Albert Nijhof 26.06.2016

What's in a name?

(Over datamanagers in forth)

In teksten, bedoeld om iets over forth duidelijk te maken aan niet-forthers, wordt soms triomfantelijk een voorbeeld gepresenteerd in de trant van

```
: 1  ." Hallo! " ;
```

Zo flexibel! In feite stuiten we hier op een fundamentele fout in de interpreter.

De eenvoudige interpreter

De forthinterpreter is heel eenvoudig: hij leest de volgende naam in de inputstroom...

Oh, voor de goede orde: een "naam" is in forth een aaneengesloten serie zichtbare tekens (ASCII 21-7E), aan beide zijden begrensd door spaties, regelbegin of regeleinde. Er zijn behalve de spatie (bijna...) geen tekens die een speciale behandeling krijgen.

Nog een keer:

▶ De interpreter leest de volgende naam uit de inputstroom en zoekt hem op in het woordenboek.

- Gevonden? - dan uitvoeren of compileren, afhankelijk van `STATE` en `IMMEDIATE`.
- Niet gevonden? - dan proberen het als getal te zien. Lukt dat niet dan stopt het proces.

Dat heb ik altijd als volgt opgevat:

▶ De interpreter leest de volgende naam uit de inputstroom, zoekt hem op in het woordenboek en executeert of compileert hem, afhankelijk van `STATE` en `IMMEDIATE`. Is de naam niet te vinden dan probeert de interpreter - uit goeiigheid - nog of hij er een getal van kan maken. Lukt ook dat niet dan stopt het proces.

Dat komt natuurlijk op hetzelfde neer. Het verschil is alleen dat mijn opvatting laat doorschemeren dat de interpreter eigenlijk geen getallen zou moeten verwerken maar dat toch doet, als een gebruikersvriendelijk gebaar naar de programmeur.

Heel merkwaardig eigenlijk: de programmeur weet dat het om een getal gaat en toch laat hij de interpreter eerst nazoeken of het een naam is.

What's in a name?

Data wordt behandeld als naam, de interpreter kan geen gedachten lezen. Daar zit natuurlijk het probleem!

In forthcode zou data ondubbelzinnig herkenbaar moeten zijn als data.

Nu is uitgerekend de opdracht aan de interpreter om data in de inputstroom te herkennen en te verwerken steeds uitgebreid. Het lijkt de hoofdtaak van de interpreter te worden (voor kleine systemen, we vergeten maar even de optimaliserende forths). Het gaat om het aan de verschijningsvorm herkennen van data (getallen, dubbelgetallen, floating point getallen, strings, enz.) door de aanwezigheid van speciale karakters in de data zoals in `#10 &10 %10 "a "b" &a 3.E2 'a C:` en wat al niet. De kennis daarover moet beschikbaar zijn in de interpreter. Dit is wat je noemt een zwitsers zakmes, er worden zelfs speciale structuren (recognizers) voor bedacht! Als die structuren productief zijn, d.w.z. uitbreidbaar door de gebruiker, wordt hier in feite een syntax binnengesmokkeld in de naam (zie Visioenen hieronder).

Jammer, maar vooral voor kleine forthsystemen is dit niet aantrekkelijk.

Het betekent ook dat namen die door hun verschijningsvorm mogelijk als data opgevat zouden kunnen worden problemen kunnen veroorzaken. Je bent dus niet vrij in de keuze van je namen. Namen als `A. 2, #3 V: BAD` kun je in forth beter niet gebruiken en dat is op zijn minst een merkwaardige beperking.

Intermezzo (Visioenen)

```
Modern forth        Ouderwets forth
------------        ---------------
:AMSTERDAM          : amsterdam
\This is comment    \ This is comment
```

en

```
CON:LONDON          constant london
VAR:PARIS           variable paris
```

Ook heel handig:

```
!PARIS              paris !
@PARIS              paris @
```

en

```
VAL:BERLIN          value berlin
TOBERLIN            to berlin
```

Buitengewoon handig:

```
,IF                 postpone if
,THEN               postpone then
```

Je hoeft je natuurlijk niet te beperken tot tweedelige samenstellingen, de mogelijkheden zijn eindeloos!

```
ON@STATE            state @ if
@STATE=0            state @ 0=
@PARIS<0            paris @ 0<
ON@STATE=0          state @ 0= if
ON@PARIS=200        paris @ 200 = if
```

En om dat vervelende postfixgedoe nog wat verder in te dammen:

```
+LONDON             london +
*LONDON             london *
-LONDON             london -
+@PARIS             paris @ +
-@PARIS             paris @ -
+245                245 +
*245                245 *
-245                ...
```

Hè, wat jammer nou. Maar dit is misschien op te lossen met

```
-245                -245
--245               -245 -
...?
```

Een eenvoudiger interpreter

Maar nu serieus: laten we even streng forth-fundamentalistisch stellen - doordravend in de geest van mijn opvatting over de interpreter - dat de interpreter helemaal geen getallen behoort af te handelen:

```
: INTERPRET
    begin bl word find   dup
    while 0< state @ and
          if compile, else execute then
    repeat
    0= abort" Name not found " ;
```

Dat noem ik nog eens een eenvoudige interpreter!
Maar wat doen we nu met data in de inputstroom?
Daar is een heel eenvoudige en forthachtige oplossing voor.

