

april 1979

Instituut voor Cultuurtechniek en Waterhuishouding
Wageningen

ASPECTEN van INFORMATIEVERWERKING

19

FORTRAN

EEN INLEIDING

dr. Ph.Th. Stol

Nota's van het Instituut zijn in principe interne communicatie-middelen, dus geen officiële publikaties.

Hun inhoud varieert sterk en kan zowel betrekking hebben op een eenvoudige weergave van cijferreeksen, als op een concluderende discussie van onderzoeksresultaten. In de meeste gevallen zullen de conclusies echter van voorlopige aard zijn omdat het onderzoek nog niet is afgesloten.

Bepaalde nota's komen niet voor verspreiding buiten het Instituut in aanmerking

ISN 178708-02



ASPECTEN VAN INFORMATIEVERWERKING

Deel	Titel	Auteur	Nota	Datum
1	Computerverwerking van lange reeksen getallen	J.B.H.M.van Gils	935	nov. 1976
2	Optimaliseren van parameters: Het gereedmaken van een functie voor toepassing in NLV	Ph.Th. Stol	943	febr. 1977
3	Registratieverwerking voor automatische pF-bepalingen	J.B.H.M.van Gils	949	febr. 1977
4	Het systematisch bepalen van de afgeleiden van een functie ten behoeve van hun programmering	Ph.Th. Stol	948	febr. 1977
5	Mogelijkheden en beperkingen bij het verzinnen van namen bij informatieverwerking	Ph.Th. Stol	1037	febr. 1978
6	Over het samenstellen van een computerprogramma voor het optimaliseren van parameters	Ph.Th. Stol	951	apr. 1977
7	De onbekend-kode in een datafile	mevr.G.van den Berg-Buitenhuis	989	juli 1977
8	Enkele methoden voor het op eenvoudige en overzichtelijke wijze systematisch aanpassen respectievelijk veranderen van functies en parameterwaarden in computerprogramma's	Ph.Th. Stol	984	aug. 1977
9	Copiëren van file naar file	mevr.G.van den Berg-Buitenhuis	1019	dec. 1977
10	Een korte toelichting op de mogelijkheden van gebruik van de nieuw te installeren terminal	Ph.Th. Stol	1023	dec. 1977
11	Een systematische samenvatting van SUEDE, de routine voor het bijwerken van files	Ph.Th. Stol	1026	dec. 1977

Deel	Titel	Auteur	Nota	Datum
12	Handleiding tot het gebruik in LOCAL-mode van de ICW Teleprint-terminal	J.R. Maassen Ph.Th. Stol	1044	mrt. 1978
13	Handleiding tot het gebruik in ON-LINE mode van de ICW Teleprint-terminal	J.R. Maassen Ph.Th. Stol	1051	apr. 1978
14	Praktijkvoorbeelden van het gebruik van de ICW Teleprint-terminal	J.R. Maassen Ph.Th. Stol	1052	mei 1978
15	Het gebruik van magneetbanden door middel van de terminal	Ph.Th. Stol	1054	apr. 1978
16	Het beheren van een niet te groot bestand met behulp van de terminal en floppy disks	Ph.Th. Stol	1060	mei 1978
17	Het programmapakket SPSS	J.R. Maassen	1069	aug. 1978
18	BATCH-verwerking door middel van de terminal	J.R. Maassen Ph.Th. Stol	1070	okt. 1978
19	Fortran, een inleiding	Ph.Th. Stol	1113	apr. 1979
20	Toepassingen van het programmapakket GENSTAT	W.van Doorne		1979
21	Over het bijwerken en tot executie brengen van een hoofdprogramma met bijbehorende subprogramma's	Ph.Th. Stol	1117	
22	A short introduction to the computer peripherals of the ICW and how to use them with the CYBER computer	J.G. Wesseling	1124	mei 1979

De nota's handelende over Aspecten van Informatieverwerking bevatten inlichtingen over de ontwikkeling van de informatieverwerking binnen het Instituut. Naast meer concluderende en toelichtende beschouwingen zal aandacht worden besteed aan het gebruik van programma's en programmapakketten en zullen zakelijke inlichtingen over praktijkervaring met en toepassing van de informatieverwerking worden gegeven

I N H O U D

	Blz.
VOORWOORD	
I. ALGEMENE INLEIDING	4
1. Inleiding	5
2. Taalgebruik	6
3. Woordenlijst	7
4. Een alledaags voorbeeld	12
5. Afspraak over naamgeving	17
6. Voorbeeld van een klein programma	18
7. Een programma uitvoeren	21
8. Getalstelsels	24
9. Te behandelen statements (systematisch)	26
10. Te behandelen statements (logisch)	27
II. HET PROGRAMMEREN	29
1. Modulair en gestructureerd programmeren	30
III. ALGEMENE AFSPRAKEN	32
1. Tekens gebruikt in Fortran	33
2. De regel-indeling	34
3. Regelverlenging	36
4. Beschrijvingen en toelichtingen	37
5. Genummerde opdrachten	37
6. Toepassingen	38
IV. GROOTHEDEN	40
1. Inleiding	41
2. Constanten	42
3. Variabelen	43

	Blz.
4. Integer en real variabelen in formules	45
5. Exponentiële versie/Floating point versie	47
6. Samenvatting, beperkingen	49
7. Logische grootheden	49
8. Hollerith constanten	51
9. Toepassingen	52
V. ARRAYS	56
1. Geïndiceerde variabelen	57
2. Het werken met indices	60
3. De grootte van arrays	61
4. Toepassingen	62
VI. PROGRAMMEREN VAN FORMULES	67
1. Rekenkundige uitdrukkingen	68
2. Relaties	71
3. Veel voorkomende functies	72
4. Vervangingsopdrachten	73
5. Toepassingen	77
VII. INTERNE BESTURING	80
1. Inleiding	81
2. Continuering	81
3. Onvoorwaardelijke sprongopdrachten	82
4. Voorwaardelijke sprongopdrachten	84
4.1. Numerieke IF	84
4.2. Logische IF	86
5. DO-loops	89
6. De implied DO	94
7. IF's in flowcharts	95
8. Toepassingen	98
VIII. INPUT EN OUTPUT	101
1. Inleiding	102
2. FORMAT algemeen	
2.1. FORMAT voor input	103
2.2. Enkele bijzondere gevallen	106
2.3. FORMAT voor output	107
F-versie	109
E-versie	109

	Blz.
3. READ en WRITE	112
4. Files en records	116
IX. HOOFD- EN SUBPROGRAMMA'S	118
1. Inleiding	119
2. Hoofdprogramma's	120
3. Routines	123
4. Statement functions	124
5. External functions	127
6. Subroutines	130
7. Toepassingen	133
X. ADMINISTRATIE	139
1. Inleiding	140
2. COMMON	140
3. DATA	142
4. Samenvatting	144
5. Toepassingen	144
XI. DE TERMINAL ALS INPUT/OUTPUT MEDIUM	149
1. Inleiding	150
2. Conversational mode	151
XII. NOGMAALS: HET PROGRAMMEREN	154
LITERATUUR	158
BIJLAGEN	
1. PDP-11 Fortran	160
2. Onder meer niet behandelde onderwerpen	161
3. Wat niet is toegestaan	162

FIGUREN	pag.
1. Schematisch overzicht over de koppeling tussen een hoofd- en een subprogramma	31
2. Voorschriften met betrekking tot de indeling van ponskaarten of van ponsdocument, respectievelijk met betrekking tot in te tikken informatie	35
3. Stroomschema ter verduidelijking van het gebruik van drie achtereenvolgende voorwaardelijke sprongoverdrachten	96

TABELLEN	pag.
1. Getallen van 0 - 16 (decimaal) in het 2-, 8- en 10-tallig stelsel	25
2. Schematisch overzicht over de opbouw van een FORTRAN hoofdprogramma en subprogramma's voor zover het de hier behandelde onderdelen betreft	28
3. Grondwaterstanden in cm-maaiveld	57
4. Veel voorkomende functies die in Fortran met naam oproepbaar zijn	73
5. Schematisch overzicht over de verschillende soorten routines en hun belangrijkste eigenschappen	125
6. Stralingsgegevens te gebruiken als input voor voorbeeld 9	128

VOORBEELDEN	pag.
1. Een programma dat $x = a + b$ berekent wanneer a = 5 en b = 6	19
2. Een programma dat $x = ab$ berekent voor a = 5, 10, ... en b = 6 totdat $x > 100$	21
3. Een programma dat $c^2 = a^2 + b^2$ berekent voor b = 4 en $1 \leq a \leq 5$	39
4. Het signaleren of het aantal bezoekers (B) aan een recreatie-object op een even dag (D) groter is dan 100 en op een oneven dag groter is dan 150	53
5. Als voorbeeld 4, nu compleet met stuuropdrachten	55
6. Als voorbeeld 5, nu met geïndiceerde variabelen	63
7. Het berekenen van gemiddelde waterstanden voor gegevens vermeld in tabel 3	64
8. Het bepalen van de frequenties waarmee waarnemingen in bepaalde klassen vallen	99
9. Berekening van verdamping uit globale straling met een hoofdprogramma en een external function	134
10. Berekening van verdamping uit globale straling met een hoofdprogramma en een subroutine	136
11. Het consequent gebruik van subroutines en hun koppeling via COMMON-statements	146

BIJLAGEN	Pag.
1. Enkele aantekeningen met betrekking tot de Fortran IV versie voor computers van het type PDP 11	160
2. Onder meer niet behandelde onderwerpen	161
3. Wat niet is toegestaan	162

VOORWOORD

In deze nota wordt een inleiding gegeven in het programmeren in de programmeertaal FORTRAN.

De belangrijkste onderdelen van deze taal zullen in het kort besproken worden. Er wordt vanuit gegaan dat de lezer na het bestuderen van de voor hem relevante onderdelen van deze nota, in staat moet zijn een compleet programma te schrijven en uit te voeren.

Het wordt echter aanbevolen, na kennis te hebben genomen van deze inleiding, regelmatig de uitvoerige handleidingen (manuals) te raadplegen, zie literatuurlijst. Hierdoor kan men zijn parate kennis vergroten en nieuwe mogelijkheden blijven ontdekken. In dit verband is het van belang op te merken dat de in de handboeken aangegeven en hier niet behandelde verdere mogelijkheden, in grote programma's veel sneller en efficiënter werken dan zelf ontworpen uitbreidingen. Het kan dus voordelig zijn om steeds eerst na te gaan of voor bepaalde bewerkingen of besturingen speciale opdrachten bestaan.

De hier gegeven FORTRAN versie is gebaseerd op Fortran IV, Extended, zoals van toepassing is op de Cyber-computers van IWIS-TNO, Den Haag, en van het ECN, Petten. Deze versie is niet sterk machine-gebonden: maar ook hier geldt weer dat kennisneming van de bij de te gebruiken computer behorende handleidingen aan het licht kan brengen of er meer mogelijkheden zijn dan wel of men zich beperkingen moet opleggen.

Enkele belangrijke verschillen met de Fortran IV versie voor computers van het type PDP-11 zullen in een bijlage worden vermeld (zie Bijlage 1).

Het 'hoe en waarom' van de verschillende onderdelen van een programmeertaal worden pas goed begrepen wanneer de gebruiker het geheel al enigszins heeft leren overzien. Het verdient dus aanbeveling deze inleiding eerst eens in zijn geheel door te nemen alvorens de verschillende hoofdstukken meer in detail te bestuderen.

Om aan te geven dat deze nota slechts een inleiding is worden in een bijlage de niet behandelde onderwerpen opgesomd. In de manuals kunnen hierover alle bijzonderheden worden opgezocht. (Zie Bijlage 2). In heel veel gevallen zal men aan hetgeen hier beschreven wordt voorlopig voldoende hebben.

Voorbeelden welke in een kader zijn geplaatst zijn volledige en geteste programma's. Op basis van deze voorbeelden kan de lezer een begin maken met het ontwerpen van eigen programma's. Bij deze voorbeelden was het niet altijd te vermijden dat gebruik moest worden gemaakt van nog niet behandelde onderwerpen. Daarvan is echter alleen op het eenvoudigste niveau gebruik gemaakt.

In Bijlage 3 tenslotte zijn de belangrijkste 'verboden' opgesomd. Het betreft hier specifieke combinaties van statements die de uitzonderingen zijn die bevestigen dat het aantal mogelijkheden in FORTRAN onbeperkt is.

Een computerprogramma is een geordende
verzameling opdrachten (instructies) met
het doel uitgaande van bepaalde gegevens
tot bepaalde resultaten te komen

Programmeren is het vervaardigen
van de bovenbedoelde verzameling

I

ALGEMENE INLEIDING

I.1. INLEIDING

Voor het verkrijgen van het juiste inzicht in wat programmeren inhoudt, moet duidelijk zijn dat schrijven van een programma niet betekent dat men een wiskundig probleem tot oplossing brengt. Wel kan men een opgelost probleem programmeren en automatisch verwerken.

Zo kan men geen Fortran-programma schrijven dat een gegeven vierkantsvergelijking of een differentiaalvergelijking oplost, maar wel een programma waarmee de numerieke uitwerking kan worden verkregen wanneer de wiskundige oplossing beschikbaar is.

Verder zijn het niet alleen numerieke berekeningen waarbij met voordeel van Fortran gebruik gemaakt kan worden. Ook moeten worden genoemd het ordenen en rangschikken van gegevens, het maken van tabellen, het vaststellen of gegevens aan bepaalde voorwaarden voldoen, de kwaliteits-controle op gegevenbestanden, enzovoort. Dit alles zijn voorbeelden van informatieverwerking waarvoor het zinvol kan zijn programma's te schrijven.

Het zal blijken dat relatief veel aandacht zal moeten worden besteed aan de wijze waarop de uit te voeren bewerkingen moeten worden georganiseerd. Ook de administratie van variabelen en van gegevens moet nauwgezet worden verzorgd. Een zorgvuldig opgezet programma zal eerder tot de gewenste resultaten leiden en makkelijker aan nieuwe wensen aangepast kunnen worden dan programma's die zonder systematische analyse van het te verwerken probleem zijn tot stand gekomen.

I.2. TAALGEBRUIK

In de informatieverwerking is een typisch vakjargon ontstaan dat sterk op het Engels is gebaseerd. Dit niet alleen door de herkomst van de apparatuur en de soft-ware systemen maar ook doordat de Engelse benamingen, door het karakter van de taal, korter zijn. Dit is dan weer van voordeel waar men zuinig met de beschikbare geheugen-ruimte moet omspringen. Voorbeelden zijn:

Werkstuk, karwei	JOB
Programma	PROGRAM
Gemeenschappelijk	COMMON
Ga naar	GO TO
Indien	IF
Roep op	CALL
Lees	READ
Schrijf	WRITE
Gegevens	DATA
Bestand	FILE
Magneetband	TAPE
Schijf	DISK
Eenheid	UNIT
Aanjager	DRIVER
Hulpmiddel (apparaat)	DEVICE

Het overvloedig gebruik van hoofdletters komt voort uit het feit dat het totaal aantal tekens op de verschillende hulpmiddelen om technische redenen beperkt is, en men in feite voor de communicatie met één lettertype kan volstaan.

Van een aantal veel gebruikte (Engelse) termen en uitdrukkingen volgt in I.3 een verklaring.

In het dagelijks spraakgebruik worden verschillende begrippen nogal eens afwisselend gebruikt waarbij de historische ontstaanswijze mede een rol speelt. Vaak wordt over 'kaart' (dat is ponskaart) gesproken waar 'regel' of 'statement' bedoeld wordt. In de regel zegt

men: 'elk programma moet met een PROGRAM-kaart beginnen' ook al is de wijze waarop men het programma aanbiedt een geheel andere dan de ponskaart, bijvoorbeeld: tape, of wordt het programma over de terminal ingetikt. In deze nota zal het woord 'kaart' dan ook in een zeer ruime betekenis worden gebruikt.

Evenzo wordt nog wel over 'ponsen' en 'ponsdocument' gesproken terwijl het invoer medium een ander is dan een ponskaart. Ook in dit opzicht is het dagelijks spraakgebruik wat slordig.

I.3. WOORDENLIJST

Argument	Variabele of uitdrukking die de waarde van een functie, of het resultaat van een opdracht bepaalt. Bijv. x in $y = f(x)$, $x + a$ in $y = \sin(x+a)$, $B > 5$ in $IF(B>5)$
Array	Geordende reeks grootheden van eenzelfde soort zoals $x_1, x_2, \dots, x_n$. In het bijzonder een vector of een matrix
Array-element	Een enkel exemplaar uit een array. Bijv. x_5 of, voor een matrix, x_{41} (vier-een)
b	kleine letter om blanks (spaties) aan te duiden bij het ponsen (bijv. 123bb89) om het juiste aantal posities aan te geven
Carriage control	Besturing(!) van de wagen van een schrijfmachine. De moderne printers kennen geen wagen meer, maar het besturen van de regelafstand, het vóórzetten van een nieuwe pagina, etc. wordt nog steeds zo aangeduid
Character	Teken, letter, cijfer, bijzonder teken. Voorbeelden: A, B, 1, 9, =, >, \$, enz.
Commando	Instructie via de terminal te geven aan de computer. Géén Fortran-opdracht

Compileren	Het door de computer vertalen van een (Fortran) programma in een code die de computer kan verwerken, de zogenaamde object-code. Compile fase. Compile file: file waarop het te vertalen programma wordt geplaatst
Computer	Electronische rekeneenheid
Computer-systeem	Computer met alle bijbehorende randapparatuur (kaartlezer, kaartponser, regeldrukker, tape-unit, plotter, etc.)
Control kaart	Besturings(!)-kaart. Opdracht op ponskaart waarmee het computer systeem kan worden bestuurd
Cursor	Rechthoekig merkteken op het beeldscherm van de terminal dat de op dat moment geactiveerde positie aanduidt
Decimal point	Letterlijk: decimale punt. 'Komma'. Voorbeeld 12.34
Default	Default-waarde; default-procedure. Waarde (of procedure) die toegekend (gevolgd) wordt wanneer de gebruiker zelf geen specificatie geeft. Standaard. Systeem-constante
Δ	Teken om het gebruik van de RETURN-toets op de terminal aan te duiden
Editor	Subroutine of programma waarmee informatie kan worden verbeterd en bijgewerkt, in de regel rechtstreeks op een file
Editten	Het verbeteren en bijwerken van informatie op file
E _x	End-of-text symbool dat op het beeldscherm van de terminal het einde van de over te zenden informatie aangeeft. Wordt verkregen door gelijktijdig aan te slaan /Control,C/

Execution	De fase waarin het computersysteem zich bevindt wanneer een vertaald (Fortran-) programma wordt uitgevoerd. Compile fase en executie fase moeten dus duidelijk worden onderscheiden al kunnen ze onmiddellijk achter elkaar worden uitgevoerd
Exponential version	Schrijfwijze waarbij een getal met decimalen en een macht van 10 wordt weergegeven. Voorbeeld: 12.13E+2 voor 1213.0 (zie fixed point, floating point)
Expression	Expressie, (e). Uitdrukking, formule
File	Bestand. Hier gebruikt als opslagmedium voor gegevens, programma's en resultaten. Files worden met het noemen van een zelfgekozen naam gecreëerd en eveneens door het noemen van de naam weer opgeroepen
Fixed point	Schrijfwijze waarbij de plaats van de komma in een getal bepaald wordt door de cijferreeks waaruit een getal bestaat. Tegenstelling: Floating point
Floating point	Zwevende (drijvende) komma. De plaats van de komma wordt niet door de cijferreeks waaruit een getal bestaat, bepaald. Tegenstelling: Fixed point
Format	Opgave van de layout van input-data en van output
Hardware	Apparatuurgedeelte van een computer systeem
Host-computer	Computer waarop een terminal is aangesloten
Identifier	Meestal een code om automatisch te controleren of de aanbieder van de job gerechtigd is het systeem te gebruiken. Ook: code die de gebruiker aan een file (of tape) meegeeft om zich als eigenaar bekend te maken

Initialiseren	Op beginwaarde zetten van variabelen of besturingsvariabelen (bijv. 0 of 1)
Informatie	Elke vorm van mededeling die voor de mens van betekenis is en door een electronisch systeem automatisch kan worden verwerkt
Job-control-language	Taal waarin de gebruiker de externe behoeften van zijn programma aan de computer mededeelt. Zie Commando
Justified, left -	Naar links tegen de kantlijn aangeschoven
Justified, right -	Naar rechts tegen de kantlijn aangeschoven
Keyword	Sleutelwoord
Label	Nummer toegekend aan een opdracht om er naar te kunnen verwijzen in een ander deel van het programma. Specificaties bevattend gedeelte van een tape
Leading (zero)	Voorafgaand aan. Bijvoorbeeld in de schrijfwijze 0023 zijn er twee leading zero's
Library	Hier: programma-bibliotheek. Verzameling programma's die aan het systeem kunnen worden opgevraagd
List	Opgave van de grootheden die ingelezen moeten worden, resp. die geprint moeten worden
Listen	Op de printer afdrukken van resultaten, meestal in de zin van: zonder meer van onverwerkte informatie een leesbare afdruk maken. Listen van een programma, listen van een kaartendeck, listen van een file. Zie ook Printen

Loop		Lus in een programma, soms in de betekenis van 'fuij' waar de besturing niet meer uitkomt
Merge		Verschillende files aan elkaar koppelen of bijeenbrengen
Nesten		In elkaar opnemen van overeenkomstige bewerkingen zoals $a(b+c(d+e(f+g)))$ welke 3-diep genest is
Operand	} {	In $a+b$ zijn a en b operands en is $+$ de operator
Operator		
Overschrijven		Informatie op een file schrijven of aan een variabele toekennen, zodanig dat de oude informatie daarbij verloren gaat
Password		Wachtwoord, alleen aan de gebruiker bekend. Code(-naam) door de gebruiker te geven aan files om deze te bewaken tegen onrechtmatig gebruik
Printen		Op de printer afdrukken van resultaten, meestal in de zin van: selectief afdrukken van waardevolle uitkomsten. Zie ook Listen
ZEOf		Einde van de informatie die via de terminal op een file in Den Haag moet worden geplaatst (procent end-of-file)
Queue		Aaneenschakeling van jobs die in het computersysteem op verdere bewerking wachten. Input-queue, output-queue. Remote output queue (queue die voor de terminal bestemde verwerkte jobs bevat)
Slash(/)		Deelstreep, scheidingsteken
Software		Programma's die los van de apparatuur ontwikkeld kunnen worden

Sprongopdracht	Fortran-opdracht die maakt dat de normale volgorde door het programma doorbroken wordt en dat met een ander gedeelte wordt verdergegaan
Statement	Eigenlijk: bewering. Hier voornamelijk instructie, opdracht. Fortran-statement. Een programma bestaat uit statements
Stuuropdracht	Opdracht om (onderdelen van) het computersysteem te besturen. Zie Commando
Subscripted variable	Geïndiceerde variabele, bijv. x_i
String	Keten, reeks. String of characters: keten (reeks) van aan elkaar aansluitende tekens, ongeacht de inhoud of betekenis. Bijv.: "RAPPORTb8" String van 9 characters inclusief een blank
Syntax	Grammatica, taaleigenschappen van een programmeertaal
Type	Hieronder wordt hier verstaan de type-aanduiding van grootheden. Te onderscheiden zijn bijvoorbeeld gehele getallen, decimale breuken, complexe getallen, enz.
Update	Updaten, up-to-date maken, bijwerken en verbeteren van informatie
User	Gebbruiker van een computer systeem. User-identificatie

I.4. EEN ALLEDAAGS VOORBEELD

Met een zeer alledaags voorbeeld zullen enkele veelvuldig voorkomende begrippen worden toegelicht.

Stel men neemt op het station Ede-Wageningen "een retourtje Utrecht". Men krijgt dan een kaartje dat voldoet aan de volgende

specificaties

Vertrekpunt(V) = Ede-Wageningen	(stationsconstante)
Eindbestemming (E) = Utrecht	(variabele)
Reistype (T) = dagretour	(variabele)
Klasse (K) = 2	(default)
Geldig op (G) = heden	(default)
D-trein toeslag (D) = geen	(default)

Het begrip 'default-waarde' wordt hiermede duidelijk. Men hoeft niet te specificeren; in de gevallen die het meest voorkomen ('normaal zijn') vult het systeem zelf de ontbrekende waarden in.

Een volledig bepaald dagreisbiljet (B) zou formeel geschreven kunnen worden als een functie B met een argument, dus

$B(V,E,T,K,G,D)$

Het argument bevat 6 variabelen, of meer algemeen 'elementen', soms ook zelf als argument aangeduid.

De symbolen V,E,T,K,G en D heten de dummy argumenten. Ze worden ten tijde van de bestelling van het kaartje vervangen door de werkelijke waarde.

Dezelfde bestelling in Driebergen geeft aanleiding tot

$B(\text{Driebergen}, \text{Utrecht}, \text{dagretour}, 2, \text{heden}, \text{geen})$

waarmee het kaartje weer geheel bepaald is.

Het is duidelijk dat elke grootheid in het argument zijn eigen plaats heeft, zijn eigen plaats moet hebben, om een automatische verwerking mogelijk te maken. Zo zal de specificatie E=2 niet relevant zijn en tot een onbegrepen uitkomst leiden. Alleen K=2 heeft hier betekenis.

Formeel kunnen we nog verschil maken tussen de uitdrukking die de werkelijke argumenten bevat en de uitdrukking waarin de dummy argumenten voorkomen.

Dus:

LEVER $B(V,E,T,K,G,D)$

GEVRAAGD $B(\text{Ede-Wag}, \text{Utrecht}, \text{enkele reis}, 2, \text{heden}, \text{geen})$

Wil men nu de gebruiker ontlasten van het opgeven van de normale waarden dan kan dat niet door het argument zonder meer in te korten. Zo is in de relatie

LEVER B(V,E,T,K,G,D)

GEVRAAGD B(Utrecht, dagretour)

niet zonder meer duidelijk welke dummy variabele nu welke werkelijke waarde krijgt.

Een eerste afspraak kan zijn dat de stationsconstante (V) steeds het eerst moet worden opgenomen en dat deze automatisch door het systeem wordt ingevuld. Men hoeft dus niet op te geven vanwaar men vertrekt.

Voor het verdere probleem komen in de informatieverwerking drie typen oplossing voor. Welke oplossing in welk geval wordt toegepast staat uitvoerig in de handleidingen beschreven. Van belang is hier de werking van deze oplossingen toe te lichten.

De oplossings-typen berusten op het maken van e  nduidige afspraken. Deze zijn:

- 1e type: afspraak over volgorde.

Men kan het systeem zo inrichten dat het reageert op in volgorde opgegeven specificaties, ook al is deze niet compleet. Slaat men een specificatie over, dan moet wel de ruimte gereserveerd worden. We nemen dus aan dat de stationsconstante (het eerste element in het argument) automatisch wordt ingevuld. We kunnen dan als GEVRAAGD-functie hebben:

B(Utrecht)	= niet volledige opgave, fout
B(Utrecht,dagretour)	= volledig ingevuld, variabelen zijn gedefinieerd, het systeem (de loketbediende) vult de default waarden in
B(Utrecht,dagretour,,,met)	= in plaats van de defaultwaarde D=geen(toeslag) krijgt de 6de dummy variabele de waarde D=met

- 2e type: definiëren van parameters.

Meer gebruikt wordt het systeem waarbij van parameters en variabelen zowel de symbolische naam als de specificatie in het argument worden genoemd. Veelal worden variabelen zoals in het vorige geval in een afgesproken volgorde genoemd; de parameters kennen dan deze restrictie niet. Voorbeelden van de GEVRAAGD-functie:

B(Utrecht,dagretour)

B(Utrecht,enkele reis,K=1)

B(Utrecht,dagretour,D=met)

Deze wijze van definiëren heeft het voordeel dat gemakkelijker onthouden kan worden wat men moet en wat men mag invullen.

Zijn de vraagmogelijkheden beperkt en eenduidig, dan kan nog een derde type onderscheiden worden:

- 3e type: eenduidigheid van vraag.

