Programmeren

Nascholingscursus Informatica

voor leraren

Programmeren

A. Kaldewaij

Nascholingscursus Informatica

voor leraren.

Programmeren.

drs. A. Kaldewaij

Onderafdeling der Wiskunde en Informatica

Technische Hogeschool Eindhoven

juli 1983.

Inhoud.

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 onderwijzen van informatica worden de komende jaren diverse zogeheten "nascholingscursussen voor leraren" georganiseerd. De onderhavige cursus is zo'n nascholingscursus. Hierin worden de basisbegrippen en basistechnieken van het programmeren behandeld.

Tot in de zeventiger jaren (en op veel plaatsen nu nog) bestaat het leren programmeren uit het leren van een programmeertaal, veelal aan de hand van voorbeelden. Het effect van de uitvoering van een programma wordt operationeel beschreven, in termen van de werking van een computer.
Ook de ontwikkeling van een programma gebeurt in operationele termen. Met behulp van testen worden "logische fouten" opgespoord en verbeterd tot een uiteindelijk bevredigend programma is ontstaan. Hulpmiddelen zijn blokschema's, beslissingstabellen, checkout-compilers etc.. Het spreken over programma's is antropomorf; "nu doet hij dit", "nu wacht hij tot je iets intypt".

Langzamerhand ontstaat het besef dat men, in plaats van het testen van programma's tot er (hopelijk) geen fouten meer in zitten, er beter aan doet van programma's aan te tonen dat zij gewenste eigenschappen bezitten. Temeer daar met testen wel de aanwezigheid maar niet de afwezigheid van fouten aangetoond kan worden.

In deze cursus wordt geabstraheerd van de uitvoering van een programma door een rekenautomaat. Taalconstructies worden zo gedefinieerd dat gewenste eigenschappen van daaruit opgebouwde programma's daadwerkelijk bewezen kunnen worden. Meer nog: bewijsregels vormen een leidraad bij het ontwerpen van programma's.

In de cursus wordt gebruik gemaakt van de programmeertaal Pascal. Van deze taal worden alleen de belangrijkste elementen gebruikt. Het wel: dit is een cursus programmeren en geen cursus Pascal. De keuze Pascal is genomen op de volgende gronden.

- Pascalstatements laten zich goed beschrijven.
- De apparatuur waarmee scholen behept zijn (worden), wordt meestal geleverd met een implementatie van Pascal.

De nadelen van Pascal worden in de cursus duidelijk.

De didactiek van het programmeren komt niet expliciet aan de orde maar is impliciet in de aangeboden stof verwerkt.

De cursus en deze handreiking zijn geïnspireerd door het boek "A discipline of programming" van prof. dr. Edsger W. Dijkstra. De opgaven zijn bijna alle afkomstig van ir. W.H.J. Feijen.

Aanbevolen litteratuur betreffende programmeren.

[0] A discipline of programming.
Edsger W. Dijkstra, Prentice-Hall.

Een mooi en moeilijk boek met een schat aan zorgvuldig uitgewerkte voorbeelden. De door de auteur ontworpen "guarded command language" wordt gebruikt. De betekenis van statements wordt gedefinieerd met behulp van weakest preconditions.

[1] Dictaat bij het college Inleiding Programmeren.
Prof. dr. Edsger W. Dijkstra, THE dictaat 2.308.

Een goede inleiding voor [0] en [2]. Er wordt gebruik gemaakt van guarded commands.

[2] The Science of Programming.
David Gries, Springer-Verlag.

Een uitstekend leerboek. Een goede inleiding voor [0].

[3] The design of well-structured and correct programs.
Suad Alagić en Michael Arbib, Springer-Verlag.

Een uitgebreide behandeling van bewijsregels voor Pascal. Meer nadruk op correctheidsbewijzen dan op programmaontwikkeling.

Aanbevolen litteratuur betreffende Pascal.

[4] Pascal, User Manual and Report.
Kathleen Jensen & Niklaus Wirth, Springer Verlag

Bevat een complete beschrijving van Pascal.

[5] Systematic Programming : An Introduction.
N. Wirth, Prentice Hall.

[6] Algorithms + Datastructures = Programs.
N. Wirth, Prentice Hall.

Een "standaardwerk" op het gebied van Pascal.

Voor hun opbouwende kritiek en waardevolle suggesties bedank ik met name ir. W.H.J. Feijen, ir. J.L.A. van de Snepscheut en ir. J.T. Udding,

drs. A. Kaldewaij
juli 1983

0. Inleiding.

In deze inleiding wordt een indruk gegeven van het soort problemen waarmee we ons gaan bezighouden. Dit gebeurt aan de hand van een in de programmeertaal Pascal geschreven programma. In de volgende hoofdstukken wordt uitgebreid ingegaan op de betekenis van dergelijke programma's. We volstaan hier met een korte informele toelichting, op een manier die geheel anders is dan de wijze waarop we later over een programma zullen spreken.
Het programma ziet er als volgt uit:

```
program maximum (input,output);
var
   a,b,c : integer ;
begin
   read (a) ;
   read (b) ;
   if a>b then c:=a else c:=b ;
   write (c)
end.
```

Toelichting.

De naam van het programma is maximum.

De namen input en output komen van het volgende:
veelal is voor de verwerking van een programma invoer vanuit de buitenwereld nodig en wordt uitvoer aan de buitenwereld geleverd. Deze invoer en uitvoer geschieden via media (bijvoorbeeld een toetsenbord en een beeldscherm), waaraan de respectieve namen input en output gekoppeld zijn.

De programmavariabelen van dit programma zijn a, b en c. Hun type is integer: zij kunnen alleen gehele waarden aannemen.

Tussen begin en end staan statements ("opdrachten"), aaneengeregen door puntkomma's. Bij executie van het programma worden deze statements na elkaar uitgevoerd.

Door de statements read(a) en read(b) krijgen a en b achtereen. volgens via het invoer medium hun waarden.
c:=a en c:=b zijn toekenningstatements. We spreken c:=a uit als "c wordt a". Door uitvoeren van c:=a wordt aan c de waarde van a toegekend.
Door de statement if a>b then c:=a else c:=b wordt aan c de waarde van a toegekend indien a>b en wordt aan c de waarde van b toegekend indien a≤b.
De statement write(c) tenslotte levert aan het uitvoermedium de waarde van c.

Om onze gedachten te kunnen beperken tot het essentiële van dit programma, namelijk de berekening van het maximum van twee getallen, abstraheren we voorlopig van de invoer en de uitvoer.
Met deze abstractie noteren we het programma als volgt:

```
var
   a,b,c : integer ;
begin
   { a=A ∧ b=B }
   if a>b then c:=a else c:=b
   { a=A ∧ b=B ∧ c=max(a,b) }
end.
```

Tussen accoladen zijn predicaten toegevoegd. Het eerste predicaat $a=A \wedge b=B$ geeft aan dat aan a en b beginwaarden toegekend zijn (we bekommeren ons er niet om hoe dit plaatsgevonden heeft). A en B zijn geen programmavariabelen; zij worden als constanten beschouwd.
Het tweede predicaat $a=A \wedge b=B \wedge c=max(a,b)$ geeft aan dat na afloop van de statement if a>b then c:=a else c:=b de variabelen a en b hun oorspronkelijke waarden hebben en dat c als waarde het maximum hiervan heeft.

Verder merken we op dat voorgaande manier van spreken over een programma erg operationeel is : eerst gebeurt dit, daarna dat. Dit sluit aan bij de manier waarop een automaat zo'n programma uitvoert.
Bij een eenvoudig programma levert deze manier van spreken niet veel problemen op. Bij ingewikkelder programma's daarentegen wordt het ondoenlijk om alle situaties die zich voordoen (of voor kunnen doen) op een dergelijke manier na te lopen.
We zullen dan ook abstraheren van een eventuele executie van het programma door een of andere rekenautomaat.

In een bepaalde zin, die later duidelijk zal worden, is de statement
if $a>b$ then $c:=a$ else $c:=b$
een oplossing van de volgende "vergelijking" in S.

```
var
  a,b,c : integer;
begin
  {a=A ∧ b=B}
  S
  {a=A ∧ b=B ∧ c=max(a,b)}
end.
```

In de volgende hoofdstukken houden we ons voornamelijk bezig met de vragen:

Wat wordt bedoeld met een oplossing S' van een dergelyke vergelyking?
Hoe kan zo'n S er uitzien en hoe vindt men een geschikte S?

1. Predicatenrekening en toestandsruimten.

In dit hoofdstuk wordt op informele wijze een gedeelte uit de logica behandeld. Tevens geven we aan hoe dit gebruikt wordt bij de beschrijving van statements.

De verzameling boolean is gedefinieerd door $\text{boolean} = \{\text{true}, \text{false}\}$.
Voor $p, q \in \text{boolean}$ definiëren we

de negatie	$\neg p$	(non p)
de conjunctie	$p \wedge q$	(p and q)
de disjunctie	$p \vee q$	(p or q)
de implicatie	$p \Rightarrow q$	(p implies q)
de equivalentie	$p \equiv q$	(p equivails q)

als volgt :

Als p true is dan is $\neg p$ false ; als p false is dan is $\neg p$ true.
Als zowel p als q true is dan is $p \wedge q$ true , anders is $p \wedge q$ false.
Als zowel p als q false is dan is $p \vee q$ false , anders is $p \vee q$ true.
De implicatie $p \Rightarrow q$ is hetzelfde als $(\neg p) \vee q$ en de equivalentie $p \equiv q$ is hetzelfde als $(p \wedge q) \vee ((\neg p) \wedge (\neg q))$.

Hierbij geven de haakjes aan in welke volgorde evaluatie plaatsvindt : eerst alles wat tussen haakjes staat, dan de rest.
Om het gebruik van haakjes te beperken, spreken we af dat $\neg$ het sterkst bindt, gevolgd door $\wedge$ en $\vee$, op hun beurt gevolgd door $\Rightarrow$ en $\equiv$.

Voor $p \Rightarrow q$ schrijven we ook $q \Leftarrow p$.

De volgende lijst van eigenschappen kunt u verifiëren met behulp van de gegeven definities.

Voor alle $p,q,r \in$ boolean geldt :

symmetrie : $p \wedge q = q \wedge p$; $p \vee q = q \vee p$; $p \equiv q = q \equiv p$

associativiteit : $p \wedge (q \wedge r) = (p \wedge q) \wedge r$; $p \vee (q \vee r) = (p \vee q) \vee r$; $p \equiv (q \equiv r) = (p \equiv q) \equiv r$

distributiviteit : $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$; $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$

false - true regels :

$p \wedge \text{false} = \text{false}$	$p \wedge \text{true} = p$	$p \wedge \neg p = \text{false}$
$p \vee \text{false} = p$	$p \vee \text{true} = \text{true}$	$p \vee \neg p = \text{true}$
$\text{false} \Rightarrow p = \text{true}$	$p \Rightarrow \text{true} = \text{true}$	$p \Rightarrow p = \text{true}$
$p \equiv \text{false} = \neg p$	$p \equiv \text{true} = p$	$p \equiv p = \text{true}$

dubbele ontkenning : $\neg\neg p = p$

de Morgan : $\neg(p \wedge q) = \neg p \vee \neg q$; $\neg(p \vee q) = \neg p \wedge \neg q$

idempotentie : $p \wedge p = p$; $p \vee p = p$

Programmapredicaten, kortweg predicaten genoemd, definiëren we met behulp van de volgende 3 regels :

(i) true en false zijn predicaten.

(ii) Wiskundige expressies waarin programmavariabelen voorkomen, zoals $a \geq b$ en $a+1 = 7$, waaraan na substitutie van waarden voor de erin voorkomende programmavariabelen op voor de hand liggende wijze de waarde true of de waarde false kan worden toegekend, zijn predicaten.

(iii) Als P en Q predicaten zijn dan ook $\neg P$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \equiv Q$ en (P); substitutie van **waarden voor daarin voorkomende** programmavariabelen levert met de gegeven definities van $\neg, \wedge, \vee, \Rightarrow$ en $\equiv$ de waarde true of de waarde false.

Heeft een programma variabelen **a en b** , type integer, dan kunnen we bijvoorbeeld de volgende predicaten beschouwen:

true, false, $a=b$, $a>1 \wedge b \leq 3$, $a \geq 0 \Rightarrow a \geq 1$, $a^2+b^2 \geq 0$ en $2 \leq 0$.

Het predicaat true heeft per definitie voor alle waarden van a en b de waarde true. Evenzo heeft het predicaat false voor alle waarden van a en b de waarde false.
$a \geq 0 \Rightarrow a \geq 1$ valt voor $a \neq 0$ samen met het predicaat true (heeft voor $a \neq 0$ de waarde true); voor $a=0$ valt het samen met het predicaat false.

Is P een predicaat dat voor alle waarden van de in P voorkomende variabelen gelijk is aan true, dan zeggen we dat P geldt.
Zo geldt als n programmavariabele is van het type integer dat
$0 \leq n \leq 10 \wedge n \neq 10 \Rightarrow 0 \leq n+1 \leq 10$

Ook geldt voor alle predicaten P en Q : $P \equiv P$, $P \wedge Q \equiv Q \wedge P$ en $P \equiv P \equiv \text{true}$

Merk op dat de ene keer true en false worden gebruikt als predicaten en de andere keer als waarden. U dient zelf uit de context op te maken wat bedoeld is.

Zijn P en Q predicaten waarvoor $P \Rightarrow Q$ geldt dan heet P sterker dan Q en Q heet zwakker dan P.
Merk op dat false sterker is dan ieder predicaat en dat true zwakker is dan ieder predicaat.

