ei"]`, enz. Deze bijzondere lokaties kunnen op dezelfde manieren gebruikt worden als normale lokaties:

```
PUT mv["are"] IN mv["aar"]
DELETE mv["are"]
```

Natuurlijk is `mv` als geheel ook een lokatie:

```
PUT mv IN meervoud
DELETE mv
WRITE meervoud["ei"]
□ eieren
```

☞ 12.1. Geef een commando om een post aan de tabel `meervoud` toe te voegen voor het woord `datum`, zodat het juiste meervoud `data` wordt opgeborgen. Na afloop moet dit werken:

```
WRITE meervoud["datum"]
□ data
```

De computer geeft een foutmelding als we proberen een post te gebruiken met een sleutel die niet in de tabel voorkomt:

```
✗ WRITE meervoud["fellah"]
```

Om te weten te komen of een bepaalde sleutel in een tabel voorkomt kunnen we gebruik maken van de speciale operator keys (Engels voor „sleutels"). Deze operator levert als resultaat de lijst van sleutels van een tabel:

```
WRITE keys meervoud
□ {"aar"; "aas"; "blad"; "ei"}
IF "fellah" in keys meervoud:
   WRITE meervoud["fellah"]
```

☞ 12.2. Definieer een commando TWEE dat het meervoud geeft van een opgegeven woord door er, als het niet in de tabel meervoud voorkomt, en achter te plaatsen, en anders het meervoud uit de tabel te gebruiken. (Aanwijzing: gebruik SHARE meervoud.)

Hier is een programma dat een tabel gebruikt om te tellen hoe vaak afzonderlijke karakters voorkomen in een lijst teksten. De {} in het tweede commando is de lege tabel; het ziet er hetzelfde uit als een lege lijst. Dit commando is nodig omdat het niet is toegestaan met PUT een lading te zetten in een post van een nog niet bestaande tabel.

```
PUT {"ja"; "nee"; "misschien"} IN tt
PUT {} IN frekw
FOR woord IN tt:
   FOR k IN woord:
      SELECT:
         k not'in keys frekw:
            PUT 1 IN frekw[k]
         ELSE:
            PUT frekw[k]+1 IN frekw[k]
FOR k IN keys frekw:
   WRITE k, frekw[k] /
□ a 1
□ c 1
□ e 3
□ h 1
□ i 2
□ j 1
□ m 1
□ n 2
□ s 2
```

Als we weten dat de teksten b.v. alleen kleine letters bevatten, kunnen we ook de volgende aanpak volgen. Op deze manier zullen de kleine letters die niet in de teksten voorkomen toch als uitvoer gegeven worden, gevolgd door een nul.

```
PUT {"ja"; "nee"; "misschien"} IN tt
PUT {} IN frekw
FOR k IN {"a".."z"}:
   PUT 0 IN frekw[k]
FOR woord IN tt:
   FOR k IN woord:
      PUT frekw[k]+1 IN frekw[k]
FOR k IN keys frekw:
   WRITE k, frekw[k] /
```

In beide voorgaande programma's kan de rol van de posten van frekw zo beschreven worden: voor alle karakters k die tot nu toe gezien zijn bevat frekw[k] het aantal keren dat het karakter k tot nu toe is voorgekomen. Anders gezegd bevat frekw in de eerste versie het aantal keren dat elk karakter gezien is *als* het tenminste erbij was. In de tweede versie bevat frekw het aantal keren dat elk van de letters van het alfabet gezien is, dus ook nullen voor de letters die (nog) niet gezien zijn.

Regels

- Alle sleutels van een tabel moeten van hetzelfde type zijn. Alle ladingen van een tabel moeten ook van hetzelfde type zijn, maar dat hoeft niet hetzelfde te zijn als de sleutels.

- Met DELETE kan een post uit een tabel geschrapt worden:

```
DELETE frekw["x"]
```

Schrappen kan niet gedaan worden met REMOVE, omdat REMOVE alleen gebruikt kan worden met een lijst-lokatie, hetgeen noch keys frekw noch frekw is. Dit is dus fout:

```
✗ REMOVE "x" FROM keys frekw
```

Een gehele tabel kan natuurlijk ook met DELETE weggegooid worden:

```
DELETE frekw
```

- Voordat met PUT iets in een post van een tabel gezet mag worden, moet de gehele tabel eerst een waarde gekregen hebben, hoe klein dan ook:

```
PUT {} IN frekw
PUT {["oorlog"]: "oorlogen"} IN mv
```

- De posten van een tabel worden bewaard en op het scherm geschreven in de volgorde van de sleutels.

- De operatoren `in`, `not'in`, `#`, `min`, `max` en `th'of` kunnen ook gebruikt worden voor tabellen. Wat deze operatoren doen met de *karakters* van een tekst en de *elementen* van een lijst doen ze ook met de *ladingen* van een tabel.

Dus als in het geheugen staat:

```
t = {[-3]: 9; [1]: 1; [2]: 4; [3]: 9}
```

dan zijn deze condities allemaal waar:

```
4 in t
5 not'in t
2 not'in t
9#t = 2
min t = 1
3 th'of t = 4
```

- Een FOR-commando kan ook voor tabellen gebruikt worden. Zoals FOR de karakters van een tekst of de elementen van een lijst van begin naar eind doorloopt, zo kan FOR ook gebruikt worden om de *ladingen* van een tabel te doorlopen. Dus met dezelfde tabel `t` krijgen we deze uitvoer:

```
FOR n IN t:
   WRITE n
□ 9 1 4 9
```

Advies

- Lijsten en tabellen zijn allebei geschikt om met verzamelingen van gegevens te werken. In het algemeen is het het beste een lijst te kiezen als de volgorde van de gegevens er niet toe doet, of als ze in de gebruikelijke volgorde moeten staan die in lijsten toch al gebruikt wordt. Een tabel is het beste als ieder gegeven eigenlijk een verband tussen iets (b.v. iemands naam) en iets anders (b.v. een telefoonnummer) is.

- Ook in gevallen waarin we een verzameling gegevens in een speciale volgorde willen bewaren kunnen we een tabel gebruiken.

In deze tabel b.v. worden drie namen bewaard in een andere volgorde dan de gebruikelijke alfabetische:

```
{[1]: "Jo"; [2]: "Jan"; [5]: "Ans"}
```

- Het is gemakkelijk om de lading te vinden die bij een bepaalde sleutel hoort, maar andersom is niet zo gemakkelijk. Daarom is het belangrijk goed uit te kiezen wat de sleutels van een tabel zullen zijn en wat de ladingen.

Als we b.v. vaak een telefoonnummer zullen willen zoeken dat bij een bepaalde naam hoort, doen we er verstandig aan de namen als sleutels van een tabel te kiezen, en de nummers als ladingen:

```
tel = {["Ans"]: 4367; ["Leo"]: 4141}
```

Maar als we het gewone telefoonboek voor dit doel gebruiken willen we misschien de computer gebruiken voor het omgekeerde: een naam vinden die bij een bepaald telefoonnummer hoort. In dat geval zullen we de nummers als sleutels kiezen, en de namen als ladingen:

```
naam = {[4141]: "Leo"; [4367]: "Ans"}
```

• In sommige gevallen lijkt het nodig om een lijst of een tabel te gebruiken, maar kan het programma eenvoudiger zonder geschreven worden. Neem b.v. dit programma om inches om te rekenen in centimeters:

```
PUT {} IN cms
FOR i IN {1..12}:
   PUT i*2.54 IN cms[i]
FOR i IN {1..12}:
   WRITE i, "inches=", cms[i], "cm" /
```

In dit geval kunnen we de resultaten heel goed op het scherm laten schrijven zonder ze eerst in een tabel op te slaan:

```
FOR i IN {1..12}:
   WRITE i, "inches=", i*2.54, "cm" /
```

• Hoewel mooie etiketten voor lijst-lokaties meestal meervouden zijn, of andere woorden voor verzamelingen:

```
FOR letter IN medeklinkers:
   REMOVE letter FROM alfabet
INSERT woord IN woorden
```

is het meestal beter een enkelvoud te kiezen als etiket voor een tabel:

```
WRITE adres["Marie"]
IF hoofdstad[land] = "Amsterdam":
   WRITE bevolking[land]
```

