)TA 1334 maart 1982

Instituur voor Cultuurtechniek en Waterhuishouding
Wageningen

ASPECTEN VAN INFORMATIEVERWERKING

30

Een inleiding in de programmeertaal BASIC

ir. J.G. Wesseling

I

NOTA 1334 maart 1982

Instituur voor Cultuurtechniek en Waterhuishouding
Wageningen

ASPECTEN VAN INFORMATIEVERWERKING

30

Een inleiding in de programmeertaal BASIC

ir. J.G. Wesseling

I N H O U D

INLEIDING

Deze nota is bedoeld voor hen die hun rekenkundige problemen met behulp van de programmeertaal BASIC (Beginners Allpurpose Symbolic Instruction Code) op willen lossen op een computer. Het feitelijk grote voordeel van BASIC ten opzichte van bijvoorbeeld FORTRAN is dat voor BASIC geen kennis van het operating system nodig is. Door de eenvoudige commando's kan iedereen na een zeer korte tijd al met BASIC werken. Vooral voor hen die al eens met een programmeerbaar rekenmachientje hebben gewerkt zal deze taal geen moeilijkheden opleveren.

In de eerste hoofdstukken van deze nota wordt een kort overzicht gegeven van de ontwikkeling en van de opbouw van een digitale computer. Ondanks het feit dat voor het werken met BASIC geen kennis van het operating system vereist is zal toch in het kort even op de funktie van het operating system worden ingegaan. Dit is alleen bedoeld om de werking van de computer te verduidelijken. In de daaropvolgende hoofdstukken zal dan op de programmeertaal BASIC worden ingegaan. Dit gebeurt aan de hand van eenvoudige voorbeelden. Tenslotte worden dan subprogramma's en speciale funkties van BASIC besproken. In appendices zullen enkele voor verschillende computers specifieke commando's worden besproken. Er is wel van uitgegaan dat de programma's op DIGITAL apparatuur draaien (PDP, DEC, VAX), daar bijvoorbeeld de CYBER geen BASIC interpreter maar een BASIC compiler heeft. Op het verschil hiertussen wordt in deze nota nog nader ingegaan.

Er wordt niet naar gestreefd een volledige cursus te geven, maar slechts wat globale informatie en een overzicht van de mogelijkheden van BASIC. De geïnteresseerde lezer zal ook hier bemerken dat, als hij zijn eigen problemen op de computer probeert op te lossen, er

meer aspekten aan het programmeren zitten dan in een inleidende cursus kunnen worden besproken. Het is daarom raadzaam na enige tijd met BASIC gewerkt te hebben de uitvoeriger handboeken over deze taal eens door te nemen.

1. DE HISTORISCHE ONTWIKKELING VAN DE COMPUTER

Afgezien van de al veel oudere telramen, is het oudst bekende hulpmiddel bij het rekenwerk het mechanische rekenapparaat van de wiskundige Pascal uit 1652. Dit apparaat kon alleen optellen en aftrekken. Leibniz gebruikte het in 1673 als basis voor zijn rekenmachine waarmee hij ook kon vermenigvuldigen en delen. De eerste rekenmachine waarbij gebruik werd gemaakt van elektriciteit werd in 1937 door Howard Aiken van de Harvard University ontworpen. Deze Automatic Sequence Controlled Calculator (ASCC) werd in 1944 in samenwerking met IBM voltooid. Het was een enorm elektromagnetisch apparaat, vijftien meter lang en 2,5 meter hoog en het bevatte telwielen en 3000 relais. Voor die tijd een razendsnel apparaat: vermenigvuldigen van twee getallen kostte 6 seconden, delen 12 seconden. In 1945 hadden Eckert en Machly van de Moore School of Electrical Engineering in opdracht van de amerikaanse regering de ENIAC (Electronical Numerical Integrator and Computer) gebouwd. Deze machine bevatte maar liefst 18000 elektronenbuizen, 70 000 weerstanden, 10 000 condensatoren en 6000 schakelaars. Het apparaat was dan ook 30 meter lang en 3 meter hoog. De machine was echter behoorlijk snel: 0,2 en 2,8 milliseconden voor respektievelijk optellen en vermenigvuldigen van twee getallen. Het energieverbruik van dit apparaat was te vergelijken met dat van een stoomlocomotief. Toch was dit apparaat nog geen computer in de eigenlijke zin van het woord. Het apparaat werd namelijk bestuurd door bedrading die voor ieder programma opnieuw moest worden aangelegd. De eerste computer die werd bestuurd door een programma in het geheugen was de Electronic Discrete Variable Automatic Computer (EDVAC) die eveneens aan de Moore School werd gebouwd en in 1952 klaar kwam.

De eerste commerciële computer was de UNIVAC waarvan de eerste in 1950 werd afgeleverd voor het verwerken van de volkstelling in de VS.

Ook IBM ging zich nu met de computermarkt bemoeien en kwam in 1953 met de 701. Intussen was in 1947 de transistor door Bardeen, Brattain en Schockley uitgevonden. Omdat een transistor veel kleiner was dan de elektronenbuis, veel minder energie gebruikte en veel betrouwbaarder was, begon hij langzamerhand de elektronenbuis te vervangen. De eerste volledig getransistoriseerde computers verschenen in 1959 op de markt, de NCR-GE 304 en de IBM 1401. Maar de ontwikkeling was nog niet beëindigd: in 1959 ontdekte Kilby van Texas Instruments dat het mogelijk was verschillende elektronische componenten op een klein plaatje silicum aan te brengen en hiermee was het geïntegreerde circuit of IC geboren. Aanvankelijk kon er maar een beperkt aantal componenten op een chip worden aangebracht. Mede onder druk van de bewapeningstechnologie en de ruimtevaart die steeds kleinere elektronische schakelingen vroegen ontwikkelde deze technologie zich snel. Nu is het reeds mogelijk vele tienduizenden componenten op enkele vierkante millimeters aan te brengen.

De eerste computer met IC's was de PDP-8/I die in 1968 op de markt kwam. De grote voordelen waren weer: grotere snelheid, goedkoper, lager energieverbruik en een stuk kleiner. Doordat men steeds meer componenten op een chip kon onderbrengen, werd het mogelijk een complete centrale verwerkingseenheid op een plaatje silicum te bakken. Dit was de zogenaamde microprocessor, waarvan de eerste werd uitgebracht door Intel in 1971. Door het aansluiten van geheugenchips en schakelingen voor de in- en uitvoer kan men met behulp van de microprocessor een volwaardige, maar in omvang kleine computer maken. Alle huis- of hobbycomputers zijn op deze manier gebouwd.

Met deze ontwikkeling is men nog verder gegaan. Zo heeft de supersnelle CRAY I computer een cyclustijd van 12,5 nanoseconden, ($12{,}5*10^{-9}$ seconden ofwel 0,0000000125 seconden). Deze computer kan maar liefst 80 miljoen instrukties per seconde uitvoeren. De snelheid van verwerking komt nu al aardig dicht bij het maximum. Dit wordt namelijk bepaald door de snelheid waarmee de elektronen hun weg afleggen, die $3*10^{8}$ m/s bedraagt. Op het ogenblik is men aan het experimenteren met supergeleiding. Dit treedt op als men materialen afkoelt tot nabij het absolute nulpunt (273°C onder nul). Men is hierbij bezig met zogenaamde Josephson schakelaars, die nu al

schakelsnelheden hebben bereikt van 15 pico seconden ($15*10^{-12}$ seconden). Dat wil zeggen dat ze meer dan 6,5 biljoen keer per seconde kunnen schakelen.

2. OVER DE OPBOUW VAN EEN COMPUTER

Een computer kan men opgebouwd denken uit de volgende elementen (zie fig. 1):

a. De processor. Dit is bij de mini- en microcomputers meestal een enkele processor of processorkaart. Enkele voorbeelden van zo'n processor zijn de Z80, de 6800, de 8080 en de 8085. Een voorbeeld van een processorkaart is de LSI-11, waarop bijvoorbeeld de PDP-11/03 is gebaseerd.

b. Een geheugen. Dit is een elektronische schakeling die in staat is signalen als 0 en 1 te onthouden (0 of +5 V). Door het kijken naar een bepaalde serie geheugenplaatsen (adres) kan de processor dan de inhoud hiervan (data) opvragen en gebruiken of veranderen.

   Er zijn verschillende soorten geheugens. Men is begonnen met het ringkerngeheugen, dat bestaat uit kerntjes waardoor een stroompje wordt gestuurd waardoor het gemagnetiseerd wordt.
   Tegenwoordig werkt men ook met IC's die geheugenschakelingen bevatten. Het is al mogelijk 131072 bits op een IC onder te brengen.

c. Achtergrond geheugen (Backgroundmemory). Daar het geheugen van een computer in grootte beperkt is, en er vaak veel data moet worden verwerkt, heeft men de achtergrond geheugens ontwikkeld. Deze werken iets trager dan wanneer men met het (werk)geheugen alleen kan werken, maar kunnen veel meer gegevens opslaan. Tegenwoordig wordt een achtergrond geheugen gecombineerd met een massaopslagapparaat (=mass storage device), bijvoorbeeld een schijf, of flexibele schijf (floppy disk). Deze units hebben een capaciteit die ligt tussen 128.10^3 en 512.10^6 bytes.

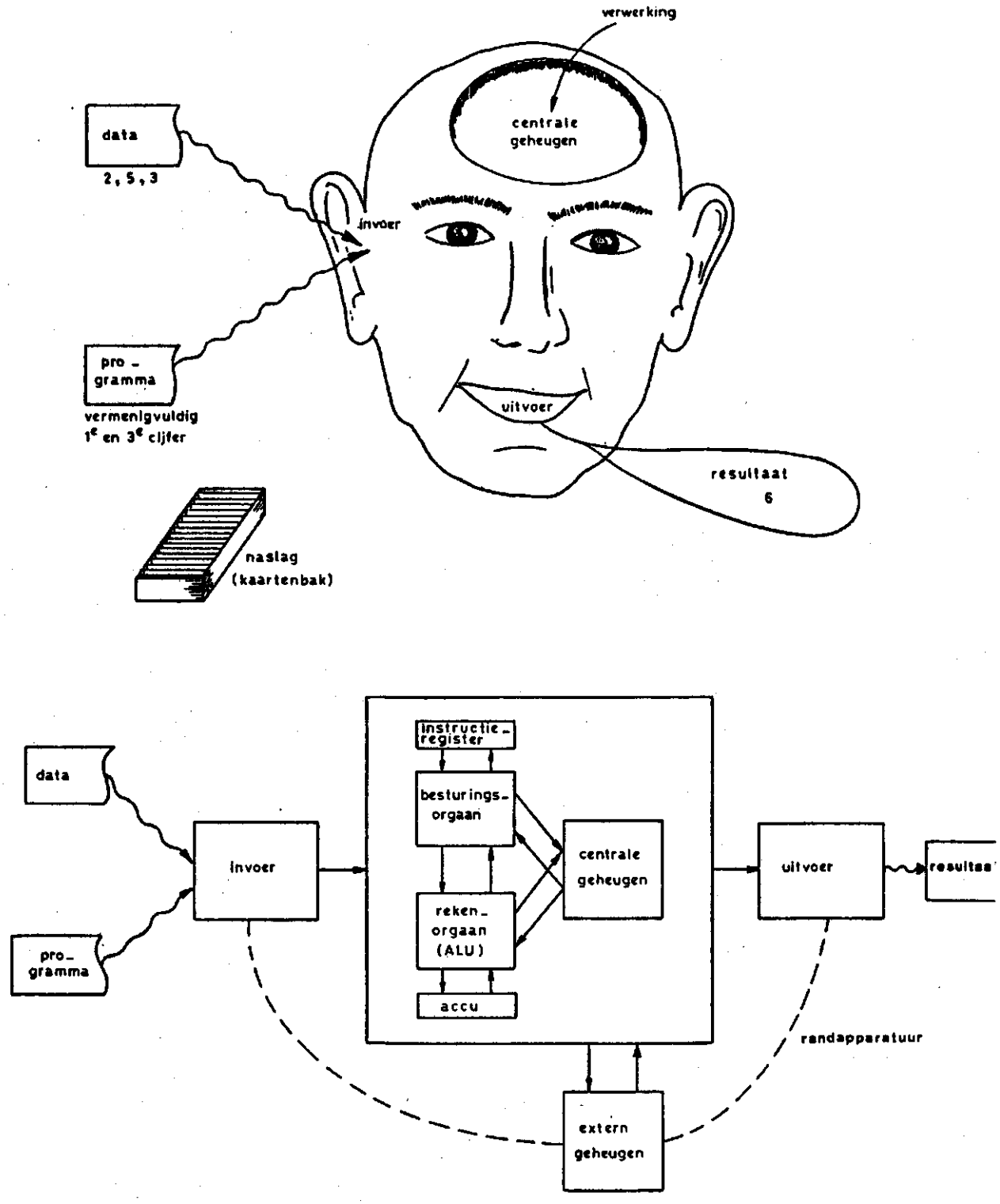


Fig. 1. Schematische opbouw van een microcomputer vergeleken met het menselijk brein. (Uit cursus micro-processors/micro-computers (1977))

d. In- en/of uitgangen van de processor, die dienen om data (=informatie) naar andere apparaten te zenden en om commando's van de randapparatuur naar de processor te brengen. Meestal worden deze uitgangen beschouwd als een geheugenplaats. Door op deze plaatsen aparte elektronische schakelingen (=interfaces) te zetten kan er een signaal naar het randapparaat worden gezonden om aan te geven dat er data klaar staan om weggezonden te worden.

e. De randapparatuur (=peripherals).
Dit zijn meestal tragere apparaten die de resultaten van de berekeningen opslaan in een vorm die gemakkelijk met andere computers uitwisselbaar is (ponskaarten, ponsbanden, magneetbanden) of die de resultaten weergeven in een vorm die voor de gebruiker te begrijpen is (printer, plotter, terminal). Ook invoerapparatuur (kaartlezer, ponsbandlezer, terminal) wordt als randapparatuur beschouwd.

Voor een uitgebreide beschrijving van bovenstaande elementen kan worden verwezen naar de cursus Computertechnicus-C (1980).

Alvorens in het kort te beschrijven hoe men zich de werking van een computer moet voorstellen zal eerst worden ingegaan op de datacodering bij computers. Daar een computer een elektronisch apparaat is, kent hij geen getallen of letters. Het enige waar hij mee werkt is spanning. Beter gezegd: het al of niet aanwezig zijn van een spanning. Dus kent hij maar 2 toestanden: 0 (geen spanning) of 1 (wel spanning). Deze toestand kan optreden in alle geheugenposities. Zo'n geheugenpositie noemt men een bit (=binary digit).

Bij ons decimale stelsel wordt gerekend met basis 10, bijvoorbeeld

$$3087 = 3*10^3+0*10^2+8*10^1+7*10^0=3000+0+80+7$$

Nu was al jaren bekend dat men niet vastzat aan de basis 10. Het is net zo goed mogelijk om de basis 2 te nemen (binair stelsel):

$$1011(2)=1*2^3+0*2^2+1*2^1+1*2^0=8+0+2+1 = 11(10)$$

waarbij het getal tussen haakjes de basis aangeeft, dus 1011 binair is 11 decimaal.

Om nu getallen te kunnen onthouden en ermee te kunnen rekenen is het geheugen van een computer onderverdeeld in 'vakjes' van 8 bits. Deze vakjes worden bytes genoemd en elk vakje heeft een eigen nummer (adres). Afhankelijk van de grootte van de computer kan het aantal bytes sterk variëren. Daar ook de adressen binair moeten worden verwerkt hangt de maximale geheugencapaciteit van een computer af van de maximale waarde die een adres kan bereiken. Het maximum aantal adressen bij een machine met een 16-bits adressering is $2^{16}-1 = 65535$ adressen. Meestal wordt de omvang aangegeven in veelvouden van 1k= 1024 dus 65535 bytes = 64 kbytes.

