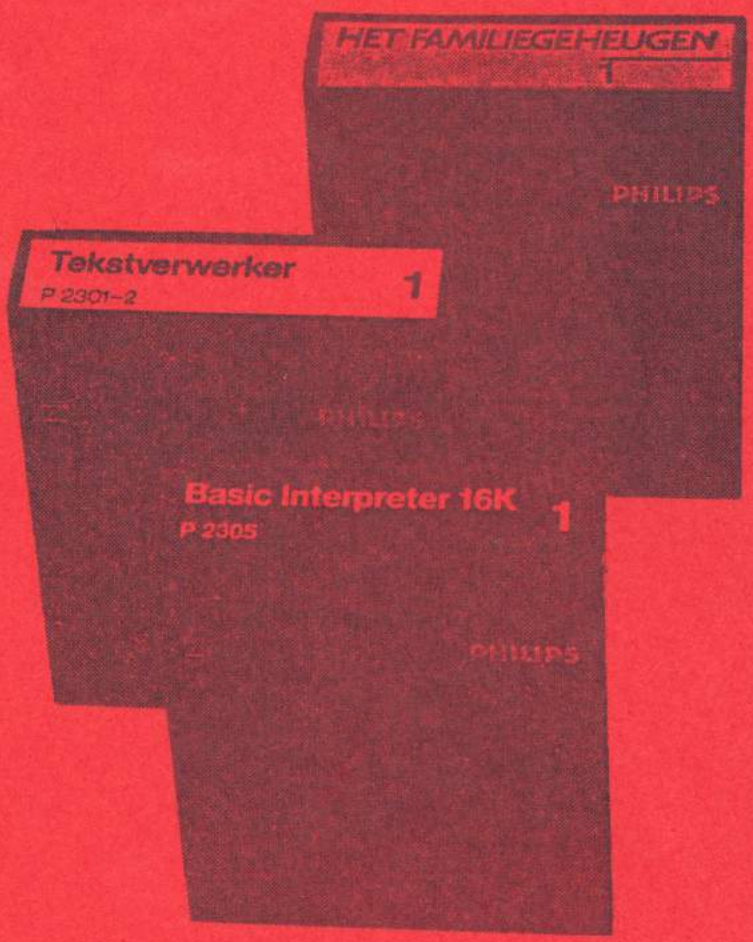
HOPPIE'S EXTENDED BASIC



P2000T

JEROEN HOPPENBROUWERS

Hoppie's Extended BASIC versie 3.1

De standaard BASIC-insteekmodule (BASIC-NL) biedt vele mogelijkheden, maar soms komt u toch echt instructies tekort. Dat is bijvoorbeeld zo als u aan gegevensverwerking wilt gaan doen: de BASIC-NL kan alleen een array in zijn geheel naar cassette schrijven, terwijl bij nagenoeg *alle* andere computers schitterende stuuropdrachten voor de recorder beschikbaar zijn.
Ook wanneer u uw programma's wat netter wilt opbouwen, of wanneer u een opstapje wilt maken naar PASCAL, blijkt BASIC echt een aantal opdrachten te missen. "Gestructureerd" programmeren *gaat* wel, maar eist een hele hoop zelfdiscipline van de programmeur. En het ziet er nog niet uit ook.

Het programma Hoppie's Extended BASIC -dat overigens ook samen met het schijfbesturingssysteem JWS-DOS en/of het 64K-geheugenbanken-schakelprogramma RDOS kan draaien- voegt een stel extra instructies toe aan de standaard BASIC. Vrijwel al deze instructies zijn behulpzaam bij het programmeren in BASIC. Ze verschaffen de programmeur handig gereedschap en noodzakelijke opdrachten om met gegevensbestanden op cassettes te kunnen werken en om aan *echt* gestructureerd programmeren te doen.

Programmeerhulpen:
Hoppie's BASIC bevat de opdrachten UPPER en LOWER, waarmee omgeschakeld kan worden tussen hoofd- en kleine letters bij het uit-LIST-en van een programma. Kleine letters lezen nl. veel prettiger. Verder een kleine screen editor zodat herhalingen van dezelfde opdracht makkelijker in te voeren zijn. Deze extra editor vult de standaard line editor aan. *Samen* vormen ze een mooi koppel waarmee u alles aankunt zonder rommel op het scherm. En als laatste de opdracht PRON, om PRINT en LPRINT te koppelen: alles wat op het scherm verschijnt, komt ook net zo op de printer. Een soort logboek dus, erg handig wanneer er "even" wat programma-uitvoer geprint moet worden. U hoeft dan niet elke PRINT te vervangen door een LPRINT en krijgt ook nog eens de normale output op uw scherm.

Files op cassette:
Met het programma worden de recorder-opdrachten uitgebreid met een stel standaardinstructies voor het verwerken van seriele bestanden. Het zijn OPEN, CLOSE, PRINT#, INPUT# en EOT. Hiermee kunt u op de normale MicroSoft-manier files aanleggen op een minicassette. Extra mogelijkheid is het later verlengen van een al aangelegde file. Alle types variabelen kunnen door elkaar verwerkt worden. Een seriele file kunt u het beste vergelijken met een (lange) rij gegevens die ooit een voor een "geschreven" worden en daarna weer een voor een "gelezen" kunnen worden. Hij komt ongeveer overeen met een DATA-regel, maar dan beschrijfbaar door het programma. Bijkomend voordeel: een seriele file staat *niet* in het geheugen en kost dus geen programma-ruimte.

Structuur:
Het is nu mogelijk geheel zonder regelnummers te programmeren. Een regelnummer achter GOTO of GOSUB kan worden vervangen door een label, bijv. GOTO "Lus" of GOSUB "Sorteer". Hiermee ondervangt u nietszeggende opdrachten als GOTO12546 of IFA=9THEN456ELSEGOSUB667:GOTO98. IF A=1 THEN GOSUB "Sorteer" is veel duidelijker! Bovendien kan het programma nu zeer simpel hernummerd worden. Verder zijn WHILE en WEND toegevoegd om conditionele lussen te kunnen maken zonder GOTO's. Deze twee opdrachten vervangen een IF-THEN GOTO-opdracht en een "terugwijzende" GOTO, en hebben duidelijk een stevige toename van de structuur tot gevolg. Een WHILE-WEND-lus vormt een geheel binnen het programma en is als zodanig direct herkenbaar.

IF-THEN-(ELSE)-opdrachten kunnen naar behoefte worden uitgerust met BEGIN- en END-statements zodat ze over meerdere regels te verdelen zijn. Dat scheelt enorm in GOTO's en duidelijkheid. U kunt nu zeven regels BASIC schrijven die worden uitgevoerd als de uitdrukking "waar" is en bijv. vijftien regels die de P2000 moet volgen bij "niet waar". Een IF-THEN-(ELSE)-opdracht laat zich duidelijk volgen, en ook het begin en het einde komen scherp naar voren.