☞ 12.3. Hier is een wat onhandige manier om te weten te komen hoe laat het op een bepaald moment in enkele wereldsteden is. Neem eens aan dat we een tabel `verschil` hebben met tijdsverschillen, b.v. `verschil["New York"] = -6`. Hoe kan die tabel gebruikt worden voor het maken van een eenvoudigere versie van het commando `TIJD`?

```
HOW'TO TIJD uur:
   WRITE "Amsterdam:", uur, "uur" /
   WRITE "Cairo:", uur+1, "uur" /
   WRITE "London:", uur-1, "uur" /
   WRITE "Madrid:", uur, "uur" /
   WRITE "New York:", uur-6, "uur" /
   WRITE "Sydney:", uur+9, "uur" /
   WRITE "Tokyo:", uur+8, "uur" /
```

☞ 12.4. Definieer een commando `LEES'RIJ` dat een rij ingevoerde woorden leest en die in een tabel zet in de volgorde waarin ze ingevoerd werden. In dit voorbeeld wordt na het vierde vraagteken een lege tekst als invoer gegeven.

```
LEES'RIJ vrienden
☐ ?
Jan
☐ ?
Bert
☐ ?
Joop
☐ ?

FOR naam IN vrienden:
   WRITE naam /
☐ Bert
☐ Jan
☐ Joop
```

☞ 12.5. Voor het vullen van een tabel met teksten als sleutels en als ladingen zou het prettig zijn als we minder zouden hoeven in te toetsen dan:

```
PUT {} IN hoofdstad
PUT "Brussel" IN hoofdstad["Belgie"]
PUT "La Paz" IN hoofdstad["Bolivia"]
PUT "Tokyo" IN hoofdstad["Japan"]
```

Definieer een commando VUL'TABEL dat op de volgende manier gebruikt kan worden voor het vullen van zo'n tabel (als antwoord op het laatste vraagteken wordt hier een lege regel gegeven):

```
VUL'TABEL hoofdstad
□ sleutel:
□ ?
Belgie
□ lading:
□ ?
Brussel
□ sleutel:
□ ?
Bolivia
□ lading:
□ ?
La Paz
□ sleutel:
□ ?
Japan
□ lading:
□ ?
Tokyo
□ sleutel:
□ ?
```

☞ 12.6. Stel dat we een tabel tel hebben waarin net als in het eerder gegeven voorbeeld telefoonnummers staan. Definieer een commando KEER dat gebruikt kan worden om de omgekeerde tabel naam uit hetzelfde voorbeeld te krijgen. Ga ervan uit dat alle telefoonnummers verschillend zijn. Dit zou dus moeten werken:

```
WRITE tel
□ {["Ab"]: 4367; ["Ans"]: 4223}
KEER tel OM'IN naam
WRITE naam
□ {[4223]: "Ans"; [4367]: "Ab"}
```

☞ 12.7. Doe nu hetzelfde voor het geval dat alle namen wel verschillend zijn, maar dat meer mensen eenzelfde telefoonnummer kunnen hebben. Het nieuwe commando KEER moet een tabel namen kunnen afleveren met een *lijst* namen als lading bij ieder telefoonnummer:

```
WRITE tel
□ {["Na"]: 74; ["Pa"]: 78}
PUT 78 IN tel["Ma"]
KEER tel OM'IN namen
WRITE namen
□ {[74]: {"Na"}; [78]: {"Ma"; "Pa"}}
WRITE namen[78]
□ {"Ma"; "Pa"}
```

13. Willekeurige keuze: het CHOOSE-commando

Tot nu toe moesten we, als we een woord of een getal wilden raden, zelf eerst dat woord of getal aan de computer opgeven. Het zou leuker zijn als de computer zelf een of ander woord of getal zou kunnen kiezen. Dat kunnen we hem laten doen met een CHOOSE-commando (Engels voor „kiezen").

Nadat dit commando uitgevoerd is:

```
CHOOSE g FROM {1..100}
```

zal de lokatie g een van de positieve gehele getallen tot en met 100 bevatten, maar we weten van te voren niet welke van de 100 mogelijkheden de computer zal kiezen.

Het CHOOSE-commando kan ook voor een tekst gebruikt worden. Op deze manier kunnen we een willekeurige letter uit de tekst mississippi kiezen:

```
CHOOSE letter FROM "mississippi"
```

Het CHOOSE-commando geeft alle mogelijke keuzen een gelijke kans, zodat de kans dat nu een i gekozen is vier keer zo groot is als de kans dat een m gekozen is.

Het is niet verrassend dat CHOOSE ook gebruikt kan worden om een willekeurige lading van een tabel te kiezen. Als tel nog steeds de waarde heeft uit opgave 12.7, dan is dit een manier om er een willekeurig telefoonnummer uit te kiezen:

```
CHOOSE nr FROM tel
WRITE nr
□ 78
```

Hier is een voorbeeld waarin het CHOOSE-commando gebruikt wordt om een tekst op te bouwen uit karakters die willekeurig uit een kleine verzameling worden gekozen:

```
HOW'TO VLOEK n:
   FOR i IN {1..n}:
      CHOOSE teken FROM "@#$%&*!?"
      WRITE teken
```

☞ 13.1. Gebruik het VLOEK-commando om 5 vloeken op het scherm te laten zien, waarbij de lengte van elke vloek willekeurig is gekozen uit 3, 5 en 8. De vloeken moeten op aparte regels worden getoond, zoals in dit voorbeeld:

```
□ ?*&
□ #?*&!
□ @*#
□ *!#@!!!@
□ $?*#*
```

Regels

- Het CHOOSE-commando werkt op de karakters van een tekst, de elementen van een lijst of de ladingen van een tabel.
- Het CHOOSE-commando geeft gelijke kansen aan alle mogelijkheden waaruit gekozen wordt.
- Het CHOOSE-commando verwijdert het gekozen ding *niet* uit de betreffende tekst, lijst of tabel.

☞ 13.2. Geef een programmaatje dat ofwel ja op het scherm schrijft ofwel nee, waarbij ja tweemaal zoveel kans moet hebben om gekozen te worden als nee.

☞ 13.3. Geef een definitie van een commando PLUK dat hetzelfde met lijsten doet als het CHOOSE-commando, maar dat het gekozen element ook verwijdert:

```
PUT {1..6} IN s
PLUK keus UIT s
WRITE keus
□ 3
WRITE s
□ {1; 2; 4; 5; 6}
```

☞ 13.4. Bedenk een commando dat de elementen van een lijst in willekeurige volgorde op het scherm schrijft. (Aanwijzing: gebruik het PLUK-commando.) De uitvoer zou er zo uit kunnen zien:

```
SCHUD {"a".."e"}
□ e
□ b
□ c
□ a
□ d
```

☞ 13.5. Wat moet er veranderd worden in de definitie van het commando GETALRAADSEL uit opgave 9.5 om de computer zelf te laten bepalen welk getal er geraden moet worden.

☞ 13.6. Definieer een commando GOOI dat een worp met een opgegeven aantal dobbelstenen imiteert. (Let op het gebruik van een extra sleutelwoord aan het eind van deze voorbeelden van het GOOI-commando, waar geen parameter meer op volgt. Dat mag, als dat sleutelwoord in de definitie ook zo voorkomt.)

```
GOOI 1 STENEN
☐ 4
GOOI 3 STENEN
☐ 13
```

☞ 13.7. Ga na hoe willekeurig het CHOOSE-commando werkt door te tellen hoe vaak een worp met een dobbelsteen een 1 oplevert, hoe vaak een 2, enzovoort. Laat de computer 120 worpen doen, en dan het resultaat van de telling tonen. (Aanwijzing: gebruik een tabel.)

☞ 13.8. Laat een tabel hoofdstad gevuld zijn op de manier van opgave 12.5. Definieer een commando dat iemands kennis van hoofdsteden op de proef stelt. Laat het commando steeds een willekeurig land uitkiezen, maar zonder een land nog eens te nemen waar al de goede hoofdstad voor geantwoord is. Als de lokatie hoofdstad alleen de sleutels Belgie, Bolivia en Japan bevat, dan zou het vraag- en antwoordspel er zo uit kunnen zien:

```
OVERHOOR hoofdstad
☐ Japan
☐ ?
Kyoto
☐ Nee, Tokyo
☐ Belgie
☐ ?
Brussel
☐ Goed
☐ Japan
☐ ?
Tokyo
☐ Goed
☐ Bolivia
☐ ?
La Paz
☐ Goed
☐ Dat was alles.
```

Het OVERHOOR-commando moet natuurlijk ook gebruikt kunnen worden voor het overhoren van andere tabellen.