Dit laatste geval is erop gebaseerd dat geen verwarring kan ontstaan welke waarde een parameter krijgt wanneer de vragen onderling niet uitwisselbaar zijn. Dan mag de GEVRAAGD-functie luiden:

B(Utrecht,dagretour,met,1)

Het systeem zelf kan nu uitzoeken dat bedoeld is D=met en K=1

Elementen in een argument mogen ook uitdrukkingen of expressies zijn. Zo zouden we kunnen bestellen

B(Utrecht+50km,dagretour,heden+7dagen,na 15de geen)

Het laatste argument is een zogenaamde logische variabele, het bevat een keuze-element, namelijk: na de 15de geen toeslag, dus onderzoek of heden + 7 dagen na de 15de van de maand valt, en zo ja lever dan een kaartje zonder D-trein toeslag en zo nee, dan één met.

Dit alledaags voorbeeld kan nu dienen om voorbeelden uit de informatieverwerking begrijpelijk te maken. Zonder daar diep op in te gaan op dit moment volgt hier een enkele aanwijzing.

ad 1)

Deze oplossing is toegepast bij het veranderen van namen van files bij het uitvoeren van programma's met een zogenaamd Load-and-

Go (LGO) instructie. Voorbeelden:

LGO (geen hernoeming)(default procedure)
LGO,MYJOB. (de eerst voorkomende file wordt hernoemd)
LGO,,,SAVE. (de derde file wordt hernoemd)

De namen MYJOB en SAVE worden door de gebruiker gegeven. Niet hernoemde files waarmee LGO correspondeert worden met komma-aanduidingen afgeteld.

ad 2)

Deze oplossing wordt veel toegepast, onder andere bij de instructie waarmee een Fortran-programma wordt vertaald. Deze luidt:

FTN.

Enkele default waarden welke door het systeem worden gegenereerd wanneer deze in het argument ontbreken, zijn:

I = input file, aangenomen wordt dat deze INPUT heet,
L = listing, aangenomen wordt dat deze gewenst is,
B = binaire object code, aangenomen wordt dat deze naar de file
LGO geschreven moet worden,
R = reference map (administratief overzicht), aangenomen wordt dat
een beknopt overzicht gewenst wordt.

Men kan echter het commando geven

FTN,I=WIT,B=ZWART,L=0.

waarmee andere file namen worden geïntroduceerd en aangegeven wordt dat men geen listing (en geen map) wenst.

Aangezien de parameters I,B en L expliciet genoemd worden is ook correct

FTN,L=0,I=WIT,B=ZWART.

ad 3)

Deze oplossing is veel toegepast in het edit systeem SUEDI. Men kan het commando geven

SAVE,MYJOB,0,N

wat betekent dat de ge-editte file bewaard moet worden als file met gekozen naam MYJOB (moet als eerste genoemd) en dat een eventueel reeds bestaande file van die naam mag worden overschreven (O) en dat geen regelnummering behoeft te worden bewaard (N). Aangezien slechts een beperkt aantal antwoorden mogelijk is mag ook:

SAVE,MYJOB,N,O

Het systeem kan zelf terugvinden bij welke vragen de gegeven antwoorden passen, de filenaam moet echter het eerst genoemd.

Het belangrijkste uit het voorgaande is wel dat blijkt dat de informatie-verwerking berust op duidelijke, voor slechts één uitleg vatbare, afspraken. Dit houdt tevens in dat het systeem zelf kan ontdekken of er fouten tegen de afspraken worden gemaakt. Hiermee wordt de gebruiker al voor een groot deel tegen een verkeerd gebruik beschermd.

I.5. AFSpraak OVER NAAMGEVING

De gebruiker is voor een groot deel vrij in het kiezen van namen voor de bestanden (files) waarop hij zijn informatie wil plaatsen. In de komende voorbeelden wordt deze vrijheid van keuze aangeduid door het schrijven van de filenaam in kleine letters. Vaak wordt echter een handig gebleken naam gebruikt, bijvoorbeeld voor het koppelen van de terminal aan een local file:

CONNECT,icw

De restricties voor file namen zijn: eerste teken moet een letter zijn waaraan ten hoogste 5 alfanumerieke tekens mogen worden toegevoegd. Voorbeelden:

A, Q, W1, WTWEE, MY1ST, BOTECH, KLEI, etc.

Korte namen zijn handiger in verband met het intikken over de terminal. Geadviseerd wordt in ieder geval uitspreekbare namen te gebruiken en geen 'coderingen'. Dus niet XPVN6B maar IKSPEV, niet AIQXI7 maar HULPJE, etc.

I.6. VOORBEELD VAN EEN KLEIN PROGRAMMA

Om reeds in dit stadium een indruk te geven wat de structuur is van een compleet programma, volgen twee eenvoudige voorbeelden.

Voorbeeld 1 (zie pag. 19)

Voorbeeld 2 (zie pag. 21)

Vooruitlopend op een gedetailleerde bespreking kunnen we reeds de volgende eigenschappen van een FROTRAN programma opsommen.

- Elk programma begint met een PROGRAM-kaart
- De statements worden regel na regel in volgorde afgewerkt tenzij besturing door middel van sprongopdrachten plaats vindt
- Berekeningen zijn van het type $y=---(expressie)---$, met slechts één variabele in het linkerlid,
- Variabelen die in het rechterlid van een uitdrukking voorkomen moeten vooraf reeds een getalwaarde hebben gekregen
- Het '='-teken heeft de betekenis van 'wordt'
- Met de symbolen A, B en X worden in feite geheugenplaatsen in de computer aangeduid
- Elk programma eindigt met een END op de laatste regel. Dit hoeft overigens niet het punt te zijn waar de uit te voeren bewerkingen ophouden, dat is bij STOP, (wanneer dat niet samenvalt met END)
- Een FORMAT is geen eigenlijke opdracht (mag overal geplaatst worden)
- Een regel die met C (van comment) in de eerste kolom begint bevat commentaar en wordt door de vertaler genegeerd. In de voorbeelden is de C-kaart gebruikt om de tekst van het programma beter leesbaar te maken door blanke regels tussen te voegen.

Variabelen kunnen op vijf manieren een waarde krijgen en wel:

- 1) Variabelen kunnen in het programma met een statement worden gedefinieerd (A en B in de voorbeelden)
- 2) Variabelen kunnen in een statement numeriek worden berekend (X in de voorbeelden)

Voorbeeld 1. Een programma dat $x = a+b$ berekent wanneer $a=5$ en $b=6$. Behalve de programmanaam 'poging' en de output-file naam 'output', mogen ook de labelnummers 3 en 10 vrij worden gekozen

C	PROGRAM poging(output,TAPE3=output)
	A=5.
	B=6.
	X=A+B
C	WRITE(3,10) X,A,B
10	FORMAT(1H 3F7.2)
C	END

De output van dit programma bestaat uit 3 floating point variabelen X, A en B in de layout zoals vastgesteld in het FORMAT, namelijk 7 posities per getal, 2 cijfers achter de komma.

1	2		} { positie- of kolomaanduiding
123456789.123456789.12345			
11.00	5.00	6.00	

In het FORMAT geeft 1Hb (blank) aan dat op de volgende regel met printen moet worden begonnen.

- 3) Variabelen kunnen los van het programma worden ingelezen met een READ-opdracht (zie hoofdstuk VIII.3)
- 4) Variabelen kunnen gedefinieerd worden door middel van een DATA-statement (zie hoofdstuk X.3)
- 5) Variabelen in het argument van subroutines kunnen worden gedefinieerd tijdens het aanroepen van de routine.

Welke van deze mogelijkheden men zal toepassen hangt af van de wijze waarop men het programma wil gaan opbouwen en zal gaan gebruiken. Soms, tijdens de ontwikkeling van het programma, kan blijken dat men achteraf beter een andere dan de eerst gekozen mogelijkheid kan nemen. De programmeur is hierin vrij en zal zelf een oplossing moeten kiezen. In grotere programma's komen de genoemde mogelijkheden meestal in een of andere vorm gezamenlijk voor.

I.7. EEN PROGRAMMA UITVOEREN

Wanneer een programma compleet is (zie voorbeelden 1 en 2) wil men het programma aan de host-computer aanbieden. Bij aansluiting op de Cyber in Den Haag gaat dit volgens onderstaande procedure.

- Inloggen in de computer volgens de gebruikelijke procedure (MAASSEN en STOL, 1978)
- Opgeven van de projectcode en het accountnummer (Afd.Wisk.Tech.Med.3:1978)

Daarna de volgende commando's intikken. Let op: dit zijn besturings-commando's die niets met FORTRAN te maken hebben. Met . wordt de plaats van de cursor aangeduid.

CONNECT,icwΔ

COPYBF,icw,progΔ

. PROGRAM mylst(data,output,TAPE2=data,TAPE3=output)

.

. (vanaf . vanaf het beeldscherm of vanaf floppy disk
 . verzenden van het complete programma)

.

. END

Voorbeeld 2. Een programma dat $x=ab$ berekent voor $a=5,10,15,\dots$
en $b=6$ totdat $x > 100$

	PROGRAM weetut(output,TAPE9=output)
C	A=.0
	B=6.
C	
40	A=A+5.
	X=A*B
	WRITE(9,20) X,A,B
20	FORMAT(1H "X=A*B"3F8.2)
C	
	IF(X.GT.100.) STOP
	GO TO 40
	END

De output van dit programma bestaat uit 3 floating point variabelen X,A en B in de layout zoals vastgesteld in het FORMAT. Tevens is een stukje tekst in de output opgenomen.

	1	2	3	{ positie- of kolomaanduiding
	123456789.	123456789.	123456789.)	
X=A*B	30.00	5.00	6.00	
X=A*B	60.00	10.00	6.00	
X=A*B	90.00	15.00	6.00	
X=A*B	120.00	20.00	6.00	

```

%EOF      (afsluiten van file 'prog')
Ex      (afsluiten over te zenden informatie)
COPYBF,icw,dataΔ
. 12.34  87.65  24.68 ---
.
. (vanaf . vanaf het beeldscherm of vanaf floppy disk
  overzenden van de gegevens)
.
. 43.21  56.78  86.42 ---
%EOF
Ex
REWIND,prog,dataΔ
FTN,I=prog,R=0Δ
LGOΔ
REWIND,outputΔ
COPYBF,output,icw/print on line/*Δ

```

In de laatste regel betekent /print on line/* het op het toetsenbord van de terminal inschakelen van de printer, zodat de output op papier bewaard blijft.

In de FTN-kaart zijn de volgende parameters van belang.

- I definieert de input file waarop het programma staat.
Default is I=INPUT. Dit wil zeggen wanneer gekozen was prog=INPUT dat volstaan kan worden met FTN,R=0
- L definieert of men een listing van het programma wil.
Default (in het bovenstaande geval dus; L wordt daar namelijk niet genoemd) is 'ja'. Wil men geen listing, bijvoorbeeld als het programma goed blijkt te werken, dan definiëren we L=0 (tevens is dan R=0)
- R definieert of men een reference map wenst, dit is een overzicht met alle organisatorische en administratieve bijzonderheden over het programma. Deze map is zeer nuttig in de testfase
R=0 men wil geen map
R=1 samenvattend overzicht
R=3 compleet overzicht. Hieraan heeft men in de testfase het meest.

Bevat het programma fouten die de machine zelf kan ontdekken, dan wordt dit op het beeldscherm gemeld onder opgave van het aantal fouten. Soms hebben een aantal fouten dezelfde bron zodat het aantal te verbeteren statements vaak minder is.

Men kan een lijst van de fouten zelf verkrijgen en op welke regel deze in het programma voorkomen met het commando

ERRORS,FTNA

Men kan de fouten verbeteren door over te gaan op de editor SUEDEI (zie ICW-nota 1026). Het kan dan soms noodzakelijk (handig) zijn de regels opnieuw te nummeren. Door in SUEDEI te geven R,1,1 verkrijgt men een nummering die overeenkomt met die in de ERRORS-list. Er is dan echter geen ruimte om regels tussen te voegen. Beter is dan R,10,10 of alleen R (default-waarden zijn namelijk 10,10). De regelnummers uit de ERRORS-list moeten dan met 10 worden vermenigvuldigd voor het verkrijgen van correspondentie met de SUEDEI-nummering.

Alvorens het verbeterde programma opnieuw te vertalen moet men geven

REWIND,output,prog,dataΔ

anders komt de nieuwe output achter de oude die men juist kwijt wil.

Heeft het programma al enige resultaten gegeven, maar heeft men het verder verbeterd of gewijzigd, dan moet alvorens het opnieuw te vertalen en uit te voeren eerst ook nog gegeven worden

RETURN,LGOΔ

De RETURN verwijdert het oude vertaalde programma. Doet men dit niet dan wordt met het commando LGO weer het oude programma geactiveerd.

Zijn de files 'prog' en 'data' met SUEDEI verbeterd, dan is voor die files geen REWIND nodig.

Het is efficient indien er veel of gecompliceerde fouten voorkomen de terminal-zitting te beëindigen en eerst de fouten achter zijn bureau op te sporen. Hiervoor is het commando

LOGOUTA

Hierbij moet men bedenken dat local files (in het voorbeeld dus 'prog' en 'data' verloren gaan. Na opnieuw inloggen moet men deze files opnieuw aanmaken, respectievelijk van floppy disk kopiëren. Ook kan men alvorens uit te loggen de files permanent maken. Zijn de files met SUEDE gesaved dan kan rechtstreeks een catalog instructie volgen. Voorbeelden (bewaren van het programma):

- zonder SUEDE,

REQUEST,p,*PFA

REWIND,progA

COPYBF,prog,pA

CATALOG,p,nieuwprog,ID=it1A

LOGOUTA

- met SUEDE

..SAVE,prog1,NA

..BYEA

COMMAND-CATALOG,prog1,nieuwprog,ID=it1A

Na opnieuw te hebben ingelogd komt de file weer beschikbaar (wordt weer local) met

ATTACH,prog,nieuwprog,ID=it1A

Voor details wordt verwezen naar ICW-nota 1026 (SUEDE) en nota 1052 (Praktijkvoorbeelden).

I.8. GETALSTELSELS

De informatieverwerking berust uiteindelijk op toepassingen van de binaire code, of wel het tweetallig stelsel. De gebruiker van Fortran merkt hier echter niets van, aangezien alle coderingen en decoderingen automatisch verlopen. Bovendien is de toepassing van

de binaire code dusdanig dat de gebruiker in een enkel geval alleen iets te maken krijgt met een octale code. Deze is gebaseerd op het $8 (= 2^3)$ -tallig stelsel. Voorbeelden van getallen en hun onderlinge samenhang in het 2-, 8- en 10-tallig stelsel worden gegeven in tabel 1.

Tabel 1. Getallen van 0-16 (decimaal) in het 2-, 8- en 10-tallig stelsel

Binaire 2-tallig	Octaal 8-tallig	Decimaal 10-tallig
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	10	8
1001	11	9
1010	12	10
1011	13	11
1100	14	12
1101	15	13
1110	16	14
1111	17	15
10000	20	16
.	.	.
.	.	.
.	.	.
	77	63
	100	64
	.	.
	.	.
	.	.

De octale code wordt gebruikt in het STOP n-statement (zie hoofdstuk IX.2). Voor verder gebruik van de octale code wordt verwezen naar ICW-nota 1070 (Batchverwerking).

I.9. TE BEHANDELEN STATEMENTS (SYSTEMATISCH)

De te behandelen Fortran-statements zullen eerst in een systematisch overzicht opgesomd worden. De statements worden gegeven zonder argument en zijn gerangschikt naar functioneel gebruik.

Ingangen

PROGRAM
SUBROUTINE
FUNCTION

Type declaraties

IMPLICIT
REAL
INTEGER
LOGICAL

Geheugen indeling, data definitie

DIMENSION
COMMON
DATA

Functie-definitie

$f(x_1, \dots, x_n) = \text{functie}$

Berekeningen

$y = \text{expressie}$

Besturing, intern

CONTINUE
GO TO
IF
DO
STOP

Besturing, extern

CALL
RETURN

Input en output

FORMAT
READ
WRITE

File organisatie

REWIND
CALL CONNEC
CALL DISCON

Afsluiten programma tekst

END

I.10. TE BEHANDELEN STATEMENTS (LOGISCH)

De te behandelen statements kunnen ook in een logische volgorde worden geplaatst, namelijk in de volgorde waarin ze in een programma moeten (resp. kunnen) voorkomen. Een aantal statements, namelijk die welke de organisatie en administratie verzorgen, moeten op een vaste plaats in het programma worden opgenomen.

In tabel 2 staat in schema-vorm aangegeven hoe de verschillende statements in een programma kunnen voorkomen. De eventuele argumenten zijn weer weggelaten. De met een streep (/) gescheiden onderdelen zijn onderling vervangbaar in het schema, maar op de genoemde

plaatsen op dezelfde wijze.

In het vak 'statements', kunnen voorkomen de volgende opdrachten in elke gewenste volgorde en in elk gewenst aantal:

CALL CONNED	CONTINUE
CALL DISCON	GO TO
REWIND	IF
READ	DO
WRITE	
y = expressie	
CALL	

Tabel 2. Schematisch overzicht over de opbouw van een Fortran hoofdprogramma en subprogramma's voor zover het de hier behandelde onderdelen betreft.

COMMENTS	PROGRAM / SUBROUTINE / FUNCTION			
	FORMAT	IMPLICIT		
		INTEGER } REAL } (type declaraties) LOGICAL }		
		DIMENSION (geheugen bezetting)		
		COMMON (geheugen indeling)		
		$f(x_1, \dots, x_n) = \text{functie}$		
		DATA		
		statements	STOP / RETURN / RETURN STOP / STOP	
		END		

De in kaders opgegeven onderdelen zijn minimaal vereist.

II

HET PROGRAMMEREN

II.1. MODULAIR EN GESTRUCTUREERD PROGRAMMEREN

De laatste jaren zijn veel discussies gevoerd over het zogenaamd modulair en gestructureerd programmeren. Met deze termen wordt aangegeven dat een programma opgebouwd moet zijn uit duidelijk te onderscheiden modules, en dat het programma in zijn geheel en zekere en liefst eenvoudige structuur moet bezitten (VAN AMSTEL, 1976).

Programma's voor wetenschappelijk rekenwerk bevatten deze eigenschappen reeds enigszins van nature. Namelijk: er wordt veelal naar één resultaat of naar een groep samenhangende resultaten toegewerkt. Toch moet het volgend onder ogen worden gezien.

- Programma's zijn nooit definitief. Op geregelde tijden moeten ze worden aangepast aan nieuwe wensen en nieuwe ontwikkelingen: men moet dan weer thuis raken in zijn eigen programma.
- Na enkele maanden is het voor de schrijver zélf zelfs moeilijk alle finesses van een programma weer snel te doorgronden.
- Gebruikers van een programma willen vaak zelf een door een ander geschreven programma naar eigen behoefte wijzigen en aanpassen (zie echter weer de vorige opmerking).

Hoewel Fortran niet een taal is waarmee naar de moderne inzichten gestructureerd kan worden geprogrammeerd, valt toch het nodige te bereiken door aan de volgende punten aandacht te schenken.

- In een programma dient voldoende beschrijvende tekst te worden opgenomen om de geïntroduceerde variabelen duidelijk te verklaren en de werking van het programma toe te lichten.
- Een programma moet in duidelijk gescheiden eenheden worden ingedeeld bijvoorbeeld: initialiseren van besturingsparameters, inlezen van gegevens, transformatie van gegevens, de eigenlijke bewerkingen, uitvoer van resultaten, afsluiten van de bewerkingen, terugspringen naar het begin van het programma.
- Bewerkingen die vaak terugkeren en een eenheid vormen, kunnen opgenomen worden in een afzonderlijk te schrijven subprogramma.

Voordelen van deze laatste werkwijze zijn nog:

- Subprogramma's kunnen los van het hoofdprogramma geschreven en ge-

test worden.

- Namen van variabelen en nummers van labels in subprogramma's zijn onafhankelijk van die van het hoofdprogramma waardoor men vrijer is in de keuze ervan.

In fig. 1 wordt schematisch weergegeven hoe een hoofd- en een subprogramma onderling samenhangen. Correspondentie tussen variabelen kan worden verkregen door middel van het gemeenschappelijk gebruik van een bepaald geheugen gedeelte (COMMON) of door middel van het argument van de CALL-opdracht. (Vergelijk hiervoor de inhoud van hoofdstuk I.4). De programmeur is vrij om hiervoor zelf een werkwijze te ontwikkelen.

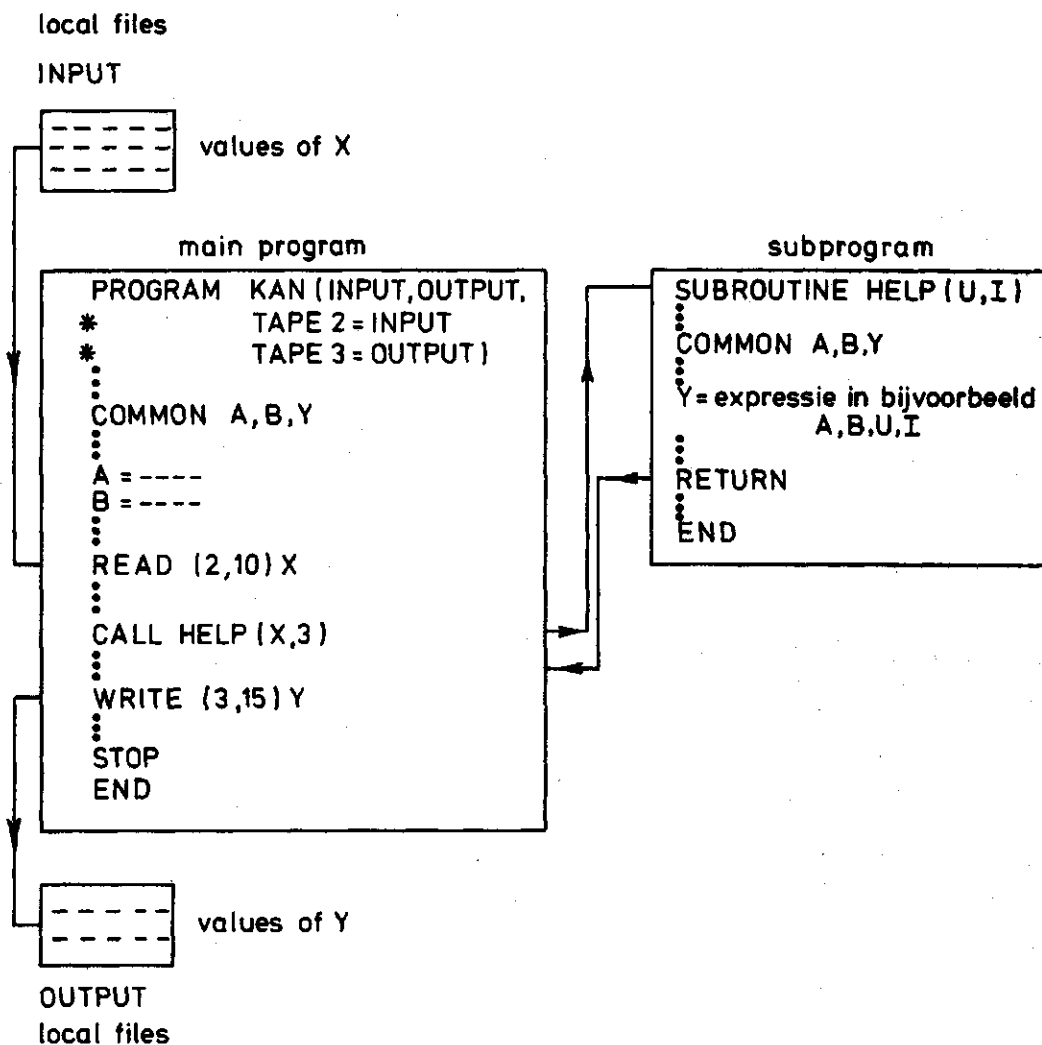


Fig. 1. Schematisch overzicht over de koppeling tussen een hoofd- en een subprogramma

III

ALGEMENE AFSPRAKEN

III.1. TEKENS GEBRUIKT IN FORTRAN

De zogenaamde characterset voor Fortran-programma's ziet er als volgt uit.

A - Z	alfabetische tekens (hoofdletters)	
0 - 9	numerieke tekens	
=	is gelijk aan	} bijzondere tekens
+	plus	
-	min	
*	maal (asterisk)	
/	deelstreep (slash)	
()	haakjes	
,	komma (niet in getallen)	
.	punt (decimal point)	
\$	dollar teken (scheidingsteken)	
	blank (spatie)	

A - Z - 0 - 9 afanumerieke tekens

In tekstgedeelten (beschrijvingen in het programma, opschriften van tabellen in de output) mogen alle op de terminal of ponsmachine voorkomende tekens worden gebruikt.

Er bestaat een afspraak om in schrijfwijze een duidelijk onderscheid te maken tussen 0 (nul) en de O (oh). De afspraak is een slash door een van beide te schrijven. Aangezien de letter O minder vaak voorkomt in numeriek werk werd vroeger voorgesteld de letter te doorstrepen dus GØ TØ 100. De moderne opvatting is dat het aanbeveling verdient het cijfer te doorstepen. Dit symbool is tevens het gebruikelijke symbool om een lege verzameling aan te duiden ($V = \emptyset$, de verzameling V bevat geen elementen). Dit leidt dan tot GO TO 1ØØ.

Veelal zal er bij het schrijven van programma's niet veel aanleiding tot verwarring zijn zodat in de praktijk het doorstrepen alleen toegepast wordt als men niet zeker weet of de tekst voor derden voldoende duidelijk is.

Evenals in gewone schrijftaal mag in Fortran al naar believen

spaties worden gebruikt wanneer dat voor de duidelijkheid gewenst of vereist is. Bijvoorbeeld worden gelijk geïnterpreteerd

GOTO10 GOTO 10 GO TO 10

De input en output apparatuur heeft geen voorziening om indices en machten op de gebruikelijke wijze te schrijven. Alles moet op regelhoogte worden genoteerd zodat we krijgen

A**2 voor a^2
X(1),X(2),X(N) voor x_1, x_2, x_n
Z(3)**4 voor z_3^4

Het voorkomen van slechts één type haakjes betekent dat bij het 'nesten' de tekst en de betekenis minder duidelijk is dan in de schrijftaal. Voor de computer is dit geen bezwaar. Een voorbeeld is

$[a + b\{c + d(x + y)\}]^2$ wat wordt $(A + B*(C + D*(X + Y)))**2$

Het vermenigvuldigingsteken is steeds verplicht.

III.2. DE REGEL-INDELING

Om aan te geven hoe een regel ingedeeld wordt zullen we spreken van 'kolommen'. Dit kunnen zijn de kolommen van een ponskaart of op een ponsdocument, maar ook de posities op het beeldscherm van de terminal. Het plaatsen van een teken in een bepaalde positie zullen we in het spraakgebruik aanduiden met: het ponsen van dat teken in een bepaalde kolom, of het intikken van dat teken in een bepaalde positie.

Een regel bestaat uit 80 kolommen. Deze kunnen (moeten) worden gebruikt zoals in fig. 2 staat aangegeven.

	kolom - nummer				
wanneer ge- bruikt voor	1	4 — 5	6	7 — 72	73 — 80
opdrachten (statements)	(label)		blank of nul	FORTTRAN-statement	vrij beschikbaar
vervolgregel	niet gebruikt		ander dan blank of nul	FORTTRAN-statement	vrij beschikbaar
commentaar (comments)	C \$ *	vrij beschikbaar voor commentaar			
gegevens (data)	vrij beschikbaar voor gegevens en commentaar				

Fig. 2. Voorschriften met betrekking tot de indeling van ponskaarten of van ponsdocument, respectievelijk met betrekking tot in te tikken informatie

Een ponskaart kan de informatie van precies één regel van 80 kolommen bevatten. Vandaar dat soms gesproken zal worden over de 'kaart-indeling' in plaats van over de 'regel-indeling'.

Een regel mag meer dan één opdracht bevatten. Deze moeten dan gescheiden worden door een \$-teken. Dit teken wordt dan opgevat weer in de 6e kolom te staan. Voorbeeld:

A = B + C \$ D = E + F

Een \$ mag niet gebruikt worden vóór een END of vóór een genummerde opdracht.

Het is toegestaan meer variabelen aan eenzelfde uitdrukking gelijk te stellen zoals

X = Y = A = B + C

III.3. REGELVERLENGING

Een Fortran-opdracht mag meer dan één regel beslaan. Om aan te geven dat de vorige opdracht afgekort is moet op de volgregel in de 6e kolom een aanduiding worden geponst. Deze aanduiding mag elk teken anders dan een blank of een nul zijn en behoeft niet persé een nummering te zijn. Bijvoorbeeld (met / in de 6e kolom):

```
| A = B + C*(D + E)*  
| /F
```

wat hetzelfde is als

```
| A = B + C*(D + E)*F
```

aangezien de blanks na de laatste * alle worden genegeerd.

