Aris Lourens

# RICHTLIJNEN VOOR ONTWIKKELING VAN COMPUTERPROGRAMMATUUR IN DE HYDROLOGIE

CIP DATA

Richtlijnen

Richtlijnen voor ontwikkeling van computerprogrammatuur in de hydrologie (redactie: J.C. Hooghart, K. Kovar en J.M.P.M. Peerboom). Delft; Commissie voor Hydrologisch Onderzoek TNO. - ill. - (Rapporten en Nota's/Commissie voor Hydrologisch Onderzoek TNO, no. 27). Eindrapport van de CHO-Werkgroep Richtlijnen Computerprogrammatuur Hydrologie.
Met literatuuropgave
ISBN 90-6743-195-8

Trefwoord: model, computerprogramma, hydrologie.

# RICHTLIJNEN VOOR ONTWIKKELING VAN COMPUTERPROGRAMMATUUR IN DE HYDROLOGIE


EINDRAPPORT VAN DE CHO-WERKGROEP
RICHTLIJNEN COMPUTERPROGRAMMATUUR HYDROLOGIE

COMMISSIE VOOR HYDROLOGISCH ONDERZOEK TNO
DELFT, 1992

RAPPORTEN EN NOTA'S No. 27

VOORWOORD

De voorliggende publikatie is opgesteld door de Werkgroep Richtlijnen Computerprogrammatuur Hydrologie. De werkgroep was ingesteld door het Klein Comité van de Commissie voor Hydrologisch Onderzoek TNO.

De huidige kwaliteit van computerprogrammatuur op het gebied van de hydrologie laat vaak te wensen over voor wat betreft betrouwbaarheid, onderhoudbaarheid, overdraagbaarheid, koppelingsmogelijkheden en efficiëntie in het gebruik van computerprogrammatuur (modelleren) in de hydrologische praktijk. De werkgroep heeft zich over deze problematiek gebogen. Tijdens de discussies binnen de werkgroep werd de problematiek geanalyseerd en zijn mogelijke oplossingen naar voren gebracht. De werkgroep heeft zich uiteindelijk beperkt tot het uitwerken van één van de oplossingen van het probleem, namelijk het opstellen van richtlijnen voor de ontwikkeling van de computerprogrammatuur. Wat betreft de overige factoren, worden enkele aanbevelingen tot maatregelen gedaan om in de toekomst tot een kwalitatief hoogwaardiger programmatuur te kunnen komen.

Dit rapport heeft ten doel aanbevelingen (richtlijnen) te presenteren ten behoeve van de ontwikkeling van computerprogrammatuur. Indien opgevolgd, zullen deze aanbevelingen op korte termijn leiden tot de verbetering van de kwaliteit van de programmatuur en dus ook een efficiëntere omgang met programmatuur in de praktijk mogelijk maken. Het rapport is weliswaar toegespitst op de praktijk van de computertoepassingen in de geohydrologie, maar verwacht wordt, dat de aanbevelingen ook in andere vakgebieden bruikbaar zullen zijn. De werkgroep pretendeert niet met geheel nieuwe inzichten te zijn gekomen, maar beoogt een bruikbaar overzicht te hebben geproduceerd, dat betrekking heeft op de belangrijkste ontwikkelfasen van computerprogrammatuur op het gebied van hydrologie.

Het rapport is bestemd voor zowel ontwikkelaars (ontwerpers en programmeurs) als voor gebruikers van hydrologische programmatuur. Van de lezer wordt verwacht dat deze over enige basiskennis van het programmeren beschikt. Beginnende ontwikkelaars kunnen uit het rapport normen ontlenen voor een te hanteren programmeerstijl, meer ervaren ontwikkelaars kunnen hun programmeerstijl toetsen en mogelijk verbeteren.

INHOUDSOPGAVE

# 1 INLEIDING

## 1.1 Achtergrond

In Nederland wordt momenteel een groot aantal computerprogramma's gebruikt op het gebied van de hydrologie en het waterbeheer (SAMWAT, 1991). Steeds vaker wordt gebruik gemaakt van deze programma's bij het oplossen van problemen in de dagelijkse praktijk. Naast een ontwikkeling van nieuwe programma's vindt ook steeds vaker een koppeling plaats van bestaande programmatuur. Terwille van een beter begrip van de in dit rapport behandelde materie wordt het wenselijk geacht om een kort historisch overzicht te geven van de ontwikkeling van dit type computerprogramma's in Nederland.

De ontwikkeling van computerprogramma's op het gebied van hydrologie is in Nederland in het midden van de zeventiger jaren begonnen. De programma's werden niet systematisch ontwikkeld, maar waren meestal een ad-hoc omzetting van een bestaand hydrologisch probleem (in navolging van elektrische analogons) in termen van FORTRAN-instructies, te realiseren binnen de kortst mogelijke tijd.

De documentatie was zeer summier of ontbrak in het geheel. Dat gaf weinig of geen problemen, want de ontwikkelaar was meestal ook de enige gebruiker. Wat betreft onderhoudbaarheid was er ook geen probleem. Immers, het onderhoud en de verdere ontwikkeling werden meestal door de oorspronkelijke maker gedaan.

Verbeteringen en veranderingen aan de programmatuur werden in de loop der tijd uitgevoerd met behoud van de oorspronkelijke gebrekkige interne structuur en overige tekortkomingen. Bij het groter worden van de programma's, door veranderingen en toevoegingen, zijn de leesbaarheid en onderhoudbaarheid van de programmacode achteruit gegaan. Meestal was alleen de ontwikkelaar zelf in staat om de programmacode goed te begrijpen. Wat in het begin geen probleem was, begon later, vanwege toenemende complexiteit van onderzoekvraagstukken, noodzaak tot koppeling van verschillende ("vreemde") programma's en de hogere eisen gesteld door de groeiende gebruikersgroep, wel problemen op te leveren wat betreft de leesbaarheid en onderhoudbaarheid. De noodzaak van de verhoging van de kwaliteit van programmacode, en het modelleren in het algemeen, is aan

het einde van de tachtiger jaren onderkend.

Daarnaast werd steeds meer onderkend dat de kwaliteit van het hydrologisch onderzoek direct beïnvloed werd door de kwaliteit van de programmatuur. Zowel ten behoeve van de verhoging van de kwaliteit van computerprogrammatuur als van de kwaliteit van het hydrologisch modelonderzoek werd een zekere standaardisatie wenselijk geacht. De problematiek van de kwaliteit van de programmatuur wordt overigens niet alleen met betrekking tot de hydrologie, maar ook in andere vakgebieden onderkend (zie bijvoorbeeld VROM/DGM, 1990).

Voor wat betreft de kwaliteitseisen, kan onderscheid worden gemaakt tussen:
- de eisen die de gebruiker stelt, en
- de eisen die de ontwikkelaar stelt.

Eisen die de gebruiker stelt zijn o.a.:
- betrouwbaarheid;
- toepasbaarheid voor het specifieke probleem van de gebruiker (dat wil zeggen voldoende algemene toepasbaarheid en aanpasbaarheid);
- goede handleiding;
- overdraagbaarheid (naar andere computers, wat met name een probleem kan zijn bij de grafische toepassingen);
- koppelingsmogelijkheden:
  . uitvoer van andere programma's kan eenvoudig gebruikt worden als invoer;
  . de uitvoer kan gemakkelijk gebruikt worden als invoer voor andere programma's;
- efficiëntie in gebruik c.q. gebruikersvriendelijkheid waarbij genoemde koppelingsmogelijkheden belangrijk zijn maar ook de volgende eigenschappen:
  . opsporing en signalering van fouten;
  . gebruikersvriendelijke invoer;
  . duidelijke presentatie van de resultaten, direct via tabellarische en grafische uitvoer of indirect via een algemeen grafisch programma of spreadsheet;
  . commentaar in de invoerbestanden is mogelijk, bij lange invoerbestanden niet alleen in een paar regels aan het begin, maar ook tussendoor.

De eisen van de ontwikkelaar betreffen onder meer:

De eisen van de ontwikkelaar betreffen onder meer:

- het beschikbaar zijn van programmastructuur-diagrammen voor diverse niveaus;
- voldoende commentaar in programmacode;
- een beschrijving van alle variabelen in het programma.

Om praktische redenen zal de informatie die de ontwikkelaar nodig heeft vaak niet in publiceerbare vorm gegoten kunnen worden. Het zal doorgaans interne rapporten betreffen.

## 1.2 Instelling werkgroep

De bovenstaande problematiek is ten dele reeds in 1976 in het Klein Comité van de Commissie voor Hydrologisch Onderzoek van TNO (CHO-TNO) aan de orde gesteld, waarna de "Ad hoc groep Grondwatermodellen en Computerprogrammatuur" werd ingesteld. Deze groep had als taak de in Nederland beschikbare computerprogrammatuur en de op dat gebied bestaande verlangens te inventariseren (CHO-TNO, 1978). Vervolgens is in 1979 de Contactgroep Grondwatermodellen ingesteld met als doelstelling het bevorderen van de samenwerking bij de ontwikkeling en toepassing van grondwatermodellen ten dienste van het grondwaterbeheer als totaliteit (CHO-TNO, 1982).

Aan het einde van de tachtiger jaren begon men zich bij vele instanties te realiseren dat er sprake was van een structureel gebrek aan uniformiteit en duidelijkheid van de diverse programma's, onder meer met betrekking tot de in computerprogramma's voorkomende variabelen. Ook de afstemming tussen de programma's onderling liet te wensen over. Bij enkele instanties werd zelfs gewag gemaakt van een 'software crisis'. Het verzoek van een van de leden van de Commissie voor Hydrologisch Onderzoek, waarin de desbetreffende problematiek aan de orde werd gesteld, was de aanleiding voor het Klein Comité om de Werkgroep Standaardisatie Naamgeving Modelvariabelen in te stellen, te weten per 15 juni 1989.

Na een verkenningsperiode was de werkgroep echter tot de conclusie gekomen dat de naamgeving van variabelen slechts een onderdeel is van de veelomvattende problematiek van de kwaliteit van de computerprogrammatuur op het gebied van hydrologie. Tegen deze achtergrond werd het Klein Comité door de Werkgroep vervolgens verzocht de taakstelling van de Werkgroep dienovereenkomstig te verruimen. Het Klein Comité is met dit verzoek akkoord gegaan en wijzigde de taakstelling tot "het opstellen van richtlijnen met betrekking

tot de ontwikkeling van computerprogrammatuur op het gebied van de hydrologie". De naam van de groep is om deze reden aangepast: Werkgroep Richtlijnen Computerprogrammatuur Hydrologie (RCPH).

Bij de samenstelling van de werkgroep werd gestreefd naar een representatieve vertegenwoordiging van instanties op het gebied van de hydrologie. De leden van de werkgroep zijn vermeld in Bijlage A.

## 1.3 Doel van het rapport

Het rapport heeft ten doel aanbevelingen te presenteren met betrekking tot computerprogrammatuur op het gebied van de hydrologie. Het ter harte nemen van deze aanbevelingen zal leiden tot een betere kwaliteit van de programmatuur en een efficiëntere omgang met programma's mogelijk maken.

De Werkgroep RCPH pretendeert niet met nieuwe inzichten te komen, maar hoopt een bruikbaar overzicht te hebben geproduceerd van belangrijke aspecten bij het ontwikkelen van hydrologische computerprogrammatuur.

Dit rapport is primair geschreven bezien vanuit de praktijk van hydrologie, niet vanuit de optiek van professionele software-engineering. Dit houdt in dat in het rapport op een aantal plaatsen wellicht iets andere terminologie wordt gebezigd dan in de bestaande literatuur voor software ontwikkeling (normen, etc.) gebruikelijk is.

Het rapport is bestemd voor zowel programmeurs van complete programma's als voor tussen- en eindgebruikers. Van de lezer wordt verwacht dat deze beschikt over enige basiskennis van het programmeren. Beginnende programmeurs kunnen uit het rapport normen ontlenen voor een te hanteren programmeerstijl, meer ervaren programmeurs kunnen hun programmeerstijl toetsen en mogelijk verbeteren. Het rapport is niet geschreven als blauwdruk voor te ontwikkelen programma's. Tijdens het schrijven bleek namelijk dat er vele wegen naar Rome leiden.

De hoofdstukken zijn zoveel mogelijk voorzien van eenvoudige voorbeelden en een kort resumé van de aanbevelingen aan het eind. De aanbevelingen zijn ook samengevat op een los bijgevoegde lijst (checklist).

## 1.4 Leeswijzer

Hoofdstuk 2 gaat in op een aantal standaard ontwikkelmethodieken voor software in het algemeen en de bruikbaarheid hiervan binnen de hydrologie. Hoofdstuk 3 behandelt de interne structurering van computerprogramma's. Het gestructureerd programmeren en de modulaire opbouw van programma's fungeert als rode draad die door het hele rapport loopt. In hoofdstuk 4 komt de naamgeving van programma-onderdelen aan bod, waaruit ook de modulaire opbouw duidelijk wordt. Hoofdstuk 5 handelt over de naamgeving van programmavariabelen, waarbij uitgegaan wordt van het gebruik van stamnamen en achtervoegsels. Als uitgebreid voorbeeld wordt een deel van de Verklarende Hydrologische Woordenlijst voorzien van namen voor corresponderende variabelen in een programma. Hoofdstuk 6 behandelt verschillende manieren waarop databestanden gestructureerd aangemaakt en gewijzigd kunnen worden. In hoofdstuk 7 wordt besproken hoe de leesbaarheid van de broncode van een programma bevorderd kan worden door een zorgvuldige keuze van de layout van de broncode. Hoofdstuk 8 gaat in op een aantal programmeeraspecten (specifiek voor FORTRAN 77), die de kwaliteit van de programmatuur beter of juist slechter kunnen maken. In hoofdstuk 9 komt de problematiek van de foutencontrole en het genereren van foutmeldingen aan de orde. Hoofdstuk 10 gaat in op het organiseren van het testen van een programma en behandelt enige testmethoden. Hoofdstuk 11 belicht de interne documentatie binnen programma's. Hoofdstuk 12 geeft een overzicht van de opbouw van programmahandleidingen en behandelt puntsgewijs de onderwerpen die in een handleiding aan de orde moeten komen. In hoofdstuk 13 worden de conclusies beschreven die de werkgroep heeft geformuleerd inzake het algemene probleem van de kwaliteit van software. Verder worden de conclusies met betrekking tot de specifieke problematiek van dit rapport gegeven. Tenslotte volgt een aantal aanbevelingen.

In het rapport worden een aantal termen gebruikt die mogelijk aanleiding kunnen geven tot verwarring. Voor de goede orde volgen hier enkele in dit rapport gehanteerde definities:

computerprogramma : een geheel van instructies (broncode) waarmee bij gegeven invoer een bepaald proces kan worden gemodelleerd (de voorbeelden van broncode die in dit rapport gegeven worden zijn allen geschreven in FORTRAN);

| | |
|---|---|
| model | : een computerprogramma samen met invoergegevens. Hiermee wordt een bepaalde concrete situatie gemodelleerd; |
| module | : een functionele eenheid in algemene zin van een computerprogramma, bijvoorbeeld het verzorgen van invoer of het uitvoeren van berekeningen; |
| routine | : kleinst mogelijke eenheid in een computerprogramma (in FORTRAN een "SUBROUTINE" of een "FUNCTION"). |

Bij de voorbeelden en richtlijnen in dit rapport wordt uitgegaan van FORTRAN als programmeertaal. Dit sluit aan bij de huidige praktijk, hoewel gesteld kan worden dat ook andere talen hun intrede gedaan hebben in het hydrologische onderzoekveld.

# 2 ONTWIKKELMETHODIEKEN

## 2.1 Inleiding

Het ontwikkelen van computerprogramma's begint meestal niet bij het opschrijven van computercode, evenzo houdt de ontwikkeling niet op als de computercode éénmaal geschreven is. In feite is het uiteindelijk programmeren van een probleem slechts een betrekkelijk klein gedeelte uit een heel traject. Dit traject wordt de systeemontwikkeling genoemd. Bij systeemontwikkeling staat het zoeken en definiëren van problemen, het ontwerpen en realiseren van oplossingen en omgaan met die oplossingen centraal. Eigenlijk gaat het schrijven van computerprogramma's altijd gepaard met systeem-ontwikkeling; impliciet of expliciet worden eisen geformuleerd, problemen gedefinieerd, ontwerpen gemaakt etc. Bij het ontwikkelen van grote informatiesystemen worden de verschillende stappen in het ontwikkelproces systematisch doorlopen, waarbij iedere stap op een vooraf vastgestelde wijze wordt geëvalueerd. Dit opsplitsen van het systeem-ontwikkelproces in geformaliseerde stappen wordt een standaard systeemontwikkel-methodiek genoemd.

Alhoewel dit hoofdstuk primair de ontwikkeling van programmatuur behandelt, speelt hierbij ook de gebruiker een belangrijke rol. Deze werkt in het begin van het ontwikkeltraject mee aan de opstelling van het pakket van eisen, te stellen aan de programmatuur. De behoeften van de gebruiker, en de ervaringen die de gebruiker met diverse programma's heeft, dienen grondig te worden geïnventariseerd teneinde een optimaal functionerend produkt te kunnen ontwikkelen.

Zoals gezegd, de standaard systeemontwikkelmethodieken zijn ontstaan en hebben hun nut bewezen, bij het ontwikkelen van grote systemen waarbij sprake is van een compleet ontwikkelteam en een grote pluriforme gebruikersgroep. Bij het ontwikkelen van relatief kleine specifieke systemen vindt systeemontwikkeling doorgaans veel minder gestruc-tureerd plaats. Met name bij het ontwikkelen van wetenschappelijke programmatuur zoals binnen de hydrologie, met vaak een kleine uniforme gebruikersgroep en een programmeur/systeemanalist/systeemontwerper in één persoon verenigd, vindt systeem-ontwikkeling (ten onrechte) vaak op ad-hoc basis plaats. Gebruik van standaard methodieken kan echter ook bij dit soort systemen grote voordelen hebben, hoewel kritisch aan de verschillende stappen in het proces

gewicht toegekend moet worden.

## 2.2 Een standaard systeemontwikkelmethodiek (SDM)

Hoewel er in naam meerdere methodieken voorhanden zijn, werken ze allen globaal volgens hetzelfde concept. Een bekende en meest algemene methode is SDM (System Development Methodology). SDM beschrijft het systeemontwikkeltraject volgens de fasen van de levenscyclus van een informatiesysteem. Iedere fase wordt afgesloten, voordat er met de volgende fase wordt begonnen. Globaal zijn de volgende fasen te onderscheiden:

Fase 0: Informatieplanning

Hierin wordt bepaald welke informatiesystemen nodig zijn om de gestelde doelen van een organisatie te kunnen realiseren. Er wordt een haalbaarheidsonderzoek uitgevoerd. Dit onderzoek omvat: plan van aanpak, belangrijkste gebruikerseisen, ontwikkelingsstrategie enz.

Fase 1: Definitiestudie

Hierin worden de globale systeemeisen gedefinieerd. Bovendien worden de uitgangspunten, waaronder de fase eindprodukten, de gebruikerseisen en richtlijnen voor de ontwerpfase vastgelegd. Tevens komt in deze fase aan de orde de financiën en tijdsplanning.

Fase 2: Basisontwerp

De architectuur van het systeem wordt gedefinieerd, zodat het voldoet aan de eisen die zijn vastgelegd in de definitiestudie. Op basis hiervan wordt het gehele ontwerp met zijn subsystemen vastgelegd.

Fase 3: Detailontwerp

Per subsysteem worden de systeemeisen verder verfijnd tot een niveau waarop deze kunnen worden vervaardigd. In feite is er aan het eind van deze fase sprake van een pakket pseudocode.

Fase 4: Realisatie

In deze fase wordt de computercode geschreven en wordt de vervaardigde programmatuur getest, op basis van de eerder geformuleerde specifieke een algemene systeemeisen. (Voor technisch-wetenschappelijke toepassingen worden meestal fase 3 en fase 4 samengenomen.)

Fase 5: Invoering

Hierbij wordt het systeem door de ontwikkelaar losgelaten en vindt het systeem zijn toepassing bij de vooraf gedefinieerde gebruiker. Fouten en afwijkingen naar aanleiding van acceptatietesten worden gecorrigeerd.

Fase 6: Gebruik en beheer

In de laatste fase is het systeem in gebruik bij externe gebruikers en moet het systeem in gebruik blijven. Hierbij moet voortdurend ingespeeld worden op nieuwe wensen en ontwikkelingen.

## 2.3 Praktisch gebruik

De hoofdfasen van SDM zijn eigenlijk voor ieder systeem toepasbaar. In feite is SDM een pleidooi om eerst na te denken en dan pas te doen (te programmeren) en om vervolgens betrokken en waakzaam te blijven. Iets dat overigens niet alleen de ontwikkeling van informatiesystemen aangaat! Het verschil tussen toepassing van SDM bij bijvoorbeeld grote administratieve systemen en toepassing voor technisch-wetenschappelijke systemen, ligt voornamelijk in de invulling van de verschillende fasen. Bij het eerste soort systemen moet zeer veel aandacht besteed worden aan gebruikersaspecten, management-aspecten, sociale aspecten, kosten-/batenanalyses etc. Dit zijn vaak ook de drijfveren voor het starten van fase 0, de informatieplanning. Inhoudelijk zijn er relatief weinig vraagpunten, problemen op het gebied van bestandsorganisatie daargelaten. Vaak is het doel, de organisatie efficiënter te laten draaien. Bij technisch-wetenschappelijk problemen zijn het vooral inhoudelijke aspecten die voorop staan. Technisch-wetenschappelijke programmatuur moet vaak niet op de eerste plaats een proces efficiënter laten verlopen, maar het moet mogelijk zijn om een bepaald inhoudelijk probleem afdoende op te lossen. Dat neemt niet weg dat ook vooraf een

prijs-/prestatie-overweging gemaakt moet worden, die door een beoogd gebruiker gedefinieerd wordt.

SDM zal bij technisch-wetenschappelijke systemen vooral gebruikt worden om op het juiste moment de juiste vragen te stellen en om een probleem gestructureerd stapsgewijs te verfijnen. Het grote voordeel daarbij is dat op logische tijdstippen tijdens het proces verslag kan worden gedaan van de keuzes die aan de orde zijn. Een grotere groep betrokkenen (in dit geval deskundigen) kan zo "meegroeien" in de programmatuur en kan zo nodig sturen. Zonder het formaliseren van de stappen in de systeemontwikkeling is het systeem tijdens het ontwikkelproces voor minder mensen toegankelijk. Met een beetje geluk kan dan op basis van een goede documentatie een oordeel over het complete systeem gegeven worden, sturing is echter nauwelijks meer mogelijk.

Tenslotte enkele kritische kanttekeningen:

- Indien de ontwikkeling van een complex technisch-wetenschappelijk programma niet door één en dezelfde persoon plaatsvindt (combinatie van materiedeskundigheid en systeembouw), gaat relatief veel tijd zitten in de informatie-overdracht van de materiedeskundige naar de systeemanalist en programmeur. Hierdoor kan de ontwikkeling traag en duur worden. In het algemeen geldt dan dat ernaar gestreefd moet worden de samenwerkingsverbanden tussen materiedeskundigen en systeem-ontwikkelaars zo nauw mogelijk te houden. Om dezelfde reden zouden de informatielijnen over zo min mogelijk mensen moeten lopen.
- Het ontwikkelen van een prototype model is wat moeilijker in te passen in het strak gedefinieerd schema van SDM. Bij de ontwikkeling van hydrologische modellen wordt vaak eerst een eenvoudig model ontwikkeld. Vervolgens wordt dit eenvoudig model toegepast, waarbij blijkt dat een aantal processen gedetailleerder of op een andere wijze beschreven moeten worden. Er wordt dan teruggesprongen naar de fase van detailontwerp of zelfs basisontwerp. In de definitiestudie kunnen de voorwaarden en mogelijkheden van dit soort "herhalingen" worden gesignaleerd en kan worden getracht dit tijdens de volgende fasen in te bouwen.

## 2.4 Resumé

- Breng systematiek aan in het ontwikkeltraject.
- Sluit zoveel mogelijk fasen af met een rapportage voor derden.
- Gebruik zo mogelijk een standaard-ontwikkelmethodiek zoals SDM.
- Vooral: denk na vóór het ontwikkelen van programma's en blijf na de ontwikkeling betrokken.

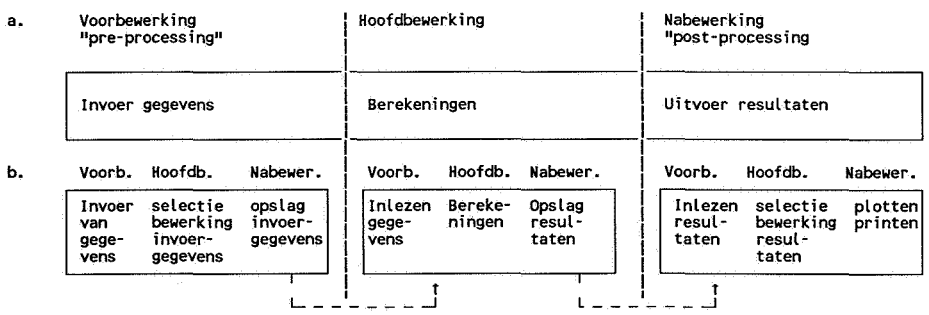
# 3 INTERNE STRUCTURERING

## 3.1 Inleiding

De hydrologische praktijk vraagt vaak om complexe computerprogramma's voor berekeningen en simulaties. Deze programma's kunnen alleen overzichtelijk gemaakt worden door een goede structuur aan te brengen. Hiertoe worden afzonderlijke taken onderscheiden, die binnen het programma verricht moeten worden. De hoofdtaken kunnen vaak weer verder opgesplitst worden in kleinere onderdelen. Op deze manier wordt een structuur van onderdelen gemaakt die elk een dusdanige omvang hebben dat ze zich in een redelijke hoeveelheid broncode laten vertalen. Door een verdeling in duidelijk afgebakende taken te maken, kan een programma gemakkelijker opgezet worden, wordt het gemakkelijk leesbaar en is het eenvoudiger later aanpassingen te maken, omdat snel duidelijk is waar in het programma een bepaalde taak verricht wordt.

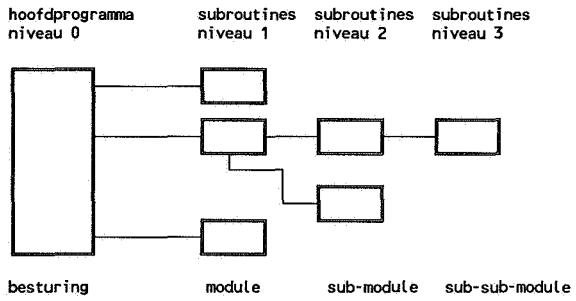
## 3.2 Modulaire indeling

Alle computerprogramma's bevatten dezelfde basiselementen: invoer van gegevens (voorbewerking), berekeningen (hoofdbewerking) en uitvoer van resultaten (nabewerking). Figuur 3.1 geeft een dergelijke indeling weer, respectievelijk voor een enkel (eenvoudig) computerprogramma en een complexer systeem van computerprogramma's.




Figuur 3.1 Basisstructuur voor probleemstelling
a) een enkel computerprogramma
b) de bewerkingen in een aantal computerprogramma's ondergebracht

De hiervoor genoemde ontleding kan gebruikt worden om een programma in onderdelen te verdelen. De hoofdonderdelen, welke modulen genoemd worden, zijn in dit geval voorbewerking, hoofdbewerking en nabewerking. Deze kunnen verder onderverdeeld worden in sub-modulen, welke op hun beurt weer uiteen kunnen vallen in sub-sub-modulen enzovoorts (zie Figuur 3.2).




Figuur 3.2 Modulaire opbouw van een computerprogramma

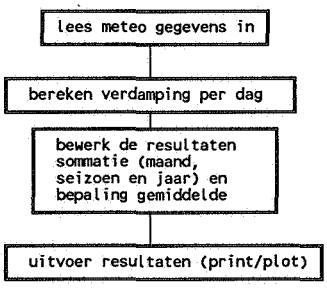
De voordelen van de modulaire opbouw zijn:

- complexe problemen zijn opgesplitst in eenvoudige deelproblemen;
- groepeer machine/operating-system/compiler afhankelijke functies;
- ontwerpen, coderen en testen kan per onderdeel gebeuren;
- modules voor vaak voorkomende taken behoeven slechts een keer gemaakt te worden en kunnen vervolgens voor verschillende programma's of programma-onderdelen gebruikt worden.

In de praktijk komt het er op neer dat het hoofdprogramma de besturing regelt door het aanroepen van routines. In het hoofdprogramma staan dan ook alleen de aanroepen van modules van niveau 1. In elke module wordt een aantal functioneel afgebakende handelingen uitgevoerd. Een module met een complexe verzameling handelingen wordt opgesplitst in sub-modulen (niveau 2). Indien noodzakelijk om het geheel overzichtelijk te houden roepen dergelijke routines van niveau 2 op hun beurt routines van niveau 3 (sub-sub-modulen) aan die nog enger afgebakende taken verrichten.

## 3.3 Voorbeeld: Programma EPOT (Staring Centrum, Wageningen)

Het berekenen van de verdamping is als voorbeeld genomen. Het programma EPOT berekent zowel de referentie grasverdamping (Makkink-verdamping) als de open water verdamping (Penman-verdamping). De berekening wordt telkens voor een dag uitgevoerd en de resultaten worden gesommeerd per maand, seizoen en jaar. Het stroomschema is gegeven in Figuur 3.3. Dit stroomschema is vertaald in de structuur van het programma EPOT, die in Figuur 3.4 gegeven is.




Figuur 3.3 Stroomschema voor berekening van verdamping

```
programma EPOT

call INIT       initialiseer constanten en open bestanden
call READ       lees meteorologische gegevens per dag
call CHECK      controleer invoergegevens

call REF        bereken Makkink-verdamping
call PENMAN     bereken open water verdamping

call SOM        sommeer verdamping en bepaal gemiddelde
call PRINT      schrijf resultaten naar printbestand
call PLOT       schrijf resultaten weg in plotbestand
```

Figuur 3.4 Structuur van programma EPOT

## 3.4 Moeilijkheden bij bestaande programmatuur

Bij het aanpassen van een bestaand programma ontstaat het dilemma of de bestaande (gebrekkige) structuur aangehouden moet worden of dat met een betere structuur gewerkt wordt. De keuze voor één van de mogelijkheden hangt af van de specifieke situatie, waarbij onder meer de volgende overwegingen een rol spelen:

- gaat het om een tijdelijke verandering voor één specifieke toepassing;
- kan de verandering in een afzonderlijke nieuwe (sub-)module ondergebracht worden;
- hoe gebrekkig is de bestaande structuur en hoe moeilijk is het deze te verbeteren.