14. Pakketten

Het laatste type van *B* hebben we al regelmatig gebruikt: het *pakket*.

Elke keer dat we iets schreven met een komma erin, gebruikten we eigenlijk een pakket:

```
WRITE n, "is deelbaar door", d
PUT a, b IN c, d
```

Omdat pakketten een echt type van de taal vormen is het ook mogelijk ze met PUT in een lokatie te zetten:

```
PUT "Jan", 3 IN naam1
```

Als gevolg van dit commando komt er in het geheugen te staan:

```
naam1 = ("Jan", 3)
```

In dit voorbeeld zijn "Jan" en 3 de *delen* van het pakket.
In tegenstelling tot andere types zijn er voor pakketten geen speciale operatoren of commando's. Ook mogen operatoren als min en th'of, die wel voor teksten, lijsten en tabellen werken, niet voor pakketten gebruikt worden. De enige dingen die we niet met andere types, maar wel met pakketten doen zijn:

1. Verscheidene waarden in één lokatie samenpakken:

```
PUT "Jan", #"Jan" IN naam1
PUT -p, p IN b
PUT 1984, "april", 4 IN datum
```

2. Een pakket-lokatie uitpakken in verscheidene lokaties:

```
PUT naam1 IN naam, lengte
PUT b IN minus, plus
PUT datum IN jaar, maand, dag
```

3. De uitkomsten van verscheidene expressies in evenveel lokaties plaatsen:

```
PUT a+1, b+1, c+1 IN b, c, a
```

Er zijn in feite vele verschillende pakket-types, zoals er ook verschillende lijst-types zijn (lijsten van getallen, lijsten van teksten, enz.) en verschillende tabel-types (tabel met getal-sleutels en tekst-ladingen, of andersom, enz.).

De volgende pakketten zijn allemaal van een verschillend type:

```
(13, 15)
(10, 12, 17)
("a", 1)
(1, "a")
```

Het volgende is dan ook fout, omdat de types door elkaar gehaald worden:

```
PUT 1900, "januari", 1 IN datum1
PUT "december", 31, 1899 IN datum2
✗ PUT datum1 IN datum2
```

Er is geen manier om het aantal delen van een pakket-lokatie te veranderen. (De enige oplossing is de lokatie eerst met DELETE op te ruimen. Daarna mag hetzelfde etiket voor een lokatie van elk type gebruikt worden.)

Net als voor elk ander type mogen de operatoren < enz. ook gebruikt worden voor twee pakketten *van hetzelfde type*.

Om te bepalen welk van twee pakketten het kleinste is, wordt dezelfde procedure gevolgd als bij teksten en lijsten: het hangt af van het eerste verschillende deel. Dus deze condities zijn allemaal waar:

```
(0, 5) < (1, 0)
(4, "Jan") < (6, "Albert")
(1, 0, 0) < (1, 0, 1)
```

Hoewel de elementen van een lijst bewaard worden in volgorde van klein naar groot, is het, door handig gebruik te maken van pakketten, mogelijk andere volgordes te gebruiken.

Als we bijvoorbeeld een lijst woorden hebben die we niet in de gebruikelijke alfabetische volgorde op het scherm willen krijgen, maar in de volgorde van hun lengte, kunnen we deze methode gebruiken:

```
PUT {} IN paar'lijst
FOR woord IN woorden:
   INSERT #woord, woord IN paar'lijst
FOR paar IN paar'lijst:
   PUT paar IN lengte, woord
   WRITE woord /
```

☞ 14.1. Gebruik dezelfde methode om een lijst getallen op het scherm te laten zetten in volgorde van groot naar klein. (Aanwijzing: als a < b, dan is −a > −b.)

Pakketten kunnen ook gebruikt worden als sleutels van een tabel. Neem bijvoorbeeld dit tweedimensionale schema waarin de verkoopcijfers van drie verkopers in vier jaren staat aangegeven:

	Ab	Jo	Wim
81	65	68	88
82	67	74	96
83	61	74	90
84	65	71	72

Als we pakketten als sleutels gebruiken kunnen we deze tabel als volgt vastleggen:

```
PUT {} IN verkoop
PUT 65 IN verkoop[81, "Ab"]
PUT 68 IN verkoop[81, "Jo"]
PUT 88 IN verkoop[81, "Wim"]
PUT 67 IN verkoop[82, "Ab"]
etc.
```

Om de totale verkoop in 1982 te vinden kunnen we nu schrijven:

```
PUT 0 IN t82
FOR naam IN {"Ab"; "Jo"; "Wim"}:
   PUT t82+verkoop[82, naam] IN t82
```

Als we de totale verkoop van Jo te beginnen in 1981 willen weten kunnen we dit doen:

```
PUT 0 IN t'jo
FOR j IN {81..84}:
   PUT t'jo+verkoop[j, "Jo"] IN t'jo
```

Op de volgende manier kunnen we het totaal van alle verkoopcijfers berekenen:

```
PUT 0 IN som
FOR sleutel IN keys verkoop:
   PUT som+verkoop[sleutel] IN som
```

Een andere manier om dezelfde verkoopcijfers vast te leggen is in een tabel waarvan de ladingen ook weer tabellen zijn:

```
PUT {} IN v
PUT {[81]: 65; [82]: 67} IN v["Ab"]
PUT {[81]: 68; [82]: 74} IN v["Jo"]}
etc.
```

Het berekenen van het totaal van alle verkoopcijfers is dan iets ingewikkelder:

```
PUT 0 IN som
FOR naam IN keys v:
   FOR j IN {81..82}:
      PUT som+v[naam][j] IN som
```

Aan de andere kant is het berekenen van het totaal voor een jaar of voor een verkoper nu gemakkelijker:

```
PUT 0 IN t82
FOR naam IN keys v:
   PUT t82+v[naam][82] IN t82
PUT 0 IN t'jo
FOR j IN keys v["Jo"]:
   PUT t'jo+v["Jo"][j] IN t'jo
```

☞ 14.2. Bedenk een handige manier om een tabel `t` te maken met 25 posten, waarvan de ladingen allemaal `0` zijn en waarvan de sleutels 25 verschillende pakketten zijn met elk twee getal-delen. Die delen moeten variëren van `-2` tot `2`, dus moeten o.a. de lokaties `t[1, -2]` en `t[0, 0]` de waarde `0` krijgen.

☞ 14.3. Schrijf een programma om te bepalen hoe vaak het in een serie worpen met een dobbelsteen voorkomt dat b.v. een 3 gevolgd wordt door een 5. Aan het eind moet een tabel `frekw` zodanig gevuld zijn dat b.v. `frekw[3, 5]` het aantal keren bevat dat een worp van 3 gevolgd is door een worp van 5, enz. Dus als de serie worpen was: 5 3 6 4 3 6 4 4 3, dan moet na afloop gelden dat `frekw[3, 6] = 2`, omdat het 2 keer is voorgekomen dat een 3 door een 6 gevolgd werd.. Laat het experiment voortduren totdat er 120 paren zijn geregistreerd.

Regels

- Er bestaan geen pakketten met maar één deel, omdat voor dat doel gewone lokaties en expressies gebruikt kunnen worden.

