De didactiek van het programmeren

Nascholingscursus Informatica voor leraren.

De didactiek van het programmeren.

drs A. Kaldewaij
Onderafdeling der Wiskunde en Informatica
Technische Hogeschool Eindhoven

augustus 1984

Inhoud

0. Voorwoord 1

1. Informatica in de onderbouw 2

1.1 Inleiding 2

1.2 De huidige situatie in de onderbouw 3

1.3 Een zinvolle invulling van het informatica-onderwijs in de onderbouw 6

2. Programmeren 8

2.1 Doelstellingen 8

2.2 De begintoestand van de leerlingen 10

2.3 De opzet van een onderwijsprogramma 12

2.4 Propositierekening en predicatenrekening 15

2.5 Syntax en semantiek van programma's 20

2.5.1 Inleiding, functionele specificaties 20

2.5.2 De toekenningsopdracht 24

2.5.3 De concatenatie 26

2.5.4 De selectie en het skip statement 27

2.5.5 De repetitie 31

2.6 Het ontwerpen van programma's 35

2.6.1 Inleiding 35

2.6.2 Arrays en quantificaties 36

2.6.3 De uitleg van het ontwerp

0. Voorwoord

Binnen nu en enkele jaren wordt informatica een verplicht vak in de onderbouw van het voortgezet onderwijs en een keuzevak in de bovenbouw van het voortgezet onderwijs. Om docenten voor te bereiden op het onderwijs in de informatica worden zogeheten "nascholingscursussen voor leraren" georganiseerd.

De onderhavige cursus is zo'n nascholingscursus. Hierin komt de didactiek (de kunst van het onderrichten) van het programmeren aan de orde.
Het zwaartepunt van de cursus ligt op het programmeeronderwijs zoals dat gegeven zou kunnen worden in de bovenbouw van het voortgezet onderwijs en in de eerste studiejaren van het hoger beroepsonderwijs. Daarnaast wordt enige aandacht besteed aan het informaticaonderwijs in de onderbouw.

Deze cursus sluit aan bij de nascholingscursus "programmeren". Van de cursist wordt verwacht dat hij bekend is met de basisbegrippen en basistechnieken van het programmeren, zoals bijvoorbeeld het toepassen van de invariantiestelling bij het ontwerpen van een algoritme.

Dit dictaat bestaat uit notities gebaseerd op de inzichten die de auteur verworven heeft bij het onderwijzen van wiskunde en van programmeren.

De literatuurverwijzingen hebben betrekking op

[1] Een methode van programmeren.
Edsger W. Dijkstra en W.H.J. Feijen, Academic Service 1984
ISBN 90 6233 128 9.

[2] Voortgezette wiskunde, deel 1
A. Kaldewaij en J. van Tiel, Bohn, Scheltema & Holkema 1982
ISBN 90 313 0535 9.

1. Informatica in de onderbouw.

1.1 Inleiding.

Door de S.L.O. (Stichting voor de Leerplanontwikkeling, Enschede) worden voor het informaticaonderwijs aan 12-16 jarigen de volgende doelstellingen gegeven :

1. leren omgaan met gegevens.
2. leren praktisch omgaan met gegevensverwerkende systemen.
3. het kennisnemen van de toepassingen van de informatie-technologie.
4. het kennisnemen van en inzicht verwerven in de maatschappelijke gevolgen van de informatietechnologie.
5. het ontwikkelen van inzicht in de basisprincipes van gegevensverwerkende systemen door
   a. het ontwikkelen van enige vaardigheid in probleemanalyse en programmeren.
   b. het ontwikkelen van modellen en denkwijzen die het mogelijk maken adequaat om te gaan met en te denken over gegevensverwerkende systemen.

Dit onderwijs staat momenteel (1984) nog bekend onder de naam "burgerinformatica". Men stelt zich voor deze naam te vervangen door "informatiekunde".

Wij beperken ons in het navolgende tot de punten 2 en 5a van deze doelstellingen.

1.2 De huidige situatie van het informatica-onderwijs

Op een groot aantal scholen wordt reeds informatica-onderwijs gegeven. De wijze waarop dit gebeurt is uniek in het onderwijs. Redenen hiervan zijn onder meer :

- Onder druk van de media, de overheid, het schoolbestuur en het teruglopend aantal leerlingen heeft menige middelbare school apparatuur aangeschaft om zo "school met de computer" te worden. Wat aan onderwijs wordt gegeven en op welke wijze dat wordt gedaan is van weinig belang, als het maar wordt gegeven.

- Veel leerboeken en dictaten zijn geschreven bij een bepaald merk microcomputer en vormen niet veel meer dan een handleiding. Uitleg gebeurt aan de hand van waarnemingen ("Wat zie je als je op RETURN drukt ?") en voorbeelden.

- Docenten beschikken niet over de juiste kennis.
  Het merendeel van de docenten heeft programmeerervaring opgedaan door met een systeem aan de slag te gaan, soms uit hobbyïsme, soms op verzoek van de school. De wijze waarop de docent (zichzelf) heeft geleerd met een computer om te gaan en programma's te ontwikkelen wordt weerspiegeld in het onderwijs.

- Programmeren heeft geen status als wetenschappelijke discipline.
  Op veel plaatsen, ook buiten het onderwijs, heeft programmeren de status van een ambacht. Al doende leert men, met veel vallen en opstaan, bijgestaan door de in het vak doorknede vakman. Kennis van alle geheimen die een systeem in zich herbergt, speelt een grote rol.

- De ter beschikking staande apparatuur en programmatuur is gebrekkig.
  Veel apparatuur en programmatuur is onderontwikkeld en zeker niet geschikt voor het onderwijs. De tijd die zou kunnen worden besteed aan relevante concepten uit de informatica wordt verbruikt aan apparatuur.

- Programmeren is een moeilijk vak. De meeste 12-16 jarigen zijn er niet voor toegerust.

Het informaticaonderwijs is sterk door deze punten bepaald en ziet er ongeveer als volgt uit:

Er wordt uitleg gegeven van de opbouw en werking van een computer. De leerling wordt vertrouwd gemaakt met de op school aanwezige apparatuur en leert en passant programmeren in BASIC. Programma's worden veelal ontwikkeld met de apparatuur bij de hand. Soms worden blokschema's (stroomschema's, flowcharts) gebruikt.

Kenmerken van deze vorm van onderwijs zijn:

- Er zijn grote individuele verschillen: de "slimmerikken" breien al snel hele programma's in elkaar en anderen krijgen de slag nooit te pakken. Een scholier die thuis een microcomputer bezit hoeft niets meer bij te leren.
- Hobbyïsme viert hoogtij: leerlingen spenderen vele uren aan het spelen met apparatuur.
- Jongens doen het in het algemeen beter.

Binnen het Hoger Beroepsonderwijs heeft het informatica-onderwijs al enige traditie. In de jaren zeventig werd op de technische afdelingen als electrotechniek, werktuigbouwkunde en natuurkunde Algol-60 onderwezen.
Het vak heette "computerkunde" en bestond voor een groot deel uit numerieke wiskunde.
Vanaf 1980 zijn de hbo-opleidingen voorzien van eigen computers met daarbij beeldschermen en toetsenborden.
Als taal wordt in het algemeen PASCAL gebruikt. De methode van doceren is eveneens ambachtelijk. Aan de hand van voorbeelden wordt geprobeerd een programma te construeren.
Er wordt geen onderliggende theorie onderwezen.

1.3 Een zinvolle invulling van het informatica onderwijs in de onderbouw

Het informaticaonderwijs in de onderbouw kan dienen als basis voor programmeeronderwijs in latere jaren. Eén van de doelen die men zich aangaande dat aspect stelt is dat de leerlingen zoveel kennis en kunde bezitten dat het vak programmeren in isolement, los van apparatuur en programmatuur, beoefend kan worden.