Als ingrijpende wijzigingen nodig zijn in een slecht gestructureerd en daardoor onoverzichtelijk programma verdient het aanbeveling eerst de nieuwe structuur te ontwerpen en vervolgens aan de hand van stroomschema's te controleren of het programma de oorspronkelijke taken nog naar behoren uitvoert en pas daarna de wijzigingen te programmeren. Bij de aanpassing van bestaande programmatuur dienen de in de oorspronkelijke broncode aangebrachte wijzigingen door middel van commentaarregels grondig te worden gedocumenteerd.

Als het bestaande programma gestructureerd is opgezet en de wijziging niet groot is, is het beter om bij de bestaande structuur aan te sluiten.

## 3.5 Moeilijkheden bij koppeling

Indien programma's op codeniveau gekoppeld moeten worden, kunnen soortgelijke moeilijkheden optreden als bij het aanpassen van programmatuur. Vaak is het ene programma anders van structuur, heeft een andere systematiek van variabele naamgeving etc. dan het andere programma. In dergelijke gevallen is het raadzaam om "het contact" tussen beide programma's zoveel mogelijk te beperken tot speciaal daarvoor ontworpen modules die als interface fungeren. Indien het onontkoombaar is om in de bestaande programmatuur wijzigingen aan te brengen, moet dit duidelijk en opvallend vermeld worden.

## 3.6 Resumé

- Splits de probleemstelling op in deelproblemen met duidelijk afgebakende taken.
- Kies voor één programma of een aantal afzonderlijke programma's.
- Kies een aantal niveaus voor de structuur.
- Bij het aanpassen van bestaande programma's: weeg aansluiten bij bestaande structurering af tegen het gebruiken van een betere structuur.

# 4 NAAMGEVING PROGRAMMA-ONDERDELEN

## 4.1 Inleiding

Veel hydrologische computerprogramma's zijn zo omvangrijk (wat de hoeveelheid broncode betreft) en dermate ingewikkeld dat ze in onderdelen zijn gesplitst.

Als elk onderdeel een duidelijk afgebakende taak vervult, is het eenvoudig het overzicht over het gehele programma te bewaren. Dit geldt in de fase van het programmeren, maar nog veel sterker bij het onderhouden en aanpassen van het programma. Veranderingen moeten namelijk vaak gebeuren als het programmeren niet meer vers in het geheugen ligt en vaak door anderen dan de makers van het programma. De namen van de onderdelen kunnen de plaats van die onderdelen in het geheel van het programma aangeven.

In het algemeen dient de naamgeving van de onderdelen het streven naar overzichtelijkheid van het programma (programmastructuur) te ondersteunen.

## 4.2 Naamgeving onderdelen

Het ligt voor de hand de naam van elk onderdeel te enten op de taak die het desbetreffende onderdeel verricht. Als een programma zodanig is, dat de plaats van een onderdeel in het programma niet duidelijk is uit de taak (alleen), dan verdient het aanbeveling (ook) de structuur van het programma te weerspiegelen in de namen van de onderdelen.

In de structuur van een programma worden verschillende niveaus onderscheiden. Hoeveel niveaus onderscheiden worden is afhankelijk van de omvang en specifieke eigenschappen. Een verdeling van de structuur op drie niveaus kan er als volgt uit zien, zie bijvoorbeeld Figuur 3.2:

1 module (niveau 1);
2 sub-module (niveau 2);
3 sub-sub-module (niveau 3).

Het hoofdprogramma (niveau 0) roept een aantal modulen (niveau 1) aan. Een module bevat op haar beurt weer sub-modulen (niveau 2). De onderdelen van de sub-modulen, sub-sub-modulen (niveau 3), zijn routines in dit geval. Een routine is de kleinste eenheid in de

broncode (voor FORTRAN een SUBROUTINE of een FUNCTION). Voor het aanbrengen van structuur in een programma wordt verwezen naar hoofdstuk 3.

Hoezeer de taak en de plaats in de structuur doorklinkt in de namen van de modulen hangt ervan af hoe belangrijk de structuur is voor het doorgronden van het programma en in hoeverre het mogelijk is de taken te identificeren voor de individuele modulen.

## 4.3 Namen van bestanden behorende bij een programma

Verbonden met de naamgeving van de onderdelen van een programma zijn de namen van de bestanden, die gebruikt worden om een run versie van het programma te maken. Hierbij zijn verschillende soorten bestanden te onderscheiden:

1 bestanden met broncode;
2 speciale bestanden die aanwezig moeten zijn bij het compileren of draaien van het programma;
3 hulpbestanden, zoals een bestand dat aangeeft welke bestanden er gelinkt moeten worden om de run versie te creëren.

Het 'operating system' DOS, dat op veel PC's gebruikt wordt, staat bestandsnamen toe van maximaal 8 lettertekens, met een extensie van maximaal 3. De extensie kan dan gebruikt worden om de soort van het bestand aan te geven:

a voor een bestand met broncode kan een extensie gebruikt worden, die de taal aangeeft (.FOR voor FORTRAN of .C voor C bijvoorbeeld) of de compiler (zoals .F5P voor versie 5 van de FORTRAN-compiler van Perkin Elmer);
b voor een zogenaamd INCLUDE bestand, dat in FORTRAN gebruikt kan worden voor onder meer COMMON BLOCKS kan de extensie .INC gebruikt worden. Voor een file met gebruikersinformatie die bij het laden van het programma wordt gelezen, zou de extensie .USR gekozen kunnen worden;
c voor een bestand dat gebruikt wordt om de files voor het linken te selecteren ligt de extensie .LNK voor de hand.

De 8 lettertekens voor de bestandsnaam worden gebruikt om het programma en/of de inhoud van het bestand aan te geven. Een mogelijkheid is om elke routine op te slaan in een apart bestand (in verband met bijvoorbeeld een 'make-facility', een hulpprogramma

waarmee de runversie automatisch aangepast wordt als er iets in de broncode veranderd wordt). In ANSI FORTRAN 77 kunnen maximaal 6 lettertekens voor de namen van routines gebruikt worden. De naam van het bestand kan dan gemaakt worden door de twee letters die het programma aanduiden te laten volgen door de naam van de routine. Een andere mogelijkheid is om alle routines van een module (of anderszins verwante routines) op te slaan in één bestand. De naam van het bestand kan dan weer op eenzelfde manier gemaakt worden: twee letters voor het programma en maximaal zes letters die de module aanduiden. Kleine programma's (zie voorbeeld 1, hieronder) worden bij voorkeur in één bestand opgeslagen.

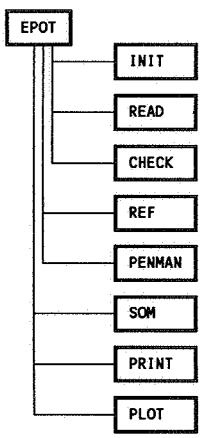
## 4.4 Voorbeelden van naamgeving van programma-onderdelen

In het navolgende wordt een aantal voorbeelden van naamgeving voor programmamodulen gegeven, waarbij de programma's steeds complexer worden. Dit uit zich in het feit dat er steeds meer structuur doorklinkt in de namen van de onderdelen.

In het eerste voorbeeld (voorbeeld 1) zijn de namen puur op grond van taak gegeven, in de volgende twee voorbeelden zijn steeds meer delen onderscheiden in het programma en is de structuur in meer niveaus in de namen verwerkt.

### Voorbeeld 1 Programma EPOT (Staring Centrum, Wageningen)

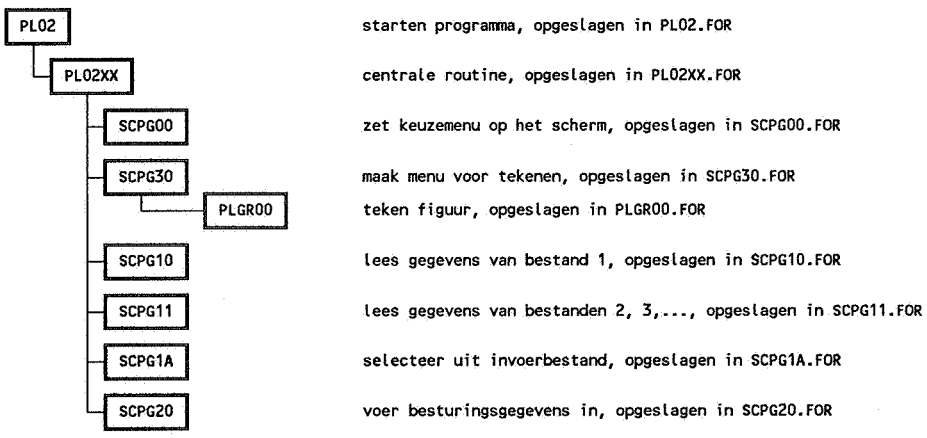
Het hoofdprogramma draagt de naam van het programma en de routines die aangeroepen worden hebben de namen van de taken die ze verrichten:




gehele programma
opgeslagen in EPOT.FOR

Voorbeeld 2 Programma AQ-PL02 (RIVM, Bilthoven)

Het programma AQ-PL02 is een grafisch programma dat deel uitmaakt van het AQ-pakket, een pakket ten behoeve van de analyse van grondwatervraagstukken. De subroutines die het programma AQ-PL02 aanroept behoren tot de grote verzameling routines die door diverse programma's van het pakket gezamenlijk gebruikt worden. Het aantal is zo groot dat geen zinvolle unieke afkortingen gemaakt konden worden. Derhalve zijn de routines genummerd. Alle routinenamen bestaan uit 6 tekens, de bestandsnaam is gelijk aan routinenaam. Het nummer wordt, waar nodig, gebruikt in posities 3 tot en met 6. Routines die specifiek bij PL02 behoren beginnen met de naam van het programma; de namen van algemene routines beginnen met SC (met betrekking tot het scherm, AQ-programma's werken interactief) of met PL (voor het plotten van een figuur op een plotter).



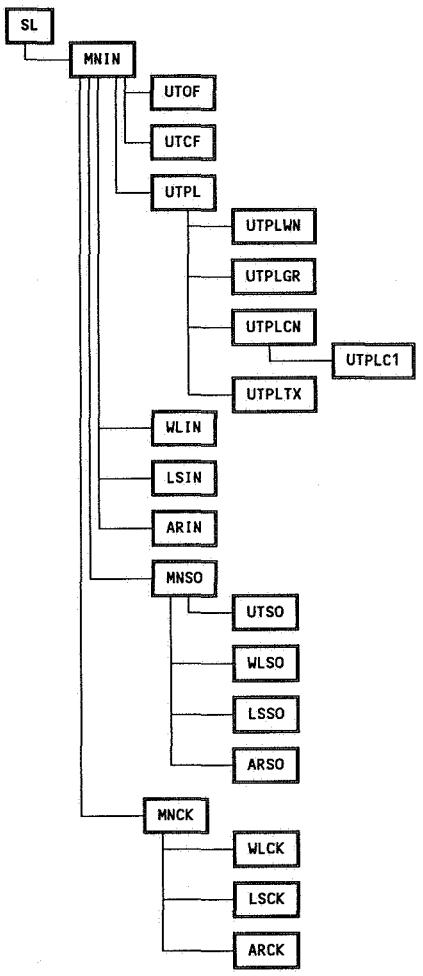

Voorbeeld 3 Programma SL (O.D.L. Strack, Minnesota, VS)

Het programma SL is een programma voor het modelleren van grondwaterstroming in een enkel watervoerend pakket met behulp van analytische elementen. In dit programma zijn de volgende modulen onderscheiden: main (MN, module die de andere modules samenbundelt), wells (WL, module van putten), line-sinks (LS, module van lijnputten), area-sinks (AR, module voor oppervlakteputten). De sub-modules zijn: input (IN, sub-module voor invoer), root (RT, module die algemene taken uitvoert), solution (SO, sub-module voor het berekenen van de onbekenden), check (CK, sub-module voor het controleren van de oplossing en het genereren van uitvoer).

De naam van een specifieke routine ziet er als volgt uit: SSPPRR, waarin:

- SS twee lettertekens zijn die de module aangeven;
- PP staat voor twee lettertekens van de sub-module;
- RR de routine met twee lettertekens specificeert.

Voor een algemene routine worden de letters SS vervangen door een code van twee letters, bijvoorbeeld 'UT van 'utility". De letters UT worden gevolgd door een aantal andere letters die de functie van de routine aanduiden.




starten programma en dimensionering
centrale invoer
openen van bestand
maken van bestand
maken van figuur met contourlijnen
definiëren coördinaten van hoekpunten van figuur
openen grafisch scherm
tekenen van contourlijnen
tekenen van 1 contourlijn
afsluiten grafisch scherm en terug naar text scherm
invoer van putten
invoer van lijnputten
invoer van oppervlakteputten
centrale deel van oplossen
routine voor oplossingen stelsel vergelijkingen
bijdrage putten aan matrix
bijdrage lijnputten aan matrix
bijdrage oppervlakteputten aan matrix
centrale controle en uitvoer
uitvoer putten
uitvoer lijnputten
uitvoer oppervlakteputten

```
SL      opgeslagen in SL.FOR
MNIN    opgeslagen in SLMNIN.FOR
MNSO    opgeslagen in SLMNSO.FOR
MNCK    opgeslagen in SLMNCK.FOR

WLIN    opgeslagen in SLWL.FOR
WLSO    opgeslagen in SLWL.FOR
WLCK    opgeslagen in SLWL.FOR
```

```
ARIN      opgeslagen in SLARIN.FOR
ARSO      opgeslagen in SLARSO.FOR
ARCK      opgeslagen in SLARCK.FOR

UTOF      opgeslagen in UTFILE.FOR
UTCF      opgeslagen in UTFILE.FOR

UTPL      opgeslagen in UTPLOT.FOR
UTPLWN    opgeslagen in UTPLOT.FOR
UTPLGR    opgeslagen in UTPLOT.FOR
UTPLCN    opgeslagen in UTPLOT.FOR
UTPLC1    opgeslagen in UTPLOT.FOR
UTPLTX    opgeslagen in UTPLOT.FOR
UTSO      opgeslagen in UTSOLVE.FOR
```

Schema van programma SL met namen en functie van de onderdelen

## 4.5 Resumé

- Vergroot overzichtelijkheid van een programma door een logische verdeling in onderdelen.
- Geef waar mogelijk de onderdelen namen die de taak van het onderdeel weerspiegelen.
- Als de structuur van het programma ingewikkeld is: voeg aan de naam een aanduiding toe van de plaats van de routine in het programma.

# 5 NAAMGEVING PROGRAMMAVARIABELEN

## 5.1 Inleiding

Het benoemen van variabelen in een computerprogramma is bijzonder belangrijk. Een goede systematiek van benoeming van variabelen maakt een programma leesbaar en overdraagbaar. Al is een programma nog zo goed gestructureerd en gedocumenteerd, als er geen goed systeem van variabelenamen is toegepast, is de broncode (die de uiteindelijke uitwerking van een oplossing is) niet goed leesbaar en kunnen wijzigingen moeilijk doorgevoerd worden. Het bedenken van een goed systeem is moeilijk en kan tijdrovend zijn, maar blijkt vaak meer dan de moeite waard. Als er een goed systeem bedacht is, moet dit ook bij de documentatie gevoegd worden, zodat de gebruiker de logica van de programmamaker kan doorgronden. Dit heeft als voordeel dat de gebruiker de betekenissen voor een groot deel direct kan begrijpen, zonder iedere keer de lijst van variabelen te moeten raadplegen. Dit neemt niet weg dat ook altijd een volledige lijst van gebruikte variabelen bij de documentatie moet worden gevoegd, zodat deze, wanneer nodig, kan worden geraadpleegd.

## 5.2 Systeem

Een sluitend concept voor een systeem van variabelenamen is moeilijk te geven. Veel hangt af van de aard van de programmatuur (onderwerp, complexiteit), de mogelijkheden van de programmeertaal (gereserveerde letters, maximaal aantal toegestane posities) etc. Dat iedereen hetzelfde systeem gebruikt is ook niet het belangrijkste. Het belangrijkste is dat iedereen een logisch en gedocumenteerd systeem gebruikt. Het kan soms zelf voordelig zijn om een, op eigen situatie toegesneden systeem te gebruiken, in plaats van krampachtig aan een standaard systeem vast te houden.

Alhoewel geen blauwdruk gegeven kan worden, kunnen wel enige conventies als richtlijnen dienen:

1 Selecteer veel gebruikte algemene grootheden en ken hieraan een letter of lettercombinatie toe. Grootheden als volumestroom (Q), hoogte (H), weerstand (R), straling (RD) komen zo vaak in verschillende afgeleide grootheden voor, dat beter iedere variabelenaam van dit type begonnen kan worden met een min of meer

gereserveerde letter(s). Om het overzicht te behouden is het raadzaam het aantal gereserveerde beginletters te beperken tot circa 10.

2 In de al of niet gereserveerde beginletter moet zo mogelijk het data-type tot uitdrukking komen. In de FORTRAN 77-standaard met betrekking tot impliciete declaraties begint een integer variabele met letter I-, J-, K-, L-, M- of N- dus I-N-integer, en een real variabele met één van de overige letters.

3 Vorm voor alle gebruikte grootheden zogenaamde stamnamen van 3 tot 5 letters. De stamnamen kunnen aangevuld worden door achtervoegsels. Houd bij de stamnamen rekening met de eerste twee conventies. Bij de naamgeving is de herkenbaarheid van de variabelen het belangrijkste, niet het al of niet consequent zijn. Voorkom zoveel mogelijk het gebruik van een karakteristieke beginletter als die niet de vastgestelde betekenis heeft, tenzij duidelijk uit het programmaverband die betekenis blijkt.

4 Selecteer veel gebruikte toevoegingen en ken hieraan twee-letterige afkortingen aan toe. Voorbeelden: MN voor minimum, MX voor maximum, SM voor som, AV voor gemiddeld.

5 Als de computer(-taal) het toelaat is het raadzaam om de onderscheiden delen van de variabele naam te scheiden door bijvoorbeeld een underscore ( _ ). Zo kan HPHREAVMN (gemiddelde laagste phreatische grondwaterstand) geschreven worden als HPHRE_AV_MN of H#PHRE_AV_MN. De laatste notatie heeft als voordeel dat meteen opvalt dat de letter H een gereserveerde beginletter is.

6 Gebruik als DO-loop variabelen (indexvariabelen) bij voorkeur minimaal twee letterige codes. Begin indexvariabelen steeds met I-, J-, K- of L-, gevolgd door het increment, bijvoorbeeld IP2 (positief met stap 2), JN (negatief met stap 1) enzovoorts.

7 Variabelen die een aantal (number) aangeven, beginnen met de de letter N-, bijvoorbeeld NTIM, NEL en NLAY. Dit geldt dus ook voor aantallen in een DO-loop.

8 Bij array dimensie: NTIMMX, NELMX, etc.

9 Vermeld in de documentatie:
   a de gehanteerde systematiek:
      - een lijst met min of meer gereserveerde beginletters en de voornaamste uitzonderingen daarop;
      - een volledige lijst met stamnamen en de hierbij gebruikte conventies;
      - een volledige lijst met achtervoegsels;
   b een volledige lijst met variabelen thematisch en alfabetisch gerangschikt.

## 5.3 Voorbeeld: de Verklarende Hydrologische Woordenlijst

Als voorbeeld van een vocabulaire van variabelenamen is een deel van de Verklarende Hydrologische Woordenlijst (CHO-TNO, 1986) omgezet naar een variabelenlijst geschikt voor een hydrologisch computerprogramma. In bijlage B wordt een deel van de woordenlijst inclusief programmavariabelen, weergegeven voor de rubrieken Atmosferisch Water, Water in de Onverzadigde Zone, Water in de Verzadigde Zone, Oppervlaktewater en Diversen. Bij de lijst is uitgegaan van stamnamen van 3 en 5 letters. De stamnamen van 3 letters kunnen gebruikt worden in geval van conventionele (FORTRAN-)compilers waarbij slechts 6 characters per naam gebruikt mogen worden (stam + achtervoegsel).

De lijst is een goed voorbeeld hoe naamgeving aangepakt en beschreven kan worden. Het is echter geen rigide lijst die in alle gevallen toegepast moet worden. Tijdens het samenstellen van de lijst is gebleken hoe moeilijk het is om een consequent systeem toe te passen, de gepresenteerde lijst is dus niet altijd consequent. Het toepassen van de conventies is echter ook niet bedoeld om iedere variabele 100% te determineren, maar slechts een hulpmiddel om een programma overzichtelijker en dus leesbaarder te laten maken.

De belangrijkste gebruikte conventies zullen onderstaand toegelicht worden.

Gereserveerde beginletters:

| Letter | Betekenis | Eenheid | Voorbeeld |
|---|---|---|---|
| D | diepte (t.o.v. mv) | cm, m | DGRWT (Depth GRoundWater Table), |
| | | | DDIFF (Depth DIFFerence) |
| E | verdamping | mm/d | EVPAN (PAN EVaporation), |
| | | | ESOIL (SOIL Evaporation) |
| F | (warmte-)flux | W/m² | FLATH (LATent Heat Flux), |
| H | hoogte (t.o.v. NAP) | m | HPHRE (PHREatic Head), |
| | | | HPRES (PRESsure Head) |
| P | druk | Pa | PTENS (TENSiometer Pressure), |
| | | | PHYDR (HYDRaulic Pressure) |
| PR | neerslag/-intensiteit | mm, m³, mm/d etc. | PRECI (PRECIpitation), |
| | | | PRDPH (PRecipitation DePtH) |
| Q | volumestroom/debiet | mm/d, m³/d | QVOLM (VOLuMe flux), |
| | | | QCAPR (CAPillary Rise flux) |
| R | weerstand | s/m, d | RCANY (CANopY Resistance), |
| | | | RDRNG (DRaiNaGe Resistance) |
| T | temperatuur | °C of K | TDEWP (DEW-Point Temperature), |
| | | | TWETB (WET-Bulb Temperature) |
| TI | tijd | s, d | TITRA (TRAvel TIme) |
| TH | vochtgehalte | - | THWLT (WiLTing point THeta), |
| | | | THFLD (FieLD capacity THeta) |
| X | X-coordinaat | m | XWELL (X-coordinate WELL), |
| | | | XCOOR (X-COORdinate) |
| Y | Y-coordinaat | m | YWELL (Y-coordinate WELL) |
| | | | YCOOR (Y-COORdinate) |
| Z | Z-coordinaat | m | ZWELL (Z-coordinate WELL) |

De stamnamen zijn in de meeste gevallen min of meer fonetisch vastgesteld, waarbij vaak enige klinkers geschrapt zijn en zoveel mogelijk de laatste (betekenisvolle) medeklinker gehandhaafd is.

Achtervoegsels:

| Achtervoegsel | Betekenis | |
|---|---|---|
| AC | actueel | (ACtual) |
| AV | gemiddeld | (AVerage) |
| CL | berekend | (CaLculated) |
| DF | verschil | (DiFference) |
| MN | minimum | (MiNimum) |
| MX | maximum | (MaXimum) |
| OB | gemeten | (OBserved) |
| PO | potentieel | (POtential) |
| PR | vorige | (PRevious) |
| RL | relatief | (ReLative) |
| SM | sommatie | (SuM) |
| XD | X-richting | (X-Direction) |
| YD | Y-richting | (Y-Direction) |
| ZD | Z-richting | (Z-Direction) |
| | (etc.) | |

## 5.4 Resumé

- Besteed veel aandacht aan het bedenken van een goed werkbaar systeem.
- Vorm voor basisgrootheden stamnamen bestaande uit 3 tot 5 letters. Reserveer voor veel voorkomende algemene grootheden één of twee beginletters.
- Selecteer twee letterige afkortingen voor veel voorkomende toevoegingen.
- Vermeld gebruikte conventies in de programmahandleiding, evenals een complete lijst van stamnamen en achtervoegsels.

# 6 DATABESTANDEN

## 6.1 Inleiding

Computerprogramma's hebben invoer nodig om te kunnen draaien. Bij veel technische programma's wordt de invoer gelezen uit databestanden die met behulp van een preprocessor of een editor zijn aangemaakt. De uitvoer van deze programma's wordt doorgaans weer naar databestanden weggeschreven.

Het verzamelen van gegevens en het schematiseren van deze gegevens tot invoer voor computerprogramma's kost veel tijd en geld. Het verdient daarom de voorkeur deze gegevens zo op te slaan dat deze zonder veel moeite door verschillende programma's (in verschillende modelstudies) kunnen worden gebruikt. Bij de opbouw van de databestanden dient hiermee rekening te worden gehouden.

Er bestaan verschillende typen van databestanden. De in de hydrologische programma's meest gebruikte bestanden zijn sequentiële geformatteerde en ongeformatteerde bestanden. In paragraaf 6.3 wordt globaal aangegeven wat één en ander inhoudt en wat de voor- en nadelen zijn. Sinds de opkomst van database management systemen (Oracle, dBase, Dataflex etc.) worden de benodigde databestanden steeds vaker binnen een database management systeem opgebouwd. In paragraaf 6.4 wordt hierop ingegaan.

## 6.2 Eisen te stellen aan databestanden

Een databestand is alleen bruikbaar voor verschillende programma's wanneer dit voldoet aan een aantal eisen:

- het bestand moet overzichtelijk zijn;
- het bestand moet eenvoudig te herstructureren zijn (uit te breiden, in te krimpen of anders te ordenen);
- de gegevens moeten (waar relevant) op een relatief eenvoudige wijze aangeboden kunnen worden aan een grafisch programma ten behoeve van grafische presentatie en visuele controle (bijvoorbeeld via een afzonderlijke hulpprogramma dat een bestand eerst converteert ten behoeve van een grafisch programma);

- de gegevens moeten in numerieke vorm leesbaar zijn (bijvoorbeeld ASCII-bestanden) of leesbaar gemaakt kunnen worden (bijvoorbeeld door middel van een hulpprogramma voor ongeformatteerde bestanden).

Aan de laatste eis wordt vaak wel voldaan. Veel programma's gebruiken ASCII-bestanden als in- en uitvoerbestanden. Als dat niet het geval is, zoals ongeformatteerde bestanden, worden speciale hulpprogramma's gemaakt, waarmee deze bestanden kunnen worden aangemaakt, gewijzigd en gevisualiseerd (tekst op scherm of print, plot).

Aan de overige drie eisen wordt vaak niet voldaan. De structuur van de datafile wordt vaak ad-hoc opgezet, afhankelijk van de aard van het probleem en de programmeerstijl van de ontwikkelaar. Met het gebruik van een database management systeem kan tegemoet worden gekomen aan alle vier eisen (zie 6.4).

## 6.3 Geformatteerde of ongeformatteerde databestanden

De kenmerkende eigenschappen van geformatteerde en ongeformatteerde bestanden worden hieronder globaal omschreven.

Geformatteerde bestanden:
Geformatteerde bestanden zijn ASCII-bestanden en zijn dus visueel leesbaar, bijvoorbeeld als printeruitvoer of op het scherm. De informatie in deze bestanden is verdeeld over regels en staat in een bepaalde volgorde opgeslagen, bepaald door het programma dat de bestanden gebruikt. Er bestaan twee manieren om geformatteerde bestanden op te bouwen, namelijk zogenaamde 'fixed format input/output' en 'free format input/output'. Bij de 'fixed format' methode heeft elk gegeven een vaste plaats op een regel, de tijdstap neemt bijvoorbeeld posities 21 tot en met 30 in op regel 3. Bij de 'free format' methode behoeft de positie op de regel niet meer vast te zijn. Informatie wordt in dit geval gescheiden door een komma of een spatie. Het aantal gegevens, op een regel en het regelnummer blijven nog wel van belang. Voordeel van een geformatteerd bestand is dat deze rechtstreeks door een professioneel programma, bijvoorbeeld spreadsheet, kan worden gelezen. Bovendien wordt de overzichtelijkheid bevorderd, hetgeen soms wel gepaard gaat met extra inspanningen bij de aanmaak van de bestanden.

Een ander voordeel is dat de geformatteerde bestanden op elke computer door elk programma als regel gelezen kunnen worden, ongeacht de compiler die gebruikt werd om een programma te ontwikkelen. Dit in tegenstelling tot ongeformatteerde bestanden waarvan de (binaire) structuur van de file van de (FORTRAN) compiler afhangt die gebruikt werd om het desbetreffende output-genererende programma te onwikkelen.

Tenslotte, het voordeel van geformatteerde bestanden is dat de gebruiker daarin eigen commentaartekst kan opnemen. Dit ter verhoging van de oriëntatie in het bestand.

Ongeformatteerde bestanden:
Ongeformatteerde bestanden zijn optisch niet leesbaar. Deze bestanden bestaan niet uit fysieke records (regels) maar uit records die een reeks waarden in binaire code bevatten. Evenals bij geformatteerde bestanden staan de gegevens in een bepaalde volgorde. Vanzelfsprekend is er een afstemming nodig tussen het programma dat deze bestanden aanmaakt en het programma dat de gegevens leest. Om ongeformatteerde bestanden aan te maken of te lezen is speciale programmatuur nodig (deze moet door de gebruiker zelf geschreven worden).

Voor- en nadelen:
In de navolgende tabel worden de belangrijkste voor- en nadelen van geformatteerde en ongeformatteerde bestanden weergegeven.

Tabel 6.1 Voor- en nadelen van geformatteerde en ongeformatteerde databestanden

| | geformatteerd | ongeformatteerd |
|---|---|---|
| aanmaken en/of aanpassen van file, visuele inspectie van inhoud | m.b.v. willekeurige editor of tekstverwerker | noodzakelijkerwijs m.b.v. speciale programmatuur |
| fouten | typefouten zijn snel gemaakt, verkeerde volgorde in gegevens of verkeerde positie leidt tot fouten | weinig kans op fouten omdat bestand alleen te maken/aanpakken door speciale programmatuur |
| grootte van datafiles lees- en schrijfsnelheid | relatief groot, relatief lange lees- en schrijftijd | relatief klein, relatief korte lees- en schrijftijd |
| rechtstreekse visuele leesbaarheid | ja | neen |
| leebaarheid door programma's ontwikkeld m.b.v. andere compiler | ja (als regel) | neen (als regel) |
| mogelijkheid opnemen van commentaartekst door gebruiker | ja | neen |

## 6.4 Gegevensbehandeling in database management systeem

Databestanden kunnen uitstekend worden opgebouwd binnen een database management systeem (DBMS). Zo'n systeem biedt standaard vele mogelijkheden om gegevens te manipuleren en te ordenen. Zo kunnen rekenkundige bewerkingen op de gegevens worden uitgevoerd en kunnen gegevens eenvoudig worden geselecteerd en gesorteerd.