Uitwerkingen van de opgaven

1. Rekenen: het WRITE-commando

1.1.
Omdat er van elke ribbe vier gelijke aan een doos te vinden zijn, is dit de goede formule:

```
WRITE 4*(8+12.5+1.75)
□ 89.00
```

1.2.
Het getal 50 moet beide keren veranderd worden in 100:

```
WRITE "100 F =", (100-32)*5/9, "C"
□ 100 F = 37.77777777777778 C
```

1.3a.

```
□ 1+1 is 2
```

1.3b.

```
□ -0.25 -0.025 -0.0025
```

1.3c.

```
□ 6 4
```

1.4a.

```
WRITE (-17+14.157+500)/3
□ 165.719
```

1.4b.

```
WRITE 142*0.013
□ 1.846
```

1.4c.

```
WRITE (35.30-1.85)/(1984-1837)
□ 0.2275510204081633
```

1.4d.

```
WRITE 0.07*103.12
□ 7.2184
```

of

```
WRITE (7/100)*103.12
```

1.4e.

```
WRITE (1-0.15)*157
□ 133.45
```

of

```
WRITE 157-0.15*157
```

1.5a.

```
WRITE round (12.7*12.7)
□ 161
```

1.5b.

```
WRITE floor (6.7/1.31)
□ 5
```

1.5c.

```
WRITE 6.7 mod 1.31
□ 0.15
```

1.5d.
```
WRITE ceiling (141/8)
```
☐ 18

1.5e.
```
WRITE 1351 mod 7
```
☐ 0

1.5f.
```
WRITE "3859 seconden =", floor (3859/60), "minuten", 3859 mod 60, "seconden"
```
☐ 3859 seconden = 64 minuten 19 seconden

1.5g.
```
WRITE 2 round (200/7)
```
☐ 28.57

1.5h.
```
WRITE (3527 mod 7)+1
```
☐ 7

De volgende oplossing is niet helemaal goed:
```
WRITE 3528 mod 7
```
☐ 0

2. Het geheugen: het PUT-commando

2.1.
```
□ 18.512 19.513 20.514
```

2.2.
```
PUT 1.5*a IN a
```

2.3.
```
PUT (p+q)/2 IN gemiddelde
```

2.4.
```
PUT a+b, a+b IN a, b
```
of
```
PUT a+b IN a
PUT a IN b
```

Dit is niet goed:
```
PUT a+b IN a
✗ PUT a+b IN b
```
omdat het eerste commando de waarde van a verandert, waarna die nieuwe waarde in het tweede commando gebruikt wordt.

2.5.
```
PUT (afstand/tijd)*60 IN snelheid
```
of
```
PUT afstand/(tijd/60) IN snelheid
```
In beide versies zijn de haakjes nodig, omdat er anders een expressie zou staan met / en * waarbij de volgorde waarin deze operatoren worden uitgevoerd verschil maakt voor de uitkomst.
De extra *60 en /60 zijn nodig omdat er 60 minuten in een uur gaan.

2.6.
```
WRITE 9*round(n/9)
```

2.7a.
```
PUT 12 IN foot
PUT 3*foot IN yard
PUT 1760*yard IN mijl
```

2.7b.
```
WRITE mijl+3*yard+2*foot+2.5
□ 63494.5
```

2.7c.
```
PUT floor (1234567/mijl), 1234567 mod mijl IN mijlen, inches
PUT floor (inches/yard), inches mod yard IN yards, inches
PUT floor (inches/foot), inches mod foot IN feet, inches
WRITE "1234567 inch =", mijlen, "ml", yards, "yrd", feet, "ft", inches, "inch"
□ 1234567 inch = 19 ml 853 yrd 1 ft 7 inch
```

Zonder gebruik van mod kan het zo:
```
PUT floor (1234567/mijl) IN mijlen
PUT 1234567-mijlen*mijl IN inches
PUT floor (inches/yard) IN yards
PUT inches-yards*yard IN inches
PUT floor (inches/foot) IN feet
PUT inches-feet*foot IN inches
WRITE "1234567 inch =", mijlen, "ml", yards, "yrd", feet, "ft", inches, "inch"
□ 1234567 inch = 19 ml 853 yrd 1 ft 7 inch
```

Hier is nog een manier, waarin `floor` niet gebruikt wordt:

```
PUT 1234567 mod mijl IN rest
PUT (1234567-rest)/mijl IN mijlen
PUT rest IN inches
PUT inches mod yard IN rest
PUT (inches-rest)/yard IN yards
PUT rest IN inches
PUT inches mod foot IN rest
PUT (inches-rest)/foot IN feet
PUT rest IN inches
WRITE "1234567 inch =", mijlen, "ml", yards, "yrd", feet, "ft", inches, "inch"
□ 1234567 inch = 1 ml 1097 yrd 28 ft 7 inch
```

2.8.

```
PUT lok1 IN derde
PUT lok2 IN lok1
PUT derde IN lok2
```

2.9.

```
PUT 10*b+4 IN b
```

2.10.

```
PUT pg mod 1 IN pg
```

of

```
PUT pg - floor pg IN pg
```

2.11.

```
PUT a, b IN b, a
```

Dat dit werkelijk hetzelfde resultaat oplevert als de oorspronkelijke drie commando's is als volgt in te zien. Veronderstel dat `a` en `b` aanvankelijk de waarden A en B hebben. Na het eerste commando hebben `a` en `b` dan de waarden A en $A-B$, na het tweede B en $A-B$, en na het derde B en A.

2.12.

Als aanvankelijk `x` = X en `y` = Y, dan geldt na het eerste commando `x` = Y en `y` = $-X$, na het tweede commando `x` = $-X$ en `y` = $-Y$, en tenslotte `x` = $-Y$ en `y` = X. Dit resultaat kan even goed bereikt worden met:

```
PUT -y, x IN x, y
```

3. Herhaling: het WHILE-commando

3.1.
Het getal 142857 moet op beide plaatsen veranderd worden in 7, en 500000 in 100:

```
PUT 7 IN veel
WHILE veel < 100:
   WRITE veel
   PUT veel+7 IN veel
```

☐ 7 14 21 28 35 42 49 56 63 70 77 84 91 98

3.2a.
☐ 2

3.2b.
☐ 2 4 6

3.2c.
Dit programma geeft geen uitvoer.

3.3.

Na afloop zal in het geheugen staan: `n = 3`

Hetzelfde resultaat kan bereikt worden met:

```
PUT n mod 17 IN n
```

3.4.
De uitvoer bestaat uit het gehele getal dat het dichtst bij n ligt en groter of gelijk is aan n. Hetzelfde resultaat kan verkregen worden met:

```
WRITE ceiling n
```

3.5.

```
PUT 0 IN n
WHILE (n+1)*142857 < 1000000:
   PUT n+1 IN n
   WRITE n, n*142857 /
```

of

```
PUT 0 IN n
WHILE (n+1)*142857 < 1000000:
   WRITE n+1, (n+1)*142857 /
   PUT n+1 IN n
```

3.6.

```
PUT 1984, 1000 IN j, som
WHILE som < 2000:
   PUT j+1, som*1.13 IN j, som
   WRITE j, 2 round som /
```

☐ 1985 1130.00
☐ 1986 1276.90
☐ 1987 1442.90
☐ 1988 1630.47
☐ 1989 1842.44
☐ 1990 2081.95

3.7.
In beide programma's is a het kleinste oneven getal dat nog niet geschreven is.

3.8a.
De getallen in deze rij zijn de machten van 2. In het volgende programma is `n` de kleinste macht van 2 die nog niet geschreven is.

```
PUT 1 IN n
WHILE n < 1000:
    WRITE n
    PUT 2*n IN n
```

□ 1 2 4 8 16 32 64 128 256 512

3.8b.
De getallen in deze rij zijn de kwadraten van de gehele getallen 1, 2, ... In het programma is `i` het kleinste positieve gehele getal waarvan het kwadraat nog niet geschreven is.

```
PUT 1 IN i
WHILE i*i < 1000:
    WRITE i*i
    PUT i+1 IN i
```

Hier is een versie die een extra lokatie `i2` gebruikt, die het kwadraat van `i` bevat:

```
PUT 1, 1 IN i, i2
WHILE i2 < 1000:
    WRITE i2
    PUT i+1 IN i
    PUT i*i IN i2
```

Een andere manier om deze rij op te vatten is: het verschil tussen twee opeenvolgende getallen van de rij is 2 groter dan het vorige verschil, het eerste verschil is 1, en het eerste getal is 1. In het volgende programma is `n` het volgende te schrijven getal, en is `verschil` het daarna te gebruiken verschil:

```
PUT 1, 3 IN n, verschil
WHILE n < 1000:
    WRITE n
    PUT n+verschil IN n
    PUT verschil+2 IN verschil
```