Er mogen 19 vervolgregels of vervolgkaarten worden gebruikt. De voorgaande regel behoeft dus niet vol te zijn om een vervolgregel te kunnen toepassen. Gemakshalve kan men vervolgregels aanduiden met een 1, 2, 3 enz. in de 6e kolom. De computer voert echter geen controle uit op deze volgorde.

Vervolgregels worden vrij veel in FORMATS toegepast. Het nadeel van het gebruik van vervolgregels bij volle regels van een FORMAT is dat het dan moeilijk is om veranderingen aan te brengen. Meestal ontbreekt de ruimte hier dan voor. Zou bijvoorbeeld in de eerste regel informatie moeten worden toegevoegd, dan werkt dit door in alle verdere vervolgregels, waardoor het aanbrengen van veranderingen een lastig karwei wordt.

In de praktijk worden vervolgregels veel gebruikt voor het creëren van meer ruimte in lange FORMATS en niet zozeer als overloop voor volle regels. Voorbeeld:

```
20 | FORMAT(1H "A (KAVELGROOTTE) = "F3.0"bbHA"  
| * /1H "B (SLOOTLENGTE ) = "F3.0"bbM"  
| * /1H "C (WEGBREEDTE ) = "F3.0"bbM")
```

Bij rekenkundige opdrachten zijn er andere middelen om de regelbreedte te beperken en de formulering overzichtelijk te houden.

Hier kan een efficiënt gebruik van vervangingsopdrachten worden gemaakt (zie hoofdstuk VI.4).

III.4. BESCHRIJVINGEN EN TOELICHTINGEN

In een programma mag men naar believen toelichtingen, beschrijvingen en gebruiksaanwijzingen opnemen. Om de computer er attent op te maken dat de komende regel geen rekenopdracht maar alleen een regel tekst bevat, een regel 'comment' dus, moet in de eerste kolom een C, of een *, of een \$ geponst worden. Voorbeeld:

```
C      |
C      | NEERSLAG P IN MM/DAG
C      |
C      | B = A*P
C      |
```

Men moet erop attent zijn dat de regel

```
C      | Y = 2. * X + B
```

een comment-regel is en dus niet door de computer wordt uitgevoerd als een rekenopdracht. De regel wordt door de compiler genegeerd.

III.5. GENUMMERDE OPDRACHTEN

De computer werkt een Fortran-programma regel voor regel af van het begin (de PROGRAM-kaart) tot aan de laatste regel (END). Dit gebeurt ongeacht of de regels al of niet van een nummer zijn voorzien. Bijvoorbeeld:

```
      | A = B + C
20    | D = E * F
      | G = H * H
10    | X = Z + A
16    | Y = X + G * D
      | K = 1
```

↓
volgorde van verwerking

Deze opdrachten worden gewoon in de opgeschreven volgorde uitgevoerd. De nummering heeft daarop geen enkele invloed. Het op deze wijze nummeren van de opdrachten of regels wordt daarom nooit toegepast, omdat het voor de computer niet relevant is of de nummering doorlopend en compleet is.

Wil men de kaarten nummeren (om ze na door elkaar raken weer in de goede volgorde te kunnen leggen) dan kan men hiervoor de kolommen 73-80 gebruiken. In de praktijk wordt dit nooit gedaan. Brengt men veranderingen aan in het programma en dus in het kaartendek, dan is de nummering toch weer meteen verstoord.

De nummers 20, 10 en 16 in het bovenstaande voorbeeld worden 'labels' of labelnummers genoemd. Men mag hiervoor gebruiken de getallen 1-99999. Labels worden gebruikt om naar bepaalde regels in het programma te kunnen refereren en daarnaar toe te springen. Dit bijvoorbeeld wanneer een bepaald gedeelte moet worden herhaald of om een of andere reden juist moet worden overgeslagen.

III.6. TOEPASSINGEN

Van de tot nu toe behandelde onderwerpen kunnen toepassingen reeds in de voorbeelden 1 en 2 worden teruggevonden. Verdere voorbeelden worden gegeven in het programma in voorbeeld 3.

Voorbeeld 3. Een programma dat $c^2 = a^2 + b^2$ berekent voor $b=4$ en $1 \leq a \leq 5$. Labelnummers en namen van programma en files mogen vrij worden gekozen.

```

C      PROGRAM a2b2(result,TAPE1=result)
C
C  DE ROUTINE HOOGT A STEEDS MET EEN
C  EENHEID OP. DE STARTWAARDE IS DUS A=0
C
      A=.0    $    B=4.
50    A=A+1. $    IF(A.GT.5.) GO TO 10
      C2=A*A+B*B
      WRITE(1,1) A,B,C2    $    GO TO 50
1    FORMAT( /1H "A="F 8.3,3X,
*           "B="F 8.3,3X,
*           "C KWADRAAT="F 8.3)
C
10    STOP
      END

```

De output van dit programma is nu als volgt (vooraf komt steeds een regel spatie); output staat op file 'result':

	1	2	3	4
	123456789.	123456789.	123456789.	123456789.12345
A=	1.000	B= 4.000	C KWADRAAT=	17.000
A=	2.000	B= 4.000	C KWADRAAT=	20.000
A=	3.000	B= 4.000	C KWADRAAT=	25.000
A=	4.000	B= 4.000	C KWADRAAT=	32.000
A=	5.000	B= 4.000	C KWADRAAT=	41.000

IV

G R O O T H E D E N

IV.1. INLEIDING

In Fortran worden verschillende soorten grootheden gedefinieerd, of zoals we dat zullen noemen grootheden van verschillend type. Elk type heeft zijn eigen mogelijkheden en beperkingen. Van elke grootheid die zal worden gebruikt moet het type aangegeven worden opdat de computer weet hoe hij met die grootheid moet gaan rekenen. Er gelden echter enkele automatismen (default regelingen) waarvan in de voorgaande voorbeelden ook reeds gebruik gemaakt is. In het nu volgende zal besproken worden hoe grootheden worden ingedeeld en op welke wijze het type waartoe ze zullen gaan behoren aan de computer bekend wordt gemaakt.

Een eerste indeling van grootheden is naar gebruik. Men onderscheidt constanten en variabelen.

Een tweede indeling van grootheden is naar type. We zullen hier de volgende typen behandelen:

Tekst	(character string)
Getallen	Integer (b.v. 5) Real (b.v. 6.75)
Logical	(5 < 10 is wāār)

We geven nog een korte toelichting alvorens meer in detail op elk in te gaan.

Constante-

De grootheid wordt met zijn getalswaarde aangeduid.

Variabele-

De grootheid wordt, zoals in de algebra, met een symbool ook wel genoemd de 'naam', of de 'symbolische naam', aangeduid.

Integer-

Een grootheid die alleen gehele waarden aanneemt zoals -100, -5, 0, 1, 519, 234567.

Real-

Een grootheid die het gehele scala van reële waarden kan aanne-

men. Populair gezegd: positieve en negatieve getallen met een komma, zoals -23.691, -0.00012, 54321.12345, 1.0, 15.15.

Logical-

Een grootheid die alleen de waarden TRUE (wáár) of FALSE (niét-waar) kan aannemen.

Aangezien elk van deze typen zijn eigen specifieke rekenregels heeft moet aan de computer opgegeven worden tot welk type elke grootheid behoort. De programmeur moet dit dus zelf opgeven omdat hij de bedoeling kent waarvoor de grootheden in zijn programma zijn geïntroduceerd.

IV.2. CONSTANTEN

Constanten worden met hun getalswaarde opgegeven. Ze kunnen alleen in het rechterlid van een uitdrukking voorkomen. Voorbeelden (integers):

1	A**3	(hier is 3 de constante)
3		
100	INDEX=5	
-16		
0		

Real constanten worden met een decimal point opgegeven. Voorbeelden (real):

1.	PI=3.14	
3.		
0.20	2.*X**2	(2x ²)
.0 of 0.		
-132.231		

IV.3. VARIABELEN

Wanneer men variabelen met slechts één letter zou mogen aanduiden, zoals in de algebra, zou in een groot programma de voorraad aan mogelijkheden snel uitgeput zijn. In FORTRAN mogen 'namen' gebruikt worden van 1 tot en met 7 alfanumerieke tekens lang, beginnend met een letter.

In een naam van een variabele mogen blanks voorkomen maar gebruikelijk is het opnemen van blanks niet. Voor de computer zijn blanks geen relevante tekens, ze worden genegeerd en tellen niet mee.

In de naamgeving van variabelen kan automatisch per default afspraak het type gedefinieerd worden. We zullen dit de type-afspraken noemen.

Integer variables-

Een variabele wordt opgevat te zijn van het integer type, wanneer de naam begint met één van de letters I, J, K, L, M of N (geheugensteun: INteger).

Real variables-

Een variabele wordt opgevat te zijn van het real type wanneer de naam begint met een andere letter dan die welke voor integers gereserveerd zijn en wel: A-H, O-Z.

Voorbeelden van namen van variabelen in Fortran:

Integer	Real
I	A
II	Al
J	Z
IJ	REGEN
MAX	GROOTST
MAXIMUM	ALPHA
ICW	STIBOKA
II23456	X123Y12
IK EERST	WY LATER
KAVEL	EVAPOR

Het automatisme in de type-aanduiding kan men doorbreken door expliciet het type te definiëren.

De opdrachten hiervoor zijn:

```
INTEGER WIJ, EERSTE, FACTOR, M  
REAL KAVEL, LAATSTE, INDEX, F
```

Het opnemen van de variabelen M en F was in feite niet noodzakelijk maar is niet verboden.

Logische variabelen moeten altijd gedeclareerd worden. Dit kan met:

```
LOGICAL WAAR, GADOOR, ZI
```

Het is mogelijk alle variabelen waarvan de naam met dezelfde letter begint tot eenzelfde type te declareren. De opdracht hiertoe is IMPLICIT. Ook kan een reeks beginletters vermeld worden. Voorbeelden

```
IMPLICIT INTEGER (A-Z)
```

Alle variabele namen zijn van het type integer.

```
IMPLICIT INTEGER (A-E,G), REAL (L, M, N)
```

Alle variabele namen die beginnen met de letters A tot en met E en de letter G zullen van het type integer zijn; alle variabele namen beginnend met de letter L of M of N zijn reals. Uitzonderingen kunnen afzonderlijk gedeclareerd worden bijvoorbeeld in aansluiting op het eerste voorbeeld

```
REAL X, Y
```

De declaratie geldt hier alleen voor de variabele naam X en Y. De variabele XVAR bijvoorbeeld is integer.

Opmerking-

De afspraken omtrent de naamgeving gelden voor meer onderdelen in Fortran. Steeds geldt dat een naam een letter is gevolgd door

0-6 alfanumerieke tekens. De type-afspraken geldt niet (is niet relevant) voor file-namen, en namen voor hoofdprogramma's en subroutine subprogramma's. Zie ook ICW-Nota 1037 (STOL, 1978).

IV.4. INTEGER EN REAL VARIABELEN IN FORMULES

In het 'dagelijks gebruik' wordt meestal geen rekening gehouden met het feit dat getallen tot verschillende typen gerekend moeten worden. Een bekend voorbeeld is het berekenen van een gemiddelde waarde, bijvoorbeeld van

13.26 12.31 15.68

som = 41.25 aantal = 3

gemiddelde = $\frac{41.25}{3} = 13.75$

wat nu geschreven is als een 'mixed operation': het delen van een real door een integer.

In Fortran is het noodzakelijk met dit verschil in type rekening te houden. Integer bewerkingen verlopen dan ook anders dan real bewerkingen. Bijvoorbeeld:

I=10 en J=8 geeft I/J=1 en J/I=0
A=10. en B=8. geeft A/B=1.25 en B/A=0.8

Bij het toekennen van waarden aan variabelen is het type van de grootte in het linkerlid doorslaggevend. Dit hulpmiddel wordt dan ook veel gebruikt om van het ene type op het andere over te gaan. In de naamgeving gebruikt men hiervoor in de praktijk vaak een I voor overgang naar integers en een F voor overgang naar reals. In het vorige voorbeeld dus:

SOM = 41.25

N = 3

FN = N

GEM = SOM/FN

Denk om de volgende complicatie:

VEEL = 321.123

IVEEL = VEEL

waarin nu IVEEL = 321 zonder decimaal gedeelte.

Schrijft men daarna

VEEL = IVEEL

dan heeft VEEL de waarde 321.000... Getallen worden op deze wijze afgekapt en niet afgerond. Voorbeelden:

I=10.	wordt verwerkt als	I=10
A=10	" " "	A=10.
N=3		
FN=N	" " "	FN=3.
I=J=A=B=3	" " "	B=3.
		A=3.
		J=3
		I=3
P=I=A=3.25	" " "	A=3.25
		I=3
		P=3.

Van de eigenschappen van integer operaties wordt onder andere gebruik gemaakt om na te gaan of een getal exact een n-voud is. In dat geval is met integers $g-(g/n)*n=0$. Zie ook voorbeeld 4.

In het rechterlid van een uitdrukking kan men beter geen integer en real variabelen door elkaar gebruiken. De enige uitzondering die geen complicaties oplevert is de exponent. Bijvoorbeeld geoorloofd is

A = B**2

wat bovendien sneller werkt als

A = B**2.

Deze laatste uitdrukking wordt verwerkt met een routine van het type

$$a = e^{2 \ln b}$$

en behoeft dus twee hulpfuncties, namelijk de exponentiële en de natuurlijke logaritme, die overigens automatisch opgeroepen worden maar voor de verwerking wel extra tijd vragen.

Wanneer het noodzakelijk is van het ene type variabele op het andere over te gaan dan kan men hiervoor gebruik maken van de bijzondere functies IFIX en FLOAT uit tabel 4, pag. 73. Het is dan niet noodzakelijk eerst op hulpvariabelen over te gaan. Voorbeelden:

D=2.

A=B**IFIX(D)

FMEAN=XSUM/FLOAT(N)

IV.5. EXPONENTIËLE VERSIE/FLOATING POINT VERSIE

Getallen, in het bijzonder uitkomsten van berekeningen, kunnen sterk in grootte verschillen. Ten einde de aandacht sterker te richten op het significante deel van een getal kan men getallen in de exponentiële versie brengen. Dit is ook van voordeel bij het maken van tabellen waarbij getallen van wisselende grootte in een vast aantal posities kan worden genoteerd.

In de exponentiële versie wordt de plaats van de komma in het getal bepaald door het eerste significante (van nul afwijkende) cijfer. De orde van grootte wordt dan weergegeven door een exponent: een macht van tien.

In de floating point versie wordt de plaats van de komma in het getal niet bepaald door de cijferreeks als zodanig maar hangt af van de orde van grootte van het getal.

De opbouw is dan

Floating point	12345. 4321
	integer deel . decimaal gedeelte
Exponentieel	.123454321E+05
	mantisse exponent

De uitdrukking E=05 betekent dat het voorgaande getal met 10^5 vermenigvuldigd moet worden. Voorbeelden zijn:

Floating point		Exponentieel
1.15	=	.115E+01
-321.65	=	-.322E+03
0.007898	=	.790E-02
-789.123456	=	-.789123456E+03
1.	=	.100000E+01
0.	=	.456543E+00
0.00	=	.6543E-08
0.01	=	.6543E-02

Hierbij moet worden opgemerkt dat de computer met meer significante cijfers werkt dan hier staat aangegeven. Bij het uitschrijven van getallen en resultaten kan de programmeur opgeven hoeveel posities het getal mag innemen en hoeveel cijfers achter de komma moeten worden vermeld. De computer voert dan een afronding uit. Dit verklaart tevens waarom er in de laatste drie voorbeelden de floating point weergave zo sterk afwijkend schijnt van de exponentiële weergave. Merk op dat 0.006543 na afronding inderdaad geeft 0.01.

Exponential reals kunnen op de volgende wijze op ponskaart worden geponst of in het programma worden opgenomen:

3.E1	wat betekent	30.
30E02	" "	3000.
31.E-1	" "	3.1

IV.6. SAMENVATTING, BEPERKINGEN

Het vorige kan schematisch als volgt worden samengevat:

	INTEGER	REAL	
		Floating	Exponentieel
Constante	-100	1.2	.012E+02
	0	-3.14	-.314E+01
	30	0.06	.619E-01
		15.26	.153E+02
Variabele	I	X	
	LAATSTE	FIRST	
	M	DRIE	

De beperkingen zijn de volgende:

Integer: 15 cijfers, maximum grootte $2^{48}-1$

Real : 15 decimalen, exponent tussen -308 en +337
grootte tussen 10^{-293} en 10^{322}

Uit de beperkingen volgt tevens dat exact nul bij reals niet voor kan komen. Dit zou zijn 0.0000... enz. Het aantal nullen blijft in de computer tot een eindig aantal beperkt dat in de praktijk echter veelal groot genoeg zal blijken te zijn om in afronding in de output 0. op te leveren.

IV.7. LOGISCHE GROOTHEDEN

Logische grootheden nemen alleen de waarde 'wáár' of 'niét-waar' aan. Bijvoorbeeld, stel dat gegeven is

x = 3

y = 5

dan is de logische variabele v_1 : $x = y$ niet-waar, en de logische variabele v_2 : $x \neq y$ waar. In Fortran is dit

```
LOGICAL V1, V2
```

```
X = 3.
```

```
Y = 5.
```

```
V1 = X.EQ.Y
```

```
V2 = X.NE.Y
```

dan is V1 niet-waar en V2 is waar.

In Fortran worden logische uitdrukkingen geschreven met punten vóór en achter de bewerkingssymbolen. De volgende bijzonderheden worden hier aangestipt.

L o g i s c h e c o n s t a n t e n

Logische constanten worden als volgt gedefinieerd:

```
V = .TRUE.           (numerieke waarde -1)
```

```
V = .FALSE.          ( " " " 0)
```

L o g i s c h e o p e r a t o r e n

```
.NOT.
```

```
.AND.
```

```
.OR.
```

Voorbeelden:

```
V1 = .NOT.L1      niet-waar, alleen als L1 waar is.
```

```
V1 = L1.AND.L2    waar, alleen als L1 en L2 beide waar zijn.
```

```
V1 = L1.OR.L2     waar, alleen als òf L1, òf L2, of beide waar zijn.
```

Een bijzonder geval is nog de ontkenning van .OR. en .AND. Hierbij verandert .OR. in .AND. en omgekeerd. Namelijk:

```
indien   B.LT.5. .OR. B.GT.10.   waar is,
```

```
dan is   B.GE.5 .AND. B.LE.10.   niet-waar.
```

in schema:



Merk op dat in het laatste geval (met .OR.)

B.GE.5 .OR. B.LE.10.

altijd wããr is en dus niet de ontkenning van de eerste uitspraak kan zijn.

IV.8. HOLLERITH CONSTANTEN

Een bijzondere soort grootheid is tenslotte de Hollerith constante. Met de naam Hollerith wordt aangeduid dat we te maken hebben met in het geheugen van de computer opgeslagen tekst en niet met getallen. In de tekst mogen alle beschikbare tekens voorkomen waarbij blanks als afzonderlijke tekens worden beschouwd en meegeteld worden.

Hollerith constanten behoeven niet als type gedeclareerd te worden zoals integer en real: ze kunnen in het programma op elke plaats waar ze gewenst zijn, gedefinieerd worden. Om aan te geven dat tekst bedoeld wordt, wordt deze voorafgegaan door de code nH waarin $1 \leq n \leq 10$ het aantal Hollerith tekens aangeeft en H de Hollerith indicatie is. Voorbeelden:

```
K=4HTAPE
TEKST=10HDITbISbEEN
NUL=5HNUL           (2 blanks achter NUL)
IB=1H               (1 blank)
STER=1H*
```

Er wordt nog onderscheid gemaakt tussen in de beschikbare ruimte links (L) of juist rechts (R) opgeschoven tekst (left or right justified). Voorbeeld:

```
5LNUL   wordt   NULbb
5RNUL   "       bbNUL
```

De H-versie zelf is left justified.

Let nog op het verschil tussen

I=115 (getal waarmee gerekend kan worden)

I=3H115 (stuk tekst van drie characters)

Hollerith constanten worden slechts in beperkte mate gebruikt. Ze kunnen dienen om bij het plotten van figuren op de lijnprinter symbolen te definiëren waarmee verschillende curven zullen worden geplot. Bijvoorbeeld:

```
IS=IH*
```

```
IX=IHX
```

```
IP=IH.
```

Een ander gebruik is om onderscheid tussen uitkomsten te maken die verbaal direct aanspreken. Bijvoorbeeld:

```
HOE=4HGOED $ IF(K.GT.100) HOE=4HFOUT
```

zodat bij het schrijven van de inhoud van HOE naar de output file direct te lezen valt GOED of FOUT. Merk op dat HOE hier feitelijk een variabele is.

IV.9. TOEPASSINGEN

Het voorgaande is toegepast in voorbeeld 4. In dit programma wordt gevraagd te signaleren of het aantal recreanten dat op een even dag een object bezoekt groter is dan 100 en of het aantal dat dat object op een oneven dag bezoekt groter is dan 150. Tevens wordt de verhouding tussen de aantallen bezoekers volgens even dagen/oneven dagen gevraagd.

In voorbeeld 5 wordt hetzelfde programma gegeven, nu inclusief de benodigde stuuropdrachten. Het programma zowel als de stuuropdrachten zijn in hun geheel als één file naar de Cyber overgezonden. De Cyber komt wél terug met de benodigde COMMAND-'s, waarop men in dit geval niet meer hoeft te reageren.

Voorbeeld 4. Het signaleren of het aantal bezoekers (B) aan een recreatie-object op een even dag (D) groter is dan 100 en op een on-even dag groter is dan 150. Neem aan dat het aantal bezoekers op dag: D1 = 15e gelijk is aan B1 = 110 en op D2 = 16e gelijk aan B2 = 130

	PROGRAM MYTEST(OUTPUT,TAPE1=OUTPUT)
	INTEGER D1,B1,D2,B2
	LOGICAL EVEN
C	
C	BEPAAAL OF B.GT.100 ALS D EVEN
C	" " B.GT.150 " D ONEVEN
	D1=15 \$ B1=110 \$ D2=16 \$ B2=130
C	
	NDAG=D1-(D1/2)*2
	EVEN=NDAG.EQ.0
	NTEST=150 \$ IF(EVEN) NTEST=100
	TEST1=2HNO \$ IF(B1.GT.NTEST)TEST1=3HYES
C	
	NDAG=D2-(D2/2)*2
	EVEN=NDAG.EQ.0
	NTEST=150 \$ IF(EVEN) NTEST=100
	TEST2=2HNO \$ IF(B2.GT.NTEST)TEST2=3HYES
C	
	CALL CONNEC(6LOUTPUT)
	WRITE(1,5) D1,B1,TEST1,D2,B2,TEST2
5	FORMAT(/1H " D1 B1 TEST1"
	*/1H 2I7,3X,A3//
	*/1H " D2 B2 TEST2"
	*/1H 2I7,3X,A3)
C	
	RATIO=FLOAT(B2)/FLOAT(B1)
	WRITE(1,6) RATIO
6	FORMAT(/1H "RATIO EVEN/ONEVEN" E15.4)
C	
	END

De output van dit programma is nu als volgt:

1	2	3
123456789.123456789.123456789.		
D1	B1	TEST1
15	110	NO
D2	B2	TEST2
16	130	YES
RATIO EVEN/ONEVEN		.1182E+01

Door gebruik te maken van de routine CONNEC wordt de terminal met de file, genoemd in het argument en dus hier, OUTPUT verbonden. De output van dit programma komt meteen op het beeldscherm. Merk op dat de output in F-versie right-justified is, de output in A-versie left-justified.

Voorbeeld 5. Als voorbeeld 4, nu compleet met stuuroopdrachten. Het gedeelte tot en met LGO is in zijn geheel vanaf het beeldscherm overgezonden.

```
RETURN,LGO,OUTPUT,T
CONNECT,ICW
COPYBF,ICW,T
```

```

PROGRAM MYTEST(OUTPUT,TAPE1=OUTPUT)
INTEGER D1, B1, D2, B2
LOGICAL EVEN
C
D1=15 $ B1=110 $ D2=16 $ B2=130
C
NDAG=D1-(D1/2)*2
EVEN=NDAG.EQ.0
NTEST=150 $ IF(EVEN) NTEST=100
TEST1=2HNO $ IF(B1.GT.NTEST) TEST1=3HYES
C
NDAG=D2-(D2/2)*2
EVEN=NDAG.EQ.0
NTEST=150 $ IF(EVEN) NTEST=100
TEST2=2HNO $ IF(B2.GT.NTEST) TEST2=3HYES
C
CALL CONNEC(6LOUTPUT)
WRITE(1,5) D1,B1,TEST1,D2,B2,TEST2
5 FORMAT(/1H " D1 B1 TEST1"
*/1H 2I7,3X,A3//
*/1H " D2 B2 TEST2"
*/1H 2I7,3X,A3)
C
RATIO=FLOAT(B2)/FLOAT(B1)
WRITE(1,6) RATIO
6 FORMAT(/1H "RATIO EVEN/ONEVEN" E15.4)
C
END
```

*)

```

XEOF
REWIND,T
FTN,I=T,L=0
LGO
EX
```

```

*) OOK IS
TOEGESTAAN
CALL CONNEC(1)
```

```

COMMAND-
COMMAND-
COMMAND-
.393 CP SECONDS COMPILATION TIME
COMMAND-
```

```

D1      B1  TEST1
15      110  NO

D2      B2  TEST2
16      130  YES

RATIO EVEN/ONEVEN      .1102E+01
END MYTEST
.022 CP SECONDS EXECUTION TIME
```

V

A R R A Y S

V.1. GEÏNDICEERDE VARIABELEN

Vaak komt het voor dat men een variabele een naam geeft en er bovendien een index aan toevoegt om verschillende exemplaren aan te duiden. Bijvoorbeeld:

$$x_1, x_2, x_3, \dots, x_i, \dots, x_{n-1}, x_n$$

Hierin wordt x_i het willekeurig (arbitrary) element genoemd. In Fortran schrijft men

$$X(1), X(2), X(3), \dots, X(I), \dots, X(N-1), X(N)$$

Er zijn in totaal drie indices (subscripts) toegestaan, bijvoorbeeld $X(I, J, K)$. Het aantal indices en de grootte van de arrays moet aan het begin van het programma gedeclareerd worden. Zie hiervoor hoofdstuk V.3.

Het komt vaak voor dat men getallen in een twee-dimensionaal array of matrix of tabel zal gaan samenvatten. Een voorbeeld wordt gegeven in tabel 3.

Tabel 3. Grondwaterstanden in cm-maaiveld

Datum	Meetpunt nr.				
	j=	30H8 (1)	21La6 (2)	302(VI) (3)	brandput (4)
i					
1 jan. 78 (1)		10	75	20	112
14 jan. 78 (2)		10	78	20	112
1 febr. 78 (3)		15	71	25	110
14 febr. 78 (4)		15	72	-	118
1 mrt. 78 (5)		29	84	-	120
14 mrt. 78 (6)		30	79	61	150
1 apr. 78 (7)		30	87	60	droog
14 apr. 78 (8)		35	93	63	droog

- niet gemeten

We hebben nu een matrix (tabel) met rijen voor datum en kolommen voor meetpunt. De volgorde van de indices is altijd X (rij, kolom) en met drie indices X(rij, kolom, tabel). Enkele voorbeelden zijn:

I=1, J=3 geeft X(I,J)=20.
I=6, J=2 " X(I,J)=79.
I=3, J=3 " X(I,J)=25.

De gemiddelde waterstand van 30H8 en 21La6, op 1 april 1978, volgt uit

$$GEM = (X(7,1) + X(7,2))/2.$$

Hieruit blijkt al het grote voordeel van het werken met indices. In de bovenstaande uitdrukking is I=7 (datum) constant. Men hoeft nu alleen de kolomnummers bij te houden en niet meer de naam van de variabele te veranderen.

Bij het gebruik van indices is de plaats van de index in het argument essentieel, en niet de naam ervan. Voorbeeld:

I=2
J=4
X(I,J)=112. maar X(J,I)=72.

Daarentegen met:

KK=2
LL=2*KK
X(KK,LL)=112.