De inzet van een database management systeem bij het beheer van gegevens levert een groot aantal voordelen op:

- dezelfde gegevens kunnen door verschillende gebruikers, eventueel tegelijkertijd gebruikt worden;
- data inconsistentie wordt vermeden;
- controle op data redundantie;
- data onafhankelijkheid kan eenvoudiger worden bereikt;
- de hoeveelheid te produceren code wordt gereduceerd omdat veel functies (functionaliteit) reeds beschikbaar zijn;
- onderhoud van applicatie programmatuur wordt vereenvoudigd;
- de integriteit van data kan gemakkelijker worden gewaarborgd;
- de veiligheid van data kan eenvoudiger worden gewaarborgd;
- er kan sneller en efficiënter op veranderende eisen worden gereageerd;
- het gebruik van gegevens kan worden gevolgd;
- overdraagbaarheid wordt door de DBMS leverancier gegarandeerd.

De structuur van een database wordt formeel gedefinieerd met behulp van een data definition language (DDL). Tegenwoordig is het mogelijk om naast de entiteiten ook de relaties tussen entiteiten en de hiervoor geldende beperkingsregels via een DDL te definieren. Hiermee neemt de hoeveelheid procedurele code die geproduceerd moet worden af en zullen een aantal datacontrole aspecten uit applicatie software verdwijnen.

Omdat een DBMS een aantal lagen definieert tussen de fysieke opslag van data en de applicatie, zal bij het gebruik van een DBMS doorgaans niet de performance worden gehaald die met opslag zonder tussenkomst van een DBMS mogelijk is. De voordelen die het gebruik van een DBMS biedt, wegen echter ruimschoots op tegen dit nadeel.

Ook bij het gebruik van een database management systeem zal een keuze gemaakt moeten worden ten aanzien van de groepering van de gegevens, bijvoorbeeld per laag, soort parameter of element. Bij een element moet in dit verband bijvoorbeeld worden gedacht aan een knooppunt in het eindige elementen netwerk of een tak in een waterlopenstelsel. Bij een parameter moet bijvoorbeeld worden gedacht aan de ruwheid van de wand van die tak, of het doorlaatvermogen van een watervoerend pakket. Indien de groepering per element de voorkeur verdient zal voor elk element een record worden aangemaakt. De parameters zullen in de velden van de records worden opgeslagen. Indien gekozen wordt voor een groepering per parameter dan zal voor elke parameter een record worden aangemaakt en zal voor elk element een veld worden gereserveerd.

De structuur van een record kan binnen een database management systeem eenvoudig worden gewijzigd. Velden kunnen worden toegevoegd of verwijderd, desgewenst kan de volgorde van de velden worden gewijzigd. Er kan eenvoudig een selectie van gegevens worden gemaakt die geconverteerd wordt naar een invoerfile van een grafisch programma, zodat een grafische controle snel te verwezenlijken is.

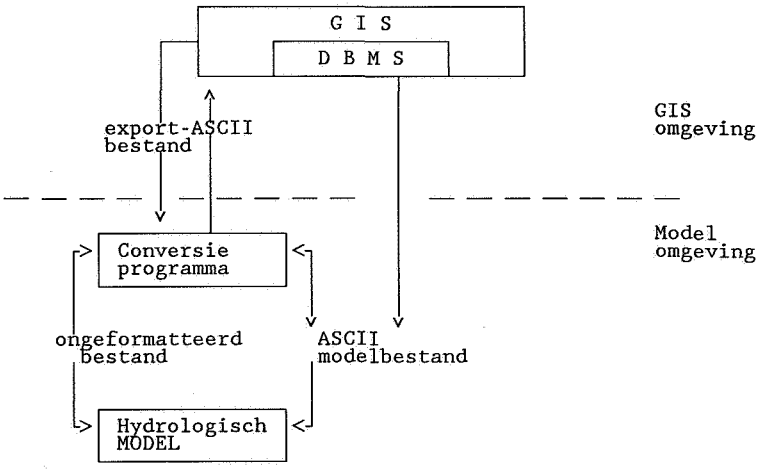
De file in een database management systeem is zo overzichtelijk als de gebruiker dat zelf wenst. Diverse overzichten kunnen op eenvoudige wijze geproduceerd worden. Files die door een database management systeem worden aangemaakt zijn over het algemeen niet van het ASCII type, maar elk database management systeem heeft een conversiemogelijkheid naar ASCII. Dit ASCII-bestand kan dan rechtstreeks door een hydrologisch programma worden ingelezen (c.q. aangemaakt), of pas nadat het door een speciaal conversie programma in een benodigd modelformaat is omgezet (hetzij wederom ASCII type, hetzij ongeformatteerd).

## 6.5 Gegevensbehandeling in GIS

Binnen een Geografisch Informatie Systeem (GIS) worden bestanden van een speciaal type gehanteerd. Deze bestanden bevatten de informatie over de verdeling (variatie) van een parameter in een twee-dimensionale ruimte, zoals de maaiveldsligging, bodemkaart of de in punten gemeten grondwaterstanden.

De interne GIS-bestanden kunnen door een hydrologisch computerprogramma niet rechtstreeks worden gelezen, noch gemakkelijk worden aangemaakt. Wel is er binnen een GIS meestal een optie aanwezig om een intern GIS bestand naar een ASCII-formaat bestand om te zetten. Deze zogenaamde "export-ASCII" bestanden hebben een vastomschreven formaat voor wat betreft de inhoud.

In de situaties waar een hydrologisch programma buiten de GIS-omgeving wordt geïmplementeerd, zal een programma-ontwikkelaar moeten beslissen of de data-communicatie (invoer/uitvoer) tussen een programma en GIS zal geschieden op basis van deze "export-ASCII" bestanden óf dat er een speciaal hulpprogramma wordt ontwikkeld ten behoeve van de conversie van een "export-ASCII" bestand naar een bestand met een benodigd modelformaat (hetzij wederom van het ASCII-type, hetzij ongeformatteerd). Indien toch een conversieslag nodig is, kan ervoor worden gekozen om de ASCII-bestanden van een benodigd modelformaat rechtstreeks binnen GIS te laten aanmaken, en wel zonder de tussenkomst van een "export-ASCII" bestand. De figuur 6.1 geeft schematisch de mogelijke soorten van gegevensstromen weer. In een concrete situatie zou het datatransport bijvoorbeeld alleen door middel van ASCII bestanden plaatsvinden en niets zoals in de figuur is gegeven, door zowel ASCII als "export-ASCII" en ongeformatteerde bestanden. Het begrip "bestand" betreft zowel modelinput als modeloutput.




Figuur 6.1 Soorten gegevensstroom tussen GIS en hydrologisch model, bij gescheiden werkomgevingen. Neerwaarts: invoergegevens uit GIS voor model. Opwaarts: modelresultaten naar GIS, bijvoorbeeld voor grafische presentatie.

## 6.6 Voorbeelden geformatteerde (ASCII) bestanden

Het grote voordeel van geformatteerde bestanden is dat deze optisch leesbaar zijn, bijvoorbeeld op het scherm en als printuitvoer.

Bij geformatteerde bestanden is de positie van in te lezen variabelen per regel (record) en de plaats op een regel van te voren door de programma-ontwikkelaar vastgelegd. De plaats ter rechterzijde van de op een regel voorkomende variabelen kan door de programmagebruiker worden benut om daar een eigen toelichtende tekst te plaatsen (toe te voegen). Een ASCII-bestand kan met een willekeurige tekstverwerker of met een speciaal hulpprogramma (pre/postprocessor) worden aangemaakt. In de praktijk zal een ASCII-bestand met behulp van een tekstverwerker worden aangemaakt en aangepast.

Naast 'fixed format' invoer is het mogelijk om een geformatteerd bestand ook door middel van 'free format' in te lezen. Bedacht dient te worden, dat altijd het benodigde aantal variabelen wordt ingelezen bij de 'FREE FORMAT' die in FORTRAN 77 is ingebouwd. Dit is ongeacht hun positie binnen een bestandsrecord of zelfs de verschillende records. Indien bijvoorbeeld 3 variabelen worden ingelezen, kunnen deze op één record aanwezig zijn of verspreid over 2 of 3 records.

Voorbeeld 1 Topology of a sewage system

In dit voorbeeld wordt een ASCII-file geïllustreerd, die door middel van een 'DO ... CONTINUE' programmastructuur kan worden ingelezen. Een file bestaat hierbij veelal uit samengestelde blokken met elk een eigen blok-header en record structuur. Een voorbeeld hiervan vormt het gebruik van een blok-header waarin titel, het aantal records en de variabelen in een record beschreven worden. De tekst van de header wordt door de programmeur opgegeven, zoals onderstaand voorbeeld illustreert.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | titel.... | : | AANTAL STROOMGEBIEDEN | | | | |
| - | nummer... | : | 13 | | | | |
| - | tekstregel | : | KNOOP | OPP. | ‰ | L | A | etc. |
| - | records.. | : | 81 | 0.070 | 20 | 30 | 2 | ... |
| | | | 82 | 4.390 | 15 | 200 | 2 | ... |

Enkele opmerkingen met betrekking tot de structuur van een record zijn:

- records bestaan uit velden waaraan waarden worden/zijn toegekend;
- de velden welke refereren aan andere records (pointers) worden als eerste genoemd;
- in het geval aan velden geen waarden worden/zijn toegekend blijft het veld leeg, de hoofdstructuur van het record blijft intact;
- gebruik altijd de decimale punt (bijvoorbeeld 1.0 of 100.0) voor het aangeven van de 'real' variabele.

AANTAL STROOMGEBIEDEN — FORMULIER D

13

| KNOOP | OPP. | % | L | A | PE | Q | H | PCT | B | VERDELINGOPPERVLAKKEN |
|---|---|---|---|---|---|---|---|---|---|---|
| 81 | 0.070 | 20 | 30 | 2 | 0 | 0.000 | 1 | 35 | | |
| 82 | 4.390 | 15 | 200 | 2 | 0 | 0.000 | 1 | 33 | | |
| 83 | 3.240 | 15 | 160 | 2 | 0 | 0.000 | 1 | 26 | | |
| 100 | 2.650 | 15 | 150 | 2 | 0 | 0.000 | 1 | 35 | | |
| 110 | 0.270 | 15 | 50 | 2 | 0 | 0.000 | 1 | 35 | | |
| 120 | 0.250 | 10 | 50 | 2 | 0 | 0.000 | 1 | 28 | | |
| 130 | 0.180 | 10 | 40 | 2 | 0 | 0.000 | 1 | 30 | | |
| 140 | 0.320 | 10 | 60 | 2 | 0 | 0.000 | 1 | 25 | | |
| 150 | 0.800 | 10 | 70 | 2 | 0 | 0.000 | 1 | 65 | | |
| 170 | 0.940 | 10 | 80 | 2 | 0 | 0.000 | 1 | 30 | | |
| 171 | 0.860 | 10 | 60 | 2 | 0 | 0.000 | 1 | 25 | | |
| 172 | 0.840 | 10 | 90 | 2 | 0 | 0.000 | 1 | 25 | | |
| 180 | 0.650 | 15 | 70 | 2 | 0 | 0.000 | 1 | 35 | | |

AANTAL RONDE PUTTEN — FORMULIER KG1

14

| KNOOP | X | Y | B.O.B. | MAAIV | T | DIAM |
|---|---|---|---|---|---|---|
| 60 | 342.0 | 1007.0 | 29.72 | 33.90 | 2 | 1.00 |
| 70 | 309.0 | 943.0 | 30.10 | 34.15 | 2 | 1.00 |
| 80 | 275.0 | 877.0 | 30.30 | 35.00 | 2 | 1.00 |
| 81 | 240.0 | 813.0 | 30.60 | 34.50 | 2 | 1.00 |
| 82 | 241.0 | 775.0 | 30.95 | 34.95 | 2 | 1.00 |
| 83 | 247.0 | 722.0 | 31.11 | 35.10 | 2 | 1.00 |
| 90 | 314.0 | 860.0 | 30.65 | 34.90 | 2 | 1.00 |
| 110 | 319.0 | 768.0 | 32.02 | 36.10 | 2 | 1.25 |
| 120 | 327.0 | 730.0 | 32.13 | 36.10 | 2 | 1.25 |
| 130 | 335.0 | 692.0 | 32.24 | 35.40 | 2 | 1.25 |
| 140 | 343.0 | 655.0 | 32.34 | 35.50 | 2 | 1.25 |
| 171 | 320.0 | 551.0 | 33.48 | 35.57 | 2 | 1.00 |
| 172 | 399.0 | 572.0 | 34.89 | 36.93 | 2 | 1.00 |
| 181 | 379.0 | 472.0 | 34.49 | 36.54 | 2 | 1.00 |

AANTAL KUNSTWERKEN — FORMULIER KG2

5

| KNOOP | X | Y | B.O.B. | MAAIV | T |
|---|---|---|---|---|---|
| 100 | 316.0 | 834.0 | 31.83 | 35.75 | 2 |
| 150 | 354.0 | 602.0 | 32.53 | 35.90 | 2 |
| 160 | 270.0 | 573.0 | 32.75 | 35.80 | 2 |
| 170 | 278.0 | 539.0 | 32.88 | 35.35 | 2 |
| 180 | 296.0 | 470.0 | 33.10 | 36.75 | 2 |

Voorbeeld 2 Voorbeeld geformatteerd bestand, invoer voor programma SIMGRO. (Staring Centrum, Wageningen)

Elke record ("regel") heeft een naam. Het programma controleert of het de juiste record inleest, aangeduid bijvoorbeeld met tekst "ROOT" (dikte van wortelzone). Foutmeldingen verwijzen naar deze naam. Per recordnaam wordt in de handleiding de invoer beschreven (verwijzing mogelijk naar andere records). Recordnaam is hulpmiddel bij het aanmaken van het bestand, of controleren van de gegevens.

```
*HEAD      Hupselse Beek catchment area     - updated : 30-11-90
*HEAD      MODEL MOGROW, module SIMGRO: Standard data deck - 1981-1985
*GNRL       283  502    1   62   10   40    6    5    1    4    0   5    0
*GNRL        1 1981   360 1985   1.        91.       .001  .55  1.3  245 0.05
*GNRL          1    1    1  0   2    5    3    0    0    0
*GNRL          0
*PLOT         10         156  52  130  146  264  177   53   62   36  166
*PLOT                     1 1    1    1    1 1    1    1    1    1
*TECH          1 GRASLAND            2 MAIS              3 GRANEN
*TECH          4 BIETEN              5 BOSSEN
*INFL         100.0    100.0     100.0   100.0      100.0  100.0
*ROOT    1      0.30 0.30 0.25 0.25 0.25 0.25
*ROOT    2      0.40 0.40 0.25 0.25 0.25 0.25
*ROOT    3      0.40 0.40 0.25 0.25 0.25 0.25
*ROOT    4      0.35 0.35 0.25 0.25 0.25 0.25
*ROOT    5      1.00 1.00 1.00 0.80 0.65 0.50
*SFAC      0.50 0.60 0.65 0.68  1.0        0.0 0.20   5.   20.
*INDX         -1    2    3    4    0
*INTC         .0   8.   5.   0.   .0
*WLOG            0    0    0    1    0
*SPRD        7.0  0.0 14.0  7.0  0.0
*BEG         120  180  150  150
*END         255  245  245  245
*PROD        3.0  0.0  1.0  2.0  0.0
*WLEV  1          0.80       0.90       1.00       1.10       1.20
*WLEV             0.0        0.0        0.0        0.0        0.0
*WLEV  2          0.80       0.90       1.00       1.10       1.20
*WLEV             0.20       0.15       0.10       0.05       0.00
*WLEV  3          0.80       0.90       1.00       1.10       1.20
*WLEV             0.30       0.25       0.20       0.15       0.10
*V25   0.0      129.7      129.7      129.7      129.7      129.7      121.7
*V25   0.1      127.2      127.2      127.2      127.2      127.2      119.2
*V25   0.2      121.5      121.5      121.5      121.5      121.5      113.5
*V25   0.3      114.5      114.5      114.5      114.5      114.5      106.3
*V25   0.4      110.0      110.0      110.0      110.0      110.0      102.0
*V25   0.5      107.0      107.0      107.0      107.0      107.0       99.0
```

Voorbeeld 3 (RIVM, K. Kovar, Bilthoven)

Voorbeeld van geformatteerd bestand, ten behoeve van vastlegging van data voor grondwaterontrekkingen in AQ-programmatuur (computer programma AQ-APIN). Dit bestand kan door middel van een teksteditor door de gebruiker worden aangemaakt. Het wordt echter ook door AQ-APIN exact in deze vorm aangemaakt. Indien de data voor het eerst door de gebruiker aan AQ-APIN worden aangeboden, dan behoeft de commentaartekst ter rechterzijde van de getallen (inclusief logische waarden en teksten) niet te worden opgegeven. AQ-APIN genereert de tekst zelf ter verhoging van de optische leesbaarheid van het bestand. Recordnummers worden door AQ-APIN automatisch toegevoegd ter oriëntatie in de bijgevoegde gebruikershandleiding van de inhoud van het bestand, per individueel record.

```
ASCII INPUT WELL                                                Record 01
Example 1 of ASCII File for Wells       <...TITXT               Record 02
This file is generated by program AQ-APIN, version 4.0, Rec.03, subr.00001
Date= 08-04-1991, Time= 14:05:46                                Rec.03, subr.00002
    3     Number of Wells NWL                                   Record 04
  .f.     Flag FLTRAN of whether Transient Case                 Record 05
  .f.     Flag FLSESS for Sequence of Steady States             Record 06
  .f.     Flag FLZL of whether Well Screen Depth                Record 08
    1 === Sequence Number of Well IWL       ==========        Record 09
Well 1, South       <............TXTWL                         Record 10
        1000.0          1000.0   XWL, YWL                     Record 11
    2     Number of Aquifer IAQF where Well located             Record 12
         -340.0     QW                                         Record 14
    2 === Sequence Number of Well IWL       ==========        Record 09
Well 2, North       <............TXTWL                         Record 10
        2000.0        -12000.0   XWL, YWL                     Record 11
    4     Number of Aquifer IAQF where Well located             Record 12
      150000.0     QW                                         Record 14
    3 === Sequence Number of Well IWL       ==========        Record 09
Well 3, South-North <............TXTWL                         Record 10
       -9999.0             0.0   XWL, YWL                     Record 11
    1     Number of Aquifer IAQF where Well located             Record 12
          35.0     QW                                         Record 14
```

## 6.7 Voorbeeld bestandsstructuur DBMS

De hierna getoonde tabellen vormen een onderdeel van het bij IGG-TNO ontwikkelde Online Grondwater Archief (OLGA). De Figuur 6.2 toont hoe de velden in de afzonderlijke tabellen zijn gedefinieerd. Tevens worden in de figuur (een deel van) de relaties die tussen

de tabellen bestaan getoond. Hierbij geld:

kaartblad = key veld

= verplicht veld

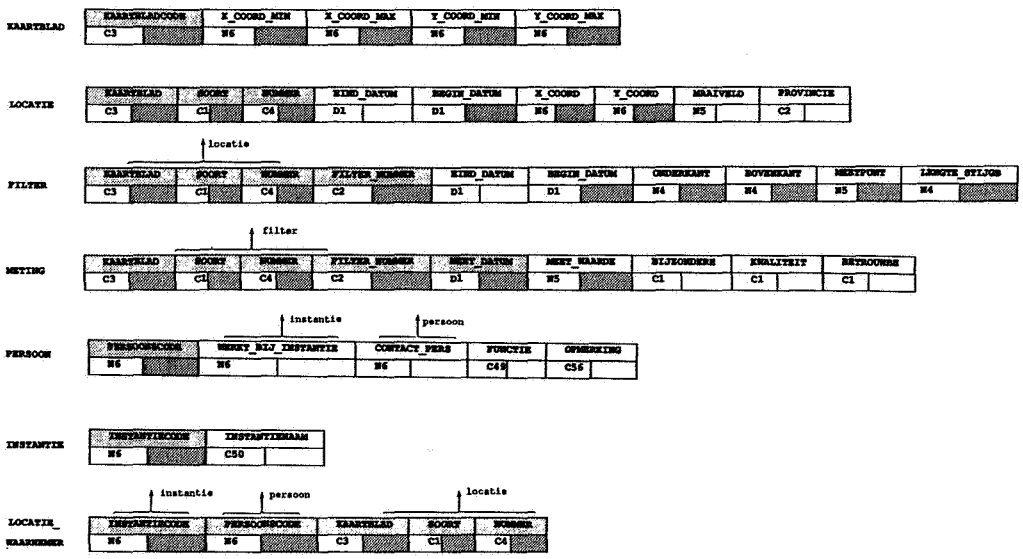
= niet verplicht veld

Cn = character veld van maximum n posities

Nn = numeriek veld van maximaal n posities

D = datum veld

Figuur 6.2 kan, evenals de definitie van de tabellen in het DBMS, automatisch worden gegenereerd door het programma waarmee de objecten en de relaties tussen objecten worden gedefinieerd.




Figuur 6.2 Voorbeeld bestandsstructuur

## 6.8 Koppeling van programma's

Een mogelijkheid om verschillende soorten programmatuur te koppelen is via het uitwisselen van databestanden; uitvoer van het ene programma kan dienen als invoer voor het andere. Hierbij is het van belang dat de programmatuur de mogelijkheid biedt om deze bestanden indien gewenst te leveren. Ook moeten de bestanden min of meer gestandaardiseerd worden, zodat ze eenvoudig in te lezen zijn in andere programma's. In de meeste gevallen zal dit concreet neerkomen op kolomgeöriënteerde bestanden van de meest relevante rekenresultaten, gegeven in standaard SI-eenheden, Min of meer esthetische toevoegingen zoals tabelkopjes, bladzijde-indelingen etc. zijn niet nodig en zelfs niet wenselijk.

## 6.9 Resumé

- Stel de volgende eisen aan databestanden:
  * overzichtelijkheid;
  * eenvoudig te herstructureren;
  * waar relevant moet een bestand aangeboden kunnen worden aan een grafisch programma, ten behoeve van grafische presentatie en visuele controle;
  * de gegevens uit een bestand moeten in numerieke vorm leesbaar zijn (ASCII-bestand of speciaal programma).
- Besteed aandacht aan de keuze tussen geformatteerde en ongeformatteerde files.
- Overweeg het gebruik van een database management systeem voor systematische opbouw en beheer van bestanden.

# 7 LAY-OUT FORTRAN 77-BRONCODE

## 7.1 Inleiding

De lay-out van de broncode van FORTRAN, maar ook van andere programmeertalen, is van groot belang voor de bevordering van de kwaliteit van software. Door een juiste lay-out wordt niet alleen in eerste instantie de leesbaarheid van de broncode zelf verhoogd, maar in het verlengde daarvan ook de onderhoudbaarheid van programmatuur als geheel. Bij een (beter) leesbare code kan men zich immers beter en sneller oriënteren, waardoor de kans op fouten en misverstanden afneemt en het onderhoudsproces efficiënter verloopt. Het kan verder worden verwacht dat de programmatuur die door de ontwikkelaar (of de organisatie van oorsprong) gemakkelijk kan worden onderhouden ook zonder bijzonder veel inspanning bij derden kan worden geïmplementeerd. Met andere woorden, de lay-out werkt ook bevorderend op de overdraagbaarheid.

Achtereenvolgens komen aan de orde:
- lettertype en positionering van de code;
- assignment statement;
- do-loop statement;
- argument van subroutines en functies;
- if-then statement;
- read en write statement.

## 7.2 Lettertype en positionering van code

* Gebruik voor de tekst van de code systematisch kleine letters, deze schijnen een beter leesbare code op te leveren. Alleen hoofdletters voor de code is af te raden.
* De tekst van de code begint in principe in kolom 7. Indien de code verder begint, gebeurt dit in verband met het inspringen (zoals bij do-loop en if-then).
* De tekst van de code loopt in principe tot en met kolom 71, de kolom 72 wordt vrijgehouden als scheiding van de mogelijke commentaartekst in het veld 73-80.
* 1 Lege (blanco) positie aanhouden tussen de variabelen, zoals in het argument van subroutines, declaraties van variabelen en datablocks.
* Gebruik voor de continuation van lijnen in het hele programma eenzelfde teken,

bijvoorbeeld het teken '&'. Gebruik bij voorkeur geen cijfers, want het gebruik van cijfers (1, 2, enzovoorts) zou verwarrend kunnen zijn indien in de buurt andere cijfers zouden liggen, bijvoorbeeld de labels van do-loops.

Aanbevolen:

```
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      call mmdiff (ip, xc, yc, diff)

      subroutine mmdiff (ip, xc, yc, diff)

      call vrdiff (iin1, iin2, iunplt, iunprt, xc(k), yc(k), zc(k),
     &             par1, par2, par3, iunxc, iunyc, iunzc, icod1, icod2,
     &             flag, flerr)

      subroutine vrdiff (iin1, iin2, iunplt, iunprt, xc, yc, zc, par1,
     &                   par2, par3, iunxc, iunyc, iunzc, icod1, icod2,
     &                   flag, flerr)
```

Afgeraden:

```
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      subroutine MMdiff (ip, Xc, Yc, diff)

      SUBROUTINE MMDIFF (ip, xc, yc, diff)
```

## 7.3 Assignment statement

* Aan linkerzijde van het '=' teken minimaal 1 blanco positie opnemen. Dit verhoogt de overzichtelijkheid.
* Aan rechterzijde van het '=' teken altijd 1 blanco positie aanhouden. Dit verhoogt de overzichtelijkheid.
* Plaats het '=' teken van achtereenvolgende assignment statements zoveel mogelijk onder elkaar. Dit verhoogt de overzichtelijkheid. Indien de positie van het '=' teken in kolom 15 wordt gekozen zullen in het veld links van het '=' teken variabelen passen met een maximale lengte van 7 posities. Als het aantal posities 7 niet toereikend is (bijvoorbeeld bij array-variabelen) verschuif dan het '=' teken naar rechts zover als nodig is.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      ar(i)   = br(i)+cr(i)
      pk(i,k) = dh(i)+1.
      imtot   = j+1
      rvhp(is,m) = ag(is)*rr(j)
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      ar(i) = br(i)+cr(i)
      pk(i,k)= dh(i) + 1.
      imtot= j+1
      rvhp(is,m)=ag(is) * rr(j)
```

* Operators (rechterzijde statement) voor verschillende NIVEAUS van bewerkingen scheiden met 1 blanco positie. Dit verhoogt de overzichtelijkheid.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      ar(i)   = br(i)+cr(i)
      pk(i,k) = (dh(i)-one) * (dh(i)-one) / fact
      imtot   = j+1-jmin
      rvhp(is,m) = ag(is) * rr(j)**3
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      ar(i)   = br(i) + cr(i)
      pk(i,k) = (dh(i)-one)*(dh(i)-one) / fact
      imtot   = j+1 - jmin
      rvhp(is,m) = ag(is) * rr(j) **3
```

## 7.4 Do-loop statement

* Labels bij 'continue' met 10 laten oplopen, beginnend bij 10.
* Labels bij 'continue' rechts aansluiten bij kolom 5.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
  10 continue
  20    continue      'continue' verschoven vanwege do-loop of if-then
  30 continue
  .. ........
 100 continue
 110 continue
 120       continue   'continue' verschoven vanwege do-loop of if-then
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
  16 continue
 249 continue
   3 continue
```

* De statements behorend bij een IF-THEN-ELSE of DO-blok 3 posities naar rechts laten inspringen ten opzichte van de IF-THEN-ELSE en DO-opdrachten.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      if (k.gt.1) then
         do 90 i = 3,11,2
            ........
            ........
   90    continue
      else
         do 110 i = 3,11,2
            ........
            ........
            if (i.eq.5) then
               do 100 j = 1,jtot
                  ........
                  ........
  100          continue
            endif
  110    continue
      endif
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      if (k.gt.1) then
       do 90 i = 3,11,2
          ........
          ........
   90  continue
       else
          do 110 i = 3,11,2
             ........
             ........
          if (i.eq.5) then
             do 100 j = 1,jtot
             ........
             ........
  100        continue
          endif
  110     continue
       endif
```

* Do-loop altijd met een 'continue' eindigen
* Elke do-loop heeft een eigen 'continue'.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      do 80 k = 1,ktot
         do 70 i = 1,nn
            asr(i,k) = ah(i,k)
            bsr(i,k) = bh(i,k)
   70    continue
   80 continue
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      do 70 k = 1,ktot
         do 70 i = 1,nn
            asr(i,k) = ah(i,k)
            bsr(i,k) = bh(i,k)
   70 continue
```

## 7.5 Argument van subroutines en functies

* 1 Lege (blanco) positie laten tussen de variabelen. Dit verhoogt de overzichtelijkheid.
* De beginpositie van de eerste argument variabele op een continuation lijn gelijk stellen aan die op de voorgaande lijn.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      call vrdiff (iin1, iin2, iunplt, iunprt, xc(k), yc(k), zc(k),
     &             par1, par2, par3, iunxc, iunyc, iunzc, icod1, icod2,
     &             flag, flerr)
```

```
      subroutine vrdiff (iin1, iin2, iunplt, iunprt, xc, yc, zc, par1,
     &                   par2, par3, iunxc, iunyc, iunzc, icod1, icod2,
     &                   flag, flerr)
```

```
      if (mdp.lt.0) then
         flag    = .true.
         do 40 k = 1,kt
            call vrdiff (iin1, iin2, iunplt, iunprt, xc(k), yc(k), zc(k),
     &                   par1, par2, par3, iunxc, iunyc, iunzc, icod1,
     &                   icod2, flag, flerr)
   40    continue
      endif
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      call vrdiff (iin1, iin2, iunplt, iunprt, xc(k), yc(k), zc(k),
     1             par1, par2, par3, iunxc, iunyc, iunzc, icod1, icod2,
     2             flag, flerr)
```

```
      subroutine vrdiff (iin1,iin2,iunplt,iunprt,xc,yc,zc,par1,
     &    par2,par3,iunxc,iunyc,iunzc,icod1,icod2,flag,flerr)
```

```
      if (mdp.lt.0) then
         flag    = .true.
         do 40 k = 1,kt
            call vrdiff (iin1, iin2, iunplt, iunprt, xc(k), yc(k), zc(k),
     $         par1, par2, par3, iunxc, iunyc, iunzc, icod1, icod2, flag,
     #         flerr)
   40    continue
      endif
```

## 7.6 If-then statement

* Gebruik in het argument geen blanco posities behalve vóór en na '.or.' en '.and.', of haakjes gebruiken. Deze blanco posities of haakjes, geven de visuele scheiding aan tussen de diverse niveaus van logische bewerkingen, zulks ter verhoging van de leesbaarheid.
* Bij een grote afstand tussen 'if' en 'endif' statements (bij belangrijke if-then statements) vlak boven de desbetreffende 'endif' ook commentaar invoegen om eraan te herinneren welke if-then statement er eindigt.