3.8c.
Het j-de getal van deze rij is j maal het vorige getal, en het eerste getal is 1. In het volgende programma is `n` het volgende te schrijven getal, en `i` het rangnummer van dat getal.

```
PUT 1, 1 IN n, i
WHILE n < 1000:
    WRITE n
    PUT i+1 IN i
    PUT n*i IN n
```

□ 1 2 6 24 120 720

3.8d.
Deze rij is te beschouwen als een rij van paren van gelijke getallen die 111 groter zijn dan die van het vorige paar. Het eerste paar bestaat uit 111 en 111. In het volgende programma bevat `n` het getal dat tweemaal in het volgende te schrijven paar voorkomt:

```
PUT 111 IN n
WHILE n < 1000:
    WRITE n, n
    PUT n+111 IN n
```

□ 111 111 222 222 333 333 444 444 555 555 666 666 777 777 888 888 999 999

3.8e.
In deze rij is elk getal de som van de vorige twee getallen, en de eerste twee getallen zijn 0 en 1. In de volgende twee programma's is `n` het volgende te schrijven getal en `m` het zojuist geschreven getal.

```
PUT 0, 1 IN m, n
WRITE m
WHILE n < 1000:
    WRITE n
    PUT n, m+n IN m, n
```

□ 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

Zonder dubbele PUT-commando's ziet het er zo uit:

```
PUT 0, 1 IN m, n
WRITE m
WHILE n < 1000:
   WRITE n
   PUT m+n IN volgende
   PUT n IN m
   PUT volgende IN n
```

In de volgende versie zijn `a` en `b` de twee getallen die zojuist geschreven zijn.

```
WRITE 0, 1
PUT 0, 1 IN a, b
WHILE a+b < 1000:
   PUT b, a+b IN a, b
   WRITE b
```

3.9.

```
WRITE start
WHILE start mod 100 <> 0 AND start mod 100 <> 8 AND start mod 100 <> 80:
   PUT 2*start IN start
   WRITE start
```

De lokatie `start` speelt de rol van het zojuist geschreven getal.

In de volgende versie bevat `eind` het getal dat bestaat uit de laatste twee cijfers van `start`.

```
WRITE start
PUT start mod 100 IN eind
WHILE eind <> 0 AND eind <> 8 AND eind <> 80:
   PUT 2*start IN start
   WRITE start
   PUT start mod 100 IN eind
```

3.10.

In het volgende programma bevat de lokatie `te'vouwen` de lengte van de zijde die gevouwen gaat worden en `andere` de lengte van de andere zijde.

```
PUT lengte, breedte IN te'vouwen, andere
WHILE te'vouwen >= 1:
   PUT te'vouwen/2, andere IN andere, te'vouwen
   WRITE te'vouwen, andere /
```

3.11.

```
PUT 2 IN a
WHILE NOT a*a >= s:
   PUT a+2 IN a
WRITE a
```

De rol van `a` kan beschreven worden als: `a` bevat een positief even getal en alle positieve even getallen kleiner dan `a` hebben een kwadraat dat niet groter of gelijk is aan `s`.

Het programma kan ook zonder `NOT` geschreven worden:

```
PUT 2 IN a
WHILE a*a < s:
   PUT a+2 IN a
WRITE a
```

3.12.

```
PUT 1, 1, 1 IN s, p, i
WHILE p*(i+1) < 1000:
   PUT i+1 IN i
   PUT p*i IN p
   PUT s+p IN s
WRITE s
```

□ 873

3.13.
In dit programma zijn `m` en `n` twee opeenvolgende getallen in de rij, terwijl alle vorige getallen, `m` inbegrepen, kleiner zijn dan 1000:

```
PUT 0, 1 IN m, n
WHILE NOT n > 1000:
   PUT n, m+n IN m, n
WRITE n
□ 1597
```

3.14.
In dit programma bevat `deler` een geheel getal groter dan 1, terwijl alle kleinere gehele getallen groter dan 1 geen deler zijn van `n`.

```
PUT 2 IN deler
WHILE n mod deler <> 0:
   PUT deler+1 IN deler
WRITE deler
```

4. Eigen commando's: HOW'TO

4.1.
```
VERVEELVOUDIG 2 TOT 51
```

4.2.
De computer begint deze uitvoer te geven en zal daar nooit mee ophouden:
```
□ -3 -6 -9 -12 -15 -18 -21 -24 -27 -30 -33 -36 -42 -45 -48 -51 -54 -57 -60 -63
```

4.3.
Dit commando vervangt de waarde van de lokatie die als feitelijke parameter wordt opgegeven door het kwadraat van die waarde, dus een betere woordkeus zou zijn:
```
HOW'TO KWADRATEER g:
   PUT g*g IN g
```

4.4.
Dit commando verwisselt de waarden van de twee lokaties die als feitelijke parameter worden opgegeven:
```
HOW'TO VERWISSEL a EN b:
   PUT a, b IN b, a
```

4.5.
```
HOW'TO GEEF'MACHTEN'VAN n:
   WRITE n, n*n, n*n*n
```

4.6.
```
HOW'TO GEEF lokatie WAARDE waarde:
   PUT waarde IN lokatie
```

4.7.
Als we een rente van 9 % zo willen kunnen opgeven:
```
VERDUBBEL 1000 TEGEN 9
```
dan ziet de nieuwe versie van de commando-definitie er zo uit:
```
HOW'TO VERDUBBEL begin TEGEN procent:
   PUT 1984, begin IN jaar, som
   WHILE som < 2*begin:
      PUT jaar+1 IN jaar
      PUT 2 round (som*(1+procent/100)) IN som
      WRITE jaar, som /
```
Dit commando moet natuurlijk alleen gebruikt worden met een positieve tweede feitelijke parameter, omdat de waarde van som nooit boven `2*begin` uit zou groeien en het commando nooit zou eindigen. In feite zou het commando ook niet eindigen voor *zeer* kleine positieve waarden van `procent`, omdat `som` dan zijn oorspronkelijke waarde zou kunnen houden doordat de afronding elke keer de zeer kleine groei van `som` teniet zou doen. Hetzelfde zou gebeuren bij een zeer klein beginkapitaal.

4.8a.
```
TEL'OP 2 EN 5 TOT 500
```

4.8b.
```
HOW'TO TEL'OP p EN q MAAL f1 EN f2 TOT grens:
   PUT p, q IN a, b
   WRITE a
   WHILE b < grens:
      WRITE b
      PUT b, f1*a+f2*b IN a, b
```

4.8c.
```
TEL'OP 1 EN 1 MAAL 1 EN 1 TOT 1000
TEL'OP 2 EN 5 MAAL 1 EN 1 TOT 1000
TEL'OP 1 EN 1 MAAL 3 EN 2 TOT 1000
TEL'OP 1 EN 3 MAAL -1 EN 3 TOT 1000
```

5. HOW'TO nader bestudeerd

5.1.

```
HOW'TO VERDUBBEL (3) TOT (100):
   PUT (3) IN a
   WHILE a < (100):
      WRITE a
      PUT a+a IN a
```

5.2.

Omdat het nu juist de bedoeling is dat het VERHOOG-commando de waarde van een *lokatie* verandert, moet er wel een lokatie als feitelijke parameter gegeven worden.

5.3.

De feitelijke parameter die bij `n` hoort mag elke expressie zijn, omdat er nooit iets met PUT aan `n` veranderd wordt. Maar de feitelijke parameter die correspondeert met `k` moet een lokatie zijn, omdat er `k/2` in `k` wordt gezet met PUT.

5.4.

```
HOW'TO T n:
   SHARE totaal
   PUT totaal+n IN totaal
   WRITE totaal
```

Het is noodzakelijk eerst de waarde `0` te geven aan de externe lokatie `totaal`, alvorens het T-commando te gebruiken voor het bijtellen van het eerste getal.

5.5b.