Wanneer indices niet opgegeven worden dan wordt dit door de computer geïnterpreteerd alsof de niet opgegeven index (indices) gelijk aan 1 is, terwijl aangenomen wordt dat indices van achteraf ontbreken. Bijvoorbeeld:

X(I,J)	betekent voor X(I,J,K) het element X(I,J,1)					
X(I)	"	"	X(I,J,K)	"	"	X(I,1,1)
X	"	"	X(I,J,K)	"	"	X(1,1,1)
X(I)	"	"	X(I,J)	"	"	X(I,1)
X	"	"	X(I,J)	"	"	X(1,1)
X	"	"	X(I)	"	"	X(1)

zodat in tabel 3, de uitdrukking

$$Y = 2. * X(5)$$

voor Y oplevert 58.

De uitzondering op dit gelijk aan 1 nemen van niet genoemde indices is de input en de output waar onder X de volledige matrix wordt verstaan, evenals in het DATA-statement (hoofdstuk X.3).

Het is goed zich het voorgaande steeds duidelijk te realiseren en zoveel mogelijk elk element uit een array met alle daarin voorkomende indices (1, 2 of 3 stuks) op te geven om vergissingen te vermijden.

Bij het vertalen van een programma wordt overigens wel gemeld (informative error, geen fatal error) dat indices van een subscripted variabele ontbreken. Het is dus goed na een eerste vertaling van een programma, de output op dit soort aanwijzingen na te lezen.

Opgemerkt wordt dat het duidelijk zal zijn dat de aantekeningen '-' en 'droog' in tabel 3 niet in een numerieke verwerking passen. Zo heeft het geen zin te definiëren:

$X(5,3) = 1H-$ en $X(7,4) = 5H\text{DROOG}$. Deze elementen van de array zijn nu een ander type grootheid geworden en kunnen niet meer in één routine worden verwerkt met de elementen die van het numerieke type zijn. Beter is het een onmogelijke waarde als onbekend-code in te voeren. Hierop kan worden getest zodat de afwijkende waarden steeds terug te vinden zijn. Bijvoorbeeld

$$X(4,3) = X(5,3) = X(7,4) = X(8,4) = 999.$$

Ook zou kunnen = -100. enz.

V.2. HET WERKEN MET INDICES

In de hier behandelde Fortran versie mogen indices bestaan uit rekenkundige uitdrukkingen. Bijvoorbeeld:

I=2

J=3

Y=X(3*I - 1, J-1)

wat oplevert Y=X(5,1) = 29.

Dergelijke uitdrukkingen behoeven niet tot integers beperkt te blijven. Wel geldt dat de index zelf een integer moet zijn, groter dan 0 maar niet groter dan de ervoor opgegeven dimensie. (Zie hoofdstuk V.3).

Het grote voordeel van het werken met geïndiceerde variabelen zal vooral blijken uit de toepassing van DO-loops (hoofdstuk VII.5). Hier wordt nogmaals gewezen op het verschil tussen X1 en X(1). In het eerste geval één enkele variabele die alleen met naam oproepbaar is en voor de computer totaal verschillend is van X2. (Zie D1, D2 en B1, B2 in voorbeeld 4). In het tweede geval een variabele die verband houdt met X(2). Beide zijn met nummer oproepbaar bijvoorbeeld in:

I=0

I=I+1 (I=1)

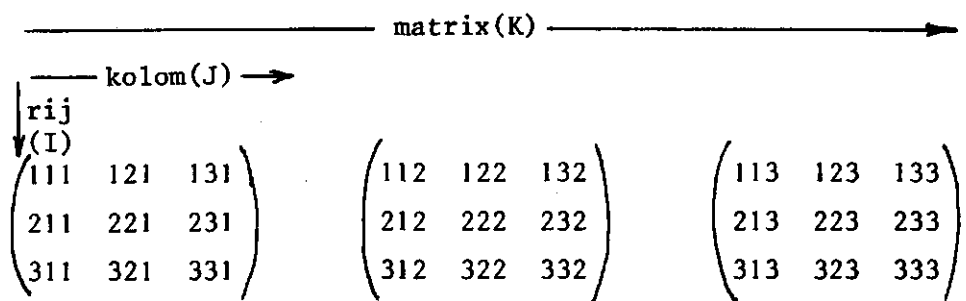
Z=2.*X(I) (opgehaald wordt X(1))

I=I+1 (I=2)

U=2.*X(I) (" " X(2))

Soms wordt in deze gevallen gesproken van 'indirecte adressering', namelijk we hebben niet de waarde nodig die in I staat opgeslagen maar I verwijst naar de plaats in de array X waar de te gebruiken waarde opgeslagen is.

De volgorde waarin elementen in arrays worden afgeteld is van links naar rechts, dus eerst de rij, dan de kolom, dan de matrix. Voorbeeld:



V.3. DE GROOTTE VAN ARRAYS

Wanneer men geïndiceerde variabelen gaat gebruiken moet aan de computer opgegeven worden wat de omvang van de arrays zal zijn. Dit is noodzakelijk voor de automatisch te verzorgen geheugen-indeling.

De instructie hiervoor is de DIMENSION. Opgegeven moet worden het maximum aan ruimte dat men denkt te gebruiken. Dit aantal kan het exacte aantal zijn maar soms zal men in de praktijk de array-grootte wat ruimer nemen ten einde bijvoorbeeld alsnog gegevens te kunnen toevoegen. Wel is het verstandig de totale aanspraak op geheugenruimte niet uit voorzorg te overdrijven. Kleinere programma's worden vaak met een hogere prioriteit verwerkt. Overigens is op Cyber-computers een array van enige duizenden elementen nog niet noemenswaardig groot.

In het voorbeeld van tabel 3 zou men kunnen volstaan met

```
DIMENSION X(8,4)
```

wil men extra ruimte reserveren, dan bijvoorbeeld nemen

```
DIMENSION X(100,10)
```

ook al wordt met gegevens uit een bepaald onderzoek dit aantal niet gebruikt.

Na een DIMENSION mogen meer dan één geïndiceerde variabelen worden genoemd. Er mogen meer DIMENSION opdrachten na elkaar voorkomen. Voorbeeld:

```
DIMENSION X(50,4), Y(11,30), Z(50,30)
DIMENSION MYJOB(26,16), KK(10,10), U(20)
DIMENSION MATRIX(24,30,50), S(2)
```

Het is geoorloofd de dimensionering te verzorgen tegelijk met de type declaratie. Bijvoorbeeld:

```
REAL M(10,10), LJK(5)
INTEGER E1(2,4,6), FM(100,20)
```

Dit is hetzelfde als bijvoorbeeld:

```
REAL M
DIMENSION M(10,10)
```

enz.

Zie ook het COMMON-statement in hoofdstuk X.2.

V.4. TOEPASSINGEN

Een aantal aspecten uit dit hoofdstuk V, worden weer toegepast in twee voorbeelden. Als eerste is gekozen voorbeeld 4 waarin de herhalingen voor D1 en D2 nu netter met indices is geprogrammeerd (zie voorbeeld 6). Als tweede toepassing is gekozen het berekenen van gemiddelde grondwaterstanden per meetpunt uit tabel 3 (zie voorbeeld 7).

Voorbeeld 6. Als voorbeeld 5 nu met geïndiceerde variabelen.

RETURN,LGO,OUTPUT,T
CONNECT,ICW
COPY8F,ICW,T

```

PROGRAM TEST100 (OUTPUT,TAPE3=OUTPUT)
INTEGER DAGBEZ(10)
LOGICAL EVEN
DATA DAGBEZ/115, 120, 170, 175, 92, 90, 101, 112, 100, 190/
C
C  WRITE HEADING
C
C  WRITE(3,5)
5  FORMAT(///4H "DAG    DAGBEZOEK    TESTWAARDE    BEZOEK-GROTER-DAN-
*TESTWAARDE"/)
C
C  INITIALISEER TELLER VOOR DAGEN
C
C  I=0
100 I=I+1
C
C  EVEN =(I-(I/2)*2) .EQ. 0
NTEST=150 $ IF(EVEN) NTEST=100
GTEST=2HNO $ IF(DAGBEZ(I).GT.NTEST) GTEST=3HYES
C
C  OUTPUT VAN RESULTATEN
C
C  WRITE(3,6) I, DAGBEZ(I), NTEST, GTEST
6  FORMAT(4H I3, I10, I14, 20X, A5)
C
C  IF(EVEN) RATIO=FLOAT(DAGBEZ(I))/FLOAT(DAGBEZ(I-1))
IF(EVEN) WRITE(3,7) RATIO
7  FORMAT(1H+70X"RATIO EVEN/ONEVEN="E15.5)
IF(I.LT.10) GO TO 100
STOP 5
END

```

XEOF
REWIND,T
FTN,I=T,L=0
LGO
REWIND,OUTPUT
COPY8R,OUTPUT,ICW
EX

COMMAND-
COMMAND-
COMMAND-
COMMAND-
COMMAND- .340 CP SECONDS COMPILATION TIME
COMMAND-
COMMAND- STOP 5
COMMAND- .044 CP SECONDS EXECUTION TIME
COMMAND-
COMMAND-

DAG	DAGBEZOEK	TESTWAARDE	BEZOEK-GROTER-DAN-TESTWAARDE	
1	115	150	NO	
2	120	100	YES	RATIO EVEN/ONEVEN= .10435E+01
3	170	150	YES	
4	175	100	YES	RATIO EVEN/ONEVEN= .10294E+01
5	92	150	NO	
6	90	100	NO	RATIO EVEN/ONEVEN= .97826E+00
7	101	150	NO	
8	112	100	YES	RATIO EVEN/ONEVEN= .11089E+01
9	100	150	YES	
10	190	100	YES	RATIO EVEN/ONEVEN= .10556E+01

Voorbeeld 7. Het berekenen van gemiddelde waterstanden voor gegevens vermeld in tabel 3.

```

RETURN,LGO,T,OUTPUT
CONNECT,ICW
COPYBF,ICW,T
PROGRAM WATER (OUTPUT,TAPE3=OUTPUT)
DIMENSION M(40,5), GEM(5), FN(5)
DATA      M/  10, 10, 15, 15, 29, 30, 30, 35, 999, 999,
*          75, 78, 71, 72, 84, 79, 87, 93, 999, 999,
*          20, 20, 25, 999, 999, 61, 60, 63, 999, 999,
*          112, 112, 118, 118, 120, 150, 999, 999, 999, 999,
*          999, 999, 999, 999, 999, 999, 999, 999, 999, 999/
C
J=0      $ WRITE(3,5)
5        $ FORMAT(4H1 "SERIE  AANTAL  GEMIDDELDE"/)
10 J=J+1  $ IF(J.GT.5) STOP 1
I=0      $ N=ISOM=0
C
20 I=I+1  $ IF(I.GT.40) GO TO 40
IF(M(I,J).GT.999.) GO TO 20
N =N  +1
ISOM=ISOM+M(I,J)  $ GO TO 20
C
40 FN (J)=N  $ SOM=ISOM  $ IF(N.EQ.0) FN(J)=1.
GEM(J)=SOM/FN(J)
WRITE(3,50) J, N, GEM(J)
50 FORMAT(4H 14,18,F13.2)
GO TO 10
END
XEOF
REWIND,T
FTN,I=T,L=0
LGO
REWIND,OUTPUT
COPYBR,OUTPUT,ICW
E
X
COMMAND-
COMMAND-
COMMAND-
COMMAND-
.398 CP SECONDS COMPILATION TIME
COMMAND-
STOP 1
.022 CP SECONDS EXECUTION TIME
COMMAND-
COMMAND-

```

SERIE	AANTAL	GEMIDDELDE
1	8	21.75
2	8	79.88
3	6	41.50
4	6	120.33
5	0	0.00

Opmerking

Wanneer in een programma tijdens de uitvoering de dimensies van een geïndiceerde variabele wordt overschreden, wordt geen foutmelding gegeven. In het geheugen wordt het getal opgezocht dat op de plaats staat die overeenkomt met de aangeduide positie.

Voorbeeld:

```
DIMENSION X(10)
```

```
.  
. .  
. .
```

```
I=15
```

```
Z=X(I)
```

```
.  
. .  
. .
```

Z heeft nu de waarde van het getal dat 5 plaatsen verder in het geheugen staat dan X(10). Meestal zal niet bekend zijn welke variabele daar staat. De uitkomsten zijn in dit geval onzeker en fout. Zijn variabelen in een COMMON opgenomen (hoofdstuk X.2) dan valt na te gaan welke variabele in zo'n geval opgeroepen wordt. Veelal zal een dergelijke behandeling van variabelen niet de bedoeling van de programmeur zijn. Hier moet men oplettend te werk gaan.

De plaats (p) waar een element van een array vanaf het beginpunt voorkomt kan met de volgende formules worden bepaald:

eerste index $1 \leq I \leq N$

tweede index $1 \leq J \leq M$

derde index $1 \leq K \leq L$

Bij één index $p = I$ (1)

Bij twee indices $p = I + (J-1)N$ (2)

Bij drie indices $p = I + (J-1)N + (K-1)NM$ (3)

In deze formules mogen I, J en K groter dan respectievelijk N, M en L gekozen worden.

Voor de matrices gegeven in hoofdstuk V.2 geldt nu N=3, M=3 en L=3. (Zie pag. 61 bovenaan).

De elementen van deze matrix worden in de volgende volgorde in het geheugen opgeborgen

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \quad \begin{pmatrix} 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{pmatrix} \quad \begin{pmatrix} 19 & 22 & 25 \\ 20 & 23 & 26 \\ 21 & 24 & 27 \end{pmatrix}$$

bijvoorbeeld:

variabele	positie volgens (3)	opmerking
X(1,1,1)	1	correct
X(2,3,2)	17	correct
X(5,4,2)	20	I > 3, J > 3 !!
X(2,1,3)	20	correct
X(3,3,3)	27	correct
X(1,1,4)	28	K > 3 !

In het laatste geval wordt de eerste variabele die ná de array voor X in het geheugen staat, aangeroepen ook al is de naam een geheel andere.

VI

PROGRAMMEREN VAN FORMULES

VI.1. REKENKUNDIGE UITDRUKKINGEN

Over het algemeen kan men rekenkundige uitdrukkingen of algebraïsche formules gemakkelijk en rechtstreeks in Fortran opschrijven. De belangrijkste regel hierbij is dat in het linkerlid van een vergelijking slechts één variabele mag voorkomen. Dit mag wél een geïndiceerde variabele zijn.

In het rechterlid van een uitdrukking mogen alleen grootheden voorkomen die reeds eerder een waarde hebben verkregen, bijvoorbeeld doordat ze ingelezen zijn of al eens in het linkerlid van een uitdrukking voorgekomen zijn of in een statement zijn gedefinieerd. (Zie hoofdstuk I.6).

De volgende bewerkingstekens of operatoren kunnen in Fortran worden gebruikt.

**	machtsverheffen	(niveau 1)
*	vermenigvuldigen	(niveau 2)
/	delen	(niveau 2)
+	optellen	(niveau 3)
-	af trekken	(niveau 3)

Voor de volgorde van bewerken gelden dezelfde regels als bij het rekenen. De bewerkingen van eenzelfde niveau worden in leesrichting uitgevoerd. In het bijzonder is nog

$$A**B**C = (A**B) **C$$

Men kan al naar het noodzakelijk is, of voor de duidelijkheid wenselijk is, haakjes gebruiken. De bewerking wordt dan uitgevoerd vanuit het binnenste paar haakjes.

Aangezien er geen orde van haakjes in de schrijfwijze tot uiting kan worden gebracht moet men opletten te werk gaan en controleren of er evenveel sluithaakjes als openingshaakjes zijn toegepast.

Voorbeeld:

$$y = a[b + c\{d + e(f + g)\}]$$

wordt

$$Y = A*(B + C*(D + E*(F + G)))$$

Er zijn dus 3 openingshaakjes en 3 sluihaakjes. Het gebruik van * is voor elke vermenigvuldiging verplicht.

Enkele voorbeelden zijn nog

$$y_{ij} = x_i^2 + 2u_j z$$

wordt

$$Y(I,J) = X(I)**2 + 2.*U(J)*Z$$

en

$$h = \frac{a^b}{c} + def - g$$

wordt geprogrammeerd als

$$H = A**B/C + D*E*F - G$$

wat door de computer als volgt wordt verwerkt:

A**B	geeft	R ₁
R ₁ /C	"	R ₂
D*E	"	R ₃
R ₃ *F	"	R ₄
R ₂ + R ₄	"	R ₅
R ₅ - G	"	resultaat

En tenslotte nog

$$h = \frac{a^b}{c + d}(ef - g)$$

wat wordt geprogrammeerd als

$$H = A**B/(C + D) * (E*F - G)$$

en door de computer wordt verwerkt als volgt:

A**B	geeft	R ₁
C + D	"	R ₂
R ₁ /R ₂	"	R ₃
E*F	"	R ₄
R ₄ - G	"	R ₅
R ₃ *R ₅	"	resultaat

Soms is het efficient een formule enigszins te analyseren en na te gaan of een andere schrijfwijze tot een snellere verwerking kan leiden. Een bekend voorbeeld is de hogere machtsfunctie.

Wanneer we uitgaan van een derde-graads functie

$$y = ax^3 + bx^2 + cx + d \quad (1)$$

dan kan als alternatieve schrijfwijze gekozen worden

$$y = (((a) x + b) x + c) x + d \quad (2)$$

Wordt nu de machtsverheffing in (1) gezien als een voortgezette vermenigvuldiging, dan is het aantal operaties als volgt

Formule	Aantal vermenigvuldigingen	Aantal optellingen
(1)	6	3
(2)	3	3

wat in deze eenvoudige formule dus al een winst van drie vermenigvuldigingen betekent.

Opmerking: Er mogen nooit twee operatoren achter elkaar voorkomen. Niet toegestaan is dus $A = X * - B$. Dit moet geschreven worden $A = X*(-B)$. Wél is toegestaan $A = -B$.

VI.2. RELATIES

Numerieke uitdrukkingen kunnen in een bepaalde relatie tot elkaar staan. Nemen we $x = 3 \cdot 5$ en $y = 2^3$ dan is $x > y$. We kunnen zes relaties onderscheiden. Deze staan in het volgende tabellarisch overzicht samengevat.

Symbool	Fortran	Betekenis
=	.EQ.	equal to
$\neq$	.NE.	not equal to
>	.GT.	greater than
$\geq$	.GE.	greater than or equal to
<	.LT.	less than
$\leq$	.LE.	less than or equal to

Het resultaat van een relatie is een logische uitkomst. Namelijk in het voorbeeld is $X.GT.Y$ wáár (TRUE). Er bestaat een Fortran-statement om te testen of $X.GT.Y$ voor een gegeven waarde van X en een gegeven waarde voor Y inderdaad wáár is of niet. Hiervoor wordt verwezen naar hoofdstuk VII.4.2 waarin de logical IF besproken wordt. Deze IF is overigens al een aantal malen in enkele voorbeelden voorkomen.

Een logische variabele wordt gedefinieerd met het statement

LOGICAL namel,...,namen

waarin namel,...,namen de namen van te gebruiken logische variabelen zijn.

Wanneer we ons afvragen of een gegeven waarde van x in het interval (x_1, x_2) ligt, dan moet aan beide volgende relaties voldaan zijn:

$$x > x_1 \quad \text{en bovendien} \quad x < x_2$$

Dit wordt geschreven (zie hoofdstuk IV.7) als

$$U = X.GT.X1 \quad .AND. \quad X.LT.X2$$

waarin U als logische variabele moet zijn gedefinieerd. U is wáár wanneer x inderdaad in het interval ligt.

Voor het werken met de operatoren .AND. en .OR. bestaan aparte regels. We noemen er hier een die nogal eens voorkomt en wel wanneer we de ontkenning willen van U. Deze houdt in dat U niét-waar is wanneer x buiten het interval (x_1, x_2) ligt. Dit nu is wáár wanneer òf $x < x_1$ òf $x > x_2$ (tegelijk kan niet) zodat

.NOT. U = X.LT.X1 .OR. X.GT.X2

De operator .AND. gaat hierbij over in .OR. Ook het omgekeerde geldt.

De volgorde waarin de genoemde bewerkingen met betrekking tot relaties (en logische variabelen) wordt uitgevoerd is (zie ook hoofdstuk VI.1, pag. 68)

relaties	(niveau 4)
.NOT.	(niveau 5)
.AND.	(niveau 6)
.OR.	(niveau 7)

VI.3. VEEL VOORKOMENDE FUNCTIES

Fortran kent een groot aantal ingebouwde functies die met naam direct oproepbaar zijn. Deze functies maken het mogelijk de meest voorkomende wiskundige uitdrukkingen en bijzondere functies direct op te schrijven.

Voor een goed begrip voor de schrijfwijze van deze functies moet erop gewezen worden dat de naamgeving ook gebonden is aan de type-afspraken.

Van alle beschikbare functies die in de handleidingen worden genoemd, zijn een aantal belangrijke in het volgende overzicht opgenomen. (Zie tabel 4).

Tabel 4. Veel voorkomende functies die in Fortran met naam oproepbaar zijn

Functie (wiskundig)	Functie (Fortran)	Voorbeeld	Type van	
			argument	functie
$z = e^a$	EXP	$Z = \text{EXP}(A)$	real	real
$z = \ln x$	ALOG	$Z = \text{ALOG}(X)$	"	"
$z = \log x$	ALOG10	$Z = \text{ALOG10}(X)$	"	"
$y = \sin u$	SIN	$Y = \text{SIN}(U)$	"	"
$y = \cos v$	COS	$Y = \text{COS}(V)$	"	"
$y = \tan x$	TAN	$Y = \text{TAN}(X)$	"	"
$x = \arcsin y$	ASIN	$X = \text{ASIN}(Y)$	"	"
$x = \arccos y$	ACOS	$X = \text{ACOS}(Y)$	"	"
$x = \arctan y$	ATAN	$X = \text{ATAN}(Y)$	"	"
$x = \arctan (a/b)$	ATAN	$X = \text{ATAN2}(A,B)$	"	"
$y = \sqrt{a}$	SQRT	$Y = \text{SQRT}(A)$	"	"
$y = a $	ABS	$Y = \text{ABS}(A)$	"	"
$j = k $	IABS	$J = \text{IABS}(K)$	integer	integer
conversie	FLOAT	$X = \text{FLOAT}(I)$	"	real
"	IFIX	$I = \text{IFIX}(X)$	real	integer
beginwaarde	RANSET	$\text{CALL RANSET}(U)$	"	-
random $U[0,1]$	RANF	$U = \text{RANF}(\text{DUM})$	-	real
datum	DATE	$T = \text{DATE}(A)$	-	Hollerith
tijdstip	TIME	$T = \text{TIME}(A)$	-	"
tijdsverloop	SECOND	$T = \text{SECOND}(A)$	real	real

VI.4. VERVANGINGSOPDRACHTEN

De vervangingsopdrachten, ook wel toekenningsopdracht genoemd (assignment statements) nemen bij het programmeren een centrale plaats in. Ze werden reeds gebruikt in het hoofdstuk over Rekenkundige uitdrukkingen (VI.1). De feitelijke betekenis van deze uitdrukkingen is echter anders dan die in de dagelijkse wiskunde. Om deze reden zal daarom nu meer in detail op deze vervangingsopdrachten

worden ingegaan.

De vorm van de vervangingsopdracht komt sterk overeen met algebraïsche uitdrukkingen of wiskundige formules. In het voorgaande werden reeds voorbeelden hiervan gegeven.

De algemene gedaante van een vervangingsopdracht is

$$y = \text{expressie}$$

Toch moet bedacht worden dat het "="-teken in vervangingsopdrachten een essentieel andere betekenis heeft dan hetzelfde teken in een wiskundige vergelijking. Dit houdt verband met het feit dat grootheden in een Fortran tekst een verwijzing betekenen naar een plaats in het geheugen van de computer en dat de inhoud van deze plaats in het geheugen een bepaald getal kan zijn.

Algebraïsch is zinvol om te schrijven

$$y = x + a$$

waarmee medegedeeld wordt dat y ontstaat door bij elke voorkomende waarde van x een vast bedrag a bij te tellen.

In Fortran betekent

$$Y = X + A$$

dat de geheugenplaats Y gevuld moet worden met de som van de inhoud van geheugenplaatsen X en A, en is dus geen algemene uitdrukking voor het toekennen van waarden aan de variabele Y.

Het verschil in betekenis van deze schrijfwijze komt het duidelijkst tot uiting in statements van het type

$$Y = Y + A$$

Algebraïsch zou dit betekenen $A=0$. In Fortran betekent dit statement: de geheugenplaats aangeduid met Y moet gevuld worden met de som van de inhoud van geheugenplaatsen Y en A.

Hier volgt meteen uit dat de computer een statement slechts kan verwerken wanneer alle genoemde variabelen in het rechterlid een getalswaarde toegekend hebben gekregen. Dit moet dan vooraf op een van de volgende manieren zijn gebeurd:

- door inlezen vanuit een file met gegevens
 - door in het programma een waarde aan een variabele toe te kennen (b.v. A=3.)
 - door voorgaande berekeningen
- (zie vollediger in hoofdstuk I.6, pag. 18 en 20).

Uit het bovenstaande wordt dus opnieuw duidelijk dat een computerprogramma niet algebraïsch maar numeriek werkt. Dus:

Y = 5.

A = 3.

Y = Y + A

betekent dat in Y nu de getalswaarde 8 is opgeslagen. De oude waarde is verloren gegaan: is overschreven.

Formeel zou men dit '='-teken moeten uitspreken als 'wordt'. Het '='-teken heeft dus een definiërend karakter en niet een gelijkstellend karakter. In programmeertalen als Pascal en Algol wordt dan ook geschreven Y:= Y + A om het bijzonder gebruik van de = tot uitdrukking te brengen.

Statements van het type Y = Y + A worden zeer veelvuldig gebruikt. Ze kunnen heel geschikt dienen om lange onoverzichtelijke formules snel en overzichtelijk te programmeren. Voorbeeld:

$$y = x^2 + 3u + 7\{\cos(t + \sqrt{z}) + \frac{\psi}{L}\}$$

wat als volgt geprogrammeerd zou kunnen worden:

Y = T + SQRT(Z)

Y = COS(Y) + PSI/FL

Y = 7.*Y

Y = X**2 + 3.*U + Y

Tot hoever men met uitsplitsen wil gaan hangt alleen van de persoonlijke voorkeur van de programmeur af en eventueel van verder uit te voeren bewerkingen. Stel men heeft

$$y = x^2 + \cos(t + \sqrt{z}) + \ln u$$

$$s = x^2 + \ln u + \sin t$$

dan kan $x^2 + \ln u$ uit de eerste formule bewaard blijven om sneller de waarde van s te kunnen bepalen met bovendien minder schrijfwerk. Bijvoorbeeld als volgt:

```
X2U = X**2 + ALOG(U)
Y   = X2U + COS(T + SQRT(Z))
S   = X2U + SIN(T)
```

Opgemerkt wordt nogmaals dat een variabele steeds de laatste opgegeven of berekende waarde heeft. Met het nalezen van een programma moet men hier steeds attent op zijn. Voorbeeld:

```
A = 5.
.
.
.
A = 8.
B = 3.
.
.
.
B = B + A
C = A + B
```

dan heeft C de waarde 19.

Het is toegestaan meer variabelen tegelijk een waarde te geven. Bijvoorbeeld

```
I = Y = F = A*B + ALOG(Z)
```

In het voorgaande is hierover reeds meer gezegd. (Zie hoofdstuk IV.4, pag. 46).

VI.5. TOEPASSINGEN

Het gebruik van de vervangingsopdrachten bij het programmeren van formules en rekenkundige uitdrukkingen behoeft na het voorgaande weinig verder commentaar.

Hier wordt nog gewezen op een toepassing bij het berekenen van een eerste afgeleide van een functie. Het toepassen van vervangingsopdrachten en het gebruik van de kettingregel bij het differentiëren kan men nu samen laten gaan.