Met Hoppie's BASIC 3.1 kunt u ook procedures definieren (DEFPROC en PEND) zodat de BASIC van de P2000 bijna helemaal als PASCAL te gebruiken is, uitgezonderd natuurlijk het compileren naar machinetaal. Een procedure is een nieuwe BASIC-instructie, bijv. SORTEER(A,B), die echter zelf ook in BASIC geprogrammeerd is! Hierdoor kunt u bepaalde handelingen eerst grof omschrijven en ze pas later verder uitwerken, eventueel door nog een paar keer van een procedure gebruik te maken. Een zo geschreven programma leest bijna als een goed boek, zonder opzoeken van regelnummers en vage subroutines.

Programma's geschreven onder Hoppie's BASIC zijn bijzonder duidelijk, ook voor iemand die het programma niet kent en vaak zelfs nog voor iemand die geen BASIC verstaat! De bekende spaghetti van regelnummers vervalt totaal en door de procedures worden programma's bijna helemaal gereduceerd tot normaal Nederlands. Aanpassingen op bestaande programma's zijn heel makkelijk uit te voeren en niet alleen door de "originele" maker!
Voor de programmeur betekent het gebruik van Hoppie's BASIC een stevige verlichting van zijn taak: programma's worden veel beter leesbaar, zijn sneller klaar en stukken gemakkelijker te onderhouden. Zeker voor onderwijstoepassingen een aantrekkelijk idee!

Als opstap naar PASCAL is het ook goed geschikt. De P2000 blijft de P2000, dus met dezelfde editor, dezelfde foutmeldingen, dezelfde instructies etc., maar door de mogelijkheid o.a. procedures te maken komt de BASIC van de P2000 kwa mogelijkheden en structuur dicht in de buurt van PASCAL. Het is zondermeer mogelijk te oefenen met de beginselen van PASCAL terwijl de P2000 nog steeds BASIC verstaat.

Hoppie's BASIC is een uitbreiding van de BASIC-NL. Toch kunt u uw BASIC-NL-module blijven gebruiken. Hoppie's BASIC is namelijk een programma (in machinetaal) dat zich inlust in de module. Zo lijkt het alsof de nieuwe instructies altijd al in de module hebben gezeten. Natuurlijk vergeet de P2000 ze weer als u hem uitschakelt. Daarom zult u Hoppie's BASIC telkens na het inschakelen van de P2000 even moeten inladen, maar dat kost nauwelijks vijftien seconden en het hoeft maar een keer per avond.

Bij een zo uitgebreid programma hoort natuurlijk een goede en complete handleiding op papier. Die is er dan ook. Vijftien pagina's (A4-formaat), vol met aanwijzingen, voorbeelden en tips. Geschreven voor de gebruiker die al wat van BASIC weet, dus bijv. het gebruik van de standaard line editor komt er niet in voor. Maar de nieuwe instructies zijn zo uitgelegd dat u beslist geen volleerd programmeur hoeft te zijn om alles te begrijpen!

Het maken van Hoppie's BASIC was een behoorlijke kluif en de handleiding is natuurlijk ook een verhaal apart, minstens zo tijdrovend. Daarom is het programmapakket niet gratis.
Hoppie's Extended BASIC kost inclusief handleiding F 15,-. Wilt u het thuisgestuurd hebben, dan F 20,- inclusief verzendkosten. Bij deze prijzen is een eventuele cassette *niet* inbegrepen, die zult u zelf op moeten sturen of af moeten geven. Eventuele nieuwe versies zijn altijd tegen porto na te bestellen en u krijgt daar ook bericht van. En natuurlijk is de auteur altijd bereid om u vooruit te helpen wanneer er zich een probleem voordoet.

Jeroen Hoppenbrouwers - Wilhelminapark 8 - 5554 JE VALKENSWAARD
Telefoon (04902)-13808 (19-21 uur of in het weekend) Vidibus 400021237

Handleiding bij het P2000T-programma Hoppie's Extended BASIC versie 3.0

Hoewel de BASIC-NL van de P2000T weinig te wensen overlaat zijn er toch altijd instructies die "ontbreken". Met name in het informatica-onderwijs was er behoefte aan een uitbreiding van de standaard-BASIC, om de P2000 te laten voldoen aan overheidseisen.
Ook vanuit de hobby-sfeer kwamen geluiden die duidelijk op een leemte wezen. Hier moet met name gedacht worden aan gebruikersvriendelijkheid van de P2000, want het kan altijd nog beter.
Na hier-en-daar wat rondvragen en afgaande op eigen ervaring heb ik toen het programma *Hoppie's Extended BASIC* in elkaar gezet, dat een stel nuttige en/of makkelijke instructies aan de BASIC-NL koppelt. Dit gaat ten koste van circa drie kilobytes werkgeheugen. De maximale programmeerruimte neemt dus wat af, maar voor wat hoort wat, nietwaar?

Het programma is geschikt voor zowel een 16K- als een 32K-P2000, het onderscheid tussen beide modellen wordt door het programma zelf gemaakt, dus als er in de klas twee types computers staan komen er geen problemen van.
Er is geen versie voor een 48K-P2000, omdat die erg zeldzaam zijn, en als er een 48K-print in zit hangen er meestal discdrives aan. Dan houdt U toch weer een 32K-machine over, dus... Gebruikt U geen drives, dan is het nuttig even bij dezelfde auteur te informeren naar RDOS, waarmee U de zes geheugenbanken van de 48K-computer ombouwt tot RAM-disc met alle normale drive-mogelijkheden inclusief random- en sequentiele files.
Hoppie's Extended BASIC werkt zonder meer samen met het standaard disc-operating system (JWS-DOS) en RDOS, hoewel de seriele files altijd gebruik van de cassetterecorder blijven maken.

Als U het programma voor de eerste keer gebruikt kunt U het beste even de listing bekijken. Vooraan staan namelijk een aantal parameters die U naar believen kunt aanpassen; ze hebben betrekking op de printerinstellingen van de P2000. Gebruikt U een printer die niet geheel standaard is, dan is het erg gemakkelijk deze getallen goed te zetten en Hoppie's Extended BASIC daarna weer te CSAVE-n. Elke keer als U daarna het programma opstart wordt de P2000 automatisch aangepast aan Uw printer.
Let op: na RUN wordt het programma gewist! Dus niet eerst even kijken.

Wanneer U het Extended BASIC opgestart heeft verschijnt er een soortgelijke mededeling als na het inschakelen van de P2000, en als er een terugspoelautomaat in Uw computer zit wordt de cassette weer aan het begin gezet.
Verder merkt U niets van de uitbreiding, alle BASIC-instructies werken precies zoals "vroeger" en programma's die onder BASIC-NL draaien werken ook nog als vanouds. Alleen: er is iets minder vrije geheugenruimte, en dat kan soms tot problemen leiden. Veel programma's beginnen namelijk met een of andere CLEAR-opdracht en in ongunstige gevallen wordt daarmee Hoppie's Extended BASIC overschreven. In dat geval kan er natuurlijk van alles gebeuren...
Gelukkig zal de Extended BASIC in de meeste gevallen voor nieuwe programma's gebruikt worden, en dan heeft U alles zelf in de hand.

De nieuwe instructies

In de volgende paragrafen zullen de extra instructies een voor een worden besproken, samen met wat voorbeelden. Omdat het geen echte nieuwe BASIC-commando's zijn, zie ik er van af alles tot in detail uit te spitten. Er wordt van uitgegaan dat U de BASIC-NL redelijk beheerst en al kleine programmaatjes kunt schrijven.