Aanbevolen:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      if (ak.lt.bk) then
      if (ak.lt.bk .or. ck.lt.dk) then
      if ((xr.lt.xmin).and.(yr.lt.ymin).and.(zr.lt.zmin)) then
      if ((at.lt.bt.or.ct.lt.dt).and.ip.eq.17) then
```

Afgeraden:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      if (ak .lt. bk) then
      if (ak.lt.bk.or.ck.lt.dk) then
      if (xr .lt. xmin .and. yr .lt. ymin .and. zr .lt. zmin) then
      if ((at.lt.bt.or.ct.lt.dt) .and. ip.eq.17) then
```

## 7.7 Read en write statement

* Zowel afzonderlijke format statements als format in read/write zijn toegestaan. Gebruik ter beoordeling aan programmeur.
* Indien format statements worden gebruikt, de labels voor read en write afzonderlijk nummeren, bijvoorbeeld respectievelijk bij 1000 en 2000 laten beginnen.
  Evenals bij do-loops, de labels van format statements met 10 laten oplopen. Indien de formats op diverse plaatsen worden gebruikt, de format statements niet onmiddellijk achter de read en write statements plaatsen maar aan het einde van subroutines, tussen de (laatste) return- en end-statement. Bij eenmalig gebruik is de plaatsing direct bij schrijfopdracht handiger, vooral omdat het format vaak een aanduiding bevat van het soort bericht dat wordt weggeschreven.
* Tussen de statements 'read', 'write' of 'format' en '(' altijd een blanco positie. Tussen de laatste ')' van read en write en lijst van te lezen/schrijven variabelen (voor zover aanwezig) ook een blanco positie. De te lezen/schrijven variabelen scheiden door een spatie.

Aanbevolen:

```
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      parameter (iin=5, iout=6)
      .......................
      read (iin,'(i5)', err=10, end=20) mdir
      write (iout,'(''MDIR = '',i5)') mdir
      .......................
      read (iin,1000, err=10, end=20) ic
      read (iin,1010, err=10, end=20) r1, r2
      .......................
      .......................
      goto 30
   10 afhandeling foutmelding
   20 afhandeling foutmelding
      .......................
   30 write (iout,2000) ic
      write (iout,2010) r1, r2
      .......................
      return
*-----------------------------------------------------------------------------
* Format statements
*
 1000 format (i5)
 1010 format (2e12.6)
 2000 format ('IC = ',i5)
 2010 format ('R1 = ',e12.6, 5x, 'R2 = ',e12.6)
      end
```

Afgeraden:

```
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
      data iin/5/, iout/6/
      .......................
      read(3,'(i5)') mdir
      write(4,'(''MDIR = '',i5)') mdir
      .......................
      read(7,723) ic
  723 format(i5)
      read(8,49) r1,r2
   49 format(2e12.6)
      .......................
      .......................
      write(6,221) ic
  221 format('IC = ',i5)
      write(6,2016) r1,r2
 2016 format('R1 = ',e12.6, 5x, 'R2 = ',e12.6)
      .......................
      .......................
      return
      end
```

## 7.8 Resumé

- Ontwerp een consistent systeem voor lay-out van broncode.
- Beschouw dit hoofdstuk als een aanbevelenswaardig voorbeeld van een dergelijk systeem.
- Wees consequent bij de implementatie van het gekozen lay-out systeem in de programmeerpraktijk.

# 8 FORTRAN 77-PROGRAMMEERASPECTEN

## 8.1 Inleiding

In dit hoofdstuk worden een aantal op zichzelf staande programma-technische keuzes behandeld die in ieder programma aan de orde zijn, maar niet onder één van de overige hoofdstukken vallen en die specifiek zijn voor het gebruik van de in de hydrologie veel gehanteerde programmeertaal FORTRAN-77. Het betreft veelal zaken waarover niet altijd eenduidig vast kan worden gelegd welke lijn gevolgd moet worden. De keuze die gemaakt wordt hangt vaak af van het specifieke probleem dat aan de orde is, de programmeerstijl etc. In bijna alle gevallen geldt echter dat consequent gebruik en overzichtelijkheid doorslaggevend zijn.

Terwille van de verhoging van de leesbaarheid en overdraagbaarheid wordt sterk aanbevolen om de compiler specifieke programmeerfaciliteiten te vermijden. Het is raadzaam om de geldende normen te volgen, bijvoorbeeld ANSI FORTRAN 77 (ANSI, 1978).

Achtereenvolgens komen aan de orde:
- declaraties;
- initialisering;
- lokale variabelen;
- argumentenoverdracht;
- gebruik van intrinsieke functies;
- lengte subroutines;
- read en write statements;
- groepering van compiler-afhankelijke functies.

## 8.2 Declaraties

Binnen FORTRAN 77 kunnen variabelen op twee manieren gedeclareerd worden:
impliciet of expliciet. Bij impliciete declaraties is de beginletter van de variabele bepalend voor het data-type als de variabele niet expliciet gedeclareerd is. Standaard is I-N voor integers, en de rest van de letters voor reals en overige data-typen. De beginletters kunnen

vastgelegd worden door het IMPLICIT-statement, bijvoorbeeld:

```
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
```

Bij volledig expliciete declaratie genereert de compiler een foutmelding bij iedere variabele die niet gedeclareerd is. Bij sommige compilers kan hiervoor het statement gegeven worden:

```
      IMPLICIT NONE (geen standaard F77)
```

of moet met behulp van een switch bij de compilatie de optie gespecificeerd worden. Zijn beide opties niet mogelijk, dan kan in de programmatestfase in vrijwel alle gevallen hetzelfde resultaat geboekt worden door het specificeren van:

```
      IMPLICIT LOGICAL (A-Z)
of    IMPLICIT CHARACTER * 1 (A-Z)
of    IMPLICIT COMPLEX (A-Z)
```

Hierbij wordt bijna altijd een compiler-foutmelding gegenereerd indien een niet gedeclareerde variabele gebruikt wordt. Het voordeel van impliciete declaraties is dat er minder programmacode geproduceerd moet worden en er dus ook navenant minder typefouten optreden tijdens de compilatie-fase. Daarnaast hebben impliciete declaraties het voordeel dat in een programma direct opvalt welk type de verschillende variabelen hebben, zonder steeds een declaratielijst te hoeven raadplegen. Het grootste nadeel van impliciete declaraties is echter dat gemaakte typefouten tijdens de runtime-fase moeilijk te detecteren fouten kunnen opleveren, aangezien een fout ingetypte variabele opgemerkt wordt als een nieuwe variabele met een type dat afhankelijk is van de beginletter. Zo zal bij impliciete declaraties het verwarren van de variabele HOLD door H0LD, tot gevolg hebben dat met een nieuwe variabele H0LD gerekend wordt met initiële waarde, in plaats van met de bestaande variabele HOLD met de bestaande waarde. Bij expliciete declaraties zou een dergelijke fout reeds tijdens de compilatiefase opgemerkt zijn.

In het algemeen lijkt het raadzaam om expliciete declaraties toe te passen, waarbij wel een conventie gehanteerd wordt van consequente beginletters voor verschillende datatypes (bij voorkeur de standaard FORTRAN 77-conventie), zodat in het programma snel een overzicht is over de gebruikte data-typen. Voorwaarde is wel dat de gebruikte conventie duidelijk in de documentatie en headings vermeld wordt.

Het is de verwachting dat in een volgende versie van FORTRAN de default impliciete declaraties (IMPLICIT INTEGER (I-N), IMPLICIT REAL (A-H, O-Z) zullen verdwijnen, zodat de programmeur gedwongen zal worden een bewuste keuze te maken.

## 8.3 Initialisering

De initiële waarde van programmavariabelen is per compiler en per processor vaak verschillend. De meeste compilers kennen aan iedere variabele tijdens de compilatie een waarde 0, 'FALSE' of ' ' toe. Bij andere compilers wordt de initiële waarde van een variabele bepaald door de toevallige geheugenplaats. Dat laatste heeft als gevolg dat bijvoorbeeld bij een eerste bewerking met de variabele vaak een zeer groot of klein niet te verwerken getal ontstaat en een runtime-fout optreedt. Het is daarom noodzakelijk om alle gebruikte variabelen te initialiseren. Dit hoeft slechts één keer plaats te vinden, namelijk voor de start van het rekenproces. Het kan worden gedaan met behulp van het DATA-statement, dat ervoor zorgt dat de gebruikte variabelen (ook die binnen een procedure) reeds tijdens de compilatie geïnitialiseerd worden op de gespecificeerde waarde. Deze initialisatie vindt slechts één keer plaats, ook al wordt de procedure vaker aangeroepen. Variabelen uit een blank COMMON-block kunnen niet door middel van een DATA-statement geïnitialiseerd worden. Variabelen uit een named COMMON-block kunnen alleen door middel van een zogenoemde BLOCK DATA subprogramma geïnitialiseerd worden. De lokale variabelen worden doorgaans geïnitialiseerd binnen een gewoon DATA-statement. Variabelen die onderdeel zijn van een argumentenlijst worden door de aanroep van de procedure geïnitialiseerd, een DATA-statement is hiervoor niet mogelijk.

Bij het initialiseren van zeer grootte array-variabelen is het in verband met het ruimtebeslag van de executeerbare code raadzaam om tijdens de programmawerking een initiële waarde toe te kennen met behulp van een DO-loop aan het begin van een programma, in plaats van een DATA-statement in de broncode.

Voorbeeld

1

```
      REAL A(1000), B(100)
      LOGICAL FLAG
      DATA A/1000 * 0./, B/100 * 0./, FLAG/.FALSE./
```

2

```
      REAL A(1000), B(100)
      LOGICAL FLAG
      COMMON/TEST/ A, B, FLAG
      ......
      ......
      END
      BLOCK DATA
      DATA A/1000 * 0./, B/100 * 0./, FLAG /.FALSE./
      END
```

3

```
      DO 100 IP = 1, 1000
         A(IP) = 0.
100   CONTINUE
      DO 110  IP = 1, 100
         B(IP)  =  0.
110   CONTINUE
      FLAG = .FALSE.
```

## 8.4 Lokale variabelen

Binnen een procedure wordt de waarde van een lokale variabele op de ene computer/-compiler wel en op de andere computer niet vastgehouden. Standaard FORTRAN 77 houdt deze waarde niet vast. Het is daarom noodzakelijk te allen tijde het SAVE-statement te gebruiken voor die lokale variabelen die vastgehouden moeten worden, dan wel de variabelen in een COMMON-block te plaatsen. Het SAVE-statement heeft geen invloed op een eventueel DATA-statement. Hoewel op de meeste compilers de mogelijkheid bestaat om via een switch bij de compilatie aan te geven of de lokale variabelen bewaard moeten worden of niet, is het SAVE-statement duidelijker. Dit geldt altijd op iedere computer, bovendien is door het gebruik van het SAVE-statement meteen duidelijk welke waarden van variabelen bij een volgende aanroep van belang zijn (namelijk argumenten, variabelen van COMMON's en SAVE).

Een SAVE-statement ziet er altijd als volgt uit:

```
      SUBROUTINE TEST (B)
      REAL A
      A = A+B
      SAVE A
      ....
      ....
      RETURN
      END
```

## 8.5 Argumentenoverdracht

Voor het overdragen van argumenten op een subroutine of functie bestaan twee mogelijkheden: via een argumentenlijst en via een COMMON-block. Doorgaans is het overdragen van argumenten via een COMMON-block efficiënter (in termen van rekentijd) dan via argumentenlijsten. Het gebruik van argumentenlijsten heeft echter een aantal voordelen die het gebruik ervan in een aantal gevallen rechtvaardigt.

Als een argumentenlijst gebruikt wordt kan een procedure eenvoudig voor meerdere doeleinden gebruikt worden waarbij de actuele variabelen per geval ingevuld worden. De lengte van CHARACTER- en ARRAY-variabelen kan daarbij variabel gemaakt worden. Het is in het programma ook direct duidelijk wat het interface van de procedure is. Bij veel argumenten in de lijst gaat deze duidelijkheid echter verloren, en kan beter een COMMON-block gebruikt worden. De lengte van variabelen, bijvoorbeeld CHARACTER en ARRAY-variabelen, in de argumentlijst wordt niet bepaald in de desbetreffende routine. Bij de aanroep van de routine vanuit een andere routine wordt alleen het type van variabelen gespecificeerd, dat wil hier zeggen CHARACTER en (bijvoorbeeld real) ARRAY, en het startadres (beginpositie) van de CHARACTER en ARRAY in de totale dataruimte. Om de data-overdracht flexibel te maken, wordt aanbevolen om in een routine bijvoorbeeld de volgende declaraties op te nemen:

```
character naam1*(*)
dimension naam2(*), naam3(nrows,*)
real naam4(*), naam5(nc,*)
```

De werkbare lengte van de actuele variabele (bijvoorbeeld CHARACTER en REAL) wordt bepaald door de lengte van de actuele variabele waarmee de desbetreffende routine wordt aangeroepen.

Het nadeel van argumentenlijsten is dat er betrekkelijk snel overdrachtsfouten gemaakt worden, bijvoorbeeld het vergeten van een argument, verwisselen van argumenten, typefouten enzovoorts. In geval van COMMON-blokken kan vaak gebruik gemaakt worden van een (per compiler verschillend) INCLUDE of INSERT-statement om een slechts éénmalig aangemaakt COMMON-BLOCK te laden in die procedures waar het blok nodig is. Bij het declareren van COMMON-blokken is het verstandig om verschillende data-typen in verschillende COMMON-blokken te plaatsen. Voor CHARACTER-variabelen is dit zelfs verplicht.

Binnen een programma zal iedere keer een afweging gemaakt moeten worden tussen duidelijkheid, foutengevoeligheid en snelheid, waarbij duidelijkheid voorop staat.

## 8.6 Intrinsieke functies

Intrinsieke functies binnen FORTRAN 77, zoals COS, MIN, SQRT etc., kunnen op twee manieren aangeroepen worden: als generieke functie of als data-type afhankelijke specifieke functie. De generieke functie produceert meestal een uitkomst voor ieder data-type, zoals de functie MAX als input zowel integers als reals kan hebben en als resultaat ook een integer respectievelijk een real heeft. De specifieke functie kan slechts een specifiek data-type als input en ook één data-type als output genereren, zoals AMAX1 slechts reals als input en als output kent, en DMAX1 alleen double precision data-typen.

Het voordeel van gebruik van generieke functies is dat een bewerking altijd een uitkomst geeft onafhankelijk het type van de input, hetgeen goed van pas komt bij overschakeling van bijvoorbeeld reals naar double precision. Het voordeel van specifieke functies is dat er een extra check is op het data-type en dus ook (bij expliciete declaraties) een extra check op typefouten.
Het is moeilijk te zeggen welke methode de beste is. Zeker hierbij is het van belang één lijn te trekken. Bij het gebruik van expliciete declaraties is het overigens verplicht zowel specifieke als generieke functies expliciet te declareren als INTRINSIC, bijvoorbeeld:

```
      Intrinsic dmax1
```

## 8.7 Lengte subroutines en functies

Zoals in hoofdstuk 3 is uiteengezet ligt het voor de hand programma's in stukjes op te delen in de vorm van subroutines en functies. Een optimale grootte van functies en subroutines is echter moeilijk te geven. In veel leerboeken wordt uitgegaan van maximaal een A4'tje FORTRAN-code per routine (ongeveer 50 regels aan statements). In grote programma's, die, in afwijking van de optimale situatie, vaak "gaandeweg" tot stand gekomen zijn, is een dergelijke norm moeilijk te handhaven en ook niet altijd even gemakkelijk.

Het opdelen van het programma moet altijd kritisch beschouwd worden. Het is bijvoorbeeld niet zinnig om rigide te streven naar een gelijke lengte van de subroutines. De statements binnen een routine moeten in de eerste plaats inhoudelijk een functionele eenheid vormen. Voorkomen moet worden dat er te veel code binnen een DO-loop of een IF-THEN-ELSE statement komt te staan, zodat begin en eind van de structuur moeilijk te onderscheiden zijn. Als een controlestructuur langer dan circa een A4 wordt, kan de routine beter opgedeeld worden.

## 8.8 Read en write statements

Het is een goede gewoonte om binnen FORTRAN 77-programma's alle READ-statements te beveiligen met een ERROR-label en END-label; of met de IOSTAT-indicator. Deze toevoegingen zorgen er voor dat bij een inleesfout of indien het einde van de file bereikt is, het programma naar een bepaalde plaats springt waar bijvoorbeeld een foutmelding kan worden gegeven. De IOSTAT-indicator geeft meer gespecificeerd aan welke fout opgetreden is.

```
      read (lin,*, end = 1000, err = 2000) a,b,c,d
1000  call error ('einde file, file ..............')
2000  call error ('fout bij inlezen, file .......')
```

of

```
      read (lin,*, iostat = is) a,b,c,d
      if (is.ne.0) then
         call error ('fout bij inlezen, file .......', is)
      end if
```

De unitnummers die in het programma gebruikt worden, kunnen het beste bovenaan in het hoofdprogramma vastgelegd worden met PARAMETER-statement. Het doorgeven van de unitnummers naar verschillende subroutines kan vervolgens via argument overdracht plaatsvinden. Op deze manier is het eenvoudig om een programma geschikt te maken voor een andere computer, met andere unitnummer-definities; het programma hoeft dan slechts op één plaats gewijzigd te worden.

```
      parameter (iin = 10)
      read (iin,*,iostat = is) a,b,c,d
```

## 8.9 Groepering van compiler-afhankelijke functies

Ten behoeve van de verhoging van overdraagbaarheid van programmatuur tussen diverse (FORTRAN) compilers, is het raadzaam om de in dit verband specifieke functies in aparte subroutines te plaatsen. Dit betreft bijvoorbeeld:

- openen van invoer- en uitvoerfiles;
- schermafhandeling;
- plotten;
- behandeling van overige randapparatuur;
- opvragen van tijd en datum.

## 8.10 Resumé

- Maak binnen een programma iedere keer de afweging tussen duidelijkheid, foutengevoeligheid en snelheid, waarbij duidelijkheid voorop staat (bijvoorbeeld impliciete/expliciete declaratie, generieke/specifieke functies).
- Initialiseer alle gebruikte programmavariabelen.
- Gebruik het SAVE-statement voor lokale variabelen, waarvan de waarde vastgehouden moet worden, dan wel plaats de variabelen in een COMMON-block.
- Deel een programma op in subroutines en functions, waarbij als streefgrootte geldt maximaal 50 regels aan statements.

# 9 FOUTENCONTROLE EN FOUTMELDINGEN

## 9.1 Inleiding

Elke gebruiker zal wel eens het onverwachte en vroegtijdige beëindigen van een computerprogramma meegemaakt hebben. Er zijn verschillende manieren waarop de gebruiker op de hoogte wordt gebracht van de aard van de opgetreden fout. Wat valt er bijvoorbeeld te denken van de volgende melding op het beeldscherm:

"Mathoverflow, trap in segment 0A38 7003",

of een ander, voor de meesten even onbegrijpelijke boodschap:

"Heap size exceeded (open file xxxx)".

Een melding die door de gebruiker wellicht nog het beste te begrijpen valt, maar waar hij/zij evenmin iets mee kan beginnen, zou kunnen zijn

"Zero divide in line xxx of module YYY".

De hiervoor gegeven meldingen zijn van het FORTRAN-systeem zelf afkomstig, dus niet tevoren door de programma-ontwikkelaar geprogrammeerd.

Duidelijk is dat het anders moet. De gebruiker zal in principe een melding moeten krijgen over wat hij nu precies fout heeft gedaan. De programma-ontwikkelaar zal dus in de huid van de gebruiker moeten kruipen en zich een voorstelling moeten maken welke vergissingen c.q. fouten gemaakt kunnen worden. Een goede documentatie is dan een eerste stap in de juiste richting (hoofdstuk 12). Er is echter meer nodig. Invoergegevens zullen getoetst moeten worden: liggen de opgegeven waarden binnen redelijke grenzen? Zo niet, dan zal in ieder geval een waarschuwing afgedrukt moeten worden. Het afdrukken van enkele statistische gegevens over de invoergegevens kan ook helpen bij het opsporen/constateren van fouten. Denk bijvoorbeeld aan: minimum, maximum en gemiddelde van parameter-waarden, som van oppervlakten van elementen in numerieke berekeningen, etc. Naast fouten aan gebruikerszijde kunnen er ook in de broncode 'geprogrammeerde fouten' zitten. Bijvoorbeeld een delen-door-nul conditie, die niet voorzien was maar wel reëel mogelijk is. Of denk aan matrix-operaties die onder bepaalde voorwaarden niet stabiel zijn. Bij het programmeren zullen deze condities ondervangen moeten worden. Niet altijd een eenvoudige zaak, maar wel belangrijk. In het navolgende wordt een aantal praktische aanwijzingen gegeven met betrekking tot foutencontrole en foutafhandeling.

## 9.2 Invoer van data

Invoer van data dient overzichtelijk en gestructureerd plaats te vinden. Dit voorkomt onnodige fouten.

Een duidelijke omschrijving van invoerparameters met hun dimensies/eenheden moet in de documentatie worden opgenomen (hoofdstuk 12).

Indien er sprake is van een speciaal voorbewerkingsprogramma voor de invoer van gegevens (zie bijvoorbeeld hoofdstuk 3, Figuur 3.1) dient er een goede afstemming te zijn tussen foutencontrole in de voorbewerker en in het hoofdprogramma. Alleen als het invoerprogramma standaard onderdeel uitmaakt van het totale programmapakket kan hier een belangrijk deel van de foutencontrole worden uitgevoerd. Zo niet, dan dient één en ander (ook) in het hoofdprogramma plaats te vinden.

Bij het opzetten van de programmastructuur dient al rekening gehouden te worden met de foutafhandeling. Hierdoor is een efficiënte en doorzichtige programmering mogelijk.

## 9.3 Foutencontrole

Plaats het invoeren van data zo veel mogelijk bij elkaar in het programma. Het programmeren van de foutencontrole wordt daardoor eenvoudiger. Daarbij moet gewerkt worden in drie stappen: inlezen, afdrukken en controleren. Alle ingelezen data moeten ook kunnen worden afgedrukt (let daarbij ook op interactieve invoer). Houd er bij het afdrukken van gegevens rekening mee dat gegevens, die ingelezen kunnen worden, ook afgedrukt moeten kunnen worden (in FORTRAN 77: afstemming van FORMAT statements). Door inlezen en afdrukken zo veel mogelijk bijeen te plaatsen, is het relatief eenvoudig controles in te bouwen voor de uitvoering van lees- en schrijfopdrachten. Foutcondities tijdens inlezen, bijvoorbeeld real getal waar integer getal wordt verwacht of end-of-file conditie, kunnen dan goed opgevangen worden (inbouwen van foutmelding).

Ga voor alle invoergegevens na of reële waarden opgegeven zijn (doorlatendheid groter dan nul, porositeit tussen 0 en 1, etc.). Irreële waarden kunnen namelijk in een later stadium van het programma moeilijkheden opleveren. Zo kunnen bijvoorbeeld bij het opstellen van

een stelsel vergelijkingen systeem-matrices ontstaan die niet voldoen aan de eisen die gesteld worden aan de geprogrammeerde oplosprocedure. Voor het opsporen van fouten kan het bijzonder nuttig zijn om enkele algemene gegevens af te drukken die toch al door het programma berekend worden. Denk bijvoorbeeld aan minimum en maximum coördinaten, oppervlakte modelgebied, kortste en langste zijde van elementen, enzovoorts. Neem waarschuwingen op indien er naar alle waarschijnlijkheid fouten in de invoer kunnen zitten. Stop de programma-executie bij ernstige fouten, natuurlijk onder vermelding van een duidelijke foutboodschap.

## 9.4 Foutmeldingen

Foutmeldingen dienen niet (alleen) een fout aan te geven maar moeten ook een verwijzing inhouden naar de oorzaak van de fout en eventueel de plaats in het programma waar de fout optreedt. Het melden van een 'delen-door-nul'-conditie of 'log-uit-niet-positief-getal' heeft weinig zin als niet aangegeven wordt welke uitdrukking nul of niet positief is. Overigens zouden deze condities voorkómen moeten worden door een goede controle van de invoer. Een controle op de array-grensoverschrijdingen dient op kritieke punten uitgevoerd te worden, bijvoorbeeld voor de uitvoering van omvangrijke DO-loop operaties. Dit type condities kan tot niet-traceerbare fouten leiden. Door het gebruik van adjustable arrays en/of het PARAMETER statement kan een goede afstemming tussen array-dimensie en foutencontrole plaatsvinden (zie voorbeeld 1).

Voorbeeld 1

```
      subroutine mfbiin (iin, iout, nn, ne)
      parameter (nnmax=1000, nemax=750)
      common /nodpar/ gwl(nnmax),
      common /elepar/ try(nemax), rvf(nemax)
      logical lerror
      .....
      .....
      lerror = .false.
      .....
      .....
      read (iin,8010) nn, ne
      if (nn.gt.nnmax) then
        write (iout,8050) nnmax, nn
        lerror = .true.
      endif

      if (lerror) stop '%%-Error in Input Data-%%'

8010 format (2I5)
8050 format (' %%-Error IN337-%% Subroutine MFBIIN',
     &       /' Maximum Number of Nodes Exceeded',
     &       /' Max #nodes = ',i4,10x,'Input Value = ',i4)
```

Het vermelden van een foutnummer in de foutmelding schept de mogelijkheid om in een gebruikershandleiding ruimer aandacht te besteden aan de optredende fout.

## 9.5 Foutafhandeling

Routines met een algemeen karakter dienen een fout te signaleren en deze door te geven aan de aanroepende routine door middel van een foutvlag. In de routine waar de fout optreedt wordt wel een foutmelding afgegeven, maar de routine zelf zorgt niet voor beëindiging van het programma. De foutvlag kan bestaan uit een integer getal die het type fout aanduidt of een logical variabele.

Voorbeeld:

IFOUT = 0 geen fout opgetreden

IFOUT = 1 onjuiste argumentwaarden

IFOUT = 2 geen convergentie

of

LERROR = .FALSE. geen fout opgetreden

LERROR = .TRUE. fout opgetreden

Geef in de programmadocumentatie duidelijk aan wat de betekenis is van de verschillende waarden van de foutvlag. Overweeg het gebruik van een aparte subroutine voor het afdrukken van foutmeldingen. Dit voorkomt het herhaaldelijk programmeren van dezelfde melding en bevordert de herkenbaarheid van de foutmeldingen. Denk bijvoorbeeld aan fouten bij input/output-instructies. Elke leesopdracht kan zo bijvoorbeeld gemakkelijk voorzien worden van een END-OF-FILE label (zie voorbeeld 2).

Voorbeeld 2

```
      read (iin,1030, err=9010, end=9030) (gwl(ix),ix=1,nx)
```

```
      call hyderr (iout, 1020, iin,  'MFGHIN', 'Groundwater level')
9030 .....
      .....
```

```
      subroutine hyderr (iout, ierror, iio, csubr, cinfo)
      character csubr*(*), cinfo*(*)
      .....
      .....
      if (ierror.eq.1020) write (iout,8020) ierror, csubr, iio, cinfo
      .....
8020 .....
      format (' %%-Error-%% Number ',i5,
     &       /' End-of-file detected in subroutine ',a,
     &       /' Reading unit ',i5,10x,'Parameter ',a)
```

## 9.6 Resumé

- Voer data overzichtelijk in en doe dit gestructureerd, dit voorkomt onnodige fouten.
- Houd reeds bij de opzet van een programma(pakket) rekening met foutafhandeling.
- Concentreer zo veel als mogelijk het invoeren van data, de foutencontrole wordt daardoor vergemakkelijkt.
- Zorg voor een uitvoerige controle op fouten, zoals irreële parameterwaarden en foutieve combinaties van (invoer)parameters.
- Foutmeldingen moeten niet alleen de aard van de fout aangeven maar ook een verwijzing inhouden naar de oorzaak van de fout en de wijze waarop het probleem kan/moet worden verholpen.
- Zorg voor signalering van array-overschrijdingen.
- Nummer of benoem anderszins systematisch de foutmeldingen, dit geeft de mogelijkheid om in een gebruikershandleiding ruimere aandacht aan de foutafhandeling te besteden.

# 10 TESTEN VAN PROGRAMMATUUR

## 10.1 Inleiding

Om een inzicht in de kwaliteit van programmatuur te krijgen is een goed georganiseerde testaanpak vereist. Vragen met betrekking tot functionaliteit, performance, betrouwbaarheid, beheersbaarheid, onderhoudbaarheid, beveiliging en documentatie kunnen uitsluitend beantwoord worden als het testproces planmatig wordt aangepakt. Omdat de meeste fouten gemaakt worden in het begin van het ontwikkeltraject en de kosten om een fout te herstellen toenemen naarmate het tijdstip van ontdekken later valt, is het van groot belang dat in alle fasen van de levenscyclus (voorbereidende) testactiviteiten plaatsvinden. Dit hoofdstuk gaat voornamelijk in op het testen van programmatuur in software-technisch opzicht. Of de door de programmatuur geïmplementeerde theorie hydrologisch gezien correct is blijft hier buiten beschouwing.