Bijvoorbeeld in

$$y = \sin(ax + b)$$

is volgens de kettingregel

$$\frac{dy}{dx} = \cos(ax + b) \cdot a$$

Door substituties toe te passen wordt verkregen,

$$p = ax + b$$

$$y = \sin p$$

zodat

$$\frac{dy}{dx} = \cos p \cdot \frac{dp}{dx}$$

waarin

$$\frac{dp}{dx} = a$$

zodat meteen al te zien valt dat geschreven kan worden

$$\frac{dp}{dx} = a$$

$$\frac{dy}{dx} = \cos p \cdot \frac{dp}{dx}$$

in de volgorde waarin de substituties zijn uitgevoerd. Door het gebruik van hulpvariabelen tot op een voldoende laag niveau van x , wordt

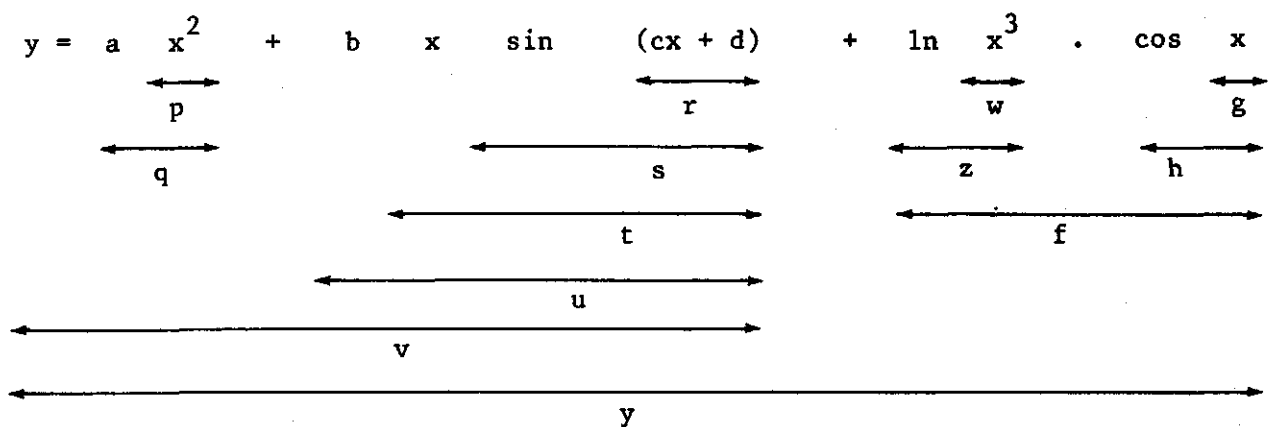
het differentiëren eenvoudiger en ook het programmeren er van. Aan-
gezien in afgeleiden vaak onderdelen van de functie zelf terugkeren,
is het invoeren van substituties ook om deze redenen efficient.

Met de te volgen werkwijze zal het aantal Fortran regels toene-
men. De programmering wordt echter aanmerkelijk eenvoudiger.

Als voorbeeld wordt aangenomen dat we hebben

$$y = ax^2 + bx \sin(cx + d) + \ln x^3 \cdot \cos x$$

We voeren de volgende hulpvariabelen in:



en programmeren dit als volgt:

```

P = X*X
Q = A*P
R = C*X + D
S = SIN(R)
T = X*S
U = B*T
V = Q + U
W = P*X
Z = ALOG(W)
G = X
H = COS(G)
F = Z*H
Y = V + F

```


Voor het programmeren van de afgeleiden zullen we schrijven PX voor dp/dx , enz. Verder moet bedacht worden dat alle hulpvariabelen weer functies zijn van x. Er komt dan direct in Fortran:

```
PX = 2.*X
QX = A*PX
RX = C
SX = COS(R)*RX
TX = S + X*SX
UX = B*TX
VX = QX + UX
WX = PX*X + P
ZX = WX/W
GX = 1.
HX = -SIN(G)*GX
FX = ZX*H + Z*HX
YX = VX + FX
```

Op deze wijze wordt zeer overzichtelijk en zonder gecompliceerde formuleringen het resultaat direct in Fortran geprogrammeerd.

Het geheel kan meer compact geschreven worden met

```
P = X*X    $  Q = A*P
R = C*X + D  $  S = SIN(R)  $  T = X*S  $  U = B*T
V = Q + U
W = X*P    $  Z = ALOG(W)  $  G = X  $  H = $ COS(G)
F = Z*H
Y = V + F
```

enz.

Voor verdere toepassingen van deze techniek onder andere voor partiële afgeleiden en hogere afgeleiden wordt verwezen naar STOL (1977).

VII

INTERNE BESTURING

VII.1. INLEIDING

De programmeur kan al naar dat het noodzakelijk is de route door de Fortran tekst aanpassen aan zijn specifieke eisen. Het kan bijvoorbeeld efficiënt zijn een bepaald gedeelte uit de tekst opnieuw te gebruiken. Het is dan niet noodzakelijk dit deel opnieuw op te schrijven maar men kan de besturing zo inrichten dat het te herhalen deel opnieuw wordt doorlopen.

Ook kan het voorkomen dat een programma gedeelte afhankelijk van vorige uitkomsten al of niet moet worden doorlopen. Hulpmiddelen zijn dan de IF-statements. De verschillende mogelijkheden hiervan zullen thans worden besproken.

VII.2. CONTINUERING

De eenvoudigste vorm van interne besturing is het normaal regel na regel vervolgen van de route door het programma. De opdracht hier toe luidt:

CONTINUE

Aangezien het 'normaal doorgaan' de gewone gang van zaken is, zal de continue meestal juist gelabeld voorkomen om er van een ander deel van het programma naar toe te kunnen springen. De continue is hiervoor niet altijd noodzakelijk maar wel vaak overzichtelijk. De volgende serie opdrachten voeren dan ook tot hetzelfde resultaat (verondersteld wordt dat men met een GO TO 210 naar label 210 wil springen):

```

      .
      .
      .
      A = B + X
C
      210 CONTINUE
C
      Z = B + X**2
      .
      .
      .

```

en, korter maar wat minder overzichtelijk,

```

      .
      .
      .
      A = B + X
      210 Z = B + X**2
      .
      .
      .

```

Let wel dat komend van $A = B + X$ het programma normaal vervolgd wordt en het label niet ter zake doet.

Labelnummers kunnen vrij gekozen worden. Het is niet noodzakelijk ze in volgorde te kiezen. Wel is het overzichtelijk om in een bepaalde eenheid in het programma bij elkaar behorende nummers te kiezen. Dit maakt het teruglezen door de programmeur zelf gemakkelijker.

Ook is het verstandig de nummering niet aansluitend te kiezen. Bij latere wijzigingen en aanvullingen kan men dan zonodig in hetzelfde tiental nummers tussenvoegen.

VII.3. ONVOORWAARDELIJKE SPRONGOPDRACHTEN

Het springen naar een ander (eerder of verder) gedeelte van het programma vindt plaats met het statement

```
GO TO n
```

waarin n het labelnummer van de opdracht is waar men naar toe wil.

Bijvoorbeeld:

```
.  
.   
.   
X = A**2/(B - C)  
GO TO 210  
.   
.   
.   
A = B + X  
210 Z = B + X**2  
.   
.   
.
```

} wordt overgeslagen

Wanneer men in een programma terugspringt moet er voor gewaakt worden dat er geen oneindig durende lussen (loops) of fuiken worden gecreëerd. Bijvoorbeeld zoals het geval is met

```
U = ALOG(Z) + 2.*V**3  
91 A = B + U  
Y = A + SQRT(U)  
GO TO 91  
Y = 10.*U
```

} wordt steeds weer herhaald

De besturing komt nu in een oneindige loop en de executie van het programma eindigt pas als de tijdlimiet die voor iedere gebruiker geldt, wordt overschreden. (Als de rem stuk is moet men doorrijden tot de benzine op is!!). Een kleine berekening kan zo een duur karwei worden!

Sprongen terug in het programma gaan dan ook meestal gepaard met voorwaardelijke sprongopdrachten welke later in dit hoofdstuk aan de orde komen.

Het behandelde geval geeft een sprong (jump) naar één ander statement. Het kan echter voorkomen dat men vanaf een bepaald punt, afhankelijk van de voorgeschiedenis, naar verschillende andere punten in het programma wil gaan. Dit kan worden bereikt met een compu-

ted-go-to. Dit statement heeft de volgende structuur.

GO TO($l_1, l_2, \dots, l_i, \dots, l_n$), i

waarin l_i labelnummers zijn en i een integer variabele. In de loop van het programma moet i worden berekend waarna vanaf de computed GO TO de sprong (many-branch) plaats vindt. Voorbeeld:

```

:
:
NU NAAR = 5
:
:
GO TO (10, 6, 1, 2, 3, 100, 6) NUNAAR
:
:
```

waarna gesprongen wordt naar het statement met labelnummer 3.

In de moderne opvatting van het programmeren wordt uitgegaan van het principe dat, althans in theorie, de GO TO een overbodig statement is en dat het gebruik er van, in de praktijk, zoveel mogelijk moet worden vermeden. Dit is mogelijk wanneer men statement groepen tot procedures samenvoegt. Dit laatste is nu juist in Fortran maar heel beperkt mogelijk. Slechts de in het volgende hoofdstuk te behandelen 'logical-if' komt in bescheiden mate aan de theoretisch juiste opzet van een programma tegemoet.

VII.4. VOORWAARDELIJKE SPRONGOPDRACHTEN

Een sprong naar een ander deel van een programma kan afhankelijk gesteld worden van een eerder verkregen uitkomst. Het statement waarmee dit bereikt wordt heet een IF-statement. Er bestaan hiervan twee hoofdvormen, de numerieke- en de logical if. De belangrijkste uit elke groep zal hier besproken worden.

VII.4.1. Numerieke - IF

De algemene vorm luidt:

IF(e) l_1, l_2, l_3

Hierin is e een uitdrukking of expressie, en zijn l_1 , l_2 en l_3 labelnummers. De besturing van deze IF verloopt nu als volgt:

```
e < 0      ga naar l1
e = 0      "    "    l2
e > 0      "    "    l3
```

Het is toegestaan om twee van de drie labelnummers gelijk te nemen, bijvoorbeeld

```
e < 0      ga naar l1
e = 0      "    "    l1
e > 0      "    "    l2
```

dus

```
IF(e) l1, l1, l2
```

Er vindt dus een sprong naar l_1 plaats als $e \leq 0$.

De cyber heeft een intern goed afrondingsmechanisme zodat voor real-variabelen de uitdrukking $e = .0$ zin kan hebben. Vaak verdient het aanbeveling een kleine marge te nemen en te stellen dat $|e| < .000001$ beschouwd zal worden de waarde nul op te leveren. Voor integers bestaat deze moeilijkheid niet en kan $e = 0$ exact juist zijn. Voorbeeld

```
      :
      :
      : IF(5*NK - 35) 10, 70, 70
      :
10  ---
      :
      :
70  ---
      :
      :
```

Ook terugspringen is uiteraard toegestaan.

De statements behorend bij labels 10 en 70 mogen ook weer IF's zijn zodat op deze wijze een complete selectie-procedure kan worden opgebouwd.

Het nadeel van deze IF is dat dit statement niet gemakkelijk 'leest' en ook dat steeds minstens twee nieuwe labelnummers moeten

worden verzonnen. Meer gebruikt wordt dan ook de volgende vorm van de IF.

VII.4.2. Logische IF

De algemene vorm luidt

IF(e) s

waarin e een logische uitdrukking is en s een statement van elk type behalve een andere logical IF of een DO-statement (zie later).

Een voorbeeld van een dergelijke IF is

IF(K.EQ.5) Y = SIN(X)

in woorden: wanneer $k = 5$, dan wordt $y = \sin x$. Is de logical expression niet waar, $k \neq 5$, dan wordt s beschouwd te zijn CONTINUE, en wordt Y niet gelijkgesteld aan SIN(X).

Met deze IF kan men, in bescheiden mate, constructies maken van het type IF-then ..., else ... Goed leesbaar wordt de tekst wanneer men bovendien gebruik maakt van het \$-teken om meer statements op één regel te krijgen. Voorbeelden:

IF(Z.GT..10) GO TO 12 \$ GRW = 120.

dus:

indien $z > 0.10$ ga dan naar 12, anders $g_{rw} = 120$

Merk echter op dat in

IF(Z.GT..10) GRW = 50. \$ GRW = 120.

voor elke waarde van Z de variabele GRW de waarde 120 zal aannemen. Beter is in dit geval GRW reeds vooraf voorlopig te definiëren, dus

GRW = 120. \$ IF(Z.GT..10) GRW = 50.

dus

$g_{rw} = 120$ maar als $z > 0.10$, dan is $g_{rw} = 50$

Dit is een veelvoorkomende vorm van de logical IF waarbij een variabele vóórgedefinieerd wordt en later zonodig wordt aangepast.

Het logical IF-statement wordt ook vaak gebruikt om een variabele binnen een gegeven interval te houden, dus:

```
IF(X.GT.XMAX) X = XMAX
```

```
IF(X.LT.XMIN) X = XMIN
```

Het testen of een variabele X in een gegeven interval ligt kan met

```
IF(X.GT.XMIN .AND. X.LT.XMAX) .....
```

Zie ook wat over dit type logische relaties is gezegd in hoofdstuk VI.2, (pag. 71 e.v.).

Aangezien in het rechterlid van de logical IF slechts één statement mag volgen kan een procedure geprogrammeerd worden door

- de logical IF te herhalen en af te sluiten met GO TO
- meer GO TO's te gebruiken

Een voorbeeld van het eerste geval:

```

:
:
Y = 1.
      IF(X.GT.XMAX) Y = 10.
      IF(X.GT.XMAX) Z = SIN(Y)
      IF(X.GT.XMAX) WRITE(2, 1)X,Y,Z
      IF(X.GT.XMAX) GO TO 50
Z = COS(Y)
WRITE(2, 1) Z
50                                CONTINUE
:
:
```

en van het tweede geval, waarmee hetzelfde bereikt wordt

```

:
:
Y = 1.
      IF(X.GT.XMAX) GO TO 25
Z = COS(Y)
WRITE(2, 1) Z $ GO TO 50
25 Y = 10.
Z = SIN(Y)
WRITE(3, 21)X, Y, Z
50 CONTINUE
:
:

```

In dit geval wordt 'haasje over' gesprongen wat minder overzichtelijk is. Hoewel alleen voor een klein aantal logical IF's efficient, is het eerste geval beter en in een duidelijker structuur geprogrammeerd.

Indien eenzelfde (lang) argument vaker voorkomt is het handiger een logical variabele in te voeren. Bijvoorbeeld

```

LOGICAL XINT
:
:
XINT=X.LT.XMIN.OR.X.GT.XMAX
:
:
Y=1. $ B=5.
IF(XINT) Y=10.
IF(XINT) U=B**2 + B
IF(XINT) WRITE(2,1)X, Y, U
enz.

```

Uit het voorgaande zal duidelijk zijn geworden dat een logical IF een makkelijk en overzichtelijk toe te passen voorwaardelijke 'sprong'opdracht is.

Stel tenslotte dat we de grootste waarde van de elementen uit de array Y(10) willen bepalen. Een gangbare procedure hiervoor is een hulpvariabele in te voeren en deze een zeer kleine waarde te geven. Ga na of Y(1) groter dan deze hulpvariabele is. Zo ja, geef de hulpvariabele dan de waarde van Y(1) en ga na of Y(2) groter is dan Y(1);

zo niet, ga na of Y(2) groter dan de hulpvariabele is, enz. Dus:

```

:
:
YMAX = -9999.
I = 0
25 CONTINUE
I = I + 1
IF(Y(I).GT.YMAX) YMAX = Y(I)
IF(I.LT.10) GO TO 25
:
:
```

In dit stukje programma is een 'lus' ingebouwd om I alle waarden van 1 tot en met 10 te laten doorlopen. Dit zou eenvoudiger kunnen met de in het volgend hoofdstuk te bespreken DO-lussen.

Zie tenslotte het voorbeeld waarin van de logical IF in een programma gebruik gemaakt werd (voorbeelden 2, 3 enz.).

Dat de IF's in praktisch alle voorbeelden reeds zijn voorgekomen toont tevens aan dat het kunnen nemen van elementaire beslissingen een van de sterke kanten van het werken met computers is.

VII.5. DO-LOOPS

Een bijzonder krachtig hulpmiddel bij het overzichtelijk en compact programmeren van weerkerende bewerkingen is de DO-loop. Dit voordeel treedt vooral duidelijk aan het licht bij toepassing op geïndiceerde variabelen (arrays). We stellen het probleem dat gevraagd wordt te berekenen de som van vijf getallen. Dit zou kunnen zijn

$$s = x + y + z + u + v$$

Aangezien hier dezelfde bewerking wordt uitgevoerd op alle variabelen wordt meestal geschreven .

$$s = x_1 + x_2 + x_3 + x_4 + x_5$$

wat weer korter wordt geschreven als

$$s = \sum_{i=1}^5 x_i$$

Hierin heet i de dummy variabele omdat deze op zichzelf niet ter zake doet, want met j is de uitkomst exact hetzelfde namelijk

$$s = \sum_{j=1}^5 x_j$$

Een DO-loop om s te berekenen ziet er nu als volgt uit*

```

      :
      :
      S = .0
[ DO 20 I = 1, 5
  20 S = S + X(I)
      :
      :
```

Hierin kunnen we de elementen waaruit een DO-loop bestaat reeds gemakkelijk onderscheiden. Ze zijn

- het DO-statement dat steeds het beginpunt is van de loop.
- het labelnummer, hier 20, dat verwijst naar de laatste regel van de loop (verder in het programma),
- de dummy variabele welke van het integer type moet zijn, hier I,
- de parameters die aangeven welke waarden de dummy variabele zal doorlopen, hier van 1 tot en met 5. De parameters zijn van het type integer,
- de opdracht met het labelnummer van de DO, van waaraf de besturing automatisch terugspringt naar de DO-statement, waarbij I opgehoogd wordt totdat $I > 5$,
- wanneer I alle gedefinieerde waarden heeft doorlopen heet de DO 'satisfied' en wordt met het statement volgend op de loop verder gegaan.

Een aantal bijzonderheden moet nog worden vermeld om een juist zicht in de werking van de DO te verkrijgen.

*De haken in de kantlijn van de volgende voorbeelden dienen om de structuur van de DO-loops te verduidelijken

- De parameters kunnen met een derde voor de stapgrootte worden uitgebreid (default = 1). Bijvoorbeeld DO 10 I = 4, 20, 2 betekent: geef I de waarden 4, 6, 8, ..., 18, 20. Voor het geval dat DO 10 I = 4, 20, 5 geprogrammeerd wordt, doorloopt I de waarden 4, 9, 14, 19, en de DO wordt verlaten als I = 24, dus groter dan 20.
- Het is niet noodzakelijk dat een geïndiceerde variabele in de DO-lus voorkomt. Bijvoorbeeld:

```

      :
      :
      K = 0
[ DO 10 I = 1, 16
      K = K + 1
10 WRITE(2,5) K
   5 FORMAT(1H "DIT IS REGEL NR" I3)
      :

```

wat aanleiding geeft tot het evenzoveel malen printen van de tekst opgegeven in het FORMAT.

- De laatste opdracht in een DO mag geen sprongopdracht zijn. Deze combinatie kan de compiler niet verwerken. Wél is een logical IF toegestaan.
- De waarde van de dummy variabele mag binnen in de DO-lus niet door een of andere opdracht veranderd worden (mag niet in een linkerlid voorkomen).
- De dummy variabele mag wél in een rechterlid of in een argument voorkomen. Voorbeelden:

```

[ DO 20 I = 1, 10
      K = 2*I + 5
      Z = X(K) - VALUE
      IF(I.EQ.5) WRITE (2,7)
      7 FORMAT(1H )
20 WRITE (2,6) Z
   6 FORMAT(1H "Z WORDT" F10.0)

```

waarmee bij elke vijfde regel van de output een extra regel spatie⁶¹ wordt ingevoegd.

- Wanneer de laatste opdracht in een DO een GO TO-statement is moet de DO met een CONTINUE worden afgesloten. Voorbeeld:

```

DO 5 I = 20, 30
  U = X(I)**2
10 Z(I + 1) = Z(I) + U $ U = U + 75.
  IF(U.GT.1000.) GO TO 5
  GO TO 10
5 CONTINUE

```

- De CONTINUE vervolgt de DO-loop totdat deze *satisfied* is.
- Het is mogelijk real variabelen van I te laten afhangen met bijvoorbeeld ALPHA = I, of te nemen f(I), eventueel met FLOAT(I).
- De parameters mogen variabelen zijn zodat de omvang van de lus bijvoorbeeld met READ kan worden ingelezen of in het programma kan worden berekend. Voorbeeld

```

IF = 2
IL = 2*IF
DO 60 I = IF,IL
.
.
.
60 CONTINUE

```

- Na het beëindigen van de lus is de waarde van de dummy variabele die van na de ophoging waarmee de DO beëindigd werd. Is het DO-statement DO 10 I = 5, 20, 7 dan is na het beëindigen van de DO de waarde van I = 26. Bij stapgrootte 1 is dus altijd I een eenheid groter dan het aangegeven eindpunt.
- Er mag vanuit de DO-loop naar een statement buiten de lus worden gesprongen. Er mag niet zonder meer naar binnen worden gesprongen. De DO-routine is dan niet geïnitieerd.

DO-loops mogen worden genest. Eindpunten van verschillende loops mogen echter niet over elkaar heengrijpen. Een eenvoudig voorbeeld is het geval waarbij alle elementen van de matrix M(5, 10) op nul moeten worden gesteld. Dit wordt bereikt met

```

DO 10 I = 1, 5
DO 10 J = 1, 10
10 M(I, J) = .0

```

De waarden die de dummy variabelen achtereenvolgens aannemen zijn
(I, J) = (1, 1), (1, 2), (1, 3), ..., (1, 10), (2, 1), (2, 2), (2, 3),
..., (2, 10), (3, 1), ..., (3, 10), ..., (5, 9), (5, 10).

Hieruit zien we dat lussen steeds van binnenuit worden afgewerkt.

Stel nu dat de matrix GW(100, 10) van 100 data de grondwaterstand
in 10 meetpunten bevat en dat gevraagd wordt de gemiddelde waterstand
in elk meetpunt te bepalen. Dit kan met het volgende programma-onder-
deel worden bereikt:

```
      :  
      :  
      FN = 100  
DO 20 NR = 1, 10  
      S = .0  
      DO 30 IDATA = 1, 100  
DO 30 S = S + GW(IDATA,NR)  
DO 20 GMEAN(NR) = S/FN  
      :  
      :
```

wat aanmerkelijk eenvoudiger en overzichtelijker is dan de construc-
tie met de IF uit voorbeeld 7, op pag. 64.

In het bovenstaande programmagedeelte worden de gemiddelde waar-
den in de array GMEAN(10) opgeslagen.

In dit voorbeeld is NR constant zolang DO 30 uitgevoerd wordt.
De hulpvariabele S moet steeds nul gesteld worden voordat met een
nieuwe reeks data wordt begonnen. De opdracht S = .0 moet dus ge-
plaatst worden binnen DO 20 maar niet in DO 30.

De parameters van de DO-loops kunnen variabel gemaakt worden
waardoor het programma meer algemeen wordt. Stel dat we ook geïnte-
reseerd zijn in het algemeen gemiddelde, dan kan dat als volgt:

```

:
:
NR1 = 1    $    NRN = 10
ID1 = 1    $    IDN = 100
TOTAAL = .0
DO 20 NR = NR1, NRN
  S = .0
  DO 30 IDATA = ID1, IDN
    30 S = S + GW(IDATA,NR)
    TOTAAL = TOTAAL + S
  20 GMEAN(NR)= S/FLOAT(IDN)
  TMEAN = TOTAAL/FLOAT(IDN*NRN)
:
:

```

Het verdient aanbeveling bij het opbouwen van een programma dit stap voor stap te doen en eventueel af en toe te testen of hetgeen reeds bereikt werd goed werkt. Zo zou in het vorige probleem de volgende stap kunnen zijn rekening te houden met data waarop niet gemeten is en de waterstand de onbekend code heeft. Ook het aantal waarnemingen moet dan aan deze situatie worden aangepast.

Merk op dat niet IDATA als deler kan worden gebruikt. Na het beëindigen van DO-loop 30 heeft deze de waarde 101.

VII.6. DE IMPLIED DO

Een bijzondere vorm van de DO is de implied DO, of ingebedde (?) DO. Deze wordt op één enkele regel genoteerd en heeft slechts een beperkt aantal specifieke toepassingen. Hier wordt genoemd de implied DO in combinatie met READ en WRITE statements. De structuur van deze opdrachten is:

```
READ(2, 10) (X(I), I = 1, 10)
```

Voor een matrix wordt dit


```
READ(2, 10) ((X(I, J), I = 1, 5), J = 1, 5)
```

Overigens mogen de parameters ook hier integer variabelen zijn en mag een stapgrootte toegevoegd worden.

De implied DO mag met een gewone DO gecombineerd worden bijvoorbeeld om steeds één regel (record) van een matrix in te lezen en te verwerken bijvoorbeeld als we willen nagaan hoeveel elementen op een rij groter dan 100.0 zijn.

```
      :  
      :  
      DO 40 IRIJ = 1, 100  
        N100 = 0  
        READ(2, 7) (X(IRIJ,KOLOM),KOLOM = 1, 10)  
        DO 41 KOLOM = 1, 10  
          IF(X(IRIJ,KOLOM).GT.100.0)N100=N100 + 1  
        41 CONTINUE  
        NTAL(IRIJ) = N100  
      40 CONTINUE  
      :  
      :
```

VII.7. IF'S IN FLOWCHARTS

Over het nut van flowcharts of stroomdiagrammen wordt verschillend gedacht. Ze kunnen dienen om de structuur van een programma topografisch te verduidelijken en de aandacht op de belangrijkste onderdelen te vestigen. Voorbeelden worden gegeven door STOL (1979).

Bij gecompliceerde reeksen van voorwaardelijke keuzen (de IF's) kunnen stroomdiagrammen dienen om de onderlinge samenhang tussen de keuzen weer te geven alvorens tot het schrijven in Fortran over te gaan.

Het volgende voorbeeld is ontleend aan het hydrologisch model zoals dat door VAN BOHEEMEN (1979) voor het project 'het Kanaal' werd opgesteld. Slechts de voor dit voorbeeld van belang zijnde on-

derdelen worden vermeld.

Gevraagd wordt de afvoer/infiltratie Q te berekenen van een perceel onder de volgende voorwaarden:

- Grondwaterstand (G) lager dan of gelijk aan de slootwaterstand (S):

$Q = \text{functie 1}$

- Grondwaterstand (G) hoger dan de slootwaterstand (S), voor niet-gedraineerde percelen ($D = 0$):

$Q = \text{functie 2}$

- Grondwaterstand hoger dan de slootwaterstand, voor gedraineerde percelen:

- grondwaterstand lager dan of gelijk aan de draindiepte (D)

$Q = \text{functie 3}$

- grondwaterstand hoger dan de draindiepte

$Q = \text{functie 4}$

We nemen gemakshalve aan dat hoger dan geschreven kan worden als $>$ en lager dan als $<$. Het stroomschema ziet er dan uit zoals afgebeeld in fig. 3.

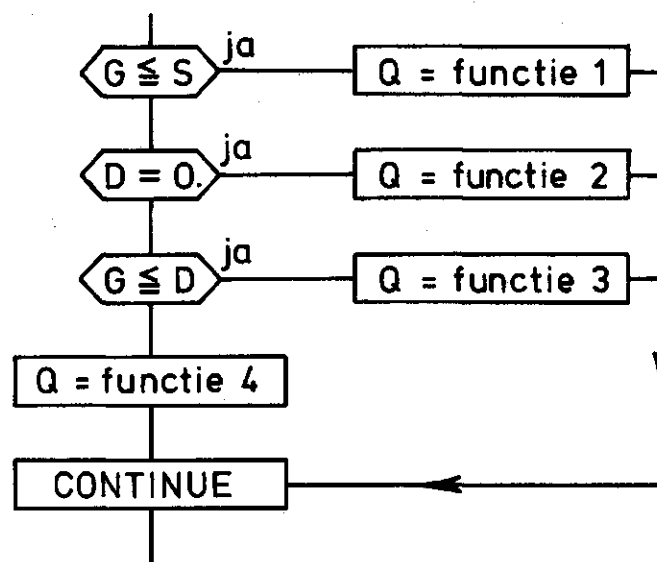


Fig. 3. Stroomschema ter verduidelijking van het gebruik van drie achtereenvolgende voorwaardelijke sprongopdrachten.

Elke keuze komt slechts éénmaal voor en er worden geen variabelen in de argumenten opnieuw gedefinieerd. De programma-tekst kan er dan als volgt uitzien:

```

.
.
.
Q = functie 4
      IF(G.LE.S)  Q = functie 1  $  IF(G.LE.S)  GO TO 10
      IF(D.EQ..0) Q = functie 2  $  IF(D.EQ..0) GO TO 10
      IF(G.LE.D)  Q = functie 3  $  IF(G.LE.D)  GO TO 10
10 CONTINUE
.
.
.