WHILE/WEND

Met deze overbekende instructie kunt U wat meer structuur brengen in Uw programma's. De WHILE/WEND-constructie maakt namelijk de GOTO-opdracht als lussluiter overbodig. In bepaalde gevallen is het in BASIC onmogelijk om *geen* GOTO te gebruiken, nl. als afhankelijk van een voorwaarde een stuk programma wel of niet, of nogmaals moet worden doorlopen. Wanneer die voorwaarde "zes keer" is kunt U de FOR/NEXT-lus gebruiken, maar in het volgende geval kan dat niet of met heel veel moeite:

```
10 A=INP("")
20 IF A=16  THEN X=X-1
30 IF A=19  THEN X=X+1
40 IF A><13 THEN GOTO 10
50 .......
```

Deze GOTO is nog wel te overzien, maar er zijn veel gevallen waarbij zover teruggesprongen wordt dat de structuur onduidelijk wordt.
Precies hetzelfde programmadeel kan nu ook zonder GOTO:

```
10 A=0: REM Als A maar >< 13 is!
20 WHILE A><13
30  A=INP("")
40  IF A=16 THEN X=X-1
50  IF A=19 THEN X=X+1
60 WEND
70 .......
```

Ondanks het feit dat er wat meer opdrachten nodig zijn is dit laatste programmaatje veel duidelijker: *zolang als* A><13 wordt de lus uitgevoerd! Als A=13 (dus als de uitdrukking achter WHILE "niet waar" is) gaat BASIC verder met de instructie die volgt op de WEND-opdracht, in dit geval dus regel 70.
Om niet in een oneindige lus te verzanden moet de uitdrukking natuurlijk wel ooit "niet waar" worden, en omgekeerd: het is heel goed mogelijk dat de hele lus niet wordt uitgevoerd, namelijk als de uitdrukking al bij de eerste WHILE "niet waar" is.
WHILE/WEND-lussen kunnen "genest" worden, dat wil zeggen dat ze binnen elkaar mogen voorkomen. U krijgt dan constructies als de volgende:

```
WHILE
  ...
  WHILE
    ...
    ...
  WEND
  ...
WEND
...
```

U ziet dat inspringen heel verhelderend kan werken, maar het is beslist niet verplicht. Op deze manier kunnen maximaal 10 lussen genest worden.
Staat er een WEND teveel in het programma, dan verschijnt de mededeling "WEND without WHILE" en wordt de uitvoering onderbroken. Een WHILE teveel veroorzaakt de foutmelding "WHILE without WEND", maar deze melding komt pas als de uitdrukking achter WHILE "niet waar" is.

Er geldt een beperking voor de plaats waar WHILE of WEND mag staan, namelijk: *altijd vooraan een nieuwe regel* (spaties uitgezonderd).

```
10 WHILE A<10: A=A+1: PRINT A:: WEND: PRINT A
```

mag dus NIET, maar

```
10 WHILE A<10: A=A+1: PRINT A:
20 WEND: PRINT A
```

geeft geen problemen. Het gaat alleen om de instructies zelf. Wanneer U desondanks toch een opdracht verkeerd neerzet, volgt de foutmelding *Wrong place*.
Overigens: WHILE/WEND dient om gestructureerd en beter leesbaar te kunnen programmeren, en het samenproppen van verschillende instructies die het programmaverloop beinvloeden op 1 regel leidt nu niet bepaald tot betere leesbaarheid...
Zo, een van de meest voorkomende GOTO's is al weggewerkt. Nu blijft het tweede geval nog over: het aanroepen van een op zichzelf staand stuk programma. Meestal wordt een programma namelijk verdeeld in een aantal kleinere stukken die een kleine, afgeronde opdracht vervullen. Zo'n onder-programma heet een sub-routine en wordt aangeroepen met GOSUB (regelnummer).
Er onstaat dan -grof gezegd- de volgende structuur:

```
10 GOSUB 1000: REM Initialisatie
20 GOSUB 2000: REM Haal teken op
30 GOSUB 3000: REM Verwerk teken
40 GOSUB 4000: REM Toon uitkomst
50 END
```

Elk deel van het programma begint op een bepaald regelnummer. Op zichzelf zegt dat nummer nog niets, en daarom wordt er vaak een REM-opdracht achter gezet die de instructie wat meer betekenis geeft.
Op dit principe is voortgeborduurd bij het maken van het

Label-systeem

Op precies dezelfde manier als in het bovenstaande voorbeeld wordt een programma onder Hoppie's Extended BASIC:

```
10 GOSUB "Initialisatie"
20 GOSUB "Haal op"
30 GOSUB "Verwerk"
40 GOSUB "Uitkomst"
50 END
```

Het grote verschil is dat een GOSUB- of GOTO-opdracht niet meer uitsluitend een regelnummer mag bevatten, maar ook een naam, een label. Ergens anders in het programma staat dan dat label, en de sprongopdracht laat BASIC verdergaan achter dat label. Bijvoorbeeld:

```
1000 "Initialisatie"
1010 DIM A(100),B(20)
10.. ......
1100 RETURN
```

en natuurlijk mag hetzelfde op andere regels, met andere labels. Zo wordt een programma een heel stuk leesbaarder! Ook als er met een GOTO gesprongen wordt kan het gebruik van een label erg verduidelijkend werken: GOTO "Stoppen" is altijd beter leesbaar dan GOTO 65432.

Een GOTO- of GOSUB "Label"-opdracht mag overal in een programma staan, dus ook midden in een regel. Het bijbehorende label *moet* echter vooraan een nieuwe regel staan, maar dat is niets nieuws want met een normale GOTO- of GOSUB-instructie kunt U ook niet midden in een regel springen! Vindt Hoppie's BASIC toch een label waar het niet mag, dan komt *Wrong place*, net zoals bij WHILE.

GOTO- of GOSUB-opdrachten mogen ook in een IF-THEN-constructie opgenomen worden, dat is bekend. Maar om Hoppie's Extended BASIC te laten detecteren of er een GOTO/GOSUB-label-instructie gebruikt wordt, *moeten* deze commando's voorafgegaan worden door een dubbele punt. Dus het BASIC-NL:

```
100 IF A<10 THEN 6588
110 IF A$="Stop" THEN GOSUB 544
```

wordt in Hoppie's Extended BASIC:

```
100 IF A<10 THEN: GOTO "Label 1"
110 IF A$="Stop" THEN: GOSUB "Label 2"
```