Een voorbeeld:

Stel de geheugenplaatsen 2045 en 2047 zien er als volgt uit:

adres	inhoud
2045	00000100
2047	00100101

De computer krijgt nu de opdracht om deze waarden bij elkaar op te tellen en in geheugenplaats 2047 te zetten. Dan ziet na de uitvoering van deze opdracht het geheugen er als volgt uit:

adres	inhoud
2045	00000100
2047	00101001

Voor het weergeven van letter, cijfers, leestekens, etc. wordt veelal gebruik gemaakt van de zogenaamde ASCII (American Standard Code for Information Interchange) code. Dit is een code die elke letter en leesteken een eigen combinatie van bitjes geeft. Zie appendix A. De meeste terminals en printers werken met deze code. Zo zal de naam JAN in ASCII in geheugenpositie X en volgenden worden gecodeerd als:

geheugenadres	code	letter
X	01001010	J
X+1	01000001	A
X+2	01001110	N

De opdrachten die een computer uit moet voeren staan in een deel van het geheugen. Een verzameling opdrachten heet een programma.

Voor het bijhouden van de plaats waar de volgende instruktie vandaan gehaald moet worden heeft de processor een aparte geheugenplaats, de programcounter (PC). Na het voltooien van een instruktie zal de processor de volgende instruktie opvragen. Deze instruktie staat op de geheugenplaats waarvan het adres zich in de PC bevindt. Zo is het voor de gebruiker mogelijk om spronginstrukties naar andere stukken programma te geven, eenvoudigweg door het veranderen van de inhoud van de PC. Nadat een instruktie uit het geheugen is gehaald, wordt de inhoud van de PC automatisch met 1 verhoogd. Zie voor een uitgebreide beschrijving de cursus Microprocessors Microcomputers (1977).

3. BASIC COMMANDO's

Zoals reeds in de inleiding vermeld, gebruikt de firma DIGITAL interpreters in plaats van compilers voor programma's geschreven in BASIC. In het vorige hoofdstuk is uiteengezet dat men van een taal die voor mensen leesbaar is, over moet gaan op een taal die voor computers te begrijpen is. Hiervoor zijn twee grote groepen programma's te onderscheiden die deze overgang (automatisch) mogelijk maken: compilers en interpreters. Dit zijn zelf ook weer programma's die onze programma's vertalen.

Een compiler vertaalt het hele programma in een keer. Voor het gebruik van een compiler moet eerst het programma op een randapparaat, bijvoorbeeld een schijf worden gezet. Om dit programma op schijf te kunnen zetten en er eventueel later weer veranderingen in aan te kunnen brengen, maakt men gebruik van weer een ander type programma: de (text)editor. Na het compileren moet dan vaak nog een derde programma worden gebruikt om alle delen van het programma op de correcte adressen te kunnen zetten: de linker. Deze zorgt tevens voor het koppelen van enkele standaardprogramma's voor in- en uitvoer. Dit is dus al een vrij ingewikkelde procedure die overigens voor het grootste deel automatisch door de computer wordt uitgevoerd.

Een interpreter daarentegen vertaalt een programma regel voor regel en voert dit tegelijkertijd uit. Door middel van eenvoudige

commando's kan vanuit de interpreter alles worden geregeld, zoals in dit hoofdstuk zal worden aangetoond. De programma's worden met een compiler iets sneller dan met een interpreter, maar hiervan zal bij normale (kleine) programma's niets te merken zijn. In fig. 2 is nogmaals het verschil tussen een compiler en een interpreter aangegeven.

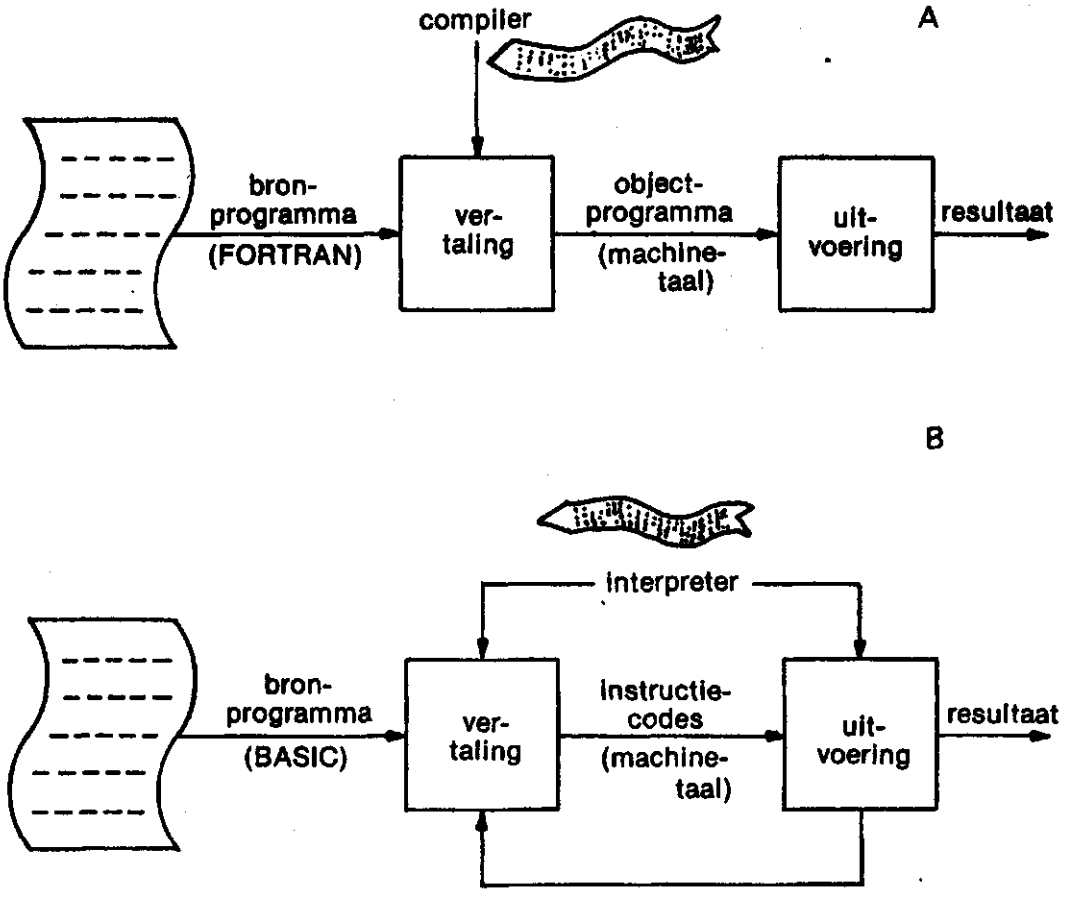


Fig. 2. Schematische voorstelling van de werking van een compiler (a) en een interpreter (b). (Uit Cursus Basic (1978))

Alvorens een kort overzicht van de commando's te geven die gebruikt moeten worden om BASIC te kunnen draaien zal, ter ver-

duidelijking, even dieper op de werking van een computer in worden gegaan. Na het starten van de computer wordt meestal automatisch het operating system (soms ook monitor genaamd) in het geheugen geladen (engels: bootstrap). Dit is een computerprogramma dat de commando's die de gebruiker intypt interpreteert en de betreffende handelingen door de computer laat uitvoeren. Als de gebruiker nu een programma wil draaien moet dit programma eerst van schijf in het geheugen worden geladen. Dit programma staat op een bepaald deel van de schijf.

Als in een programma wat veranderingen zijn aangebracht en men wil het programma weer op schijf zetten, zal de oude versie van het programma worden verwijderd of onder een andere naam worden opgeslagen. Ook datafiles (gebieden op schijf waar gegevens op staan) moeten kunnen worden ingelezen. Dit alles is de taak van het operating system. Hierover hoeft de gebruiker van de computer zich verder geen zorgen te maken, zolang hij de correcte commando's maar geeft.

Om het BASIC systeem te starten kan worden volstaan met een commando, dat echter voor de meeste operating systems anders is:

op.syst.	computer	commando
RT-11	PDP-11/03	R BASIC
RSX	PDP-11/70	BAS
DECOPS	DEC-10	BASIC

Hierna zal de computer het woordje READY geven, hetgeen wil zeggen dat men kan beginnen met nieuwe commando's in te typen of een nieuw programma in te typen. Dit programma, of het nu ingetypt wordt of van schijf wordt ingelezen, wordt in het geheugen geplaatst, zodat ermee gewerkt kan worden. Als het programma van schijf wordt gelezen (geladen), wordt het in zijn geheel naar het geheugen getransporteerd.

Bij apparatuur van de firma Digital worden programma's en data naar schijf weggeschreven onder namen met de vorm filenm.ext. De filenaam mag men zelf kiezen. Hij mag bestaan uit 1-6 alphanumerieke tekens, als de eerste maar een letter is. De extensie wordt meestal

gebruikt om aan te geven welk soort file het is: .BAS voor BASIC programma's, .DAT voor datafiles, .PAS voor Pascal-programma's etc.

In onderstaande commando beschrijvingen wordt gesproken van een BASIC-identificatie. Dat wil zeggen, dat, elke keer als een LIST of RUN commando wordt gegeven, eerst een regel verschijnt met het nummer van de BASIC-versie die momenteel in gebruik is en de datum van het moment waarop het RUN-commando wordt gegeven.

De meest belangrijke commando's in BASIC zijn:

NEW {filenm.ext} - er zal een nieuw programma worden ingetypt dat later moet worden weggeschreven naar schijf. Na het geven van een RETURN zal, indien geen filenaam is opgegeven, de computer naar de naam van het programma vragen.
Indien .ext wordt weggelaten, zal er worden aangenomen dat hier .BAS moet komen.

OLD {filenm.ext} - haal een programma dat onder de naam filenm.ext op schijf staat naar het werkgeheugen. Ook hier geldt weer dat .BAS wordt aangenomen.

SAVE {filenm.ext} - schrijf het programma weg naar schijf. Als filenm.ext wordt opgegeven zal het programma worden weggeschreven onder die naam. Als filenm.ext niet wordt opgegeven wordt het programma weggeschreven onder de naam waarmee het bij 'NEW' werd aangemaakt. Als het programma al bestond, zal bij dit commando een foutmelding op het beeld verschijnen. In dit geval zal namelijk het volgende commando moeten worden gebruikt:

REPLACE{filenm.ext} - vervang het programma met de naam filenm.ext door het programma in het werkgeheugen. Als

filenm.ext niet wordt gegeven wordt aangenomen dat de naam dezelfde is als waaronder het programma van schijf is gehaald.

SCRATCH	- maak het werkgeheugen schoon.
LIST	- geef het programma op de terminal.
LIST nnn-mmm	- geef de regels nnn tot mmm op de terminal.
RUN	- laat het programma uitvoeren.
RUNNH	- idem, maar zonder identificatie en datum.

Een meer volledig overzicht van de BASIC-commando's die gegeven kunnen worden op de bovengenoemde computers wordt gegeven in de appendices B, C en D.

4. GETALWEERGAVE EN EXPRESSIES

Alvorens op de eigenlijke programmeertaal BASIC in te gaan wordt in dit hoofdstukje de manier behandeld waarop BASIC getallen verwerkt. BASIC behandelt alle getallen, of het nu gehele (integer) of gebroken getallen (real) zijn als gebroken getallen. Dit wil zeggen dat alle getallen worden beschouwd alsof er een decimaalpunt in staat. Integers worden wel zo lang mogelijk als integers beschouwd, maar als ermee wordt gerekend worden zij als reals beschouwd.

BASIC accepteert drie getalvormen: integer, real en exponential. Bij een integer wordt het getal ingetypt zonder decimaalpunt. Bij een real getal wordt een decimaalpunt meegegeven, en bij exponentials wordt een basisgetal maal een macht van 10 gegeven. Van alle drie de mogelijkheden wordt een voorbeeld gegeven. Deze voorbeelden zijn allemaal uitdrukkingen voor hetzelfde getal:

1234
1234.
1234.0000
1234.E+00 (= 1234 10^0 = 1234 * 1 = 1234)
1.234E+3 (= 1.234 10^3 = 1.234 * 1000 = 1234)

In BASIC kan men werken met getallen tussen -10^{38} en 10^{38}, de kleinste waarde van de exponent is -38. Intern werkt de interpreter met getallen van 24 bits (3 bytes) nauwkeurigheid. Bij het uitprinten van getallen worden waarden

$.01 \leq n \leq 99999$

geschreven als real getallen. Waarden die buiten deze grenzen vallen worden als getal met exponent gegeven. Er worden van een getal maximaal 6 digits gegeven. Enkele voorbeelden:

getal	BASIC
1.	1.
.001	1.E-3
9.34567890	9.34568
1000000	1.E+6

BASIC kent de volgende wiskundige operaties:

symbool	voorbeeld	betekenis
+	A + B	tel A bij B op
-	A - B	trek B van A af
*	A * B	vermenigvuldig A en B
/	A / B	deel A door B
^	A ^ B	verhef A tot de macht B

Indien in een expressie een aantal operaties moeten worden uitgevoerd, houdt BASIC zich aan de normale rekenregels die ook in de wiskunde gelden. Dit betekent dat ook hier de volgende rangorde geldt:

1. Uitdrukkingen tussen haakjes hebben de hoogste prioriteit
2. Wisselen van teken (+/-)
3. Machtsverheffen
4. Delen en vermenigvuldigen (zelfde prioriteit)
5. Optellen en aftrekken (zelfde prioriteit)

Als er geen prioriteiten zijn, zal BASIC van links naar rechts een expressie verwerken. Hiermee moet men oppassen, zoals uit enkele voorbeelden zal blijken.

a. De uitdrukking A*B/C zal op de volgende manier worden verwerkt:

1. A*B
2. (resultaat van stap 1)/C

b. A^B^C wordt verwerkt als

1. A^B
2. (resultaat van stap 1)^C

c. B^4/2 wordt verwerkt als

1. B^4
2. (resultaat van stap 1)/2

d. B^(4/2) wordt verwerkt als

1. 4/2
2. B^(resultaat van stap 1)

e. 7*(A+B)*((B-3)^(A*B)) wordt verwerkt als

1. B-3
2. A*B
3. (resultaat van stap 1) ^ (resultaat van stap 2)
4. A+B
5. 7 * (resultaat van stap 4)
6. (resultaat van stap 5) * (resultaat van stap 3)

BASIC kent ook de zogenaamde 'relational operators', dit wil zeggen de vergelijkende uitdrukkingen. Deze zullen veel worden gebruikt in voorwaardelijke sprongopdrachten als men afhankelijk van de waarde van een bepaalde variabele een aantal opdrachten door de computer wil laten uitvoeren. De operators die men in BASIC kan gebruiken zijn de volgende:

wiskundig symbool	BASIC symbool	voorbeeld	beschrijving
=	=	A = B	A is gelijk aan B
<	<	A < B	A kleiner dan B
≤	<= of =<	A <= B	A kleiner dan of gelijk aan B
>	>	A > B	A is groter dan B
≥	>= of =>	A >= B	A groter dan of gelijk aan B
≠	<> of ><	A <> B	A niet gelijk aan B

De symbolen =<, => en >< worden door de interpreter wel geaccepteerd, maar onmiddellijk omgevormd tot <=, >= en <>, in welke vorm ze op het beeldscherm of papier zullen verschijnen bij het uitlijsten van het programma.

5. PRINCIPES VAN BASIC

Iedereen heeft tegenwoordig wel eens met een (al of niet programmeerbare) rekenmachine gewerkt. In feite is een computer niets anders. De opdrachten zien er iets anders uit, maar ze hebben dezelfde betekenis. Een programma bestaat uit een aantal opdrachten (statements). Deze opdrachten worden voorafgegaan door een regelnummer. De computer zal de opdrachten uitvoeren naar oplopend regelnummer. Dit wordt alleen onderbroken als er een sprongopdracht voorkomt. Dit is een opdracht die de computer vertelt naar een bepaalde regel te gaan en het programma daar verder uit te voeren.

Voor BASIC moet men steeds op het volgende attent zijn:

1. Elke opdracht (=regel in een programma) moet worden voorafgegaan door een nummer. Deze nummers hoeven niet aansluitend te zijn. Er wordt in het algemeen aangeraden de nummers met stappen van 10 op te hogen. Dit is vooral in verband met opdrachten die eventueel later moeten worden tussengevoegd. Als men regelnummers intypt die niet in oplopende volgorde zijn, zal BASIC ze automatisch in de volgorde van regelnummer zetten. Een regel tussenvoegen wordt dus eenvoudig het gewenste regelnummer intypen gevolgd door het bijbehorende statement. Op deze wijze kan men ook regels overschrijven, dit wil zeggen door een nieuwe tekst vervangen. Het verwijderen van een regel uit een programma geschiedt door het geven van het regelnummer gevolgd door het aanslaan van de RETURN toets.
   De algemene vorm van een BASIC opdracht is nu

   <regelnummer> opdracht {variabelenlist of expressie}

   waarin alles wat tussen <> staat verplicht is, terwijl alles wat tussen {} staat niet strikt noodzakelijk is, maar afhangt van de eisen van de programmeur.