Laat a en b de programmavariabelen (van het type integer) van een programma zijn. Executie van een statement van dat programma zal in het algemeen resulteren in het wijzigen van de waarden van a of b.
Na executie van een statement bevindt de automaat waarop het programma verwerkt wordt zich in een bepaalde toestand die wordt gekarakteriseerd door de waarden van a en b op dat moment, zeg $a=x_0$ en $b=y_0$. Deze toestand wordt ook gekarakteriseerd door het paar (x_0,y_0).
We definiëren daarom de <u>toestandsruimte</u> T van dit programma door $T = \{ (x,y) \mid x \in \mathbb{Z} \wedge y \in \mathbb{Z} \}$. De elementen van T heten <u>toestanden</u>. We noemen een toestand ook wel een punt van de toestandsruimte. De 1^e coördinaat van zo'n punt correspondeert met de waarde van a en de 2^e coördinaat met de waarde van b, volgens afspraak.

Is P een programmapredicaat in a en b dan kan P in elk punt van de toestandsruimte geëvalueerd worden. Levert substitutie van x voor a en van y voor b in P de waarde true dan zeggen we dat het punt (x,y) aan P voldoet.
Identificeren we een predicaat P met de verzameling punten van de toestandsruimte die aan P voldoen, dan geldt:

conjunctie correspondeert met doorsnede,
disjunctie correspondeert met vereniging,

Verder correspondeert true met T en false met ϕ.

Uitspraken over een statement S zullen we doen in termen van predicaten. We gebruiken hiervoor de notatie $\{P\}\ S\ \{Q\}$ waarbij P en Q predicaten zijn. De operationele beschrijving van $\{P\}\ S\ \{Q\}$ luidt:

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

Bij het introduceren van statements beschrijven we deze eerst operationeel, in termen van toestanden. Daaruit leiden we een beschrijving af in termen van predicaten. Een aldus verkregen uitspraak verheffen we dan tot regel ("axioma").
Bij beschouwingen over programmateksten beroepen we ons alleen op de gegeven regels.

Als illustratie van het bovenstaande leiden we een algemene regel af. Veronderstel dat we weten $\{P\}\ S\ \{Q\}$. Is P' sterker dan P dan zal een toestand die aan P' voldoet ook aan P voldoen. Uitvoering van S beginnend in een toestand die aan P' voldoet eindigt dan ook in een toestand die aan Q voldoet en we concluderen $\{P'\}\ S\ \{Q\}$. Uit deze beschouwing leiden we de regel af:

Regel.

Is P' sterker dan P dan volgt uit $\{P\}\ S\ \{Q\}$ dat $\{P'\}\ S\ \{Q\}$.

Geef een analoog betoog ter ondersteuning van de volgende regel.

Regel.

Is Q' zwakker dan Q dan volgt uit $\{P\}\ S\ \{Q\}$ dat $\{P\}\ S\ \{Q'\}$.

Voordat we overgaan op de opgaven noemen we een drietal "bewijsmethoden".

Als eerste het "gepast" gevallenonderscheid. We lichten dit toe door te bewijzen dat voor predicaten P en Q geldt $P \wedge Q \Rightarrow P$.

> In een punt van de toestandsruimte waar P samenvalt met true staat er $true \wedge Q \Rightarrow true$ en dit geldt voor alle Q (false-true regel).
> In een punt van de toestandsruimte waar P samenvalt met false staat er $false \wedge Q \Rightarrow false$ ofwel (false_true regel) $false \Rightarrow false$, hetgeen geldt.
> Dus geldt $P \wedge Q \Rightarrow P$.

Een tweede vorm wordt veel gebruikt bij opgaven van de vorm "bewijs $P \Rightarrow Q$" of "bewijs $P \equiv Q$". Een bewijs van $P \Rightarrow Q$ verloopt meestal met tussenstappen, bijvoorbeeld $P \Rightarrow P_0$ en $P_0 \Rightarrow P_1$ en $P_1 \equiv Q$.
We noteren zo'n bewijs als volgt:

P
$\Rightarrow$ {argument waarom $P \Rightarrow P_0$}
P_0
$\Rightarrow$ {argument waarom $P_0 \Rightarrow P_1$}
P_1
$=$ {argument waarom $P_1 \equiv Q$}
Q

Voor een bewijs van $P \equiv Q$ geldt een analoge opzet. Als voorbeeld bewijzen we dat geldt $(P \wedge B \Rightarrow R) \equiv (P \Rightarrow (B \Rightarrow R))$.

$P \wedge B \Rightarrow R$
$=$ {definitie $\Rightarrow$}
$\neg(P \wedge B) \vee R$
$=$ {de Morgan}
$\neg P \vee \neg B \vee R$
$=$ {definitie $\Rightarrow$}
$P \Rightarrow \neg B \vee R$
$=$ {definitie $\Rightarrow$}
$P \Rightarrow (B \Rightarrow R)$

De derde methode is het bewijzen van $P \Rightarrow Q$ door Q aan te tonen met gebruikmaking van P. Als voorbeeld bewijzen we $(P \Rightarrow R) \Rightarrow (P \wedge Q \Rightarrow R)$.

Veronderstel $P \Rightarrow R$.

$P \wedge Q$

$\Rightarrow$ { zojuist bewezen met gevallenonderscheid }

P

$\Rightarrow$ { gegeven $P \Rightarrow R$ }

R

dus $(P \Rightarrow R) \Rightarrow (P \wedge Q \Rightarrow R)$

Opgaven.

In de volgende opgaven zijn P, Q en R programmapredicaten behorend bij eenzelfde programma.

1. Bewijs: a. $P \wedge Q \Rightarrow P$
   b. $P \Rightarrow P \vee Q$
   c. $P \wedge Q \Rightarrow P \vee Q$

2. a. Bewijs de absorptieregels: $(P \wedge Q) \vee P \equiv P$
   $(P \vee Q) \wedge P \equiv P$
   b. Toon met behulp van de distributiviteit van $\vee$ over $\wedge$ aan dat
   $((P \wedge Q) \vee P \equiv P) \equiv ((P \vee Q) \wedge P \equiv P)$

3. Bewijs of weerleg: a. $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$
   b. $((P \Rightarrow Q) \Rightarrow R) \Rightarrow (P \Rightarrow R)$
   c. $(P \Rightarrow (Q \Rightarrow R)) \equiv ((P \Rightarrow Q) \Rightarrow R)$

4. Ga de volgende beweringen na (x is programmavariabele van het type integer).
   a. $x \geq 1$ is zwakker dan $x = 7$
   b. $x \geq 1$ is sterker dan $x \geq 0$
   c. $x = 7$ is sterker dan $x \geq 1$
   d. $x \geq 0$ is zwakker dan $x^2 \geq 0$

5. a. Bepaal het zwakste predicaat X dat voldoet aan $X \Rightarrow P$
   b. Idem voor $X \wedge Q \Rightarrow P$
   c. Idem voor $P \Rightarrow X$

6. a. Bepaal het sterkste predicaat X dat voldoet aan $P \Rightarrow X$
   b. Idem voor $P \Rightarrow X \vee Q$
   c. Idem voor $X \Rightarrow P$

7. Bewijs:
   a. $(P \equiv Q) \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$
   b. $(P \equiv R) \wedge (R \equiv Q) \Rightarrow (P \equiv Q)$

8. Toon uit de gegeven regels aan dat uit $\{P\}\ S\ \{Q\}$ geconcludeerd kan worden: a. $\{false\}\ S\ \{Q\}$
   b. $\{P\}\ S\ \{true\}$
   Geef een operationele beschrijving van a. en b. in termen van toestanden.

9. Bewijs:
   a. $P \equiv P$
   b. $(P \equiv Q) \Rightarrow (P \Rightarrow Q)$
   c. $(P \wedge Q) \vee (Q \wedge R) \vee (R \wedge P) \equiv (P \vee Q) \wedge (Q \vee R) \wedge (R \vee P)$.

2. De assignment statement en de concatenatie.

2.1 De assignment statement.

De assignment statement (toekenningsopdracht) is van de vorm $x := E$ waarbij x een programmavariabele en E een toegestane expressie is. Een definitie van toegestane expressies vindt u in de syntaxdiagrammen in appendix A.

We beperken ons voorlopig tot programmavariabelen die hetzij van het type integer hetzij van het type boolean zijn.

Programmavariabelen van het type boolean kunnen alleen de waarde true of de waarde false aannemen.

In toegestane integerexpressies kunnen de operaties $+, -, *$, div en mod voorkomen. Hierbij staat + voor optelling, - voor aftrekking, * voor vermenigvuldiging, div voor quotient bij gehele deling en mod voor de rest bij gehele deling.
Zijn a en b gehelen met $a \geq 0$ en $b > 0$ dan zijn a div b en a mod b geheel, $a = b * (a \text{ div } b) + a \text{ mod } b$ en $0 \leq a \text{ mod } b < b$. Voor $b = 0$ zijn a div b en a mod b niet gedefinieerd.
*, div en mod binden sterker dan + en −. Met behulp van haakjes kan hiervan op de gebruikelijke wijze worden afgeweken.

In toegestane booleanexpressies kunnen de operatoren not, and en or voorkomen. Deze komen respectievelijk overeen met de negatie, de conjunctie en de disjunctie.

We spreken $x := E$ uit als "x wordt E". Het effect van $x := E$ is dat (de waarde van) x wordt vervangen door (de waarde van) E.
Veronderstel dat uitvoering van $x := E$ eindigt in een toestand die aan Q voldoet.
Daar aan x de waarde van E is toegekend zal de toestand vóór uitvoering van $x := E$ hebben voldaan aan het predicaat P dat uit Q ontstaat indien ieder voorkomen van programmavariabele x wordt vervangen door E.

Wil bijvoorbeeld na uitvoering van $x := x + 3$ een toestand bereikt zijn die voldoet aan $x > 10$ dan zal in de toestand voor uitvoering hebben gegolden $x + 3 > 10$ ofwel $x > 7$.

Geven we met Q^x_E het predicaat aan dat uit Q ontstaat door elk voorkomen van x door E te vervangen dan geldt $\{P\}\ x:=E\ \{Q\}$ blijkbaar dan als $P \Rightarrow Q^x_E$.

Geïnspireerd door deze operationele beschouwingen geven we voor de assignment statement de volgende <u>regel</u> .

<u>Regel van de assignment statement</u> .

$\{P\}\ x:=E\ \{Q\}$ is equivalent met $P \Rightarrow Q^x_E$.

Voorbeelden.

1. Het zwakste predicaat P dat voldoet aan $\{P\}\ x:=x+1\ \{x\geq 2\}$ wordt gegeven door $P = (x\geq 2)^x_{x+1} = (x+1\geq 2) = (x\geq 1)$.
   Daar $x\geq 2$ sterker is dan $x\geq 1$ geldt ook $\{x\geq 2\}\ x:=x+1\ \{x\geq 2\}$.

2. Het zwakste predicaat P met $\{P\}\ c:=a\ \{c=max(a,b)\}$ is $P = (c=max(a,b))^c_a = (a=max(a,b)) = (a\geq b)$ waaruit we concluderen $\{a\geq b\}\ c:=a\ \{c=max(a,b)\}$.
   Merk op dat ook geldt $\{a=b\}\ c:=a\ \{c=max(a,b)\}$.

3. Het zwakste predicaat P met $\{P\}\ x:=3\ \{x=3\}$ wordt gegeven door $P = (x=3)^x_3 = (3=3) = true$. Dus $\{true\}\ x:=3\ \{x=3\}$ en derhalve voor elk predicaat Q ook $\{Q\}\ x:=3\ \{x=3\}$.

In de uitdrukking $\{P\}\ S\ \{Q\}$ noemen we P de preconditie en Q de postconditie. De regel van de assignment statement kan ook aldus geformuleerd worden:

De zwakste preconditie P met $\{P\}\ x:=E\ \{Q\}$ wordt gegeven door $P = Q^x_E$.

In [6] wordt de betekenis van een statement S gedefinieerd door bij gegeven eindconditie Q de zwakste beginconditie P te geven zodanig dat $\{P\}\ S\ \{Q\}$ geldt.

De gebruikte notatie hiervoor is $wp(S,Q)$ waarbij wp een afkorting is van weakest precondition.
Met deze notatie geldt per definitie:

$$wp(x:=E,Q) \equiv Q_E^x .$$

2.2 De concatenatie.

Het programma uit de inleiding bevat statements "aaneengeregen" door puntkomma's. Zijn S_1 en S_2 statements dan heet $S_1 ; S_2$ de concatenatie van S_1 en S_2, met als operationele betekenis "voer eerst S_1 uit en daarna S_2".

Voldoen P, Q en R aan $\{P\}\,S_1\,\{R\}$ en $\{R\}\,S_2\,\{Q\}$ dan zal beginnend in een toestand die aan P voldoet na uitvoering van S_1 een toestand bereikt zijn die aan R voldoet. Daarna uitvoeren van S_2 leidt tot een toestand die aan Q voldoet. Met andere woorden $\{P\}\,S_1 ; S_2\,\{Q\}$. Dit leidt tot:

Regel van de concatenatie.

$\{P\}\ S_1 ; S_2\ \{Q\}$ is equivalent met

er is een R zodanig dat $\{P\}\,S_1\,\{R\}$ en $\{R\}\,S_2\,\{Q\}$.

Met behulp van de weakest precondition wordt de concatenatie gedefinieerd door

$$wp(S_1 ; S_2 , Q) \equiv wp(S_1 , wp(S_2, Q))$$

Definiëren we de afbeeldingen f_1 en f_2 door $f_1(X) \equiv wp(S_1, X)$ en $f_2(X) \equiv wp(S_2, X)$ dan geldt voor de samenstelling $f_1 \circ f_2$ van f_1 en f_2:

$f_1 \circ f_2(X) = f_1(f_2(X)) = f_1(wp(S_2,X)) = wp(S_1, wp(S_2,X))$. De ; correspondeert blijkbaar met functiesamenstelling en is derhalve associatief : $(S_1 ; S_2) ; S_3$ is hetzelfde als $S_1 ; (S_2 ; S_3)$, reden waarom geen haakjes gebruikt worden.