Samengevat: als er gewoon naar een bekend regelnummer gesprongen moet worden, dan mag de hele structuur onveranderd blijven. *Alleen* bij een sprong naar een label *moet* de GOTO of GOSUB voluit geschreven worden en direct achter een regelnummer of dubbele punt staan.
Een label wordt opgezocht volgens het principe: "Wat niet weet, dat niet deert". Als er vier tekens opgegeven worden, leest de P2000 ook alleen de eerste vier tekens van de labels in het programma. GOTO "LAB" springt dus naar het eerste label in het programma dat met "LAB" begint. Wat er achter nog allemaal staat doet er niet toe.
Labels mogen ook een stringvariabele zijn!
Constructies als GOSUB A$, GOTO 'LAB"+STR$(N) etcetera zijn zonder meer toegestaan. Dat biedt een flink aantal extra programmeer-hulpmiddelen. Als er bijvoorbeeld in een programma drie verschillende sorteer-routines voorkomen, dan kan ergens in het begin met 1 opdracht bepaald worden welke van de drie routines aangeroepen wordt: SO$="Sorteer (1, 2 of 3)". Elke keer als er gesorteerd moet worden typt U GOSUB SO$ en de juiste routine wordt uitgevoerd.
Het omgekeerde mag ook: een label dat los in het programma staat vervangen door een variabele of een string-expressie. Natuurlijk zou

```
10 A$: PRINT "Subroutine": RETURN
```

een syntax error opleveren. BASIC weet niet dat A$ hier als label staat. Daarom MOET een label waar naartoe gesprongen wordt altijd met een aanhalingsteken beginnen:

```
10 ""+A$: PRINT "Subroutine": RETURN
```

is volkomen legaal. De lege string "" verandert de inhoud van A$ niet.
Het zal wel duidelijk zijn dat door deze extra mogelijkheden van het labelsysteem (vervanging door variabelen) de structuur en leesbaarheid van een programma flink geweld aan kan worden gedaan. In sommige gevallen levert het echter grote besparingen op en wanneer het met verstand wordt toegepast is het alleen maar erg nuttig en makkelijk.

Bovenstaande twee uitbreidingen van de BASIC-NL zijn ontworpen met de bedoeling het gestructureerd programmeren wat aantrekkelijker te maken. Het is niet zo dat ineens alles met WHILE/WEND of labels moet, maar zeker het gebruik van WHILE is zeer aan te bevelen omdat de programmeur dan verplicht wordt *eerst na te denken* voordat hij aan het typen slaat. Het is ook erg nuttig kwistig met spaties en REM's te strooien, en het programma te verdelen in op zich staande modules (subroutines, zo U wilt). Toegegeven, het kost meer geheugen, maar het programma is gegarandeerd sneller klaar en veel gemakkelijker te onderhouden. Dat laatste lijkt misschien een nadeel (het is best leuk dat alleen U het programma "snapt") maar wie even doordenkt komt al snel tot de conclusie dat dat maar mooipraterij is.

Het gebruik van sequentiele files

De standaard BASIC-NL heeft alleen de mogelijkheid array's naar cassette te schrijven. In veel gevallen is dat niet de meest ideale oplossing, maar de rare recorder van de P2000 (geen kwaad woord overigens!) maakt het nu eenmaal moeilijk losse gegevens op tape te zetten. Deze recorder werkt namelijk op de professionele manier met blokken van een kilobyte (1024 tekens), en de oorspronkelijke ontwerpers van de BASIC hadden alleen maar cassetteroutines geschreven voor gebruik met een audio-cassetterecorder die geen blokken kent. De mogelijkheden om met gegevensbestanden te werken werden zo flink besnoeid. Bij de ontwikkeling van Hoppie's Extended BASIC kwamen deze problemen al snel bovendrijven. Na wat "research" bleek het best mogelijk te zijn om een klein maar volledig file-system bij te maken. Omdat de P2000 maar 1 recorder heeft verviel de mogelijkheid meerdere bestanden tegelijk te openen. Dat hoeft echter geen probleem te zijn, zolang er nieuwe programma's gemaakt worden. *Het zondermeer overtypen van programma's die wel veronderstellen dat er twee recorders of zelfs discdrives aan de computer hangen kan dus op problemen stuiten, hoewel de syntax volledig overeenstemt met die van Hoppie's Extended BASIC.*

Bij de afhandeling van fileopdrachten wordt er gebruik gemaakt van een stuk gereserveerd geheugen van 1 kilobyte (precies een cassetteblok dus). Met speciale instructies kunnen gegevens in deze zogenaamde *buffer* worden geschreven of eruit worden gelezen. Wanneer de buffer "vol" raakt, wordt zijn gehele inhoud naar de cassette geschreven en wanneer de buffer leeggelezen is laadt de computer automatisch het volgende blok van de cassette in.

Stel, U hebt een zinnetje en dat moet opgenomen worden op de cassette. U zet dan de weg te schrijven tekens bijvoorbeeld in een stringvariabele:

```
10 A$="Dit moet geschreven worden"
```

Vervolgens geeft U BASIC de opdracht de buffer te wissen en klaar te maken om er in te schrijven:

```
20 OPEN "O",#1,"Bestandsnaam"
```

Na deze vloek is er het volgende gebeurd:

- de filebuffer is leeggemaakt.
- de P2000 weet dat U wilt gaan schrijven, dus dat U de file geopend hebt (zo heet dat) voor OUTPUT. Daar dient die "O" voor.
- de buffer heeft nummer 1 gekregen. Hoewel de P2000 maar 1 buffer heeft (er is tenslotte maar 1 recorder) moet U hier toch iets invullen. Dat is alleen maar om compatible te blijven met andere systemen. Het cijfer dat U invult heeft geen betekenis, maar het ligt nu wel vast. In het vervolg heet de buffer "1". Merk op dat een buffernummer altijd begint met een #!
- als laatste geeft U de naam op waaronder het bestand op de cassette gezet moet worden.

Tevens wordt de cassette in de recorder teruggespoeld.

Schrijven

U schrijft nu de gegevens in de buffer met de opdracht

```
30 PRINT #1, A$
```

Met het "#" geeft U aan dat A$ niet naar het scherm moet, maar naar de filebuffer. Achter het hekje volgt Het buffernummer, dat gelijk moet zijn aan het bij OPEN opgegeven nummer. Zoniet, dan volgt de foutmelding "File not open".

Het resultaat van de expressie achter PRINT #1,... wordt nu in de filebuffer gekopieerd. Die expressie mag zowel numeriek als alfanumeriek zijn, PRINT #1,66 mag dus ook en PRINT #1,STR$(655)+"Test1"+LEFT$(A$,4) mag ook. De P2000 zet bij elk weggeschreven gegeven (dat heet een "record") ook het type in de buffer. Achteraf, als de file weer uitgelezen wordt, is dus meteen bekend of een gelezen record van het string-, integer-, single precision- of double precision-type is. Deze voorziening zit meestal niet in de andere BASIC-dialecten en leidt dan ook vaak tot fouten.
Achter een PRINT#-opdracht mogen verschillende gegevens volgen, mits ze gescheiden worden door komma's:

```
PRINT #1,A$,B,544,"Test"
```

In tegenstelling tot andere BASIC-versies hebben de komma's alleen een scheidingsfunctie en bepalen ze niet of de weggeschreven records wel- of niet aan elkaar geplakt worden zoals bij PRINT. Daar is namelijk wel een groot verschil tussen een ";" of een "," of niets. Bij Hoppie's Extended BASIC wordt elk gegeven van het vorige gescheiden door een type-byte en niet door een carriage return of komma. Dit bespaart een hele hoop ergernis en problemen.