2. Elke gebruikte geheugenplaats (of serie geheugenplaatsen) heeft een eigen naam. De gebruiker is vrij in het kiezen van deze namen, met de volgende beperking:
   een naam mag bestaan uit een letter of een letter gevolgd door een cijfer.

Voorbeeld:

toegestaan	niet toegestaan
A	AA
A1	1A
Z9	JAN
B4	A12

Deze namen worden door de BASIC-interpreter 'vertaald' naar adressen van geheugenplaatsen. Een ander woord voor deze namen is 'variabelen'.

3. In de meeste gevallen zal een computer ook moeten werken met stukken tekst die in het geheugen worden opgeslagen. Dit kan zijn een naam, maar ook een stukje tekst om de uitvoer te verduidelijken. Een tekst kan verschillende bytes geheugen beslaan. De computer houdt zelf bij hoeveel plaatsen de tekst inneemt. Het symbolische beginadres van een tekst in het geheugen wordt aangegeven door een naam met een dollarteken erachter. De regels voor deze namen zijn hetzelfde als bij 2. Enkele voorbeelden:

toegestaan	niet toegestaan
A$	1$
Z2$	1X$
A5$	JAN$

In het programma, bijvoorbeeld als men een tekst (string) in het geheugen wil plaatsen, moet deze string altijd tussen quotes worden gegeven, bijvoorbeeld

"WAT IS UW NAAM?"
"HET RESULTAAT IS"

Een eenvoudig programmavoorbeeldje is nu:

```
LISTNH
10 REM EERSTE VOORBEELD
20 LET A=10
30 LET B=1
40 LET C=A+B
50 PRINT C
60 STOP
70 END

READY
RUNNH

 11


TIME:  0.01 SECS.

READY
```

(N.B.: De meeste van de programma's die in deze nota beschreven worden zijn geschreven en hebben gedraaid op de DEC-10, vandaar dat de gebruikte CPU-tijd erbij vermeld is. Dit is de pure rekentijd die een programma nodig heeft gehad.)

We zullen bovenstaand programma regel voor regel bespreken. In de eerste regel komen we meteen al een belangrijk statement tegen. Dit statement kan worden gebruikt om commentaar in programma's te geven (remark). Als de interpreter de lettercombinatie REM als eerste in een statement tegenkomt, weet hij dat deze regel niet als opdracht behandeld hoeft te worden. Het is vooral in langere programma's aan te bevelen regelmatig gebruik van dit REM-statement te maken, daar het de programma's aanzienlijk verduidelijkt. De algemene vorm van het REM statement is

<regelnummer> REM {tekst}

In de volgende drie regels wordt er een waarde in een geheugenplaats gezet, ofwel de variabelen A, B en C krijgen een waarde. Officieel moeten deze opdrachten worden voorafgegaan door het woordje LET. Bij apparatuur en software van DIGITAL is dit niet nodig. In de rest van deze cursus zal het woordje LET niet meer worden gebruikt. De algemene vorm van de assign-opdracht is:

<regelnummer> {LET} <variabele=expressie>

Opdracht 20 laat de waarde 10 in geheugenplaats A zetten.

Opdracht 30 zet de waarde 1 in geheugenplaats B.

Opdracht 40 moet als volgt gelezen worden:

Haal de waarde in geheugenplaats A op
Haal de waarde in geheugenplaats B op
Tel deze waarden bij elkaar op
Bewaar het resultaat (de som) in geheugenplaats C

Daar de programmeur natuurlijk niet zonder meer kan kijken wat geheugenplaats C nu voor waarde bevat, moet er een opdracht zijn om het resultaat van een serie opdrachten aan hem te laten zien. Dit kan gebeuren met het PRINT commando. Zo zal in opdracht van het bovenstaande programma de inhoud van geheugenplaats C op de gebruikersterminal verschijnen. In dit geval dus 11.
De algemene vorm van het PRINT commando is:

<regelnummer> PRINT {list}

Hierbij kan de list een serie namen van variabelen bevatten, maar er kan ook een text in voorkomen, ingesloten door quotes ("). Indien men geen argumenten achter het PRINT commando geeft, zal er op het beeldscherm een lege regel verschijnen.

Een paar voorbeelden van het PRINT commando zijn:

```
1  PRINT  "DIT IS EEN VOORBEELD"
11 PRINT
29 PRINT  "A=",A,", B=",B,", A+B=",A+B
```

In de laatste regel worden alleen de komma's die tussen quotes staan op de terminal uitgeprint. De andere dienen om de variabelen te scheiden. De STOP opdracht vertelt de computer dat hij kan stoppen met het programma. Deze opdracht geeft (onder RT-11 en RSX) op de terminal de mededeling

```
STOP AT LINE  60
```

zodat de programmeur weet waar het programma is gestopt. Dit is vooral gedaan omdat het mogelijk is meer dan een STOP opdracht in

een programma te hebben. Het is echter niet noodzakelijk van een STOP-opdracht gebruik te maken. De algemene vorm van het STOP commando:

<regelnummer> STOP

De END opdracht geeft aan de BASIC-interpreter door dat dit het einde van het programma is. Deze opdracht moet altijd aan het fysieke einde van het programma staan. Als de computer deze opdracht tegenkomt zal hij ook automatisch stoppen met de uitvoering van het programma. Algemene vorm:

<regelnummer> END

Een voorbeeldje met strings is nu

```
10 REM GEBRUIK VAN STRINGS
20 A$="MIJN NAAM IS JAN "
30 B$="EN JOUW NAAM IS PIET"
40 PRINT A$
50 PRINT B$
60 PRINT A$;B$
70 END

READY
RUNNH


MIJN NAAM IS JAN
EN JOUW NAAM IS PIET
MIJN NAAM IS JAN EN JOUW NAAM IS PIET


TIME:  0.01 SECS.

READY
```

De eerste tekstregel wordt nu geplaatst in de serie geheugenplaatsen met de symbolische naam A$, de tweede in die met naam B$. In regel 40 wordt nu de tekst in de serie geheugenplaatsen A$ uitgeprint. Regel 50 doet hetzelfde met de serie B$. In regel 60 ziet de computer na A$ een komma staan gevolgd door nog een variabele. Hij weet in dit geval dat hij op dezelfde regel moet blijven en de tweede tekst direkt achter de eerste uitschrijven. Op het gebruik van "," en ";" wordt nog nader ingegaan.

Het is mogelijk om meer opdrachten op een regel te plaatsen. Dit is in de meeste gevallen niet aan te bevelen, en wel om 2 redenen:

a. Als er wat aan een opdracht veranderd moet worden, de hele regel opnieuw moet worden ingetypt.

b. De later te bespreken GOTO opdracht springt naar een regelnummer. Als er bijvoorbeeld 3 opdrachten op een regel staan, kan men nooit naar de tweede of derde opdracht springen.

Indien men toch enkele opdrachten op een regel zet, moeten deze gescheiden worden door een " " (BACKSLASH). Voorbeelden van het gebruik van verschillende opdrachten op een regel:

```
10 A=10 B=20 PRINT A,B,A+B
20 A=-10 B=1 C=A+B D=A*B PRINT A,B,C,D,C-D
```

6. INPUT EN OUTPUT

Een van de meest belangrijke onderdelen van een programma is de communicatie tussen programmeur en computer. Indien dit niet mogelijk was zou men nooit resultaten van programma's kunnen krijgen. De PRINT opdracht is reeds in het voorgaande hoofdstuk genoemd. Deze diende om data vanuit de computer aan de gebruiker door te geven. Het omgekeerde, dus het doorgeven van data aan de computer is ook mogelijk. Dit gebeurt met het commando INPUT. Algemene vorm:

```
<regelnummer>  INPUT   {list}
```

waarin list een rij van minstens 1 variabelenaam is. Als de computer een INPUT statement tegenkomt, zet hij een ? op de terminal. Dat wil zeggen dat de gebruiker een waarde of tekst in moet typen. Met het aanslaan van de RETURN toets geeft de gebruiker aan dat hij klaar is met intypen.
Een voorbeeld:

```
LISTNH
10 REM VOORBEELD INPUT
20 INPUT A
30 INPUT B
40 C=A+B
50 PRINT "C=",C
60 END

READY
RUNNH


 ?6
 ?7
C=              13


TIME:  0.03 SECS.

READY
```

In dit programma worden de waarden van A en B ingelezen via de terminal. Deze waarden worden opgeteld en dit totaal wordt uitgeprint. Een iets nettere uitvoering van dit programma is:

```
LISTNH
10 REM BETER VOORBEELD INVOER
20 PRINT "GEEF WAARDE VOOR A";
30 INPUT A
40 PRINT "GEEF WAARDE VOOR B";
50 PRINT
60 C=A+B
70 PRINT "A=";A;"   B=";B;"   SOM=";C
80 STOP
90 END

READY
45 INPUT B
RUNNH


GEEF WAARDE VOOR A ?6
GEEF WAARDE VOOR B ?7

A= 6    B= 7    SOM= 13


TIME:  0.03 SECS.

READY
```

(Bij het nalezen van het programma alvorens de opdracht 'RUN' te geven zien we dat INPUT B vergeten is, zodat we dit alsnog intikken.) Hierbij betekent de puntkomma achter de printopdracht dat de terminal op dezelfde regel moet blijven staan. Zonder deze puntkomma gaat de terminal naar de volgende regel bij het INPUT statement, zodat het vraagteken weer in de eerste positie zou komen. Het is ook mogelijk meer getallen op een regel in te typen. In dit geval moeten de getallen gescheiden worden door komma's en door hetzelfde INPUT statement worden ingelezen. Een voorbeeldje:

```
LISTNH
10 REM VOORBEELD INVOER MEER GETALLEN OP EEN REGEL
20 PRINT "GEEF GETALLEN";
30 INPUT A,B,C
40 S=A+B+C
50 PRINT "SOM IS";S
60 END

READY
RUNNH


GEEF GETALLEN ?3,5,-1
SOM IS 7


TIME:  0.03 SECS.

READY
```

Ook strings kunnen door het INPUT-statement worden ingevoerd. Dit gebeurt op precies dezelfde wijze als met normale variabelen.
Een voorbeeld:

```
LISTNH
10 REM INPUT STRINGS
20 PRINT "WAT IS UW NAAM?"
30 INPUT N$
40 PRINT "WAAR WOONT U";
50 INPUT W$
60 PRINT "GOEDENDAG ";N$;" UIT ";W$
70 END

READY
RUNNH


WAT IS UW NAAM?
 ?J.G.WESSELING
WAAR WOONT U ?WAGENINGEN
GOEDENDAG J.G.WESSELING UIT WAGENINGEN


TIME:  0.04 SECS.

READY
```

Let ook hier weer op het verschil tussen de beide PRINT-statements. Bij de eerste staat geen komma achteraan, dus zal BASIC het ? van het INPUT-statement op de volgende regel plaatsen. In regel 40 staat een komma achter de tekst, zodat het vraagteken achter deze tekst wordt geplaatst. Binnen de quotes is dan ook geen vraagteken geplaatst.

In de vorige paragraaf is het PRINT-statement al besproken. Hierbij kan men echter de positie van het getal dat wordt uitgeprint nog enigszins zelf bepalen door middel van de komma (",") of puntkomma (";") achter de variabele-naam te plaatsen. BASIC beschouwt een regel op een terminal of printer als opgebouwd uit 5 zones van 14 posities. Als men twee getallen wil uitprinten en ze zijn gescheiden door een komma, zal de printkop (of cursor) minstens een positie opschuiven, zodat elk getal in een aparte zone komt. Als men bij de uitvoer een van de zones over wil slaan kan dit door het plaatsen van een extra komma. Wil men een kompaktere opbouw, dan kan gebruik worden gemaakt van de puntkomma (";"). In dit geval worden de getallen achter elkaar weggeschreven, gescheiden door een spatie en een positie voor een eventueel minteken.

Dit zal worden verduidelijkt aan de hand van het volgende voorbeeld:

```
LISTNH
10 REM VOORBEELD GEBRUIK KOMMA EN PUNTKOMMA
20 A=1
30 B=-20
40 C=300
50 D=-100
60 PRINT A,B,C,D
70 PRINT A,B,,D
80 PRINT A;B;C;D
90 END

READY
RUNNH


 1             -20            300          -100
 1             -20                         -100
 1 -20  300 -100


TIME:  0.03 SECS.

READY
```

Indien men van tevoren al weet dat een variabele een aantal waarden moet doorlopen, kan men gebruik maken van de READ - DATA combinatie. Dit is een veelgebruikte combinatie van 2 opdrachten. Algemene vorm:

<regelnummer> READ {var1, var2,.....,varn}

<regelnummer> DATA waarde 1,{waarde 2,.....,waarde n}

Het gebruik van deze combinatie zal aan de hand van een voorbeeld worden behandeld.

```
LISTNH
10 REM VOORBEELD GEBRUIK READ-DATA
20 READ A,B,C
30 D=A+B+C
40 READ X1,X2
50 Y=X1-X2
60 READ Z,Z2
70 E=D+A+Y-Z*Z2
80 PRINT A,B,C,D
90 PRINT X1,X2,Y
100 PRINT Z,Z2,E
110 DATA 1,2,3,-1,2.5,3.0
120 DATA 0.5
130 END

READY
RUNNH

 1             2             3             6
-1             2.5          -3.5
 3             0.5           2


TIME:  0.02 SECS.

READY
```

Als in een programma een READ statement voorkomt, zal de computer automatisch naar het bijbehorende DATA statement zoeken. Zo zal hij in regel 20 automatisch naar regel 110 springen om daar de waarden van de variabelen te halen. In dit geval krijgt de variabele A de waarde 1, B de waarde 2 en C de waarde 3. Intern houdt de computer een teller bij hoeveel waarden hij ingelezen heeft van een DATA statement. Als hij bij regel 40 is aangekomen, weet hij dat de variabelen X1 en X2 respectievelijk de vierde en vijfde waarden uit het DATA statement moeten krijgen, in dit geval -1 en 2,5. In regel 60 krijgen de variabelen Z en Z2 hun waarden. In het DATA statement van regel 110 vindt hij echter alleen de waarde voor Z. Voor de waarde van Z2 zoekt hij dan naar het volgende DATA statement dat staat in regel 120.

DATA statements hoeven niet op een bepaalde plaats te staan. Zij mogen overal in het programma voorkomen, zelfs voor een READ-statement. Het is aan te bevelen het DATA statement echter consequent ofwel na het READ-statement ofwel aan het einde van een programma te plaatsen.

Indien het nodig is om meer keren dezelfde serie getallen te gebruiken die in een DATA-statement staan, kan het RESTORE commando worden gebruikt. Dit commando zet de teller die bijhoudt waar in een DATA-statement moet worden gelezen terug op 1, zodat bij het volgende READ-statement het eerste getal weer wordt gelezen.
Een voorbeeld:

```
LISTNH
10 REM RESTORE BIJ READ-DATA
20 READ A,B,C,D$
30 PRINT D$,A,B,C
40 S=A+B+C
50 PRINT "DE SOM VAN DE GETALLEN IS",S
60 PRINT
70 RESTORE
80 READ F,G,H
90 PRINT F;G;H
100 P=F*G*H
110 PRINT "HET PRODUKT IS";P
120 DATA 2,4,6,"GETALLEN:"
130 END

READY
RUNNH


GETALLEN:       2              4              6
DE SOM VAN DE GETALLEN IS      12

 2  4  6
HET PRODUKT IS 48


TIME:  0.03 SECS.
```

In regel 20 worden de data van regel 120 ingelezen. Hier kan men zien dat ook strings in een DATA-regel mogen voorkomen. Na het sommeren wordt de som van de waarden uitgeprint. In regel 70 wordt de teller weer op 1 gezet, en in regel 80 worden dezelfde waarden

nogmaals gelezen. Uit de output van bovenstaand voorbeeld blijkt dat het programma inderdaad dezelfde waarden weer inleest.

Als programma is het bovenstaande natuurlijk veel te omslachtig. Het is alleen bedoeld om het RESTORE commando te demonstreren. Hoe men met een veel kleiner programma dezelfde resultaten kan bereiken wordt ter oefening aan de lezer overgelaten. Het inlezen en wegschrijven van data van en naar schijf zal in een van de volgende hoofdstukken worden besproken.