De definitie wordt gekopieerd:

```
g = 4  p = 8  k = 6  pp = 2  f = 1.5
```

```
HOW'TO MAAL g:
   SHARE pp, f
   PUT g*f IN p
   WRITE p
   PUT pp*p IN pp
```

De etiketten van de interne lokaties worden uniek gemaakt:

```
g = 4  p = 8  k = 6  pp = 2  f = 1.5
```

```
HOW'TO MAAL g:
   SHARE pp, f
   PUT g*f IN p'
   WRITE p'
   PUT pp*p' IN pp
```

De voorlopige parameters worden vervangen door de feitelijke parameters, tussen haakjes geplaatst:

```
g = 4  p = 8  k = 6  pp = 2  f = 1.5
```

```
HOW'TO MAAL (k+1):
   SHARE pp, f
   PUT (k+1)*f IN p'
   WRITE p'
   PUT pp*p' IN pp
```

De HOW'TO- en SHARE-regels worden weggegooid:

```
g = 4  p = 8  k = 6  pp = 2  f = 1.5
```

```
PUT (k+1)*f IN p'
WRITE p'
PUT pp*p' IN pp
```

Het overblijvende programma wordt uitgevoerd, waarbij het extra geheugen gebruikt wordt voor de interne lokatie p′:

```
g = 4   p = 8   k = 6   pp = 21   f = 1.5
```

```
PUT (k+1)*f IN p′
WRITE p′
PUT pp*p′ IN pp

p′ = 10.5
```

□ 10.5

Tenslotte wordt het extra stuk geheugen weggegooid:

```
g = 4   p = 8   k = 6   pp = 21   f = 1.5
```

5.5c.
Het produkt van de externe lokatie f en de feitelijke parameter van het MAAL-commando wordt als uitvoer geschreven, en dan wordt de externe lokatie pp met dat produkt vermenigvuldigd. De rest van het geheugen blijft onveranderd.

6. Teksten

6.1.
```
HOW'TO ONDERSTREEP tekst:
   WRITE tekst^"!" /
   WRITE "-"^^(#tekst+1)
```
De +1 zorgt voor het extra min-teken onder het uitroepteken.

6.2.
```
WHILE #antwoord < 10:
   PUT antwoord^"!" IN antwoord
```
of
```
PUT antwoord^("!"^^(10-#antwoord)) IN antwoord
```

6.3.
```
HOW'TO WENS:
   OMLIJST "Goede morgen"
```

6.4.
```
PUT t|1 IN t
PUT t@2 IN t
PUT t@#t IN t
PUT t|(#t-1) IN t
PUT "" IN t
```

6.5.
```
HOW'TO HAK woord:
   WHILE woord|1 = "e":
      PUT woord@2 IN woord
   WHILE woord@#woord = "e":
      PUT woord|(#woord-1) IN woord
```

6.6.
```
HOW'TO ELK'EINDE t:
   PUT t IN w
   WHILE w > "":
      WRITE w /
      PUT w@2 IN w
```

6.7.
```
HOW'TO ELK'BEGIN t:
   PUT t IN w
   WHILE w > "":
      WRITE w /
      PUT w|(#w-1) IN w
```

6.8.
```
HOW'TO SCHUIF1 tekst:
   PUT tekst@2^tekst|1 IN tekst
```

6.9.
```
HOW'TO SCHUIF start:
   PUT start IN t
   WRITE t /
   SCHUIF1 t
   WHILE t <> start:
      WRITE t /
      SCHUIF1 t
```

6.10.

```
HOW'TO TOON'REST tekst:
    PUT tekst IN t
    WHILE #t > 1 AND t|1 = t@#t:
        PUT t@2|(#t-2) IN t
    WRITE "De rest is: ", t
```

In deze definitie speelt de interne lokatie `t` de rol van het middenstuk van `tekst` nadat enkele gelijke letters paarsgewijze aan begin en eind zijn weggehaald.
De extra conditie `#t > 1` is nodig voor toepassingen als `TOON'REST "lepel"`. Na twee herhalingen zou dit woord geslonken zijn tot de enkele letter `p`, hetgeen tot moeilijkheden zou leiden met `t@2` in de volgende regel van de definitie, omdat het onmogelijk is in een tekst van één letter bij de tweede letter te beginnen.
De extra lokatie `t` is werkelijk nodig. Zonder `t` ziet de definitie er zo uit:

```
HOW'TO TOON'REST tekst:
    WHILE #tekst > 1 AND tekst|1 = tekst@#tekst:
        PUT tekst@2|(#tekst-2) IN tekst
    WRITE "De rest is: ", tekst
```

Het probleem is dat deze kortere versie alleen gebruikt kan worden met een *lokatie* als feitelijke parameter, zodat een toepassing als:

```
✗ TOON'REST "kook"
```

fout zou zijn, omdat dat zou leiden tot dit foute commando:

```
✗ PUT ("kook")@2|(#("kook")-2) IN ("kook")
```

6.11.

De alfabetische volgorde is: `"#!< NRS nr1 nr12 nr2 nrs ~@#?`

7. Types

7.1a.
Bij de voorlopige parameter `l` hoort een feitelijke parameter van het type tekst, wegens het voorkomen van de conditie `t|1 = l`, waaruit blijkt dat `l` hetzelfde type moet hebben als `t|1`, dat duidelijk van type tekst is.
Bij de voorlopige parameter `t` hoort ook een tekst, wegens `t|1` en `t@2`.
Bij `f` hoort een feitelijke parameter van het type getal, wegens `PUT 0 IN f` en `PUT f+1 IN f`.
Verder moeten de feitelijke parameters die bij `t` en `f` horen lokaties zijn, omdat in `t` en `f` iets geplaatst wordt met `PUT`, terwijl de feitelijke parameter die bij `l` hoort elke expressie van het type tekst mag zijn.

7.1b.
Beide feitelijke parameters mogen elke expressie van elk type zijn, omdat `PUT` niet gebruikt wordt en er geen speciale operatoren voorkomen.

7.1c.
Alle feitelijke parameters moeten lokaties zijn, omdat er in iedere voorlopige parameter met `PUT` iets geplaatst wordt. Bij `r` hoort een feitelijke parameter van type getal, wegens `r+1`. Het type behorend bij `p` moet hetzelfde zijn, omdat `p` in `r` geplaatst wordt, en `r+1` in `p`. Om soortgelijke redenen moeten de types van de feitelijke parameters die bij `q` en `s` horen hetzelfde zijn, maar het mogen bij de ene toepassing twee getallen zijn en bij een andere twee teksten.

8. Invoer: het READ-commando

8.1.
We moeten het commando PUT 0 IN t uit de definitie halen. Dat betekent dat we, voordat we het commando TEL'IN uitgaven voor het eerst geven, we dit commando moeten geven:

```
PUT 0 IN uitgaven
```

8.2.

```
HOW'TO DOOS:
   WRITE "Geef 3 afmetingen in cm." /
   READ h EG 0
   READ b EG 0
   READ l EG 0
   PUT h*b, h*l, b*l IN a1, a2, a3
   WRITE "Zij-oppervlakken:  ", a1, a2, a3, "cm2" /
   PUT 2*(a1+a2+a3) IN opp
   WRITE "Totaal oppervlak:  ", opp, "cm2" /
   PUT h*b*l IN volume
   WRITE "Volume:            ", volume, "cm3" /
   WRITE "Oppervlak per cm3: ", opp/volume, "cm2" /
```

8.3a.

```
HOW'TO DOZEN:
   WRITE "Geef 3 afmetingen in cm." /
   READ h EG 0
   WHILE h <> 0:
      READ b EG 0
      READ l EG 0
      PUT h*b, h*l, b*l IN a1, a2, a3
      WRITE "Zij-oppervlakken:  ", a1, a2, a3, "cm2" /
      PUT 2*(a1+a2+a3) IN opp
      WRITE "Totaal oppervlak:  ", opp, "cm2" /
      PUT h*b*l IN volume
      WRITE "Volume:            ", volume, "cm3" /
      WRITE "Oppervlak per cm3: ", opp/volume, "cm2" /
      WRITE "Geef 3 afmetingen van volgende doos. Typ 0 om te eindigen." /
      READ h EG 0
```

We zien dat de gebruiker van dit programma als hij wil stoppen alleen een enkele 0 hoeft op te geven als eerste afmeting. De andere twee afmetingen hoeven dan niet meer ingetoetst te worden.