```

Door de benodigde functies apart in een functie-statement (zie hoofdstuk IX.4) te definiëren bereikt men bovendien dat de te gebruiken functies los komen te staan van de besturing. Eventuele wij-

zigen in de functies beïnvloeden de rest van het programma dan niet.

VII.8. TOEPASSINGEN

In 'Berekening en bevoeiing in Nederland in het zeer droge jaar 1976' (DE WILDE en LINTHORST, 1978) komt op pag. 9 een tabel voor waaraan we ontleen:

Watergiften in mm in 1976, berekening op grasland

Groningen	172	Noordholland	200
Friesland	200	Zuid-Holland	192
Drenthe	215	Zeeland	376
Overijssel	304	Noord-Brabant	484
Gelderland	339	Limburg	518
Utrecht	353	IJsselmeerpolders	146

Gevraagd wordt een programma te schrijven dat afturft hoe vaak deze giften behoren tot klassen van 100-199, 200-299, 300-399, ... enz. mm.

Een mogelijke opzet voor een dergelijk programma is gegeven in voorbeeld 8.

Voorbeeld 8. Het bepalen van de frequenties waarmee waarnemingen in bepaalde klassen vallen.

```
CONNECT,ICW
RETURN,OUTPUT,TALB,T
COPYBF,ICW,T
```

```
PROGRAM TALLY(OUTPUT,TAPE3=OUTPUT)
C
C   INTEGER X
C   LOGICAL KLASSE
C
C   DIMENSION X(20),NTAL(10)
C
C   DATA X / 172, 200, 215, 304, 339, 353,
C   *       200, 192, 376, 484, 518, 146, 8*0 /
C   DATA NTAL /10*0/
C
C   WRITE(3,1)
C   1 FORMAT(////1H "AANTAL, BOVENGRENS"/)
C
C   DO 20 I=1,20
C     IF(X(I).EQ.0) GO TO 30
C     DO 15 K=1,20
C       KLASSE=(K-1)*100.LE.X(I).AND. X(I).LT.(K*100)
C       IF(KLASSE) NTAL(K)=NTAL(K)+1
C     15 IF(KLASSE) GO TO 20
C   20 CONTINUE
C
C   30 WRITE(3,40) (NTAL(K/100),K,K=100,1000,100)
C   40 FORMAT( 1H I4,I11)
C   END
```

```
ZEOP
REWIND,T
FTN,I=T,L=0,B=TALB
CONNECT,OUTPUT
TALB
EX
```

```
COMMAND-
COMMAND-
COMMAND-
COMMAND-
.340 CP SECONDS COMPILATION TIME
COMMAND-
COMMAND-
```

AANTAL, BOVENGRENS

0	100
3	200
3	300
4	400
1	500
1	600
0	700
0	800
0	900
0	1000

END TALLY

.034 CP SECONDS EXECUTION TIME

In dit programma worden de klassen (door hun eenvoudig karakter) gecreëerd in DO-lus 15, op basis van de index K. Voor elke waarde van K is $(K - 1) \cdot 100$ de ondergrens en $K \cdot 100$ de bovengrens van de klasse. Getest wordt achtereenvolgens of wáár is dat

$$100 \leq X < 200$$

of dat

$$200 \leq X < 300$$

enz.

Dit wordt afgecheckt in DO-loop 15. Wanneer de juiste klasse is gevonden wordt het aantal tellingen in die klasse met één verhoogd en wordt de loop (voortijdig) verlaten.

Uiteraard kan dit programma meer algemeen gemaakt worden door bijvoorbeeld de klassen afzonderlijk te definiëren en niet van K te laten afhangen.

VIII

I N P U T E N O U T P U T

VIII.1. INLEIDING

De input en output van een programma heeft betrekking op de informatie-uitwisseling tussen media buiten de computer en de computer zelf.

De opdrachten hiertoe zijn de READ, om informatie in het kerngeheugen van de computer op te slaan en zo voor het programma bereikbaar te maken, en WRITE om informatie weer in een voor de mens leesbare vorm te brengen.

De READ en WRITE opdrachten bestaan eigenlijk uit twee afzonderlijke delen.

1. De feitelijke READ en WRITE, waarbij in een lijstje (list) opgegeven wordt welke variabelen ingelezen moeten worden, respectievelijk uitgeschreven moeten worden. Bijvoorbeeld: X, Y, A(1), BETHA
2. De organisatie en administratie van het lezen en schrijven of wel het FORMAT dat de layout bevat van de in te lezen gegevens, respectievelijk de layout verzorgt van de te printen gegevens.

De READ en de WRITE functies hebben een argument met twee parameters waarvan de eerste (die ook variabele mag zijn) verwijst naar de file waarmee informatie uitgewisseld wordt en de tweede naar het toe te passen FORMAT. Voorbeeld:

```
READ(2, 5)X, Y, A(1), BETHA
5  FORMAT(4 F10.1)
```

We zullen eerst de FORMAT's bespreken.

VIII.2. FORMAT ALGEMEEN

Het FORMAT dient om aan te geven wat de layout is van de gegevens die moeten worden gelezen en wat de layout zal zijn van de resultaten die moeten worden geprint.

De algemene vorm van het FORMAT-statement is:

40 FORMAT (---specificatie---)

waarin 40 een zelf te verzinnen labelnummer is, waarmee in een lees- of schrijfpdracht naar het FORMAT kan worden verwezen. Zie hiervoor de eerder gegeven programma-voorbeelden. De ---specificatie--- moet door de programmeur worden opgegeven. Hiervoor is het volgende de noodzakelijke voorkennis.

- Bij het inlezen van gegevens moet bekend zijn op welke wijze het ponsdocument ingedeeld is, respectievelijk hoe de gegevens op ponskaart staan of op een file gezet zijn. Men kan zich het inlezen als volgt voorstellen:

De computer leest steeds één volledige kaart tegelijk en beschikt dan over 80 karakters aan informatie. De programmeur moet dan opgeven wáár de van belang zijnde getallen beginnen, hoeveel posities die innemen, waar eventueel de komma geplaatst moet worden, enz.

- Bij het printen van resultaten moet bekend zijn op welke wijze de tabellen, overzichten, toelichtingen en dergelijke ingedeeld moeten worden. Bijvoorbeeld: of op een nieuwe pagina moet worden begonnen, hoeveel decimalen men wenst, hoeveel ruimte tussen de kolommen moet worden opengelaten, wanneer er spaties moeten komen, hoe de toelichtende teksten over een regel moet worden verdeeld, enz.

VIII.2.1. F O R M A T v o o r i n p u t

Voor de input van gegevens via beeldscherm respectievelijk via ponskaarten, zijn 80 posities beschikbaar. Het totaal aan gespecificeerde kolommen in een FORMAT mag bij deze wijze van input dit bedrag dus niet overschrijden.

De volgende specificaties kunnen worden gebruikt.

- | | |
|-----|---|
| nX | n kolommen overslaan, bijvoorbeeld 10X (default: bX = 1X) |
| In | er moet een integer constante worden ingelezen welke in de eerstvolgende n kolommen (right justified) voorkomt, bijvoorbeeld I7 |
| kIn | er moeten k integer constanten worden ingelezen, elk |

	(right justified) voorkomend in n kolommen ($k \times n$ -eerstvolgende kolommen), bijvoorbeeld 5I10
Fn.m	er moet een floating point constante worden ingelezen, voorkomend in de eerstvolgende n kolommen (right justified) en er moeten m cijfers achter de komma worden geplaatst. (Zie par. 2.2, pag. 107)
kFn.m	er moeten k floating point constanten worden ingelezen, elk - right justified - voorkomend in n kolommen. Er moeten m cijfers achter de komma worden geplaatst ($k \times n$ -eerstvolgende kolommen), bijvoorbeeld 3F10.2
An	er moeten n Hollerith tekens tekst worden ingelezen ($1 \leq n \leq 10$), bijvoorbeeld A8
kAn	er moeten k maal n Hollerith tekens tekst worden ingelezen, bijvoorbeeld 12A6
,	scheidingsteken tussen de specificaties wanneer er gevaar voor verwarring bestaat. Toegestaan is 3X5I3 maar niet I2F6.2 dat moet zijn I2, F6.2
/	de volgende kaart (regel) moet worden gelezen, bijvoorbeeld I10, 5F10.0/I10, 5F10.0
k(/)	sla k kaarten (regels) over
Tn	begin te lezen met kolom n (tabellarische specificatie), bijvoorbeeld T70
"bbb" of *bbb*	lees net zoveel Hollerith tekst van de kaart als met het aantal blank gelaten posities b overeenkomt. Deze tekst blijft dan in het FORMAT aanwezig. Dit is een mogelijkheid een stuk tekst als variabele in te voeren, bijvoorbeeld de naam van een project en dergelijke. Steeds in de eerste kolom van de in te lezen een <u>blank</u> opnemen (zie bij carriage control, pag. 108, 109)
k(In ₁ , Fn ₂ .m)	er moeten k groepen ingelezen worden, elk bestaand uit een integer in de eerstvolgend n ₁ kolommen en een floating point getal in de daaropvolgende n ₂ kolommen. Elk floating point getal heeft m cijfers achter de komma, bijvoorbeeld 3(I5, 2X, F7.1)

Op de laatst aangegeven wijze kunnen meer combinaties van mogelijkheden gemaakt worden.

Voorbeeld van invoergegevens:

Kaart	1										2										3									
no	1	2	3	4	5	6	7	8	9	.	1	2	3	4	5	6	7	8	9	.	1	2	3	4	5	6	7	8	9	.
1				N	I	E	U	W	E		U	I	T	K	O	M	S	T	E	N		1	9	1	2					
2				1	5						3		1	3	4						6			9	8	7	.	6	3	2
3		1	2	3	3	2	1	9	8	7	7	8	9																	
4				G	E	E	N				H	A	V	E	R															

te lezen met de volgende FORMAT's, achtereenvolgens:

```
10 FORMAT("TEKST KOMT HIER DUS-----")
15 FORMAT(2I5, F5.0, 2X, I3, F10.5)
16 FORMAT(3I3, F3.5)
20 FORMAT(A10)
```

Dit geeft aanleiding tot opslag in het geheugen van de volgende informatie:

```
bb NIEUWE UITKOMSTEN 1912
15
3
134.
6
987.632      (zie par. 2.2, pag. 107)
123
321
987
0.00789      ( " " " )
GEEN HAVER
```

Later wordt nog besproken hoe de getalswaarden aan de relevante variabelen moeten worden toegekend. Momenteel zijn de volgende opmerkingen van belang.

- bij het programmeren moet men met veel zorg de administratie en organisatie van de gegevens uitvoeren,
- ponsdocumenten moeten zo systematisch mogelijk worden ingevuld,

- dat vereenvoudigt de FORMAT-specificatie,
- men is geheel vrij in de wijze van indelen van het ponsdocument; de gegevens mogen zelfs zonder tussenruimte worden geponst. Dit kan men doen om ruimte te besparen, maar wel moet bedacht worden dat bij het (ter controle) listen van zo'n cijfermassa een onleesbare hoeveelheid aan informatie ontstaat.

Opgemerkt wordt nog dat in het voorbeeld het verschil tussen FORMAT 10 en FORMAT 20 is dat in nr 10 de ingelezen tekst "NIEUWE UITKOMSTEN 1912" in het argument opgenomen wordt en niet meer bereikbaar is. Wél kan de tekst weer uitgeschreven worden. In nr 20 kan de tekst ingelezen worden in een variabele (zie par. 3) bijvoorbeeld ISERWAT. Stel dat de oppervlakte aan haver opgeslagen wordt in HAVEROP. Dan kan bijvoorbeeld volgen

```
IF(HAVEROP.GT..0) ISERWAT=10HTOCH HAVER
```

welke 'waarde' van ISERWAT weer naar output geschreven kan worden.

VIII.2.2. E n k e l e b i j z o n d e r e g e v a l l e n

Het aantal in te lezen Hollerith karakters is bij de specificaties A10 en 2A5 of (2A4, A2) even groot. Het verschil is dat in het eerste geval 10 Hollerith karakters op één aanroepbare plaats in het geheugen worden opgeslagen, in het tweede geval op twee en in het laatste geval op 3.

Het is mogelijk dat een getal in input een komma-aanduiding heeft en dat dit getal met een afwijkende komma-aanduiding in het FORMAT wordt ingelezen. De volgende regels zijn nu van kracht:

Voorbeelden:

Hieruit blijkt dat bij het intikken van gegevens mēt komma (decimal point) de plaats binnen het veld niet van belang is. Dit geeft de mogelijkheid gegevens met komma left justified op kaart te ponsen, wat gemakkelijker maar minder overzichtelijk is.

Voor het printen van de output van een programma zijn 132 posities op de printer beschikbaar. Dit is de regelbreedte die thans vrij algemeen op de output apparaten wordt toegepast en ook geldt op de voor het ICW toegankelijke line-printers van de Cyber- en PDP-computers.

Dat het beeldscherm van de terminal slechts regels van 80 posities kent is geen bezwaar. De terminal is zo ingericht dat de rest van een regel (maximaal 52 posities dus) op de volgende regel wordt afgebeeld en dus niet verloren gaat.

De FORMAT-specificaties voor de output komen overeen met die voor de input maar mogen in totaal dus 132 posities bestrijken. Het belangrijkste verschil is dat bij de output altijd het eerste teken

voor carriage control wordt gebruikt. Dit moet dan een Hollerith teken zijn. Reserveert men dit niet dan wordt het eerst beschikbare teken door de computer als zodanig geïnterpreteerd wat de nadelen heeft dat

- de printer-besturing afhangt van het teken dat toevallig op de eerste positie voorkomt,
- het eerste teken om deze reden verloren gaat.

Men kan aan dit nadeel tegemoet komen door de output-file met een 'shift' (één positie naar rechts opgeschoven) naar de terminal te kopiëren met het commando (geen Fortran, maar besturing):

COPYSBF, output, icwΔ

Dit is overigens niet de normale procedure, normaal is een output FORMAT steeds met de specificatie 1Hb te beginnen.

De volgende specificaties kunnen worden gebruikt.

1Hi Elke output specificatie moet met één Hollerith karakter beginnen voor carriage control.

i = b (blank), begin op eerstvolgende regel,

i = l, begin bovenaan een nieuwe pagina,

i = +, print op de vóórstaande regel,

i = 0, sla een regel over alvorens te printen.

Bijvoorbeeld: 1H, 1Hl, 1H+

Elk ander teken wordt verwerkt als b

nX sla n posities over, bijvoorbeeld 60X

In

kIn

Fn.m

An

kAn

,

als bij input

/1Hb ga verder op de volgende regel, bijvoorbeeld I3,/1H I5

//1Hb sla een regel over en ga verder op de dan volgende regel.

Ook: 2(/)1Hb

/.../1Hb sla eerst /... regels over (stippen staan voor slashes)

- Tn als bij lezen, maar de eerste positie wordt geïnterpreteerd als 1Hb voor carriage control
- En.m er moet een real getal in de exponentiële versie worden geprint in de eerstvolgende n kolommen (right justified) en er moeten m significante cijfers worden geprint, voorbeeld E15.3
- kEn.m er moeten k real getallen in de exponentiële versie worden geprint in de eerstvolgende k x n kolommen (right justified) en er moeten van elk m significante cijfers worden geprint, bijvoorbeeld 6E15.5
- "..." of print de tussen " " respectievelijk tussen * * opgegeven
 ... tekst, bijvoorbeeld "PROJECT INFILTRATIE DROOG-LANDEN".

Een paar aanvullende opmerkingen moeten worden gemaakt.

F-versie

Bij de opgave Fn.m, bijvoorbeeld F10.3, is de output bijvoorbeeld

```
1 2 3 4 5 6 7 8 9 0      (posities)
- 1 2 . 1 2 3
```

waaruit blijkt dat één positie voor het teken (bij negatieve getallen) en één positie voor de decimale punt benodigd zijn zodat altijd genomen moet worden $n \geq (m + 2 + c)$ waarin c het aantal cijfers vóór de komma voorstelt. Men moet dus op de hoogte zijn van de orde van grootte van het getal. Zo zal bijvoorbeeld 1234567.345 niet in het FORMAT passen en aanleiding zijn tot een foutindicatie. Er verschijnt dan ***** in de output.

E-versie

Bij de opgave En.m, bijvoorbeeld E10.3, is de output bijvoorbeeld

```
1 2 3 4 5 6 7 8 9 0      (posities)
- . 1 2 1 E + 0 2
```

waaruit blijkt dat één positie voor het teken (bij negatieve getallen) en één positie voor de decimale punt en vier posities voor het vermelden van de exponent benodigd zijn. Men moet dus altijd nemen

$n \geq (m + 6)$. Men behoeft geen kennis te bezitten van de orde van grootte van het getal. Het getal 1234567.345 wordt geprint als

```
1 2 3 4 5 6 7 8 9 0      (posities)
. 1 2 3 E + 0 7
```

wat wél in het FORMAT past.

Voorbeeld:

```
10 FORMAT(1H1///3X"EERSTE PROEF"///)
20 FORMAT(1H "RIJ NR="I3,"bKOLOM NR="I3,/1H
  "BEREGENING GIFT W="E14.5"bM**3"/1H
  "GEMIDDELDE      Z="F10.2"bMM")
21 FORMAT(1H "+"8("-")"+")
25 FORMAT(1H+30X"-----/")
```

wat aanleiding geeft tot de volgende output:

----- (scheurlijn v.h. output
formulier)

EERSTE PROEF

```
RIJ NR=  2  KOLOM NR=  5
BEREGENING  GIFT W=    .12345E+01 M  3
GEMIDDELDE   Z=      3.12 MM
+-----+          -----/
```

Belangrijk 1

Bij het copiëren van een file naar de terminal wordt het eerste karakter als carriage opgevat. Dit betekent dat ongewild besturing van de printer kan optreden. Stel dat een file bestaat uit waterstanden gemeten op diverse data bijvoorbeeld:

1963	01	20	110	75	30
1963	02	16	105	70	36
1963	03				
1963			enz.		
1963					

Alle getallen 1 in de eerste positie zullen bij copiëren naar de lineprinter opgevat worden als papiertransport naar een nieuwe pagina. De output bestaat dus uit 5 vellen met op de bovenste regel van pagina 1

963 01 20 110 75 30

enz. Dit kan worden vermeden door te copiëren met een shifted binary file (COPYSBF-commando). De informatie wordt dan één positie naar rechts opgeschoven. De vrijgekomen open eerste positie betekent dan: print op volgende regel.

Dit voorbeeld illustreert dat men met zorg files moet samenstellen en beheren.

Belangrijk 2

We zagen reeds dat bijvoorbeeld in een output FORMAT de aanwijzing voor carriage control 1H* hetzelfde is als 1Hb (volgende regel). In dit geval staat in de eerste positie op de betrokken file op deze regel een *. Dit kan worden toegepast wanneer de output file vanaf die regel weer input file voor een volgend program is. Men kan deze file dan lezen tot * gevonden is. Bijvoorbeeld met 5 READ(1, 2) STAR met 2 FORMAT(A1) en dan IF(STAR.NE.1H*) GO TO 5.

Belangrijk 3

FORMAT fouten worden niet altijd door de compiler ontdekt. Het kan dan voorkomen dat er tijdens de executie een foutmelding komt. Meestal wordt hierbij opgegeven dat het een FORMAT fout betreft, wat de aard van de fout is, in welk programma de fout voorkomt, en op welke regel in dat programma. Tevens volgt een output van de foute regel met indicatie op welke plaats de output stopte. Bijvoorbeeld

```
FORMAT (1H I5F4.06(1X3F6.1)
1234567890.'.....123456 enz.
```

Hieruit valt te zien dat I5F4.0 nog correct is vertaald. (Toch is het beter te schrijven I5, F4.0.) De fout treedt op bij de 06 dit moet zijn F4.0, 6(1X3F6.1) met een, als scheidingsteken.

VIII.3. READ EN WRITE

READ- en WRITE-statements dienen om informatie in het geheugen van de computer in te lezen, respectievelijk om in het geheugen aanwezige informatie uit te schrijven. Bij deze laatste handeling blijft de informatie in het geheugen bewaard voor verder gebruik.

Het organiseren van de invoer los van het programma heeft een aantal voordelen. We noemen:

- Het programma kan los van het gegevenbestand worden geschreven en ontwikkeld. Zelfs hoeft men het juiste FORMAT niet te kennen, dit kan later nog gewijzigd worden.
- Het programma kan met een gering aantal gegevens getest worden, later kan men dit aantal gemakkelijk opvoeren.
- Nieuwe gegevens kunnen in het gegevenbestand opgenomen worden zonder dat daarvoor wijzigingen in het programma zelf nodig zijn.
- Andere gegevenbestanden kunnen zonder meer verwerkt worden.
- Het programma kan op een binaire file in vertaalde versie geplaatst worden. Bij gebruik met nieuwe series gegevens hoeft het programma dan niet opnieuw vertaald te worden.

De algemene gedaante van deze statements is:

READ (u, l) list

WRITE(u, l) list

waarin

u = unit nummer van de file waarop de informatie staat (voor READ), respectievelijk waarop de informatie geplaatst moet worden (voor WRITE). Hiervoor moet u overeenkomen met het cijfer achter de TAPE u in de program-kaart

l = label dat verwijst naar het te gebruiken FORMAT

list = lijst van variabelen die aangeeft aan welke geheugenplaatsen de ingelezen informatie achtereenvolgens moet worden toegekend (bij READ), lijst van variabelen die aangeeft van welke geheugenplaatsen de te printen informatie achtereenvolgens moet worden onttrokken (bij WRITE)

Wanneer we aannemen dat op de file BLAUW de volgende getallen staan

1	2	3	4	5	6	7	8	9	0	(positie)
		1	6				3	.	9	

dat bij deze getallen 30 moet worden opgeteld en dat de uitkomsten geschreven moet worden naar de file WIT, dan kan een programma daarvoor er als volgt uitzien:

	PROGRAM KLEUR (BLAUW, WIT, TAPE5=BLAUW, TAPE1=WIT)
	READ(5, 10) N, F
10	FORMAT(I4, F6.1)
	N30 = N + 30
	F30 = F + 30
	WRITE(1, 6) N30, F30
6	FORMAT(1H "N30="I5, "bb F30=" F10.1)
	END

Op de file WIT staat nu

	1	2	3
	123456789.	123456789.	123456789.
N30=	46	F30=	33.9

Merk op dat de eerste positie van de output blank is. Deze wordt gebruikt als carriage control bij afdrucken van de informatie op de printer, zodat dan de N in de eerste positie staat.

Men is geheel vrij het tijdstip te kiezen waarop gegevens moeten worden ingelezen, mits dit is vóórdat de variabelen genoemd in de list in het rechterlid van een vervangingsstatement voorkomen.

Uit het voorgaande blijkt dat bij het inrichten van een ponsdocument voor de input, het belangrijkste is dat dat systematisch gebeurt. Is een bepaalde kolom-indeling ontworpen dan moet men deze blijven aanhouden.

Zijn toevallig de kolommen van twee variabelen op het ponsdocument verwisseld dan is de eenvoudigste oplossing de READ-list en het

FORMAT aan te passen, bijvoorbeeld (zie vorig voorbeeld):

```
READ (5, 10) F, N
10 FORMAT (F6.1, I4)
```

Dit levert minder fouten dan een ponsbureau vragen de desbetreffende kolommen in omgekeerde volgorde te ponsen.

Wat de output betreft verdient het aanbeveling de WIRTE-opdrachten zoveel mogelijk over een programma en de subroutines te verdelen of te verspreiden. Dit heeft het voordeel dat in de testfase bij het optreden van fouten de plaatsen waar de fouten kunnen zijn opgetreden gemakkelijker kunnen worden gelocaliseerd door na te gaan waar nog wel en waar niet meer goede resultaten werden geprint.

Formats mogen op elke plaats in een programma worden opgenomen en dus niet persé na de betreffende READ of WRITE statement. Ze mogen ook vóór het READ of WRITE statement komen. Verschillende READ's of WRITE's mogen naar één Format verwijzen, zelfs wanneer in de list andere variabelen voorkomen, mits het type van de variabelen overeenstemt met de specificaties in de list.

Bij het nalezen van een Fortran tekst is het plezieriger de Format's direct bij de READ of WRITE's terug te vinden. Vooral wanneer het Format enige tekst bevat wordt de bedoeling van een programma onderdeel dan spoedig duidelijker.

Een schematisch voorbeeld is nog:

```
10 FORMAT---
:
20 FORMAT---
:
:   READ(5, 10) X, Y, MAX
:
:   READ(5, 10) SAMPLE
:
:   WRITE(9, 20)
:
:   WRITE(9, 25) U, V $ WRITE(9, 26) SOM
25 FORMAT(1H1"UITKOMSTEN U EN V" 2F15.3)
26 FORMAT(1H "SOM="E15.5/)
```

Lees en schrijfoopdrachten worden beëindigd wanneer de list is uitgeput. Dit kan betekenen dat 1) het FORMAT niet wordt afgewerkt, 2) het FORMAT precies past, 3) het FORMAT wordt herhaald.

Voorbeeld van het eerste geval

```
WRITE(3, 5) K, X
5 FORMAT(1H I5,2F3.0"bbeINDE DEEL 1"/)
```

Het gedeelte tekst en de spatie (/) komen nu niet op de output file omdat van het FORMAT alleen I5 en één F3.0 gebruikt wordt.

Met

```
WRITE(3, 5) K, X, Y
```

past het FORMAT precies. Dit zal de algemene gang van zaken zijn.

Met

```
WRITE(3, 5) K, X, Y, K, X, Z
```

wordt verkregen (bijvoorbeeld)

```
12345 .6 .7 EINDE DEEL 1
12345 .6 .9 EINDE DEEL 1
```

waarmee de tekst een aantal extra keren wordt geprint, wat veelal niet de bedoeling zal zijn.

Met

```
WRITE(3, 5) K, X, Y, I, J
```

ontstaat een foutmelding: de J (integer) correspondeert nu met een F specificatie, nadat het Format is herhaald, wat een onjuist gebruik van het FORMAT is.

Eenzelfde FORMAT mag vaker gebruikt worden, op geheel verschillende plaatsen in een programma. Een FORMAT mag ook zowel vóór READ als WRITE statements worden geplaatst. Er moet enige zorg betracht worden aangaande het gebruik van de 1H carriage control. Deze positie moet bij het lezen worden meegeteld.

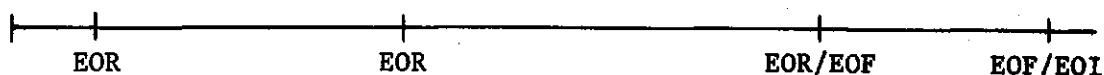
VIII.4. FILES EN RECORDS

De eenheden waarin informatie wordt opgeslagen in de computer zijn files en records. Een file kan zijn onderverdeeld in records.

Bij het programmeren heeft men hiermee niet zoveel te maken. Het zal echter blijken dat bij het kopiëren van output naar de terminal soms een boodschap wordt toegevoegd met aanwijzingen omtrent het gedeelte van de file dat gecopieerd is.

Voorbeeld:

De opbouw van een file kan zijn:



EOR = end-of-record

EOF = end-of-file

EOI = end-of-information

De totale file bestaat uit 2 files en de eerste file bestaat uit 3 records.

Output van een job wordt naar een file geschreven in de vorm van één record. Men moet dus met de volgende commando's werken (stel dat de output staat op local file WIT):

REWIND, WITA

COPYBR, WIT, ICWA

waarmee het eerste record wordt gecopieerd.

Zou men geven

REWIND, WITA

COPYBF, WIT, ICWA

dan verschijnt na de output de mededeling

EOI ENCOUNTERED AFTER COPY OF FILE

0, RECORD 1

Dit is geen foutmelding maar een aanwijzing dat men er attent op

moet zijn dat men een file wil copiëren, maar dat alleen records aanwezig zijn.

Onder gebruikmaking van de COPYBF kan men deze vermelding vermijden door de output als file af te sluiten. Dit kan in Fortran met

```
ENDFILE u
```

waarin u het unitnummer van de file is. Uiteraard moet dit aan het slot van een bewerking gedaan worden wanneer geen bijbehorende output meer volgt. Voorbeeld:

```
PROGRAM CLOSE(WIT,TAPE12 = WIT)
:
:
WRITE(12, 6) X, Y
6 FORMAT(---)
:
:
ENDFILE 12
:
:
END
```

IX

HOOFD- EN SUBPROGRAMMA'S

IX.1. INLEIDING

In hoofdstuk II werd reeds uiteengezet dat het voordelen geeft en dus aanbeveling verdient een programma in duidelijk te scheiden onderdelen op te splitsen.