Als voorbeeld tonen we aan dat voor integer variabelen x en y geldt

$\{x=A \wedge y=B\}\ \ x:=x+y\ ;\ y:=x-y\ ;\ x:=x-y\ \ \{x=B \wedge y=A\}.$

De zwakste preconditie P met $\{P\}\ x:=x-y\ \{x=B \wedge y=A\}$ is
$(x=B \wedge y=A)^{x}_{x-y} = (x-y=B \wedge y=A) = (x=A+B \wedge y=A).$

De zwakste preconditie Q met $\{Q\}\ y:=x-y\ \{x=A+B \wedge y=A\}$ is
$(x=A+B \wedge y=A)^{y}_{x-y} = (x=A+B \wedge x-y=A) = (x=A+B \wedge y=B).$

Tenslotte geldt $(x=A+B \wedge y=B)^{x}_{x+y} = (x+y=A+B \wedge y=B) = (x=A \wedge y=B).$

We hebben nu aangetoond

$$\begin{array}{lll} \{x=A \wedge y=B\} & x:=x+y & \{x=A+B \wedge y=B\} \\ \{x=A+B \wedge y=B\} & y:=x-y & \{x=A+B \wedge y=A\} \\ \{x=A+B \wedge y=A\} & x:=x-y & \{x=B \wedge y=A\} \end{array}$$

waaruit volgt

$\{x=A \wedge y=B\}\ \ x:=x+y\ ;\ y:=x-y\ ;\ x:=x-y\ \ \{x=B \wedge y=A\}.$

Ga na dat we hebben laten zien $x=A \wedge y=B \equiv (((x=B \wedge y=A)^{x}_{x-y})^{y}_{x-y})^{x}_{x+y}$
waarbij de substituties "van binnen naar buiten" uitgevoerd worden.

Merk op dat
$$\begin{aligned} & wp(x:=x+y\ ;\ y:=x-y\ ;\ x:=x-y\ ,\ P) \\ &= wp(x:=x+y, wp(y:=x-y, wp(x:=x-y, P))) \\ &= ((P^{x}_{x-y})^{y}_{x-y})^{x}_{x+y} \end{aligned}$$

Opgaven.

In de volgende opgaven zijn m, n, x, y en z programmavariabelen van het type integer en a en b programmavariabelen van het type boolean.

1. Bepaal het zwakste predicaat P waarvoor

a. $\{P\}\ x := (x-1)*(x+1)\ \{x \geq 0\}$

b. $\{P\}\ x := x-1\ \{x \geq 10\}$

c. $\{P\}\ x := 2*x+y\ \{x = 2x+y\}$

d. $\{P\}\ x := x+y-z\ \{x \geq 0\}$

e. $\{P\}\ b := a \text{ or } b\ \{b \equiv a\}$

f. $\{P\}\ x := 3\ \{x \geq 2\}$

g. $\{P\}\ x := 3\ \{x \leq 2\}$

h. $\{P\}\ b := a \text{ and } b\ \{b \Rightarrow a\}$

2. Idem:

a. $\{P\}\ z := x\ ;\ x := y\ ;\ y := z\ \{x = A \wedge y = B\}$

b. $\{P\}\ y := y+2*x+1\ ;\ x := x+1\ \{y = x^2\}$

c. $\{P\}\ y := y+3*z+3*x+1\ ;\ z := z+2*x+1\ ;\ x := x+1\ \{y = x^2 \wedge z = x^2\}$

d. $\{P\}\ x := x \text{ div } 2\ ;\ y := y*y\ \{z = y^x\}$

e. $\{P\}\ z := z*y\ ;\ x := x+1\ \{z = y^x\}$

3. Vul "geschikte" statements in (A en N zijn constanten).

a. $\{0 \leq n < N \wedge x = 3^n\}\ \ldots\ ;\ n := n+1\ \{1 \leq n \leq N \wedge x = 3^n\}$

b. $\{2x+3y = A\}\ \ldots\ \{2y+3x = A\}$

c. $\{0 \leq n < N \wedge x = 2^n \wedge y = 3^n \wedge z = 2^n+3^n\}\ \ldots\ ;\ n := n+1\ \{0 < n \leq N \wedge x = 2^n \wedge y = 3^n \wedge z = 2^n+3^n\}$

d. $\{x = A \wedge x > 0\}\ \ldots\ \{10x+y = A \wedge 0 \leq y < 10\}$

4. Bepaal het zwakste predicaat met

a. $\{P\}\ n := n+1\ ;\ x := n*n*n+x\ \{0 < n \leq 100 \wedge x = \sum_{i=1}^{n} i^3\}$

b. $\{P\}\ x := x+n*n*n\ ;\ n := n+1\ \{0 < n \leq 100 \wedge x = \sum_{i=1}^{n} i^3\}$

c. $\{P\}\ y := 2*x+1\ ;\ y := (y-1) \text{ div } 2\ \{y = x\}$

d. $\{P\}\ y := (x-1) \text{ div } 2\ ;\ y := 2*y+1\ \{y = x\}$

5. Bepaal een statement S zodanig dat voor alle predicaten P geldt

$\{P\}\ y := 2*y+1\ ;\ S\ \{P\}$

6.* De expressie a mod b is voor b=0 niet gedefinieerd. Ook predicaten waarin a mod b voorkomt zijn niet gedefinieerd voor b=0. Dergelijke predicaten noemen we partiële predicaten.
Is P een predicaat en Q een partieel predicaat dan zijn P cand Q en P cor Q als volgt gedefinieerd.

P cand Q.

In een punt waar P met false samenvalt heeft P cand Q de waarde false.
In een punt waar P met true samenvalt heeft P cand Q de waarde van Q in dat punt.

P cor Q.

In een punt waar P met true samenvalt heeft P cor Q de waarde true.
In een punt waar P met false samenvalt heeft P cor Q de waarde van Q in dat punt.

Is E een expressie dan is def(E) het predicaat dat op het definitiegebied van E samenvalt met true en daarbuiten samenvalt met false. Zo is def(a mod b) ≡ (b≠0) en def(a div (m-n)) ≡ (m≠n).

Een vollediger beschrijving van de assignment geeft de definitie

$wp(x:=E, Q) \equiv def(E) \ \underline{cand} \ Q^x_E$.

Ga na:
a. cand en cor zijn niet symmetrisch.
b. $\neg(P \ \underline{cand} \ Q) \equiv \neg P \ \underline{cor} \ \neg Q$ en $\neg(P \ \underline{cor} \ Q) \equiv \neg P \ \underline{cand} \ \neg Q$.
c. cand en cor zijn associatief.
d. cand distribueert over cor en cor distribueert over cand.

7. Een statement S kan gedefinieerd worden in termen van de weakest precondition. Niet elke zomaar gedefinieerde S zal zinvol zijn. We leggen daarom de volgende eisen op:

(i) $wp(S, false) \equiv false$
(ii) $wp(S, P \wedge Q) \equiv wp(S,P) \wedge wp(S,Q)$ voor alle predicaten P en Q.

a. Geef (i) en (ii) weer in termen van toestanden.

b. Toon aan: als $P \Rightarrow Q$ dan $wp(S,P) \Rightarrow wp(S,Q)$
(Hint: bewijs dat $(P \Rightarrow Q) \equiv (P \wedge Q \equiv P)$).

c. Idem voor: $wp(S,P) \vee wp(S,Q) \Rightarrow wp(S, P \vee Q)$.

d. Ga na dat de assignment statement voldoet aan (i) en (ii).

e. Veronderstel dat S_1 en S_2 aan (i) en (ii) voldoen.
Toon aan dat ook de concatenatie $S_1; S_2$, gedefinieerd door
$wp(S_1; S_2, Q) \equiv wp(S_1, wp(S_2, Q))$ aan (i) en (ii) voldoet

f. Geef een interpretatie van een derde eis, te weten
(iii) $wp(S,P) \vee wp(S,Q) \equiv wp(S, P \vee Q)$ voor alle predicaten P en Q.

8. Bepaal het zwakste predicaat P met:

a. $\{P\}\ x := x - y\ \{x > 0 \wedge y > 0 \wedge ggd(x,y) = 7\}$

b. $\{P\}\ x := x - y\ ;\ y := x + y\ ;\ x := y - x\ \{x = A \wedge y = B\}$

3. De empty statement, de alternatieve statement en de compound statement.

3.1 De empty statement.

De empty statement wordt in Pascal aangegeven door niets op te schrijven. Hierdoor is deze statement lastig te herkennen. De concatenatie van $x := x+1$, de empty statement en $y := 2*x$ ziet er als volgt uit :

$x := x+1 \;;; y := 2*x$.

Operationeel gezien komt uitvoering van de empty statement neer op het onveranderd laten van de heersende toestand. Uitvoering van de empty statement beginnend in een toestand die aan P voldoet eindigt in een toestand die aan P voldoet. Dit leidt tot

Regel van de empty statement

$\{P\}\ \{Q\}$ is equivalent met $P \Rightarrow Q$

Geven we de empty statement aan met skip dan luidt de definitie in termen van de weakest precondition :

$wp(skip, Q) \equiv Q$

Het nut van de empty statement wordt in volgende paragrafen duidelijk.

3.2 De alternatieve statement.

De alternatieve statement geeft de mogelijkheid om van de begintoestand af te laten hangen welk statement uitgevoerd wordt.
De statement heeft de vorm

if B then S1 else S2

waarin B een toegestane boolse expressie is en S1 en S2 statements zijn.

Uitvoering van if B then S_1 else S_2 in een toestand die aan B voldoet komt neer op uitvoering van S_1. Uitvoering in een toestand die aan $\neg B$ voldoet komt neer op uitvoering van S_2.

Geldt $\{P \wedge B\}\, S_1\, \{Q\}$ en $\{P \wedge \neg B\}\, S_2\, \{Q\}$ dan zal uitvoering van if B then S_1 else S_2 beginnend in een toestand die aan P voldoet (dus aan $P \wedge (B \vee \neg B)$) eindigen in een toestand die aan Q voldoet. Dit leidt tot

Regel van de alternatieve statement (1).

Als $\{P \wedge B\}\, S_1\, \{Q\}$ en $\{P \wedge \neg B\}\, S_2\, \{Q\}$ dan $\{P\}$ if B then S_1 else S_2 $\{Q\}$

Als voorbeeld bepalen we S zodanig dat $\{true\}\ S\ \{c = max(a,b)\}$ en S a en b onverlet laat.

Herschrijving met de definitie van max levert

$\{true\}\ S\ \{(c=a \vee c=b) \wedge c \geq a \wedge c \geq b\}$

Keuze van $c := a$ voor S geeft als zwakste preconditie

$$((c=a \vee c=b) \wedge c \geq a \wedge c \geq b)^c_a$$
$$= ((a=a \vee a=b) \wedge a \geq a \wedge a \geq b)$$
$$= (a \geq b)$$

Hieruit concluderen we: $\{a \geq b\}\ c := a\ \{c = max(a,b)\}$

en door naamswisseling: $\{b \geq a\}\ c := b\ \{c = max(a,b)\}$

Daar $b > a$ sterker is dan $b \geq a$ geldt ook $\{b > a\}\ c := b\ \{c = max(a,b)\}$.
Verder is $b > a \equiv \neg(a \geq b)$ en we hebben aangetoond

$\{a \geq b\}\ c := a\ \{c = max(a,b)\}$
$\{\neg(a \geq b)\}\ c := b\ \{c = max(a,b)\}$

De regel van de alternatieve statement levert

$\{true\}$ if $a \geq b$ then $c := a$ else $c := b$ $\{c = max(a,b)\}$.

Kiezen we voor S_2 de empty statement dan wordt de regel

Als $\{P \wedge B\}\ S_1\ \{Q\}$ en $P \wedge \neg B \Rightarrow Q$ dan $\{P\}$ if B then S_1 else $\{Q\}$.

Voor dit speciale geval bestaat in Pascal een tweede versie van de alternatieve statement van de vorm if B then S. Naar aanleiding van het bovenstaande geven we hiervoor de volgende regel:

Regel van de alternatieve statement (2)

Als $\{P \wedge B\}\ S\ \{Q\}$ en $P \wedge \neg B \Rightarrow Q$ dan $\{P\}$ if B then S $\{Q\}$.

In termen van de weakest precondition kan de alternatieve statement als volgt gedefinieerd worden:

$$wp(\text{if } B \text{ then } S_1 \text{ else } S_2, Q) \equiv (B \Rightarrow wp(S_1, Q)) \wedge (\neg B \Rightarrow wp(S_2, Q))$$

De equivalentie met de regel van de alternatieve statement volgt uit de equivalentie van $\{P \wedge B\}\ S_1\ \{Q\}$ en $P \Rightarrow (B \Rightarrow wp(S_1, Q))$ en die van $\{P \wedge \neg B\}\ S_2\ \{Q\}$ en $P \Rightarrow (\neg B \Rightarrow wp(S_2, Q))$.
Ter illustratie tonen we aan dat $\{P \wedge B\}\ S_1\ \{Q\}$ equivalent is met $P \Rightarrow (B \Rightarrow wp(S_1, Q))$.