Nog concreter:

- de leerling kan abstraheren van de uitvoering van een programma op een machine.
- de leerling kan abstraheren van invoer en uitvoer van gegevens.

Om dit te verwezenlijken laten we de leerlingen in de onderbouw kennis maken met apparatuur en programmatuur, met daar als doel

- enige kennis van de opbouw en werking van een klassiek computersysteem.
- enige ervaring met het bedienen van een computer.
- enige ervaring in het laten uitvoeren van programma's door een computer.
- enige ervaring met het schrijven en wijzigen van zeer eenvoudige programma's.

De benadering zal, mede gezien de leeftijd van de doelgroep, operationeel van aard zijn. Programma's zullen voornamelijk uit lees- en schrijfopdrachten bestaan. De gekozen taal is in dit stadium nauwelijks relevant.

Het onderwijs dient zich te beperken tot de genoemde doelen, opdat het niet ontaardt in hobbyisme.

Opgaven.

1. Schrijf een les waarin de opbouw en werking van een klassiek computersysteem wordt uitgelegd.

2. Geef voorbeelden van "zeer eenvoudige programma's"

3. Ontwerp een leerplan waarin de in 1.3 genoemde doelen verwerkt zijn. Ga uit van 1 lesuur per week in het tweede leerjaar van de onderbouw.
   Geef ook de practica met inhoud aan.

4. Wat voor eisen stelt u aan apparatuur en programmatuur, die gebruikt dient te worden in het onderwijs?

5. Geef van de onderstaande begrippen, zo u ze kent, de betekenis en geef tevens aan welke u van belang acht te weten.

   microcomputer, byte, adres, geheugen, main frame, store, LISP, memory, bit, stack, ALU, disk, CPU, drum, control unit, CVE, interrupt, time slice, PSW, driver, register, queue, tape, operating system, editor, COBOL, machinecode, nucleus, channel, block, I/O, floppy disk, overflow, woordlengte, instructiecode, release, task management, IBM, parity bit, assembler, Kb, virtueel geheugen, headcrash, linkage editor, hardware, RAS, chip, FIFO, weakest precondition, cache, pipelining, bestand, compiler, SR-flipflop, VLSI, load module, sequentieel, object code, ISO, normaalvorm, MIDAS, multiprogramming, flow chart, MVS, abstract data type, TSO, blending, MVS.

2. Programmeren

2.1 Doelstellingen

Een algemene doelstelling van het programmeeronderwijs is :

de leerling die kennis en kunde bijbrengen die nodig is voor het oplossen van een programmeerprobleem.

Een programmeerprobleem bestaat uit een ondubbelzinnige beschrijving van een preconditie (begintoestand) en een postconditie (eindtoestand). Een dergelijke beschrijving noemen we in navolging van [1] een functionele specificatie.

In het voortgezet onderwijs beperkt men zich tot "eenvoudige" programmeerproblemen :

een "eenvoudig programmeerprobleem" laat zich oplossen door het toepassen van standaardtechnieken.

De oplossing van een programmeerprobleem bestaat uit een programmatekst voorzien van dusdanig commentaar dat de correctheid (met betrekking tot de erbij behorende specificatie) ervan duidelijk is.

Bovenstaande doelstelling is nog alleszins vaag. Ook de erbij vermelde begrippen zijn vaag. Voor het verwezenlijken van de gegeven doelstelling leidt men andere, minder vage, af, zoals :

- Standaardstrategieën kunnen toepassen
  (Men denke aan : het weglaten van een conjunct in de eindrelatie, ter verkrijging van een geschikte invariant)
- De "lineaire zoekmethode" kunnen toepassen
- De betekenis van de variabelen welke een rol spelen in een repetitie kunnen omschrijven (" de invariant")

etc..

Naast de genoemde doelstellingen zijn nog algemenere te geven, die ook van toepassing zijn op andere vakken zoals Nederlands en wiskunde. We noemen er een paar:

- Een gedachtengang helder en beknopt kunnen formuleren.
- In een probleem hoofdzaken en bijzaken kunnen onderscheiden.
- Een bewijs duidelijk en overzichtelijk kunnen opschrijven.
- Kunnen lezen.
- Een formeel systeem kunnen hanteren.

Opgaven

1. Formuleer concrete doelstellingen die men nastreeft bij de behandeling van:
   - de toekenningsopdracht
   - de repetitie
   - functionele specificaties
   - de lineaire zoekmethode
   - de binaire zoekmethode

2. Bij het wiskundeonderwijs zijn de gewenste vaardigheden vastgelegd door exameneisen zoals:
   - de grafiek van $a \cos px + b \sin px$ kunnen schetsen.
   - kwadratische vergelijkingen $(ax^2 + bx + c = 0)$ kunnen oplossen.

   Geef voor het programmeeronderwijs soortgelijke eisen.

3. Welke algemenere doelstellingen passen bij een behandeling van de syntax van een programmeertaal ?

2.2 De begintoestand van de leerlingen.

Alvorens tot de invulling van een onderwijsprogramma over te gaan bespreken we de gewenste begintoestand van de leerlingen.

De gewenste voorkennis op het gebied van de informatica is genoemd in hoofdstuk 1:

- enige kennis van de opbouw en werking van een computer.
- enige ervaring met het laten uitvoeren van eenvoudige programma's door een computer.

Is deze kennis niet aanwezig dan begint men hiermee, zoals geschetst in hoofdstuk 1.

Bij de gewenste voorkennis op het gebied van de wiskunde ligt het minder eenvoudig. Hierbij speelt kennis niet zo'n grote rol maar bijvoorbeeld:

- het verschil weten tussen een definitie en een stelling.
- een bewijs netjes kunnen opschrijven.
- beschikken over rekenvaardigheid, zowel met getallen als met algebraïsche formules ("haakjes verdrijven", "ontbinden in factoren").
- een stelling kunnen toepassen.

Inzicht is veelal belangrijker dan technieken.

Gezien de rol van de predicatenrekening bij het programmeren zou men wellicht verwachten ook onderwerpen als verzamelingenleer en propositierekening aan te treffen. Het <u>gebruik</u> hiervan is in het huidige wiskundeonderwijs (nog) minimaal. In het programmeeronderwijs wordt de predicatenrekening als <u>gereedschap</u> ingezet. Het lijkt op dit moment verstandig om propositie- en predicatenrekening op te nemen in het programmeeronderwijs.

Opgaven.

1. Het maximum van twee getallen a en b, genoteerd als a $\underline{max}$ b, is gedefinieerd door

$$x = a \underline{max} b \quad \equiv \quad (x=a \vee x=b) \wedge x \geq a \wedge x \geq b$$

(i) Is dit een correcte definitie?

(ii) Bewijs de distributieregel $a + (b \underline{max} c) = (a+b) \underline{max} (a+c)$

(iii) Ga de volgende uitspraken na:

$\underline{max}$ is associatief

$\underline{max}$ distribueert over $\underline{max}$

$(-a) \underline{max} (-b) = -(a \underline{max} b)$

$x \geq a \wedge x \geq b \quad \equiv \quad x \geq a \underline{max} b$

$a = a \underline{max} b \quad \equiv \quad a \geq b$

$a - (b \underline{max} c) = (a-b) \underline{max} (a-c)$

(iv) Waarover wordt in (ii) en in (iii) "gequantificeerd"?

2. Wat heeft opgave 1 met deze paragraaf van doen?

3. Hoe is het te verklaren dat vrijwel geen enkele 16-jarige scholier in staat is om 381^4 foutloos te berekenen?

4. Los de volgende stelsels op: $\begin{cases} 2a + 3\alpha = 13 \\ 3a - \alpha = 3 \end{cases}$ $\begin{cases} \alpha + a = 5 \\ 2\alpha - a = 1 \end{cases}$

Waarom denkt menig leerling dat deze stelsels gelijkwaardig zijn?