Dit kan men bereiken door groepen bij elkaar behorende statements van een eigen opschrift (COMMENT) te voorzien, of bepaalde programma onderdelen in de layout meer te laten inspringen.

Daarnaast zijn er mogelijkheden om afzonderlijke procedures of routines te gebruiken en deze al naar believen op te roepen. Deze routines kan men ontwerpen met een argument zodat bij het oproepen ervan de dan geldende waarden aan de variabelen in het argument worden toegekend. (Zie hoofdstuk I.4).

Of en wanneer men routines moet schrijven, en welke omvang deze moeten hebben hangt geheel van het persoonlijk inzicht van de programmeur af. De voordelen van het afzonderlijk kunnen schrijven en testen van subroutines moet men afwegen tegen enige moeite die men moet getroosten om de onderlinge aansluiting goed te programmeren. Veelal overwegen bij grote programma's de voordelen, te meer waar veranderingen in aparte routines weer makkelijker zijn aan te brengen dan in grote programma's die als één geheel zijn geschreven.

Bij routines die met argument voorkomen is de correspondentie tussen de argumenten (werkelijk te gebruiken variabelen en dummy variabelen) naar volgorde van genoemde variabelen, naar aantal (speciaal van belang bij arrays) en naar type. Deze regels moeten correct worden toegepast. Niet altijd zal bij slordigheid een syntax error ontstaan terwijl de uitkomsten toch onjuist zijn en men niet gewaarschuwd wordt.

Dummy argumenten mogen niet voorkomen in COMMON- en DATA-statements van de betreffende routine en dus als het ware niet tevens 'algemeen' gedefinieerd worden (zie hoofdstuk X).

Routines kan men nog onderscheiden in onderdelen van een programma en apart geschreven programma's. In de naamgeving van programma's zullen we nu onderscheid maken tussen Hoofdprogramma's en Subprogramma's. Subprogramma's zijn zelfstandige, los van het hoofdpro-

programma geschreven programma's met eigen namen van variabelen (die overigens dezelfde mogen zijn als in het hoofdprogramma) en eigen labelnummers.

In de volgende paragrafen zal op de onderlinge relatie tussen de verschillende soorten programma's worden ingegaan. Eerst echter wordt nog de specifieke rol van het hoofdprogramma in dit verband, behandeld.

IX.2. HOOFDPROGRAMMA'S

Elk Fortran-programma moet beginnen met een PROGRAM-kaart waarop de door de programmeur te verzinnen naam van het programma wordt opgegeven. Op dezelfde kaart moeten de te gebruiken files worden gedefinieerd ten behoeve van de informatie overdracht 'naar buiten'.

Vrij veel wordt gebruik gemaakt van een statement met de volgende specificaties,

```
PROGRAM club(input, output, TAPEi=input, TAPEj=output)
```

waarin

club = de verzonden naam van het programma

input = file waarop de door het programma te lezen informatie voorkomt

output = file waarop de door het programma te produceren informatie moet worden geschreven

TAPEi = unit i, door de programmeur te specificeren naar keuze, echter $1 \leq i \leq 99$

TAPEj = unit j, door de programmeur te specificeren naar keuze, echter $1 \leq j \leq 99$ ($i \neq j$)

De correspondentie 'naar buiten' is dan gedefinieerd met

```
READ(i, 11)
```

en

```
WRITE(j, 12)
```

waarin l_1 en l_2 FORMAT labelnummers zijn.

In het argument van het PROGRAM-statement moeten alle te gebruiken file-namen het eerst worden genoemd en volgen de gelijkstellingen daarna.

Stel dat men de invoergegevens heeft geplaatst op een file DATA en dat men de output wenst te schrijven op een file result, dan wordt de instructie:

```
PROGRAM NIEW(DATA,RESULT,TAPE2=DATA,TAPE3=RESULT)
```

waarbij unit 2 in de READ en unit 3 in de WRITE moet voorkomen.

In het programma zelf mag het unitnummer u variabel gekozen worden met $1 \leq u \leq 99$. Voorwaarde is dat voor elke waarde van u een tape unit in de PROGRAMkaart wordt vermeld.

Deze mogelijkheid kan dienen om afhankelijk van de resultaten relevante uitkomsten op verschillende files bijeen te brengen. Bijvoorbeeld

```
PROGRAM KEUZE(DATA,RESULT1,RESULT2,  
* TAPE2=DATA,TAPE5=RESULT1,TAPE8=RESULT2)  
:  
:  
K = 5  
WRITE(K, 10)X  
10 FORMAT(1H F10.5)  
:  
:  
IF(X.LT.0) K = 8  
WRITE(K, 10) X $ K = 5  
:  
:
```

waarmee dus alle uitkomsten voor X geschreven worden op file RESULT1 en de negatieve uitkomsten ook nog gezamenlijk op file RESULT2.

De tekst van een programma moet altijd eindigen met het statement

```
END
```

op de laatste regel. Deze opdracht is nodig om de vertaler van het programma de aanwijzing te geven dat er geen regels tekst meer komen en de vertaalfase kan worden afgesloten en alle onderlinge verwijzingen op hun volledigheid kunnen worden gecontroleerd.

Vaak zal het voorkomen (zie de reeds eerder gegeven voorbeelden) dat de feitelijke bewerkingen op een andere plaats dan de laatste regel eindigen. Dit moet dan ter plaatse worden aangegeven met

STOP of STOP n of STOP "text"

waarin n een octaal getal is van maximaal 5 cijfers, en tekst een string van 1-70 characters bijvoorbeeld: STOP"WAARDE VAN X TE GROOT". De toevoeging van n is van gemak wanneer een programma op verschillende plaatsen kan worden beëindigd, bijvoorbeeld wanneer een controle op inputgegevens is ingebouwd. De waarde van n wordt op het beeldscherm afgebeeld zodat men meteen een indicatie heeft bij welke bewerking respectievelijk via welke route het programma is gestopt. Voorbeeld:

```
IF(X.LT..0) STOP 505
```

Wanneer een route door het programma inderdaad op de laatste regel eindigt, dan wordt het END-statement tevens als STOP opgevat, en is een apart STOP-statement niet vereist.

Netter is nog bij een alternatieve STOP tevens een boodschap te printen zodat men af kan lezen om welke reden het programma voortijdig is beëindigd, bijvoorbeeld:

```

:
:
IF(X.LT..0) WRITE(3, 15) X
15 FORMAT(//1H"X="E14.5"bbbSTOP 505")
IF(X.LT..0) STOP 505
:
:
```

Een file waarvan gegevens zullen worden gelezen moet op zijn beginpunt staan. Via de terminal kan dit worden bereikt met het commando REWIND, inputΔ respectievelijk REWIND,DATAΔ. Wanneer in het

programma de file na gelezen te zijn opnieuw moet worden gelezen dan moet het unitnummer van de te lezen file worden vermeld dus, in Fortran

REWIND 2

de file staat dan weer op zijn beginpunt.

Belangrijk

Foutmeldingen tijdens het vertalen geproduceerd kunnen worden opgehaald met het commando ERRORS,FTNA. Foutmeldingen tijdens de executie worden door het systeem geschreven naar een file met naam OUTPUT. Zeker in de testfase van een programma verdient het daarom aanbeveling steeds een file OUTPUT in de program-kaart te definiëren. Dit voorkomt dat foutmeldingen verloren gaan of niet volledig zijn.

IX.3. ROUTINES

Onder routines verstaan we programma's of programma-onderdelen die afzonderlijk geschreven kunnen worden en later weer in een programma kunnen worden opgeroepen om een onderdeel van een nieuw probleem te helpen oplossen. Routines kunnen zowel vaste routines zijn die zich reeds in het computersysteem bevinden en voor iedere gebruiker beschikbaar zijn, als door de gebruiker geschreven routines die bij zijn programma of programma-pakket behoren.

De belangrijkste soorten routines kunnen als volgt worden ingedeeld:

Hulpmiddel	Soort	Te schrijven als
Functie routine	Built-in functies	(zijn beschikbaar)
	Statement functies	Statement
	external functies	Subprogramma
Procedure routine	Subroutines	

Built in functions zijn reeds behandeld in Hoofdstuk VI.3. Ze zijn van het type SIN, COS, ALOG, SQRT enz. Deze zijn oproepbaar in elke soort routine en in het hoofdprogramma door het noemen van de functie en de functie een argument mee te geven (zie tabel 4, pag.73).

De overige routines worden in de volgende paragrafen besproken. Een schematisch overzicht staat in tabel 5.

Argumenten van routines mogen expressies zijn maar ook namen van arrays zijn toegestaan. In elk van de genoemde routines mogen niet meer dan 63 dummy-argumenten voorkomen. De External functies en subroutines moeten tegelijk met het hoofdprogramma geladen worden alvorens tot executie kan worden overgegaan. Het eenvoudigst is dit door alle programma's op één file te plaatsen. Zie voorbeelden 10 en 11 en ICW-nota 1117.

Subroutines kunnen zelf ook weer subroutines oproepen (zie tabel 5). Om deze reden zal er sprake zijn van het 'roepende' programma en van het 'opgeroepen' programma wat algemener is in dit verband dan Hoofdprogramma en Subprogramma's.

IX.4. STATEMENT FUNCTIONS

Deze routine komt de in de wiskunde in gebruik zijnde vormen het dichtst nabij. Voorbeelden:

$$y = f(x) = ax + b$$

Indien gegeven is dat $a = 5$ en $b = 10$ dan is $f(2) = 20$, $f(0) = 10$, $f(-3) = -5$, enz. Ook kan geschreven

$$f(u) = au + b$$

waaruit blijkt dat x een dummy variabele is. Verder geldt bijvoorbeeld

$$f(3v^2 + c) = a(3v^2 + c) + b$$

waarmee overal waar x voorkomt gesteld wordt dat gelezen moet worden

$$x = (3v^2 + c)$$

Tabel 5. Schematisch overzicht over de verschillende soorten routines en hun belangrijkste eigenschappen

Routine	Wijze van voorkomen	Wijze van oproepen	Bevat (kan bevatten)	Regels met betrekking tot de naamgeving	Plaats in het programma	Wordt als volgt opgeroepen	Retourneert uitkomsten als volgt	Restricties in gebruik
Statement function	In hoofdprogramma's, subroutines en external functions	Door het noemen van de naam van de functie in het programma waarin de functie is gedefinieerd	Definitie van een functie	Overeenkomstige regels voor type aanduiding. Gebruik type statement indien geen default type	Ná alle specificatie statements (REAL, INTEGER, LOGICAL, DIMENSION, COMMON, DATA) en vóór het begin van het eigenlijke programma	Als operand in een expressie	Als functiewaarde, door middel van de functie-naam	Functie-naam en expressie moeten van hetzelfde type zijn. Alleen te gebruiken in het definiërend program
External function	Als afzonderlijk te programmeren routine (subprogramma)	Door de naam in het programma of external function of subroutine in de External function gewenst wordt	Meer dan één statement	Overeenkomstige regels voor type aanduiding. Type aanduiding vóór functienaam indien geen default type	-	Als operand in een expressie	Door middel van de functie-naam als functie waarde en via definitie of herdefinitie van argumenten. Via COMMON	Dummy argumenten mogen niet voorkomen in COMMON en DATA. Mag niet bevatten SUBROUTINE of een andere FUNCTION definitie
Subroutine	Als afzonderlijk te programmeren routine (subprogramma)	Door een CALL in het programma of external function of subroutine in de subroutine gewenst wordt	Meer dan één statement	Vrij	-	Met een CALL statement	Door middel van de argumenten en via COMMON	Dummy argumenten mogen niet voorkomen in COMMON en DATA. Mag niet bevatten FUNCTION of een andere SUBROUTINE definitie

Merk het verschil op met bijvoorbeeld

$$f(a, b) = ax + b$$

In dit geval moet x gegeven zijn en zijn a en b dummy variabelen in het argument. Gegeven $x = 4$, dan is

$$f(3, -6) = 6$$

ook zou gebruikt mogen worden een vorm als

$$f(2d + u^2, -3v + c) = (2d + u^2) x - 3v + c$$

waarmee elke a in het rechterlid overgaat in $(2d + u^2)$ en elke b in $(-3v + c)$.

Belangrijk is de wijze waarop de functie gedefinieerd is. In het tweede voorbeeld is de definitie

$$f(a, b) = ax + b$$

zodat de aanroep $f(b, a)$ de functiewaarde geeft voor

$$f(b, a) = bx + a$$

In Fortran wordt een statement functie voor de voorgaande gevallen gedefinieerd met

```
FLINEAR(X) = A*X + B
```

```
PARAM2(A,B)= A*X + B
```

Namen voor de functie mogen vrij verzonnen worden, rekening houdend met het type van de functie. De definitie is formeel, dat wil zeggen A en B in de eerste functie en X in de tweede functie behoeven niet eerder gedefinieerd te worden. Wél moeten ze ten tijde van de executie bekend zijn. Dus bijvoorbeeld in het eerste geval:

```
⋮  
⋮  
A = 5.    $    B = 10  
Z = FLINEAR(2.)  
⋮  
⋮
```


waarmee dan $Z = 20$. geworden is.

Het meer gecompliceerde voorbeeld wordt dan

```
:  
:  
A = 5.    $    B = 10.    $    V = 8.    $    C = -2.*B/A  
Z = FLINEAR(3.*V**2 + C)  
:  
:
```

Statement functies behoren tot de groep 'definities' en moeten dus aan het eigenlijke programma vooraf gaan. Zie hiervoor hoofdstuk X.

IX.5. EXTERNAL FUNCTIONS

Externe functies worden onafhankelijk van het hoofdprogramma geprogrammeerd. Het zijn zelfstandige programma's die afzonderlijk kunnen worden vertaald en getest. Het opschrift van zulk een programma bevat de naam, de dummy argumenten en de type-aanduiding wanneer de type afspraak niet gevolgd wordt. In het algemeen geldt als schrijfwijze

```
t FUNCTION f(a1, a2, ..., an)
```

of

```
FUNCTION f(a1, a2, ..., an)
```

waarin

t = een van de aanduidingen INTEGER of REAL (andere worden hier niet behandeld)

f = naam van de functie. Wanneer t niet opgegeven wordt moet men de type-afspraken volgen

a_i = dummy argument welke mag zijn een naam van een variabele, naam van een array, of een andere externe functie ($1 \leq i \leq 63$).

afspraak gemaakt worden bijvoorbeeld output altijd naar unit 3. Ook kan dit met behulp van het argument bestuurd worden (of via COMMON zie hoofdstuk X.2). Voorbeeld:

```
PROGRAM SOS(OUTPUT,TAPE3=OUTPUT)
:
:
INK = 3
:
:
----- HELPME(-,-, INK) (Statement waarin HELPME voorkomt)
:
:
END

FUNCTION HELPME(-,-, IO)
:
:
WRITE(IO,-)
:
:
END
```

- De routine springt terug naar het roepende programma op last van een RETURN. In het roepende programma wordt dan met het eerstvolgend statement verder gegaan.
- Eindigt een route door het subprogramma bij END, dan volgt automatisch een return.
- Het kan van voordeel zijn een of meer WRITE's in een subprogramma onder te brengen zodat men in de testfase een duidelijk beeld heeft of de sprong naar het subprogramma is gelukt, en hoever men daarin is gevorderd.

IX.6. SUBROUTINES

De meest uitgebreide vorm van een afzonderlijk subprogramma is de subroutine in Fortran SUBROUTINE.

In een SUBROUTINE wordt niet een enkele functiewaarde berekend die centraal staat maar wordt een afgerond onderdeel van een programma ondergebracht. De naam van de SUBROUTINE dient niet voor informatie-overdracht en de type afspraak heeft hier dan ook geen betekenis.

Een SUBROUTINE mag ook een procedure bevatten zoals een output routine, zonder dat een argument wordt gebruikt. Bovendien kan vanuit een SUBROUTINE naar verschillende plaatsen in het roepende programma worden teruggesprongen.

De statements luiden als volgt:

Plaats van voorkomen	Mag weggelaten	Mag weggelaten
In subprogramma:		
SUBROUTINE naam	$(a_1, \dots, a_n)$	RETURNS($b_1, \dots, b_m$)
In roepend programma:		
CALL naam	$(\alpha_1, \dots, \alpha_n)$	RETURNS($\beta_1, \dots, \beta_m$)

Hierin is

- SUBROUTINE = de eerste regel van een subroutine
- CALL = het statement waarmee de subroutine wordt opgeroepen
- naam = de door de programmeur aan de subroutine te geven naam
- a_i = dummy namen ($1 \leq i \leq 36$) van variabelen
- b_i = dummy namen voor alternatieve labelnummers (geen type afspraak)
- α_i = werkelijk te gebruiken grootheden, die in type, volgorde en aantal met a_i moeten overeenkomen. De α_i mogen een expressie zijn
- β_i = werkelijk te gebruiken labelnummers in het roepende programma

Een voorbeeld van de besturing met alternatieve returns is:

```

SUBROUTINE SUBJE(A, B), RETURNS(A1, A2)
:
:
IF(A.GT.B) RETURN A2
.
IF(A.GT.B**2) RETURN A1
.
RETURN
:
:
END

```

In het roepende programma moet dan voorkomen

```

:
:
20 .....
:
:
CAL SUBJE(X, Y), RETURNS(20, 130)
:
:
130 -----
:
:
```

De RETURN zonder verdere indicatie zorgt dat direct na de CALL wordt verdergegaan in het roepende programma.

De volgende opmerkingen kunnen nog worden gemaakt

- wanneer een argument in het roepende programma een external function is of subroutine naam moet het corresponderende dummy argument op dezelfde wijze worden gebruikt
- een argument in het roepende program moet de naam van een variabele (of een array) zijn wanneer het correspondeert met een dummy argument die in het subprogramma wordt gedefinieerd of opnieuw wordt gedefinieerd. (Anders zou "2" bijvoorbeeld de waarde 10 krijgen).

De laatste opmerking wordt toegelicht met de statements

```

:
:
CALL INT(2)
:
:
END

SUBROUTINE INT(I)
:
:
I = 10
:
:
END
```

waarin dus het argument (I) in het subprogramma (I) opnieuw gedefinieerd wordt (I = 10), terwijl het niet gekoppeld is aan een variabel argument in de CALL (dat is n.l. de waarde 2).

Voor een toepassing van het gebruik van subroutines wordt weer verwezen naar par. 7, (voorbeeld 10).

IX.7. TOEPASSINGEN

Als voorbeelden voor het gebruik van subprogramma's is gekozen het berekenen van verdamping uit stralingsgegevens (zie par. 5 en tabel 6).

Voorbeeld 9 geeft een programma voor dit probleem met behulp van een external function, voorbeeld 10 geeft een programma met een subroutine.

Beide voorbeelden bevatten tevens de volledige job-control voor verwerking over de terminal.

Beide programma's zijn nagenoeg identiek aan elkaar. Het wezenlijke verschil komt tot uiting in de wijze waarop de berekende waarden naar het hoofdprogramma worden teruggevoerd.

Met een external function is dat door middel van de functie-naam GLAS. Deze moet dan ook in de FUNCTION ergens in het linkerlid gedefinieerd worden.

Met een subroutine is dat door middel van het argument en niet via de subroutine naam.

In de argumenten vindt correspondentie plaats tussen NDAG en T (dummy). Dit betekent dat in de subprogramma's de variabele T gedeclareerd moet worden als zijnde van het type integer. Eventuele arrays die als dummy argument in een routine voorkomen moeten evenzo in de routine zelf met een dimensie-statement worden verantwoord (zie volgend hoofdstuk).

Uit de output blijkt dat de tijd benodigd voor het vertalen van beide programma's even lang geweest is (.459 sec.). Het programma met de subroutine heeft echter een wat snellere verwerkingstijd (.086 sec. tegen .092).

De zinnen

EOI ENCOUNTERED AFTER COPY OF FILE

0, RECORD 1

Voorbeeld 9. Berekening van verdamping uit globale straling met een
hoofdprogramma en een external function

```
CONNECT,ICW
COPYBF,ICW,DATA
STRALINGSGEGEVENS H
(DECADESOMMEN)
```

	JAAR	MAAND	DECADE	H
*	1977	4	1	2946
	1977	4	2	3616
	1977	4	3	3177
	1977	5	4	3050
	1977	5	5	4441
	1977	5	6	6660
	1977	6	7	3352
	1977	6	8	3564
	1977	6	9	4482

```
%EOF
RETURN,LGO,PROG,OUTPUT
COPYBF,ICW,PROG
```

```
PROGRAM EVAP(DATA,OUTPUT,TAPE2=DATA,TAPE3=OUTPUT)
REWIND 2 $ REWIND 3
WRITE(3,35)
35 FORMAT(////1H 7X"JAAR MAAND DECADE NDAGEN H"9X"EMAX"/)
C
3 READ(2,1) STAR
1 FORMAT(A1)
IF(STAR.EQ.1H*) GO TO 5 $ GO TO 3
C
5 READ(2,10) J,M,ID,H
10 FORMAT(I11,I5,I8,F9.0)
IF(%EOF(2).EQ.1) STOP 1
NDAG=10 $ IF(ID.EQ.6) NDAG=11
C
CALL GLASC(EMAX,NDAG,H)
C
WRITE(3,40) J,M,ID,EMAX
40 FORMAT(1H+I11,I5,I8,20X,F10.2)
GO TO 5
END
```

```
C
SUBROUTINE GLASC(EMAX,T,S)
INTEGER T
A=7.106*10.**(-3)
B=0.6
EMAX=A*S+B*FLOAT(T)
WRITE(3,35) T,S
35 FORMAT(1H 22X,I10,F12.2)
END
```

```
%EOF
REWIND,PROG
FTN,I=PROG,L=0
LGO
REWIND,OUTPUT
COPYBF,OUTPUT,ICW
```

E_x

COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 .459 CP SECONDS COMPILATION TIME
 COMMAND-
 STOP 1
 .092 CP SECONDS EXECUTION TIME
 COMMAND-
 COMMAND-

JAAR	MAAND	DECADE	NDAGEN	H	EMAX
1977	4	1	10	2946.00	26.93
1977	4	2	10	3616.00	31.70
1977	4	3	10	3177.00	28.58
1977	5	4	10	3050.00	27.67
1977	5	5	10	4441.00	37.56
1977	5	6	11	6660.00	53.93
1977	6	7	10	3352.00	29.82
1977	6	8	10	3564.00	31.33
1977	6	9	10	4482.00	37.85

Voorbeeld 10. Berekening van verdamping uit globale straling met een hoofdprogramma en een subroutine

```
CONNECT, ICW
RETURN, LGO, OUTPUT, DATA
COPYBF, ICW, DATA
STRALINGSGEGEVENS H
(DECADESOMMEN)
```

	JAAR	MAAND	DECADE	H
*	1977	4	1	2946
	1977	4	2	3616
	1977	4	3	3477
	1977	5	4	3050
	1977	5	5	4441
	1977	5	6	6660
	1977	6	7	3352
	1977	6	8	3564
	1977	6	9	4482

```
XEOF
RETURN, LGO, PROG, OUTPUT
COPYBF, ICW, PROG
```

```
PROGRAM EVAP(DATA, OUTPUT, TAPE2=DATA, TAPE3=OUTPUT)
REWIND 2 $ REWIND 3
WRITE(3,35)
35 FORMAT(///4H 7X"JAAR MAAND DECADE NDAGEN H"9X"EMAX"/)
C
3 READ(2,1) STAR
4 FORMAT(A1)
IF(STAR.EQ.4H*) GO TO 5 $ GO TO 3
C
5 READ(2,10) J,M,ID,H
10 FORMAT(I11,I5,I8,F9.0)
IF(EOF(2).EQ.1) STOP 4
NDAG=10 $ IF(ID.EQ.6) NDAG=11
C
EMAX=GLAS(NDAG,H)
C
WRITE(3,40) J,M,ID,EMAX
40 FORMAT(1H+I11,I5,I8,20X,F10.2)
GO TO 5
END
```

```
C
FUNCTION GLAS(T,S)
INTEGER T
A=7.106*10.**(-3)
B=0.6
GLAS=A*S+B*FLOAT(T)
WRITE(3,35) T,S
35 FORMAT(1H 22X,I10,F12.2)
END
```

```
XEOF
REWIND, PROG
FTN, I=PROG, L=0
LGO
REWIND, OUTPUT
COPYBR, OUTPUT, ICW
Ex
```


COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 .459 CP SECONDS COMPILATION TIME
 COMMAND-
 STOP 1
 .086 CP SECONDS EXECUTION TIME
 COMMAND-
 COMMAND-

JAAR	MAAND	DECADE	NDAGEN	H	EMAX
1977	4	1	10	2946.00	26.93
1977	4	2	10	3616.00	31.70
1977	4	3	10	3177.00	28.58
1977	5	4	10	3050.00	27.67
1977	5	5	10	4441.00	37.56
1977	5	6	11	6660.00	53.93
1977	6	7	10	3352.00	29.82
1977	6	8	10	3564.00	31.33
1977	6	9	10	4482.00	37.85

EOF ENCOUNTERED AFTER COPY OF FILE
 0, RECORD 1

zijn een gevolg van het feit dat als commando gebruikt is een COPYBF (copieer een binaire file). Aangezien de output uit records is opgebouwd, wordt gemeld dat er 1 record gelezen is van file 0 en dat er geen EOF teken gelezen is.

Met de COPYBR (copieer een binair record) wordt er precies één record gelezen (wat de bedoeling was) en volgt er geen informatieve melding.

Zie ook voorbeeld 11 in hoofdstuk X.5 dat dezelfde berekening uitvoert met uitsluitend subroutines.

X

A D M I N I S T R A T I E

X.1. INLEIDING

Over de administratie rond een programma is reeds een en ander meegedeeld. Voor een deel betreft het de naamgeving van de variabelen om aan te geven hoe er mee gerekend moet worden, en het aantal posities dat een array zal innemen. Zie hiervoor de hoofdstukken IV en V.

Besproken moet nog worden de invloed die de programmeur op de geheugen-indeling kan uitoefenen en het definiëren (initialiseren) van waarden van variabelen aan het begin van een programma. Het betreft hier de COMMON- en DATA-statements.

X.2. COMMON

Het COMMON-statement regelt het reserveren van geheugenruimte voor zover men daar invloed op wil uitoefenen. De opdracht luidt

```
COMMON a1, a2, ..., an
```

waarin a_i namen van variabelen zijn of array namen of array elementen. Er mogen meer dan één COMMON-statements gebruikt worden.

Het belang van de COMMON is dat dezelfde COMMON ook in subprogramma's mag voorkomen. Hierdoor kunnen dezelfde variabelen over en weer gebruikt worden. Dit werkt dus principieel anders dan in gelijkstelling van argument. Tijdens de call in het roepende programma kunnen de variabelen in common niet alsnog een waarde gegeven worden zoals in het argument. De tijdens de call geldende waarden blijven gelden in het aan te roepen programma. Voorbeeld

```
PROGRAMMA UITGAAN(---)
COMMON A, B, C
COMMON X, Y, Z
:
:
A = 5.    $    B = 10.    $    X = 3HYES    $    Y = A*B
:
:
CALL RECREA -----
:
:
END
```

```

SUBROUTINE RECREA -----
COMMON A, B, C
COMMON X, Y, Z
:
:
Z = A + B + Y
IF(X.EQ.3HYES) C = Z $ A = 10.
:
:
END

```

Na de CALL en de executie van het subprogramma hebben de variabelen de volgende waarde:

```

A = 10.
B = 10.
C = 65.
X = 3HYES
Y = 50.
Z = 65.

```

- Dimensies mogen tegelijk met het COMMON-statement worden gedefinieerd. Bijvoorbeeld COMMON X(100, 10), KLM(20, 20).
- Het is niet noodzakelijk in een subprogramma dezelfde namen in COMMON te gebruiken. Overzichtelijk is het vaak wel. Zo zou men:

```

SUBROUTINE RECREA -----
COMMON R, S, T

```

of

```

COMMON U(3)

```

Het belangrijkste is de gelijkheid van plaats en type.