7. SPRONG OPDRACHTEN

De programma's in de voorgaande hoofdstukken worden maar een keer uitgevoerd, en wel elke keer van boven naar beneden, precies volgens regelnummer. Om hiervan af te kunnen wijken, maakt men gebruik van sprongopdrachten. Het resultaat van een sprongopdracht is dat de instructieteller niet met 1 wordt opgehoogd, zodat de volgende instruktie wordt opgehaald uit het geheugen: bij een sprongopdracht wordt de instruktieteller gevuld met de positie van een instruktie ergens in het programma. Men kan zowel vooruit als achteruit springen. De algemene vorm van de eenvoudigste sprongopdracht is:

```
<regelnummer>  GOTO regelnummer
```

Dit is de zogenaamde onvoorwaardelijke sprongopdracht ("unconditional branch instruction"). Iedere keer wanneer deze opdracht wordt bereikt, zal de computer de sprongopdracht uitvoeren. Een voorbeeldje:

```
LISTNH
10 REM DE SPRONGOPDRACHT
20 PRINT "GEEF A";
30 INPUT A
40 PRINT "GEEF B";
50 INPUT B
60 C=A+B
70 PRINT "A+B=",C
80 GOTO 20
90 STOP
100 END

READY
RUNNH


GEEF A ?1
GEEF B ?2
A+B=            3
GEEF A ?4
GEEF B ?-3
A+B=            1
GEEF A ?↑C

READY
```

Dit programma vraagt om twee getallen in te typen. In regel 60 worden deze getallen bij elkaar opgeteld en in regel 70 wordt het resultaat gegeven. Na het uitprinten van deze getallen komt het programma bij regel 80. Dit is de sprongopdracht die de computer vertelt terug te springen naar opdracht nummer 20. Dat wil zeggen dat hij weer om twee getallen vraagt, en weer optelt, en zo voort. Het is nu duidelijk geworden dat dit programma nooit op een normale manier zal worden beëindigd, en nooit aan END toekomt. Er zal altijd weer worden gesprongen naar regel 10. Dit soort programma's moeten altijd op de 'harde' manier worden afgebroken. Sprongopdrachten hebben wel degelijk nut, maar dan moeten zij worden gecombineerd met de voorwaardelijke sprongopdrachten ("conditional branch instructions"). De algemene vorm van zo'n voorwaardelijke sprongopdracht is:

<regelnummer> IF expressie GOTO regelnummer

of

<regelnummer> IF expressie THEN regelnummer

Deze mogelijkheden hebben precies hetzelfde resultaat. In bovenstaande statements is expressie een uitdrukking die ofwel waar is of niet waar is. Als de uitdrukking waar is, zal de sprong-opdracht uitgevoerd worden. Is hij niet waar dan zal de sprong-opdracht worden genegeerd en zal het programma met de volgende opdracht doorgaan.

Er wordt in een expressie gebruik gemaakt van een van de relational operators. Een paar voorbeelden van het IF statement ter verduidelijking:

```
10  IF A>1 GO TO 20
40  IF C<A GOTO 50
70  IF B1 >= 0 THEN 210
120 IF Z1$<W9$ THEN 900
```

In het voorbeeld (regel 10) wordt naar opdrachtnummer 20 gesprongen als de variabele A een waarde heeft die groter is dan 1. Als A kleiner of gelijk aan 1 is, zal met de volgende opdracht worden doorgegaan, met andere woorden de sprongopdracht zal worden genegeerd. In regel 40 wordt naar 50 gesprongen als de waarde van C kleiner is dan die van A. De sprongopdracht in regel 70 wordt uitgevoerd als B1 groter of gelijk is aan 0. De opdracht in regel 120 verdient wat extra aandacht, daar deze met strings werkt. Als met strings wordt gewerkt en een relational operator, zal de string karakter voor karakter worden bekeken. Zo zal eerst het eerste karakter worden bekeken. Zijn deze gelijk, dan zal het tweede worden bekeken. Daar een computer natuurlijk alleen getallen kan bewerken, zal hij werken met de ASCII waarden van de letter. Als bijvoorbeeld de 3de letter van Z1$ een W is, en de derde letter van W9$ een K, beschouwt hij W9$ als kleiner dan Z1$, ongeacht de volgende letters. Van deze eigenschap kunnen we uitgebreid gebruik maken bij het alfabetiseren van bijvoorbeeld namen.

Onder RSX kan men nog van een derde mogelijkheid gebruik maken:

<regelnummer> IF expressie THEN statement

Dit statement kan nu een assign-statement zijn, of een subroutine-call, die later besproken zal worden, of een willekeurig ander statement. Enkele voorbeelden:

```
   20 IF A>10 THEN A=10
   30 IF Z2 = 0 THEN X=A*B
 1940 IF N$='JAN' THEN V$='PIET'
```

Daar dit alleen bij RSX kan, en niet bij de DEC-10 of onder RT-11 zal hier verder geen gebruik van worden gemaakt in de voorbeelden.

Een voorbeeld van het gebruik van een voorwaardelijke sprong-opdracht:

```
LISTNH
10 REM VOORWAARDELIJKE EN ONVOORWAARDELIJKE
20 REM SPRONGOPDRACHTEN.
30 PRINT "GEEF GETAL";
40 INPUT Z
50 IF Z>=999 THEN 140
60 IF Z>0 THEN 100
70 IF Z=0 GOTO 120
80 PRINT "GETAL IS KLEINER DAN 0"
90 GO TO 30
100 PRINT "GETAL IS GROTER DAN 0"
110 GO TO 30
120 PRINT "GETAL IS GELIJK AAN 0"
130 GO TO 30
140 PRINT "EINDE PROGRAMMA"
150 END

READY
RUNNH


GEEF GETAL ?-5
GETAL IS KLEINER DAN 0
GEEF GETAL ?0
GETAL IS GELIJK AAN 0
GEEF GETAL ?5
GETAL IS GROTER DAN 0
GEEF GETAL ?999
EINDE PROGRAMMA



TIME:  0.06 SECS.

READY
```

Dit programma maakt duidelijk gebruik van zowel de voorwaardelijke als de onvoorwaardelijke sprongopdracht. Na het inlezen van een getal wordt in regel 50 gekeken of dit getal groter is of gelijk is aan 999. Is dit het geval, dan wordt gesprongen naar opdrachtregel 140. Deze print de eindboodschap. Als A echter niet groter is dan 999, zal gekeken moeten worden of het getal positief, nul of negatief is. In regel 60 wordt gesprongen als Z positief is. In dit geval zal naar regel 100 worden gesprongen, waar de boodschap wordt geprint dat het getal positief is. Na het printen springt het programma weer naar opdracht 30, en zal een nieuw getal worden gevraagd. Indien echter het getal niet groter dan nul was, zal in regel 70 worden gekeken of het nul is. Is dit ook niet het geval, dan zal het automatisch negatief moeten zijn. Vandaar dat er niet wordt getest of het getal negatief is. Zoals in het programma goed te zien is worden voorwaardelijke en onvoorwaardelijke sprongopdrachten vaak samen gebruikt. Dit programma heeft een duidelijke methode ingebouwd om met de executie van het programma te kunnen stoppen.

Een derde, voor sommige doeleinden zeer bruikbare sprongopdracht is de ON-GOTO combinatie. Algemene vorm:

<regelnummer> ON expressie GOTO regelnummers

Hierin mag expressie staan voor zowel een vergelijking als voor een variabele. Achter het GOTO staan nu een aantal regelnummers. In principe is het aantal regelnummers achter GOTO onbeperkt. In de praktijk is het aantal echter beperkt door het aantal dat op een commandoregel kan worden geplaatst.

Afhankelijk van de waarde van de expressie wordt nu naar een van de regelnummers gesprongen. Aan de hand van een voorbeeldje zal dit commando worden besproken.

```
20 ON I GOTO 50,30,100,200
```

Afhankelijk van de waarde van I wordt nu een sprongopdracht uitgevoerd, en wel op de volgende wijze:

als 1 <= I < 2 wordt gesprongen naar 50,
2 <= I < 3 wordt gesprongen naar 30,
3 <= I < 4 wordt gesprongen naar 100,
4 <= I < 5 wordt gesprongen naar 200.

Als I<1 of I>=5 verschijnt een foutmelding op de terminal.

Een voorbeeldje met deze instruktie is het volgende: stel het Ministerie van Financiële zaken heeft een nieuwe belasting ontwikkeld wegens een of ander tekort. Deze belasting werkt als volgt: bedragen beneden de honderd gulden en boven de vijfhonderd gulden worden niet belast. De bedragen hiertussen worden als volgt belast:

100 <= bedrag < 200 : 20%
200 <= bedrag < 300 : 30%
300 <= bedrag < 400 : 40%
400 <= bedrag < 500 : 50%

De vraag is nu om een programma te schrijven dat na intypen van een willekeurig bedrag de te betalen belasting geeft. Dit kan als volgt worden gedaan:

```
10 REM BEREKENEN VAN BELASTING
20 PRINT "VAN WELKE BEDRAG MOET DE BELASTING WORDEN BEREKEND";
30 INPUT B
40 IF B>=500 THEN 170
50 IF B<100 THEN 170
60 ON B/100 GOTO 70,90,110,130
70 X=20
80 GOTO 140
90 X=30
100 GOTO 140
110 X=40
120 GOTO 140
130 X=50
140 T=X*B/100
150 PRINT "DE TE BETALEN BELASTING IS";X;"% EN HET BEDRAG F.";T
160 GOTO 180
170 PRINT "U BOFT. U HOEFT GEEN BELASTING TE BETALEN."
180 PRINT
190 PRINT "WILT U NOG MEER BEDRAGEN BEREKENEN? TYPE JA OF NEE";
200 INPUT A$
210 IF A$="JA" GOTO 20
220 PRINT
230 PRINT "DAN WENS IK U VERDER NOG EEN PRETTIGE DAG."
240 END

READY
RUNNH
```

```
VAN WELKE BEDRAG MOET DE BELASTING WORDEN BEREKEND ?150
DE TE BETALEN BELASTING IS 20 % EN HET BEDRAG F. 30

WILT U NOG MEER BEDRAGEN BEREKENEN? TYPE JA OF NEE ?JA
VAN WELKE BEDRAG MOET DE BELASTING WORDEN BEREKEND ?60
U BOFT. U HOEFT GEEN BELASTING TE BETALEN.

WILT U NOG MEER BEDRAGEN BEREKENEN? TYPE JA OF NEE ?NEE

DAN WENS IK U VERDER NOG EEN PRETTIGE DAG.


TIME:  0.07 SECS.

READY
```

Laten we nu het eerste geval eens doornemen. In de regels 40 en 50 wordt gecontroleerd of het bedrag binnen de belastbare grenzen valt. Als dit niet het geval is wordt de zin in regel 170 uitgeprint. Als wel belasting moet worden betaald zal in regel 60 worden uitgerekend in welke klasse het bedrag zit. De ingetypte 150 gulden worden hier door 100 gedeeld, waardoor het getal 1,5 ontstaat. Volgens de bovenstaande beschrijving wordt nu naar het eerste regelnummer in de rij gesprongen, waardoor X op 20% wordt gezet. Vervolgens wordt verdergegaan op regel 140 waar het bedrag wordt berekend wat betaald moet worden. Na het uitprinten van dit bedrag en het percentage wordt gesprongen naar regel 190 waar een vraag wordt uitgeschreven. Hierop verwacht de computer een antwoord in karakters (A$). In regel 190 wordt deze string vergeleken met de string JA. Als de twee strings precies gelijk zijn zal het programma terugspringen naar regel 20. Dit zal uiteraard alleen gebeuren als men precies het woord JA intypt. Indien men wat anders intypt, zal het programma stoppen.

8. CONVERSATIONELE PROGRAMMA's

Wanneer men de wijze van programmeren gebruikt zoals in het vorige hoofdstuk beschreven is, kan een zeer duidelijk interaktief programma worden geschreven, door veel vragen te laten stellen die met ja of nee beantwoord moeten worden. Afhankelijk van het gegeven antwoord wordt naar het relevante programma-onderdeel gesprongen. Programma's die op deze wijze zijn opgebouwd heten conversationele programma's (er vindt een soort 'conversatie' plaats tussen gebruiker en computer). Het is mogelijk om vragen in plaats van met ja en nee met 0 en 1 te laten beantwoorden, maar de meeste mensen zullen aan het gebruik van ja en nee de voorkeur geven. Het mogen natuurlijk ook andere antwoorden zijn, afhankelijk van de vraagstelling.

Bij het programmeren dient men er om te denken de te geven antwoorden in de vragen op te nemen. Het gebruik van een programmabeschrijving is dan niet meer nodig en het programma wordt gebruikersvriendelijker.

Enkele voorbeelden van vragen zijn nog:

```
"WILT U JA/NEE DOORGAAN?"

"ALS U WILT STOPPEN, TYPE DAN STOP"

"TIK IN:  I=INTIKKEN NIEUWE GEGEVENS"
"         U=UITSCHRIJVEN TUSSENRESULTATEN"
"         T=TESTEN OP NEGATIEVE WAARDEN"
"         S=STOPPEN"
"IK KIES NU:"

"WELKE HANDELING MOET IK UITVOEREN? TYPE HELP VOOR HULP"
```

Een klein voorbeeldje is nu het simuleren van een rekenmachientje dat alleen kan optellen, aftrekken, vermenigvuldigen en delen:

```
10 REM REKENMACHIENTJE
20 PRINT "DIT PROGRAMMA SIMULEERT EEN EENVOUDIG REKENMACHIENTJE"
30 PRINT "DAT ENKELE REKENKUNDIGE BEWERKINGEN OP 2 GETALLEN KAN UITVOEREN EN"
40 PRINT "DAT WERKT MET DE REVERSE POLISH METHOD. DE ENIGE BEWERKINGEN"
50 PRINT "DIE UITGEVOERD KUNNEN WORDEN ZIJN OPTELLEN, AFTREKKEN,"
60 PRINT "VERMENIGVULDIGEN EN DELEN."
70 PRINT
80 PRINT "DE COMMANDO'S ZIJN:"
90 PRINT "     + : TEL DE GETALLEN BIJ ELKAAR OP"
100 PRINT "     - : TREK HET TWEEDE GETAL VAN HET EERSTE AF"
110 PRINT "     * : VERMENIGVULDIG DE GETALLEN MET ELKAAR"
120 PRINT "     / : DEEL HET EERSTE GETAL DOOR HET TWEEDE"
130 PRINT
140 PRINT "OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD";
150 INPUT X,Y
160 PRINT "WELKE BEWERKING (TYPE +,-,* OF /)";
170 INPUT B$
180 IF B$<>"+" GOTO 220
190 REM OPTELLEN
200 PRINT X;"+";Y;"=";X+Y
210 GO TO 410
220 IF B$<>"-" GO TO 260
230 REM AFTREKKEN
240 PRINT X;"-";Y;"=";X-Y
250 GOTO 410
260 IF B$<>"*" GOTO 300
270 REM VERMENIGVULDIGEN
280 PRINT X;"*";Y;"=";X*Y
290 GOTO 410
300 IF B$<>"/" GOTO 380
310 REM DELEN
320 REM DOOR 0?
330 IF Y=0 GOTO 360
340 PRINT X;":";Y;"=";X/Y
350 GOTO 410
360 PRINT "DOOR 0 DELEN KAN GEEN ENKELE REKENMACHINE"
370 GO TO 410
380 REM BEWERKING ONBEKEND
390 PRINT "DIE BEWERKING KEN IK NIET. GEEF EEN ANDERE BEWERKING";
400 GO TO 170
410 PRINT
420 PRINT "WILT U NOG MEER BEREKENEN? TYPE JA OF NEE";
430 INPUT A$
440 IF A$="JA" GOTO 130
450 PRINT
460 PRINT "DAN IS DIT HET EINDE VAN DE SIMULATIE"
470 END

READY
RUNNH
```

```
DIT PROGRAMMA SIMULEERT EEN EENVOUDIG REKENMACHIENTJE
DAT ENKELE REKENKUNDIGE BEWERKINGEN OP 2 GETALLEN KAN UITVOEREN EN
DAT WERKT MET DE REVERSE POLISH METHOD. DE ENIGE BEWERKINGEN
DIE UITGEVOERD KUNNEN WORDEN ZIJN OPTELLEN, AFTREKKEN,
VERMENIGVULDIGEN EN DELEN.

DE COMMANDO'S ZIJN:
     + : TEL DE GETALLEN BIJ ELKAAR OP
     - : TREK HET TWEEDE GETAL VAN HET EERSTE AF
     * : VERMENIGVULDIG DE GETALLEN MET ELKAAR
     / : DEEL HET EERSTE GETAL DOOR HET TWEEDE

OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD ?5,4
WELKE BEWERKING (TYPE +,-,* OF /) ?+
 5 + 4 = 9

WILT U NOG MEER BEREKENEN? TYPE JA OF NEE ?JA

OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD ?6,-13
WELKE BEWERKING (TYPE +,-,* OF /) ?↑
DIE BEWERKING KEN IK NIET. GEEF EEN ANDERE BEWERKING ?*
 6 *-13 =-78

WILT U NOG MEER BEREKENEN? TYPE JA OF NEE ?JA

OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD ?8,4.5
WELKE BEWERKING (TYPE +,-,* OF /) ?/
 8 : 4.5 = 1.77778

WILT U NOG MEER BEREKENEN? TYPE JA OF NEE ?NEE

DAN IS DIT HET EINDE VAN DE SIMULATIE


TIME:  0.18 SECS.

READY
```