## 10.2 Aanpak bij testen

Al tijdens het opstellen van de programma-eisen en het globale ontwerp van de te bouwen programmatuur wordt aandacht besteed aan testen in de vorm van het specificeren van de tests die in latere stadia uitgevoerd moeten worden om de functionaliteit en performance te controleren (verificatie eisen) en het systeem in gebruik te nemen (acceptatie eisen). Dit betekent dat alle eisen die aan de programmatuur gesteld worden in principe testbaar moeten zijn, dat wil zeggen dat ze binnen een eindige hoeveelheid tijd gecontroleerd moeten kunnen worden.

Tijdens de implementatie worden alle afzonderlijke onderdelen getest en vervolgens geïntegreerd in subsystemen en uiteindelijk in het complete werkende systeem (computer programma). Bij deze incrementele opbouw van het systeem kan met testen worden begonnen bij zowel de modules op het laagste niveau (bottom-up testen, waarbij de omgeving van de module met behulp van een test-driver wordt gesimuleerd) als bij modules op het hoogste niveau (top-down testen, waarbij modules op een lager niveau door zogenaamde teststubs (dummy routines) geëmuleerd worden). Met het testen wordt nagegaan of de afzonderlijke modules aan de gestelde specificaties voldoen (moduletest), of

ze samen het beoogde resultaat opleveren (integratietest) en of de documentatie en gebruikershandleiding in overeenstemming zijn met de programmatuur (systeemtest).

Na de (voorlopige) oplevering van de programmatuur wordt het systeem getest op basis van de gebruikers specificaties (applicatietest). Hier ligt de nadruk op het testen van de bruikbaarheid van de geleverde functionaliteit. Hierbij dient aangetekend te worden dat het voor een groter systeem onmogelijk is alle paden door de programmatuur te testen.

Tenslotte wordt getest of het systeem zich in de produktie-omgeving net zo gedraagt als in de omgeving waarbinnen het ontwikkeld werd (installatietest). Hierbij dient aangegeven te worden onder welke condities de executable was aangemaakt (compiler, linker, etc.). Eventueel kan ook aandacht worden besteed aan het schatten van de betrouwbaarheid van de programmatuur, bijvoorbeeld met behulp van betrouwbaarheidsmodellen.

De resultaten van alle genoemde tests dienen vastgelegd te worden in gestandaardiseerde documenten en beschikbaar te zijn in latere fasen van het ontwikkelingstraject. Dit houdt onder andere in dat in de produktiefase, nadat het programma is opgeleverd, beschikt kan worden over een aantal bij het programma behorende testproblemen. De documentatie van deze tests omvat in ieder geval een beschrijving van het probleem, een complete opsomming van de invoer data, alle bij het probleem behorende uitvoer en een verklaring van de resultaten en eventuele afwijkingen. Deze tests kunnen te allen tijden gebruikt worden om te controleren of het programma, bijvoorbeeld na installatie op een ander systeem, nog steeds het juiste gedrag vertoont. Waar mogelijk is het raadzaam om de toekomstige gebruikers bij het ontwerp van de tests te betrekken.

## 10.3 Testmethoden

Zoals in de vorige paragraaf al werd opgemerkt, is het in de praktijk vrijwel altijd onmogelijk het correct functioneren van een programma te bewijzen. Hiertoe zou de programmatuur namelijk uitputtend getest moeten worden, zodat alle mogelijke paden in de de programmatuur doorlopen zouden worden. Dit is in het bijzonder het geval bij geohydrologische programma's die een betrekkelijk kleine gebruikersgroep kennen en waar dus veel inspanning bij het testen vaak niet opweegt tegen de "opbrengst" daarvan. Dit is

overigens juist een van de gesignaleerde knelpunten die tot de instelling van de CHO-TNO Werkgroep RCPH (dit rapport) hebben geleid. Om economische en praktische redenen is het van groot belang een zo klein mogelijke (maar voldoende representatieve) verzameling van testgevallen vast te stellen. Uitgangspunt hierbij kan zowel de specificatie als de implementatie van de programmatuur zijn (functionele respectievelijk structurele analyse). In het eerste geval bestaat er geen theoretisch onderbouwde methode om de verzameling testgevallen te bepalen, maar wordt aan de hand van de eigenschappen van de invoer van het programma een set samengesteld. Hierbij wordt zowel met representatieve waarden (bijvoorbeeld klassegemiddelden) als met uitzonderingsgevallen (extreme waarden) rekening gehouden. Ook andere, onafhankelijk verkregen (bijvoorbeeld analytische) oplossingen kunnen hierbij worden gebruikt. In het tweede geval zijn de tests gebaseerd op de interne structuur van de programmatuur en bestaan er hulpmiddelen (computer aided testing (CATE), software analyse tools en dergelijke) om vast te stellen in hoeverre de tests de programmatuur volledig dekken. Hierbij kan geprobeerd worden met de tests 100% "dekking" te bereiken of om de complexiteit van de onderzochte code die van de gehele programmatuur te laten naderen (wat in beide gevallen overigens geen garantie voor correctheid is).

In de praktijk worden in de ontwerp- en implementatiefase naast de bovengenoemde technieken een aantal methoden gebruikt om fouten in ontwerp en programmatuur op te sporen die niet theoretisch onderbouwd zijn, maar die wel op duidelijk afgesproken regels gebaseerd kunnen worden:

- het zorgvuldig doorlezen van de code door de programma-ontwikkelaar zelf. Nadeel hierbij is de (onwillekeurige) neiging die iedere programmeur heeft om die testgevallen te selecteren die aantonen dat de code wèl aan de specificaties voldoet;

- het in teamverband doorlopen en met behulp van testgegevens executeren van het programma. Hierbij kunnen zowel de bij het ontwerp als de bij de implementatie gemaakte keuzes worden verklaard en bediscussieerd. Bovendien kunnen aan de hand van de ervaring van de teamleden frequent voorkomende fouten gedetecteerd worden.

## 10.4 Resumé

- Stel een verzameling van testgevallen samen, waarbij (vaak aan de hand van invoer) rekening wordt gehouden met zowel representatieve als extreme waarden.
- Test alle onderdelen (modulen) tijdens de implementatie eerst afzonderlijk. Test vervolgens modulen geïntegreerd in subsystemen en uiteindelijk in het complete systeem (programma).
- Lees in ieder geval zorgvuldig de code door en beproef het programma in teamverband met behulp van testgegevens.

# 11 INTERNE DOCUMENTATIE IN COMPUTERCODE

## 11.1 Inleiding

Met interne documentatie wordt de tekst bedoeld, die als commentaar wordt toegevoegd aan de broncode van een programma.

De interne documentatie is een belangrijk middel om specifieke informatie van het programma en de werking vast te leggen voor de programma-ontwikkelaar en mogelijke toekomstige ontwikkelaars. De informatie kan nodig zijn voor onderhoud of wijzigingen aan het programma.

De hoeveelheid informatie die gegeven wordt is afhankelijk van:
- het gebruik van het programma en de levensduur ervan;
- de omvang van het programma;
- de complexiteit van het programma.

Een programma dat eenmalig gebruikt wordt en vervolgens vernietigd, behoeft niet van commentaar voorzien te worden. Dit geldt in zekere mate ook als het programma alleen door de ontwikkelaar zelf wordt gebruikt. Bedacht dient echter te worden dat ook als het programma alleen door de ontwikkelaar wordt gebruikt, een redelijk niveau van documentatie aan te bevelen is omdat hij/zij na verloop van tijd zelf ook alle details niet meer weet. Een groot en ingewikkeld programma dat overgedragen zal worden aan anderen voor gebruik en onderhoud moet volledig gedocumenteerd worden.

De interne documentatie wordt verdeeld in twee categorieën:
- informatie die vooraan in de routine gegeven wordt (zogenaamde "program header");
- informatie tussen de programma-instructies van de routine.

De informatie voor de routine is algemeen van aard (zoals de naam van het programma waar hij deel van uit maakt, een korte omschrijving van de routine, de datum en programmeur). De informatie geplaatst tussen de programma-instructies verduidelijkt de acties die uitgevoerd worden. Dit commentaar moet met mate toegepast worden om het overzicht over de instructies niet te verliezen. Indien een heldere lay-out voor de instructies gebruikt wordt, kan met enkele aanvullende opmerkingen volstaan worden.

De interne documentatie dient visueel gescheiden te worden van de programma-instructies.

## 11.2 Documentatie aan begin van routine

Volledige documentatie aan het begin van een routine, inclusief het hoofdprogramma bestaat uit:

- naam van het programma + versienummer* + datum;
- vermelding van het pakket als het programma deel uit maakt van een pakket;
- doel/actie van het programma met uitgangspunten en beperkingen, invoer en uitvoer en bijzonderheden;
- implementatie*:
  - . machine, operating system;
  - . compiler en linker met opties en versienummers;
  - . bibliotheken;
- lijst met beperkende dimensies (van arrays bijvoorbeeld) en hoe dit aangepast kan worden;
- programmeur(s) met bedrijf;
- beheerder;
- copyright/eigenaar;
- update informatie: wanneer welke wijzigingen in de brontekst zijn aangebracht;
- gebruikte methode, met zonodig referentie naar artikel of boek;
- opsomming van argument variabelen en gebruikte common block variabelen:
  - . vermelding type (CHARACTER, LOGICAL, COMPLEX etc.);
  - . vermelding betekenis;
- vermelding van INCLUDE files met beschrijving van inhoud;
- opsomming van aangeroepen routines;
- opsomming van Input/Output files.

N.B.: Onderdelen die alleen voor een hoofdprogramma van toepassing zijn, zijn door middel van een * aangegeven.

## 11.3 Documentatie tussen de regels met FORTRAN-instructies

De documentatie tussen de regels met broncode moet erop gericht zijn om, in samenhang

met de broncode de acties toe te lichten die ter plaatse door het programma worden uitgevoerd. De documentatie moet summier zijn, zodat de broncode niet over een onoverzichtelijk grote lengte wordt uitgesmeerd. Er kan verwezen worden naar de externe documentatie, zoals de handleiding, beschrijvingen van gebruikte algoritmen.
Een tweede deel van de documentatie in de broncode is het aangeven van wijzigingen, bij voorkeur met datum en auteur. Voor toevoegingen kan bovendien een andere letter worden gebruikt (hoofdletters als oorspronkelijke code in kleine letters is).

Het commentaar dient visueel gescheiden te worden van de code. Dit kan op verschillende manieren geschieden, bijvoorbeeld:
- scheiding door regel met *....., *------, *======= etc.;
- scheiding door regel met c....., c------, c======= etc.;
- instructies in kleine letters en commentaar in hoofdletters, of andersom.

Het eerste teken (in kolom 1) van een commentaarregel dient een "*" of een "c" te zijn.

Delen van de FORTRAN-code met verschillende taken kunnen ook visueel gescheiden worden met behulp van lege regels, zodat de instructies, die samen een taak vervullen als een groep herkenbaar zijn.

De commentaartekst kan als volgt worden gepositioneerd:
- Elke commentaarregel beginnen op een vaste positie (kolom), bijvoorbeeld op positie 4;
- Commentaartekst inspringen afhankelijk van de beginpositie van de broncode, waarop het desbetreffende commentaar betrekking heeft.

## 11.4 Voorbeelden

Voorbeeld 1 Subroutine SCPP06 (RIVM, Bilthoven, K. Kovar)

Dit is een algemene routine die gebruikt wordt door diverse programma's van de zogenaamde AQ-programmapakketten van het RIVM. De AQ-programmapakketten worden niet alleen intern maar ook extern gebruikt. Onderhoud en aanpassingen zijn te voorzien en dus is de documentatie vrij uitgebreid.

```
************************************************************************
*                                                                      *
*                    >>  SUBROUTINE SCPP06  <<                         *
*                                                                      *
*----------------------------------------------------------------------*
*  purpose : Read 3 records from the file IIN. The data contain the    *
*            following "XP" records for the profiles:                  *
*            1) record XPFG                                            *
*            2) record YPFG                                            *
*            3) record MPPFG                                           *
*----------------------------------------------------------------------*
*  parameters :                                                        *
*                                                                      *
*  IIN    = logical unit number of input file                  (i)     *
*           (integer variable)                                         *
*  ESC    = ASCII character 27                                 (i)     *
*           (character variable)                                       *
*  CBLON  = character string for switching bold char. on       (i)     *
*           (character*5 variable)                                     *
*  CRVOF  = character string for switching reverse video off   (i)     *
*           (character*5 variable)                                     *
*  NPPFG0 = number of points along the profile IPF             (i)     *
*           (integer variable, 1 < NPPFG0 =< NPPFGX)                   *
*  XPFG   = x-coordinates of profile IPF                       (o)     *
*           (real array, size >= NPPFGX)                               *
*  YPFG   = y-coordinates of profile IPF                       (o)     *
*           (real array, size >= NPPFGX)                               *
*  MPPFG  = definition of original/extra points on profile IPF (o)     *
*           (integer array, size >= NPPFGX)                            *
*  IREC   = number of the "XP" record                          (i/o)   *
*           (integer variable, 1 =< IREC(input))                       *
*  FLERR  = flag of whether an error message was issued (yes:t) (o)    *
*           (logical variable)                                         *
*----------------------------------------------------------------------*
*  subprograms    : iorfai, iorfar, scmcu2                             *
*----------------------------------------------------------------------*
*  error messages : 1                                                  *
*----------------------------------------------------------------------*
*  implementation : 1) Ensure that  1 =< IREC(input),                  *
*                      IREC(output) = IREC(input) + 3                  *
*                   2) The input and output value of IREC concerns     *
```

```
*                     the next record to be loaded or skipped.       *
*--------------------------------------------------------------------*
*  Copyright RIVM, The Netherlands         Last update : 21-MAR-1991  *
**********************************************************************
      subroutine scpp06 (iin, esc, cblon, crvof, nppfg0, xpfg, ypfg,
     &                   mppfg, irec, flerr)
      implicit double precision (a-h,o-z)
*---------------------------------------------------------------------
      dimension xpfg(*), ypfg(*), mppfg(*)
      character esc, cblon*5, crvof*5, txt*37, help*49
      logical   flerr
*---------------------------------------------------------------------
*  Set the text for the error message
*
      data txt/'Error on Loading Profile Data Record '/
*---------------------------------------------------------------------
*  Read XPFG
*
      call iorfar (iin, xpfg, nppfg0, flerr)
      if (flerr) goto 10
      irec    = irec + 1
*---------------------------------------------------------------------
*  Read YPFG
*
      call iorfar (iin, ypfg, nppfg0, flerr)
      if (flerr) goto 10
      irec    = irec + 1
*---------------------------------------------------------------------
*  Read MPPFG
*
      call iorfai (iin, mppfg, nppfg0, flerr)
      if (flerr) goto 10
      irec    = irec + 1
      return
*---------------------------------------------------------------------
*  Write error message in the message window
*
   10 if (irec.lt.10) then
         write (help,'(a37,''XP,''i1)') txt, irec
      elseif (irec.gt.9 .and. irec.lt.100) then
         write (help,'(a37,''XP,'',i2)') txt, irec
      else
         write (help,'(a37,''XP,'',i3)') txt, irec
      endif
      call scmcu2 (esc, cblon, crvof, 22, 25, help, 49)
      return
      end
```

Voorbeeld 2 Hoofdprogramma van FLAIRS (IWACO B.V., Rotterdam)

Dit is het hoofdprogramma van het centrale rekenprogramma (hoofdbewerkingsmodule) van het eindige elementen pakket TRIWACO. De informatie over de werking van het programma is niet in de kop boven de routine verwerkt, het is in de externe documentatie gegeven.

```
************************************************************************
*                                                                      *
*                      MAIN Program of FLAIRS                          *
*                                                                      *
*   part of TRIWACO (Finite Element Package for simulating             *
*    Groundwater Flow)                                                 *
*    (c) IWACO, Rotterdam                                              *
*    IWACO B.V. Consultants for Water and Environment                  *
*    P.O. Box 183                                                      *
*    3000 AD  Rotterdam                                                *
*    the Netherlands                                                   *
*                                                                      *
************************************************************************
*                                                                      *
*    Version    : 2.67   1/23/1990                                     *
*    Programmer : A. Leijnse                                           *
*                 C.T. de Graaf                                        *
*                 M.J. Emke                                            *
*                                                                      *
*    Operating                                                         *
*    System     : Microsoft DOS Version 3.3                            *
*                 IBM DOS Version 3.3                                  *
*                 IBM OS/2 Version 1.10                                *
*                                                                      *
*    Compiler   : Microsoft FORTRAN Version 5.0                        *
*                 (work arrays put in common block in main program     *
*                 to prevent compilation error DATA GREATER THAN       *
*                 64K)                                                 *
*                                                                      *
*    Libraries  : LLIBFORE.LIB (no C compatibility, Huge Memory        *
*                 Model, in line Math Emulation) FLADOLIB.LIB          *
*                                                                      *
*    Routines   : FLAIRS, FLAWOR, START, FRPAGE, RESREA, INNUIN,       *
*                 DIM1, INPGRD, CREBP, DETFRA, INRIUT, INSOUR,         *
*                 INBOUN, INRIVE, INPARM, FILPAR, DIM2, POINT,         *
*                 DIM3, GRADAT, INMAFI, INTIDA, STUMAT, RECMAT,        *
*                 SOUMAT, RIVMAT, BOUMAT, ADDAD, LEAMAT, CONGRA,       *
*                 DTNEW, DEBUPD, WRIFLA, PARBAL, PARCBA, BALANS,       *
*                 GRAFIL, CUMBAL, PROUT, RESWRI, WRIMAP, HEADER,       *
*                 CAPITAL, PAREPR, STNEEL, AFVOER, HORPOS, STORE,      *
*                 FRETRA, VECPRO, SCALE, PAINPR, MATVEC, WRIGRA,       *
*                 FRESTO, TRANTR, INTFAC, CHPOAR, FILLRP, MATFLO,      *
*                 NNDRAIN, DRAINDIT, PRECPOL, DITCH, NNEERST,          *
*                 CAPFLOW, SOILIB, CHANNEL, PRECIP, FILLIN, DREGYP,    *
*                 NNPOLINF, DRAIN, MOZAMBIK                            *
*                                                                      *
*                 stored in files with the same name and extension.    *
*                 FOR except for flairs: use FLAIRSOS.FOR for OS/2     *
```

```
*                                                    FLAIRSDOS.FOR for DOS   *
*  Parameters    : NDR    integer       size of real work arrays       *
*                : NDI    integer       size of integer work arrays    *
*                                                                      *
*  Variables     : R1     real array    storage array                  *
*                  R2     real array    storage array                  *
*                  R3     real array    storage array                  *
*                  R4     real array    storage array                  *
*                  R5     real array    storage array                  *
*                  R6     real array    storage array                  *
*                  I1     integer array storage array                  *
*                  I2     integer array storage array                  *
*                  IN     integer       logical unit for input file    *
*                  IOUT   integer       logical unit for print file    *
*                  IGRD   integer       logical unit for grid file     *
*                  IFM    integer       logical unit for output file   *
*                  IFRS   integer       logical unit for restart       *
*                  IFIL   integer       logical unit for parameter file*
*                  ISCR1  integer       logical unit for scratch file  *
*
*
**********************************************************************

      PROGRAM FLAIRS

*     set size of work arrays (which determines the capacity of the
      program;
*     the maximum number of nodes, the maximum number of layers and the
*     maximum number of equations are interrelated)

      PARAMETER (NDR=180000)
      PARAMETER (NDI=NDR)

      COMMON /R1/ R1(NDR)
      COMMON /R2/ R2(NDR)
      COMMON /R3/ R3(NDR)
      COMMON /R4/ R4(NDR)
      COMMON /R5/ R5(NDR)
      COMMON /R6/ R6(NDR)

      COMMON /I1/ I1(NDI)
      COMMON /I2/ I2(NDI)

*     set logical unit numbers

      DATA  IIN, IOUT, IGRD, IFM, IFRS, IFIL, ISCR1
      DATA / 1 ,  2  ,  3  ,  4 ,  5  ,  6  ,  7   /

*     call central routine

      CALL FLAWOR(I1, I2, R1, R2, R3, R4, R5, R6, NDI, NDR,
                  IIN, IOUT, IGRD, IFM, IFRS, IFIL, ISCR1)

      STOP' end  F L A I R S'

      END
```

## 11.5 Resumé

- Voorzie elke routine van een kop van commentaar regels met algemene informatie.
- Verduidelijk werking van het programma door commentaar regels tussen de broncode.
- Scheid functionele onderdelen visueel van de broncode met behulp van regels met scheidingstekens (*-------- etc.) en/of lege regels.
- Maak de documentatie direct bij aanvang van het programmeerwerk, niet achteraf. Het komt dan in de verdrukking.

# 12 PROGRAMMAHANDLEIDING

## 12.1 Inleiding

Dit hoofdstuk richt zich op de beschrijving van de documentatie-eisen die moeten gelden voor de algemeen toepasbare hydrologische programmatuur. Vaak is dit de programmatuur die aan derden als broncode beschikbaar wordt gesteld. De beschrijving heeft dus geen betrekking op allerlei ad-hoc programmatuur die voor eigen gebruik of voor gebruik in beperkte kring bedoeld is, hiervoor behoeven uiteraard minder verregaande eisen gesteld te worden.

Het verschil tussen documentatie van wetenschappelijke programmatuur (waar onder hydrologische) en andere gebruiksprogrammatuur, is dat ervan uitgegaan moet worden dat bij bepaalde omgevingen de gebruiker niet alleen als "echte" gebruiker bij de programmatuurontwikkeling betrokken is. Dat wil zeggen dat de programmatuur geen "Black Box" mag zijn; niet alleen de in- en output maar ook de inhoud moet zowel theoretisch als op programmacode-niveau uitputtend beschreven worden. De mogelijkheid moet geboden worden om "verdachte" uitkomsten op codeniveau te herleiden. Daarnaast moet de programmatuur (wetenschappelijk!) te manipuleren zijn. De kring van gebruikers van hydrologische programmatuur is in vergelijking met die van andersoortige wetenschappelijke programmatuur relatief klein, zodat modellen hoogst zelden uitputtend getest kunnen worden. Weliswaar is er tegenwoordig programmatuur die alle paden in een programma kan doorlopen en testen. Maar vanwege de tijdsinspanning die dit vergt wordt het meestal achterwege gelaten. De noodzaak om als gebruiker op programmaniveau kennis te hebben, is dan ook wenselijk.

Het verschil tussen wetenschappelijke programmatuur en normale gebruiksprogrammatuur betekent dat de gebruikersdocumentatie al gauw een complete systeemdocumentatie hoort te zijn. Grofweg kunnen dan vijf delen onderscheiden worden:

- globale systeembeschrijving (doelgroep = potentiële gebruiker);
- theoretische systeembeschrijving (probleemaanpak);
- technische systeembeschrijving (programmatechnische realisatie);
- gebruikersbeschrijving (doelgroep = eindgebruiker);
- programma-evaluatie (wetenschappelijk testverslag).

## 12.2 Globale systeembeschrijving

In de globale systeembeschrijving worden de noodzakelijke formaliteiten over de programmatuur vermeld en wordt een beeld gegeven van wat de programmatuur globaal inhoudt.

Tot de noodzakelijke formele gegevens die vermeld moeten worden behoren:

- de ontwikkelaar van de programmatuur;
- de contactpersoon en de beherende instantie;
- het versienummer en de datum;
- de minimumeisen aan de apparatuur;
- de minimumeisen aan de gebruiker (opleidingsniveau);
- basisdocumenten achter de programmatuur;
- trefwoorden.

In de globale beschrijving van de programmatuur wordt in grote lijnen aangegeven welke bewerking de programmatuur uitvoert en volgens welke methode, wat de in- wat de uitvoer is en wat de voornaamste beperkingen bij het gebruik zijn. De beschrijving is in de eerste plaats bedoeld om selectie van een programma voor het oplossen van een bepaald probleem mogelijk te maken. De potentiële gebruiker moet in één oogopslag kunnen concluderen of het programma geschikt is voor het probleem. De gegevens in de beschrijving moeten voldoende zijn om de programmatuur op een sluitende manier in een retrieval systeem te plaatsen. De beschrijving heeft daarnaast als doel om voor de gebruiker een begrippenkader te geven, waarmee de overige documentatie beter te lezen is. In deze beschrijving moeten dan ook min of meer alle aspecten aan bod komen die in de rest van de documentatie (uitgebreid) aan de orde komen. De globale systeembeschrijving wordt ook vaak als folder uitgegeven om potentiële gebruikers/kopers te attenderen.

## 12.3 Theoretische systeembeschrijving

De theoretische systeembeschrijving geeft de wetenschappelijke achtergrond van de programmatuur. De beschrijving moet een zodanige beschrijving geven van de probleemaanpak in de programmatuur, dat de gebruiker op grond van de beschrijving kan beoordelen of zijn specifieke probleem inhoudelijk afdoende door de programmatuur opgelost kan

worden. In feite moet de theoretische systeembeschrijving het hele rekenproces beschrijven in wiskundige termen, dan wel verwijzen naar publikaties waarin (delen van) het rekenproces beschreven zijn.

De beschrijving moet onder meer bevatten:

- stroomdiagrammen van de opvolging van de rekenacties en de plaats van de voornaamste grootheden hierin;
- beschrijving en afleiding van op te lossen differentiaal vergelijkingen;
- de oplossingsmethode van de vergelijkingen;
- alle gebruikte formules en herkomst ervan;
- een motivatie van de gebruikte methoden en hun beperkingen;
- een lijst van symbolen (zie Verklarende Hydrologische Woordenlijst (CHO-TNO, 1986));
- een uitgebreide literatuurlijst.

## 12.4 Technische systeembeschrijving

De technische systeembeschrijving beschrijft de vertaling van de theoretische systeembeschrijving naar programmatuur en wat daaraan vast zit. De beschrijving moet erop gericht zijn dat de gebruiker zelfstandig de programmatuur kan implementeren en de programmacode kan begrijpen met als doel, het traceren van (fouten in) modeluitkomsten en aanpassen/uitbreiden van de programmatuur.

De technische systeembeschrijving bevat:

- een beschrijving van de programmastructuur;
- een vocabulaire van variabelenamen en modulenamen;
- een beschrijving van de hardware-omgeving;
- een beschrijving van de software-omgeving;
- de programmacode.

Programmastructuur:

De beschrijving van de programmastructuur dient als kader waarbinnen de programmacode gelezen wordt. Met behulp van de structuurbeschrijving moet het mogelijk zijn om te traceren in welk deel van de programmatuur met name bepaalde input-variabelen van

waarde veranderen, in welke volgorde de bewerkingen op de input-data zich voltrekken etc. Ook moet de structuurbeschrijving aangeven waarom de bewerkingen zich in de aangegeven volgorde voltrekken (met name belangrijk bij wijzigingen).

De beschrijving van de programmastructuur bevat in ieder geval:
- een beschrijving van alle programmaniveaus;
- een stroomdiagram van minimaal de eerste drie niveaus van de programmatuur;
- een uitputtende lijst van subroutines en functies met documentatie van argument- en COMMON-variabelen.

Variabele- en modulenamen:
Het vocabulaire van variabele- en modulenamen dient als referentie bij het lezen van de programmacode. Bij het lezen van de programmacode moet op ieder moment uit het vocabulaire de betekenis van de variabelen en de modulen te herleiden zijn. Bij het samenstellen van het vocabulaire dienen de aanbevelingen hieromtrent in dit rapport (hoofdstuk 4 en 5), bij voorkeur nagevolgd te worden. De in- en output-variabelen en de variabelen behorend bij een COMMON-gebied moeten expliciet verklaard worden, evenals alle routinenamen.

Het vocabulaire bestaat derhalve uit:
- een beschrijving van de conventie aan de hand waarvan variabele- en modulenamen zijn samengesteld (vertaalsleutel);
- een alfabetische lijst van input-, output- en COMMON-variabelen met betekenis, eenheid en datatype;
- een opsomming van de COMMON-gebieden en variabelen hierin;
- een opsomming van alle routines en de betekenis.

Hardware-omgeving:
De hardware-omgeving bepaalt of de programmatuur in zijn volle omvang geïmplementeerd kan worden. De beperkende factoren zijn daarbij de geheugenruimte, de snelheid en de beschikbare randapparatuur.

Bij de beschrijving dient te worden vermeld:

- welk type computer (parallel/sequentieel) met welke geheugenruimte minimaal vereist is;
- welke veranderingen het beste in de programmatuur aangebracht kunnen worden om het geheugenbeslag te beperken en de snelheid te vergroten (bijvoorbeeld array-grenzen);
- welke randapparatuur benodigd is;
- welke modules uitgeschakeld kunnen worden om de hoeveelheid randapparatuur te beperken;
- op welke computer de programmatuur ontwikkeld en getest is.

Software-omgeving:

Evenals de hardware-omgeving, bepaalt de software-omgeving of de programmatuur in volle omvang geïmplementeerd kan worden. De beperkende factoren zijn daarbij de compiler, het operating systeem en externe softwarebibliotheken.

Bij de beschrijving dient te worden vermeld:

- voor welke compiler is de programmatuur geschreven en welke andere compilers ook geschikt zijn;
- welke wijzigingen moeten aangebracht worden in de programmatuur om deze voor andere compilers geschikt te maken;
- onder welk operating systeem de programmatuur ontwikkeld is;
- welke programmabibliotheken en welke modules hiervan zijn gebruikt;
- zijn deze modules meegeleverd of moeten ze apart aangeschaft worden (hoe, bij wie, prijs);
- welke programmagedeelten hoe uitgeschakeld kunnen worden indien de externe software niet beschikbaar is;
- literatuur met betrekking tot compiler, operating system, softwarepakket enzovoorts.

Programmacode:

Gezien de eerder beschreven problematiek, is bij hydrologische programmatuur het beschikbaar zijn van de broncode belangrijk. Deze maakt dan ook integraal deel uit van een goede programmadocumentatie. Voor eisen te stellen aan de FORTRAN-broncode zie hoofdstukken 7 en 8 van deze publikatie.

## 12.5 Gebruikersbeschrijving

De gebruikersbeschrijving beschrijft op welke manier de programmatuur van invoer voorzien moet worden, hoe de uitvoer geïnterpreteerd moet worden en wat gedaan moet worden bij voorkomende moeilijkheden.
De beschrijving bestaat uit 3 delen:

- invoerbeschrijving;
- uitvoerbeschrijving;
- beschrijving van de foutmeldingen.