Na een aantal PRINT#'s zal de buffer volraken. Als dat de eerste keer na het openen van de file gebeurt zal de recorder gaan spoelen en een leeg stukje band zoeken. Wordt er een bestand gevonden dat met dezelfde letter begint als de opgegeven naam, dan wordt dit gewist. Precies zoals bij CSAVE kan de P2000 geprogrammeerd worden om netjes "Hier overheen ?" te vragen, namelijk met POKE &H60AC,128.
LET OP: omdat de lengte van de sequentiele file nog onbekend is, wordt alles wat verderop op de cassette staat ook gewist!
Overtuig U er dus van dat er na de te wissen file niets belangrijks meer op de tape staat, want het is onherroepelijk verloren.
Naarmate het programma vordert zal de buffer waarschijnlijk nogmaals volraken. Ook dan wordt hij weer in zijn geheel naar de cassette geschreven, maar omdat de band nog steeds op de goede plaats staat gaat dat heel snel (circa drie seconden). Er ontstaan grote problemen als de cassette inmiddels verspoeld is (door een CSAVE of een OUT 16,68 bijvoorbeeld) of als iemand zo leuk is geweest hem te verwisselen of om te draaien. Tegen de eerste twee fouten is geen actie ondernomen maar de echte mechanische pesterijen straft de P2000 onmiddelijk af. Opent U namelijk de cassettehouder terwijl er nog een file openstaat, dan verschijnt "Geen cassette" en stopt de programma-afhandeling. Plaatst U dan netjes de cassette terug en typt U "CONT" dan is er verder niets aan de hand. Haalt U echter de cassette eruit terwijl de recorder aan het spoelen was, begin dan maar opnieuw. "Geen cassette" is normaal met ON ERROR GOTO af te vangen, U kunt dus een complete kermis programmeren om tegen dit soort mishandelingen van het systeem te protesteren.

Wanneer alle gegevens met PRINT# weggeschreven zijn, bent U nog niet klaar. Er staan in de meeste gevallen nog records in de buffer, want het zal zelden voorkomen dat de buffer precies vol raakt en nog net op de band gezet wordt. Met de opdracht

```
CLOSE #1
```

dwingt U de computer zijn buffer weg te schrijven. Hij zet dan meteen een EOT-mark erachteraan om het einde van de cassette te markeren. U kunt hier nog gewoon andere dingen achter zetten, maar ze verdwijnen weer als sneeuw voor de zon wanneer U dezelfde file nog eens opent om te schrijven. Let daar op! De file komt op de band onder de opgegeven naam, met als extensie "SEQ".

Bij het testen van een programma kan het gebeuren dat er na het openen van een file een programmafout optreedt. De P2000 komt dan terug in de directe stand met al zijn variabelen en ook de filebuffer nog intact. De file staat nog steeds open. Geeft U nu opnieuw RUN of START, dan wordt deze file uit veiligheids-overwegingen NIET vanzelf gesloten of gewist! Zodra het OPEN-statement gepasseerd wordt komt de melding "File still open". U dient eerst CLOSE te geven.

Ook kan het gebeuren dat U helemaal niet wilt dat er iets op de cassette geschreven wordt. Opent U de houder, dan komt "Geen cassette" en zolang de P2000 geen cassette krijgt blijft hij mokken. Steeds zal de foutmelding terugkomen. Alleen CLOSE geeft de computer zijn zin, maar dat wilde U nou net niet. Voor dit soort crisis-situaties is de CLEAR-opdracht wat uitgebreid: hij werkt precies zoals bij de BASIC-NL maar wist ook een nog eventueel openstaande file. Overigens, voordat de P2000 bereid is zijn file te vergeten moet U er wel een cassette instoppen! Daar gebeurt dan verder niets mee, maar om te voorkomen dat per ongeluk gegevens verloren gaan is het wel nuttig.

Lezen

Als U een file netjes afgesloten hebt, zal de band weer teruggespoeld worden naar het begin en kunt U de weggeschreven gegevens terug inlezen.
Daarvoor moet U eerst weer een file openen:

```
10 OPEN "I",#1,"Bestandsnaam"
```

Het enige verschil is de "I": nu moet de buffer geschikt gemaakt worden voor Input. Hij krijgt weer nummer 1 toegewezen, en het eerste blok van de file waarvan de naam begint met een "B" wordt in de buffer geladen.
Het eerste record van de zojuist geopende file wordt nu gelezen met

```
20 INPUT #1, A$
```

Eigenlijk spreekt dit voor zichzelf: het eerste (of eerstvolgende) record uit de buffer wordt in A$ gestopt. Als dit record van het verkeerde type blijkt te zijn, in dit geval geen string maar een getal, dan komt de foutmelding "Type mismatch". U dient dus goed uit te kijken of de geschreven records en de gelezen records (dus de PRINT#- en de INPUT#-opdrachten) overeenstemmen!
Wordt het einde van de buffer bereikt, dan gaat de recorder weer draaien en haalt de P2000 het volgende blok van de file binnen. Ook bij INPUT# mogen verschillende variabelen, gescheiden door komma's, opgegeven worden.

Alles gaat goed totdat het einde van de file bereikt wordt. Wanneer het laatste record gelezen is en er volgt nog een INPUT#, dan verschijnt de fout "Input past end" en stopt de P2000 met zijn programma. De computer moet dus te weten kunnen komen of er nog wel iets in de file zit. Dat gaat met een speciale hulpvariabele, namelijk EOF (End Of File).
Als deze variabele "waar" is, dan is het einde van de file bereikt. Is hij "niet waar", dan volgt er nog minstens een record. Ter verduidelijking geef ik hierbij een programmaatje dat de gegevens van een file inleest en op het scherm zet. De file "Bestand" moet van tevoren gevuld worden met strings.

```
10 OPEN "I",#1,"Bestand"
20 WHILE NOT EOF
30  INPUT #1, A$
40  PRINT A$
50 WEND
60 CLOSE #1
70 END
```

Ook hier weer een voorbeeld van het gebruik van WHILE/WEND: zolang als het einde van de file niet bereikt is blijft de computer records lezen en printen. Nadat het laatste record gelezen is moet de file nog afgesloten worden. Hoewel de P2000 natuurlijk "weet" dat hij het laatste record gehad heeft en dus de file automatisch zou kunnen sluiten moet U het toch zelf doen. Dit is uitsluitend uit veiligheidsoverwegingen.

De variabele EOF is een gewone variabele met de eigenlijke naam "EO". Het is duidelijk niet de bedoeling zelf hiermee te gaan spelen! Het type van EOF is single precision (dat is namelijk het standaardtype van BASIC) en dat dient ook zo te blijven. Let dus een beetje op met DEF(type), anders gebeuren er ongelukken.

Het kan gebeuren dat U halverwege het lezen van een file opnieuw wilt beginnen. U typt dan gewoon CLOSE #... en de P2000 veegt onmiddellijk zijn buffer leeg en spoelt de band terug. Daarna kunt U weer een OPEN-opdracht geven om het bestand opnieuw in te lezen.