5. Geef voorbeelden van formalismen in de middelbare school-wiskunde.

6. Op welke manier bewijst u in een HAVO-3 klas de stelling

"Een getal is deelbaar door 9 dan en slechts dan als de som van de cijfers van (de decimale reprensentatie van) dat getal deelbaar is door 9."

2.3 De opzet van een onderwijsprogramma.

Voor de indeling van de leerstof schetsen wij twee methoden.

De eerste methode behandelt diverse onderwerpen zo breed mogelijk en begint pas met een nieuw onderwerp indien de daarvoor noodzakelijke kennis en vaardigheden alle behandeld zijn. Laag voor laag wordt de kennis uitgebreid tot een hecht bouwwerk.

Als nadeel van deze methode wordt het feit genoemd, dat bij de behandeling van een laag de zinvolheid ervan niet duidelijk is ("waar is het goed voor").

De tweede methode behandelt problemen en stuit daarbij op een gemis aan houvast, aan onderliggende theorie. Waar nodig wordt deze theorie behandeld.

Nadeel van deze tweede methode is de vermenging van diverse op zich staande aspecten en langdurige onduidelijkheid over de spelregels. Regelmatig dient een en ander "op een rijtje gezet te worden".

Bij de eerste methode past een indeling als volgt:

0. Inleiding
1. Elementaire propositie- en predicatenrekening
2. Syntax en semantiek van programma's
   - 2.1 functionele specificaties
   - 2.2 de toekenningsopdracht
   - 2.3 de skip-opdracht
   - 2.4 de concatenatie
   - 2.5 de selectie
   - 2.6 de repetitie
3. Quantificaties, rekenregels
4. Het ontwerpen van algoritmen
   - 4.1 Standaard strategieën
   - 4.2 Niet-standaard strategieën
5. Varia.

Bij de tweede methode past bijvoorbeeld:

0. Inleiding
1. De structuur van een programma ; syntax
2. Programma's voorzien van asserties
3. Voorbeelden van het gebruik van de repetitie
   - 3.1 De betekenis van de variabelen ("de invariant")
   - 3.2 Eindigingsargumenten ("de variante functie")
4. Regels voor de asserties
   - 4.1 De skip-statement
   - 4.2 De assignment-statement
   - 4.3 De selectie
   - 4.4 De concatenatie
   - 4.5 De repetitie
5. Het rekenen met predicaten
6. Functionele specificaties ; $\underline{A}$, $\underline{E}$, $\underline{\subseteq}$, $\underline{N}$, $\underline{MAX}$ en $\underline{MIN}$.
7. Het ontwerpen van algoritmen
8. Manipuleren met arrays.

Vanzelfsprekend ligt er een heel gebied van methoden tussen de twee genoemde. De groep waaraan onderwijs gegeven wordt is mede bepalend voor een te kiezen aanpak.

Als afsluiting van deze paragraaf nog enkele opmerkingen:

- Vertel geen onwaarheden.
  Als een bepaald begrip te moeilijk is laat het dan achterwege of behandel het intuitief, maar geef er geen onjuiste definitie van (vergelijk de behandeling van het limietbegrip in hoofdstuk 2 van [2])

- Gebruik erg <u>eenvoudige</u> voorbeelden en opgaven.
  Doel van het onderwijs is niet een kennismaking met spectaculaire algoritmen en ingewikkelde datastructuren. Te vaak ziet men voorbeelden die het bevattingsbegrip van de leerlingen te boven gaan.

Opgaven.

1. Wat is het nut van de behandeling van de syntax van programmeertalen?

2. Wat is een programma-variabele?

3. Geef inhoud aan "Elementaire propositie- en predicatenrekening".

4. Geef voorbeelden van programma's waarmee de repetitie geïntroduceerd kan worden.

5. Formuleer de lineaire zoekmethode ("linear Search"). Is deze van toepassing op het probleem
   "bepaal van een gegeven niet-lege rij integers de kleinste index waarvoor de rij een maximum aanneemt"?
   Zo nee, waarom niet?

6. Is het van belang om bij het behandelen van $\underline{A}$, $\underline{E}$, $\underline{S}$ enz. de quantificaties over een leeg domein te vermelden? Zo ja, hoe denkt u deze uit te leggen?

2.4 Propositierekening en predicatenrekening.

In het huidige wiskundeonderwijs op de middelbare school wordt weinig aandacht besteed aan het onderwerp logica. Vanouds is logica een vak dat verbonden is met de grondslagen van de wiskunde.

Wel wordt enige aandacht besteed aan bewijzen. De daarbij gebruikte symbolen als $\longrightarrow$, $\Rightarrow$ of $\therefore$ worden echter niet als logische connectieven opgevat maar als afkorting voor zinsneden als "hieruit volgt" en "het geheel van bovenstaande samen levert". Ook symbolen als $\wedge$ en $\vee$ worden als afkorting gebruikt (Het is niet ongebruikelijk dat een T.H.-student opschrijft $x=3 \vee 4$).

Bij het programmeren is logica als gereedschap onontbeerlijk. Men heeft dit gereedschap nodig bij

- behandeling van Boolse expressies (waaronder true en false)
- het specificeren van een programmeerprobleem met behulp van predicaten
- het beschrijven van de semantiek van statements
- het leveren van bewijzen

Bij dit laatste, het leveren van een bewijs, doet zich nog het probleem voor dat het bewijs zelf een logische structuur heeft terwijl in het bewijs vrijwel altijd regels uit de logica gebruikt worden.

Teneinde logica als gereedschap te kunnen gebruiken zal hiermee geoefend dienen te worden. Evenals het mechanisch kunnen differentiëren van een functie als $f(x) = \frac{x}{x^2+1}$ dienen formules als

$p \wedge (p \vee q) \equiv p \vee (p \wedge q)$ afgeleid te kunnen worden.

Pas nadat enige vertrouwdheid met een calculus is verkregen, kan deze als gereedschap worden ingezet.

Een van de veelgemaakte fouten bij het onderwijs in de logica is de verkeerde keuze van voorbeelden. Zo tracht men de implicatie aan de hand van "als ... dan ..." voorbeelden duidelijk te maken, zoals in "als het regent dan kom ik niet".

Het juiste antwoord op de vraag "en als het dan niet regent?" is "als het niet regent, dan kom ik wel".

Zo is dat nu eenmaal bij het gebruik van de Nederlandse taal door Nederlanders.

(Overigens vindt men hetzelfde euvel bij het behandelen van verzamelingen, waar menig leraar niet schroomt te spreken over "de verzameling leerlingen die wiskunde leuk vinden".)

Wanneer om wille van de begripsvorming voorbeelden nodig zijn dan beperke men zich tot "bedachte objecten", zoals de natuurlijke getallen.
Zo zullen niet veel leerlingen uit "als $x > 3$ dan $x > 0$" concluderen "als $x \leq 3$ dan $x \leq 0$".

Het is niet goed om lang bij voorbeelden stil te staan: niet de voorbeelden zijn het houvast, maar de afspraken (rekenregels).

Omdat de symbolen $\rightarrow$, $\Rightarrow$ en $\Leftrightarrow$ meestal reeds in gebruik zijn bij wiskunde en andere vakken is het gewenst om voor het spel der logica andere symbolen te kiezen zoals bijvoorbeeld in [1]:

$\Rightarrow$ voor de implicatie
$\Leftrightarrow$ voor de equivalentie.