Het programma kan worden verkort door getallen te gebruiken in plaats van string-commando's:

```
LISTNH
10 REM REKENMACHIENTJE
20 PRINT "DIT PROGRAMMA SIMULEERT EEN EENVOUDIG REKENMACHIENTJE"
30 PRINT "DAT ENKELE REKENKUNDIGE BEWERKINGEN OP 2 GETALLEN KAN UITVOEREN EN"
40 PRINT "DAT WERKT MET DE REVERSE POLISH METHOD. DE ENIGE BEWERKINGEN"
50 PRINT "DIE UITGEVOERD KUNNEN WORDEN ZIJN OPTELLEN, AFTREKKEN,"
60 PRINT "VERMENIGVULDIGEN EN DELEN."
70 PRINT
80 PRINT "DE COMMANDO'S ZIJN:"
90 PRINT "     1 : TEL DE GETALLEN BIJ ELKAAR OP"
100 PRINT "     2 : TREK HET TWEEDE GETAL VAN HET EERSTE AF"
110 PRINT "     3 : VERMENIGVULDIG DE GETALLEN MET ELKAAR"
120 PRINT "     4 : DEEL HET EERSTE GETAL DOOR HET TWEEDE"
130 PRINT
140 PRINT "OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD";
150 INPUT X,Y
160 PRINT "WELKE BEWERKING (1=+,2=-,3=*,4=/)";
170 INPUT B
180 ON B GOTO 200,230,260,300
190 REM OPTELLEN
200 PRINT X;"+";Y;"=";X+Y
210 GO TO 350
220 REM AFTREKKEN
230 PRINT X;"-";Y;"=";X-Y
240 GOTO 350
250 REM VERMENIGVULDIGEN
260 PRINT X;"*";Y;"=";X*Y
270 GOTO 350
280 REM DELEN
290 REM DOOR 0?
300 IF Y=0 GOTO 330
310 PRINT X;":";Y;"=";X/Y
320 GOTO 350
330 PRINT "DOOR 0 DELEN KAN GEEN ENKELE REKENMACHINE"
340 GO TO 350
350 PRINT
360 PRINT "WILT U NOG MEER BEREKENEN? TYPE JA OF NEE";
370 INPUT A$
380 IF A$="JA" GOTO 130
390 PRINT
400 PRINT "DAN IS DIT HET EINDE VAN DE SIMULATIE"
410 END

READY
RUNNH
```

```
RUNNH

DIT PROGRAMMA SIMULEERT EEN EENVOUDIG REKENMACHIENTJE
DAT ENKELE REKENKUNDIGE BEWERKINGEN OP 2 GETALLEN KAN UITVOEREN EN
DAT WERKT MET DE REVERSE POLISH METHOD. DE ENIGE BEWERKINGEN
DIE UITGEVOERD KUNNEN WORDEN ZIJN OPTELLEN, AFTREKKEN,
VERMENIGVULDIGEN EN DELEN.

DE COMMANDO'S ZIJN:
     1 : TEL DE GETALLEN BIJ ELKAAR OP
     2 : TREK HET TWEEDE GETAL VAN HET EERSTE AF
     3 : VERMENIGVULDIG DE GETALLEN MET ELKAAR
     4 : DEEL HET EERSTE GETAL DOOR HET TWEEDE

OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD ?-3,5
WELKE BEWERKING (1=+,2=-,3=*,4=/) ?2
-3 - 5 =-8

WILT U NOG MEER BEREKENEN? TYPE JA OF NEE ?JA

OP WELKE TWEE GETALLEN MOET DE BEWERKING WORDEN UITGEVOERD ?4,0
WELKE BEWERKING (1=+,2=-,3=*,4=/) ?4
DOOR 0 DELEN KAN GEEN ENKELE REKENMACHINE

WILT U NOG MEER BEREKENEN? TYPE JA OF NEE ?NEE

DAN IS DIT HET EINDE VAN DE SIMULATIE


TIME:  0.14 SECS.

READY
```

Het eerste voorbeeld is iets groter, maar een stuk duidelijker. Bovenstaande programma's kunnen op eenvoudige wijze worden uitgebreid met andere bewerkingen zoals de mogelijkheid om getallen in 'geheugenplaatsen' van het rekenmachientje vast te houden.

9. FOR-NEXT OPDRACHTEN

Indien een serie opdrachten herhaaldelijk moet worden uitgevoerd kan men gebruik maken van een sprongopdracht, zoals in hoofdstuk 7 besproken. Als het aantal malen dat deze opdrachten moeten worden uitgevoerd bekend is, zal men liever gebruik maken van een zogenaamde FOR-NEXT lus, welke te vergelijken is met de DO-loop in FORTRAN en de FOR-opdracht in PASCAL. Deze lus bestaat uit twee delen.

Het eerste deel is het FOR-commando. De algemene vorm is

<regelnummer> FOR variabele=expressie1 TO expressie2 {STEP expressie3}

De variabele wordt gebruikt als een soort teller. Bij het binnenkomen in de lus krijgt de variabele de waarde van expressie1. De volgende opdrachten worden nu uitgevoerd totdat het programma een NEXT opdracht tegenkomt. De algemene vorm van deze opdracht is

<regelnummer> NEXT variabele

Bij deze NEXT-opdracht weet de computer dat hij nu de waarde van de variabele met de waarde van expressie2 moet vergelijken. Als de waarde van de variabele groter is dan of gelijk is aan de waarde van expressie2, zal worden verdergegaan met de opdracht na de NEXT-opdracht. Als de variabele de waarde van expressie2 nog niet heeft bereikt zal de waarde van expressie3 bij de waarde van de variabele worden opgeteld en de lus nogmaals worden uitgevoerd. Als STEP wordt weggelaten neemt de computer voor de stapgrootte +1.

Enkele voorbeelden van FOR opdrachten zijn:

```
20 FOR K=1 TO 20
38 FOR L1=.5 TO 2.5 STEP .1
123 FOR A9=-.1 TO -.5 STEP -.01
1340 FOR K=A/B TO L1/L3 STEP (A2-B3)/(C4*D5)
```

en van de bijbehorende NEXT opdrachten:

```
31 NEXT K
80 NEXT L1
190 NEXT A9
2040 NEXT K
```

Zoals in bovenstaande voorbeeldjes te zien is mag in het FOR-statement zowel met gebroken als met gehele getallen worden gewerkt. Een voorbeeld van het gebruik van een FOR-NEXT loop samen met hetzelfde programma met een IF-statement wordt nu gegeven:
Stel de opdracht luidt om een programma te schrijven dat de kwadraten geeft van de getallen tussen 1 en 10. Dit kan op de volgende wijze gebeuren:

```
LISTNH
10 REM LOOP
20 I=0
30 I=I+1
40 J=I*I
50 PRINT I,J
60 IF I<10 GOTO 30
70 END

READY
RUNNH

 1            1
 2            4
 3            9
 4            16
 5            25
 6            36
 7            49
 8            64
 9            81
 10           100


TIME:  0.03 SECS.

READY
```

```
LISTNH
10 REM FOR-NEXT LOOP
20 FOR I=1 TO 10
30 J=I*I
40 PRINT I,J
50 NEXT I
60 END

READY
RUNNH

 1            1
 2            4
 3            9
 4            16
 5            25
 6            36
 7            49
 8            64
 9            81
 10           100


TIME:  0.03 SECS.

READY
```

Het is mogelijk om binnen een lus de telvariabele te veranderen. Dit wordt in het algemeen niet aangeraden, maar het volgende voorbeeld werkt wel:

```
LISTNH
10 REM LOOP MET VERANDERDE VARIABELE
20 FOR I=1 TO 30
30 J=I*I
40 PRINT I,J
50 I=J
60 NEXT I
70 END

READY
RUNNH

 1            1
 2            4
 5            25
 26           676


TIME:  0.02 SECS.

READY
```

Bij het binnenkomen in de lus wordt I nu 1 gesteld. Het kwadraat van 1 is 1 dus in regel 50 zal de waarde van I niet veranderen. Bij de volgende maal dat de lus wordt uitgevoerd is I met 1 opgehoogd, dus heeft de waarde 2. In regel 50 wordt I nu 4 gesteld, zodat bij de hieropvolgende doorgang het kwadraat van 5 wordt berekend, en zo voort tot de lus 'satisfied' is (in dit geval dus als 676 groter blijkt te zijn dan 30) en het programma vervolgd wordt.

Het is mogelijk om binnen een lus weer een lus te maken. De lussen zijn dan 'genest'. Hierbij gelden echter wel enkele beperkingen. De lussen mogen elkaar niet kruisen, en ze moeten gebruik maken van verschillende variabelen. Sprongopdrachten zijn wel toegestaan. Enkele voorbeelden van deze 'geneste' lussen zijn:

toegestaan

```
10 FOR A=1 TO 20
20 FOR J=3 TO 21 STEP 3
30 NEXT J
40 NEXT A
```

niet-toegestaan

```
10 FOR I=1 TO 20
20 FOR X=-3 TO 3
30 NEXT I
40 NEXT A
```

```
10 FOR A=3 TO 20 STEP 4
20 FOR B=1 TO 3
30 FOR C=1 TO -3 STEP -1
40 NEXT C
50 FOR D=5 TO 100 STEP 10
60 NEXT D
70 NEXT B
80 NEXT A
```

```
10 FOR A=1 TO 20
20 FOR B=3 TO 40
30 FOR C=2 TO B+A
40 NEXT C
50 FOR D=4 TO 10
60 NEXT D
70 NEXT A
80 NEXT B
```

Terwille van de duidelijkheid is hier tussen de FOR en de NEXT geen serie opdrachten geplaatst. Dit zijn dus geen eigenlijke programma's maar voorbeelden van lussen. Een voorbeeld van een programma met een geneste loop is gegeven in het volgende. Het gestelde probleem is hier om alle combinaties van getallen tussen 1 en 5 te vinden waarvan de som kleiner dan 5 is.

```
LISTNH
10 REM GENESTE LOOPS
20 PRINT " A"," B","SOM"
30 FOR A=1 TO 5
40 FOR B=1 TO 5
50 S=A+B
60 IF S>=5 GOTO 90
70 PRINT A,B,S
80 NEXT B
90 NEXT A
100 END

READY
RUNNH

 A            B            SOM
 1            1             2
 1            2             3
 1            3             4
 2            1             3
 2            2             4
 3            1             4


TIME:  0.04 SECS.

READY
```

In dit probleem wordt ervan uitgegaan dat, als twee getallen een som hebben die gelijk aan of groter dan 5 is geworden, de som van alle volgende waarden van de variabelen A en B ook een grotere som hebben, namelijk: variabele A wordt in de binnenste lus constant gehouden terwijl variabele B in grootte toeneemt.

Als de som groter wordt dan 5, wordt door de sprong uit de binnenste lus A een opgehoogd, terwijl de binnenste lus weer met 1 begint. Als alle waarden voor A geprobeerd zijn stopt het programma.

10. ARRAYS EN MATRICES

Als veel variabelen gebruikt moeten worden blijkt het geven van namen vaak een probleem te worden. Daarnaast treft men vaak aan dat grote aantallen variabelen een zekere relatie in volgorde tot elkaar hebben. Daarom heeft men de zogenaamde arrays ingevoerd. Dit zijn variabelen, die allen dezelfde naam hebben, maar voorzien zijn van een soort volgnummer. Deze arrays kunnen veel werk besparen omdat het mogelijk is het volgnummer (subscript) in een FOR-NEXT lus als lopende variabele te gebruiken. Ook strings kunnen in een array worden geplaatst. Men komt zo tot 1-dimensionale arrays of vectoren. Heeft men behoefte aan meer dataruimte, dan kan gebruik worden gemaakt van matrices. Dit zijn 2-dimensionale arrays. Deze variabelen hebben twee tellers (subscripts). In het algemeen geeft de eerste subscript het rijnummer van een element van de matrix, de tweede subscript het kolomnummer. Daar deze arrays (of matrices) een aaneengesloten stuk geheugen nodig hebben, is het nodig deze ruimte van tevoren te reserveren. Dit houdt wel in dat men vantevoren de grootte van de arrays moet kennen. Het reserveren van geheugenruimte gebeurt door middel van het DIM (dimension) statement. De algemene vorm is

```
<regelnummer> DIM {variabele(n),variable$(m),variable(o,p),
                                              variable$(q,r)}
```

Hierin zijn n,m,o,p,q en r integers die de maximale waarden aangeven die de subscripts kunnen bereiken, ofwel de dimensies van de arrays en matrices. Alleen de variabelen die gebruikt worden in het programma moeten worden gedefinieerd. Meestal wordt een DIM-statement aan het begin van een programma geplaatst. Enkele voorbeelden van het DIM-statement zijn:

```
10 DIM A(9),B(250)
20 DIM C(10,10),X$(20)
30 DIM V1$(20,100),X9(3),A2$(10)
```

Een naam die in een DIM-statement is gedefinieerd mag niet meer als normale variabele worden gebruikt. In BASIC is de ondergrens van

een subscript altijd 0. Zo loopt de array A in regel 10 van A(0) tot A(9) en heeft dus 10 elementen. Hier moet men wel degelijk rekening mee houden. Als in een programma een matrix nodig is van 10 bij 20 elementen, zal men hem in het algemeen dimensioneren als A(9,19).

Een voorbeeld van het gebruik van arrays is het volgende programma.

```
LISTNH
10 REM SORTEERPROGRAMMA
20 DIM A(5)
30 FOR I=0 TO 5
40 READ A(I)
50 NEXT I
60 DATA 4,1,2,10,6,3
70 FOR I=0 TO 4
80 FOR J=I+1 TO 5
90 IF A(I)<A(J) GOTO 140
100 REM OMWISSELEN VAN A(I) EN A(J)
110 W=A(I)
120 A(I)=A(J)
130 A(J)=W
140 NEXT J
150 NEXT I
160 REM DE UITVOER
170 FOR I=0 TO 5
180 PRINT A(I);
190 NEXT I
200 PRINT
210 END

READY
RUNNH

 1  2  3  4  6  10

TIME:  0.03 SECS.

READY
```

In het programma is uitgegaan van 6 getallen met de naam A, dus een array A(5). Deze getallen worden in de regels 30-60 ingelezen van een DATA-regel. Het eigenlijke sorteren gebeurt in de regels 70-150. De sorteermethode zal hier niet worden besproken. Het is aan de lezer om uit te vinden hoe deze methode werkt. Dit zal, na de voorgaande hoofdstukken, weinig of geen problemen leveren.

In de regels 170-200 worden de getallen op de terminal afgedrukt. De PRINT in regel 200 is alleen bedoeld om de cursor op de volgende regel te krijgen zodat er een blanke regel tussen de getallen en het READY bericht staat.

Als dit programma zonder arrays wordt geschreven, wat natuurlijk ook mogelijk is, zal men uitkomen op een grote hoeveelheid IF opdrachten, daar elk getal met alle andere vergeleken wordt. In bovenstaand programma kan het aantal getallen eenvoudig worden veranderd door de regels 20,30,70,80 en 170 aan te passen. In de meeste gevallen zullen de getallen echter niet via een DATA-statement worden ingelezen, maar staan ze meestal in een datafile op schijf. Hoe deze wordt ingelezen zal in een van de volgende hoofdstukken worden behandeld. Het principe van het programma verandert echter niet.