- Het gebruik van COMMON is efficiënter dan het gebruik van argumenten.
- COMMON-statements mogen ook gebruikt worden in FUNCTION-subprogramma's.
- Bij gebruik van veel COMMON-variabelen in verschillende subprogramma's verdient het aanbeveling de COMMONS's exact te kopiëren, bij-

voorbeeld met SUEDE (zie ICW-Nota 1026, 'Aspecten' dl 11).

X.3. DATA

Het DATA-statement wordt gebruikt om op snelle wijze variabelen een (begin-)waarde in het programma te geven. Bijvoorbeeld de waarde .0 aan optelvariabelen, de waarde 1. aan factoren, enz. De schrijfwijze is

```
DATA list1/getallen1/, ..., listn/getallenn/
```

waarin

list₁ = lijstje van variabelen (namen, arrays, elementen uit een array, array met implied DO). Er mogen geen dummy argumenten in voorkomen.

getallen₁ = lijstje van getalwaarden die achtereenvolgens aan de variabelen, genoemd in list₁, worden toegekend. Er mogen ook Hollerith constanten in voorkomen. Getallen uit de list mogen tussen haakjes worden gegroepeerd en een herhalingsfactor worden meegegeven.

Arrays moeten van tevoren in een DIMENSION statement zijn gedeclareerd. Bij afwijking van de type-afpraak moet dit van tevoren zijn opgegeven. Voorbeelden

```
DATA X, Y, Z/1., 2., 3./, I, K/2*(0)/
```

en met arrays,

```
DIMENSION A(10, 10)
```

```
DATA (A(1, K), K = 1, 10)/5*100., 4*99, 2HNO/
```

Hiermee is verkregen voor de matrix A

100.	100.	100.	100.	100.	99.	99.	99.	99.	NO
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
⋮					niet gedefinieerd				⋮
enz.									

Het '*'-teken betekent hier voeg 5-maal het getal 100. toe aan genoemde variabelen enz.

Worden geen dimensies vermeld, dan worden alle elementen van de array gedefinieerd met het aantal beschikbare getallen. Voorbeeld:

```
DIMENSION A(5, 5), B(5, 5)
DATA A/30*10./, B/20*3./
```

De overvloedige getallen 10. worden niet gebruikt. De laatste 5 elementen van B hebben geen waarde toegewezen gekregen.

De volgorde van opvullen bij arrays wanneer dit niet gespecificeerd wordt is (voor een drie-dimensionale array met dimensies 5 x 10 x 15):

```
A(1, 1, 1)
A(2, 1, 1)
:
:
A(5, 1, 1)
A(1, 2, 1)
A(2, 2, 1)
:
:
A(5, 2, 1)
:
:
A(5, 10, 1)
A(1, 1, 2)
A(2, 1, 2)
:
:
A(5, 10, 15)
```

zodat van links naar rechts alle indices worden afgewerkt, respectievelijk voor rij, kolom, matrix. Merk op dat de nummering net andersom oploopt dan bij getallen.

X.4. SAMENVATTING

Alle administratieve specificaties moeten vóór in het programma worden gegeven. Specificatie statements mogen worden herhaald als hieraan behoefte bestaat en dit de duidelijkheid van de indeling ten goede komt.

Veelal zullen niet alle specificatie soorten nodig zijn, maar als ze voorkomen, dan is de volgende volgorde verplicht.

PROGRAM		
INTEGER	(dimensies)	} mogen worden verwisseld
REAL	(dimensies)	
LOGICAL	(dimensies)	
DIMENSION	(dimensies)	
COMMON	(dimensies)	
statement function		
DATA		
programma statements		
:		
:		

Dimensies mogen slechts één maal worden opgegeven, maar dit kan in verschillende statements. Wordt de dimensie van een variabele die in COMMON voorkomt opgegeven in DIMENSION, dan moet dit in elk subprogramma eveneens worden gedaan. Ook indien een type definitie voorkomt moet dit voor variabelen die naar een subprogramma worden overgebracht (via argument of COMMON), in het subprogramma worden herhaald.

Gedimensioneerde variabelen die in COMMON voorkomen kunnen dus het eenvoudigst met dimensies in COMMON worden gedefinieerd.

X.5. TOEPASSINGEN

Het gebruik van COMMON-statements wordt geïllustreerd in voorbeeld 11. Dit voorbeeld is een uitwerking van hetzelfde probleem als dat van voorbeelden 9 en 10 (zie vorig hoofdstuk).

Merk op dat het hoofdprogramma tot een minimum is gereduceerd. Het vervult alleen nog de functie van het in juiste volgorde aanroepen van de verschillende subroutines. Om zeker te zijn dat alle subroutines met dezelfde grootheden werken, zijn deze met COMMON in een gemeenschappelijk deel van het geheugen ondergebracht. Vanuit elke subroutine zijn de variabelen in COMMON bereikbaar.

De subroutine HEADING maakt geen gebruik van een of meer variabelen en behoeft dus zelf geen COMMON-statement te bevatten.

Elke subroutine kan afzonderlijk geprogrammeerd worden. Het enige gemeenschappelijke met het hoofdprogramma is verder nog het gebruik van de tape-unitnummers voor in- en output.

Het voordeel van deze werkwijze is dat wanneer bijvoorbeeld een andere output layout vereist is, dit afzonderlijk geprogrammeerd kan worden (en aangepast kan worden) in subroutine WRITE.

Door in COMMON tevens de dimensies van de variabelen op te geven is de gehele administratie in één statement verzorgd.

Het is niet noodzakelijk in alle COMMON's dezelfde namen te gebruiken. De onderlinge relatie is alleen naar type en aantal. Zo zal bij gebruik van

```
COMMON JAARTAL, JAARTJE
```

na de inlees-routine de volgende waarden toegekend zijn:

```
aan JAARTAL dus (JAAR(1)): 1977
```

```
aan JAARTJE dus (JAAR(2)): 1977
```

Voorbeelden van het gebruik van DATA zijn reeds eerder gegeven. Zie voorbeeld 8, pag. 118, en voorbeeld 7, pag. 78. Bij variërende inputgegevens zal aan het gebruik van READ veelal de voorkeur worden gegeven boven het gebruik van DATA.

Opmerking

Door het onderling oproepen van routines te beperken en per routine DO-loops te gebruiken is het programma uit voorbeeld 11 efficiënter geworden dan uit voorbeeld 10.

De vertaaltijd was bijna 2 x zo lang (.861 sec. tegen .459 sec.)

Voorbeeld 11. Het consequent gebruik van subroutines en hun koppeling via COMMON-statements.

```

CONNECT,ICW
COPYBF,ICW,DATA
STRALINGSGEGEVENS H
(DECADESOMMEN)
      JAAR  MAAND  DECADE    H
*
      1977    4      1    2946
      1977    4      2    3616
      1977    4      3    3177
      1977    5      4    3050
      1977    5      5    4441
      1977    5      6    6660
      1977    6      7    3352
      1977    6      8    3564
      1977    6      9    4482
ZEOF
RETURN,LGO,PROG,OUTPUT
COPYBF,ICW,PROG
PROGRAM EVAP(DATA,OUTPUT,TAPE2=DATA,TAPE3=OUTPUT)
COMMON JAAR(9),MAAND(9),IDEC(9),NDAGEN(9),H(9),EMAX(9)
CALL HEADING
CALL READ
CALL GLAS
CALL WRITE
END
C
SUBROUTINE HEADING
REWIND 3
WRITE(3,35)
35 FORMAT(////1H 7X"JAAR  MAAND  DECADE  NDAGEN          H"9X"EMAX"/)
END
C
SUBROUTINE READ
COMMON JAAR(9),MAAND(9),IDEC(9),NDAGEN(9),H(9),EMAX(9)
REWIND 2
3 READ(2,35) STAR
35 FORMAT(A1)
IF(STAR.EQ.4H*) GO TO 5    $    GO TO 3
C
5 DO 20 I=1,9
  READ(2,40) JAAR(I),MAAND(I),IDEC(I),H(I)
40 FORMAT(I4,I5,I8,F9.0)
  IF(EOF(2).EQ.4) STOP 4
  NDAGEN(I)=40    $    IF(ID.EQ.6) NDAGEN(I)=44
20 CONTINUE
END
C
SUBROUTINE WRITE
COMMON JAAR(9),MAAND(9),IDEC(9),NDAGEN(9),H(9),EMAX(9)
DO 10 I=1,9
10 WRITE(3,40) JAAR(I),MAAND(I),IDEC(I),NDAGEN(I),H(I),EMAX(I)
40 FORMAT(1H I4,I5,I8,F12.0,F12.2)
END
C
SUBROUTINE GLAS
COMMON JAAR(9),MAAND(9),IDEC(9),NDAGEN(9),H(9),EMAX(9)
A=7.106*10.**(-3)
B=0.6
DO 10 I=1,9
10 EMAX(I)=A*H(I)+B*FLOAT(NDAGEN(I))
END
ZEOF
REWIND,PROG
FTN,I=PROG,L=0
LBO
REWIND,OUTPUT
COPYBR,OUTPUT,ICW
E
X

```

COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 COMMAND-
 .861 CP SECONDS COMPILATION TIME
 COMMAND-
 END EVAP
 .067 CP SECONDS EXECUTION TIME
 COMMAND-
 COMMAND-

JAAR	MAAND	DECADE	NDAGEN	H	EMAX
1977	4	1	10	2946.	26.93
1977	4	2	10	3616.	31.70
1977	4	3	10	3177.	28.58
1977	5	4	10	3050.	27.67
1977	5	5	10	4441.	37.56
1977	5	6	10	6660.	53.33
1977	6	7	10	3352.	29.82
1977	6	8	10	3564.	31.33
1977	6	9	10	4482.	37.85

door de grotere interne organisatie. De verwerkingstijd is echter korter (.067 sec. tegen .086 sec.).

XI

DE TERMINAL ALS INPUT/OUTPUT MEDIUM

XI.1. INLEIDING

In hoofdstuk I.7 is behandeld hoe men informatie geschreven op een output file, zoals resultaten van een Fortran-programma, op het beeldscherm kan krijgen en kan laten printen.

Het is ook mogelijk de output van een programma direct op het beeldscherm te laten afbeelden. De terminal speelt dan de rol van local file, maar dat moet eerst aan de computer bekend worden gemaakt. De Fortran-opdrachten hiertoe zijn:

```
CALL CONNEC(nLfile)
```

waarin n het aantal characters is waaruit de filenaam (file) bestaat.

Loskoppelen kan met

```
CALL DISCON(nLfile)
```

Voorbeeld:

```
PROGRAM THUIS(OUTPUT,TAPE6=OUTPUT)
:
:
CALL CONNEC(6LOUTPUT)
WRITE(6, 10) Z
10 FORMAT(1H F10.3)
:
:
```

De waarde van Z wordt nu in een F10.3 format op het beeldscherm afgebeeld.

Ook het inlezen van informatie kan op deze wijze worden georganiseerd en rechtstreeks vanaf het scherm plaatsvinden.

Bij het schrijven naar de terminal moet begonnen worden met een 1Hb (blank) specificatie. Bij het omtlenen van informatie vanaf het beeldscherm is dit niet noodzakelijk.

Door de in- en uitvoermogelijkheden te combineren kan een wijze van werken worden opgebouwd waarbij de computer vraagt en de gebruiker antwoord. Een dergelijk gebruik van de apparatuur heet werken in

conversational mode.

Deze wijze van werken vergt meer programmeerarbeid maar heeft het voordeel dat de gebruiker met weinig toelichtingen op het programma aan het werk kan. Voorkomende fouten kunnen door het programma worden ontdekt en het verzoek op een correct antwoord kan worden herhaald. Van belang is:

- vragen kort en bondig te formuleren
- geen (misschien wel leuke) niet ter zake doende vragen te stellen (deze gaan gauw vervelen en belemmeren een vlotte produktie)
- de gebruiker te helpen bij het kiezen van het juiste format (hierover aanwijzingen geven)

Het volgende programma-onderdeel geeft een illustratie van een conversationeel programma (zie par. 2).

XI.2. CONVERSATIONAL MODE

In deze paragraaf gaan we ervan uit dat een aantal gegevens moet worden ingelezen, betrekking hebbend op een reeks van jaren. Nadat een jaar compleet is ingelezen kan de gebruiker van het programma beslissen of een volgend jaar zal worden ingelezen dan wel of verder gegaan moet worden met de eigenlijke bewerking. Het conversationele gedeelte van het programma kan er dan als volgt uitzien:

```

        PROGRAM OVER(SCREEN,OUTPUT,
*TAPE1=SCREEN,TAPE3=OUTPUT)
        :
        :
        CALL CONNEC(6LSCREEN)
        :
50 : (input van gegevens over één jaar)
        :
60 WRITE(1, 10)
10 FORMAT(1H "IS INPUT JA/NEE GEREED?"/)
    READ(1, 20) KLAAR
20 FORMAT(A3)
    IF(KLAAR.EQ.2HJA) GO TO 55
    IF(KLAAR.EQ.3HNEE) GO TO 50
    WRITE(1, 30)  $   GO TO 60
30 FORMAT(//1H"ANTWOORD ONJUIST, ZEG JA/NEE")
C
55 CONTINUE
    :
    :
    WRITE(3,---) ---
    :
    :
    END

```

Aangezien de READ gekoppeld is aan unit 1, welke gekoppeld is aan de terminal, zal de executie van het programma hier stoppen totdat de invoer van de variabele KLAAR heeft plaatsgevonden.

De handelingen op de terminal zijn hiervoor het intikken van

JAA

of

NEEA

ter plaatse waar de cursor zich bevindt.

Ten einde geen misverstand te laten ontstaan worden beide antwoorden gecheckt. Zijn beide antwoorden fout, dan wordt een foutmelding gegeven (WRITE(1, 30)) en wordt teruggesprongen naar de oorspronkelijke vraag (label 60).

Wanneer een aantal getallen moet worden opgegeven via de terminal, dan kan de programmeur de gebruiker behulpzaam zijn door de posities op het beeldscherm af te beelden. Bijvoorbeeld: wanneer opgegeven moet worden een klasse voor grondgebruik en een daarbij behorend oppervlakte in ha in F6.2 format, dan kan op het beeldscherm worden afgebeeld

```
VERMELD KLASSE GRONDGEBRUIK  EN  HA
                                X          XXX.XX
                                -
```

Met - wordt de positie van de cursor aangeduid. Met het READ-FORMAT moet vanaf deze positie verder gegaan worden. Voorbeeld:

```
WRITE(1, 10)
10 FORMAT(1H "VERMELD KLASSE GRONDGEBRUIK"
  * " EN HA"/1H 13X"X"17X"XXX.XX"/1H 13X)
  READ(1,11) KLAS, HA
11 FORMAT(1I, 13X, F10.2)
```

Indien tijdens het intikken van een antwoord een fout wordt gemaakt, kan men deze aan de terminal herstellen door de cursor over de ingetikte informatie naar links te sturen (knop met rode pijl naar links gericht). De gepasseerde tekens zijn nu uitgewist en men kan (of ook moet) vanaf de plaats waar de cursor zich bevindt de fout verbeteren.

Opgemerkt wordt dat de edit-functies (knoppen rechts op het toetsenbord) niet werken wanneer de terminal in on-line mode staat.

XII

NOGMAALS : HET PROGRAMMEREN

Uit al het voorgaande zal duidelijk geworden zijn dat er geen vaste regels zijn om voor een gegeven probleem een Fortran-programma te schrijven. Veel hangt af van de specifieke wensen van de gebruiker. Beslissingen moeten worden genomen over:

- Moeten de gegevens in het programma worden gedefinieerd of van een afzonderlijke file worden ingelezen.
- Moet een kwaliteitscontrole op de gegevens worden ingebouwd.
- Hoe groot moeten de dimensies worden gekozen.
- Is het noodzakelijk een deel van de bewerkingen in een subprogramma onder te brengen.
- Zo ja, moet overdracht van variabelen dan via argument of via COMMON plaatsvinden.
- Hoe algemeen moet de output zijn.
- Moet de output op het beeldscherm komen of op een aparte file en bewaard blijven.
- Hoeveel toelichting moet er in het programma worden opgenomen.

Veelal zal blijken dat een programma steeds groeit. Na enige tijd in gebruik te zijn geweest ontstaan nieuwe wensen die men achteraf wil inbouwen. Van belang is wel alle constanten variabel te maken en voorin in het programma te definiëren met begeleidend commentaar over het gebruik. Men is dan vrij de omvang van bijvoorbeeld DO-lussen gemakkelijk aan te passen.

In manuals over het programmeren wordt vaak weinig aandacht besteed aan de wijze van programmeren en adviezen hierover komen slechts zelden ter sprake. Uit een aantal verspreide publikaties volgen hier een aantal praktische aanwijzingen voor het schrijven van een efficient programma.

Bij het lezen van de volgende overwegingen moet wel bedacht worden dat programma's van zo'n 1000 statements dat een 10-tal keren wordt doorlopen nog niet tot de grote programma's gerekend wordt. Vanuit dat oogpunt moeten de volgende opmerkingen worden gelezen.

Voor de oorspronkelijke bronnen van de volgende citaten wordt verwezen naar referenties onder 'programmering' in de literatuurlijst.

- Gebruik voldoende COMMENT-regels. Zij behoren niet tot het programma in functioneel opzicht en vertragen de werking ervan dus niet.
- Geef vooral een duidelijke toelichting op geïntroduceerde besturingsvariabelen. Deze zijn geen onderdeel van het eigenlijke probleem en later is vaak moeilijk te begrijpen waarom ze werden toegepast.
- Gebruik de GO TO zo weinig mogelijk en vermijd het heen en weer springen door een programma.
- Maak gebruik van de logical IF en vermijd de numerieke IF, en computed GO TO. De bedoelingen worden daarmee meestal veel duidelijker.
- Computed GO TO's worden pas efficiënter boven een aantal van 3 labels. Maak gebruik van een serie logical IF's als hieraan niet voldaan is.
- Zorg er voor dat bij een logical IF een eventuele GO TO behoort bij de minst waarschijnlijke uitkomst van de voorwaardelijke expressie.
- Laat DO-loops, ook - en vooral -, bij nesten, ieder in een eigen CONTINUE eindigen. Slechts zelden is de compacte schrijfwijze voor geneste DO's overzichtelijk.
- Haakjes mogen steeds vrij gebruikt worden. Toepassing ervan, ook indien algebraïsch niet strikt noodzakelijk, verduidelijkt de bedoelingen van de programmeur.
- Verduidelijk lange condities in logical IF's door het argument onder te verdelen en de onderdelen verticaal te plaatsen met behulp van vervolgregels.
- Maak de labelnummering enigszins systematisch maar niet aansluitend om later nog nieuwe nummers te kunnen tussenvoegen.
- Gebruik zinvolle namen voor variabelen en kort deze niet teveel af. Geen code's gebruiken voor de naamgeving.
- Gebruik geen 'mixed-mode' expressies. De door de computer uit te voeren conversie vergt vaak meer tijd dan de berekening zelf. De uitkomst is bovendien vaak onzeker. Wel kan veilig toegepast worden conversies van het type $F = I$ en $I = F$.
- Voor kleine, vaak herhaalde routines is het gebruik van subroutines en functies niet zinvol.
- Beperk de in- en output gedurende de testfase tot het allernood-

zakelijkste.

- Maak waar mogelijk gebruik van implied DO-loops.
- Plaats berekeningen van constanten buiten de DO-loop.
- COMMON-gebieden zijn efficiënter dan argumenten voor waarden (variabelen) overdracht.
- Test geen REAL variabelen op exacte gelijkheid maar geef een zekere tolerantie mee.
- Gebruik SQRT(X) in plaats van $X^{.5}$, en schrijf $A*A*A*...$ voor $A**N$ als N niet te groot is.

Deze opsomming is bedoeld om in geval van twijfel een aantal richtlijnen te geven. De grondstelregel moet echter blijven:

LEG UZELF GEEN BEPERKINGEN OP DIE DE OVERZICHTELIJKHEID EN HET GEMAK VAN GEBRUIK VAN UW PROGRAMMA KUNNEN SCHADEN.

LITERATUUR

AFDELING WISKUNDE. Technische Mededeling 3. Accounting System.

10-aug-1978 pp 4.

AMSTEL, J.J. VAN, 1976. EIT diktatenserie, Programmeer-problemen, technieken en opgaven. Academic Service, den Haag; pp 135.

BOHEEMEN, P.J.M. VAN, 1979. Het hydrologisch model in het onderzoek met betrekking tot het project uitgevoerd door de Werkgroep 'Kanaal Waddinxveen-Voorburg' (concepten, interne communicatie).

FORTTRAN EXTENDED, Version 4, Reference manual (revised edition).

Control Data Corporation, Minneapolis. USA, 1975. Revision D (11-30-73). Publication No. 60305601., met toevoegingen tot en met revision G. Fortran voor Cyber 170 series, 70 series, 6000 and 7000 series.

HAMAKER, Ph., 1979. Berekening verdamping voor de glastuinbouw met betrekking tot het project uitgevoerd door de Werkgroep 'Kanaal Waddinxveen-Voorburg' (concepten, interne communicatie).

MAASSEN, J.R. en Ph.Th. STOL, 1978. Aspecten van Informatieverwerking dl 13: Handleiding tot het gebruik in ON-LINE mode van de ICW Teleprint-terminal. ICW-Nota 1051; pp 37.

—— en Ph.Th. STOL, 1978. Aspecten van Informatieverwerking dl 14: Praktijkvoorbeelden van het gebruik van de ICW Teleprint terminal. ICW-Nota 1052; pp 77.

—— en Ph.Th. STOL, 1978. Aspecten van Informatieverwerking dl 18: Batch-verwerking door middel van de terminal. ICW-Nota 1070; pp 33.

PDP-11 Fortran Language Reference Manual. Digital Equipment Corporation, Maynard, Mass. USA. 1975. Update Notice 1. June 1977. Publication DEC-11-IFLRA-C-DN1. Fortran voor PDP en DEC-computers.

PROGRAMMERING (a), 1972. How to write a readable FORTRAN program. RC-Bulletin No. 79.13, overgenomen uit DATA PROCESSING DIGEST, Vol. 19 No. 1(1972), McCracken, D.D., DATAMATION, Oct. 1972, pp. 73-75,79.

- PROGRAMMERING (b). Verhoging van efficiency en verkorting van reken-
tijden van FORTRAN-programma's. R.C.-Bulletin No. 60.9 (na-
slagartikel).
- STOL, Ph.Th., 1977. Aspecten van Informatieverwerking, dl 4: Het sys-
tematisch bepalen van de afgeleiden van een functie ten be-
hoeve van hun programmering. ICW-Nota: 948 pp 21.
- 1977. Aspecten van Informatieverwerking, dl 11: Een syste-
matische samenvatting van SUEDE, de routine voor het bijwer-
ken van files. ICW-Nota 1026; pp 31.
- 1978. Aspecten van Informatieverwerking, dl 5: Mogelijkheden
en beperkingen bij het verzinnen van namen bij informatiever-
werking. ICW-Nota: 1037 pp 11.
- 1979. Aspecten van Informatieverwerking, dl 21: Over het
bijwerken en tot executie brengen van een programmapakket.
ICW-Nota: 1117.
- 1979. Proceedings of the 1976 ESCAP Roving Seminar on the
Use of Computers in Hydrology and Water Resources planning.
Part II. i Hydrology and Water Resources Planning: Programs
for the analysis of hydrological data in water resources
planning. United Nations, Water Resources Series. In Print.
- WILDE, J.G.S. DE en Th.J. LINTHORST, 1978. Berekening en bevoeding
in Nederland in het zeer droge jaar 1976. Landbouwk. Tijd-
schrift/pr nr 3. pp 3-11. (ICW-verspreide overdrukken 212).

Bijlage 1

PDP-11 Fortran

De hier behandelde Fortran versie kan ook worden gebruikt op de PDP-11 computers die over een Fortran IV compiler beschikken. Er zijn echter verschillen die maken dat het raadplegen van de bijbehorende handleidingen wel gewenst is. Hieronder volgen enkele voorbeelden. De beschrijving gaat uit van PDP-11 Fortran

- De stapgrootte in een DO mag negatief zijn
- Voor comment regels is alleen de C toegestaan
- Namen van variabelen mogen willekeurig lang zijn, characters na de 6e worden echter genegeerd
- De range van getallen is
integer (absoluut) ≤ 32767
real (") $[0.29 \times 10^{-38}, 1.7 \times 10^{38}]$
- Variabelen mogen gedefinieerd worden met een aantal van 1-7 dimensies
- DO parameters mogen integer expressies zijn
- ACCEPT (input via terminal keyboard)
- TYPE (output naar terminal display)
- apostrophe in plaats van quote (' i.p.v. ")
(gebruik van nH is dus meer algemeen)
- In een FORMAT mogen variabelen worden toegepast
- In PROGRAM-kaart mag wel een naam maar mogen geen argumenten voorkomen
- Er is direct access mogelijk (schijfgeheugen)
- A**B**C wordt verwerkt als A**(B**C)

Onder meer niet behandelde onderwerpen

- Complexe variabelen en constanten
- Getallen met grotere nauwkeurigheid (double precision)
- Mixed-mode uitdrukkingen (reals en integers in één opdracht)
- Masking uitdrukkingen (bit operaties)
- Assigned GO TO
- Two branch IF (beslist op $\neq 0$ of $= 0$)
- Unformatted READ en WRITE
- Diverse veld specificaties in FORMAT's
- Variabel FORMAT (aanpassing aan uitkomsten)
- ENCODE/DECODE (rechtstreeks schrijven naar en lezen uit het geheugen)
- Variabele DIMENSION's (in subroutines)
- Labelled COMMON (indeling COMMON block)
- EQUIVALENCE (gelijkstelling van variabelen met verschillende namen)
- ENTRY (alternatieve beginpunten in subroutines)
- Vervolg op de lijst met 'Built-in' functies (tabel 4, pag. 73)

Bijlage 3

Wat niet is toegestaan

Binnen de regels van Fortran gelden nog enkele 'verboden' waarvan een aantal van de belangrijkste hier worden samengevat.

Niet toegestaan is:

- programma tekst in posities (kolommen) 73-80
- meer dan 10 Hollerith characters in nH conversies
- file namen van meer dan 6 (zes) characters in de program kaart
- sprong naar een labelnummer dat bij een FORMAT behoort
- een GO TO als laatste statement van een DO-loop
- een DO-loop te beëindigen met
 - arithmetic IF
 - RETURN
 - STOP
 - DO
 - two-branch logical IF,
 - logical IF met als statement een van de vorige
- waarden ≤ 0 voor DO parameters
- DO parameters in de DO zelf opnieuw definiëren
- een label toe te kennen aan een END statement
- het schrijven van
 - FORMAT
 - END
 - gelabelde statements
 - na een \$ als scheidingsteken
- een aantal characters in output groter dan de gegeven specificatie (output geeft dan ****)
- een waarde < 1 toe te kennen aan een index van een geïndiceerde variabele
- variabelen in COMMON of DATA te gebruiken als dummy in een subprogram argument, wanneer deze COMMON of DATA in dat subprogramma voorkomen
- het gebruik van een volgende FUNCTION of SUBROUTINE in een FUNCTION of SUBROUTINE
- namen van een FUNCTION OF SUBROUTINE sub-programma gebruiken als

variabele, anders dan het gebruik waarvoor ze bedoeld zijn.

Uiteraard is het gebruik van getallen die geen bewerking toelaten niet toegestaan. Dit zal vaak van de gebruikte gegevens afhangen en hoeft niet in eerste instantie op een programmeerfout te duiden. Wel kunnen aanwijzingen worden verkregen de nodige tests in een programma te moeten inbouwen. Bijvoorbeeld bij $A = X/Z$, ga na of $Z \neq 0$ en sla anders de g over, enz.

In plaats van de verwachte getallen kunnen de volgende tekens verschijnen:

***	getal past niet in het FORMAT
III	getal is onbepaald (indefinite) bijvoorbeeld $\frac{0}{0}$ (F-conversie of E-conversie)
RR	getal is groter dan de computer verwerken kan (out of range) (F- en E-conversie)
XXX	idem (I-conversie)

De range is:

I-conversie	$2^{48} - 1$
F- "	10^{-293} tot 10^{322}
E- "	

Voor het opsporen van dit type fouten kan het dienstig zijn vóór de LGO opdracht te geven het commando

MODE,0

De verwerking wordt dan in ieder geval voortgezet. In de output valt dan af te lezen wat nog wel en wat niet meer bruikbare resultaten gaf.