$\{P \wedge B\}\ S_1\ \{Q\}$

$=$ { definitie van deze notatie }

$P \wedge B \Rightarrow wp(S_1, Q)$

$=$ { definitie van $\Rightarrow$ }

$\neg(P \wedge B) \vee wp(S_1, Q)$

$=$ { de Morgan }

$\neg P \vee \neg B \vee wp(S_1, Q)$

$=$ { definitie van $\Rightarrow$ }

$P \Rightarrow \neg B \vee wp(S_1, Q)$

$=$ { definitie van $\Rightarrow$ }

$P \Rightarrow (B \Rightarrow wp(S_1, Q))$

3.3 De compound statement.

Is in de alternatieve statement if B then S_0 else S_1 de statement S_1 een concatenatie zeg $S_2 ; S_3$ dan is niet duidelijk of

	if B then S_0 else $S_2 ; S_3$
gelezen moet worden als	(if B then S_0 else S_2) ; S_3
of als	if B then S_0 else ($S_2 ; S_3$)

De eerste lezing wordt bedoeld; een concatenatie als in de tweede lezing geven we in Pascal aan door de concatenatie vooraf te laten gaan door begin en af te sluiten met end. Een dergelijke omsloten concatenatie heet een compound statement. De betekenis ervan is die van de omsloten concatenatie. De tweede lezing wordt dus aangegeven door if B then S_0 else begin $S_2 ; S_3$ end.

Ga de juistheid na van : $\{ a>b \wedge b \neq c \}$
if $a<b$ then begin $c:=a ; a:=b ; b:=a$ end
$\{ b \neq c \}$

en van : $\{ a>b \wedge b \neq c \}$
if $a<b$ then $c:=a ; a:=b ; b:=c$
$\{ b=c \}$

Een andere dubbelzinnigheid is de volgende constructie :

	if B_0 then if B_1 then S_0 else S_1
te lezen als:	if B_0 then begin if B_1 then S_0 else S_1 end
of als:	if B_0 then begin if B_1 then S_0 end else S_1

Het Pascal Report [4] geeft aan dat de eerste lezing is bedoeld. Waar dit de duidelijkheid bevordert kan het gebruik van overbodige begin en end geen kwaad.

Voordat we overgaan tot de opgaven geven we aan de hand van voorbeelden aan hoe programmeerproblemen gespecificeerd worden.

We spreken af dat programmavariabelen die een beginwaarde hebben niet gewijzigd mogen worden door het programma tenzij de specificatie van de opgave expliciet aangeeft dat dit is toegestaan.

De volgende vier specificaties zijn een specificatie van eenzelfde probleem:

(i) Schrijf een programma dat van twee gegeven getallen het maximum bepaalt.

(ii) Gegeven integers x en y. Schrijf een programma dat $z = max(x,y)$ bewerkstelligt.

(iii) Bepaal S zodanig dat
$$\{ x = X \wedge y = Y \}$$
$$S$$
$$\{ x = X \wedge y = Y \wedge z = max(x,y) \}$$

(iv) Bepaal S met
$$\{ x = X \wedge y = Y \}$$
$$S$$
$$\{ z = max(x,y) \}$$

Van specificaties (i) en (ii) zal men meestal overgaan naar (iii) of (iv) teneinde gegeven regels te kunnen toepassen. Wegens bovengenoemde afspraak zijn (iv) en (iii) dezelfde specificatie.

Een ander probleem zou kunnen luiden:

Bepaal S zodanig dat
$$\{ x = X \wedge y = Y \}$$
$$S$$
$$\{ x = max(X,Y) \}$$

Hier wordt expliciet van de afspraak afgeweken: het is kennelijk de bedoeling dat aan x een waarde wordt toegekend.
Ga na dat een oplossing van dit laatste probleem gegeven wordt door S: **if** $x \leq y$ **then** $x := y$ door aan te tonen de "bewijsverplichtingen":

$$x = X \wedge y = Y \wedge x \leq y \Rightarrow (x = max(X,Y))^x_y$$

$$x = X \wedge y = Y \wedge \neg(x \leq y) \Rightarrow x = max(X,Y)$$

Opgaven.

1. Toon aan: $\{\ y=2x \ \vee \ y=-2x+1\}$

   if y mod $2=0$ then $x:=x-y$ else $x:=x+y$

   ; $y:=y+1$

   $\{\ y=2x \ \vee \ y=-2x+1\}$

2. Idem: $\{\ q*w+r=x \ \wedge \ r\geq 0\}$

   if $w\leq r$ then begin $r:=r-w$; $q:=q+1$ end

   $\{\ q*w+r=x \ \wedge \ r\geq 0\}$

3. Bepaal S met $\{\ x=X\}$

   S

   $\{\ y=|x| \ \wedge \ x=X\}$

4. Idem voor $\{\ x=X\}$

   S

   $\{\ x=|X|\}$

5. Idem voor $\{\ x=X \ \wedge \ y=Y\}$

   S

   $\{\ b=(x\geq y)\}$

6. Gegeven integers x en y. Schrijf een programma dat bepaalt of een van hen een geheel veelvoud is van de ander.

7. Wat dient men te bewijzen teneinde te kunnen concluderen:

   $\{P\}$ if B_1 then S_1 else if B_2 then S_2 $\{Q\}$?

8. Bewijs: $\{\ ggd(x,y)=G \ \wedge \ x>0 \ \wedge \ y>0\}$

   if $x>y$ then $x:=x-y$ else if $y>x$ then $y:=y-x$

   $\{\ ggd(x,y)=G \ \wedge \ x>0 \ \wedge \ y>0\}$

9* We korten if B then S_1 else S_2 af met IF. Per definitie geldt

   $wp(IF,Q) \equiv (B \Rightarrow wp(S_1,Q)) \ \wedge \ (\neg B \Rightarrow wp(S_2,Q))$.

   Veronderstel dat $wp(S_i,\text{false}) \equiv \text{false}$

   en $wp(S_i,P\wedge Q) \equiv wp(S_i,P) \ \wedge \ wp(S_i,Q)$ $\quad (i=1,2)$.

   Toon aan: $wp(IF,\text{false}) \equiv \text{false}$

   en $wp(IF,P\wedge Q) \equiv wp(IF,P) \ \wedge \ wp(IF,Q)$.

4. De repetitieve statement.

De repetitieve statement, kortweg repetitie genoemd, is van de vorm
while B **do** S

Hierin is B een toegestane boolse expressie en S een statement.
Operationeel beschrijven we **while** B **do** S als volgt.

De boolse expressie B wordt geëvalueerd.
In een toestand waarin dit de waarde false oplevert reduceert verdere uitvoering van de repetitie tot de empty statement.
In een toestand waarin evaluatie van B de waarde true geeft wordt statement S uitgevoerd waarna het proces herhaald wordt.

Anders geformuleerd:

while B **do** S is equivalent met

if B **then** **begin** S; **while** B **do** S **end**

We bekijken de uitvoering van **while** B **do** S beginnend in een toestand die aan zeker programmapredicaat P voldoet.

Voldoet deze begintoestand ook aan $\neg B$ dan eindigt de repetitie volgens bovenstaande in een toestand die aan $P \wedge \neg B$ voldoet.
Dus: $\{P \wedge \neg B\}$ **while** B **do** S $\{P \wedge \neg B\}$

Voldoet de begintoestand aan B, dus aan $P \wedge B$, dan wordt S uitgevoerd. Veronderstel dat S zodanig is dat uitvoering van S beginnend in een toestand die aan $P \wedge B$ voldoet eindigt in een toestand die weer aan P voldoet, i.e. $\{P \wedge B\}$ S $\{P\}$. Als dan de repetitie eindigt, dat wil zeggen dat na eindig veel executies van S een toestand wordt bereikt die aan $\neg B$ voldoet, dan geldt voor de bereikte eindtoestand wederom $P \wedge \neg B$.

Dit leidt tot de volgende regel.

Regel van de repetitieve statement (1) (C.A.R. Hoare).

Als $\{P \wedge B\}\ S\ \{P\}$ dan $\{P\}$ **while** B **do** S $\{P \wedge \neg B\}$,

mits deze repetitie eindigt.

Een predicaat P met $\{P \wedge B\}\ S\ \{P\}$ heet een invariant van de repetitie **while** B **do** S.

We lichten de regel van Hoare toe met een klassiek voorbeeld.
Gevraagd wordt ggd met specificatie: $\{x>0 \wedge y>0 \wedge ggd(x,y)=G\}$

ggd

$\{x=G \wedge y=G\}$

Omdat $G>0$ en $ggd(x,x)=x$ voor $x>0$ kunnen we de eindconditie ook schrijven als $x>0 \wedge y>0 \wedge ggd(x,y)=G \wedge x=y$ (ga na!).
Met P gedefinieerd als

P: $x>0 \wedge y>0 \wedge ggd(x,y)=G$

herschrijven we de specificatie van ggd: $\{P\}$

ggd

$\{P \wedge x=y\}$

We zoeken nu voor ggd een oplossing van de vorm
while $x \neq y$ **do** S, met S zodanig dat $\{P \wedge x \neq y\}\ S\ \{P\}$ en uitvoering van S ons "dichter bij" het einddoel $x=y$ brengt.

Uit $x>y \wedge y>0 \wedge ggd(x,y)=G \Rightarrow x-y>0 \wedge y>0 \wedge ggd(x-y,y)=G$
en $y>x \wedge x>0 \wedge ggd(x,y)=G \Rightarrow y-x>0 \wedge x>0 \wedge ggd(x,y-x)=G$
volgt:
$\{P \wedge x>y\}\ x:=x-y\ \{P\}$
en $\{P \wedge x<y\}\ y:=y-x\ \{P\}$

en dus:
$\{P \wedge x \neq y\}$ **if** $x>y$ **then** $x:=x-y$ **else** $y:=y-x$ $\{P\}$.

Dit levert als oplossing voor ggd

while $x \neq y$ **do** **if** $x>y$ **then** $x:=x-y$ **else** $y:=y-x$

mits deze repetitie eindigt.

Nu de eindiging van deze repetitie. Men zou kunnen hopen, gezien het einddoel $x=y$, dat het verschil tussen x en y, i.e. $|x-y|$, in elke slag afneemt.
Ga zelf na dat deze hoop niet gerechtvaardigd is.

We tonen de eindiging als volgt aan.

De waarde van $x+y$ is geheel, wegens de invariant P positief, en neemt bij elke repetitieslag af.

Bewijs dit zelf door aan te tonen

(i) $P \wedge x \neq y \Rightarrow x+y > 0$

(ii) $\{P \wedge x \neq y \wedge x+y = C\}$ <u>if</u> $x > y$ <u>then</u> $x := x-y$ <u>else</u> $y := y-x$ $\{x+y < C\}$
voor elke constante C.

We noemen $x+y$ een variante functie van de repetitie.

Uit het voorgaande volgt een tweede regel voor de repetitie waarin de eindiging begrepen is.

<u>Regel van de repetitieve statement (2) (E.W. Dijkstra).</u>

Als (i) $\{P \wedge B\}\ S\ \{P\}$

en vf is een integer functie van de toestand met

(ii) $P \wedge B \Rightarrow vf \geq 0$ en

(iii) $\{P \wedge B \wedge vf = VF\}\ S\ \{vf < VF\}$

dan $\{P\}$ <u>while</u> B <u>do</u> S $\{P \wedge \neg B\}$.

We besluiten dit hoofdstuk met een aanvulling op de begrippen en notaties van de predicatenrekening.

De disjunctie van programmapredicaten is in hoofdstuk 1 ingevoerd. Zo'n disjunctie heeft in een punt van de bijbehorende toestandsruimte precies dan de waarde true indien ten minste één van deze predicaten in dat punt de waarde true heeft.

Zijn $P(0)$ tot en met $P(n-1)$ n programmapredicaten dan noteren we de disjunctie hiervan als

$$(\underline{E}\, i : 0 \leq i < n : P(i)) .$$

Merk op:

als $n=0$ dan $(\underline{E}\, i : 0 \leq i < n : P(i)) \equiv \text{false}$

als $n \geq 0$ dan $(\underline{E}\, i : 0 \leq i < n+1 : P(i)) \equiv (\underline{E}\, i : 0 \leq i < n : P(i)) \vee P(n)$.

De onderstreepte letter $\underline{E}$ staat voor existentiële quantificatie. In de uitdrukking $(\underline{E}\, i : 0 \leq i < n : P(i))$ heet i een gebonden variabele of ook wel dummy variabele.
De naam van een gebonden variabele doet er niet toe mits we geen bestaande namen (zoals bijvoorbeeld van programmavariabelen) gebruiken: de naam moet "vers" zijn en bestaat alleen binnen de quantificatie.
Tussen de dubbele punten staat het domein $0 \leq i < n$ van de gebonden variabele. Voor $n \leq 0$ is het domein leeg, voor $n > 0$ bevat het domein de getallen 0 tot en met $n-1$. Tenzij expliciet anders aangegeven is het domein steeds een deel van $\mathbb{Z}$.

Voor $(\underline{E}\, i : 0 \leq i < n : (\underline{E}\, j : 0 \leq j < m : P(i,j)))$ schrijven we ook
$(\underline{E}\, i,j : 0 \leq i < n \wedge 0 \leq j < m : P(i,j))$.

De conjunctie van een stel programmapredicaten heeft in een punt van de bijbehorende toestandsruimte precies dan de waarde false indien tenminste één van deze predicaten in dat punt de waarde false heeft.
De conjunctie van $P(0)$ tot en met $P(n-1)$ noteren we als

$$(\underline{A}\, i : 0 \leq i < n : P(i)) .$$

De onderstreepte letter $\underline{A}$ staat voor universele quantificatie. Merk op dat

als $n=0$ dan $(\underline{A}i : 0 \le i < n : P.i) \equiv true$

als $n \ge 0$ dan $(\underline{A}i : 0 \le i < n+1 : P.i) \equiv (\underline{A}i : 0 \le i < n : P.i) \wedge P.n$

Het verband met existentiële quantificatie wordt gegeven door de wetten van de Morgan:

$(\underline{A}i : 0 \le i < n : P.i) \equiv \neg(\underline{E}i : 0 \le i < n : \neg P.i)$

$(\underline{E}i : 0 \le i < n : P.i) \equiv \neg(\underline{A}i : 0 \le i < n : \neg P.i)$

We generaliseren genoemde quantificaties door andere domeinen (eventueel oneindige) toe te staan. De algemene vorm voor existentiële quantificatie is $(\underline{E}i : R : P)$ en voor universele quantificatie $(\underline{A}i : R : P)$.
Hierin is i een naam en zijn R en P predicaten.

