*voor Bilthoven, december 2023 (an)*

# PARSING in forth / noForth

## PARSE & BL WORD

```
PARSE ( ch <tekst> -- a n )

ch " parse spelfout"
type

ch " parse spelfout"
2dup 2drop type

cr ch st parse     doorzetterstype

3 parse lollig type \ wat is het resultaat?
```

De afsluiter (delimiter) is geen onderdeel van de string.

Regeleindes zoals vanaf een terminal en CR en/of LF in files maken geen deel uit van de inhoud van de invoer, ze dienen om de invoerstroom regel voor regel aan forth aan te kunnen bieden en komen niet in de invoerbuffer terecht.

```
WORD    ( ch <naam> -- csa )  \ standaard
BL-WORD ( <naam> -- csa )     \ noForth

bl-word broodjes count type
bl-word drankjes count 3 + type

bl-word cognac find . .
```

Samenvattend:
PARSE en WORD zijn onhandig bij interactief gebruik.
Binnen definities verdwijnt het houdbaarheidsprobleem:

```
: .(   ( <tekst> -- )     ch ) parse type ; immediate
: '    ( <naam> -- xt )   bl-word find 0= ?abort ;
: CHAR ( <naam> -- ch )   bl-word 1+ c@ ;
```

# PARSE & BL WORD

```
PARSE ( ch <tekst> -- a n )

@)ch " parse spelfout"↵  OK.2
@)type↵ spelfout OK.0

@)ch " parse spelfout"↵  OK.2
@)2dup 2drop type↵ typefout OK.0

@)cr ch st parse     doorzetterstype↵
    doorzetter OK.0

@)3 parse lollig type \ wat is het resultaat?↵  OK.2
@)type↵ lollig type \ wat is het resultaat? OK.0
```

De afsluiter (delimiter) is geen onderdeel van de string.

Regeleindes zoals ↵ vanaf een terminal en CR en/of LF in files maken geen deel uit van de inhoud van de invoer, ze dienen om de invoerstroom regel voor regel aan forth aan te kunnen bieden en komen niet in de invoerbuffer terecht.

```
WORD    ( ch <naam> -- csa )  \ standaard
BL-WORD ( <naam> -- csa )     \ noForth

@)bl-word broodjes count type↵ TYPET OK.0
@)bl-word drankjes count 3 + type↵ TYPETjes OK.0

@)bl-word cognac find . .↵ -1 304C  OK.0
@)' find .↵ 304C  OK.0
```

Samenvattend:
PARSE en WORD zijn onhandig bij interactief gebruik.
Binnen definities verdwijnt het houdbaarheidsprobleem:

```
: .(   ( <tekst> -- )     ch ) parse type ; immediate
: '    ( <naam> -- xt )   bl-word find 0= ?abort ;
: CHAR ( <naam> -- ch )   bl-word 1+ c@ ;
```

# Woorden die iets met de invoerstroom doen

De rode woorden zijn typisch noForth.

De basiswoorden
    PARSE
    WORD PARSE-NAME *(beide niet in noForth)*
    BL-WORD BL- BEYOND

De invoerstroom
    REFILL INTERPRET

Nieuwe namen
    CREATE  :  CODE ROUTINE
    VARIABLE CONSTANT VALUE VOCABULARY MARKER SHIELD

Bestaande namen
    '  ['] POSTPONE TO +TO ADR INCR

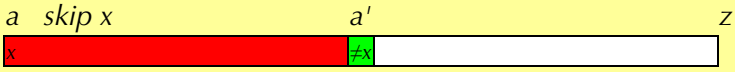
Getallen of bestaande namen
    HX DM BN DN

Tekst
    CHAR [CHAR]  CH  S"  C"  ."  .(  ABORT"
    [IF] [ELSE]  (  \  (*

# SKIP en SCAN,
# de woordjes die het eigenlijke werk doen (in noForth)

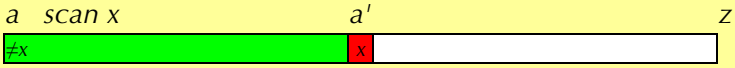
a en z zijn begin- en eindadres van het te doorzoeken gebied.
Gebieden met het 8-bits getal x zijn rood.

SKIP ( z a x -- z a' ) \ Negeer x-bytes voorop




- a' is het adres van het eerste byte ≠ x.
- Wordt a' = z dan was er geen byte ≠ x.