Dit is een van de vele voorbeelden die mogelijk zijn met arrays. Een andere toepassing is bijvoorbeeld het gebruik van arrays voor het sorteren op alfabet van namen met bijbehorende adressen en telefoonnummers.

N.B. Het sorteren op de hier beschreven manier verloopt weinig efficient. Er wordt op gewezen dat voor het op volgorde plaatsen van lange reeksen gegevens snellere methoden beschikbaar zijn. (Zie bijvoorbeeld Van Amstel, red., 1976).

Ook voor 2-dimensionale arrays (matrices) zijn veel toepassingen denkbaar. Een voorbeeld van het gebruik van deze arrays is het volgende programma:

```
LISTNH
10 REM MATRICES
20 DIM S(4,4)
30 REM DEFINIEER S
40 FOR I=0 TO 4
50 FOR J=0 TO 4
60 S(I,J)=I+J
70 NEXT J
80 NEXT I
90 REM UITVOER
100 PRINT "I \ J",
110 FOR J=0 TO 4
120 PRINT J;
```

```
130 NEXT J
140 PRINT
150 PRINT
160 FOR I=0 TO 4
170 PRINT I,
180 FOR J=0 TO 4
190 PRINT S(I,J);
200 NEXT J
210 PRINT
220 NEXT I
230 END

READY
RUNNH


I \ J          0  1  2  3  4

 0             0  1  2  3  4
 1             1  2  3  4  5
 2             2  3  4  5  6
 3             3  4  5  6  7
 4             4  5  6  7  8


TIME:  0.04 SECS.

READY
```

In dit programma worden eerst de elementen van de matrix gedefinieerd als de som van hun rij- en kolomnummer. Vervolgens worden de kolomnummers geprint, gevolgd door de rijnummers en de waarden van de matrixelementen. Deze matrix is in regel 20 gedefinieerd als S(4,4) en heeft dus 5*5=25 elementen.

11. SUBROUTINES EN FUNKTIES

Een subroutine is een blok opdrachten in een programma waar men naar toe kan springen. Het verschil tussen een normale sprongopdracht en een subroutine aanroep (call) is dat na het afwerken van de instructies in de subroutine, het programma verdergaat met de opdracht die staat op de plaats na de subroutine aanroep. De algemene vorm van een subroutine aanroep is

<regelnummer> GOSUB regelnummer

Als de computer een GOSUB opdracht tegenkomt plaatst hij het nummer van de hieropvolgende instruktie in een speciale geheugenpositie. Na afwerken van de instrukties in de subroutine wordt dan teruggesprongen naar de opdracht met het regelnummer dat in het geheugen is geplaatst. Door deze interne organisatie kan men van verschillende punten in hetzelfde programma naar dezelfde subroutine springen. De volgorde in de hoofdbewerking wordt hierdoor niet verstoord.

Een subroutine moet altijd eindigen met een RETURN opdracht, zodat de computer weet dat hij aan het einde van de subroutine is en terug moet springen naar 'waar hij vandaan kwam'. Algemene vorm:

<regelnummer> RETURN

Het grote voordeel van subroutines is dat als een bepaald stuk programma op meer plaatsen gebruikt moet worden, het niet nodig is dit stuk te herhalen. Door het eenvoudig aanroepen van de subroutine worden de instrukties uitgevoerd. Een voorbeeld van een subroutine en het gebruik ervan is het volgende:

```
LISTNH
10 REM OPLOSSEN VAN DE VERGELIJKING A*X↑2+B*X+C
20 PRINT "GEEF A,B EN C";
30 INPUT A,B,C
40 GOSUB 500
50 IF T=1 GOTO 70
60 PRINT "DE OPLOSSINGEN ZIJN: ";X1;"EN ";X2
70 STOP
500 REM BEGIN SUBROUTINE
510 REM BEREKEN DISKRIMINANT
520 D=B*B-4*A*C
525 T=0
530 IF D<0 GOTO 600
540 IF D=0 GOTO 700
550 X1=(-B-SQR(D))/(2*A)
560 X2=(-B+SQR(D))/(2*A)
570 RETURN
600 T=1
610 PRINT "GEEN REELE OPLOSSINGEN"
620 RETURN
700 X1=-B/(2*A)
710 X2=X1
720 RETURN
999 END
```

```
READY
RUNNH

GEEF A,B EN C ?1,-2,1
DE OPLOSSINGEN ZIJN:  1 EN  1


TIME:  0.03 SECS.

READY
```

Dit programma geeft de oplossingen van de vergelijking A*X^2+B*X+C=0 na het intypen van de waarden voor A,B en C. Na het inlezen van de waarden wordt naar de subroutine gesprongen. Afhankelijk van de discriminant, zoals berekend in regel 520 wordt nu ofwel een foutmelding gegeven, ofwel de oplossing berekend. SQR(D) staat voor de wortel uit D. Deze funktie wordt in het volgende hoofdstuk nog verder besproken. In dit voorbeeld is tevens te zien dat het mogelijk is meer RETURN-statements in een subroutine te hebben. Nadat het programma de eerstvolgende RETURN is tegengekomen wordt teruggesprongen naar regel 60 waar de waarden worden uitgeprint.

Het is mogelijk om binnen een subroutine een andere subroutine aan te roepen (nesten). Dit kan men 20 keer doen. Gebeurt het vaker, dan zal er een foutmelding op de terminal verschijnen.

Evenals met het GOTO-statement, is het mogelijk de subroutine aanroep te laten afhangen van de waarde van een expressie. Dit gebeurt met het ON-GOSUB commando. Algemene vorm:

<regelnummer> ON expressie GOSUB regelnummers

Enkele voorbeelden van dit commando:

```
20 ON A GOSUB 1000,2000,3000
100 ON X1 GOSUB 990,300,5000,1200,1300
200 ON X3/A9+3 GOSUB 1000,4000,2000,4000
```

Evenals bij het ON-GOTO commando wordt hier eerst de expressie uitgewerkt, dan afgebroken naar een geheel getal, en aan de hand van

deze waarde zal gesprongen worden naar het bijbehorend alternatief. Indien men alleen van een bepaalde subroutine gebruik wenst te maken als aan een voorwaarde is voldaan, kan met gebruik maken van de IF-GOSUB mogelijkheid. Algemene vorm:

<regelnummer> IF voorwaarde GOSUB regelnummer

Deze sprong zal alleen worden uitgevoerd als aan de voorwaarde is voldaan. Voorbeelden:

```
10 IF A>1 GOSUB 200
100 IF A$=C$ GOSUB 1000
133 IF X<0 GOSUB 500
```

Een klein voorbeeldje is nu het volgende

```
LISTNH
10 REM IF-GOSUB
20 PRINT "GEEF GETAL";
30 INPUT A
40 IF A<0 GOSUB 100
50 IF A>0 GOSUB 200
60 IF A<>0 GOTO 20
100 REM EERSTE SUBROUTINE
110 PRINT A;"IS KLEINER DAN 0"
120 RETURN
200 REM TWEEDE SUBROUTINE
210 PRINT A;"IS GROTER DAN 0"
220 RETURN
500 END

READY
```

Voor dit soort opdrachten is het natuurlijk niet nodig een programma met subroutines te schrijven. Bij langere programma's kan het echter wel vaak de opzet van het programma verduidelijken. Men krijgt dan vaak een vrij kort hoofdprogramma en een aantal aparte subroutines.

N.B. ON-GOSUB en IF-GOSUB kunnen op de DEC-10 niet worden gebruikt.

Indien men een subprogramma gaat schrijven dat slechts uit 1 regel bestaat en waarin slechts 1 waarde wordt berekend, kan men beter gebruik maken van de zogenaamde funkties. Dit zijn rekenopdrachten

die tevoren met een apart commando in het programma moeten zijn gedefinieerd. De funktie wordt als volgt gedefinieerd:

<regelnummer> DEF FNa(var1,{var2..var5})=expressie

waarin a een letter van het alfabet is, var1 t/m var5 dummy variabelen zijn en expressie voor een rekenkundige of string-bewerking staat. In het algemeen zal deze bewerking gebruik maken van de variabelen die in de aanhef worden gegeven. Een funktie moet tenminste 1 argument hebben. Dit hoeft echter niet te worden gebruikt, zoals uit de voorbeelden duidelijk zal worden. Als een expressie gebruik maakt van variabelen die niet als argument zijn opgegeven, wordt de waarde gebruikt die de variabele op dat ogenblik heeft. Enkele voorbeelden:

```
10 DEF FNX(X,G)=X^G
30 DEF FNC(D)=(D-A)/C
40 DEF FNU(X,Y)=0,5*(FNC(X)+FNX(Y))
50 DEF FNK(A,B,C)=A*A+B*B+C*C
60 DEF FNS$(A$,B$)=A$&B$
```

Een DEF-opdracht mag overal in een hoofdprogramma worden geplaatst. In de meeste gevallen is het aan te bevelen ze aan het begin van een programma te zetten. Zoals uit de voorbeelden blijkt mag een funktie gebruik maken van een andere funktie. Ook strings kunnen op deze manier worden behandeld. Zo zal FNS$ twee strings A$ en B$ met elkaar verbinden tot een string (&). Enkele voorbeelden van het aanroepen van bovenstaande funkties:

```
110 F$=FNS$(X$,Z$)
120 Y=FNX(X,2)
230 K=A+FNK(U,V,W)/FNC(E)
```

In regel 110 zullen de strings X$ en Z$ worden samengevoegd tot de string F$. In regel 120 zal de variabele Y de waarde krijgen van X in het kwadraat. In regel 230 wordt een combinatie van funkties gebruikt alsof het variabelen zijn. Ook dit is toegestaan. De namen van variabelen in een DEF en in de aanroep hoeven niet gelijk te zijn. BASIC zal in plaats van de zogenaamde dummy variabelen in het argument van een DEF funktie zelf de waarden invullen van de variabelen die door de gebruiker in de aanroep worden gewenst. Wel moet

men ervoor waken geen strings in een aanroep te plaatsen als in het DEF statement rekenkundige variabelen zijn gegeven of omgekeerd. In dit geval zal er een foutmelding worden ontvangen.

Een voorbeeldje met funkties:

```
LISTNH
10 REM FUNKTIES
20 DEF FNT(A,P)=A+0.01*P*A
30 PRINT "WAT IS UW NAAM";
40 INPUT N$
50 PRINT "HOEVEEL KAPITAAL HEEFT U";
60 INPUT K
70 U=FNT(K,6)
80 PRINT
90 PRINT "WAARDE HEER ";N$;","
100 PRINT "ALS U UW KAPITAAL BIJ ONZE BANK BELEGT, IS HET NA"
110 PRINT "EEN JAAR GEGROEID TOT";U;"GULDEN."
120 END

READY
RUNNH


WAT IS UW NAAM ?JANSEN
HOEVEEL KAPITAAL HEEFT U ?100

WAARDE HEER JANSEN,
ALS U UW KAPITAAL BIJ ONZE BANK BELEGT, IS HET NA
EEN JAAR GEGROEID TOT 106 GULDEN.


TIME:  0.05 SECS.

READY
```

Dit programma berekent het nieuwe kapitaal van iemand als hij het een jaar op de bank zet tegen 4% rente. FNT berekent het nieuwe kapitaal en FNN koppelt de strings 'WAARDE HEER' en de N$ aaneen.

N.B. Op de DEC-10 is het gebruik van strings in funkties niet altijd toegestaan. Zie hiervoor de desbetreffende handleidingen.

12. BASIC - FUNKTIES

Voor het uitvoeren van een aantal standaardberekeningen beschikt BASIC over zogenaamde standaardfunkties. Deze werken op identieke wijze als de in het vorige hoofdstuk beschreven funkties, maar hebben het voordeel dat ze niet door de gebruiker tevoren hoeven te worden gedefinieerd. Het aantal standaardfunkties kan van computer tot computer verschillen. Er is echter een kleine groep die op elke computer aanwezig is. Deze groep zal hier in het kort worden besproken.

SGN(X)	bepaalt het teken van X
ABS(X)	neemt de absolute waarde van X
INT(X)	berekent het integer deel van X
TAB(X)	plaatstabulatorstop (geeft X spaties)
SQR(X)	berekent de vierkantswortel van X
EXP(X)	bepaalt de waarde e^X (e=2.71828)
LOG(X)	bepaalt de natuurlijke logaritme van X
LOG10(X)	bepaalt de logaritme met basis 10
SIN(X)	berekent de sinus van X
COS(X)	berekent de cosinus van X
TAN(X)	berekent de tangens van X
ATN(X)	berekent de arctangens van X
RND(X)	geeft een willekeurig getal tussen 0 en 1

In dit hoofdstuk zal van elke funktie een voorbeeldje worden gegeven. Dit zullen in de meeste gevallen een beetje flauwe voorbeelden zijn die alleen bedoeld zijn om een inzicht in de werking van deze funkties te geven.

a. SGN,ABS en INT

De funktie SGN kan drie waarden aannemen:

-1 als X < 0
 0 als X = 0
 1 als X > 0

De funktie ABS geeft de absolute waarde van een getal, dit wil zeggen als het getal negatief is, zal het teken worden omgedraaid, ofwel

ABS(X)=X als X>=0

ABS(X)=-X als X < 0

Een programma om dit te illustreren:

```
LISTNH
10 REM FUNKTIES SGN EN ABS
20 PRINT "GEEF GETAL";
30 INPUT A
40 B=SGN(A)
50 C=ABS(A)
60 PRINT A,B,C
70 IF A<>0 GOTO 20
80 END

READY
RUNNH


GEEF GETAL ?3
 3             1             3
GEEF GETAL ?-4
-4            -1             4
GEEF GETAL ?-11
-11           -1             11
GEEF GETAL ?0
 0             0             0


TIME:  0.05 SECS.

READY
```

De funktie INT rondt een gebroken getal naar beneden af om er een integer van te maken. Dit moet vooral bij negatieve getallen in de gaten worden gehouden. Zo is INT(-2.2) = -3 en INT(2.2)=2, en zal men bij voorkeur kiezen:

```
20 X1=INT(X)
30 IF X>0 GOTO 50
40 X1=X1+1
50 ...
```

Dit stukje programma rondt getallen af zoals we dat in het normale rekenwerk gewend zijn. In het voorbeeld bij de goniometrische funkties wordt van deze funktie gebruik gemaakt.

b. TAB

De TAB funktie werkt identiek aan de tabulator op een schrijfmachine. De printkop of cursor op een terminal zal X posities overslaan (spaties plaatsen) als in een programma PRINT TAB(X) voorkomt. X mag slechts waarden tussen 0 en 72 hebben.

Een voorbeeld van deze funkties zal worden gegeven bij de goniometrische funkties.

c. SQR, EXP, LOG en LOG10

Bij deze funkties hoeft weinig gezegd te worden, daar zij voor zichzelf spreken. Bij de funkties SQR, LOG en LOG10 moet men wel oppassen dat het argument X niet negatief kan worden. Is dit namelijk het geval, dan zal een foutmelding worden gegeven en zal het programma stoppen.

Een klein programma om deze funkties te illustreren is het volgende:

```
LISTNH
10 REM BASIC FUNKTIES
20 READ A
30 DATA 2
40 B=SQR(A)
50 C=EXP(B)
60 D=LOG(C)
70 E=LOG10(D)
80 PRINT A,B,C,D,E
90 END

READY
RUNNH

 2             1.41421       4.11325       1.41421       0.150515


TIME:  0.02 SECS.

READY
```

In regel 20 wordt A ingelezen. Dit had ook met een direkt assignment statement (A=2) gekund. In regel 40 wordt aan B de waarde gegeven van de wortel uit 2. C wordt vervolgens berekend als de e-macht van wortel 2. In regel 60 wordt hier weer de natuurlijke logaritme van genomen, zodat het eer te getal weer terugkomt. Van dit getal wordt dan in regel 70 de 10-log genomen. In regel 80 worden de waarden uitgeprint.

d. De goniometrische funkties

Alle goniometrische funkties in BASIC zijn gebaseerd op radialen. Dit betekent dat als men in graden wil werken, de bekende omreken-faktor 2.πi/360 moet worden gebruikt. Nu is bekend dat de arctangens van 1 gelijk is aan π/4. Dus kan men de funktie ATN gebruiken voor het bepalen van π. Het is natuurlijk ook mogelijk de waarde π in te typen (π=3.14159), maar dit zal meestal meer werk zijn, daar de gemiddelde nederlander π slechts tot op 2 of 3 decimalen nauwkeurig kent, zodat deze dan eerst opgezocht moet worden.