We geven enkele rekenregels.

$(\underline{A}i : false : P) \equiv true$

$(\underline{E}i : false : P) \equiv false$

Komt i niet voor in Q dan: $(\underline{A}i : R : P \vee Q) \equiv (\underline{A}i : R : P) \vee Q$

$(\underline{E}i : R : P \wedge Q) \equiv (\underline{E}i : R : P) \wedge Q$

Is R een niet-leeg domein, i.e. $(\underline{E}i : R : true) \equiv true$, en komt i niet voor in Q dan: $(\underline{A}i : R : P \wedge Q) \equiv (\underline{A}i : R : P) \wedge Q$

$(\underline{E}i : R : P \vee Q) \equiv (\underline{E}i : R : P) \vee Q$

Verder geldt

$(\underline{A}i : R : P) \wedge (\underline{A}i : R : Q) \equiv (\underline{A}i : R : P \wedge Q)$

$(\underline{E}i : R : P) \vee (\underline{E}i : R : Q) \equiv (\underline{E}i : R : P \vee Q)$

Het aantal verschillende waarden in domein R waarvoor P de waarde true heeft geven we aan met $(\underline{N} i : R : P)$.

De som over alle verschillende waarden in domein R van E geven we aan met $(\underline{S} i : R : E)$.

Het maximum over alle waarden in domein R van E geven we aan met $(\underline{MAX}\ i : R : E)$ en evenzo het minimum met $(\underline{MIN}\ i : R : E)$.

Geven we het maximum van a en b aan met de infix-notatie $a \underline{max} b$ dan geldt voor $n \geq 1$ en f een integer functie :
$(\underline{MAX}\ i : 0 \leq i < n+1 : f(i)) = (\underline{MAX}\ i : 0 \leq i < n : f(i))\ \underline{max}\ f(n)$.

(Hoe definieert u het maximum over een leeg domein ?).

Is voor $0 \leq i < N$ $X[i]$ een geheel getal, hetgeen we ook aangeven met $X(i : 0 \leq i < N)$ is een integer array, dan geldt
als $n = 0$ dan $(\underline{S} i : 0 \leq i < n : X[i]) = 0$
als $0 \leq n < N$ dan $(\underline{S} i : 0 \leq i < n+1 : X[i]) = (\underline{S} i : 0 \leq i < n : X[i]) + X[n]$

We gebruiken rechte haken bij $X[i]$ om aan te sluiten bij Pascal-notaties.

Is $f(i : i \geq 0)$ een boolean functie dan kunnen we de uitspraak dat k het op een na kleinste natuurlijke getal is met $f(k) = true$ op verschillende manieren weergeven :
als
$$(\underline{E} j : 0 \leq j < k : f(j)) \wedge (\underline{A} i : 0 \leq i < j : \neg f(i)) \wedge (\underline{A} i : j+1 \leq i < k : \neg f(i)) \wedge f(k)$$

of als :
$$k = (\underline{MIN}\ i : i \geq 0 \wedge f(i) \wedge (\underline{E} j : 0 \leq j < i : f(j)) : i)$$

of als :
$$f(k) \wedge ((\underline{N} i : 0 \leq i < k : f(i)) = 1)$$

Opgaven.

1. Het volgende algoritme bepaalt quotient q en rest r bij deling van x door y. Toon de correctheid aan.

$\{ x \geq 0 \wedge y > 0 \}$

$q := 0 \; ; \; r := x$

{ invariant P : $r \geq 0 \wedge x = q \cdot y + r \wedge y > 0$

variante functie : r }

; **while** $r \geq y$ **do**

begin $r := r - y \; ; \; q := q + 1$ **end**

$\{ x = q \cdot y + r \wedge 0 \leq r < y \}$

2. Toon aan dat de volgende repetitie eindigt.

$\{ x = X \wedge y = Y \wedge x > 0 \wedge y > 0 \}$

while $x \neq y$ **do**

if $x > y$ **then begin** $z := x \; ; \; x := y \; ; \; y := z$ **end**

else $y := y - x$

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

3. Een rangeerterrein bevat een aantal treinen. Elke trein bestaat uit 1 of meer wagons.

Men start een proces dat per slag een trein uitkiest en

- deze verwijdert indien de trein uit 1 wagon bestaat
- deze splitst in twee treinen indien de trein uit meer dan 1 wagon bestaat.

Het proces eindigt indien het rangeerterrein geen trein meer bevat.

Bewijs dat het proces eindigt, i.e. bedenk een variante functie.

4. De variabelen a en b zijn van het type boolean.

Bewijs:

$\{ a = A \wedge b = B \}$ $a := (a = b) \; ; \; b := (a = b) \; ; \; a := (a = b)$ $\{ a = B \wedge b = A \}$

5. Bewijs de correctheid van de volgende "drie-deler" waarin alleen gebruik gemaakt wordt van div 4 en mod 4.

```
{a ≥ 0}
q := 0 ; r := a+1
; while r ≥ 4 do
  begin
      x := r div 4 ; y := r mod 4
    ; q := q+x ; r := x+y
  end
; r := r-1
{ a = 3q + r  ∧  0 ≤ r < 3 }
```

6. De cijfersom C(x) van een natuurlijk getal x is gedefinieerd door:

$C(0) = 0$

$C(n) = C(n \text{ div } 10) + n \text{ mod } 10$ voor $n \geq 1$.

Toon aan:

```
{ x ≥ 0 }
c := 0 ; y := x
{invariant P :  x ≥ 0 ∧ y ≥ 0 ∧ C(x) = c + C(y)
 variante functie : y }
; while y ≠ 0 do
  begin
      c := c + y mod 10 ; y := y div 10
  end
{ c = C(x) }
```

7. Bepaal S zodanig dat

```
{ N ≥ 0 ∧ X(i : 0 ≤ i < N) integer array }
  S
{ r = (Σ i : 0 ≤ i < N : X[i]) }
```

8. Gegeven is een integer N met $N \geq 1$ en een integer array $X(i: 0 \leq i < N)$. Geef de volgende uitspraken in een formule weer.

a. Alle waarden in de rij zijn positief.
b. X[k] is de op drie na grootste van de rij.
c. De rij bevat alleen nullen en enen en bestaat uit 0 of meer nullen gevolgd door 0 of meer enen.
d. p komt even vaak voor in $X(i: 0 \leq i < j)$ als in $X(i: j \leq i < N)$.
e. r is het aantal verschillende waarden van de rij.
f. $X(i: 0 \leq i < N)$ is een permutatie van de getallen 0 tot en met N-1.
g. r is de maximale lengte van enige consecutieve deelrij van $X(i: 0 \leq i < N)$ waarvan alle elementen dezelfde waarde hebben.
h. Alle waarden van de rij zijn een drievoud.
i. De rij bevat geen twee dezelfde waarden.
j. De rij $a(i: 0 \leq i < N)$ bevat de elementen van $X(i: 0 \leq i < N)$ in opklimmende volgorde.
k. Als de rij een 0 bevat dan ook een 1.
l. De waarde w komt in de rij voor.
m. k is de kleinste index met $X(k) = w$.
n. w is de mediaan van de rij.
o. r is de som van de array elementen met even index.
p. r is de som van de positieve getallen van de rij.
q. v is de minimale som van enige consecutieve deelrij van $X(i: 0 \leq i < N)$.
r. y is het aantal indexparen (i,j) waarvoor geldt dat de som van en met X[0] tot en zonder X[j] gelijk is aan de som van en met X[j] tot en zonder X[N].
s. r is het aantal indices i waarvoor geldt dat X(i) ten minste al zijn voorgangers in de rij is.
t. r is het aantal keren dat het maximum van de rij wordt aangenomen.

5. De structuur van een Pascalprogramma ; invoer en uitvoer.

Alvorens over te gaan tot ons eigenlijke doel, het ontwerpen van programma's, geven we een informele beschrijving van Pascal. We beperken ons hierbij tot gedeelten waarvan we gebruik zullen maken. Voor een uitgebreidere beschrijving zie [4] en [6].

Elke programmavariabele is van een bepaald type. In een programma mogen de variabelen alleen overeenkomstig hun type gebruikt worden. We onderscheiden :

Elementaire types en wel

(i) standaardtypes :

integer	de gehele getallen.
boolean	de waarden true en false.
char	de karakters '0', ..., '9', 'a', ..., 'z', 'A', ..., 'Z', '+', ';', ...

Deze zijn geordend ; de integers op de gebruikelijke manier, de verzameling boolean met false < true, de verzameling char zodanig dat de ordening tussen de letters en die tussen de cijfers als gebruikelijk is, dus '3' < '5' en 'x' < 'y'.

(ii) enumeratietypes :

de waardenverzameling wordt opgesomd.
Voorbeelden zijn (rood, geel, groen)
en (on, off).

Een enumeratietype geeft een ordening volgens de opsomming, in (rood, geel, groen) is rood < geel < groen.

(iii) subrange types :

dit zijn deelverzamelingen van een standaardtype of een enumeratietype. Een subrange is een niet-leeg aaneengesloten deel binnen de ordening. Voorbeelden zijn

0..10	de elf gehele getallen van en met 0 tot en met 10.
'a'..'d'	de karakters 'a', 'b', 'c', 'd'.

Samengestelde types en wel

(i) array types array [<index type>] of <component type>

Dit type definieert voor elke waarde van het index type een variabele van het component type. Het index type moet boolean, char, enumeratie of subrange zijn.
Voorbeelden:

var a: array [0..9] of integer, definieert een variabele bestaande uit 10 array elementen a[0] tot en met a[9].

var x: array [0..4] of boolean, definieert een variabele bestaande uit 5 array elementen a[0], a[1], a[2], a[3], a[4].

Merk op dat deze variabele x 32 verschillende waarden kan hebben.

Meerdimensionale arrays kunnen in Pascal ook aangegeven worden door meerdere indexranges te specificeren. Zo kan een 5 bij 5 matrix A van integers gedeclareerd worden door
var A : array [0..4, 0..4] of integer (dit is hetzelfde als
array [0..4] of array [0..4] of integer).

(ii) record types

```
record  <veldnaam> : <veldtype> ;
                  ⋮
        <veldnaam> : <veldtype>
end
```

Dit type definieert voor elke veldnaam een variabele van het type veldtype. Selectie van een component van een variabele van dit type gebeurt door <variabele>.<veldnaam>.
Voorbeeld:

```
var datum : record  dag   : 1..31 ;
                    maand : (jan, feb, mrt, apr, mei, jun, jul,
                             aug, sep, okt, nov, dec) ;
                    jaar  : 1900..1999
            end
```

definieert een variabele bestaande uit datum.dag, datum.maand en datum.jaar.

Pascal biedt de mogelijkheid om, voorafgaand aan de declaraties van de variabelen, in een typedeclaratie namen toe te kennen aan zelfgedefinieerde types. Voorafgaand aan typedeclaraties kunnen in een constantedeclaratie aan constanten namen worden toegekend.
Dit geeft de volgende programma opbouw :

```
program  <programmanaam> (input,output);
const    <naam> = <constante> ; ... <naam> = <constante> ;
type     <naam> = <type> ; ... <naam> = <type> ;
var      <naamlyst> : <type> ; ... <naamlijst> : <type> ;
begin
        <statement> ; ...<statement>
end.
```

Voorbeeld van een constante.declaratie :

```
const   n = 9 ; plus = '+' ;
```

Voorbeeld van een typedeclaratie :

```
type    index = 0..9 ;
        vector = array [index] of integer ;
        kleur  = (rood , geel , groen) ;
```

Voorbeeld van een variabelendeclaratie :

```
var     a,b : integer ; i : index ; x,y : vector ; c : kleur ;
        z : array [0..99] of vector ;
```

Naast de reeds besproken statements beschouwen we nog twee procedurestatements, te weten read en write. We beperken ons wat deze betreft tot een operationele beschrijving.
De input van een programma vatten we op als een geordende rij waarden. Is deze rij leeg dan geven we dit aan met input = <>. Bevat de rij bijvoorbeeld de waarden w_0, w_1 en w_2 (in deze volgorde) dan geven we dat aan met input = $\langle w_0, w_1, w_2 \rangle$. Hierbij is w_0 "de meest linkse waarde" van de inputrij.

De statement read(x) heeft tot gevolg dat x als waarde de meest linkse waarde van de inputrij krijgt en dat de inputrij de rij wordt die overblijft na verwijdering van het meest linkse element.

Ook de output vatten we op als een geordende rij. Is E een expressie dan heeft write(E) tot gevolg dat de outputrij aan de rechterkant wordt aangevuld met de waarde van E.

We sluiten dit hoofdstuk af met twee geannoteerde Pascalprogramma's waarvan de tekst voor zich spreekt. Merk daarbij op dat de relationele operatoren in Pascal als volgt genoteerd worden:

a = b	a gelijk aan b
a <> b	a verschilt van b
a < b	a kleiner dan b
a <= b	a ten hoogste b
a > b	a groter dan b
a >= b	a ten minste b

De prioriteit van de relationele operatoren is lager dan die van <u>not</u>, <u>and</u> en <u>or</u>. (In onze annotaties hebben de relationele operatoren hogere prioriteit dan $\wedge$ en $\vee$!)