Invoerbeschrijving:
De invoerbeschrijving moet ondubbelzinnig beschrijven welke gegevens nodig zijn voor het draaien van het programma en hoe ze ingevoerd moeten worden.
Voor alle invoervariabelen moet aangegeven worden:

- de programmavariabele naam;
- de betekenis van variabele (bijv. optievariabele);
- het datatype;
- het input-formaat;
- de wetenschappelijke naam (zie Verklarende Hydrologische Woordenlijst);
- de (SI-)eenheid;
- een minimum en maximum grootte;
- bij optienummers: een lijst met opties;
- een verwijzing naar formules uit de theoretische beschrijving (indien van toepassing).

Indien het interactieve invoer betreft, dienen de afzonderlijke invoerschermen (letterlijk) afgebeeld te worden in logische volgorde.

Indien invoer vanuit bestanden gegeven moet worden, moet een thematisch overzicht van de verschillende bestanden gegeven worden. Per bestand moet de invoer regelgewijs, met inachtneming van bovenstaande specificaties, gegeven worden. Tevens moet het type van het bestand gegeven worden (geformateerd, ongeformateerd, direct access etc.).

Uitvoer:

Aan de hand van de uitvoer worden de prestaties van het model beoordeeld. Bij de uitvoer moet duidelijk zijn welke grootheid gepresenteerd wordt en in welke vorm dit gebeurt. Hierbij moet aangegeven worden:

- naar welk medium de uitvoer gaat (schijf, scherm, plotter enzovoorts);
- de programmavariabele namen waarvan de uitkomst gepresenteerd wordt;
- de wetenschappelijke naam van de variabele (zie Verklarende Hydrologische Woordenlijst);
- de (SI-)eenheid;
- een minimum en maximum mogelijke grootte;
- een verwijzing naar formules uit de theoretische beschrijving (indien van toepassing);
- indien uitvoer naar bestand: de bestandsnaam en -type.

Foutmeldingen:

Foutmeldingen moeten de gebruiker wijzen op situaties waarin het programma stuk loopt, dan wel situaties die door het programma "verdacht" worden geacht. Het is belangrijk dat de gebruiker door een foutmelding direct een fout kan localiseren en herstellen. Daartoe moet in de beschrijving opgenomen worden:

- een alfabetische en/of thematische lijst met de letterlijke tekst van de foutmelding zoals het programma die geeft;
- een toelichting over de betekenis van de foutmelding;
- een concrete beschrijving van de uit te voeren actie;
- de plaats waar de foutmelding in het programma gegenereerd wordt.

## 12.6 Programma-evaluatie

De programma-evaluatie verschaft in aanvulling op de theoretische beschrijving, aan de gebruiker informatie over de bruikbaarheid van de programmatuur. Aan de theoretische beschrijving kan getoetst worden of de juiste processen in de programmatuur vervat zijn, aan de hand van de programma-evaluatie kan getoetst worden of de praktische uitwerking in de programmatuur afdoende is. De evaluatie doet wetenschappelijk verslag van een (aantal) test(s) die met het model uitgevoerd is/zijn.

In de programma-evaluatie zijn verwerkt:

- een beschrijving van de toepassingsmogelijkheden aan de hand van praktisch uitgewerkte voorbeeldstudies;
- gevoeligheidsanalyses;
- discussie over de resultaten, met daarbij aangegeven sterke en zwakke punten;
- praktische tips bij het gebruik;
- een uitputtende lijst met literatuur van studies die met behulp van de programmatuur uitgevoerd zijn (steeds up to date gehouden).

## 12.7 Resumé

- Ga bij de programmahandleiding uit van een globale, theoretische, technische en gebruikersbeschrijving, en voeg een verslag van de modelevaluatie bij.
- De globale programmabeschrijving moet de potentiële gebruiker direct een idee geven over de bruikbaarheid van de programmatuur.
- De theoretische beschrijving beschrijft het rekenproces in wiskundige termen.
- De technische beschrijving bevat een beschrijving van de programmastructuur, een vocabulaire van variabelenamen, de hardware- en software-omgeving en een volledige programmacode.
- De gebruikersbeschrijving bevat de invoerbeschrijving, de uitvoerbeschrijving en een lijst met foutmeldingen en beschrijving ervan.
- De programma-evaluatie doet wetenschappelijk verslag over de concrete prestaties van de programmatuur aan de hand van case-studies.

# 13 CONCLUSIES EN AANBEVELINGEN

## 13.1 Inleiding

De werkgroep "Richtijnen Computerprogrammatuur in de Hydrologie" heeft zich gebogen over het probleem van de te kort schietende kwaliteit van computerprogrammatuur die binnen de hydrologie gebruikt wordt. Tijdens de discussies binnen de werkgroep is de problematiek steeds duidelijker geworden en hebben mogelijke oplossingen meer vorm gekregen. De werkgroep heeft zich uiteindelijk beperkt tot het uitwerken van één oplossing van het probleem, namelijk het opstellen van richtlijnen voor programmatuur. In paragraaf 13.2 zal ingegaan worden op de conclusies van de werkgroep met betrekking tot de aard en de oorzaken van het probleem. Paragraaf 13.3 zal nader ingaan op de conclusies van de werkgroep naar aanleiding van de opgestelde richtlijnen. In de laatste paragraaf zal ingegaan worden op enige aanbevelingen.

## 13.2 Het probleem van de kwaliteit

De werkgroep heeft tijdens haar werk geconstateerd dat er daadwerkelijk sprake is van een kwaliteitsprobleem bij de ontwikkeling en het gebruik van programmatuur binnen de hydrologie. Het probleem van de slechte kwaliteit uit zich onder andere in:

1 de betrouwbaarheid;
2 de onderhoudbaarheid;
3 de overdraagbaarheid;
4 de mogelijkheden tot uitbreiding;
5 de mogelijkheden tot koppeling.

In eerste instantie lijken deze problemen slechts een grote rol te spelen bij de gebruikers, die actief programmerend aan de slag gaan met bestaande programmatuur. Bij een nadere beschouwing blijkt echter dat ook de ontwikkelaars en de eind-gebruikers wel degelijk nadelen van deze moeilijkheden ondervinden. De programmatuur wordt zo complex en de eisen zo hoog dat zelfs de ontwikkelaars van de programmatuur moeilijkheden hebben met het onderhoud en het garanderen van de kwaliteit. De eind-gebruikers worden uiteraard geconfronteerd met problemen bij de betrouwbaarheid, maar ook met lange wachttijden bij

wijziging en uitbreiding van de programmatuur. Kortom: de efficiëntie is in het geding.

De uiteindelijke oorzaak van de problemen ligt volgens de werkgroep in het feit dat er bij de ontwikkeling van programmatuur weliswaar veel aandacht is voor de hydrologische aspecten maar veel minder voor de informatica-aspecten. Enige redenen hiervoor zijn:

1 In het verleden was de programmatuur relatief eenvoudig. Veel aandacht voor informatica-aspecten was niet nodig.
2 Ontwikkelaars van hydrologische programmatuur zijn opgeleid als hydrologen die tijdens hun studie te weinig geschoold zijn in de informatica om complexe programma's te kunnen ontwikkelen.
3 Instanties die programmatuur ontwikkelen beschikken vaak niet over voldoende informatici en/of onderkennen het belang van specifieke informaticakennis niet.
4 Het verband tussen de kwaliteit van programmatuur en de kwaliteit van het onderzoek wordt in onvoldoende mate onderkend, noch door onderzoekers noch door het management.

Een van de mogelijkheden om het probleem te hanteren is het opstellen van richtlijnen voor programmatuur, waartoe in deze publikatie een poging is gedaan. Het zal echter duidelijk zijn dat er meer structurele veranderingen nodig zijn.

## 13.3 Het opstellen van richtlijnen

Het opstellen van richtlijnen voor programmatuur is een moeilijke opgave gebleken. De werkgroep kon vaak niet tot een gelijkluidend standpunt komen. Enerzijds kwam dit omdat iedereen zijn eigen methoden ontwikkeld heeft en daar moeilijk vanaf te brengen is, anderzijds bleek dat voor het oplossen van hetzelfde probleem vaak verschillende goede oplossingen mogelijk waren. Vaak bleek ook dat voor het oplossen van het ene probleem beter de ene systematiek toegepast kon worden en voor het andere probleem een andere. Het formuleren van richtlijnen in detail is derhalve ook niet nagestreefd. De richtlijnen hebben betrekking op hoofdlijnen. De richtlijnen die geformuleerd zijn, zullen ook steeds kritisch bekeken moeten worden. De snelle ontwikkeling van computers en computertalen kunnen een aantal richtlijnen overbodig maken en andere weer meer belang geven.

Het opstellen van een hydrologische variabelenlijst in analogie met de Verlarende Hydrologische Woordenlijst (de oorspronkelijke doelstelling van de werkgroep) is niet gelukt. De werkgroep is van mening dat een dergelijke lijst niet flexibel genoeg zou zijn en dat de huidige lijst niet voldoende op de (computer-)praktijk is toegespitst. Wel is bij wijze van voorbeeld een vertaling naar programmavariabelen van een aantal begrippen uit de lijst uitgevoerd. Dit is echter slechts gedaan om het toepassen van een systematische aanpak toe te lichten.

De werkgroep komt tot de conclusie dat het minder belangrijk is dat iedereen op uniforme wijze programma's ontwikkelt. Belangrijk is wel dat iedereen een duidelijk en consequent systeem toepast dat voor zijn situatie het meest geschikt is en dat op een adequate wijze gedocumenteerd wordt. Het gebruikte systeem moet uiteraard wel voldoen aan de in dit rapport aangegeven basiseisen. Het gebruik van een standaard ontwikkelmethodiek, ook in zijn meest elementaire vorm, kan een stap in de goede richting betekenen.

## 13.4 Maatregelen op langere termijn

Zoals al eerder werd opgemerkt is het opstellen van richtlijnen voor computerprogrammatuur (zoals in dit rapport is gebeurd) op langere termijn niet voldoende om te komen tot een structurele kwaliteitsverbetering. In de toekomst zullen meer informatica-aspecten geïntegreerd moeten worden bij de ontwikkeling van hydrologische programmatuur.

Teneinde een betere kwaliteit op langere termijn te garanderen beveelt de werkgroep het volgende aan:

1 Bij het onderwijs in de hydrologie (HBO/Academisch en post-HBO/post-Academisch) moet meer dan nu het geval is aandacht besteed worden aan informatica-aspecten, in de vorm van op de hydrologie toegespitste (keuze-) modules.

2 De kwaliteit van de programmatuur in de hydrologie is slecht en verdient verhoogd en gewaarborgd te worden. Tijdens studiedagen, cursussen en dergelijke dient hierop gewezen te worden en oplossingen te worden aangedragen.

3 Bij instanties die hydrologische programmatuur ontwikkelen moet gestimuleerd worden dat ontwikkelaars en gebruikers van programmatuur voortdurend terug kunnen vallen op ondersteuning van ter zake kundige informatici.

4 Op termijn zal in analogie met de sterlabs, gestreefd moeten worden naar een keurmerk voor hydrologische computerprogrammatuur. Het toekennen van een keurmerk staat dan borg voor kwaliteit. Dit uit zich dan in een systematische opbouw, betrouwbaarheid en uitwisselbaarheid van programmatuur. Zowel certificering van programmatuur als van "programmeeromgevingen" zijn een mogelijkheid.

## LITERATUUR

AMERICAN NATIONAL STANDARDS INSTITUTE INC; 1978. Programming Language FORTRAN, ANSI X 3-9-1978, ISO - 1539-1980 (E). ANSI, New York, New York.

CHO-TNO; 1978. Verslag en aanbevelingen van de ad hoc Groep Grondwatermodellen en Computerprogrammatuur TNO, Serie Rapporten en Nota's no. 2.

CHO-TNO; 1982. Rapport en aanbevelingen van de Contactgroep Grondwatermodellen, Serie Rapporten en Nota's no. 10.

CHO-TNO; 1986. Verklarende Hydrologische Woordenlijst, Serie Rapporten en Nota's no. 16.

GILB, T; 1988. Principles of software engineering management. Wokingham (England), Addison-Wesley Publishing Company.

HAMMER, D.K EN K.M. VAN HEE; 1990. Fasering en documentatie in software engineering. Informatie jaargang 32 nr. 2.

JANSEN, H.; 1984. JSP, Jackson structureel programmeren. Academic Service, Den Haag.

NATIONAL RESEARCH COUNCIL; 1990. Ground Water Models, Scientific and Regulatory Applications. National Academy Press, Washington D.C.

PBNA; 1987. Poly-Automatiserings Zakboekje, Koninklijke PBNA BV, Arnhem.

REDISH, PH.D. J.C.; 1986. Writing and designing effective software manuals. American Institutes for Research. P. 5-6

SAMWAT; 1991. Komputermodellen in het waterbeheer; het SAMWAT Modellenbestand, SAMWAT-Rapport no. 7.

VROM/DGM; 1990. Kwaliteitscriteria voor modellen om luchtverontreiniging te berekenen. Publicatiereeks lucht, Rapport no. 90.

# BIJLAGEN

A Samenstelling CHO-Werkgroep Richtlijnen Computerprogrammatuur Hydrologie

B Voorbeeld Naamgeving Programmavariabelen: de Verklarende Hydrologische Woordenlijst

C Voorbeeld Programma AQ-HL02 (RIVM)

D Voorbeeld Programma EPOT (Staring Centrum)

# A SAMENSTELLING CHO-WERKGROEP RICHTLIJNEN COMPUTERPROGRAMMATUUR HYDROLOGIE

| | | |
|---|---|---|
| G. van Barneveld | - | RIZA |
| R.H. Boekelman/A. Koçan | - | TUD (Wetenschappelijk Onderwijs) |
| J.P. van der Eem | - | KIWA N.V. |
| K. Kovar | - | RIVM (voorzitter) |
| A.C.W. Lambrechts | - | TAUW Infra Consult B.V. |
| J.M.P.M. Peerboom | - | Staring Centrum (tot 15.4.90) daarna op persoonlijke titel |
| E.P. Querner | - | Staring Centrum (vanaf 15.4.90) |
| F. Waardenburg | - | IGG-TNO/RGD |
| W.J. Zaadnoordijk | - | IWACO B.V. |
| J.C. Hooghart | - | CHO-TNO (secretaris) |

## B VOORBEELD NAAMGEVING PROGRAMMAVARIABELEN: DE VERKLARENDE HYDROLOGISCHE WOORDENLIJST (VHW)

| Variable | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| | **ALGEMENE TERMEN** | | | | |
| WUS WATUS | water use | $L^3T^{-1}$ | | | 7 |
| WCO WATCO | water consumption | $L^3T^{-1}$ | | | 8 |
| | **ATMOSFERISCH WATER** | | | | |
| RMX RAMIX | mixing ratio | dim.loos | - | $r$ | 10 |
| SHU SPHUM | specific humidity | dim.loos | - | $q$ | 11 |
| AHU ABHUM | absolute humidity | $L^{-3}M$ | $g.m^{-3}$ | $d_v$ | 12 |
| DMA DENMA | density of moist air | $L^{-3}M$ | $kg.m^{-3}$ | $\rho_v$ | 13 |
| WVM WVMOF | mole fraction of water vapour | dim.loos | - | $x_v$ | 14 |
| PVP PVAPR | vapour pressure | $L^{-1}M\ T^{-2}$ | mbar of hPa | $e$ | 15a |
| PSV PSAVR | saturation vapour pressure | $L^{-1}M\ T^{-2}$ | mbar of hPa | $e_s$ | 15b |
| RHU RLHUM | relative humidity | dim.loos | % of - | - | 16 |
| SAD SATDF | saturation deficit | $L^{-1}M\ T^{-2}$ | mbar of hPa | $\Delta e$ | 17 |
| TDP TDEWP | dew-point temperature, dew-point | $\Theta$ | °C of K | $T_d$ | 18 |
| TWB TWETB | wet-bulb temperature | $\Theta$ | °C of K | $T_w$ | 19 |
| TVI TVIRT | virtual temperature | $\Theta$ | °C of K | $T_v$ | 20 |
| GAM GAMMA | psychrometric constant | $L^{-1}MT^{-2}\Theta^{-1}$ | $mbar.K^{-1}$ of $hPA.K^{-1}$ | $\gamma$ | 21 |
| SVP SLVPR | slope of the saturation water vapour pressure curve | $L^{-1}MT^{-2}\Theta^{-1}$ | $mbar.K^{-1}$ of $hPa.K^{-1}$ | $s$ | 22 |
| SHV SPHVP | specific latent heat of vaporization | $L^2T^{-2}$ | $J.kg^{-1}$ | $\lambda$ | 32 |
| FLH FLATH | latent heat flux density | $M\ T^{-3}$ | $W.m^{-2}$ | $\lambda E$ | 33 |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| FSH FSENH | sensible heat flux density | $M\ T^{-3}$ | $W.m^{-2}$ | $H$ | 34 |
| FGH FGROH | soil heat flux density (ground) | $M\ T^{-3}$ | $W.m^{-2}$ | $G$ | 35 |
| FNR FNETR | net radiation flux density | $M\ T^{-3}$ | $W.m^{-2}$ | $Q*$ | 36 |
| FGR FGLOR | global solar radiation flux density, global radiation, shortwave radiation | $M\ T^{-3}$ | $W.m^{-2}$ | $K^{\downarrow}$ | 37 |
| FSR FNETS | net solar radiation flux density | $M\ T^{-3}$ | $W.m^{-2}$ | $K*$ | 38 |
| FTR FTERR | net terrestrial flux density | $M\ T^{-3}$ | $W.m^{-2}$ | $L*$ | 39 |
| BOW BOWEN | Bowen ratio | dim.loos | - | $\beta$ | 41 |
| ALB ALBDO | albedo, reflectivity | dim.loos | - | $r$ | 42 |
| PRM PRMSS | precipitation mass | M | $m^3$ | - | 50 |
| PRD PRDPH | precipitation depth | $L^{-2}M$ | mm | - | 51 |
| PRI PRINI | instantaeneous precipitation intensity | $L^{-2}M\ T^{-1}$ | $mm.min^{-1}$ | - | 52a |
| PRI PRINY | precipitation intensity | $L^{-2}M\ T^{-1}$ | $mm.min^{-1}$ | - | 52b |
| PRE PRECI | (gross) precipitation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $P$ | 53 |
| PIN PRINT | interception | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_i$ | 54 |
| PRN PRNET | net precipitation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $P_n$ | 55 |
| DPR (mx) DPREC | maximum precipitation deficit | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 56a |
| EPR (mx) EPREC | maximum precipitation excess | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 56b |
| DPR DPREC | precipitation deficit | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 57a |
| EPR EPREC | precipitation excess | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 57b |
| QRE QRECH | natural groundwater recharge | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 58 |
| PRF PREFF | effective precipitation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 59 |
| AWD AWADE | additional water demand of crops | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 60 |
| CRT CRDRT | critical rainfall duration | T | min | - | 62 |
| CRD CRDPH | critical rainfall depth | $L^{-2}M$ | mm | - | 63 |

| Variabele | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| SND SUMND | n-day sum | $L^{-2}M$ | mm | - | 65a |
| SNM SUMNM | n-minute sum | $L^{-2}M$ | mm | - | 65b |
| EOW EOWAT | open water evaporation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_o$ | 69 |
| EPA EVPAN | pan evaporation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_{pan}$ | 70 |
| EIN EINTC | evaporation of intercepted water | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_i$ | 71a |
| ESO ESOIL | soil evaporation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_s$ | 71b |
| EVP EVAPO | evaporation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | - | 71c |
| ETR ETRSP | transpiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_t$ | 72 |
| EVT EVTRA | (actual) evapotranspiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E$ | 73 |
| ESO (p) | potential soil evaporation | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_{sp}$ | 74a |
| ETR (p) | potential transpiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_{tp}$ | 74b |
| EVT (p) | potential evapotranspiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_p$ | 74c |
| EVT (r) | relative evapotranspiration | dim.loos | - | - | 75 |
| EWC EWETC | wet crop evapotranspiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_w$ | 76 |
| EGR EGREF | reference grass evapo-transpiration | $L^{-2}M\ T^{-1}$ | $mm.d^{-1}$ | $E_{grass}$ | 77 |
| RAE RAEVP | aerodynamic resistance to water vapour | $L^{-1}T$ | $s.m^{-1}$ | $r_a$ | 78 |
| RCA RCANY | canopy resistance | $L^{-1}T$ | $s.m^{-1}$ | $r_c$ | 79 |
| | **WATER IN DE ONVERZADIGDE ZONE** | | | | |
| DMC DMOCY | differential water (moisture) capacity | $L\ M^{-1}T^2$ | $Pa^{-1}$ | $C_\theta$, $C_w$ | 103 |
| SEC SEQCT | specific equilibrium soil water content | L<br>L | m<br>m | $W_e^+$<br>$W_e^-$ | 105a<br>105b |
| SMD SMODF | specific soil water deficit, specific moisture deficit | L | m | $S_d$, $W_d$ | 106 |
| STO STORG | storage | $L^3$ | $m^3$ | $V$ | (n.v.t.) |
| UST USTOR | uns. zone storage | | | | |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| SDF SATDF | storage capacity, saturation deficit | $L^3$ SSC SSTCY | $m^3$ specific | - storage | 108 capacityLm-109 |
| THF THFLD | field capacity | dim.loos | - | $\Theta_{FC}$ | 110 |
| THW THWLT | wilting point | dim.loos | - | $\Theta_{WP}$ | 111 |
| SWC SWACY | soil water retention, water holding capacity | L | m | $S_{FC}$ | 112 |
| SWA SWAVL | available soil water, available soil moisture | L | m | - | 113 |
| ..... | (vrijkomend poriëngehalte) | dim.loos | - | - | 114 |
| HCY HCNTY | (hydraulic) conductivity | $L\ T^{-1}$ | $m.d^{-1}$ | $K,k$ | 130b |
| PMY PERMY | intrinsic permeability | $L^2$ | $m^2$ | $k$,k | 131 |
| SWD SWDIF | soil water (moisture) diffusivity | $L^2T^{-1}$ | $m^2.d^{-1}$ | $D_\theta$ | 132 |
| POM POMST | moisture potential, soil water potential | $L^2T^{-2}$ | $J.kg^{-1}$ | $\psi,\psi_m,\psi..$ | 140a |
| | | $L^{-1}M\ T^{-2}$ | Pa | $p,p_m,p..$ | 140b |
| | | L | m | $h,h_p,h..$ | 140c |
| POT POTEN | tensiometer pressure potential | $L^2T^{-2}$ | $J.kg^{-1}$ | $\psi_p$ | 141 |
| POG POGRV | gravitational potential | $L^2T^{-2}$ | $J.kg^{-1}$ | $\psi_g$ | 142 |
| POH POHYD | hydraulic potential | $L^2T^{-2}$ | $J.kg^{-1}$ | $\psi_h$ | 143 |
| PWA PWABS | absolute water pressure | $L^{-1}M\ T^{-2}$ | Pa | $p_{abs}$ | 144 |
| PWR PWREL | water pressure (relative) | $L^{-1}M\ T^{-2}$ | Pa | $p$ | 145 |
| PSU PSUCT | suction | $L^{-1}M\ T^{-2}$ | Pa | $p$ | 146 |
| PTE PTENS | tensiometer pressure | $L^{-1}M\ T^{-2}$ | Pa | $p$ | 147 |
| PMA PMATR | matric(al) pressure | $L^{-1}M\ T^{-2}$ | Pa | $p_m$ | 148 |
| PAE PAIRE | air entry value | $L^{-1}M\ T^{-2}$ | Pa | $p_{ae}$ | 149 |
| HPR HPRES | pressure head | L | m | $h_p$ | 150 |
| HEL HELEV | elevation head | L | m | z | 151 |

| Variabele | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| HHY HHYDR | hydraulic head | L | m | $h$ | 152 |
| CAH CAPHT | height of capillary fringe, capillary height | L | m | $h_c$ | 153 |
| PEF - | $pF$ | dim.loos | - | $pF$ | 156 |
| QSP QSPEC | specific discharge<br>Darcy flux | $L\ T^{-1}$ | $m.d^{-1}$ | $v,q$ | 170 |
| VEF VEFCT | effective velocity | $L\ T^{-1}$ | $m.d^{-1}$ | $v_e$ | 171 |
| QVO QVOLM | volume flux | $L^3T^{-1}$ | $m^3.d^{-1}$ | $Q, q_v$ | 172 |
| QVD QVLMD | volume fluxdensity | $L\ T^{-1}$ | $m.d^{-1}$ | $v$ | 173 |
| QIF QINFT | infiltration rate | $L\ T^{-1}$ | $m.d^{-1}$ | $f_i$ | 175 |
| QIC QINCY | infiltration capacity | $L\ T^{-1}$ | $m.d^{-1}$ | $f_p$ | 176 |
| CIF CFINF | infiltration coefficient | dim.loos | - | - | 178 |
| QPE QPERC | percolation | $L\ T^{-1}$ | $m.d^{-1}$ | $v_z$* | 179b |
| QCR QCAPR | upward capillary migration, capillary rise | $L\ T^{-1}$ | $m.d^{-1}$ | $v_z$ | 180b |
| SBG SBGRD | subsidence (of groundlevel) | L | cm | - | 191 |
| SBP SBPZH | subsidence by lowering of the piezometric head | L | cm | - | 193 |
| SBS SBSKG | shrinkage<br>subsidence by shrinkage | dim.loos<br>L | -<br>cm | -<br>- | 194a<br>194b |
| SWL SWELL | swelling | dim.loos<br>L | -<br>cm | -<br>- | 195a<br>195b |
| CON CONCT | concentration | | | | (n.v.t.) |
| | **WATER IN DE VERZADIGDE ZONE** | | | | |
| EPO EFPOR | effective porosity | dim.loos | - | $n_e$ | 210 |
| STO STORG | storage | $L^3$ | $m^3$ | $V$ | 211 |
| SSY SPSTY | specific storativity | $L^{-1}$ | $m^{-1}$ | $S_s$ | 214 |
| STC STOCF | storage coeffient | dim.loos | - | $S$, $\mu$ | 215 |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| SYD SPYLD | specific yield | dim.loos | - | $S_y$ | 216 |
| HCY HCNTY | (hydraulic) conductivity | L $T^{-1}$ | $m.d^{-1}$ | $K$, $k$ | 224b |
| PMY PERMY | (intrinsic) permeability | $L^2$ | $m^2$ | $k$, $K$ | 225 |
| TRY TRMTY | transmissivity (coefficient of transmissibility) | $L^2T^{-1}$ | $m^2.d^{-1}$ | T,kD | 226 |
| CLK CFLKG | leakance, leakage coefficient | $T^{-1}$ | $d^{-1}$ | $\gamma$ | 227 |
| RAQ RAQRD | hydraulic resistance of aquitard | T | d | $c$ | 229 |
| RRV RRIVR | river resistance | T | d | | (n.v.t.) |
| RDR RDRNG | drainage resistance | T | d | $v$ | 230 |
| RRF RRADF | radial flow resistance | $L^{-1}T$ | $m^{-1}.d$ | $\Omega$ | 231 |
| REN RENTR | entrance flow resistance | $L^{-1}T$ | | | (n.v.t.) |
| QSP QSPEC | apparent velocity, specific discharge | L $T^{-1}$ | $m.d^{-1}$ | $v$, $q$ | 242 |
| VEF VEFCT | effective velocity | L $T^{-1}$ | $m.d^{-1}$ | $v_e$ | 243 |
| QVO QVOLM | volume flux | $L^3T^{-1}$ | $m^3.d^{-1}$ | $Q$, $q_v$ | 244 |
| QVD QVLMD | volume flux density | L $T^{-1}$ | $m.d^{-1}$ | $v$ | 245 |
| QGW QGRWR | groundwater discharge | $L^3T^{-1}$ | $m^3.d^{-1}$ | $Q_g$ | 246 |
| QGS QSGRW | specific groundwater discharge | L $T^{-1}$ | $m.d^{-1}$ | $U$ | 247 |
| QIN QINFT | infiltration rate | L $T^{-1}$ | $m.d^{-1}$ | $f_i$ | 252 |
| QSE QSEPD | (kwelintensiteit) flux density | L $T^{-1}$ | $m.d^{-1}$ | $U_k$ | 259 |
| QAQ QAQRD | aquitard flux density | L $T^{-1}$ | $m.d^{-1}$ | | (n.v.t.) |
| HHY HHYDR | hydraulic head, piezometric head, piezometric level | L | m | $h$ | 270 |
| HPR HPRES | pressure head | L | m | $h_p$ | 271 |
| HEL HELEW | elevation head | L | m | $z$ | 272 |
| HFW HFRWA | fresh-water head | L | m | $h_f$ | 273 |
| HSW HSAWA | salt-water head | L | m | $h_s$ | 274 |