De gehele structuur van een sequentieel bestand vertoont grote overeenkomsten met de READ/DATA-constructie van BASIC. In beide gevallen heeft U te maken met een lange rij gegevens (records) die *een voor een* gelezen kunnen worden. Juist daar liggen ook de problemen: sorteren van een sequentiele file gaat niet. De hele file moet dan in een array gestopt worden. Een array is wel te sorteren, omdat daar de records willekeurig toegankelijk zijn (hoeft niet een voor een). Na het sorteren kunt U het array dan weer in de vorm van een sequentiele file op cassette zetten.
Dezelfde procedure, maar dan zonder het sorteren, moet U ook volgen om een bestand te kopieren. Wanneer de P2000 twee recorders zou hebben kon het wel direct, maar nu moet het door tussenkomst van een array.

Uit het bovenstaande blijkt al dat sequentiele files niet zo geschikt zijn om achteraf even te wijzigen. Daarvoor kunt U beter een array nemen, eventueel met de Stringsaveroutines erbij. Dat kost wel veel tijd en geheugen. Discsystemen en ook RDOS hebben de mogelijkheid van "Random acces" files, waarbij willekeurig gelezen en geschreven kan worden door de hele file heen. Met een cassetterecorder is dat vrijwel niet te doen zonder er het hele geheugen aan op te offeren.

Verlengen van een file die al op cassette staat

Het verlengen van een al aangelegde file is bij andere BASIC-versies meestal niet mogelijk. Bij Hoppie's Extended BASIC kan het echter wel! U opent dan de file met:

```
OPEN "A",#1,"Bestandsnaam"
```

De "A" staat voor "Append", verlengen dus. De P2000 leest de genoemde file in, draait de band een blok terug en zet de recordwijzer na het laatste geschreven record. Daarna is alles precies zoals bij een gewone outputfile. Ook hier geldt weer dat alles na de file op de band gewist wordt.

Wanneer U een OPEN-opdracht geeft terwijl er al een file openstaat (dus zonder CLOSE), dan geeft de P2000 de foutmelding *File still open*.

Procedures

Met Hoppie's Extended BASIC kunt U zogenaamde *procedures* definieren. Dat zijn een soort subroutines, maar er is een heel belangrijk verschil: *aan een procedure mogen gegevens meegegeven worden.* Deze gegevens hoeft U niet in speciaal daarvoor bestemde variabelen te zetten, maar U kunt ze meteen intypen achter de procedurenaam. Op enkele kleine dingen na is er dus geen verschil tussen een standaard BASIC-procedure zoals PRINT en een zelfgemaakte procedure die bijvoorbeeld SORTEER heet.
Alle opdrachten die de procedure moet uitvoeren om een bepaalde taak te verrichten worden op 1 plaats verzameld. In feite houdt zo'n procedure een compleet BASIC-programma in, dus het programmeert precies hetzelfde. Alleen moet U het begin en het einde van de procedure netjes markeren en natuurlijk ook de naam en de mee te nemen gegevens opgeven. Aan de hand van een voorbeeld "SORTEER" zal het een en ander worden uitgelegd.

Het begin van een procedure wordt als volgt aangegeven:

```
10 DEF PROC "Sorteer" (gegevens)
```

Dus: "*def*inieer de *procedure* Sorteer". De gegevens kunnen op een aantal verschillende manieren meegegeven worden, hierover hebben we het later nog. Na de definitie volgen gewone BASIC-opdrachten die op de gegevens een bewerking uitvoeren. De procedure wordt afgesloten met

```
90 PEND
```

wat staat voor *procedure end*. PEND kunt U vergelijken met RETURN, maar U mag ze niet verwisselen: een procedure is echt iets anders als een subroutine.

Een procedure wordt als volgt aangeroepen vanuit het programma:

```
450 ^"Sorteer" (gegevens)
```

Op het pijltje en de aanhalingstekens na dus een gewone BASIC-opdracht!

Nu zijn er twee manieren om een gegeven (getal of string) aan zo'n procedure op te geven: "enkele reis" of "retour". De procedure kan een gegeven nodig hebben zonder het te veranderen, alleen input dus, of kan nu juist het gegeven aanpassen. Een procedure die bijv. een op te geven aantal seconden wacht heeft alleen input -het aantal seconden- nodig, een sorteerprocedure levert de input gesorteerd weer terug, als output dus. Wanneer er alleen input gevraagd wordt mag U ook een expressie zoals 45+7-A opgeven, want er komt toch niets terug. Maar als er *wel* output terugkomt, *moet* U een losse variabele zoals A$ gebruiken omdat Hoppie's BASIC anders niet weet waar die ouput moet blijven!

Om deze fundamenteel verschillende mogelijkheden netjes gescheiden te houden werkt Hoppie's BASIC als volgt:
wanneer U de verschillende gegevens scheidt door *komma's*, dan neemt de P2000 aan dat alle gegevens zowel voor input als voor output zijn. U bent dan dus verplicht om enkelvoudige variabelen te gebruiken. Voorbeeld:

```
570 ^"Sorteer" (A$,B$,C$,D$,E$,F$)
```

U kunt nu aangeven dat bijv. D$, E$ en F$ *alleen voor input* dienen met:

```
570 ^"Sorteer" (A$,B$,C$; D$,E$,F$)
```

Na de punt-komma volgen de gegevens die alleen voor input gebruikt worden. De punt-komma scheidt dus de velden voor in- en output en de velden voor input.

Na de punt-komma mag U expressies gebruiken, dus

```
570 ^"Sorteer" (A$,B$,C$; "Een","Twee","Drie")
```

is toegestaan. Zoals gezegd mogen alle gegevens zowel getallen als strings zijn.
Wanneer *alle* gegevens *alleen voor input* zijn, dan kunt U de lijst beginnen met een punt-komma:

```
360 ^"Wacht" (;30)
```

De procedure "Wacht" zal zich nu helemaal niet met het gegeven TIJD bemoeien, hij kijkt er alleen maar naar. Daarom mag TIJD ook een expressie zijn!

Als gegeven mogen zowel normale variabelen als array-elementen zoals A(8) gebruikt worden. Maar U kunt ook hele arrays in 1 keer aan een procedure opgeven, door een @ voor de arraynaam te plaatsen. Om het stringarray A$, dat de elementen 0 t/m 100 heeft, te sorteren dus bijvoorbeeld

```
660 ^"Sorteer" (@A$)
```

Moet het array bijvoorbeeld op kolom vier geprint worden, dan zou U een procedure kunnen maken die als volgt aangeroepen wordt:

```
530 ^"Print array" (;@C,4)
```

Zowel het array C als het kolomnummer 4 zijn alleen inputgegevens, ze worden dus voorafgegaan door een punt-komma. Daarom mag het 4 zijn i.p.v. KOLOM. Maar

```
530 ^"Print array" (@C;4)
```

mag ook, als de procedure tenminste van array C afblijft.

Misschien lijkt het systeem wat ingewikkeld, maar na enige oefening werkt het prima. Wanneer U een foutje maakt met de komma of punt-komma, dan komt de waarschuwing *Illegal function call* of *Syntax error*. Deze foutmeldingen verschijnen ook als de gegevens in de aanroep van een procedure niet "passen" op die van de procedure-definitie.