Bijzondere aandacht dient te worden besteed aan het zorgvuldig en overzichtelijk noteren van een bewijs.
Bij de propositierekening vermeldt men bij bewijsstappen welke regel is toegepast.
Zo is onderstaande een mogelijke afleiding van $(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$

$$
\begin{array}{ll}
 & p \Rightarrow q \\
= & \{ \text{definitie } \Rightarrow \} \\
 & \neg p \vee q \\
= & \{ \text{symmetrie } \vee \} \\
 & q \vee \neg p \\
= & \{ \text{dubbele ontkenning} \} \\
 & \neg\neg q \vee \neg p \\
= & \{ \text{definitie } \Rightarrow \} \\
 & \neg q \Rightarrow \neg p
\end{array}
$$

Bij de predicaten zijn sommige stappen op grond van logica en andere op grond van aritmetiek.
Een voorbeeld hiervan is het bewijs van

$$x \geq 0 \wedge y \geq 0 \Rightarrow x^2 + 2 \cdot y^2 - 2 \cdot x \cdot (y-1) \geq 0$$

Bewijs:

$$
\begin{array}{ll}
 & x^2 + 2 \cdot y^2 - 2 \cdot x \cdot (y-1) \geq 0 \\
= & \{ \text{algebra} \} \\
 & x^2 + y^2 - 2 \cdot x \cdot y + y^2 + 2 \cdot x \geq 0 \\
= & \{ \text{algebra} \} \\
 & (x-y)^2 + y^2 + 2 \cdot x \geq 0 \\
\Leftarrow & \{ \text{algebra} \} \\
 & (x-y)^2 \geq 0 \wedge y^2 \geq 0 \wedge 2 \cdot x \geq 0 \\
= & \{ \text{algebra} \} \\
 & \text{true} \wedge \text{true} \wedge x \geq 0 \\
= & \{ \text{logica} \} \\
 & x \geq 0 \\
\Leftarrow & \{ \text{logica} \} \\
 & x \geq 0 \wedge y \geq 0
\end{array}
$$

Hierbij vermeldt men desgewenst bij "algebra" en "logica" precieser waarop men zich beroept.

De behandeling van predicaten kàn worden uitgesteld tot de behandeling van (de semantiek van) statements, waarbij een verband gelegd wordt tussen toestanden en predicaten.

De (constante) predicaten false en true blijken moeilijk uit te leggen. Voorbeelden willen wel eens helpen. Zo valt in de tweedimensionale toestandsruimte, opgespannen door de integer variabelen x en y, het predicaat $x^2 \geq 0 \wedge y^2 \geq 0$ samen met true en $x^2 < 0 \wedge y^2 < 0$ samen met false.

De afspraak dat "Bewijs P" betekent "voor alle waarden van de in P voorkomende variabelen heeft P de waarde true" moet een keer expliciet genoemd worden. Een analogon voor "Weerleg P" eveneens.
(Het valt te overwegen om het predicaat true anders te noteren dan de waarde true, of om voor het predicaat zelfs een andere naam te kiezen. Dezelfde opmerking geldt voor false).

Gewend om te denken in termen van oorzaak en gevolg, hebben veel leerlingen moeite met de implicatie. Proposities als $2=3 \Rightarrow 6=8$ gaan wel, maar $2=3 \Rightarrow 6=6$ wil niet.
Strikt toepassen van de definitie "$p \Rightarrow q$ is hetzelfde als $\neg p \vee q$", is de enige oplossing van dit probleem.
Oefening baart kunst.

De begrippen "sterker" en "zwakker" alsook de termen "versterken" en "verzwakken" kunnen aan tal van voorbeelden worden geïllustreerd. De moeilijkheid van deze begrippen schuilt onder meer in het feit dat zij geen totale maar een partiële orde induceren. In de onderbouw is zo'n partiële orde aan bod gekomen bij de inclusie in de verzamelingleer.

Opgaven.

1. Geef overzichtelijke bewijzen c.q. afleidingen voor de volgende problemen.

   a. Bewijs dat voor alle gehele x geldt $x^2 + 3x + 3 \geq 1$

   b. Bereken de afgeleide van $x \to x \cdot \sin(x^2)$

   c. Bewijs $p \wedge (p \vee q) \equiv p \vee (p \wedge q)$

   d. Bewijs $1 + \tan^2 x = \frac{1}{\cos^2 x}$

   e. Bereken $\int_0^1 \frac{1}{1+\sqrt{x}}\, dx$

2. Maak een overzicht van stellingen uit de propositierekening en een collectie daarbij passende opgaven.

3. Wat is het nut van "waarheidstabellen" ?

4. Geef een (formele) syntax van "logische expressies".

5. De verzameling B bestaat uit 2 elementen, true en false. De operatoren $\wedge$ en $\vee$ zijn op B gedefinieerd door:

   Voor $p \in B$ is $p \wedge \text{true} = p$ $\quad p \vee \text{false} = p$
   $p \wedge \text{false} = \text{false}$ $\quad p \vee \text{true} = \text{true}$

   Geef uw commentaar op deze definities

6. Wat valt te zeggen van de synoniemen "waar" voor "true" en "onwaar" voor "false" ?
   En 0 voor false, 1 voor true ?

2.5 Syntax en semantiek van programma's.

2.5.1 Inleiding, functionele specificaties.

Als introductie op het beschrijven van statements en programma's passen

- voorbeelden van automaten met een beschrijving van de toestanden waarin deze zich kunnen bevinden.
- voorbeelden van algoritmen (in de betekenis van eenduidig vastgelegde voorschriften).

Bij deze voorbeelden worden de termen toestand, begintoestand, tussentoestand en eindtoestand ten tonele gevoerd.

Een berekening van een computer wordt afgeschilderd als een proces dat aanvangt in een begintoestand en leidt tot een eindtoestand.
De begintoestand is bepaald door de invoer en de eindtoestand legt de uitvoer vast.

> Bij een begin- en eindtoestand hoort een collectie programmavariabelen ("de coördinaten van de toestandsruimte")
> De invoer wordt weergegeven door het voorschrijven van de waarde van één of meer variabelen (de begintoestand) en de uitvoer wordt weergegeven door de waarde van één of meer variabelen (de eindtoestand).

Bij de behandeling van het bovenstaande komen zaken aan de orde die met een programmanotatie van doen hebben, zoals

> programmavariabelen, naam en type-aanduiding
> begintoestand en eindtoestand

Men leze hiervoor [1].

Op een of andere wijze wordt vastgelegd :

- de invoer : een collectie variabelen met type aanduiding en een preconditie

- de uitvoer : een collectie variabelen met type aanduiding en een postconditie

- een (naam van een) programma

Bijvoorbeeld 20 :

(i) Gegeven zijn integer variabelen x, y en z, met $x>0$, $y>0$ en $z>0$. Schrijf een programma S dat aan integer variabele r de waarde $max(x,y,z)$ toekent.

(ii) $\{ x>0 \wedge y>0 \wedge z>0 \}$
S
$\{ r = max(x,y,z) \}$

(iii)
```
const  x,y,z : integer ;
var    r     : integer ;
{ x>0 ∧ y>0 ∧ z>0 }
 S
{ r = max (x,y,z) }
```

Waarbij in (iii) met const wordt aangegeven dat S de waarden van x, y en z onverlet dient te laten en met var dat S aan r een waarde mag toekennen.
Zie ook het gedeelte over functionele specificaties in [1].

Merk op dat voor het specificeren van een programma dezelfde notatie kan worden gebruikt als die voor het definiëren van een statement.

Opgaven.