SCAN ( z a x -- z a' ) \ Zoek byte=x in a,z



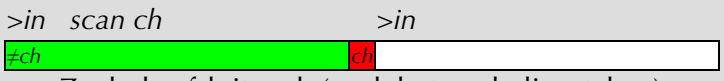

- a' is het adres van het eerste x byte.
- Wordt a' = z dan was er geen x byte.

```
: ADRLEN ( z a -- a n ) tuck - ;
s" suikeroom" bounds .s
ch r scan             .s
2dup adrlen type
ch r skip             .s
2dup adrlen type
ch r scan             .s
```

# De basis parse-woorden in de forth standaard:
# PARSE  WORD  PARSE-NAME

De delimiter ch is steeds rood gemaakt. We gaan er van uit dat een karakter 1 byte is.

PARSE  ( ch ccc<*ch*> -- a n )

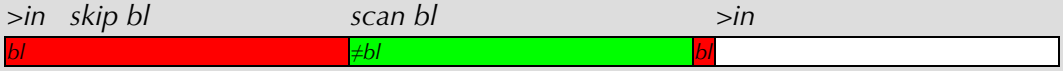



- Zoek de afsluiter ch (vaak het aanhalingsteken).
- Het groene gebied a,n is de string in de invoerbuffer zonder de afsluiter ch.
- >IN wijst naar vlak achter de afsluiter ch of naar het buffereinde.
- Een programma mag de string niet veranderen.

WORD  ( ch <*ch's*>ccc<*ch*> -- csa )


- Skip ch's. Zoek daarna de afsluiter ch (eigenlijk altijd de spatie).
- Kopieer het groene gebied als counted string naar csa in een werkgebied.
- >IN wijst naar vlak achter de afsluiter ch of naar het buffereinde.

PARSE-NAME  ( <*spaties*>naam<*spatie*> -- a n )




- SKIP beginspaties. Doe daarna BL PARSE.

# De basis parse-woorden in noForth:
# PARSE  BL-  BEYOND

PARSE  ( ch ccc<*ch*> -- a n )

BL-  ( <*spaties*> -- )

*>in  skip bl  >in*


- Zoek de eerste niet-spatie, REFILL indien nodig.
- >IN wijst naar die eerste niet-spatie.

BEYOND  ( ch ccc<*ch*> -- )

*>in  scan ch  >in*

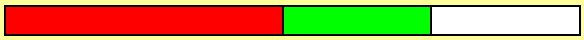

- Zoek ch, REFILL indien nodig.
- >IN wijst naar vlak achter het gevonden karakter ch.

- NoForth leest alle control-tekens in de invoerstroom als spaties.
- NoForth negeert in de invoerstroom karakters groter dan hx 7F.

```
: BL-WORD    ( <naam> -- a n ) bl- bl parse >fhere ;
: (          ( <tekst> -- )    ch ) beyond ; immediate
```

# Code van de noForth parse-woorden

```
REFILL ( -- gelukt/mislukt )  = laat de volgende regel binnenkomen
IB                            = adres van de actuele invoerbuffer
PAREA ( -- z a )              = resterende actuele parse area
a = het adres waar de offset >IN naar wijst ( IB >IN @ + )
z = eindadres van de inhoud van de invoerbuffer
```



*ib* *a* *z* *end-buf*

```
: PARSE  ( ch -- a n ) \ no refill
    >r parea tuck           \ a z a
    r> scan                 \ a z a'
    tuck                    \ a a' z a'
    1+ umin   ib - >in !    \ a a'
    over - ;

: BL- ( -- )           \ skip spaces (with refill)
    begin   parea           \ z a
            bl skip         \ z a'
            dup ib - >in !  \ z a'
            =               \ end-of-buffer?
    while   REFILL 0=
    until then ;

: BEYOND ( ch -- )     \ scan for ch (with refill)
    >r
    begin   parea           \ z a
            r@ scan         \ z a'
            tuck            \ a' z a'
            =               \ a' end-of-buffer?
    while   drop
            REFILL 0= ?abort
    repeat rdrop            \ a'
    1+ ib - >in ! ;

: BL-WORD ( -- csa )   \ BL WORD (with refill)
    bl-   bl parse          \ a n
    >fhere ;                \ move to workspace as counted string
```