8.3b.

```
HOW'TO DOZEN:
   WHILE 1 = 1:
      WRITE "Geef 3 afmetingen in cm." /
      READ h EG 0
      READ b EG 0
      READ l EG 0
      PUT h*b, h*l, b*l IN a1, a2, a3
      WRITE "Zij-oppervlakken:  ", a1, a2, a3, "cm2" /
      PUT 2*(a1+a2+a3) IN opp
      WRITE "Totaal oppervlak:  ", opp, "cm2" /
      PUT h*b*l IN volume
      WRITE "Volume:            ", volume, "cm3" /
      WRITE "Oppervlak per cm3: ", opp/volume, "cm2" /
```

Als we gebruik maken van het DOOS-commando van opgave 2 kunnen we deze veel kortere versie maken:

```
HOW'TO DOZEN:
    WHILE 1 = 1:
        DOOS
```

De conditie 1 = 1 ziet er misschien wat vreemd uit, maar werkt hier wel goed: deze conditie is altijd waar, zodat het WHILE-commando herhaald blijft worden totdat de stop-toets wordt gebruikt.

8.4.

```
HOW'TO GETALRAADSEL:
    PUT 1, 2 IN p, q
    PUT 2, 1 IN a, b
    WRITE "Een rij getallen begint met:", a, b /
    WRITE "Raad het volgende getal." /
    PUT p*a+q*b IN nieuw
    READ poging EG 0
    WHILE poging <> nieuw:
        WRITE "Nee,", nieuw /
        PUT b, nieuw IN a, b
        PUT p*a+q*b IN nieuw
        READ poging EG 0
    WRITE "Goed."
```

9. Voorwaardelijke uitvoering: de IF- en SELECT-commando's

9.1.
Het zou geen verschil maken als de eerste twee alternatieven zouden worden verwisseld, maar verwisseling van het vierde en het vijfde alternatief zou wel verschil kunnen maken. Dan zou nl. bij een woord als `koning` het eerste alternatief gekozen worden waarvan de conditie klopt, en dat zou dan leiden tot `koningetje` i.p.v. `koninkje`.

9.2.
```
HOW'TO VERKLEIN w:
   PUT w@#w IN z
   SELECT:
      z = "n" OR z = "r" OR z = "l" OR z = "j" OR z = "w":
         WRITE w, "tje"
      z = "m":
         WRITE w, "pje"
      z = "e" OR z = "i" OR z = "u":
         WRITE w, "tje"
      z = "a" OR z = "o":
         WRITE w, z, "tje"
      #w > 5 AND w@(#w-2) = "ing":
         WRITE w|(#w-1), "kje"
      #w > 2 AND w@(#w-2) = "ing":
         WRITE w, "etje"
      ELSE:
         WRITE w, "je"
```

9.3.
De eerste twee toepassingen zijn goed; ze geven als uitvoer:
☐ 3 5
☐ 3 5
De derde toepassing is fout, en de computer zal melden dat de types door elkaar gehaald worden.
De laatste twee toepassingen zijn weer goed:
☐ sportzaal
☐ sportzaal

9.4.
```
HOW'TO TOON'REST tekst:
   PUT tekst IN t
   WHILE #t > 1 AND t|1 = t@#t:
      PUT t@2|(#t-2) IN t
   SELECT:
      #t <= 1:
         WRITE "Ja, ", tekst, " is een palindroom."
      ELSE:
         WRITE "Nee, ", tekst, " is geen palindroom."
```
We zien dat de gevallen `#t = 0` en `#t = 1` in één alternatief behandeld kunnen worden.

9.5.

```
HOW'TO GETALRAADSEL:
   WRITE "Probeer het geheime getal te raden." /
   PUT 44 IN x
   READ poging EG 0
   WHILE poging <> x:
      SELECT:
         poging < x:
            WRITE "hoger" /
         poging > x:
            WRITE "lager" /
      READ poging EG 0
   WRITE "Goed!"
```

10. Lijsten

10.1.
```
HOW'TO HAAL'ELKE element UIT lijst:
   WHILE element in lijst:
      REMOVE element FROM lijst
```

10.2.
```
IF z in {"n"; "r"; "l"}:
   WRITE woord, "tje"
```

10.3a.
```
WRITE min dromen
```

10.3b.
```
WRITE rivier|1
```

10.3c.
```
WRITE max dromen
```

10.4a.
De lijst `{"zeven"; "acht"}` wordt gesorteerd behandeld, dus als `{"acht; "zeven"}`. De uitvoer is dan ook: ☐ `zeven`

10.4b.
In dit geval wordt om het tweede element van een tekst gevraagd, wat neerkomt op het tweede karakter: ☐ `e`

10.4c.
De operator `th'of` wordt hier fout gebruikt, omdat het rechter operand een lijst of een tekst moet zijn, geen getal.

10.4d.
Ook hier wordt de operator `th'of` verkeerd gebruikt: het linker operand moet een (geheel) getal zijn, geen tekst.

10.5a.
In de goede volgorde geplaatst ziet de lijst eruit als `{"111"; "22"; "3"}`, dus de uitvoer is: ☐ `111`

10.5b.
Deze lijst bevat maar één element, dat dus ook het kleinste element is: ☐ `{}`

10.5c.
Het commando:
```
WRITE min min {{5; 2}; {5; 2; 1}}
```
komt neer op:
```
WRITE min min {{2; 5}; {1; 2; 5}}
```
wat weer neerkomt op:
```
WRITE min min {{1; 2; 5}; {2; 5}}
```
Na uitvoering van de (rechtse) `min`-operator blijft er dit over:
```
WRITE min {1; 2; 5}
```
De uitvoer zal dus zijn: ☐ `1`

10.5d.
De lijst `{}` bevat 0 elementen, dus de uitvoer is: ☐ `0`

10.6.
```
WRITE 2 th'of {a; b; c}
```

10.7.

```
HOW'TO VIND'DUBBELE:
   PUT {} IN gehad
   READ regel RAW
   WHILE regel not'in gehad:
      INSERT regel IN gehad
      READ regel RAW
   WRITE "Dat waren", #gehad, "verschillende regels." /
```

10.8.

Na afloop zal de lokatie `h` de hoofdletter bevatten die hoort bij de kleine letter in `k`. Als b.v. `k = "d"`, dan voert de computer dit commando uit:

```
PUT (#{"a"..."d"}) th'of {"A"..."Z"} IN h
```

Nu is `{"a"..."d"}` een kortere schrijfwijze voor `{"a"; "b"; "c"; "d"}`, waar vier elementen in voorkomen. Dat betekent dat de computer uitvoert:

```
PUT 4 th'of {"A"..."Z"} IN h
```

Het vierde element van `{"A"..."Z"}`, waarin de hoofdletters in alfabetische volgorde staan, is `"D"`. De computer voert dus uit:

```
PUT "D" IN h
```

11. Een andere vorm van herhaling: het FOR-commando

11.1.
```
HOW'TO TOON'LIJST lijst:
   FOR element IN lijst:
      WRITE element /
```

11.2.
```
FOR getal IN {a; b; c; d}:
   WRITE getal
```

11.3.
In het volgende programma speelt totaal de rol van de totale lengte van alle waarden die de lokatie woord heeft gehad in de voorgaande herhalingen van het FOR-commando. Het eerste commando is nodig omdat anders het commando PUT totaal+#woord IN totaal niet zou kunnen worden uitgevoerd doordat totaal geen waarde zou hebben.

```
PUT 0 IN totaal
FOR woord IN dromen:
   PUT totaal+#woord IN totaal
WRITE totaal
```

11.4.
```
HOW'TO FAHRENHEIT f:
   WRITE f, "F =", 2 round ((f-32)*5/9), "C" /
```

11.5.
```
HOW'TO ELKE'FAHRENHEIT lijst:
   FOR f IN lijst:
      FAHRENHEIT f
```

11.6.
```
HOW'TO PIJL n:
   FOR i IN {1..n}:
      WRITE "***"^^i /
   FOR v IN {1..n-1}:
      WRITE "***"^^(n-v) /
```
of
```
HOW'TO PIJL n:
   FOR i IN {1..n}:
      WRITE "*"^^(3*i) /
   FOR v IN {1..n-1}:
      WRITE "*"^^(3*(n-v)) /
```