1. Geef voorbeelden van automaten met bijbehorende toestandsbeschrijvingen.

2. Geef een duidelijker specificatie (notatie) van de volgende opgaven.

   a. Bereken het maximum van twee getallen.
   b. Bereken het verschil van twee gehele getallen.
   c. Bereken de absolute waarde van een getal.
   d. Bepaal de wortel uit een gegeven getal.
   e. Bereken het snijpunt van 2 gegeven rechten.
   f. Bereken de afmeting van de kleinste rechthoek om een gegeven ruit.
   g. Bepaal of een gegeven getal priem is.
   h. Bepaal de middelste van 3 gegeven getallen.
   i. Bereken of een serie getallen geordend is naar grootte.
   j. Bereken de oppervlakte van een gegeven rechthoek.

3. Gegeven zijn 4 punten X, Y, Z, P in het platte vlak. Geen drie ervan liggen op één rechte. De punten zijn gegeven door hun (geheeltallige) coördinaten.
   Schrijf een programma dat bepaalt of P binnen dan wel buiten ΔXYZ ligt.

   a. Geef commentaar op deze opgave.
   b. Geef een specificatie met pre- en postconditie.
   c. Maak de opgave.

4. Een permutatie van de getallen 1 tot en met N is een rangschikking van deze getallen.
   Zo worden de permutaties van de getallen 1 t/m 3 gegeven door 123, 132, 213, 231, 312, 321. Aldus opgesomd staan zij lexicografisch geordend: als getal gelezen staan ze in stijgende volgorde.

a. Geef de permutaties van de getallen 1 t/m 4 in lexicografische volgorde.

b. De lexicografische opvolger van 6 1 3 2 5 7 4 is 6 1 3 2 7 4 5.
Ga dit na.

c. Bepaal de lexicografische opvolgers van 1 2 3 5 4 6 7 en van 1 6 7 3 5 4 2.

d. Geef een voorschrift waarmee men van een gegeven permutatie van de getallen 1 tot en met 7 de lexicografische opvolger kan vinden.
Test het voorschrift met: 1 2 3 4 5 6 7,
5 7 6 4 3 2 1 en
7 6 5 4 3 2 1.

5. Verklaar het nut van opgave 4.

6. Met $\{P\}\ S\ \{Q\}$ bedoelen we:

Uitvoering van S in een toestand die aan P voldoet eindigt gegarandeerd in een toestand die aan Q voldoet.

Geef commentaar op de volgende beweringen.

a. Als $\{P_0\}\ S\ \{Q_0\}$ en $\{P_1\}\ S\ \{Q_1\}$ dan ook:
$\{P_0 \wedge P_1\}\ S\ \{Q_0 \wedge Q_1\}$
en $\{P_0 \vee P_1\}\ S\ \{Q_0 \vee Q_1\}$.

b. Als $\{P\}\ S\ \{Q \vee R\}$ dan $\{P\}\ S\ \{Q\}$ of $\{P\}\ S\ \{R\}$.

c. Voor alle Q geldt $\{\text{false}\}\ S\ \{Q\}$.

d. Voor alle P geldt $\{P\}\ S\ \{\text{true}\}$.

2.5.2 De toekenningsopdracht.

De toekenningsopdracht is een fundamentele bouwsteen. Met deze statement is het mogelijk de toestand van de automaat te wijzigen. Het is van groot belang dat de betekenis ervan duidelijk is.
Veel voorkomende fouten bij de introductie van de toekenningsopdracht zijn uitleggingen als

" Door $x := E$ wordt x gelijk aan E "

" Na $x := E$ heeft x de waarde E "

" Na $x := E$ geldt $x = E$ " (Een erg elementaire blunder)

" Met behulp van de toekenningsopdracht kun je aan een variabele een waarde geven ; zo geldt na $x := 3$ dat $x = 3$ ".

Bij de uitleg van de toekenning dienen naast voorbeelden als $x := y$ en $x := 3$ ook $x := x + 1$ en $x := -x$ aan bod te komen. Het beste geeft men, als introductie op een formelere definitie, bij de voorbeelden ook asserties.
Bijvoorbeeld :

$\{x = 0\}\ \ x := 3\ \ \{x = 3\}$
$\{x = 3\}\ \ x := x + 1\ \ \{x = 4\}$
$\{x \geq 7\}\ \ x := 2 * x\ \ \{x \geq 14\}$
$\{x > 5\}\ \ x := 2 * x + 1\ \ \{x > 11\}$
$\{x = 2 \wedge y = 3\}\ \ x := x + y\ \ \{x = 5\}$

De "regel" van de toekenning laat men het beste ontdekken door de klas met achtereenvolgens bijvoorbeeld te vragen:

Wat moet er 'tenminste' op de stippeltjes staan:

$\{\ \ldots\ \}\ \ x := x + 1\ \ \{x > 16\}$
$\{\ \ldots\ \}\ \ x := x + 1\ \ \{x = 7\}$
$\{\ \ldots\ \}\ \ x := x + 1\ \ \{x^2 > 30\}$
$\{\ \ldots\ \}\ \ x := x + 1\ \ \{x^3 + 3x > 46\}$

En: $\{ \ldots \}\quad x := x+1\quad \{ x \geq 16 \}$

$\{ \ldots \}\quad x := x * x\quad \{ x \geq 16 \}$

$\{ \ldots \}\quad x := x*x - 3*x\quad \{ x \geq 16 \}$

$\{ \ldots \}\quad x := x*x*x - 6*x*x + 2\quad \{ x \geq 16 \}$

Daarnaast later de wat moeilijker

$\{ \ldots \}\quad x := 3\quad \{ x = 3 \}$

$\{ \ldots \}\quad x := 3\quad \{ x = 4 \}$

$\{ \ldots \}\quad x := y\quad \{ y > 0 \}$

en vragen naar de juistheid van bijvoorbeeld

$\{ x > 1 \}\quad x := x * x\quad \{ x > 1 \}$

$\{ x \geq 0 \}\quad x := x + x\quad \{ x > 0 \}$

$\{ x > 0 \wedge y > 0 \}\quad y := x + y\quad \{ x > 0 \wedge y > 0 \}$

Na voldoende veel voorbeelden zal men ontdekt hebben dat bijvoorbeeld $\{ \ldots \}\ x := E\ \{ x > 0 \}$ correct is indien op de plaats van de puntjes $E > 0$ staat.

En soortgelijk Q^x_{x+1} bij $\{ \ldots \}\ x := x+1\ \{ Q \}$, waarbij uiteraard de notatie Q^x_{x+1} ten tonele is gevoerd.

De regel $\{ Q^x_E \}\ x := E\ \{ Q \}$, of zo mogelijk de regel "$\{ P \}\ x := E\ \{ Q \}$ betekent $P \Rightarrow Q^x_E$", vormt de explicitering van de opgedane ervaring.

Opgaven

1. Maak een serie lessen met opgaven over de toekenningsopdracht.

De concatenatie.

In de meeste leerboeken wordt de ; beschreven als scheidingsymbool.
"een programma bestaat uit statements gescheiden door punt-komma's".
Of: "na elk statement komt een puntkomma met uitzonderingen:

voor end hoeft geen puntkomma,
voor else mag geen puntkomma,
na de laatste end komt een punt."

De puntkomma dient niet als scheider van statements maar als plakker van statements. Met behulp van de puntkomma worden statements aaneengeregen.
Het is te betreuren dat de puntkomma als scheidingsymbool bij declaraties wordt gebruikt. Beter kieze men daar een ander symbool, bijvoorbeeld |.

De uitleg van de concatenatie schept niet veel problemen. Bekende voorbeelden zijn:

- $\{x=X \wedge y=Y\}$ $x := x+y$; $y := x-y$; $x := x-y$ $\{x=Y \wedge y=X\}$
- $\{x=X \wedge y=Y\}$ $x := x-y$; $y := x+2*y$ $\{x=X-Y \wedge y=X+Y\}$
- $\{x=X \wedge y=Y\}$ $z := x$; $x := y$; $y := z$ $\{x=Y \wedge y=X\}$

En daarnaast uiteraard de onjuiste

$\{x=X \wedge y=Y\}$ $x := y$; $y := x$ $\{x=Y \wedge y=X\}$
$\{x=X \wedge y=Y\}$ $x := x-y$; $y := x+y$ $\{x=X-Y \wedge y=X+Y\}$
$\{x=X\}$ $x := x*x$; $x := x*x$ $\{x=X^3\}$

Opgave:
Bedenk ten minste 20 opgaven bij de concatenatie.