| Variabele | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| HPH HPHRE | phreatic level, groundwater level | L | m | $h$ | 275 |
| DFH DPHRE | depth of the groundwater level (phreatic level) below ground-surface | L | m | $h*$ | 276 |
| PGW PGRWA | groundwater pressure | $L^{-1}M\ T^{-2}$ | Pa | $p$ | 284 |
| QRV QRIVR | river-groundwater discharge | $L^3T^{-1}$ | | | (n.v.t.) |
| QDR QDRNG | drainage discharge rate | $L\ T^{-1}$ | | | (n.v.t.) |
| QWL QWELL | well discharge | $L^3T^{-1}$ | | | (n.v.t.) |
| CON CONCT | concentration | | | | (n.v.t.) |
| CWL CWELL | concentration in well infiltration water | | | | (n.v.t.) |
| CRV CRIVR | concentration in water infiltrating from river into groundwater | | | | (n.v.t.) |
| CDR CDRNG | concentration in water infiltrating by diffuse drainage into groundwater | | | | (n.v.t.) |
| CRE CRECH | concentration in water recharging aquifer | | | | (n.v.t.) |
| | **OPPERVLAKTEWATER** | | | | |
| STO STORG | storage | $L^3$ | $m^3$ | $V$ | 304 |
| STY STCPY | storage capacity | $L^3$ | $m^3$ | $V$ | 305 |
| TID TIDIS | discharge period | T | uur | - | 375 |
| HHW HHIWA | high water | L | cm | $HW$ | 404a |
| HLW HLOWA | low water | L | cm | $LW$ | 408a |
| HTR HTIDR | tidal range | L | cm | - | 409 |
| HMS HMSEA | mean sea level | L | m | $\bar{Z}$ | 412 |
| HTU HTDUP | rise of the tide (up) | L | cm | - | 413b |
| HTD HTDDN | fall of the tide (down) | L | cm | - | 414b |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| VTI VTIDL | tidal volume, tidal prism | $L^3$ | $m^3$ | - | 424 |
| VEB VLEBB | ebb volume | $L^3$ | $m^3$ | - | 425 |
| VFL VLFLD | flood volume | $L^3$ | $m^3$ | - | 426 |
| FTC FETCH | fetch | L | cm | $F$ | 433 |
| HWD HWSDN | (wind) set down | L | cm | - | 434b |
| HWU HWSUP | (wind) set up | L | cm | - | 435b |
| HSU HSETU | set up | L | cm | - | 436 |
| HRU HWARU | wave run up | L | cm | $z$ | 440 |
| WAR WAREA | wetted area | $L^2$ | $m^2$ | $A$ | 507 |
| WPE WPERI | wetted perimeter | L | m | $P$ | 508 |
| CCT CCNTR | coefficient of contraction, contraction coefficient | dim.loos | - | $\mu$ | 509 |
| RDH RADHY | hydraulic radius | L | m | $R$ | 510 |
| DPH DEPHY | hydraulic depth | L | m | $D$ | 511 |
| WDH WADPH | water depth | L | cm | $y$ | 512 |
| CDH CPDPH | critical depth | L | cm | $y_c$ | 513 |
| EDH EQDPH | equilibrium depth | L | cm | $y_n$ | 514 |
| HWL HWALV | water level, stage | L | cm | $h$ | 517 |
| HEN HENRG | energy head | L | cm | $H$ | 518 |
| GHY GHYDR | hydraulic gradient | dim.loos | - | $s$ | 520 |
| GEN GENRG | energy gradient | dim.loos | - | $S$ | 521 |
| GCR GCROS | cross gradient | dim.loos | - | $s$ | 522 |
| CRG CRGHS | roughness coefficient | var. | var. | diverse | 524 |
| QVO QVOLM | flow rate | $L^3T^{-1}$ | $m^3.s^{-1}$ of $liter.s^{-1}$ | $Q$ | 527 |
| QVO QVOLM | discharge | $L^3T^{-1}$ | $m^3.s^{-1}$ | $Q$ | 528 |
| QBS QBASE | base flow | $L^3T^{-1}$ | $m^3.s^{-1}$ | $Q_o$ | 529 |

| Variabele | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| QDR QDIRC | direct runoff | $L^3T^{-1}$ | $m^3.s^{-1}$ | - | 531 |
| QPK QPEAK | peak discharge | $L^3T^{-1}$ | $m^3.s^{-1}$ | $\hat{Q}$ | 532 |
| QDM QDOMI | dominant discharge | $L^3T^{-1}$ | $m^3.s^{-1}$ | - | 533 |
| QDS QDSGN | design discharge | $L^3T^{-1}$ | $m^3.s^{-1}$ | - | 534 |
| QSP QSPEC | specific discharge | $L\ T^{-1}$ | liter. $s^{-1}.ha^{-1}$ | - | 535 |
| CDS CDISC | discharge coefficient | dim.loos | - | - | 536 |
| FXD FEXCD | frequency of exceedance of discharge | dim.loos | - | *P* | 537 |
| QCY QCAPY | discharge capacity | $L^3T^{-1}$ | $m^3.s^{-1}$ | - | 538 |
| TIT TITRA | travel time | T | d | - | 549 |
| WRE WIDRE | regulation width | L | m | - | 553 |
| WSR WIDSR | stream width | L | m | *b* | 554 |
| WSO WIDST | storage width | L | m | *B* | 556 |
| HHI HHIGH | high water level | L | cm | *h* | 557 |
| HNO HNORM | normal water level | L | cm | *h* | 558 |
| HFC HFCRS | (flood) crest stage | L | cm | - | 560 |
| WES WEQSW | water equivalent of snow | L | mm | - | 584 |
| EFI EFFIR | irrigation efficiency | dim.loos | - | *E* | 600 |
| HDB HDRBS | drainage base | L | cm | *h* | 611 |
| DGW DGRWT | depth to the groundwater table | L | cm | - | 612 |
| DFR DFREB | freeboard | L | cm | - | 613 |
| DDF DDIFF | differential head | L | cm | - | 614 |
| HPO HPOLD | polder water level | L | cm | *P.P.* | 620 |
| HTR HTARG | target level | L | cm | *S.P.* | 621 |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| | **DIVERSEN** | | | | |
| DIS DISTC | distance | L | | | (n.v.t.) |
| THS THICS | thickness | L | | | (n.v.t.) |
| XCO XCOOR | X-coordinate | L | | | (n.v.t.) |
| YCO YCOOR | Y-coordinate | L | | | (n.v.t.) |
| ZCO ZCOOR | Z-coordinate | L | | | (n.v.t.) |
| TIM TIME | time | T | | | (n.v.t.) |
| NOD - | number of nodes | | | | (n.v.t.) |
| XNO XNODE | X-coordinate of nodes | L | | | (n.v.t.) |
| YNO YNODE | Y-coordinate of nodes | L | | | (n.v.t.) |
| NEL - | number of elements | - | | | (n.v.t.) |
| NBP - | number of boundary points | - | | | (n.v.t.) |
| INO - | sequential number of nodes (array index) | - | | | (n.v.t.) |
| IEL - | sequential number of elements (array index) | - | | | (n.v.t.) |
| IBP - | sequential number of boundary nodes | - | | | (n.v.t.) |
| NWL - | number of wells | - | | | (n.v.t.) |
| IWL - | sequence number of wells (array index) | - | | | (n.v.t.) |
| XWL XWELL | X-coordinate of well | L | | | (n.v.t.) |
| YWL YWELL | Y-coordinate of well | L | | | (n.v.t.) |
| QWL QWELL | well discharge | $L^3$ | | $T^{-1}$ | |
| TWL TWELL | time instance for transient well rate | - | | $T$ | (n.v.t.) |
| KWL - | number of aquifer containing well | - | | | (n.v.t.) |
| NRV - | number of rivers | - | | | (n.v.t.) |

| Variabele | Description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| IRV - | sequence number of rivers (array index) | - | | | (n.v.t.) |
| NPR - | number of points on river | - | | | (n.v.t.) |
| XRV XRNR | X-coordinate of river points | L | | | (n.v.t.) |
| YRV YRIVR | Y-coordinate of river points | L | | | (n.v.t.) |
| HRV HRIVR | river water level | L | | | (n.v.t.) |
| NAQF | number of aquifers | - | | | (n.v.t.) |
| NAQT | number of aquitards | - | | | (n.v.t.) |
| NLAY | number of layers | - | | | (n.v.t.) |
| IAQF | sequence number of aquifers | - | | | (n.v.t.) |
| IAQT | sequence number of aquitards | - | | | (n.v.t.) |
| ILAY | sequence number of layers | - | | | (n.v.t.) |
| NSUB | number of node subregions | - | | | (n.v.t.) |
| ISUB | sequence number of subregions | - | | | (n.v.t.) |
| NPOL | number of polygons | - | | | (n.v.t.) |
| IPOL | sequence number of polygons | - | | | (n.v.t.) |
| NPPO | number of points constituting polygon | - | | | (n.v.t.) |
| XPO XPOLY | X-coordinate of polygon points | L | | | (n.v.t.) |
| YPO YPOLY | Y-coordinate of polygon points | L | | | (n.v.t.) |
| VPO VPOLY | value in a polygon | - | | | (n.v.t.) |
| NRP | number of random points | - | | | (n.v.t.) |
| IRP | sequence number of random points | - | | | (n.v.t.) |
| XRP XRPNT | X-coordinate of random points | L | | | (n.v.t.) |
| YRP YRPNT | Y-coordinate of random points | L | | | (n.v.t.) |
| VRP VPPNT | value at random point | - | | | (n.v.t.) |
| NPL | number of path lines | - | | | (n.v.t.) |

| Variabele | description | Dimension | Units | Symbol | No. in V.H.W. |
|---|---|---|---|---|---|
| IPL | sequence number of path lines | - | | | (n.v.t.) |
| NPPL | number of points on path lines | - | | | (n.v.t.) |
| XPL | X-coordinate of path lines | L | | | (n.v.t.) |
| YPL | Y-coordinate of path lines | L | | | (n.v.t.) |
| ZPL | Z-coordinate of path lines | L | | | (n.v.t.) |

# C VOORBEELD PROGRAMMA AQ-HL02 (RIVM)

## Inleiding

Deze bijlage is in het rapport opgenomen ter illustratie van een aantal aspecten met betrekking tot in het bijzonder:

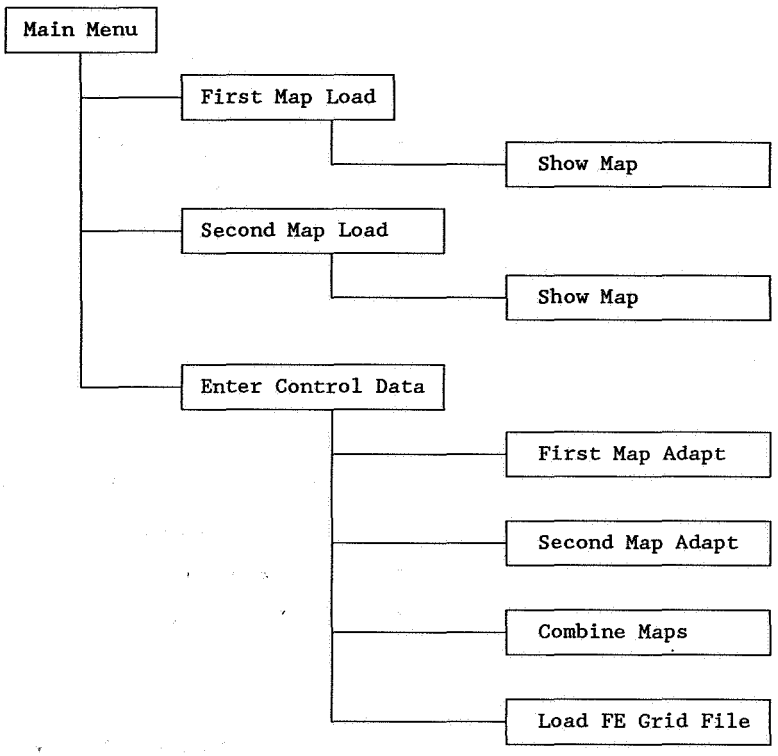
a de structurering van programma's (Hoofdstuk 3);
b de naamgeving van programma-onderdelen (Hoofdstuk 4);
c de lay-out van de broncode (Hoofdstuk 7);
d de foutencontrole en foutmeldingen (Hoofdstuk 9);
e de interne documentatie in computercode (Hoofdstuk 11).

Het programma AQ-HL02 is een onderdeel van het pakket AQ-HLP. AQ-HLP maakt deel uit van het omvangrijke systeem van de zogenaamde AQ-computerprogramma's ten behoeve van de simulatie van de vraagstukken met betrekking tot grondwater, zowel wat betreft de kwantiteits- als de kwaliteitsaspecten. De AQ- (voorheen AquiSoft) programmatuur is bij het RIVM beschikbaar.

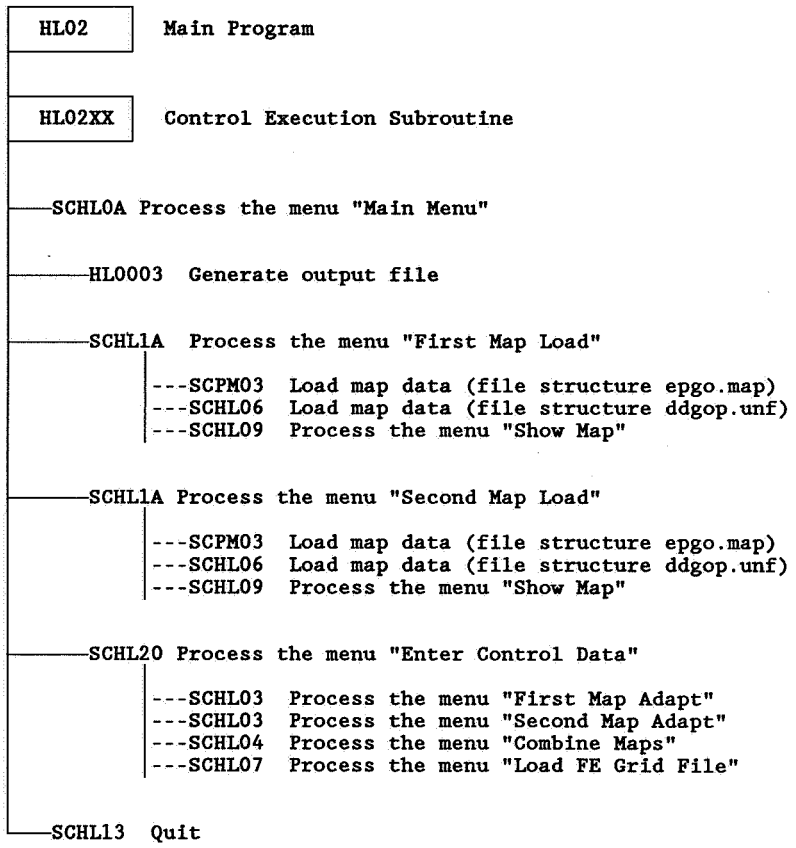
Het programma AQ-HL02 is ontwikkeld met behulp van de Microsoft FORTRAN F77. Het werkt volledig menugestuurd (gebruikmakend van de extended ASCII set) en draait onder het DOS en OS2 operatingsystem. Om het programma te kunnen draaien moet in de opstart-file "config.sys" het commando "device=ansi.sys" worden opgenomen.

Het programma leest de gegevens van twee (verschillende) files, combineert deze gegevens tot een nieuwe datagroep, en schrijft het resultaat vervolgens naar een output file. Alle files zijn ongeformatteerd (dus niet ASCII), dat wil zeggen niet leesbaar met een tekst editor.

Schema van Menu's



Schema aanroep belangrijke subroutines



```
************************************************************************
*                                                                      *
*                        >>  PROGRAM HL02  <<                          *
*                                                                      *
*----------------------------------------------------------------------*
*  purpose : Main module for program HL02                              *
*----------------------------------------------------------------------*
*  parameters : none                                                   *
*----------------------------------------------------------------------*
*  subprograms     : hl02xx                                            *
*----------------------------------------------------------------------*
*  error messages : none                                               *
*----------------------------------------------------------------------*
*  implementation : none                                               *
*----------------------------------------------------------------------*
*  Copyright RIVM, The Netherlands          Last update : 09-DEC-1989  *
************************************************************************
      program hl02
      implicit double precision (a-h,o-z)
*-----------------------------------------------------------------------
      data iinl/1/, iin2/2/, ife/3/, iout/4/
*-----------------------------------------------------------------------
*  Call control execution subroutine
*
      call hl02xx (iinl, iin2, ife, iout)
      stop  'End Program HL02'
      end
```

```
*************************************************************************
*                                                                       *
*                    >>  SUBROUTINE HL02XX  <<                          *
*                                                                       *
*-----------------------------------------------------------------------*
*  purpose : Control execution of the program HL02.                     *
*            Generate a file, the format identical to that of a file    *
*            from the program DDGO, containg the combination of two     *
*            map files which were previously made either and or:        *
*            1) as output from APGO, EPGO or ECGO                       *
*            2) as output from DDGO, HL02 or HL03                       *
*                                                                       *
*            Code number ICMFL defines the type of map file:            *
*            -----------------------------------------------            *
*            ICMFL=1: file is generated by APGO, EPGO or ECGO           *
*            ICMFL=2: file is generated by DDGO, HL02 or HL03           *
*                                                                       *
*            The following options are available in 'Main Menu':        *
*            IOPT  Option Text          IME00                           *
*            ----  ------------------   -------                         *
*              1   Output File          not relevant   (FLOUT)         *
*     menu >>  2   First Map Load       -41            (FLFIL1)         *
*     menu >>  3   Second Map Load      -42            (FLFIL2)         *
*     menu >>  4   Enter Control Data    -3            (FLCNT)          *
*              5   Generate Output      -2                              *
*              6   Next Output          -1            (FLNEXT)         *
*              7   Quit                  1                              *
*                                                                       *
*            The following options are available in the menu            *
*            'First Map Load':                                          *
*            IOPT  Option Text          IME1A                           *
*            ----  ------------------   -----                           *
*              1   Input File           not relevant (FLINP1, FNINP1)  *
*              2   Parameter Number     not relevant (IPAR1)           *
*              3   Load Map             not relevant (FLFIL1)          *
*     menu >>  4   Show Map             not relevant (FLFIL1)          *
*              5   Return                0                             *
*              6   Quit                  1                             *
*                                                                       *
*            The following options are available in the menu            *
*            'Second Map Load':                                         *
*            IOPT  Option Text          IME1B                           *
*            ----  ------------------   -----                           *
*              1   Input File           not relevant (FLINP2, FNINP2)  *
*              2   Parameter Number     not relevant (IPAR2)           *
*              3   Load Map             not relevant (FLFIL2)          *
*     menu >>  4   Show Map             not relevant (FLFIL2)          *
*              5   Return                0                             *
*              6   Quit                  1                             *
*                                                                       *
*            The following options are available in the menu            *
*            'Enter Control Data':                                      *
*            IOPT  Option Text          IME20                           *
*            ----  ------------------   -------                         *
*              1   Title Output File    not relevant (TITXT)           *
*              2   Name of Parameter     not relevant (LABPAR)          *
*     menu >>  3   First Map Adapt       not relevant (PAR1, ...)       *
*     menu >>  4   Second Map Adapt      not relevant (PAR2, ...)       *
*     menu >>  5   Combine Maps          not relevant (PARC, FLCMB)     *
*     menu >>  6   Load FE Grid File     not relevant (FLLDFE, FLFEM)   *
*              7   Parent Menu           0                              *
*              8   Quit                  1                              *
*-----------------------------------------------------------------------*
*  parameters :                                                         *
*                                                                       *
*  IIN1   = logical unit number of input file 1                 (i)    *
```

```
*           (integer variable)                                          *
*  IIN2   = logical unit number of input file 2                (i)     *
*           (integer variable)                                          *
*  IFE    = logical unit number of the FE input data file      (i)     *
*           (integer variable)                                          *
*  IOUT   = logical unit number of output file FNOUT           (i)     *
*           (integer variable)                                          *
*--------------------------------------------------------------------*
*  subprograms    : hl0003, scg009, schl01, schl0a, schl13, schl1a,   *
*                   schl20, scmcu2, vnhl02                            *
*--------------------------------------------------------------------*
*  error messages : 3                                                 *
*--------------------------------------------------------------------*
*  implementation : none                                              *
*--------------------------------------------------------------------*
*  Copyright RIVM, The Netherlands          Last update : 22-JAN-1991  *
**********************************************************************
      subroutine hl02xx (iin1, iin2, ife, iout)
      implicit double precision (a-h,o-z)
*--------------------------------------------------------------------
*  Arrays required if ICMFL=1 (input file made by APGO, EPGO or ECGO)
*
      parameter (nparmx=99)
      common/a1/ itypa1(nparmx), ilopa1(nparmx), isypa1(nparmx),
     &           iunpa1(nparmx), zl1(nparmx)
      common/a2/ itypa2(nparmx), ilopa2(nparmx), isypa2(nparmx),
     &           iunpa2(nparmx), zl2(nparmx)
      common/b1/ iuntil(nparmx), tim1(nparmx), id1(nparmx),
     &           im1(nparmx), iy1(nparmx), ihr1(nparmx), imn1(nparmx),
     &           isc1(nparmx)
      common/b2/ iunti2(nparmx), tim2(nparmx), id2(nparmx),
     &           im2(nparmx), iy2(nparmx), ihr2(nparmx), imn2(nparmx),
     &           isc2(nparmx)
*--------------------------------------------------------------------
*  LABPA1 and LABPA2 are required if ICMFL=2 (input file made by DDGO,
*  HL02 or HL03)
*
      character labpa1(nparmx)*30, labpa2(nparmx)*30
*--------------------------------------------------------------------
*  LABPAR is required for the output file
*
      character labpar*30
*--------------------------------------------------------------------
*  Set the FE grid variable NODMX (NELMX and NBPMX and needed here)
*
$include:'fegrid.dim'
*--------------------------------------------------------------------
*  The following arrays are required to load the two parameters
*
      common /c/ par1(nodmx), par2(nodmx)
*--------------------------------------------------------------------
*  Array PARC to store the combination of the two maps
*
      common /d/ parc(nodmx)
*--------------------------------------------------------------------
*  Arrays PAR01 and PAR02 are used to store original map values, to
*  be used to reset to original setting if required.
*
      common /e/ par01(nodmx), par02(nodmx)
*--------------------------------------------------------------------
*  IDCOR is loaded from the map parameter input file IIN (FNINP),
*  but is not used in this program
*
      dimension idcor(6)
*--------------------------------------------------------------------
*  Variables relating to the FE grid data file IFE
```

```
*
      character fnine*30
      logical   flldfe, flfem, fline
*-----------------------------------------------------------------------
      character pgn*8, vn*3, fnout*30, fninp1*30, fninp2*30
      character esc, clrs*5, clrl*4, crvon*5, crvof*5, cblon*5, cv
      character titxt1*40, titxt2*40, titxt*40, titmen*18, txtbl*30
      logical   flgenx, flinp1, flinp2, flout, flfil1, flfil2, flcnt
      logical   flnext, flzl1, flzl2, fltim1, fltim2, fldat1, fldat2
      logical   flcmb, flchg1, flchg2
*-----------------------------------------------------------------------
*  Set the text associated with the error messages
*
      character tx10*45, tx11*45, tx12*37, tx13*37, help*49
      character tx21*45, tx22*40, tx23*40
      character tx31*45, tx32*43, tx33*43
      data tx10/'----------------- E R R O R -----------------'/
      data tx11/'----- NON IDENTICAL NUMBER OF GRID NODES ----'/
      data tx12/'Number of Nodes on First Map File  = '/
      data tx13/'Number of Nodes on Second Map File = '/
      data tx21/'----- NON IDENTICAL NUMBER OF ELEMENTS ------'/
      data tx22/'Number of Elements on First Map File  = '/
      data tx23/'Number of Elements on Second Map File = '/
      data tx31/'-- NON IDENTICAL NUMBER OF BOUNDARY NODES ---'/
      data tx32/'Number Boundary Nodes on First Map File  = '/
      data tx33/'Number Boundary Nodes on Second Map File = '/
      data txtbl/' '/
*-----------------------------------------------------------------------
*  Set program name and program version
*
      data pgn/'AQ-HL02'/
      call vnhl02 (vn)
*-----------------------------------------------------------------------
*  Initialize the sequence number IFILE of the file to be generated.
*  IFILE is incremented by one after a file is generated.
*
      ifile   = 1
*-----------------------------------------------------------------------
*  Initialize the flag to indicate that first file be generated
*
      flgenx  = .false.
*-----------------------------------------------------------------------
*  Set parameters for the control of the screen and cursor moving
*
      call scg009 (esc, clrs, clrl, crvon, crvof, cblon)
*-----------------------------------------------------------------------
*  Display the unchanging part of the menu box on the screen
*
      call schl01 (esc, clrs, crvon, crvof, pgn, vn, cv)
*-----------------------------------------------------------------------
*  Initialize project title, to be re-initialized for IFILE=1.
*
      titxt   = ' '
*-----------------------------------------------------------------------
*  Initialize variables relating to the FE grid data file.
*  The values relating to the grid file used for the previous
*  combination map remain valid, unless overwritten by reloading or
*  reinitialized (see below) automatically if the grid file data do not
*  fit to the map data, i.c. if NODFEM is not =NOD1 (error check) via
*  subroutine SCHL20.
*
      fnine   = ' '
      fline   = .false.
      flfem   = .false.
*-----------------------------------------------------------------------
*  Initialize the error code number IERR. IERR can be reset (>0) if
```

```
*  'Enter Control Data'. Then in SCHL0A it will be reset =0 again.
*
      ierr    = 0
*-----------------------------------------------------------------------
*  Initialize number of lines in options and text window
*
   10 nlopw   = 0
      nltxw   = 0
*-----------------------------------------------------------------------
*  Initialize variables
*
      flnext  = flgenx
      flout   = .false.
      fnout   = ' '
      flfil1  = .false.
      flfil2  = .false.
      flcnt   = .false.
      fninp1  = ' '
      flinp1  = .false.
      fninp2  = ' '
      flinp2  = .false.
      labpar  = ' '
*
*  IPAR1 and IPAR2 must be =0 on the first entry of SCHL1A
*
      ipar1   = 0
      ipar2   = 0
*
*  FLCMB is reset via SCHL20 and indicates whether PARC was filled.
*  FLCHG1 and FLCHG2 indicate (via SCHL20) whether PAR1 and PAR2 were
*  adapted before PARC to be generated.
*
      flcmb   = .false.
      flchg1  = .false.
      flchg2  = .false.
*-----------------------------------------------------------------------
*  Main menu
*
   20 call schl0a (esc, clrl, crvon, crvof, cblon, cv, fnout, flout,
     &             flfil1, flfil2, flcnt, flnext, titxt, nlopw, nltxw,
     &             ifile, ime00, ierr, nlerr, ilerr)
*-----------------------------------------------------------------------
*  IME00=-2 : option 'Generate Output'
*
      if (ime00.eq.-2) then
*
*  Open the file FNOUT and generate the file
*
   ..........................................
   ..........................................
      endif
*-----------------------------------------------------------------------
*  IME00=1 : option 'Quit' from the 'Main Menu'
*
      if (ime00.eq.1)  call schl13 (clrs, pgn, vn, ifile-1)
*-----------------------------------------------------------------------
*  Proceed with either 'First Map Load' (IME00=-41),
*  'Second Map Load' (IME00=-42),
*  or 'Enter Control Data' (IME00=-3).
*
   ..........................................
   ..........................................
*-----------------------------------------------------------------------
*  Treat the option 'First Map Load'
*
   ..........................................
```

```
   ..........................................
*
*  IME1A=0 : --> 'Return' (= 'Main Menu')
*  IME1A=1 : 'Quit'
*
      if (imela.eq.0)  goto 20
      if (imela.eq.1)  call schl13 (clrs, pgn, vn, ifile-1)
*-----------------------------------------------------------------------
*  Treat the option 'Second Map Load'
*
   ..........................................
   ..........................................
*
*  IME1B=0 : --> 'Return' (= 'Main Menu')
*  IME1B=1 : 'Quit'
*
      if (imelb.eq.0)  goto 20
      if (imelb.eq.1)  call schl13 (clrs, pgn, vn, ifile-1)
*-----------------------------------------------------------------------
*  Treat the option 'Enter Control Data'
*
   ..........................................
   ..........................................
      call schl20 (esc, clrl, crvon, crvof, cblon, cv, nodl, titxt,
     &             labpar, parc, icmfll, itpl, iunl, iunpol, parl,
     &             parol, parmnl, parmxl, paravl, icmfl2, itp2, iun2,
     &             iunpo2, par2, paro2, parmn2, parmx2, parav2, flldfe,
     &             flfem, ife, fnine, fline, nelfem, nbpfem, nlopw,
     &             nltxw, ime20, flcmb, flchgl, flchg2)
*
*  At this point IME20 is either =0 (Parent Menu) or =1 (Quit)
*
      if (ime20.eq.1)  call schl13 (clrs, pgn, vn, ifile-1)
*
*  Re-set the flag and return to the 'Main Menu'
*
      if (flcmb)  flcnt = .true.
*
*  'Enter Control Data' is not fully processed if NELFEM and NBPFEM
*  have not been defined (FLFEM=false) while it was required.
*  FLFEM=false means that either the grid file IFE was not loaded
*  and/or NODFEM was not =NOD1.
*
      if (flldfe) then
         if (.not.flfem)  flcnt = .false.
      endif
      goto 20
      end
```

```
*************************************************************************
*                                                                       *
*                    >>  SUBROUTINE SCHL0A  <<                          *
*                                                                       *
*-----------------------------------------------------------------------*
*  purpose : Generate the 'Main Menu" for a program for generating of   *
*            a file containing a combination of two map parameters.     *
*                                                                       *
*            The following options are available in 'Main Menu':        *
*            IOPT  Option Text          IMECO                           *
*            ----  -----------------    -------                         *
*              1   Output File           not relevant   (FLOUT)        *
*     menu >>  2   First Map Load       -41            (FLFIL1)        *
*     menu >>  3   Second Map Load      -42            (FLFIL2)        *
*     menu >>  4   Enter Control Data    -3            (FLCNT)         *
*              5   Generate Output       -2                            *
*              6   Next Output           -1            (FLNEXT)        *
*              7   Quit                   1                            *
*                                                                       *
*            FLNEXT=false, permitted options are 1,2,3,4,5 and 7.       *
*            FLNEXT=true , permitted option are 6 and 7.                *
*                                                                       *
*            Error code number IERR:                                    *
*            -----------------------                                    *
*            IERR = 0 : no error was issued                             *
*                                                                       *
*            IERR > 0 : error was issued (NLERR lines in the text       *
*                       window, starting in line ILERR)                 *
*                       Permitted options are:                          *
*                       'First Map Load', 'Second Map Load' or 'Quit'.  *
*                       IERR is reset =0 before return to the calling   *
*                       module.                                         *
*-----------------------------------------------------------------------*
*  parameters :                                                         *
*                                                                       *
*  ESC    = ASCII character 27                                  (i)    *
*           (character variable)                                        *
*  CLRL   = character string for clearing the line              (i)    *
*           (character*4 variable)                                      *
*  CRVON  = character string for switching reverse video on     (i)    *
*           (character*5 variable)                                      *
*  CRVOF  = character string for switching reverse video off    (i)    *
*           (character*5 variable)                                      *
*  CBLON  = character string for switching bold char. on        (i)    *
*           (character*5 variable)                                      *
*  CV     = vertical bar for restoring menu box                 (i)    *
*           (character*1 variable)                                      *
*  FNOUT  = output file name, defined on input if FLOUT(i)=true (i/o)  *
*           (character*30 variable)                                     *
*  FLOUT  = flag whether output file is defined                 (i/o)  *
*           (logical variable)                                          *
*  FLFIL1 = flag of whether file data 1 is loaded               (i)    *
*           (logical variable)                                          *
*  FLFIL2 = flag of whether file data 2 is loaded               (i)    *
*           (logical variable)                                          *
*  FLCNT  = flag of whether control data is loaded              (i)    *
*           (logical variable)                                          *
*  FLNEXT = flag of whether option "Next Output" allowed (y:t)  (i)    *
*           (logical variable)                                          *
*  CTITL  = title text                                          (i)    *
*           (character*40 variable)                                     *
*  NLOPW  = number of lines currently in option window          (i/o)  *
*           (integer variable)                                          *
*  NLTXW  = number of lines currently in text window            (i/o)  *
*           (integer variable)                                          *
*  IFILE  = sequence number of the file to be generated         (i)    *
```

```
*           (integer variable)                                          *
*  IMECO  = code number defining next menu to be displayed      (o)    *
*           (integer variable)                                          *
*  IERR   = code number whether error was issued                (i/o)  *
*           (integer variable)                                          *
*  NLERR  = number of error lines in text window (if IERR>0)    (i)    *
*           (integer variable, 0 < NLERR)                               *
*  ILERR  = first line of error text in text window (if IERR>0) (i)    *
*           (integer variable, 0 < ILERR)                               *
*--------------------------------------------------------------------*
*  subprograms    : scg001, scg004, scg007, scg008, scg010, scg016,   *
*                   scmcu1, scmcu2                                     *
*--------------------------------------------------------------------*
*  error messages : 9                                                 *
*--------------------------------------------------------------------*
*  implementation : none                                              *
*--------------------------------------------------------------------*
*  Copyright RIVM, The Netherlands          Last update : 06-DEC-1989  *
**********************************************************************
      subroutine schl0a (esc, clrl, crvon, crvof, cblon, cv, fnout,
     &                   flout, flfil1, flfil2, flcnt, flnext, ctitl,
     &                   nlopw, nltxw, ifile, imeco, ierr, nlerr,
     &                   ilerr)
      implicit double precision (a-h,o-z)
*---------------------------------------------------------------------
      parameter (nopt=7)
      character chopt(nopt)*18, chfst(nopt)
      logical   flfst(nopt), flusac(nopt), flfstp(nopt)
*---------------------------------------------------------------------
      character esc, clrl*4, crvon*5, crvof*5, cblon*5, cv
      character fnout*30, ctitl*40
      character txer1*49, txer2*49, txer3*49, txer4*49, txer5*49
      character txer6*49, txer7*49, txer8*49, txer9*49
      logical   flout, flfil1, flfil2, flcnt, flnext, flreke, flinrk
*---------------------------------------------------------------------
*  Set the options and arbitrary text
*
      character chtxt*17, txtbl*49
      character txt1*18, txt2*18, txt3(2)*10
      data chopt /'Output File','First Map Load', 'Second Map Load',
     &            'Enter Control Data', 'Generate Output',
     &            'Next Output','Quit'/
      data chtxt/'Output File    = '/
      data txt1/'Map Parameter 1 : '/
      data txt2/'Map Parameter 2 : '/
      data txt3/'Loaded    ','Not Loaded'/
      data txtbl/' '/
*---------------------------------------------------------------------
*  Set the text for error messages
*
      data txer1/'Error: Option "Next Output" is Not Allowed'/
      data txer2/'Error: Select "Next Output" or "Quit"'/
      data txer3/'Error: "Output File" Must be Defined First'/
      data txer4/'Error: Permitted are "Next Output" or "Quit"'/
      data txer5/'Error: "Enter Control Data" first'/
      data txer6/'Error: Neither of 2 Map Parameters is Loaded'/
      data txer7/'Error: First Map Parameter is not Loaded'/
      data txer8/'Error: Second Map Parameter is not Loaded'/
      data txer9/'Permitted are "First or Second Map Load or "Quit"'/
*---------------------------------------------------------------------
*  Set flag whether the ASCII characters defining the options (CHFST)
*  be re-defined such that a carriage return is included after an
*  option is selected.
*
      data flusac /7*.true./
      flreke  = .true.
```

```
*---------------------------------------------------------------------
*  Initialize the flag FLINRK. FLINRK is set =false after the flag
*  array FLFSTP is initialized later in the program. After that
*  FLINRK remains =false.
*
      flinrk  = .true.
*---------------------------------------------------------------------
*  Display the title text (normal mode)
*
      call scmcul (esc, 4, 17, ctitl, 40)
*---------------------------------------------------------------------
*  Display the menu name in reverse video at the bottom of options box
*
      call scmcu2 (esc, crvon, crvof, 22, 3, '    Main Menu     ', 18)
*---------------------------------------------------------------------
*  1) Define the first character of options text (CHFST)
*  2) Display the options in options window (normal characters)
*  3) Clear options window, clear text window and
*  4) Reset NLOPW and NLTXW
*
      call scg004 (esc, chopt, chfst, nopt, nlopw, nltxw)
      nlopw   = nopt
      nltxw   = 1
      ......................
      ......................
*---------------------------------------------------------------------
*  Display the loading status of the two map files (lines 17 and 18).
*  Because this text is not comprised in the value of NLTXW,
*  the text is to be wiped out before each return.
*  The text is written starting in position 25.
*
*  Do not write if an error message was issued earlier.
*  To avoid programmimg complexity set IL1 and IL2.
*
      il1     = 17
      il2     = 18
      ncl     = 18
      if (ierr.lt.1) then
         call scmcul (esc, il1, ipos, txt1, ncl)
         call scmcul (esc, il2, ipos, txt2, ncl)
         if (flfil1) then
            call scmcul (esc, il1, ipos+ncl, txt3(1), 10)
         else
            call scmcul (esc, il1, ipos+ncl, txt3(2), 10)
         endif
         if (flfil2) then
            call scmcul (esc, il2, ipos+ncl, txt3(1), 10)
         else
            call scmcul (esc, il2, ipos+ncl, txt3(2), 10)
         endif
      endif
*---------------------------------------------------------------------
*  Set the flag for 'Quit' that remains highlighted
*
      flfst(7) = .true.
*---------------------------------------------------------------------
*  Define the relevant options (FLFST)
*
*  Initialize (or reset) flags
*
   10 flfst(1) = .true.
      flfst(2) = .true.
      ......................
      ......................
*---------------------------------------------------------------------
*  Initialize the flag FLFSTP to economize the re-setting of the first
```

```
*  character CHFST in subroutine SCG016.
*
   20 if (flinrk) then
         flfstp(1) = .not.flfst(1)
         flfstp(2) = .not.flfst(2)
         ......................
         ......................
      endif
*---------------------------------------------------------------------
*  Write the first character CHFST of the NOPT options
*
      call scg016 (esc, cblon, crvof, chfst, nopt, flfst, flfstp)
*---------------------------------------------------------------------
*  Move cursor to select option position and select an option
*
   30 call scg008 (esc, clrl, cblon, crvof, cv, chfst, nopt, flreke,
     1             flusac, iopt)
*---------------------------------------------------------------------
*  If an error message of was issued elsewhere, wipe out the message
*  (NLERR text lines, starting in line ILERR) and re-set the flag
*  IERR, otherwise on the next entry of this subroutine wrong action.
*
      if (ierr.gt.0) then
         do 40 i = ilerr,ilerr+nlerr-1
            call scmcu1 (esc, i, 25, txtbl, 49)
   40    continue
      endif
      ......................
      ......................
*---------------------------------------------------------------------
*  For a change of the menu, set the value of IMECO to -41 (First Map
*  Load), -42 (Second Map Load), -3 (Enter Control Data), -2 (Generate
*  Output), -1 (Next Output) or 1 (Quit)
*
   50 if (iopt.ge.2) then
*
*  IOPT = 2,3,4,5,6 or 7.
*  Option IOPT=1 is to be treated separately later.
*
         if (iopt.eq.2)  imeco = -41
         if (iopt.eq.3)  imeco = -42
         if (iopt.eq.4)  imeco = -3
         if (iopt.eq.5)  imeco = -2
         if (iopt.eq.6)  imeco = -1
         if (iopt.eq.7) then
*
*  Option 'Quit'
*
            imeco    = 1
            call scmcu1 (esc, il1, ipos, txtbl, 49)
            call scmcu1 (esc, il2, ipos, txtbl, 49)
            return
         endif
*
*  Check whether a meaningful option was selected, IOPT=4,5 or 6
*
         if (iopt.eq.4 .and. (.not.flfil1 .and. .not.flfil2)) then
            call scmcu2 (esc, cblon, crvof, 22, 25, txer6, 49)
            goto 30
         endif
         if (iopt.eq.4 .and. .not.flfil1) then
            call scmcu2 (esc, cblon, crvof, 22, 25, txer7, 49)
            goto 30
         endif
         if (iopt.eq.4 .and. .not.flfil2) then
            call scmcu2 (esc, cblon, crvof, 22, 25, txer8, 49)
```

```
            goto 30
         endif
         if (iopt.eq.5 .and. .not.flout) then
            call scmcu2 (esc, cblon, crvof, 22, 25, txer3, 49)
            goto 30
         endif
         ......................
         ......................
         return
      endif
*-------------------------------------------------------------------
*  IOPT=1 : Check whether FLNEXT=true, illegal option
*
      if (flnext) then
         call scmcu2 (esc, cblon, crvof, 22, 25, txer4, 49)
         goto 30
      endif
*
*  Define the output file
*
      ......................
      ......................
*-------------------------------------------------------------------
*  Return to re-paint the options window
*
      goto 10
      end
```

```
HL-subroutines:
HLnnXX : control execution in programs HLnn, nn=01, 02 and 03
HL00nn : generally applicable procedures in all HL-programs

IO-subroutines:
IORFtt : read a record from unformatted file
IOSUtt : set up the text string defining units of parameter
IOXXnn : setting of the basic parameters for all AQ-programs

SC-subroutines:
SCDDnn : screen handling in all AQ-programs treating DD-data
SCG0nn : generally applicable screen handling in all AQ-programs
SCHLnn : screen handling in programs HL01, HL02 and HL03
SCMCUn : basic screen manipulation, applicable in all AQ-programs
SCPMnn : screen handling in all AQ-programs treating map-data

VN-subroutines:
VNHLnn : set program version for programs HLnn, nn=01, 02 and 03

Toelichting: n = cijfer (0, 1, 2, etc)
             t = letter (a, b, c, etc)

       'nn' increases sequentially, starting from 01
       'n'  increases sequentially, starting from 1
```