Onderstaand programma bewerkstelligt bij

begintoestand : $input = \langle X_0, X_1 \rangle \wedge X_0 > 0 \wedge X_1 > 0$,

eindtoestand : $output = \langle ggd(X_0, X_1) \rangle$.

```
program ggd (input,output);
{ input = <X0,X1> ∧ X0>0 ∧ X1>0 }
var
      a,b : integer ;
begin
      read (a) ; read (b)   { a=X0 ∧ b=X1 ∧ X0>0 ∧ X1>0, ergo }
      {invariant P:  a>0 ∧ b>0 ∧ ggd (a,b) = ggd (X0,X1)
       variante functie :  vf = a+b }
      ; while  a <> b  do
       begin
          {P ∧ a≠b}
          if  a>b  then  {P ∧ a>b}  a:=a-b  {P}
                  else   {P ∧ a<b}  b:=b-a  {P}
          {P}
       end
       {P ∧ a=b, ergo  a = ggd (a,b) = ggd (X0,X1) }
       ; write (a)
       { output = < ggd (X0,X1) > }
end.
```

Het volgende programma bewerkstelligt bij

begintoestand : $input = \langle N, X(i : 0 \leq i < N) \text{ array of integer} \rangle \wedge N \geq 0$,

eindtoestand : $output = \langle (\underline{S}\, i : 0 \leq i < N : X(i)) \rangle$

```
program som (input, output);
{ input = < N, X(i: 0≤i<N) > ∧ N≥0 }
var
    n, k, s, x : integer;
begin
    read(n); k:=0; s:=0
    {invariant P:  s = (Σ i : 0≤i<k : X(i)) ∧ 0≤k≤n ∧ n=N ∧
                   input = < X(i : k≤i<N) >
     variante functie:  vf = N-k }
    ;while k<>n do
     begin
       {P ∧ k≠n , derhalve input ≠ <> }
       read(x)   { s=(Σ i : 0≤i<k : X(i)) ∧ 0≤k<n ∧ n=N ∧ x=X(k) ∧
                   input = < X(i : k+1≤i<N) > }
      ;s:=s+x    { s=(Σ i : 0≤i<k+1 : X(i)) ∧ 0≤k+1≤n ∧ n=N ∧
                   input = < X(i : k+1≤i<N) > }
       ; k:=k+1
       {P}
     end
     {P ∧ k=n , ergo  s=(Σ i : 0≤i<N : X(i)) }
     ;write(s)
     {output = <(Σ i : 0≤i<N : X(i))> }
end.
```

Opgave.

Maak een overzicht van de tot nu toe behandelde regels i.e.

regel van de assignment statement
regel van de compound statement
regel van de alternatieve statement (1)
regel van de alternatieve statement (2)
regel van C.A.R. Hoare voor de repetitieve statement
regel van E.W. Dijkstra voor de repetitieve statement .

6. Enkele strategieën.

6.0 Inleiding.

In de voorafgaande hoofdstukken zijn statements gedefinieerd met behulp van bewijsregels. In dit hoofdstuk laten we zien hoe deze regels een leidraad vormen bij het ontwikkelen van een programma. Veelal betreft het een repetitie, eventueel voorafgegaan door een initialisatie. Het gaat dan om het bepalen van een invariant en een variante functie.
In de volgende paragrafen komen een aantal strategieën aan bod. Deze worden toegelicht met voorbeelden.

6.1 Het weglaten van een conjunct.

We bekijken de volgende opgave.

Gegeven is een natuurlijk getal N.
Schrijf een programma dat aan de integer variabele a de naar beneden afgeronde waarde van de vierkantswortel uit N toekent.

Een formelere specificatie hiervoor is :

$\{N \geq 0\}$
square root
$\{a^2 \leq N \wedge (a+1)^2 > N\}$

De eindrelatie $R : a^2 \leq N \wedge (a+1)^2 > N$ is de conjunctie van twee termen. Weglating van de conjunct $(a+1)^2 > N$ levert een mogelijke invariant :

$P : a^2 \leq N \wedge a \geq 0$

Omdat $N \geq 0$ is P eenvoudig te initialiseren met $a := 0$. De ontkenning van de weggelaten term $(a+1)^2 > N$ levert de "guard" van de repetitie, te weten $(a+1)^2 \leq N$.
Dit leidt tot

$a := 0$; **while** $(a+1) * (a+1) \leq N$ **do** S .

Statement S dienen we zo te kiezen dat eindiging van de repetitie geconcludeerd kan worden

Daar a begrensd wordt door $\sqrt{N}$ ligt het verhogen van a voor de hand. Omdat tevens geldt $(a+1)^2 \leq N \equiv (a^2 \leq N)^a_{a+1}$ kiezen we voor S de statement $a := a+1$.

Dit leidt tot de volgende geannoteerde oplossing.

$\{N \geq 0\}$

$a := 0$

{invariant P : $a^2 \leq N \wedge a \geq 0$, variante functie: $vf = N - a^2$}

; <u>while</u> $(a+1)*(a+1) \leq N$ <u>do</u>

<u>begin</u>

$\{P \wedge (a+1)^2 \leq N\}$

$a := a+1$

$\{P\}$

<u>end</u>

$\{P \wedge (a+1)^2 > N$, ergo : $a^2 \leq N \wedge (a+1)^2 > N\}$.

De eindrelatie R behorend bij square root kan ook geformuleerd worden als:

a is de minimale oplossing van de vergelijking $x : x \geq 0 \wedge (x+1)^2 > N$.

Een algemener toepasbaar programma kunnen we hieruit afleiden. Zij Bx voor gehele x een boolse uitdrukking die niet voor alle natuurlijke x false is. In dat geval heeft de volgende specificatie zin :

$\{true\}$

linear search

{a is de minimale oplossing van de vergelijking $x : x \geq 0 \wedge Bx$}.

De eindrelatie behorend bij linear search noemen we weer R en geven we formeler weer.

$R: \quad a \geq 0 \wedge (\underline{A}i : 0 \leq i < a : \neg Bi) \wedge Ba$

Weglaten van de laatste term van deze conjunctie levert een invariant

$$P: \quad a \geq 0 \wedge (\underline{A} i : 0 \leq i < a : \neg B.i)$$

met initialisatie $a := 0$.
De waarde 0 voor a kan niet te groot maar wel te klein zijn en we beschouwen derhalve de statement $a := a+1$.
Berekening van P^a_{a+1} geeft

P^a_{a+1}
$= \{$substitutie$\}$
$a+1 \geq 0 \wedge (\underline{A} i : 0 \leq i < a+1 : \neg B.i)$
$\Leftarrow \{$definitie van universele quantificatie$\}$
$a \geq 0 \wedge (\underline{A} i : 0 \leq i < a : \neg B.i) \wedge \neg B.a$
$= \{$definitie van $P\}$
$P \wedge \neg B.a$

Dit levert, onder voorbehoud van eindiging, het volgende programma.

```
{true}
a := 0   {P}
while ¬B.a do
begin
    {P ∧ ¬B.a , dus  P^a_{a+1}}
    a := a + 1
    {P}
end
{P ∧ B.a , ergo    a ≥ 0 ∧ (A i : 0 ≤ i < a : ¬B.i) ∧ B.a}
```

Indien A het gevraagde antwoord is, kan de eindiging aangetoond worden met behulp van de variante functie $A - a$ (ga na). Er is geen bezwaar tegen het gebruik van het eindantwoord in de invariant of de variante functie.

Bekijken we het voorgaande programma operationeel dan zien we dat de natuurlijke getallen in opklimmende volgorde onderzocht worden. De eerste die true oplevert is dan de kleinste. Deze eigenschap staat bekend als

"The linear Search Theorem":

Om uit een collectie het kleinste element te bepalen dat aan zekere voorwaarden voldoet dienen de elementen van die collectie in opklimmende volgorde onderzocht te worden; omgekeerd levert een onderzoek in afnemende volgorde een maximum waarde. (Dit alles onder de veronderstelling dat zo'n waarde bestaat).

Opgaven.

1. In het voorgaande is een programma voor square root ontwikkeld door weglaten van de conjunct $(a+1)^2 > N$. Leid een ander programma af door weglaten van de conjunct $a^2 \leq N$.

2. Gegeven is een geheel getal N met $N \geq 1$.
   Schrijf een programma dat het grootste gehele getal bepaalt dat (i) een macht van 2 is
   en (ii) ten hoogste N is.

3. Schrijf een programma dat voor gegeven N, met $N \geq 1$, de naar beneden afgeronde waarde van $^3\log N$ bepaalt.

4. Gegeven is een integer N met $N \geq 2$ en een integer array $X(i : 0 \leq i < N)$ met $X(0) < X(1)$.
   Schrijf een programma ter bepaling van de maximale k met $0 < k < N$ en $X(k-1) < X(k)$.

6.2 Constanten vervangen door variabelen.

Wy beschouwen het volgende probleem.
Gegeven integer N met $N \geq 0$ en integer array $X(i: 0 \leq i < N)$. Schryf een programma dat aan de variabele r de som van de elementen van X toekent.

De eindconditie R van het gevraagde programma schrijven we als

$$R: \quad r = (\underline{S}\, i : 0 \leq i < N : X[i])$$

We introduceren een integer variabele n en vervangen de constante N in R door n waarbij we tevens de grenzen van n aangeven. Dit levert een mogelijk geschikte invariant P met

$$P: \quad r = (\underline{S}\, i : 0 \leq i < n : X[i]) \wedge 0 \leq n \leq N$$

Initialisatie is eenvoudig ; ga na dat $\{N \geq 0\}\ n := 0\ ;\ r := 0\ \{P\}$.
Verder geldt $P \wedge n = N \Rightarrow R$, een geschikte guard voor de repetitie is de ontkenning van $n = N$, te weten $n \neq N$.
Om eindiging te kunnen concluderen zullen we n verhogen. We berekenen P^n_{n+1}.

P^n_{n+1}
$= \{$ definitie P $\}$
$r = (\underline{S}\, i : 0 \leq i < n+1 : X[i]) \wedge 0 \leq n+1 \leq N$
$\Leftarrow \{$ definitie sommatie $\}$
$r = (\underline{S}\, i : 0 \leq i < n : X[i]) + X[n] \wedge 0 \leq n < N$

Indien we in de laatste regel r vervangen door $r + X[n]$ staat er juist $P \wedge n \neq N$ waaruit we concluderen

$$P \wedge n \neq N \Rightarrow (P^n_{n+1})^r_{r+X[n]}$$

ofwel $\{P \wedge n \neq N\}\ r := r + X[n]\ ;\ n := n+1\ \{P\}$

Dit leidt tot het volgende geannoteerde programma.

```
{ N≥0 }
n:=0 ; r:=0
{invariant  P :   r = (Σ i : 0≤i<n : X[i]) ∧ 0≤n≤N
 variante functie:  vf = N-n }
; while n <> N do
 begin
    {P ∧ n≠N}
    r := r + X[n]
    ;n := n+1
    {P}
 end
 {P ∧ n=N, ergo  r = (Σ i : 0≤i<N : X[i]) }
```

Vervanging van de constante 0 door n in de eindrelatie R levert invariant Q met

Q : $r = (\Sigma i : n \leq i < N : X[i]) \wedge 0 \leq n \leq N$

Onderstaande programma's zijn door Q geïnspireerd. Geef zelf een correctheidsbewijs en voorzie de programma's van geschikte annotatie.

```
n := N ; r := 0
; while n <> 0 do
 begin
   r := r + X[n-1]
   ;n := n-1
 end
```

```
n := N ; r := 0
; while n <> 0 do
 begin
    n := n-1
    ; r := r + X[n]
 end
```

Als tweede voorbeeld beschouwen we nogmaals square root uit 6.1 met eindrelatie

$R: \quad a^2 \leq N < (a+1)^2$

We vervangen de constante 1 door variabele c en proberen invariant

$P: \quad a^2 \leq N < (a+c)^2$

Initialisatie is eenvoudig : $\{N \geq 0\}\ a:=0\ ;\ c:=N+1\ \{P\}$.
Verder geldt $P \wedge c=1 \Rightarrow R$.

Dit levert de volgende oplossing:

```
a:=0 ; c:=N+1   {P}
;while c<>1 do
begin
   {P ∧ c>1}
   "verlaag c onder invariantie van P"
   {P}
end
```

Voor het verlagen van c is te denken aan $c:=c-1$. Een efficiënter algoritme is te verwachten indien we c halveren.
Berekening van $P^c_{c \,\underline{div}\, 2}$ levert $a^2 \leq N < (a + c \,\underline{div}\, 2)^2$, dus

$\{P \wedge (a + c \,\underline{div}\, 2)^2 > N\}\ c := c \,\underline{div}\, 2\ \{P\}$

Voor het geval $P \wedge (a + c \,\underline{div}\, 2)^2 \leq N$ ligt naast $c := c \,\underline{div}\, 2$ het verhogen van a met $c \,\underline{div}\, 2$ voor de hand. Berekening van $(P^a_{a+c})^c_{c \,\underline{div}\, 2}$ levert $(a + c \,\underline{div}\, 2)^2 \leq N < (a + 2 \cdot c \,\underline{div}\, 2)^2$ (ga na!).

Uit $N < (a+c)^2$ kan echter niet geconcludeerd worden dat $N < (a + 2 \cdot c \,\underline{div}\, 2)^2$, daar voor positieve oneven c geldt $2 \cdot (c \,\underline{div}\, 2) < c$.
Er staan twee wegen open voor een oplossing van dit probleem:

(i) Neem in het geval $P \wedge (a + c \,\underline{div}\, 2)^2 \leq N$ niet $c := c \,\underline{div}\, 2\ ;\ a := a + c$ maar $a := a + c \,\underline{div}\, 2\ ;\ c := c - c \,\underline{div}\, 2$.

(ii) Draag er zorg voor dat c even is, indien $c > 1$.