4 De selectie en het skip statement.

Eenvoudige voorbeelden ter illustratie van de selectie zijn de berekening van het maximum van twee getallen en de berekening van de absolute waarde van een getal.

De constructie van het statement **if** B **then** S_0 **else** S_1 is bijzonder ongelukkig om de volgende redenen.

(i) berekening van een preconditie bij gegeven postconditie en een gekozen statement levert een nevenconditie: de guard. Andere keuzen van het statement leveren (meestal) andere guards. Zo vindt men een structuur als

$\{P\}$ **if** B_0 **then** $\{P \wedge B_0\}\ S_0\ \{Q\}$
B_1 **then** $\{P \wedge B_1\}\ S_1\ \{Q\}$
B_2 **then** $\{P \wedge B_2\}\ S_2\ \{Q\}$
⋮
B_n **then** $\{P \wedge B_n\}\ S_n\ \{Q\}$
$\{Q\}$

In PASCAL is de keuze B, $\neg B$ opgelegd, waardoor het onoverzichtelijke

$\{P\}$ **if** B_0 **then** $\{P \wedge B_0\}\ S_0\ \{Q\}$
else $\{P \wedge \neg B_0\}$ **if** B_1 **then** $\{P \wedge \neg B_0 \wedge B_1\}\ S_1\ \{Q\}$
else $\{P \wedge \neg B_0 \wedge \neg B_1\}$ **if** ....

ontstaat.

(ii) De keuze tussen B en $\neg B$ leidt tot symmetrieverstoring, met als gevolg meer bewijsverplichtingen.
Zo volgt uit $\{x \geq y\}\ m := x\ \{m = max(x,y)\}$ door naamswisseling
$\{y \geq x\}\ m := y\ \{m = max(y,x)\}$
, maar het fraaie programma

$\{true\}$ if $x \geq y \rightarrow m := x$
[] $y \geq x \rightarrow m := y$
fi
$\{m = max(x,y)\}$

is in PASCAL niet mogelijk.

(iii) De keuze tussen B en $\neg B$ baart non-determinisme uit, met als gevolg dat vroegtijdig beslissingen moeten worden genomen.
Bij het programma if $B_0 \rightarrow S_0$ [] $B_1 \rightarrow S_1$ fi kan met het oog op efficiëntie de guard B_1 versterkt worden tot $B_1 \wedge \neg B_0$, een beslissing die los staat van correctheid en daarvan gescheiden wordt genomen.

(iv) Het ontbreken van een "afsluiter" als fi noodzaakt het gebruik van haakjes begin end. Bij de behandeling van de selectie komt dan ook het "compound statement" om de hoek kijken. Dit verstoort de uitleg.

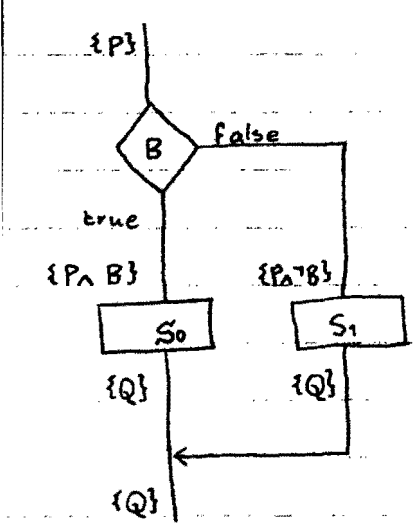
(v) De constructie if B then S laat zich slecht annoteren. De bewijsverplichting $P \wedge \neg B \Rightarrow Q$ voor $\{P\}$ if B then S $\{Q\}$ wordt gemakkelijk vergeten.

De beste remedie voor deze kwalen is PASCAL niet gebruiken maar bijvoorbeeld de "guarded command language" uit [1].
Wenst men PASCAL te handhaven dan is het euvel genoemd onder (v) op te lossen door de introductie van skip.
Annotatie wordt

{P} if B then {P∧B} S {Q}
else {P∧¬B} skip {Q}
{Q}.

De PASCAL conventie om skip te noteren door niets te noteren wordt in [1] vergeleken met de afspraak om nul te noteren door niets op te schrijven.

In het klassieke programmeeronderwijs gebruikt(e) men plaatjes als "Nassi-Schneidermann diagrammen" en "stroomdiagrammen".
In het gunstigste geval ziet zo'n plaatje er als volgt uit:



Veel duidelijker is annotatie.
Overigens laten veel programma's zich niet makkelijk op één A4 tekenen (in zo'n plaatje).

{P} B then S0 else S1

Opgaven.

Schrijf een les (of serie lessen) waarin u de selectie uitlegt.

Geef commentaar op de volgende opgaven:

(i) Vier punten X, Y, P en Q in het platte vlak zijn gegeven door hun geheeltallige coördinaten.
Schrijf een programma dat berekent of P en Q aan weerszijden van de lijn door X en Y liggen.

(ii) Gegeven zijn de lijnen l en m in het platte vlak. Schrijf een programma dat aan de boolean variabele c de waarde "l snijdt m" toekent.

Maak de opgaven uit opgave 2.

Maak een collectie van ten minste 20 opgaven die kunnen worden opgelost met gebruik van skip, assignment, selectie en concatenatie.

Maak een grammatica voor boolse expressies analoog aan [1] en een collectie daarbij passende opgaven.

Bestudeer de paragraaf "De alternatieve statement" uit [1].

Bestudeer opgaven 18 tot 26 uit [1], "Functionele specificaties en bewijsverplichtingen".

De repetitie.

Een operationele uitleg van **while** B **do** S in termen van "zolang B doe S" is niet moeilijk. Een uitleg waarbij het begrip invariant wordt gebruikt daarentegen wel.
De leerling is gewend om bij observaties te kijken naar wat verandert en niet naar wat invariant blijft.

Bij het plausibel maken van de "regel van de repetitie" laat men in eerste instantie eindiging buiten beschouwing.
Klassiek voorbeeld voor de introductie van het begrip invariant is de berekening van de grootste gemene deler van twee positieve gehele getallen.
Andere voorbeelden zijn

- de berekening van het product van 2 natuurlijke getallen zonder gebruik te maken van *.
- machtsverheffing.

Bij de gegeven voorbeelden komt steeds duidelijk naar voren:

$$P \wedge \neg B \Rightarrow R$$
$$\{P \wedge B\}\ S\ \{P\}$$

Na de voorbeelden komt de term "invariant" aan de orde en worden de bovenstaande bewijsregels expliciet gemaakt.

Hoe een geschikte invariant te vinden is een zaak voor later.
Eindigingsargumenten zullen in het begin steeds informeel zijn.

De constructie **while** B **do** S van PASCAL vertoont dezelfde gebreken als die van de selectie.
Vergelijk bijvoorbeeld de ggd-algoritme:

guarded-command language

```
do  x>y  →  x:=x-y
 ▯  y>x  →  y:=y-x
od
```

PASCAL

```
while  x≠y  do
if  x>y  then  x:=x-y  else  y:=y-x
```

of het sorteren van a,b,c en d in opklimmende volgorde:

guarded command language

```
do  a>b  →  a,b := b,a
 ▯  b>c  →  b,c := c,b
 ▯  c>d  →  c,d := d,c
od
```

PASCAL

```
while (a>b) or (b>c) or (c>d) do
if a>b then begin h:=a ; a:=b ; b:=h end else
if b>c then begin h:=b ; b:=c ; c:=h end else
if c>d then begin h:=c ; c:=d ; d:=h end
```

Opgaven.

a. Is het mogelijk om met een even aantal paardesprongen vanuit een hoekpunt van een schaakbord elk van de andere hoekpunten te bereiken?

b. Bewijs dat op een $N \times N$ schaakbord elk hoekpunt vanuit elk hoekpunt te bereiken is met een aantal paardesprongen dat een drievoud is. ($N \geq 3$).