Datamanagers

Datamanagers zijn woorden die data in de inputstroom **aankondigen**, **lezen** en **interpreteren**. Het zijn specialisten. Elk datatype heeft zijn eigen datamanager. Ook het datatype hoeft dus niet meer gedetecteerd te worden.

We maken allereerst een woord `N` dat enkelgetallen verwerkt. Let op, tussen datamanager en data mogen uitsluitend spaties staan. Voorbeeld:

```
HEX
: VISIBLE? ( x -- flag )
  n 21 n 7F within ;
```

Voordat je nu afhaakt:
Ik ga hier niet voorstellen om deze `N` in forth in te voeren, ik vraag slechts om in de geest van deze oplossing na te denken over al die andere soorten data. Moeten die echt allemaal door dat kleine "goeiigheidsgaatje" in de interpreter?

Als ik een interpreter was zou ik zeggen:
"Daar pas ik voor, je geeft ze een vinger en ze nemen de hele hand!
Enkelgetallen wil ik nog voor ze afhandelen, maar verder moeten ze het zelf uitzoeken."

Voorbeelden van datamanagers

```
NN 100 10 \ 2 getallen
DN 100    \ dubbelgetal
FL 100    \ floating point getal
HX 100    \ hex getal
DM 100    \ decimal getal
BN 100    \ binair getal
CH C      \ ASCII code van C
CTRL C    \ control-C als getal
XT C@     \ token van C@
S "ccc"   \ adres en lengte van de string
CS "ccc"  \ adres van counted string
```

Even een zijsprongetje over strings: haal de delimiter van `S"` af en plak hem vooraan de string. Het resultaat is beter leesbaar dan met de klassieke `S"` waarbij je die ene spatie steeds weg moet denken. Bovendien kun je nu je eigen delimiter kiezen (het eerste zichtbare teken na de `S` is de delimiter), bijvoorbeeld voor strings waar aanhalingstekens in voorkomen:

```
S "red wine" S -"green" wine- CS    ' Ciao! '
S "red wine" S -"green" wine- CS    ' Ciao! '
```

Gevolgen

▶ In plaats van de interpreter **tevergeefs** eerst data te laten opzoeken in het woordenboek en vervolgens te laten analyseren om vast te stellen wat hij er mee aan moet, zet de programmeur een datamanager voor de data.

▶ Die datamanager weet hoe hij de data moet behandelen. Je hoeft het datatype en het feit dat het om data gaat niet meer zichtbaar te maken aan de data zelf. Dit verhoogt de **leesbaarheid**, zowel voor de interpreter als voor de mens. (Op den duur dan, want `DN 100` of `FL 100` is natuurlijk wel even schrikken voor hen die anders gewend zijn. Ontwennen moet men willen.

▶ De datamanager vormt samen met de data een **tandem**. Die tandem gedraagt zich als geheel state-smart, zoals een getal in een gewone forth dat ook doet: afhankelijk van `STATE` wordt de data op stack gezet of gecompileerd. De datamanager is immediate en wordt zelf nooit gecompileerd. Het POSTPONEn van een datamanager is voor eigen risico. Zie ook "state-smart angst" hieronder.

▶ Je kunt de datamanagers eventueel zien als onderdeel van een applicatie, want in ieder forthsysteem kun je voor ieder denkbaar datatype datamanagers erbij definiëren **met gewone forthdefinities**. Dat is vooral handig voor kleine forthsystemen. Die hoeven hierdoor niet op allerlei datatypes voorbereid te zijn.

▶ De definitie van een 'naam' in forth wordt onbeperkt geldig: een aaneengesloten serie van zichtbare tekens, waarvan er **geen enkele** een speciale behandeling krijgt.

> Met `N 1` is het lek van `: 1 ." Hallo! " ;` gedicht, want er is nu verschil tussen `1` als naam en `N 1` als getal.

State-smart angst, CH en XT

`[CHAR]` en `[']` zijn ooit in de standaard opgenomen. Datamanagers zoals bijvoorbeeld

```
CH *
XT DROP
```

zouden handiger geweest zijn:

Gebruik `CH` en `XT` indien de data onmiddelijk volgt,
gebruik `CHAR` en `'` indien de data niet onmiddelijk volgt.

Deze gebruiksregels zijn minder esoterisch dan die voor `[CHAR]` en `[']` en dan hebben we het nog niet over de grafische onaantrekkelijkheid van deze namen met rechte haken.

De state-smart angst zou ik willen relativeren. Inderdaad, je kunt state-smart constructies verzinnen - met FOO, BAR en voldoende POSTPONEs of EVALUATEs erin - om aan te tonen dat daarbij soms vreemde dingen gebeuren, maar niemand dwingt je om dat soort programma's te maken.

P.S.

> Een forth interpreter zou alleen single numbers moeten accepteren. Alle andere data hebben dan een datamanager nodig.