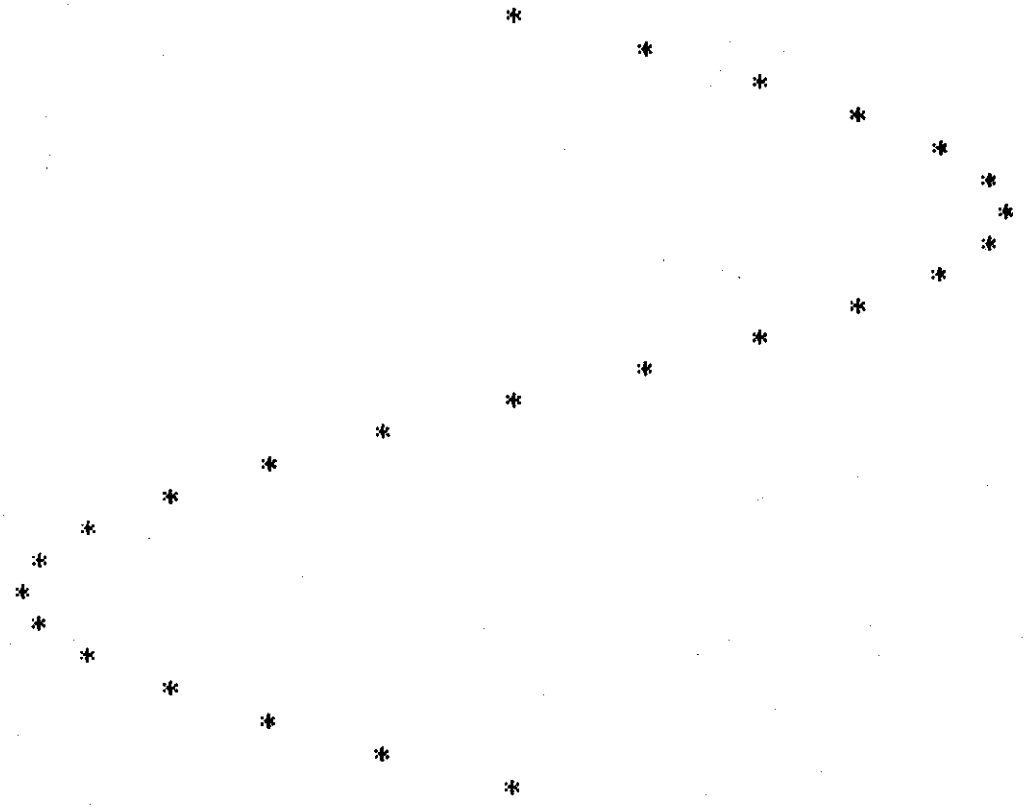
Een programmavoorbeeld van het gebruik van deze funkties:

```
LISTNH
10 REM VOORBEELD GONIO FUNKTIES
20 P=ATN(1)
30 F=P/45
40 PRINT "GEEF HOEK IN GRADEN";
50 INPUT H
60 REM OMREKENEN GRADEN NAAR RADIALEN
70 H=F*H
80 REM SINUS,COSINUS EN TANGENS
90 PRINT "SINUS IS",SIN(H)
100 C=COS(H)
110 PRINT "COSINUS IS",C
120 PRINT "TANGENS IS",TAN(H)
130 END

READY
RUNNH


GEEF HOEK IN GRADEN ?120
SINUS IS        0.866025
COSINUS IS     -0.5
TANGENS IS     -1.73205


TIME:  0.03 SECS.

READY
```

In regel 20 wordt de waarde van $\pi/4$ berekend. In regel 30 wordt de omrekenfaktor van graden naar radialen berekend. Regel 70 rekent de graden om naar radialen. Let op dat hier dezelfde variabele wordt gebruikt. Dit kan alleen omdat de hoek in graden verder in het programma niet meer wordt gebruikt. In regel 90 wordt de waarde van de sinus direkt geprint. Bij de cosinus is de ietwat omslachtiger methode gekozen om de waarde van de cosinus eerst aan een variabele toe te kennen en daarna de waarde van de variabele pas uit te voeren. Beide methodes werken evengoed. Als de waarde van de cosinus verderop in het programma nog gebruikt zou worden, is de tweede methode aan te bevelen. Dan hoeft de waarde maar een maal berekend te worden en kan verder C direkt worden gebruikt. Als het slechts een eenmalig benodigde waarde is, is de eerste methode aan te bevelen daar deze een regel minder programma vraagt.

Een tweede voorbeeld is er een die men ook vaak tegenkomt in boeken over programmeerbare rekenmachientjes met aangesloten printertje. Dit programmaatje tekent een sinuscurve.

```
LISTNH
10 REM TEKENEN VAN SINUSOIDE
20 D=8*ATN(1)/24
30 U=-D
40 FOR T=1 TO 25
50 U=U+D
60 A=SIN(U)
70 N=35+INT(30*A+0.4999)
80 PRINT TAB(N);"*"
90 NEXT T
100 END

READY
RUNNH
```



```
TIME:  0.08 SECS.

READY
```

In regel 20 wordt de stapgrootte berekend. Er worden 25 punten geprint over het trajekt 0 tot 2π. Dit wil zeggen dat er 24 intervallen tussen zitten. In regel 50 wordt direkt begonnen met het ophogen van de waarden, vandaar dat in regel 30 eerst de waarde gelijk aan de negatieve stapgrootte wordt gezet, zodat de eerste waarde van het argument van de sinus 0 wordt. In regel 60 wordt de waarde van de sinus berekend. In regel 70 wordt de plaats van het te plotten symbool in de output (gekozen is '*') berekend. Dit gebeurt aan de hand van het feit dat de sinus altijd waarden tussen -1 en +1 aanneemt. Als de waarde 0 is, zal de * op positie 36 worden geplaatst. Als de sinus -1 is, wordt de * op positie (35-30)+1=6 geplaatst. Als de sinus +1 is, zal de * op positie (35+30)+1=66 worden geplaatst.

e. RND

Met behulp van de funktie RND kan men willekeurige (random) getallen genereren. Deze getallen zijn uniform verdeeld tussen 0 en 1 (0 <= getal < 1), dat wil zeggen dat alle getallen tussen 0 en 1 dezelfde kans hebben om 'aangemaakt' te worden. Het argument X heeft geen doel, en mag elk willekeurig getal zijn. Dit is alleen gebruikt om alle standaard funkties eenzelfde vorm te geven. Bij het gebruik van een randomgenerator moet nog even worden opgemerkt, dat iedere keer dat het programma wordt gestart, met hetzelfde getal wordt begonnen. Een klein voorbeeld ter illustratie:

```
LISTNH
10 REM RANDOM GENERATOR
20 X=RND(0)
30 PRINT X
40 END

READY
RUNNH

 0.217873


TIME:  0.02 SECS.

READY
RUNNH

 0.217873


TIME:  0.02 SECS.

READY
```

Dit kan worden vermeden door het toevoegen van het RANDOMIZE commando. Als dit wordt toegevoegd, zal het programma iedere keer dat het wordt gestart met een andere waarde beginnen. De algemene vorm van dit commando is

<regelnummer> RANDOMIZE

en het voorgaande voorbeeldje wordt nu

```
LISTNH
10 REM RANDOMIZE
20 RANDOMIZE
30 X=RND(0)
40 PRINT X
50 END

READY
RUNNH

 0.944181


TIME:  0.02 SECS.

READY
RUNNH

 3.49399E-2


TIME:  0.01 SECS.

READY
```

Met een IF-statement zou men RANDOMIZE kunnen uitschakelen om in de testfase van het programma over vooraf vast te stellen waarden te kunnen beschikken.

Een wat uitgebreider voorbeeld is het volgende:
Iemand gooit duizend keer met een dobbelsteen. Hoe zal (bij een eerlijke dobbelsteen) de score van ieder puntenaantal zijn? De verwachting is dat ieder vlakje min of meer eenzelfde aantal malen boven zal liggen.
Het volgende programma kan hiervoor worden gebruikt.

```
LISTNH
10 REM DOBBELSTEEN
20 DIM N(6)
30 RANDOMIZE
40 REM SCHOONMAKEN VAN TELLERS
50 FOR T=1 TO 6
60 N(T)=0
70 NEXT T
80 REM HET GOOIEN
90 FOR T=1 TO 1000
100 I=6*RND(1)+1
110 N(I)=N(I)+1
120 NEXT T
130 REM UITVOER
140 FOR T=1 TO 6
150 PRINT "ER IS";N(T);"MAAL";T;"GEGOOID"
160 NEXT T
170 END

READY
RUNNH


ER IS 180 MAAL 1 GEGOOID
ER IS 180 MAAL 2 GEGOOID
ER IS 146 MAAL 3 GEGOOID
ER IS 156 MAAL 4 GEGOOID
ER IS 176 MAAL 5 GEGOOID
ER IS 162 MAAL 6 GEGOOID


TIME:  0.11 SECS.

READY
```

Het zal de lezer zijn opgevallen dat N(0) niet gebruikt is. Indien dit wel het geval was geweest, was de +1 in regel 100 ook overbodig geweest. Deze zorgt er nu voor dat het laagste getal wat gemaakt kan worden 1 is. Uit programmatechnisch oogpunt bekeken was deze oplossing beter geweest, maar uit overwegingen van duidelijkheid is voor de bovenstaande oplossing gekozen. N(I) geeft nu het aantal malen aan dat het getal I is gegooid.

Tot zover dit hoofdstuk over funkties en subprogramma's. Afhankelijk van de computer waarop men werkt kunnen er meer of minder funkties zijn die de gebruiker ter beschikking staan. Bovenstaande

funkties zijn de meest algemene, die op praktisch elke computer aanwezig zijn. De gebruiker wordt naar de bij de computer behorende handboeken verwezen waarin de volledige set van te gebruiken funkties in de regel staat opgegeven.

13. DATA OPSLAG

Een computer zou van weinig nut zijn als men er geen data (tijdelijk) in op kon slaan en later weer kon teruglezen. Deze data zal meestal naar een datafile op schijf worden geschreven of ervan worden teruggelezen. Alvorens de data te lezen of te schrijven zal men de computer moeten vertellen welke datafile moet worden gebruikt. Bij RT-11 BASIC wordt de lineprinter ook als datafile beschouwd. In het programma gebruikt men nu het OPEN statement, dat de naam van een file of device verbindt met een kanaalnummer. Aan de hand van dit kanaalnummer, dat bij een lees of schrijfopdracht wordt gegeven, zal de data dan worden weggeschreven of ingelezen van de correcte file.
De algemene vorm van de OPEN statement is:

<regelnummer> OPEN "dev:filnam.ext" {FOR INPUT} AS FILE#digit

of

<regelnummer> OPEN "dev:filnam.ext" {FOR OUTPUT} AS FILE#digit

Waarin dev:filnam.ext de naam van de datafile op device dev: is met extensie .ext. Als de extensie wordt weggelaten zal deze automatisch .DAT worden. Indien FOR INPUT of FOR OUTPUT worden weggelaten, zal het afhangen van de eerste opdracht in het programma of er een nieuwe file wordt gemaakt (WRITE) of dat het programma een oude datafile verwacht (INPUT). Digit is het kanaalnummer. Dit getal mag liggen tussen 0 en 7 (incl.). Als men met kanaal 0 werkt, zonder dat dit kanaal gedefinieerd is, zal de computer kanaal 0 aan de terminal toekennen. In dit geval wordt echter geen ? getypt als er input wordt verwacht. Er zijn nog meer mogelijke toevoegingen aan het OPEN statement mogelijk, zoals de lengte van de file, de lengte van de buffer, enz. Daar deze in de regel toch niet worden gebruikt, zullen zij ook niet worden behandeld.

Het inlezen van data van een file geschiedt op analoge wijze als het inlezen van data van de terminal. De algemene vorm van het lees-statement wordt nu echter:

<regelnummer> INPUT ≠digit: {variabelen}

en het wegschrijven naar deze file:

<regelnummer> PRINT ≠ digit: {variabelen}

Hierin geeft disit het kanaal aan waarvan de data moeten worden gelezen. De dubbele punt (:) scheidt het kanaalnummer van de variabelenlijst. Enkele voorbeelden zijn nu:

```
LISTNH
10 REM LEZEN EN SCHRIJVEN NAAR SCHIJF
20 OPEN "NAAM.DAT" FOR INPUT AS FILE #1
30 OPEN "LEEFT" AS FILE #2
40 OPEN "NEW.NEW" AS FILE #3
50 FOR I=1 TO 10
60 INPUT #1:N$
70 INPUT #2:L
80 PRINT #3:N$,",",L
90 NEXT I
100 CLOSE #1
110 CLOSE #2
120 CLOSE #3
130 END

READY
```

Bij dit programma is ervan uitgegaan dat er twee datafiles zijn, een met namen en een met bijbehorende leeftijden. Bij deze OPEN opdrachten wordt geen device gegeven. In zulke gevallen neemt de computer automatisch de schijf waarop de gebruiker zijn programma's en data heeft staan. Regel 30 heeft geen extensie opgegeven, zodat deze file LEEFT.DAT zal moeten heten. Er is zowel bij kanaal 2 als bij kanaal 3 geen input of output opgegeven. Daar bij kanaal 2 in regel 30 geen INPUT of OUTPUT gegeven is, en in regel 70 INPUT#2 is gegeven, zal de computer de file LEEFT.DAT als input file beschouwen. Om dezelfde reden wordt NEW.NEW als outputfile beschouwd. De extra komma in regel 80 die wordt geschreven tussen de data is als voorzorgsmaatregel bedoeld, daar anders later de data die op

een regel is geschreven niet meer als aparte data kan worden gelezen. Bij het lezen van terminal moeten de gegevens immers ook door komma's worden gescheiden.

Na het inlezen of wegschrijven van data moeten de files die bewaard moeten worden, worden afgesloten door middel van het CLOSE commando. Algemene vorm:

<regelnummer> CLOSE #digit

Als men dit niet doet, kan het zijn dat de data verloren gaan, afhankelijk van het soort computer. In ieder geval zal men van het CLOSE commando gebruik moeten maken als men meer dan 8 datafiles moet gebruiken. Dan zal elke keer een van de files moeten worden afgesloten voordat een volgende kan worden geopend.

De datafile die in het bovenstaande voorbeeld is gemaakt kan nu weer worden ingelezen door een programma'tje dat de namen en leeftijden op de printer afdrukt (in RT-11 BASIC):

```
LISTNH
10 REM LEZEN VAN GEMAAKTE FILE
20 OPEN "NEW.NEW" FOR INPUT AS FILE #2
30 OPEN "LP:" FOR OUTPUT AS FILE #3
40 FOR T=1 TO 10
50 INPUT #2:N$,L
60 PRINT #3:N$,L
70 NEXT T
80 CLOSE #2
90 CLOSE #3
100 END

READY
```

Dit programma'tje geeft de data die zijn ingelezen van de file NEW.NEW weer op de printer. Als in het vorige voorbeeld geen komma was geplaatst tussen de twee variabelen die op een regel werden weggeschreven, zou nu in dit programma in regel 50 een foutmelding veroorzaken.

Een laatste commando dat met het lezen en schrijven van data te maken heeft is het RESTORE commando. Algemene vorm:

<regelnummer> RESTORE #digit

waarbij #digit weer naar een kanaalnummer verwijst, zoals gedefinieerd in een OPEN statement.

Als #digit wordt weggelaten zal, zoals eerder in hoofdstuk 6 is behandeld, de READ-DATA procedure weer van voren af aan beginnen. Bij het transport van de data van en naar peripherie kan men de werking van het RESTORE commando het best begrijpen als men ervan uitgaat dat bij het lezen en schrijven een teller wordt bijgehouden die aangeeft de hoeveelste regel wordt gelezen of geschreven. Geeft men nu een RESTORE commando, dan zal de teller weer op 1 worden gezet. Op deze manier kan een datafile meerdere keren worden gelezen zonder hem iedere keer te moeten sluiten en openen, wat (relatief) nogal veel tijd kost. Bij het wegschrijven van data is dit natuurlijk een zaak om goed mee uit te kijken, want indien in de file al data stonden, worden deze overschreven en zijn dan voor verder gebruik verloren gegaan.
DEC-10 BASIC werkt in plaats van met het OPEN-statement met het FILES-statement. Een voorbeeldje met DEC-10 BASIC:

```
LISTNH
10 REM DEC-10 BASIC FILE HANDLING
20 FILE #3:"DATA.DAT"
30 SCRATCH #3
40 FOR T=1 TO 10
50 K=T*T
60 D=K*T
70 U4=D*T
80 U5=U4*T
90 PRINT #3: T,K,D,U4,U5
100 NEXT T
110 END

READY
```

Het SCRATCH commando dient om aan te geven, dat eventuele oudere files met dezelfde naam van schijf verwijderd mogen worden.