Wat heeft opgave 1 met de repetitie van doen?
Bedenk analoge opgaven.

Bepaal P zodanig dat voldaan is aan

a. $\{P\}$ while $x \neq 0$ do $x := x+1$ $\{x=0\}$

b. $\{P\}$ while $x<0$ do $x := x+1$ $\{x=0\}$

c. $\{P\}$ while $x<0$ do $x := x+1$ $\{x \geq 0\}$

d. $\{P\}$ while $x \neq 0$ do $x := x+1$ $\{x \geq 0\}$

(Probeer P zo zwak mogelijk te kiezen).

Bedenk een twintigtal opgaven bij de repetitie (géén programmeeropgaven).

Schrijf een les (of serie lessen) waarin u de repetitie behandelt.

Bestudeer opgave 13 en de uitwerking ervan in [1]

Geef een definitie van "de grootste gemeenschappelijke deler van twee positieve gehele getallen".

Bereken gehele a en b waarvoor $84a + 128b = 4$.

Breid het programma

$\{X>0 \wedge Y>0\}$ $x,y := X,Y$; do $x>y \rightarrow x:=x-y \;\square\; y>x \rightarrow y:=y-x$ od

$\{x=y=ggd(X,Y)\}$

zodanig uit dat na afloop tevens voldaan is aan $a\cdot X + b\cdot Y = ggd(X,Y)$.

In vrijwel elk boek over PASCAL staat bij de beschrijving van de repetitie een opmerking in de trant van

"Indien de boolse expressie initieel false is wordt de statement niet uitgevoerd".

Wat is uw commentaar hierop?

Evenzo voor

"The user is responsible for guaranteeing that the conditions for termination are eventually met".

Kunt u een bewijsregel geven voor

$\{P\}$ repeat S until B $\{Q\}$?

Kunt u voorbeelden geven waarbij het gebruik van deze statement voordelen heeft boven de while constructie?

2.6 Het ontwerpen van programma's.

2.6.1 Inleiding

Nadat de betekenis van de bouwstenen van de programmeertaal is uitgelegd komen we toe aan het uiteindelijke doel: het ontwerpen van algoritmen. Bij een inleiding in het programmeren (en daarna veelal ook) komt dit neer op het bedenken van geschikte invarianten.
In het begin, aansluitend op de behandeling van de repetitie, laten we de leerlingen programma's ontwikkelen waarbij de invariant gegeven is. Voorbeelden hiervoor zijn:

- berekening van quotient en rest zonder div en mod.
- berekening van de naar beneden afgeronde waarde van $\sqrt{N}$ $(N \geq 0)$.
- berekening van de naar beneden afgeronde waarde van ${}^{3}\log N$ $(N \geq 1)$.
- berekening van x^N $(N \geq 0,\ x \geq 1)$.
- berekening van de som van de cijfers (van de decimale representatie) van N $(N \geq 0)$.

Bij de behandeling van deze opgaven wordt verteld hoe de gegeven invariant uit de eindconditie was afgeleid.
(De gegeven voorbeelden leveren invarianten door "een conjunct weglaten", "een constante door een variabele vervangen" en "staartrecursie").

Het beoefenen van de diverse standaardtechnieken wordt zonder geschikte notaties al snel moeizaam. Bij de lineaire zoekmethode is reeds sprake van universele quantificatie. We besteden de volgende paragraaf aan dit onderwerp.

2.6.2 Arrays en quantificaties.

Met de toevoeging van het array ontstaat de mogelijkheid om programma's te schrijven die betrekking hebben op een willekeurige (eindige) hoeveelheid gegevens.

Het array wordt geïntroduceerd als rij van variabelen. Daarbij komen begrippen als definitiegebied (domein, range) en index (subscript, argument) aan de orde.
Afhankelijk van het niveau van de klas kan het array geïntroduceerd worden als functie op een eindig aaneengesloten deel van de gehele getallen.

In PASCAL heeft de declaratie van array X met domein $i: 0 \leq i < 10$, type integer, de volgende vorm:

```
X : array [0..9] of integer
```

Wij prefereren:

```
X(i: 0≤i<10) : array of int
```

om onder meer de volgende redenen:

- Voor $p \leq q$ heeft X(i: $p \leq i < q$) precies $q-p$ elementen, waar X(i: $p \leq i \leq q$) juist $q-p+1$ elementen bevat

- Veel programma's hebben de volgende structuur bij array X(i: 0≤i<N)

```
"Initialiseer voor lege rij"
; n:=0
; while  n ≠ N  do
  begin
     {" Voor  X(i: 0≤i<n) is het resultaat bewerkstelligd"}
      " betrek  X(n) in de berekening"
     {" Voor  X(i: 0≤i<n+1) is het resultaat bewerkstelligd "}
      ; n:=n+1
  end
```

Hierbij is n:=n+1 veelal opgelegd door eindigingseisen. De variabele n geeft het aantal reeds beschouwde elementen aan en X(n) is het eerst te beschouwen element.

Na het maken van een aantal voorbeelden waarmee het werken met arrays wordt toegelicht, ontstaat al snel de behoefte aan geschikte notaties. Behalve de compactheid die daarmee wordt verkregen, geeft een goede notatie de mogelijkheid om invarianten te bedenken en om een calculus te ontwikkelen.

Het aanleren van nieuwe notaties is een moeizaam proces en vergt veel oefening.

Het onderscheid tussen vrije en gebonden variabelen (programma-variabelen en dummies) dient te worden benadrukt. Een argumentatie van

{ X(i: 0≤i<N) : array of int , N≥0 } s:=0
;i:=0 ; while i≠N do begin s:=s+X(i) ; i:=i+1 end
{ s = (Σ i : 0≤i<N : X(i)) }

is alleen mogelijk indien de dummy i hernoemd wordt.
Een belangrijke regel is dan ook:

gebruik voor een dummy nooit de naam van een programmavariabele.

Het goede voorbeeld van de docent bij de keuze van geschikte namen voor programmavariabelen en voor dummies is van groot belang.

Bij het gebruik van formalismen dreigt (in het algemeen) het gevaar van overdrijven. Men make van het middel geen doel: kies de formele weg indien het helpt.

Opgaven

1. Maak een les waarin het array geïntroduceerd wordt.

2. Geef een serie opgaven die alle betrekking hebben op tellen, zoals bijvoorbeeld

   " Gegeven zijn een integer N, N≥1, en een integer array X(i: 0≤i<N). Schrijf een programma ter berekening van het aantal indices i, 0<i<N, waarvoor X(i-1) ≤ X(i) ".