In de gegevenslijst van de procedure zelf, dus na DEF PROC, kunnen heel andere variabelen staan als bij de aanroep (dat is nu juist de kracht van een procedure!). Zodra de procedure aangeroepen wordt zoekt Hoppie's BASIC de juiste procedure op aan de hand van de naam, en gaat dan de variabelen in de aanroep-opdracht (na het pijltje) kopieren naar de variabelen in de DEF PROC-opdracht. Hierna wordt de procedure uitgevoerd, die natuurlijk gebruik maakt van de variabelen en evt. arrays die zijn opgegeven bij DEF PROC. Het is mogelijk om ook "externe" variabelen te gebruiken, die dus *niet* in de procedure zelf zijn aangemaakt, maar daarmee moet U wel voorzichtig zijn omdat ze het hele proceduresysteem eigenlijk nutteloos maken. Wanneer de opdracht PEND gepasseerd wordt kopieert de P2000 alle variabelen (*tot aan de punt-komma*) weer terug in de gegevens "na het pijltje" en gaat dan verder met het hoofdprogramma. Om dat kopieren mogelijk te maken moeten de verschilende gegevenslijsten natuurlijk precies op elkaar passen, dus string op string en getal (van het juiste type!) op getal. Bij afwijkingen volgt eén foutmelding.
Voordat ik een compleet uitgewerkt voorbeeld laat zien, nog even wat extra informatie:
Bij een onvolledige procedure-definitie wijzen de fouten *DEFPROC without PEND* en *PEND without DEFPROC* U op ontbrekende instructies. Zowel DEFPROC als PEND moeten weer vooraan op een regel staan, bij fouten verschijnt *Wrong place*. Wanneer een PEND-opdracht gepasseerd wordt terwijl een WHILE/WEND-lus nog niet is beeindigd (of omgekeerd), dan waarschuwt de P2000 met *Not yet finished*, en als U een onbekende procedurenaam opgeeft komt *Label not found*. Dat laatste geldt overigens ook voor de opdrachten GOTO "Label" en GOSUB "Label".
Nesten van procedures is toegestaan (maximaal tien), maar helaas bleek het ondoenlijk *recursie* in te bouwen (het aanroepen van de procedure door zichzelf). Dat komt door de manier van boekhouden van de BASIC: gescheiden variabelenvelden zijn onmogelijk en er zou trouwens een enorm geheugengebrek ontstaan.

Het volgende voorbeeld verduidelijkt het gebruik van een procedure aan de hand van een simpele sorteerroutine. Het gebruikte algoritme is "bubble sort", nu niet bepaald een efficiente manier maar lekker kort en daardoor als voorbeeld goed geschikt.

```
10 CLEAR 1000: REM Reserveer voldoende stringruimte
20 GOSUB "Initialitatie"
30 ^ "Vul array" (@A$)
40 PRINT "Ongesorteerd:"
50 ^ "Print array" (:@A$)
60 ^ "Sorteer" (@A$)
70 PRINT "Gesorteerd:"
80 ^ "Print array" (:@A$)
90 END

100 "Initialisatie"
110  TRUE=-1: FALSE=0
120  DIM A$(10), B$(10)
130  PRINT CHR$(28) "Demonstratie van sorteerprocedures"
140 RETURN

200 DEF PROC "Vul array" (@B$)
210  PRINT "Tien woorden ingeven s.v.p.:"
220  FOR TELLER=1 TO 10
230   LINE INPUT B$(TELLER)
240  NEXT TELLER
250 PEND

300 DEF PROC "Print array" (:@B$)
310  FOR TELLER=1 TO 10
320   PRINT B$(TELLER)
330  NEXT TELLER
340 PEND

400 DEF PROC "Sorteer" (@B$)
410  C=FALSE: REM Beginvoorwaarde
420  WHILE C=FALSE
430   C=TRUE
440   FOR TELLER=2 TO 10
450    IF B$(TELLER-1) > B$(TELLER)
        THEN SWAP B$(TELLER-1), B$(TELLER):
        C=FALSE
460   NEXT TELLER
470  WEND
480 PEND
```

Misschien is er wat overdadig van procedures gebruik gemaakt, maar het programma is wel heel goed leesbaar! Omdat dit zo'n simpel voorbeeld is hadden de procedures ook als subroutines uitgevoerd kunnen zijn, dus met vaste variabelen en arrays, maar daar ging het niet om. Bij dit voorbeeld staan alle procedures achteraan, maar het is meestal de gewoonte om ze juist vooraan te zetten. Bij de normale programma-afloop slaat Hoppie's BASIC namelijk de stukken tussen DEF PROC en PEND gewoon over, die worden alleen gebruikt bij een rechtstreekse aanroep van de procedure.

Extra mogelijkheden bij het werken met de computer

We hebben nu alle extra programmainstructies van Hoppie's Extended BASIC gehad en komen zo toe aan de kleine dingen die de P2000 nog gebruikersvriendelijker maken.

Is het U wel eens opgevallen dat een programma veel prettiger las toen U het intypte dan achteraf, na LIST? Dat komt omdat de P2000 zijn LISTings in hoofdletters op het scherm zet. Uit psychologische onderzoekingen is gebleken dat kleine letters veel prettiger lezen dan hoofdletters (kijk maar eens naar de richtingsborden van de ANWB- allemaal onderkastletters!). Hoppie's Extended BASIC biedt de mogelijkheid de P2000 op alleen onderkast over te schakelen. Natuurlijk mag dit niet zomaar, want op zijn tijd zijn hoofdletters (bovenkast) echt wel nuttig. Daarom werkt het grapje alleen maar in de directe stand en zitten er foefjes in zodat de normale gang van zaken nauwelijks verstoord raakt. U schakelt de onderkastconversie in met

LOWER

en weer terug naar hoofdletters met

UPPER

Bij bepaalde EDIT-opdrachten kunnen er ineens hoofdletters omgezet worden in kleine letters. U voorkomt deze problemen door ook op het einde van een regel een sluitteken te gebruiken. Blijkt het geval erg halsstarrig, zet dan de computer even op UPPER en het leed is geleden.

Als laatste extra instructie is er een printerbesturings-opdracht bijgekomen. Het komt vaak voor dat een programma goed loopt en keurig zijn resultaten op het beeldscherm zet. Maar wanneer U nu eens een uitdraai op de printer wilt hebben, kunt U met de hand of met een hulproutine alle PRINT's veranderen in LPRINT! En dan verschijnt er weer niets op het scherm.
Aan deze ellende komt een eind na het commando

PRON

wat een afkorting is van PRinter ON. Als de printer on-line staat worden gewoon alle tekens die naar het scherm gestuurd worden, doorgelust naar de printer. Hierop is een uitzondering, namelijk de CHR$(12) en CHR$(28)-codes die het scherm schoonmaken. In een normaal programma komen die zo vaak voor dat het meters papier gaat kosten, en elke keer als U in de directe stand gedachtenloos de knop rechtsboven indrukt draait er weer een vel uit... (en dat is natuurlijk leuk speelgoed in de klas). Daarom worden deze codes omgezet in linefeeds. Wilt U ze toch perse meenemen, dan dient U de instructie

PRON*

te gebruiken, die alles ongewijzigd doorlaat.
Het synchroon meeprinten wordt beeindigd na een printerfout of na

PROFF

Bovenstaande commando's zijn eigenlijk niet bedoeld om in de directe stand een soort logboek bij te houden, maar om snel een hard copy te maken van programma-uitvoer. Vooral bij editten van een regel kan de printer erg vreemd gaan doen...