We werken de tweede oplossing uit door de oorspronkelijke invariant te versterken tot

$P: \quad a^2 \leq N < (a+c)^2 \;\wedge\; (\underline{E}\, i : i \geq 0 : c = 2^i)$

Het linear Search Theorem helpt bij de initialisatie:
$\{N \geq 0\}$ c:=1 ; <u>while</u> $c*c \leq N$ <u>do</u> c:=c+c $\{P\}$.
Onderstaand treft u het geannoteerde programma aan. Geef zelf een bewijs voor de eindiging.

```
{N≥0}
c:=1 ; while c*c ≤ N do c:=c+c ; a:=0
{invariant   P: a² ≤ N < (a+c)² ∧ (E i : i≥0 : c = 2^i)
 variante functie: vf = c }
; while  c<>1  do
 begin
   { P ∧ c>1 }
   c:= c div 2    { a² ≤ N < (a+2·c)² ∧ (E i : 0≤i : c = 2^i) }
  ; if (a+c)*(a+c) ≤ N  then  a:= a+c
   {P}
 end
```

In het eerste programma dat wij ontwierpen in 6.1 is het aantal benodigde slagen gelijk aan $\sqrt{N}$. In het bovenstaande programma is het aantal benodigde slagen evenredig met $^2\log N$.

In de eindrelatie $R: \quad a^2 \leq N < (a+1)^2$ was de constante 1 vervangen door de variabele c. In plaats hiervan kan ook a+1 vervangen worden door een variabele b leidend tot invariant Q met

$Q: \quad a^2 \leq N < b^2$, met ongeannoteerd programma:

```
a:=0 ; b:=N+1
;while  a+1 <> b  do
 begin
   x:= (b+a) div 2
  ;if  x*x ≤ N  then  a:= x  else  b:=x
 end
```

Het volgende voorbeeld is een generalisatie van het algoritme voor square root en luidt:

Gegeven een integer N met $N \geq 0$, een niet-dalend ("ascending") array $X[i: 0 \leq i < N)$ en een integer a.
Gevraagd wordt een programma te schrijven dat aan de boolean variabele p de waarde $(\underline{E} i : 0 \leq i < N : a = X[i])$ toekent.

(Om misverstanden te voorkomen spreken we af: een rij $X[i: 0 \leq i < N)$ heet

increasing	als	$(\underline{A} i : 0 < i < N : X(i-1) < X(i))$
decreasing	als	$(\underline{A} i : 0 < i < N : X(i-1) > X(i))$
ascending	als	$(\underline{A} i : 0 < i < N : X(i-1) \leq X(i))$
descending	als	$(\underline{A} i : 0 < i < N : X(i-1) \geq X(i))$).

Komt a voor in de rij dan zal dit waarschijnlijk vastgesteld worden door de bepaling van i met $0 \leq i < N$ zodanig dat $a = X[i]$.
Komt a niet voor in de rij dan kan i aangeven "waar a hoort te staan", dat wil zeggen $X[i] < a < X[i+1]$.
Combinatie van deze beschouwing geeft een mogelijk einddoel, te weten

$$X[i] \leq a < X[i+1]$$

Deze uitspraak is te sterk daar a kleiner dan alle elementen of groter dan alle elementen kan zijn.
We lossen dit op door in gedachten (dat wil zeggen in onze uitspraken, maar niet in de programmatekst zelf) de rij uit te breiden met $X[-1] = -\infty$ en $X[N] = +\infty$.
We stevenen af op de relatie

$$R: \quad X[i] \leq a < X[i+1] \quad \wedge \quad -1 \leq i < N$$

Als R eenmaal geldt dan kent, omdat de rij X ascending is, de volgende statement aan p de juiste waarde toe.

```
if i=-1 then p:= false else p:= (a= X[i])
```

(qa dit na).

Uit R leiden we invariant P af met

P: $-1 \leq i < j \leq N \quad \wedge \quad X[i] \leq a < X[j]$.

Initialisatie is eenvoudig: $\{N \geq 0\}$ i:=-1 ; j:=N $\{P\}$. Verder geldt $P \wedge i+1=j \Rightarrow R$. Hieruit leiden we het volgende programma af.

```
i:=-1 ; j:=N  {P}
;while  i+1 <> j  do
 begin
      "verminder  j-i  onder invariantie van P"
 end
 ;if  i=-1  then  p:=false  else  p:=(a=X[i])
```

Indien $P \wedge i+1 \neq j$ dan is er een integer h met $i<h<j$ (en derhalve $0 \leq h < N$). Gegeven zo'n h dan wordt j-i zowel door i:=h als door j:=h vermindert.
Uit $P^i_h \equiv -1 \leq h < j \leq N \wedge X[h] \leq a < X[j]$ volgt $P \wedge X[h] \leq a \Rightarrow P^i_h$
even zo geldt (ga na) $P \wedge a < X[h] \Rightarrow P^j_h$

Derhalve kan bij gegeven h met $i<h<j$ j-i verminderd worden onder invariantie van P middels
if $X[h] \leq a$ then i:=h else j:=h .

De keuze van h is nog vrij. Kiezen we h:=i+1 of h:=j-1 dan zal het aantal repetitieslagen N bedragen.
Een betere keuze is h:=(i+j) div 2 waardoor in elke slag j-i ongeveer gehalveerd wordt. Het aantal slagen is dan van de orde $^2\log N$.

We krijgen zo het volgende programma, bekend onder de naam Binary Search.

```
{N≥0 , X(i: 0≤i<N) is ascending, X[-1]=-∞ en X[N]=∞ }
i:=-1 ; j:=N
{invariant P : -1 ≤ i < j ≤ N  ∧  X[i] ≤ a < X[j]
 variante functie : j-i }
;while j<>i+1 do
 begin
    h:= (i+j) div 2     { P ∧ i<h<j , dus ook 0≤h<N }
   ;if X[h] ≤ a then i:=h else j:=h    {P}
 end
{P ∧ i+1=j , ergo  X[i] ≤ a < X[i+1]  ∧  -1 ≤ i < N }
;if i=-1 then p:=false else p:= (X[i]=a)
{p ≡ (E t : 0≤t<N : a=X[t])}
```

We besluiten dit voorbeeld met enkele vragen cq opmerkingen.

- Waar wordt in dit algoritme gebruik gemaakt van het feit dat de rij X(i: 0≤i<N) gesorteerd is ?

- Wat verwacht u van de "versnelling" door vervanging van

  ```
  if X[h] ≤ a then i:=h else j:=h     door
  ```

  ```
  if X[h]=a then begin i:=h ; j:=h+1 end
            else if X[h] ≤ a then i:=h else j:=h     ?
  ```

- De variabele h wordt lokaal gebruikt, alleen binnen de compound statement. De taal Pascal staat helaas niet toe dat een dergelijke variabele lokaal gedeclareerd wordt. (h komt niet voor in P !).

- Bewijs $0 \le i+1 < j \Rightarrow i < (i+j) \text{ div } 2 < j$

Opgaven.

1. De rij $F(N: N \geq 0)$ van Fibonacci is gedefinieerd door
$F0 = 0$, $F1 = 1$ en $F(i+2) = Fi + F(i+1)$ voor $i \geq 0$.
Schrijf een programma dat voor gegeven natuurlijk getal k
Fk berekent.

2. Gegeven zijn integer N met $N \geq 1$ en integer array $X(i: 0 \leq i < N)$.
Schrijf een programma dat aan de boolean variabele b de
waarde " $X(i: 0 \leq i < N)$ is ascending " toekent.

3. Gegeven zijn integer N met $N \geq 1$ en integer array $X(i: 0 \leq i < N)$.
$X(i: 0 \leq i < N)$ is ascending.

Een consecutieve deelrij $X(i: p \leq i < q)$ is een plateau ter lengte
$q-p$ indien $(\underline{A}\, i,j: p \leq i \leq j < q: X[i] = X[j])$.

Maak een programma ter bepaling van de grootste lengte van
enig plateau.

4. (Zie opgave 3). Gegeven integer N met $N \geq 1$ en ascending array
$X(i: 0 \leq i < N)$.
Maak een programma ter bepaling van het aantal plateaus van
X.

5. Gegeven integer N met $N \geq 0$ en integer array $X(i: 0 \leq i < N)$ met
$(\underline{A}\, i: 0 \leq i < N: X[i] = 0 \vee X[i] = 1)$.
Schrijf een programma dat aan boolean variabele b een waarde
toekent zodanig dat
$b \equiv (\underline{E}\, n: 0 \leq n \leq N: (\underline{A}\, i: 0 \leq i < n: X[i] = 0) \wedge (\underline{A}\, i: n \leq i < N: X[i] = 1))$

6. Gegeven integer N met $N \geq 0$, integer array $X(i: 0 \leq i < N)$ en integer a.
schrijf een programma dat bewerkstelligt
$R: \quad s = (\underline{S}\, i: 0 \leq i < N: X[i] \cdot a^{N-i})$

7. Als opgave 6 met $s = (\underline{S}\, i: 0 \leq i < N: X[i] \cdot a^{i})$

6.3 Uitbreiden van het domein van variabelen.

We beschouwen het volgende probleem. Gegeven zijn twee ascending integer functies $f(i: i \geq 0)$ en $g(i: i \geq 0)$. Het is bekend dat f en g een waarde gemeen hebben, i.e. $(\underline{E}\ i,j : i \geq 0 \wedge j \geq 0 : f(i) = g(j))$. Gevraagd wordt een programma te schrijven dat aan integer r de kleinste gemeenschappelijke waarde van f en g toekent.

De eindrelatie kunnen we schrijven als

$$R: \quad r = (\underline{MIN}\ i,j : i \geq 0 \wedge j \geq 0 \wedge f(i) = g(j) : f(i))$$

Wellicht is het eenvoudiger om aan te sturen op de relatie R' met

$$R': \quad p = x \wedge q = y$$

waarbij p en q integer variabelen zijn en x en y de kleinste natuurlijke getallen zijn met $f(x) = g(y)$. (Waarom bestaan x en y?). Daar $\{R'\}\ r := f(p)\ \{R\}$ is met R' ons doel bereikt.

Uit R' leiden we een invariant P af door het domein van p en q te vergroten:

$$P: \quad 0 \leq p \leq x \wedge 0 \leq q \leq y$$

Initialisatie is eenvoudig: $\{x \geq 0 \wedge y \geq 0\}\ p := 0 ; q := 0\ \{P\}$ en $P \wedge f(p) = g(q) \Rightarrow R'$ levert als guard van een repetitie $f(p) \neq g(q)$.

Als mogelijke statements beschouwen we $p := p+1$ en $q := q+1$. Berekening van P^P_{p+1} geeft

$$\begin{aligned}
& P^P_{p+1} \\
= & \{\text{def. } P\} \\
& 0 \leq p+1 \leq x \wedge 0 \leq q \leq y \\
\Leftarrow & \{\text{def. } P\} \\
& P \wedge p \neq x
\end{aligned}$$

$p \neq x$ kunnen we niet als boolean expressie gebruiken (Waarom niet?). $p \neq x$ betekent dat $f(p)$ niet het kleinste gemeenschappelijk element is. We kunnen $p \neq x$ concluderen uit $f(p) < g(q)$:

$P \wedge f(p) < g(q)$

$= \{$ definitie $P\}$

$0 \le p \le x \wedge 0 \le q \le y \wedge f(p) < g(q)$

$\Rightarrow \{ g$ is ascending $\}$

$0 \le p \le x \wedge 0 \le q \le y \wedge f(p) < g(y)$

$\Rightarrow \{ g(y) = f(x) \}$

$0 \le p \le x \wedge 0 \le q \le y \wedge f(p) < f(x)$

$\Rightarrow \{$ algebra $\}$

$0 \le p \le x \wedge 0 \le q \le y \wedge p \neq x$

$\Rightarrow \{$ algebra $\}$

$0 \le p+1 \le x \wedge 0 \le q \le y$

$= \{$ definitie $P\}$

P^p_{p+1}

Symmetrie levert $P \wedge g(q) < f(p) \Rightarrow P^q_{q+1}$ en we krijgen het volgende ongeannoteerde programma.

```
p := 0 ; q := 0
; while f(p) <> g(q) do if f(p) < g(q) then p := p+1 else q := q+1
; r := f(p)
```

Geef zelf de geannoteerde versie (wat kiest u als variante functie?).

Opgave.

1. Gegeven drie ascending integer functies $f(i : i \ge 0)$, $g(i : i \ge 0)$ en $h(i : i \ge 0)$ met de eigenschap $(\underline{E}\ i,j,k : i \ge 0 \wedge j \ge 0 \wedge k \ge 0 : f(i) = g(j) = h(k))$. Schrijf een programma dat de kleinste gemeenschappelijke waarde van f, g en h berekent.

   (Opmerking: bovenstaand probleem staat bekend onder de namen "de f,g,h-rijen" en "The Welfare Crook").

7. Niet-standaard toepassingen van de strategieën.

Bij het toepassen van de in hoofdstuk 6 behandelde strategieën staan dikwijls veel keuzen open. In dit hoofdstuk geven we voor bepaalde typen problemen een mogelijk geschikte keuze. We lichten deze toe met voorbeelden.

Als eerste bekijken we een probleem bekend onder de naam Coincidence Count:

> Gegeven zijn integers N en M met $N \geq 0$ en $M \geq 0$ en twee increasing integer arrays $f(i : 0 \leq i < N)$ en $g(i : 0 \leq i < M)$. Gevraagd wordt een programma dat het aantal coincidenties van f en g bepaalt, i.e. dat aan integer variabele r het aantal gemeenschappelijke waarden van f en g toekent.

De gewenste eindrelatie formuleren we als

$$R: \quad r = (\underline{N}\, i,j : 0 \leq i < N \wedge 0 \leq j < M : f(i) = g(j))$$

Om de symmetrie niet te verstoren vervangen we zowel N als M door een variabele. Dit levert invariant P met

$$P: \quad r = (\underline{N}\, i,j : 0 \leq i < p \wedge 0 \leq j < q : f(i) = g(j)) \wedge 0 \leq p \leq N \wedge 0 \leq q \leq M$$

Uit $P \wedge p = N \wedge q = M \Rightarrow R$ leiden we de guard van de repetitie af: $p \neq N \vee q \neq M$.
Merk op dat met deze guard binnen de repetitie niet zonder meer aan $f(p)$ of $g(q)$ gerefereerd kan worden.