Voor een uitgebreidere beschrijving van het FILES-commando op de DEC-10 en het gebruik van andere soorten files, zoals virtual files (RT-11) en direct-access files (DEC-10) kan worden verwezen naar de desbetreffende handleidingen.

14. IMMEDIATE MODE BASIC

Het is mogelijk met behulp van BASIC de computer als normale rekenmachine te gebruiken. Dit gebeurt door een van de eerder besproken commando's zonder regelnummer in te typen. Op die manier wordt de opdracht direkt uitgevoerd. Het is natuurlijk niet mogelijk om op deze manier met lussen te werken, maar dat is dan ook niet de bedoeling. Het is alleen de bedoeling om wat kleine berekeningen gemakkelijk te kunnen voeren. Bijvoorbeeld:

```
PRINT 3*4
      12
```

Na de return zal op het beeld dadelijk het produkt verschijnen. Een iets ingewikkelder voorbeeld:

```
A=SIN(.5)
B=COS(.5)
PRINT A*A+B*B
      1
```

In geheugenplaats A zal nu de waarde van de sinus van .5 worden geplaatst, in geheugenplaats B de cosinus. Daar de som van het kwadraat van sinus en cosinus altijd 1 is, zal deze waarde ook op het beeld verschijnen.

Deze mogelijkheid van BASIC zal niet vaak gebruikt worden, maar is soms gemakkelijk als men enige resultaten snel en eenvoudig wil controleren.

15. NAWOORD

Deze nota kan natuurlijk niet volledig de hele taal BASIC behandelen. De belangrijkste eigenschappen heb ik er echter uitgelicht. Ik wil er nogmaals op wijzen dat er kleine verschillen zijn tussen BASIC voor de ene computer en BASIC voor de andere computer. De basisideeën zijn echter voor alle computers gelijk. In de hierna

volgende appendices worden de belangrijkste commando's voor RT-11 BASIC, RSX-BASIC en DEC-10 BASIC gegeven.

Zoals ook al in de inleiding is gesteld, kan men door het lezen van een nota geen BASIC leren. Men kan de basisprincipes leren, maar het eigenlijke programmeerwerk moet achter een terminal worden uitgeprobeerd. Hiervoor zal men vaak de handleiding van de betreffende computer moeten raadplegen. Na verloop van tijd zal men dan ook de kleinere trucjes van het programmeren leren kennen. Het grote voordeel van BASIC is dat men met het eigenlijke operating system (wat vaak ingewikkeld kan zijn) niets te maken heeft. Vanuit de BASIC interpreter kunnen alle commando's worden gegeven die nodig mochten zijn.

LITERATUUR

AMSTEL, J.J. VAN, red.: Programmeerproblemen, technieken en opgaven. EIT Diktatenserie 2. Academic Service, Den Haag, 1976

_____ Cursus BASIC. Elektronika Opleidingen Dirksen, Arnhem, 1978.

_____ Cursus Computertechnicus-C. Elektronika Opleidingen Dirksen, Arnhem, 1980

_____ Cursus Micro-processors/Micro-computers Elektronica Opleidingen Dirksen, 1977

Appendix A

DE ASCII-CODE

Decimal Value	ASCII Character	Usage
Ø	NUL	FILL character
1	SOH	
2	STX	
3	ETX	CTRL/C
4	EOT	
5	ENQ	
6	ACK	
7	BEL	BELL
8	BS	
9	HT	HORIZONTAL TAB
1Ø	LF	LINE FEED
11	VT	VERTICAL TAB
12	FF	FORM FEED
13	CR	CARRIAGE RETURN
14	SO	
15	SI	CTRL/O
16	DLE	
17	DC1	
18	DC2	
19	DC3	
2Ø	DC4	
21	NAK	CTRL/U
22	SYN	
23	ETB	
24	CAN	
25	EM	
26	SUB	CTRL/Z
27	ESC	ESCAPE[1]
28	FS	
29	GS	
3Ø	RS	
31	US	
32	SP	SPACE
33	!	
34	"	
35	#	
36	$	
37	%	
38	&	
39	'	APOSTROPHE
4Ø	(	
41	)	
42	*	
43	+	
44	,	COMMA
45	-	
46	.	
47	/	
48	Ø	
49	1	
5Ø	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
6Ø	<	
61	=	
62	>	
63	?	
64	@	
65	A	
66	B	
67	C	
68	D	
69	E	
7Ø	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
8Ø	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
9Ø	Z	
91	[	
92	\	Backslash
93	]	
94	^	or ↑
95	_	or ←
96	`	Grave accent
97	a	
98	b	
99	c	
1ØØ	d	
1Ø1	e	
1Ø2	f	
1Ø3	g	
1Ø4	h	
1Ø5	i	
1Ø6	j	
1Ø7	k	
1Ø8	l	
1Ø9	m	
11Ø	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
12Ø	x	
121	y	
122	z	
123	{	
124	|	Vertical Line
125	}	
126	~	Tilde
127	DEL	RUBOUT

[1]ALTMODE (ASCII 125) or PREFIX (ASCII 126) keys which appear on some terminals are translated internally into ESCAPE.

RT-11 BASIC

The following key commands halt program execution, erase characters or delete lines.

Key	Explanation
ALTMODE	Deletes the entire current line. Echoes DELETED message (same as CTRL/U). On some terminals the ESC key must be used.
CTRL/C	Interrupts execution of a command or program and returns control to the RT-11 monitor. BASIC can be restarted without loss of the current program by using the monitor RE command.
CTRL/O	Stops output to terminal and returns BASIC to READY message when program or command execution is completed.
CTRL/U	Deletes the entire current line. Echoes DELETED message (same as ALTMODE).
←	(SHIFT/O) Deletes the last character typed and echoes a backarrow (same as RUBOUT). On VT05 or LA30 use the underscore (-) key
RUBOUT	Deletes the last character typed and echoes a backarrow (same as ←).

The following commands list, punch, erase, execute and save the program currently in core.

Command	Explanation
CLEAR	Sets the array and string buffers to nulls and zeroes.
LIST	Prints the user program currently in core on the terminal.

LIST line number

LIST -line number

LIST line number-[END]

LIST line number-line number

Types out the specified program line(s) on the terminal.

LISTNH line number

LISTNH -line number

LISTNH line number-[END]

LISTNH line number-line number

Lists the lines associated with the specified numbers but does not print a header line.

NEW "filnam" Does a SCRatch and sets the current program name to the one specified.

OLD"file" Does a SCRatch and inputs the program from the specified file.

RENAME "filnam" Changes the current program name to the one specified

REPLACE "dev:filnam.ext"

Replaces the specified file with the current program.

RUN Executes the program in core.

RUNNH Executes the program in core area but does not print a header line.

SAVE "dev:filnam.ext"

Outputs the program in core as the specified file.

SCRatch Erases the entire storge area.

RSX-BASIC

APPEND {file specification}

Merges the program in your area in memory with the program specified by the file specification.

CLEAR

Initializes all variables to 0 and all string variables to nulls and deletes arrays.

COMPILE {file specification}

Saves a compiled version of the program

DEL line specification {,line specification,...}

Deletes specified lines.

LENGTH

Prints on your terminal the size of the program in memory and the size of the remaining free memory.

LIST{NH}{line specification,line specification2,...}

Prints on the terminal the specified line(s) of the program currently in memory. NH suppresses the printing of the header line.

NEW {program name}

Erases your storage area and sets the current program name to the one specified.

OLD {file specification}

Erases your storage area and inputs the program from the specified file.

RENAME program name

Changes the current program name to the one specified.

REPLACE {file specification}

Replaces the specified file with the current program.

RESEQ[{new line number},{old line number 1}{-old line number 2},{increment}]

Resequences program as specified.

RUN{NH}

Executes the program in memory. NH suppresses the printing of the header line.

RUN{NH}file specification

Erases your storage area, inputs the program from the specified file, and then executes the program. Does not print header line in any case.

SAVE {file specification}

Outputs the program in memory to the specified file.

SCR

Erases your storage area and changes the program name to NONAME.

SUB line numberxstring1xstring2{xinteger}

Substitutes the integer occurrence of string1 with string2 on line specified. x is a delimiter and can be any character such as @.

UNSAVE file specification

Deletes specified file.

Key Commands

CTRL/C

Interrupts execution of a command or program and causes BASIC to print the READY message. See your BASIC-11 user's quide for more information about CTRL/C.

CTRL/O

Causes all further terminal output to be discarded. If an INPUT statement is encountered, CTRL/O is retyped, or the program is terminated, printing resumes.

CTRL/Q

Continues output to the terminal; cancels effect of CTRL/S.

CTRL/S

Temporarily suspends all output to terminal until CTRL/Q is typed; allows alphanumeric display terminals to be read or photographed before data is moved off screen.

CTRL/U

Deletes the entire current input line (provided the RETURN key has not been typed).

RUBOUT

Deletes the last character typed.

DEC-10 BASIC

THIS IS THE HELP FILE FOR DECSYSTEM-10 BASIC VERSION 17E.

THE FOLLOWING IS A SHORT (TWO PAGE) DESCRIPTION OF SOME OF THE MOST COMMONLY USED COMMANDS. FOR MORE INFORMATION, SEE THE BASIC MANUAL IN THE DECSYSTEM-10 SOFTWARE NOTEBOOKS.

BYE
LOGS THE USER'S JOB OFF THE SYSTEM.

CATALOG DEV:
LISTS ONTO THE USER'S TERMINAL THE NAMES OF THE USER'S FILES WHICH EXIST ON THE SPECIFIED DEVICE. EX: CATALOG DTA4:

COPY DEV:FILENM.EXT > DEV:FILENM.EXT
COPIES THE FIRST FILE ONTO THE SECOND.

DELETE LINE NUMBER ARGUMENTS
ERASES THE SPECIFIED LINES FROM CORE. FOR EXAMPLE:

```
DELETE 11,44-212,13
```

ERASES LINE 11, LINES 44 THROUGH 212, AND LINE 13. IF NO ARGUMENTS ARE SPECIFIED, AN ERROR MESSAGE IS RETURNED. (IT IS NOT NECESSARY TO USE THE DELETE COMMAND TO ERASE LINES. YOU CAN ERASE A LINE SIMPLY BY TYPING ITS LINE NUMBER AND THEN DEPRESSING THE RETURN KEY.).

LIST LINE NUMBER ARGUMENTS
LISTS THE SPECIFIED LINES OF THE PROGRAM CURRENTLY IN CORE ONTO THE USER'S TERMINAL. THE LINE NUMBER ARGUMENTS ARE OF THE FORM DESCRIBED UNDER THE DELETE COMMAND. IF NO ARGUMENTS ARE SPECIFIED, THE ENTIRE PROGRAM IS LISTED.

NEW DEV:FILENM.EXT
THE PROGRAM CURRENTLY IN CORE IS ERASED FROM CORE AND THE SPECIFIED FILENAME IS ESTABLISHED AS THE NAME OF THE "PROGRAM CURRENTLY IN CORE". N.B., AT THE END OF EXECUTION OF THIS COMMAND, NO LINES EXIST IN CORE.

OLD DEV:FILENM.EXT
THE PROGRAM CURRENTLY IN CORE IS ERASED FROM CORE AND THE SPECIFIED FILE IS PULLED INTO CORE TO BECOME THE NEW "PROGRAM CURRENTLY IN CORE".

QUEUE FILENM.EXT
QUEUES THE SPECIFIED FILE FROM THE USER'S DISK AREA FOR OUTPUT TO THE LINE PRINTER. TWO OPTIONAL SWITCHES AVAILABLE WITH THIS COMMAND ARE /UNSAVE AND /&COPIES, WHERE & IS A NUMBER FROM 1 TO 63. THE SWITCHES FOLLOW THE FILENM.EXT ARGUMENT; FOR EXAMPLE:

```
QUEUE OUT.A/UNSAVE/2COPIES
```

```
REPLACE
    SEE SAVE.

RESEQUENCE
    CHANGES THE LINE NUMBERS OF THE PROGRAM CURRENTLY IN CORE
    TO 10,20,30,....  (LINE NUMBERS WITHIN LINES (AS, GO TO 1000)
    ARE CHANGED APPROPRIATELY.).

RUN
    COMPILES AND EXECUTES THE PROGRAM CURRENTLY IN CORE.

SAVE DEV:FILENM.EXT
    WRITES OUT THE PROGRAM CURRENTLY IN CORE AS A FILE WITH
    THE SPECIFIED NAME.  BASIC WILL RETURN AN ERROR MESSAGE IF SAVE
    ATTEMPTS TO WRITE OVER AN EXISTING FILE; TO WRITE OVER A FILE
    YOU MUST TYPE "REPLACE" INSTEAD OF "SAVE".

SCRATCH
    ERASES FROM CORE THE PROGRAM CURRENTLY IN CORE.

SYSTEM
    EXITS FROM BASIC TO MONITOR LEVEL.  N.B., THE CONTENTS OF CORE
    ARE LOST.

UNSAVE DEV:FILENM.EXT
    DELETES THE SPECIFIED FILE.  MORE THAN ONE FILE CAN BE
    SPECIFIED; FOR EXAMPLE:
            UNSAVE DSK:ONE.F4, DTA4:TEST.BAK


IN THE COMMANDS ABOVE WHICH ACCEPT SUCH ARGUMENTS, IF "DEV:" IS OMITTED,
"DSK:" IS ASSUMED; IF ".EXT" IS OMITTED, ".BAS" IS ASSUMED; IF "EXT"
IS OMITTED, A NULL EXTENSION IS ASSUMED.  THE SAVE, REPLACE, AND UNSAVE
COMMANDS ALLOW THE "FILENM" PART OF THE ARGUMENT TO BE OMITTED, IN WHICH
CASE ".EXT" MUST BE OMITTED ALSO AND THE NAME AND EXTENSION OF THE
PROGRAM CURRENTLY IN CORE ARE ASSUMED.


THE KEYWORDS OF COMMANDS (CATALOG, LIST, ETC.) MAY BE ABBREVIATED
TO THEIR FIRST THREE LETTERS.  ONLY THE THREE LETTER ABBREVIATION
AND THE FULL WORD FORM ARE LEGAL;  INTERMEDIATE ABBREVIATIONS
SUCH AS CATAL, FOR EXAMPLE, ARE NOT ALLOWED.  IF AN INTERMEDIATE
ABBREVIATION IS USED, THE EXTRA LETTERS WILL BE SEEN AS PART OF THE
COMMAND ARGUMENT (BECAUSE BASIC DOES NOT SEE BLANK SPACES OR TABS AT
COMMAND LEVEL.).  FOR EXAMPLE: CATAL DSKB: WOULD BE SEEN AS A REQUEST
TO CATALOG THE DEVICE ALDSKB.  AN EXAMPLE OF A LEGAL ABBREVIATED
COMMAND IS: CAT DTA4:


WHENEVER BASIC FINISHES EXECUTING A COMMAND, IT TYPES "READY".  IT
DOES NOT ANSWER "READY" AFTER DELETING A LINE BY THE ALTERNATE METHOD
DESCRIBED UNDER THE DELETE COMMAND OR AFTER RECEIVING A LINE FOR THE
PROGRAM CURRENTLY IN CORE FROM THE USER'S TERMINAL.  (TO INSERT OR
REPLACE A LINE IN THE PROGRAM CURRENTLY IN CORE, SIMPLY TYPE THE LINE
AND THEN DEPRESS THE RETURN KEY.  BASIC DISTINGUISHES BETWEEN LINES
(WHICH MUST BE STORED OR ERASED) AND COMMANDS (WHICH MUST BE PROCESSED)
BY THE FACT THAT A LINE ALWAYS BEGINS WITH A DIGIT (PART OF THE LINE
NUMBER) WHEREAS COMMANDS NEVER DO.).


READY
```