# D VOORBEELD PROGRAMMA EPOT (STARING CENTRUM)

```
      PROGRAM EPOT
C
C
C
C           PROGRAMMA HEADER
C
C
C*************************************************************************
C
C         CALCULATE EVAPOTRANSPIRATION FROM METEO DATA ON A DAILY BASE
C
C*************************************************************************
C*********************************************************  COMMON  ****
C
C         PARAMETERS, COMMON AND CHARACTERS
      INCLUDE 'EPOT.INC'
C
C*****************************************************  PRE-PROC.  ****
C
C          INITIALIZE - CONSTANTS AND OPEN FILES
      CALL INIT
C
C          READ INPUT DATA
      CALL READ
C
C          CHECK INPUT DATA
      CALL CHECK
C
C*****************************************************  MAIN PROG  ****
C
C          CALC. REFERENCE EVAPOTRANSPIRATION (MAKKINK)
      CALL REF
C
C          CALC EVAPOTR. (Eo-PENMAN)
      CALL PENMAN
C
C          CALC. EVAPOTR. FOR PINE AND DECIDIOUS FOREST
      CALL FOREST
C
C*****************************************************  OUTPUT DATA  ****
C
C          SUMMATION OF DAILY DATA PER MONTH, SEASON, ETC.
      CALL SUM
C
C          PRINT OUTPUT
      CALL PRINT
C
C          SAVE DATA TO PLOT
      CALL PLOT
C
```

```
C*************************************************************************
C
      CLOSE (IN)
      CLOSE (IO)

      CLOSE (IS)
C
C         END OF PROGRAMME
      WRITE(IW,1)
C
C*********************************************************  FORMATS  ****
C
C
    1 FORMAT(/'       READY'//
     & '  The accumulated results per half year are also written to',
     &  ' file :  METEO.SUM'/ )
      END
      SUBROUTINE INIT
C
C
C        ******    SUBROUTINE HEADER    ******
C
C        INITIALIZE AND OPEN FILES
C
C*********************************************************  COMMON  ****
C
C        PARAMETERS, COMMON AND CHARACTERS
      INCLUDE 'EPOT.INC'
C
C*************************************************************************
C
C        CONSTANTS
C
      DATA GAMMA/0.66713/
      DATA BETA/2.00/
      DATA IMND/0,31,59,90,120,151,181,212,243,273,304,334,365/
C
C        INTERCEPTION RESERVOIR PINE FOREST
      RES = 0.15
C
C        CONVERSION FACTOR FOR INPUT METEO DATA TO CM
C        PRESENT SETTING METEO DATA IN MM
      MFAC = 10.
C
C
C*************************************************  OPEN FILES  ****
C
C         IN  - meteo data on daily base
C         IO  - output evapotr. (daily base)
C         IS  - file to save results for plot of data and results
C         IR  - read from screen
C         IW  - write to screen
C
C         FILE CODES
      IN = 01
      IO = 02
      IS = 04
      IR = 05
      IW = 06
C
C
      WRITE(IW,1)
C
  100 WRITE(IW,2)
      READ(IR,'(A20)') FILEIN
```

```
C
      WRITE(IW,3)
      READ(IR,'(A20)') FILEIO
C
      OPEN(IN,FILE=FILEIN,STATUS='OLD',ERR=110)
C
      OPEN(IO,FILE=FILEIO,STATUS='NEW',CARRIAGECONTROL='LIST')          VAX
C      OPEN(IO,FILE=FILEIO,STATUS='UNKNOWN')                            MIC
C
      OPEN(IS,FILE='METEO.SUM',STATUS='NEW',CARRIAGECONTROL='LIST')     VAX
C      OPEN(IS,FILE='METEO.SUM',STATUS='UNKNOWN')                       MIC
      GO TO 120
C
C        INPUT METEO FILE NOT FOUND
  110 WRITE(IW,4) FILEIN
      GO TO 100
C
  120 CONTINUE
C
      RETURN
C
C*******************************************************  FORMATS  ****
C
    1 FORMAT(//5X,'CALCULATION OF EVAPOTRANSPIRATION DATA'// )
    2 FORMAT(/' Give filename with meteo data  ------------------ ? ',$)
    3 FORMAT(/' Give filename for results  ---------------------- ? ',$)
    4 FORMAT(/' File :',A20,'   could not be attached ? - try again' )
      END
      SUBROUTINE READ
C
C
C        ******    SUBROUTINE HEADER    ******
C
C        READ TYPE OF DATA AND ACTUAL METEO DATA (PER DAY)
C
C*******************************************************  COMMON  ****
C
C        PARAMETERS, COMMON AND CHARACTERS
      INCLUDE 'EPOT.INC'
C
C*************************************************************************
C
C        ISUMB  - begin of summer period at day 90
C        ISUME  - end of summer period at day 270
C            The Netherlands
      ISUMB =  90
      ISUME = 270
C
      WRITE(IW,1)
      READ *, IHEM
C
      IF ( IHEM .EQ. 2 ) THEN
```

```
C            Southern hemisphere
         ISUMB = 270
         ISUME =  90
      ENDIF
C
      WRITE(IW,2) ISUMB,ISUME
C
C*********************************************************************
C
C         NSK  - number of records to skip in input file with text
C         IDI  - type of input data
C         IPR  - output half year accumulated results to screen (1=yes)
C
  100 WRITE(IW,3)
      READ *, NSK
C
      DO 110 I=1,NSK
         READ(IN,'(A1)') TXT
  110 CONTINUE
C
C          INPUT TYPE OF METEO DATA
  120 WRITE(IW,4)
      READ *, IDI
      IF ( IDI .LE. 0 .OR. IDI .GT. 5 ) GO TO 120
C
      IF ( IDI .EQ. 5 ) THEN
C             SHOW TEXT AND FIRST DATA RECORD
         REWIND(IN)
         WRITE(IW,5)
C              TEXT
         DO 130 I=1,NSK
         READ(IN,6) AREC
         WRITE(IW,7) AREC
  130    CONTINUE
C              FIRST DATA RECORD
         READ(IN,6) AREC
         WRITE(IW,7) AREC
         BACKSPACE (IN)
         GO TO 120
      ENDIF
C
C        RESULTS PER HALF YEAR TO SCREEN ?
      WRITE(IW,8)
      READ(IR,'(A1)') ANT
C
      IPR = 0
      IF ( ANT .EQ. 'Y' .OR. ANT .EQ. 'y' ) THEN
         IPR = 1
         WRITE(IW,9)
      ENDIF
C
C*********************************************************************
C
C        WRITE HEADING FOR OUTPUT FILE AND FOR SCREEN
      IF ( IPR .EQ. 1 ) THEN
         IF ( IDI .EQ. 4 ) THEN
           WRITE(IW,11)
         ELSE
```

```
          WRITE(IW,10)
        ENDIF
      ENDIF
      IF ( IDI .EQ. 4 ) THEN
        WRITE(IO,11)
        WRITE(IS,11)
      ELSE
        WRITE(IO,10)
        WRITE(IS,10)
      END IF
C
C*********************************************  READ DATA PER DAY  ****
C
C         IDAG  - day number
C         IJAAR - year
C         PREC  - precipitation (mm)
C         TEM   - temperature (degrees)
C         RH    - relative humidity
C         DCL   - degree of cloud cover (n/N) (-)
C         WIND  - windspeed ( not used )
C         HSH   - incoming global ( short wave ) radiation ( W/m2 )
C         HNT   - nett radiation ( W/m2 )
C
      ID = 0
C
  140 CONTINUE
C
C         RECORDS COUNTER
      ID = ID + 1
C
      IF ( IDI .EQ. 1 ) THEN
C
         READ(IN,*,END=150) IDG,MND,IJAAR(ID),PREC(ID),TEM(ID),RH(ID),
     &        WIND(ID),HSH(ID),HNT(ID)
C
C             CONVERT DAY AND MONTH TO DAY NUMBER
         IDAG(ID) = IDG + IMND(MND)
C
      ELSEIF ( IDI .EQ. 2 ) THEN
C
         READ(IN,*,END=150) IDAG(ID),PREC(ID),TEM(ID),RH(ID),HSH(ID)
C
      ELSEIF ( IDI .EQ. 3 ) THEN
C
         READ(IN,*,END=150) IDAG(ID),IJAAR(ID),PREC(ID),TEM(ID),
     &        RH(ID),HSH(ID)
C
      ELSEIF ( IDI .EQ. 4 ) THEN
C
         READ(IN,*,END=150) IDAG(ID),IJAAR(ID),PREC(ID),TEM(ID),
     &        RH(ID),HSH(ID),DCL(ID),WIND(ID)
C
      ENDIF
C
      GO TO 140
C
C         END OF FILE REACHED
  150 CONTINUE
```

```
C
C        NUMBER OF DAYS WITH METEO DATA
      NUMD = ID - 1
C
      RETURN
C*******************************************************  FORMATS  ****
C
    1 FORMAT(/' Starting date of summer (1=Northern; 2=Southern) ',$)
    2 FORMAT(/' Summer period from day: ',I4,'  to',I4 )
    3 FORMAT(/' How many records must be skipped with text  ----- ? ',$)
    4 FORMAT(/' Which parameters must be read'/
     &    5X,'1 - IDAG,MND,IJAAR,PREC,TEM,RH,WIND,HSH,HNT (Makkink)'/
     &    5X,'2 - IDAG,PREC,TEM,RH,HSH                    (Makkink)'/
     &    5X,'3 - IDAG,IJAAR,PREC,TEM,RH,HSH              (Makkink)'/
     &    5X,'4 - IDAG,IJAAR,PREC,TEM,RH,HSH,DCL,WIND      (Penman)'/
     &    5X,'5 - unknown, show first data record and start again'//
     &       '                              ENTER CHOICE  ---------- ? ',$)
    5 FORMAT(/' Data of first record is : '/ )
    6 FORMAT(A70)
    7 FORMAT(2X,A70)
    8 FORMAT(/' Accumulated half year results to screen '/
     &       '  ( Y or N )  ---------------------------------- ? ',$)
    9 FORMAT(//)
   10 FORMAT('  Day  Year   Rainfall  ETp gras  ETp pine  ETp deci',
     &   '  evap. fac.'/
     &      '                 [mm]      [mm]      [mm]      [mm]  ',
     &   ' fallow soil' )
   11 FORMAT('  Day  Year   Rainfall  Eo (Pen)  ETp pine  ETp deci',
     &   '  evap. fac.'/
     &      '                 [mm]      [mm]      [mm]      [mm]  ',
     &   ' fallow soil' )
      END
      SUBROUTINE REF
C
C
C       ******    SUBROUTINE HEADER    ******
C
C
C         CALC. REFERENCE EVAPOTR. (MAKKINK)
C
C*******************************************************  COMMON  ****
C
C       PARAMETERS, COMMON AND CHARACTERS
      INCLUDE 'EPOT.INC'
C
C*************************************************************************
C
C        NUMD   - number of records (days) with data
C        TEM    - temperature (degrees)
C        HSH    - incoming global ( short wave ) radiation ( W/m2 )
C        MFAC   - conversion from cm to default unit (Subr. INIT)
C
C       TEMP     - temperature in K
C       EV      - verzadigingsdampdruk
C       DEL     - helling van de verzadigingsdampdrukcurve
C
      DO 100 ID=1,NUMD
```

```
C
      TEMP   = TEM(ID) + 273.15
      WED    = .0583 * TEMP - 2.1938
      EV     = 1.3332 * EXP( (1.08872*TEMP-276.4884) / WED)
      DEL    = 13.7315 * EV / (WED**2)
C
C               EQUATION OF MAKKINK - EVAPOTRANSPIRATION IN CM
C               CONVERSION TO DEFAULT UNIT (MM OR M) BY MFAC
      EPG  = 0.65 * HSH(ID) * DEL / (DEL+GAMMA)
      EPGRAS(ID) = 0.00352 * EPG * MFAC
C
  100 CONTINUE
C
      RETURN
      END
```

Aantekeningen

COMMISSIE VOOR HYDROLOGISCH ONDERZOEK TNO

RAPPORTEN EN NOTA'S

No. 1. Tweede rapport en aanbevelingen
van de Contactgroep Archivering en Automatische Verwerking van hydrologische gegevens TNO.
Januari 1977.

No. 2. Verslag en aanbevelingen
van de ad hoc-Groep Grondwatermodellen en Computerprogrammatuur TNO.
Juli 1978.

No. 3. De droogte in 1976.
Een samenvatting en overzicht van de over de droogte van 1976 verschenen literatuur - P.K.M. v.d. Heijde.
Augustus 1978.

No. 4. Nederlandse activiteiten in internationaal hydrologisch verband.
Lezingserie, gehouden op 25 april 1978 te Delft, aangevuld met (schematische) overzichten van internationale organisaties en een overzicht van hun vertegenwoordigers in Nederland.
Augustus 1978.

No. 5. Waterkwaliteit in grondwaterstromingsstelsels.
Verslag van de Workshop op 1 en 2 april 1980 te Wageningen - (red. J.C. Hooghart), aangevuld met discussiebijdragen en een inventarisatie van het onderzoek in Nederland.
Augustus 1980.

No. 6. Derde rapport en aanbevelingen
van de Contactgroep Archivering en Automatische verwerking van hydrologische gegevens TNO.
Februari 1981.

No. 7. Overzicht van de wensen van hydrologen en waterbeheerders ten aanzien van het operationele regenwaarnemingennet van het KNMI - J.C. Hooghart.
Oktober 1981.

No. 8.*) Verklarende Hydrologische Woordenlijst van de Gespreksgroep Hydrologische Terminologie.
8a. I. Water in de onverzadigde zone
II. Water in de verzadigde zone
Januari 1982.
8b. III. Atmosferisch water
Juni 1983.
8c. IV. Oppervlaktewater
Maart 1985.

*) Verouderd: vervangen door Rapporten + Nota's no. 16.

No. 9. Waterkwaliteit en waterkwantiteit in het IJsselmeergebied.
Verslag van de 2e CHO-studiebijeenkomst op 2 en 3 november 1981, De Eemhof, Zuidelijk Flevoland - (red. J.C. Hooghart), aangevuld met discussiebijdragen.
Februari 1982.

No. 10. Rapport en aanbevelingen
van de Contactgroep Grondwatermodellen, CHO-TNO.
April 1982.

No. 11. Inventarisatie Grondwaterkwaliteitsmodellen.
L.J.M. Boumans.
Oktober 1982.

No. 12. Grondwaterkwaliteit in relatie met onderzoek en beleid.
Verslag van de 3e CHO-studiebijeenkomst op 15 maart 1983 te Wageningen - (red. J.C. Hooghart), aangevuld met discussiebijdragen.
Juni 1983.

No. 12a. Voorlopig overzicht van inventarisaties waarin grondwater(kwaliteits)modellen voorkomen of hiermee in verband staan.
J.C. Hooghart.
Januari 1984.

No. 13. Vergelijking van modellen voor het onverzadigd grondwatersysteem en de verdamping.
Verslag van de 4e CHO-studiebijeenkomst op 24 oktober 1984, georganiseerd in samenwerking met de Studiegroep Hupselse Beek - (red. J.C. Hooghart).
Maart 1985.

No. 14. Meten, meetnetten en optimale meetnetontwerpen ten dienste van het waterbeheer.
Verslag van:
- Voorjaarsbijeenkomst van de KIvI Sectie Waterbeheer:
  "Meten voor waterbeheer", mei 1984.
- Colloquium van de Studiegroep Statistiek in de hydrologie CHO-TNO:
  "Meetontwerp en optimalisatie", november 1984.
  (red. P. v.d. Kloet en J.C. Hooghart).
  Januari 1986.

No. 15. Het hydrologisch systeem in het grensgebied Luik-Maasbracht.
Le système hydrologique dans la région frontalière Liège- Maasbracht.
Verslag van de 5e CHO-studiebijeenkomst op 13 december 1985, georganiseerd in samenwerking met de Nationale IHP-comité's van België en Nederland en de Contactgroep Hydrologie van het Nationaal Fonds voor Wetenschappelijk Onderzoek uit België.
(red. J.C. Hooghart).
April 1986.

No. 16. Verklarende Hydrologische Woordenlijst van de Gespreksgroep Hydrologische Terminologie, waarin opgenomen de hoofdstukken:
I Algemene termen
II Atmosferisch Water
III Water in de onverzadigde zone
IV Water in de verzadigde zone
V Oppervlaktewater
Oktober 1986, hernieuwde uitgave.

No. 17.*) Duurzaamheid rioolleidingen; een literatuurstudie naar aantastingsmechanismen.
R.B. Polder.
Februari 1987.
*) Uitverkocht.

No. 18. Ruimtelijke variabiliteit van bodem en water.
Verslag van de 6e CHO-studiebijeenkomst op 22 oktober 1986.
(red. J.C. Hooghart).
Februari 1987.

No. 19. Van Penman naar Makkink; een nieuwe berekeningswijze voor de klimatologische verdampingsgetallen.
Eindrapport van de KNMI-Projectgroep en de CHO-Begeleidingsgroep Verdampingsberekeringen.
(red. J.C. Hooghart en W.N. Lablans).
December 1988.

No. 20. Tijdreeksen in bodem en water.
Inleidingen van de lezingendag op 25 januari 1989 van de NRLO-Werkgroep Ruimtelijke variabiliteit in bodem en water en de Studiegroep Statistiek in de Hydrologie van de CHO-TNO.
December 1988.

No. 21. Neerslagmeting en -voorspelling; toepassing van modern technieken, zoals radar- en satellietwaarnemingen.
Verslag van de 7e CHO-studiebijeenkomst, georganiseerd in samenwerking met SAMWAT, op 16 november 1988.
(red. J.C. Hooghart).
Februari 1989.

No. 22. Integraal Waterbeheer in het Goois/Utrechts stuwwallen- en plassengebied.
Verslag van de op 7 april 1989 in Bussum gehouden themadag, georganiseerd door het Zuiveringschap Amstel en Gooiland en de Provincie Utrecht, in samenwerking met de CHO-TNO.
(red. L. van Liere, R.M.M. Roijackers en P.J.T. Verstraelen).
Augustus 1989.

No. 23. Bodemwaterkwaliteit in wisselwerking met biologische, chemische en hydrologische processen.
Verslag van de 8e CHO-studiebijeenkomst op 8 mei 1990.
(red. J.C. Hooghart)
September 1990.

No. 24. Ruimtelijke statistiek van bodem en water.
Inleidingen van de lezingendag op 24 januari 1991 van de NRLO-werkgroep Ruimtelijke variabiliteit van bodem en water en de Studiegroep Statistiek in de Hydrologie van de CHO-TNO.
(red. J.C. Hooghart)
Januari 1991.

No. 25. Geo-informatie in Nederland.
Inleidingen van de lezingendag op 2 mei 1991 in samenwerking met het Samenwerkingsverband Aardkundige Gegevensverstrekkende Instituten (SAG II).
(red. J.C. Hooghart)
Mei 1991.

No. 26. Het hydrologisch systeem in het grensgebied Luik-Maasbracht; onderzoeksresultaten 1985-1990.
Le système hydrologique dans la région frontalière Liège-Maasbracht; résultats des recherche 1985-1990.
Verslag van de 9e CHO-studiebijeenkomst op 9 januari 1991, georganiseerd in samenwerking met de Nationale IHP-comité's van België en Nederland en de Contactgroep Hydrologie van het Nationaal Fonds voor Wetenschappelijk Onderzoek uit België.
(red. J.C. Hooghart).
Augustus 1991.

No. 27. Richtlijnen voor computerprogrammatuur in de hydrologie.
Eindrapport van de CHO-Werkgroep Richtlijnen Computerprogrammatuur Hydrologie.
(red. J.C. Hooghart, K. Kovar en J.M.P.M. Peerboom)
Oktober 1992.

No. 28 Integraal (water)beheer in de praktijk haalbaar?
Verslag van de op 7 april 1992 in Amsterdam gehouden themadag, met aanvullingen.
(red. R.M.M. Roijackers, P.J.T. Verstraelen en L. van Liere).
(in druk).

No.29 HYDRO - LOGISCH; wetenschap en toepassing.
Verslag van het symposium op 5 oktober 1992 ter gelegenheid van het afscheid van H.J. Colenbrander van de CHO-TNO.
(red. J.C. Hooghart en C.W.S. Posthumus).
Oktober 1992.

Voor bestellingen en informatie: CHO-TNO
Postbus 6067
2600 JA DELFT
Telefoon: 015 - 69 72 81