11.7.
In het volgende programma bevat resultaat de letters van tekst die tot nu toe met het FOR-commando zijn doorlopen, maar dan in omgekeerde volgorde.
```
HOW'TO KEER'OM tekst:
   PUT "" IN resultaat
   FOR letter IN tekst:
      PUT letter^resultaat IN resultaat
   WRITE resultaat
```

11.8.

```
PUT {} IN veelvouden
PUT 11 IN voud
WHILE voud < 100:
   INSERT voud IN veelvouden
   PUT voud+11 IN voud
PUT 13 IN voud
WHILE voud < 100:
   INSERT voud IN veelvouden
   PUT voud+13 IN voud
FOR getal IN veelvouden:
   WRITE getal
□ 11 13 22 26 33 39 44 52 55 65 66 77 78 88 91 99
```

of

```
PUT {} IN veelvouden
FOR enkel IN {11; 13}:
   PUT enkel IN voud
   WHILE voud < 100:
      INSERT voud IN veelvouden
      PUT voud+enkel IN voud
FOR getal IN veelvouden:
   WRITE getal
□ 11 13 22 26 33 39 44 52 55 65 66 77 78 88 91 99
```

Een volkomen verschillende aanpak wordt gevolgd in dit programma, waarin `m11` het kleinste veelvoud van 11 bevat dat nog niet op het scherm geschreven is, en `m13` het kleinste nog niet geschreven veelvoud van 13:

```
PUT 11, 13 IN m11, m13
WHILE min {m11; m13} < 100:
   PUT min {m11; m13} IN mm
   WRITE mm
   SELECT:
      m11 = mm:
         PUT m11+11 IN m11
      m13 = mm:
         PUT m13+13 IN m13
□ 11 13 22 26 33 39 44 52 55 65 66 77 78 88 91 99
```

12. Tabellen

12.1.
```
PUT "data" IN meervoud["datum"]
```

12.2.
```
HOW'TO TWEE woord:
   SHARE meervoud
   SELECT:
      woord in keys meervoud:
         WRITE meervoud[woord]
      ELSE:
         WRITE woord, "en"
```

12.3.
```
HOW'TO TIJD uur:
   SHARE verschil
   FOR stad IN keys verschil:
      WRITE stad, ":", uur+verschil[stad], "uur" /
```

12.4.
```
HOW'TO LEES'RIJ s:
   PUT {} IN s
   READ tekst RAW
   WHILE tekst > "":
      PUT tekst IN s[1+#s]
      READ tekst RAW
```

In de volgende versie wordt de lokatie `n` gebruikt om het aantal posten aan te geven dat al in `s` voorkomt:
```
HOW'TO LEES'RIJ s:
   PUT {} IN s
   PUT 0 IN n
   WHILE tekst > "":
      \n = #s
      PUT n+1 IN n
      PUT tekst IN s[n]
      READ tekst RAW
```

12.5.
```
HOW'TO VUL'TABEL tabel:
   PUT {} IN tabel
   WRITE "sleutel: "
   READ sleutel RAW
   WHILE sleutel <> "":
      WRITE "lading: "
      READ lading RAW
      PUT lading IN tabel[sleutel]
      WRITE "sleutel: "
      READ sleutel RAW
```

of, zonder de interne lokatie lading:

```
HOW'TO VUL'TABEL tabel:
   PUT {} IN tabel
   WRITE "sleutel: "
   READ sleutel RAW
   WHILE sleutel <> "":
      WRITE "lading: "
      READ tabel[sleutel] RAW
      WRITE "sleutel: "
      READ sleutel RAW
```

12.6.

```
HOW'TO KEER tabel OM'IN andersom:
   PUT {} IN andersom
   FOR sleutel IN keys tabel:
      PUT sleutel IN andersom[tabel[sleutel]]
```

of

```
HOW'TO KEER tabel OM'IN andersom:
   PUT {} IN andersom
   FOR links IN keys tabel:
      PUT tabel[links] IN rechts
      PUT links IN andersom[rechts]
```

12.7.

```
HOW'TO KEER tabel OM'IN andersom:
   PUT {} IN andersom
   FOR sleutel IN keys tabel:
      SELECT:
         tabel[sleutel] in keys andersom:
            INSERT sleutel IN andersom[tabel[sleutel]]
         ELSE:
            PUT {sleutel} IN andersom[tabel[sleutel]]
```

13. Willekeurige keuze: het CHOOSE-commando

13.1.
```
FOR i IN {1..5}:
   CHOOSE lengte FROM {3; 5; 8}
   VLOEK lengte
   WRITE /
```

13.2.
```
CHOOSE antwoord FROM {"ja"; "ja"; "nee"}
WRITE antwoord
```

13.3.
```
HOW'TO PLUK element UIT lijst:
   CHOOSE element FROM lijst
   REMOVE element FROM lijst
```

13.4.
```
HOW'TO SCHUD l:
   PUT l IN lijst
   WHILE lijst > {}:
      PLUK element UIT lijst
      WRITE element /
```

13.5.
We hoeven alleen het commando: `PUT 44 IN x`

te veranderen in b.v.: `CHOOSE x FROM {30..300}`

De definitie ziet er dan zo uit:
```
HOW'TO GETALRAADSEL:
   WRITE "Probeer het geheime getal te raden." /
   CHOOSE x FROM {30..300}
   READ poging EG 0
   WHILE poging <> x:
      SELECT:
         poging < x:
            WRITE "hoger" /
         poging > x:
            WRITE "lager" /
      READ poging EG 0
   WRITE "Goed!"
```

13.6.
```
HOW'TO GOOI n STENEN:
   PUT 0 IN totaal
   FOR d IN {1..n}:
      CHOOSE ogen FROM {1..6}
      PUT totaal+ogen IN totaal
   WRITE totaal /
```

13.7.
We zullen de tabel keren gebruiken om het aantal keren te bewaren dat elke uitkomst van een worp tot nu toe is voorgekomen:

```
PUT {} IN keren
FOR ogen IN {1..6}:
   PUT 0 IN keren[ogen]
FOR i IN {1..120}:
   CHOOSE ogen FROM {1..6}
   PUT keren[ogen]+1 IN keren[ogen]
FOR ogen IN {1..6}:
   WRITE ogen, ":", keren[ogen] /
```

13.8.
In dit programma bevat de interne lokatie kopie steeds dat deel van de tabel waarvoor nog geen goed antwoord is gegeven. Zodra er voor een bepaalde sleutel wel het goede antwoord is gegeven wordt de betreffende post met DELETE uit kopie geschrapt. Het is beter niet te schrappen in de oorspronkelijke tabel, omdat die anders leeg is aan het eind, en dus later niet meer gebruikt kan worden.

```
HOW'TO OVERHOOR tabel:
   PUT tabel IN kopie
   WHILE kopie > {}:
      CHOOSE sleutel FROM keys kopie
      WRITE sleutel /
      READ antwoord RAW
      SELECT:
         antwoord = kopie[sleutel]:
            WRITE "Goed" /
            DELETE kopie[sleutel]
         ELSE:
            WRITE "Nee, ", kopie[sleutel] /
   WRITE "Dat was alles."
```

14. Pakketten

14.1.

```
PUT {} IN neg'pos
FOR getal IN getallen:
   INSERT -getal, getal IN neg'pos
FOR paar IN neg'pos:
   PUT paar IN neg, pos
   WRITE pos /
```

Het is in dit geval ook mogelijk een eenvoudigere methode te gebruiken:

```
PUT {} IN negatieven
FOR pos IN getallen:
   INSERT -pos IN negatieven
FOR neg IN negatieven:
   WRITE -neg /
```

14.2.

```
PUT {} IN t
FOR i IN {-2..2}:
   FOR j IN {-2..2}:
      PUT 0 IN t[i, j]
```

14.3.

In dit programma bevat oude'worp steeds de uitkomst van de laatse worp:

```
PUT {} IN frekw
FOR i IN {1..6}:
   FOR j IN {1..6}:
      PUT 0 IN frekw[i, j]
CHOOSE oude'worp FROM {1..6}
FOR i IN {1..120}:
   CHOOSE nieuwe'worp FROM {1..6}
   PUT frekw[oude'worp, nieuwe'worp]+1 IN frekw[oude'worp, nieuwe'worp]
   PUT nieuwe'worp IN oude'worp
```