3. Maak een les waarin u de universele en de existentiële quantificatie uitlegt.

4. Bedenk een serie opgaven die alle betrekking hebben op
   a. universele quantificatie
   b. existentiële quantificatie

5. In het volgende staan expressies die berekend dienen te worden bij gegeven integer N, N≥3, en gegeven integer array X(i: 0≤i<N). Geef deze opgaven weer in goed Nederlands. Geef ook commentaar op de opgaven (als programmeeropgave)

   a. (N i,j : 0≤i≤j<N : X(i) = X(j))
   b. (N i : 0≤i<N : X(i) = X(N-1-i))
   c. (A i : 0<i<N : X(i) = X(i-1))
   d. (MAX i : 0≤i<N : X(i))
   e. (E i : 0<i<N : X(i) = X(i-1))
   f. (MIN i : 0≤i<N ∧ X(i) = X(N-1) : i)
   g. (MAX i : 0≤i<N ∧ X(i) = X(0) : i)

h. $(\underline{MAX}\ i,j : 0 \le i < j < N : X(i) + X(j))$

i. $(\underline{S}\ i : 0 \le i^3 < N : X(i^3))$

j. $(\underline{A}\ i,j : 0 \le i \le j < N : X(i) \neq X(j))$

k. $(\underline{N}\ i,j : 0 \le i < j < N : X(j) > X(i))$

l. $(\underline{E}\ i,j : 0 \le i \le j < N : X(j) - X(i) = 47)$

m. $(\underline{MAX}\ i,j : 0 \le i \le j \le N : (\underline{S}k : i \le k < j : X(k)))$

n. $(\underline{MIN}\ i,j,k : 0 \le i < j < k < N : X(i) + X(j) - X(k))$

o. $(\underline{N}\ i,j : 0 \le i \le j < N : X(i)$ is even $\wedge$ $X(j)$ is oneven$)$

p. $(\underline{S}\ i : 0 \le i < N : X(i) \cdot 2^i)$

q. $(\underline{A}\ i : 0 \le i < N : X(i) = 47)$

r. $(\underline{S}\ i,j : 0 \le i < j < N : X(i) + X(j))$

s. $(\underline{E}\ n : 0 \le n \le N : (\underline{A}\ i : 0 < i < n : X(i-1) \le X(i)) \wedge (\underline{A}\ i : n < i < N : X(i-1) \ge X(i)))$

t. $(\underline{N}\ i : 0 \le i < N : X(i) = i)$

u. $(\underline{MAX}\ i : 0 \le i < N : |X(i) - i|)$

v. $(\underline{A}\ i,j : 0 \le i \le j < N : X(i) \le X(j) + 1)$

w. $(\underline{A}\ i,j : 0 \le i \le j < N : X(i) \cdot X(j) \ge 0)$

x. $(\underline{N}\ i : 0 \le i < N :$ $X(i)$ is een drievoud$)$

y. $(\underline{GGD}\ i : 0 \le i < N : X(i))$

z. $(\underline{N}\ p :$ p priem$: (\underline{E}\ i : 0 \le i < N : X(i)$ is een p-voud $\wedge\ X(i) \neq 0))$.

6. Bedenk 10 opgaven bij de lineaire zoekmethode.

7. Maak een les waarin u een algoritme behandelt dat bij twee gesorteerde rijen het aantal gemeenschappelijke elementen berekent.

Didactische aanwijzingen.

Menig lezer van het voorafgaande heeft zich afgevraagd in hoeverre het gestelde haalbaar is.
In een 6-VWO-klas met leerlingen die een "exact pakket" gekozen hebben is het mogelijk om propositie- en predicatenrekening netjes te behandelen. Ook regels als

" $\{P\}\ x:=E\ \{Q\}$ betekent $P \Rightarrow Q^x_E$ "

en

" $\{P\}$ if B then S_0 else S_1 $\{Q\}$ betekent $\{P \wedge B\}\ S_0\ \{Q\}$ en $\{P \wedge \neg B\}\ S_1\ \{Q\}$ "

kunnen daar ten tonele worden gevoerd.

In een 5-HAVO-klas zullen de bovengenoemde regels voorlopig mysterieus blijven. Zo goed als de regel

" $\lim_{x \to a} f(x) = \ell$ betekent $(\underline{A}\,\varepsilon : \varepsilon > 0 : (\underline{E}\,\delta : \delta > 0 : (\underline{A}\,x : 0 < |x-a| < \delta : |f(x)-\ell| < \varepsilon)))$ ".

niet begrepen wordt.
Vanuit een formeel standpunt heeft het weinig zin om over de afgeleide van een functie te beginnen indien het limietbegrip niet duidelijk is. In de praktijk valt dit wel mee, mits men zich beperkt tot problemen waar geen beroep gedaan wordt op het begrip limiet.
Zo is het ook bij programmeeronderwijs. De opmerking " als $x \geq 1$ dan is $x+1 \geq 2$ " rechtvaardigt $\{x \geq 1\}\ x := x+1\ \{x \geq 2\}$ en met dergelijke rechtvaardigingen komt men een heel eind, mits men zich beperkt tot problemen waar geen beroep gedaan hoeft te worden op de precieze definitie van $x := E$.

Overigens verwachten wij dat het onderzoek in het vakgebied Informatica zoveel invloed zal hebben op de wiskunde dat in de verdere toekomst het wiskundeonderwijs een basis zal hebben gelegd voor het programmeeronderwijs.

In een 4-MAVO-klas zal het programmeeronderwijs nog weer anders zijn. Het zal zich beperken tot een kleine klasse problemen. Bijvoorbeeld bij de repetitie tot de "linear search" en "eenvoudige telproblemen". (Een "eenvoudig telprobleem" is de berekening van $(\underline{N}\, i : 0 \leq i < N : B(X(i)))$ waarbij $X(i : 0 \leq i < N)$ een integer array en $B(y)$ een (voor integer y) toelaatbare boolse expressie is.)
De (te abstracte) generalisatie van de klasse problemen wordt uiteraard niet expliciet gemaakt.

We herhalen in dit verband nog eens de regel:

Vertel liever niets dan iets dat onjuist is.

Een andere regel die bij het bovenstaande past is de volgende:

Geef alleen opgaven die met de verstrekte kennis kunnen worden opgelost

Als besluit van dit hoofdstuk nog enkele algemene aanwijzingen.

- Besteed geruime tijd aan één probleem.

  Het is niet zinvol om een grote collectie "programmeersommen" in één lesuur te behandelen. Beter is het om één probleem tot in alle details te behandelen. Men overtuige zich ervan dat eenieder de daarbij optredende moeilijkheden en daarvoor gegeven oplossingen begrijpt.

- Werk langzaam en netjes.

  Kladwerk is geen werk. Zorg voor een juiste bordindeling en schrijf zo netjes als u kunt.
  Gebruik een gedeelte van het bord voor de programmatekst, een gedeelte voor de daarbij behorende argumentatie en een gedeelte voor details of voorbeelden.
  Jedere programmeeropgave en ieder voorbeeld vergt voorbereiding!

- Leer de leerlingen aantekeningen maken.

  Het beste verbiedt men de leerlingen om tijdens de uitleg aantekeningen te maken. Hierbij zorgt u er uiteraard voor dat het laatste kwartier gebruikt kan worden om een en ander van het bord over te nemen. Bij deze periode van overschrijven geeft u acht op de zorgvuldigheid waarmee dit gebeurt.

Geef duidelijk aan langs welke wegen oplossingen verkregen zijn.
Het doel is niet de oplossing, het algoritme, maar de methode waarmee de oplossing is verkregen.
Resumeer na de behandeling van een opgave hóe de oplossing tot stand kwam.

ogaven.

1. Wat dunkt u van het laten voormaken van opgaven door leerlingen.

2. Na een aantal inleidende lessen over de repetitie komt de volgende explicitering:

Bij een repetitie hoort een invariant, een relatie tussen de variabelen die lijkt op het gewenste resultaat.
Om dat gewenste eindresultaat te bereiken worden statements bedacht, waardoor de invariant in het algemeen wordt verstoord. Je moet er dan voor zorgen dat de invariant weer wordt hersteld.
Geef hierop uw commentaar.

3. Is het duidelijk dat $w := w/2$ de relatie $q = w.a + p$ verstoort en dat $a := 2*a$ deze relatie weer herstelt?