Het kan vaak erg handig zijn een al ingetypte en uitgevoerde opdracht met een druk op de knop weer tevoorschijn te toveren, zodat U de hele rataplan niet nog eens in hoeft te typen. Ook bij het hernummeren van kleine stukjes programma kan zo'n edithulp nuttig zijn. Daarom is in Hoppie's BASIC een kleine screen editor ingebouwd. Zodra U op de knop "accent aigu" drukt (geeft normaal 1/4) kunt U met de cursor over het hele scherm lopen en overal veranderingen

aanbrengen. Wanneer U daarna op ENTER drukt wordt de regel waar de cursor op dat moment op staat van het scherm gelezen en door de P2000 opnieuw ingetypt. Het lijkt dan net alsof U het zelf ingetypt heeft. Door dan nogmaals op ENTER te drukken wordt de instructie uitgevoerd.
De screen editor is een edit*hulp*, dus gebruik hem met mate. Als U namelijk te gretig over het scherm jaagt ontstaat er een grote puinhoop en is het voordeel meteen weg.

Foutmeldingen

Hieronder volgt een opsomming van alle fouten die bij het werken met Hoppie's Extended BASIC kunnen optreden.

Syntax error	Spelfout, of een niet passende set gegevens bij een procedure.
RETURN without GOSUB	Zie de BASIC-handleiding.
Illegal function call	Fout bij een procedureaanroep. Zie Syntax error.
Type mismatch	U gebruikt een verkeerd type variabele bij een INPUT#-opdracht, bijv. een string i.p.v. een getal.
Geen cassette	De cassettehouder is geopend terwijl er nog een seriele file openstond. Sluiten en CONT typen.
File still open	Er wordt een poging gedaan een file te openen terwijl de vorige file nog niet afgesloten is (met CLOSE).
Input past end	De file was al leeggelezen terwijl er nog een INPUT#-opdracht volgt.
File not open	De file is nog niet geopend of op de verkeerde manier geopend (bijv. om uit te lezen bij een PRINT#-opdracht).
Label not found	Bij een GOTO of GOSUB "Label"-opdracht of een procedureaanroep is er een onbekende naam opgegeven.
Wrong place	Een van de opdrachten WHILE, WEND, DEF PROC, PEND of een label staat niet vooraan op een regel.
WEND without WHILE/WHILE without WEND	Spreekt voor zichzelf.
PEND without DEFPROC/DEFPROC without PEND	Spreekt voor zichzelf.
Not yet finished	U doet een poging een WEND uit te laten voeren terwijl de procedure nog niet beeindigd is met PEND, of omgekeerd: PEND zonder dat de WHILE/WEND-lus is afgesloten. Beide fouten wijzen op een verkeerde programmastructuur.

De nieuwe fouten hebben geen eigen nummer, zodat ze niet met ON ERROR GOTO zijn af te vangen. Omdat alle fouten altijd wijzen op fouten in het programma is dat ook niet nodig: treedt er zo'n fout op, dan *moet* U het programma verbeteren.

Zoals U gezien heeft wordt de BASIC van de P2000 aardig opgekrikt. De instructies zijn meestal niet echt "nieuw" (het file system bijvoorbeeld is standaard MicroSoft) en daarom zult U er waarschijnlijk weinig moeite mee hebben. De procedures zijn ingewikkelder, maar daar is ook een voorbeeldprogramma bij gegeven.

Mochten er onverhoopt complicaties optreden bij het gebruik van Hoppie's Extended BASIC, dan zou ik het zeer op prijs stellen als U mij daarover zou willen inichten. Ook wanneer U opmerkingen of aanvullingen heeft, neem contact op! Zonder te overdrijven kan ik zeggen dat vrijwel alles mogelijk is, voor zover de hardware geen beperkingen oplegt. Het is echter niet mijn bedoeling zoveel mogelijk instructies bij te maken, enig realisme is in dit opzicht zeker noodzakelijk.

Ik hoop dat dit programma zijn weg naar de gebruiker zal vinden en dat het nuttig gebruikt zal worden. Aan mij zal dat niet liggen!

Valkenswaard, 2-11-1986
Jeroen Hoppenbrouwers
Wilhelminapark 8
5554 JE VALKENSWAARD
Telefoon: (04902)-13808
Vidibus: 400021237

Aanvulling handleiding Hoppie's BASIC 3.1

Bij de versie 3.1 is alles hetzelfde als bij versie 3.0. Er zijn echter twee instructies bijgekomen die het gestructureerd programmeren aanzienlijk vereenvoudigen. Deze instructies zijn *BEGIN* en *END*, bekend vanuit PASCAL en bijna de spil waar deze taal om draait.
In Hoppie's BASIC zijn wat consessies gedaan aan de plaats waar bepaalde opdrachten mogen staan. WHILE, WEND, DEFPROC etc. *moeten* altijd aan het begin van een nieuwe regel staan. Dit is gedaan om het opzoeken van deze instructies eenvoudig te houden en dus een (veel) sneller programma te krijgen.
BEGIN en END vormen hierop geen uitzondering zodat U ze altijd vooraan een regel moet plaatsen.
Overigens is dat typografisch gezien ook de enig juiste plaats, want dergelijke structuurbepalende opdrachten horen thuis op een duidelijke, overzichtelijke plaats en dat is dus vooraan een regel.

Waarom BEGIN en END?
Zoals u weet moeten bij een IF-THEN-opdracht zowel alle instructies na THEN als alle instructies na een eventuele ELSE *op een en dezelfde regel staan*. Dat is soms geen probleem, maar vaak wordt zo'n IF-THEN-(ELSE)-regel een knoeiboel en het komt ook geregeld voor dat het hele zaakje niet eens op 1 regel past. Er moet dan weer een GOTO van stal gehaald worden en daarmee wordt weer een flink stuk overzichtelijkheid ingeleverd.
Met BEGIN en END-opdrachten kunt u nu zo'n IF-THEN-(ELSE) verdelen over meerdere regels zodat de structuur erg duidelijk blijft. Er is verder nauwelijks iets over te vertellen, voorbeelden spreken voor zichzelf. Let wel op dat u precies dezelfde plaats van de opdrachten gebruikt, anders loopt het programma in de fout.

```
 10 IF A=0 THEN ... (ELSE ...)                    ; Normale structuur

 20 IF A=0 THEN                                   ; THEN met BEGIN-END
 30  BEGIN                                        ;  ELSE normaal
 40   ...
 50   ...
 60  END
(70 ELSE ...)

 80 IF A=0 THEN ... ELSE                          ; ELSE met BEGIN-END
 90  BEGIN                                        ;  THEN normaal
100  ...
110  ...
120  END

130 IF A=0 THEN                                   ;Beide met BEGIN-END
140  BEGIN
150  ...
160  END
(170 ELSE
180  BEGIN
190  ...
200  END)
```

Jeroen Hoppenbrouwers, 03-01-1987