Initialisatie van P kan op velerlei manieren, zoals $p := 0 ; q := M ; r := 0$ of $p := N ; q := 0 ; r := 0$. Wederom om reden van symmetrie kiezen we $p := 0 ; q := 0 ; r := 0$.

We merken op dat voor $0 \leq p < N \wedge 0 \leq q < M$, daar f en g increasing zijn, geldt

$$\begin{aligned} f(p) = g(q) \Rightarrow\ & (\underline{N}\, i,j : 0 \leq i < p \wedge j = q : f(i) = g(j)) = 0 \ \wedge \\ & (\underline{N}\, i,j : i = p \wedge 0 \leq j < q : f(i) = g(j)) = 0 \ \wedge \\ & (\underline{N}\, i,j : i = p \wedge j = q : f(i) = g(j)) = 1 \end{aligned}$$

ofwel

$$f(p) = g(q) \Rightarrow (\underline{N}\, i,j : 0 \le i < p+1 \wedge 0 \le j < q+1 : f(i) = g(j)) = (\underline{N}\, i,j : 0 \le i < p \wedge 0 \le j < q : f(i) = g(j)) + 1$$

en derhalve (ga na)

$$P \wedge p \neq N \wedge q \neq M \wedge f(p) = g(q) \Rightarrow ((P^{p}_{p+1})^{q}_{q+1})^{r}_{r+1}$$

We onderzoeken het geval $P \wedge p \neq N \wedge q \neq M \wedge f(p) \neq g(q)$.
Is $f(p) < g(q)$ dan $(\underline{A}\, i : 0 \le i \le p : f(i) < g(q))$ en verhogen van q met 1 laat P invariant. Door verhogen van q met 1 wordt $f(p) < g(q)$ niet verstoord en even zinnig, zo niet zinniger, is $q := M$ in plaats van $q := q+1$. Daar vroeg of laat p verhoogd moet worden zien we niet veel heil in verhoging van q in het geval $P \wedge p \neq N \wedge q \neq M \wedge f(p) < g(q)$.

De vraag is of we p in dit geval kunnen verhogen zonder P te verstoren.
Uit $(N\, i,j : 0 \le i < p+1 \wedge 0 \le j < q : f(i) = g(j)) = (\underline{N}\, i,j : 0 \le i < p \wedge 0 \le j < q : f(i) = g(j)) + (N\, j : 0 \le j < q : f(p) = g(j))$
volgt dat dit kan mits $(N\, j : 0 \le j < q : f(p) = g(j)) = 0$ hetgeen wordt geïmpliceerd door $f(p) > g(q-1)$ dank zij de monotoniciteit van g.
Een analoge beschouwing van het geval $f(p) > g(q)$ levert een neveneis $g(q) > f(p-1)$.

We breiden daarom onze invariant uit tot P_1 met

$$P_1 : r = (N\, i,j : 0 \le i < p \wedge 0 \le j < q : f(i) = g(j)) \wedge 0 \le p \le N \wedge 0 \le q \le M \wedge f(p-1) < g(q) \wedge g(q-1) < f(p)$$

waarbij we $f(-1) = g(-1) = -\infty$ en $f(N) = g(M) = +\infty$ definiëren.

Een eenvoudige initialisatie van P_1 is opgelegd: $p := 0 ; q := 0 ; r := 0$.
Verder geldt dank zij het increasing zijn van f en g dat
$P_1 \wedge (p = N \vee q = M) \Rightarrow R$ (ga na).
Dit levert een sterkere guard voor de repetitie: $p \neq N \wedge q \neq M$.

De lezer wordt uitgenodigd om te bewijzen

$$P_1 \wedge p \neq N \wedge q \neq M \wedge f(p) = g(q) \Rightarrow ((P_1{}^{p}_{p+1})^{q}_{q+1})^{r}_{r+1} \quad \text{en}$$
$$P_1 \wedge p \neq N \wedge q \neq M \wedge f(p) < g(q) \Rightarrow P_1{}^{p}_{p+1}$$

waarna de volgende oplossing geannoteerd kan worden.

```
p:=0 ; q:=0 ; r:=0
;while (p <> N) and (q<> M) do
begin
   if f[p]=g[q] then begin r:=r+1 ; q:=q+1 ; p:=p+1 end else
   if f[p]<g[q] then p:=p+1 else
   if f[p]>g[q] then q:=q+1
end
```

De eenvoud van het programma vormt een schril contrast met de complexiteit van de invariant en de afleiding daarvan. Er blijkt zoals we direct zullen zien een eenvoudiger invariant P_2 te bestaan die tot hetzelfde programma leidt.

Voor het vinden hiervan bekijken we de integer functie F gedefinieerd door

$$F(i,j) = f(i) - g(j) \qquad (0 \leq i < N \;,\; 0 \leq j < M)$$

F is increasing in het eerste argument en decreasing in het tweede argument : $(\underline{A}\, j : 0 \leq j < M : (\underline{A}\, i : 0 < i < N : F(i-1,j) < F(i,j))) \wedge$
$(\underline{A}\, i : 0 \leq i < N : (\underline{A}\, j : 0 < j < M : F(i,j-1) > F(i,j)))$

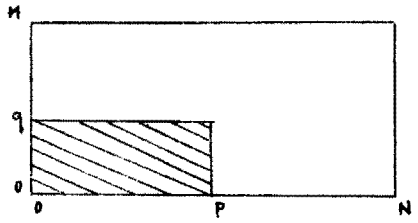
De eindrelatie R luidt in termen van F

$$R : \quad r = (\underline{N}\, i,j : 0 \leq i < N \wedge 0 \leq j < M : F(i,j) = 0)$$

en invariant P

$$P : \quad r = (N\, i,j : 0 \leq i < p \wedge 0 \leq j < q : F(i,j) = 0) \wedge 0 \leq p \leq N \wedge 0 \leq q \leq M$$

We kunnen P weergeven als in onderstaand plaatje.



" r = het aantal nulpunten van F in het gearceerde gebied ".

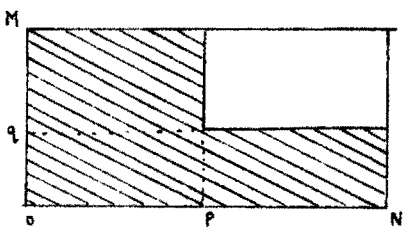
Versterking van P tot P_1 leverde

P_1 : $r = (\underline{N}\, i,j : 0 \le i < p \wedge 0 \le j < q : F(i,j) = 0) \wedge 0 \le p \le N \wedge 0 \le q \le M \wedge F(p-1, q) < 0 \wedge F(p, q-1) > 0$

$F(p-1, q) < 0$ impliceert $(\underline{N}\, i,j : 0 \le i < p \wedge q \le j < N : F(i,j) = 0) = 0$ en
$F(p, q-1) > 0$ impliceert $(\underline{N}\, i,j : p \le i < N \wedge 0 \le j < q : F(i,j) = 0) = 0$
Derhalve impliceert P_1 de invariant P_2 met

P_2 : $r + (\underline{N}\, i,j : p \le i < N \wedge q \le j < M : F(i,j) = 0) = (\underline{N}\, i,j : 0 \le i < N \wedge 0 \le j < M : F(i,j) = 0) \wedge 0 \le p \le N \wedge 0 \le q \le M$

We kunnen P_2 weergeven als in onderstaand plaatje.



" r = het aantal nulpunten van F in het gearceerde gebied ".

Definiëren we voor $0 \le p \le N$ en $0 \le q \le M$ $N(p,q)$ door
$N(p,q) = (\underline{N}\, i,j : p \le i < N \wedge q \le j < M : F(i,j) = 0)$ dan laten R en P_2 zich schrijven als

R : $r = N(0,0)$
P_2 : $r + N(p,q) = N(0,0) \wedge 0 \le p \le N \wedge 0 \le q \le M$

Er geldt $\{N \geq 0 \wedge M \geq 0\}\ p := 0\ ;\ q := 0\ \{P_2\}$

en $P_2 \wedge (p = N \vee q = M) \Rightarrow R$

Met behulp van

$$0 \leq p < N \wedge 0 \leq q < M \Rightarrow N(p,q) = (\underline{N}\ j : q \leq j < M : F(p,j) = 0) + N(p+1,q) \wedge$$
$$N(p,q) = (\underline{N}\ i : p \leq i < N : F(i,q) = 0) + N(p,q+1)$$

vindt u de gevonden oplossing op veel eenvoudiger wijze terug.

De keuze van P_2 in plaats van P noemt men wel "winkelhaak in plaats van rechthoek", refererend aan de plaatjes.

Refererend aan de formulering van P_2, populair uitgedrukt als "wat al berekend is" + "wat nog berekend moet worden" = "wat te berekenen is", spreekt men van "tail recursion". Bij tail recursion kan de + uit de vorige regel ook een andere operator zijn.

Als laatste nog een moraal.

> Is in een programmeerprobleem sprake van een integer functie van twee argumenten, stijgend in het ene en dalend in het andere argument, dan levert tail recursion een goede kans op een eenvoudig en efficiënt programma.
>
> Betrapt men zich bij een gekozen invariant op een grote mate van vrijheid bij de initialisatie hiervan, onderzoek dan andere mogelijkheden voor een invariant.

We passen tail recursion toe op het volgende probleem.

Gegeven integers X en Y met $X \geq 0$ en $Y \geq 0$. Gevraagd wordt een programma dat aan de integer variabele r een dusdanige waarde toekent dat voldaan is aan

$R: \quad r = X^Y$ (we spreken af $0^0 = 1$).

Tail recursion levert een invariant van de vorm
$r \cdot$ "wat nog berekend moet worden" $= X^Y$.
Dit leidt tot de introductie van integer variabelen x en y en invariant P met

$$P: \quad r \cdot x^y = X^Y \wedge x \geq 0 \wedge y \geq 0$$

Initialisatie is eenvoudig: $\{X \geq 0 \wedge Y \geq 0\}$ $x := X ; y := Y ; r := 1$ $\{P\}$
en verder geldt $P \wedge y = 0 \Rightarrow r = X^Y$.

We krijgen zo een programma van de volgende structuur:

```
{ X≥0 ∧ Y≥0 }
x := X ; y := Y ; r := 1 { P }
; while y <> 0 do
  begin
     { P ∧ y > 0 }
     "verlaag y onder invariantie van P"
     { P }
  end
{ P ∧ y = 0 , ergo r = X^Y }
```

Uit $y > 0 \Rightarrow (r \cdot x) \cdot x^{y-1} = x^y$ volgt $\{P \wedge y > 0\}$ $r := r * x ; y := y - 1$ $\{P\}$
Dit levert een oplossing van de orde Y; de machtsverheffing wordt gerealiseerd door herhaald vermenigvuldigen.
Een efficiëntere oplossing verkrijgen we door te bedenken dat voor even y geldt $x^y = (x * x)^{y \,\underline{\text{div}}\, 2}$. Het resultaat wordt

```
{X≥0 ∧ Y≥0}
x:= X ; y:= Y ; r:=1
{invariant P:  r.x^y = X^Y  ∧  x≥0  ∧  y≥0  , variante functie: y }
; while  y <> 0  do
 begin
    {P ∧ y≠0 }
    if  y mod 2 = 0  then  begin  y:= y div 2 ; x:= x * x  end
                     else  begin  y:= y-1 ;  r:= r * x  end
    {P}
 end
 {P ∧ y=0 , ergo  r = X^Y }
```

Merk op dat het aantal repetitieslagen ten hoogste $1 + {}^2\log(Y+1)$ is.

Met nadruk wijzen we erop dat de aangedragen strategieën een mogelijk geschikte invariant opleveren. Veelal dient naast het toepassen van de strategieën nog denkwerk verricht te worden voor het vinden van een uiteindelijk bevredigende invariant en een uiteindelijk bevredigend algoritme.

Een andere manier om een geschikte invariant te vinden is de volgende. Schrijf "het gevraagde" om tot een gelijkheid of een equivalentie en vervang in een van de leden hiervan constanten door variabelen.
Een voorbeeld hiervan is het berekenen van de grootste gemene deler van twee positieve getallen.
De specificatie hiervan is $\{ X>0 \wedge Y>0 \}$ ggd $\{ r = ggd(X,Y) \}$.

We leiden een invariant P af door in $ggd(X,Y) = ggd(X,Y)$ de constanten in het linkerlid te vervangen door variabelen:

P: $ggd(x,y) = ggd(X,Y) \wedge x>0 \wedge y>0$

Initialisatie: $\{ X>0 \wedge Y>0 \}\ x := X\ ;\ y := Y\ \{P\}$,
en er geldt $\{ P \wedge x=y \}\ r := x\ \{ r = ggd(X,Y) \}$. Met behulp hiervan vinden we de oplossing voor ggd uit paragraaf 4.0.

Een tweede voorbeeld van deze techniek is de reeds besproken Binary Search.

Gegeven integer N, $N \geq 1$, een ascending integer array $X[i : 0 \leq i < N]$ en integer a. Gevraagd wordt aan boolean variabele b een dusdanige waarde toe te kennen dat R geldt met

R: $b \equiv (\underline{E}\, i : 0 \leq i < N : a = X[i])$

Toepassen van genoemde techniek levert invariant P met

P: $(\underline{E}\, i : p \leq i < q : a = X[i]) \equiv (\underline{E}\, i : 0 \leq i < N : a = X[i]) \wedge 0 \leq p < q \leq N$

Initialisatie $\{ N \geq 1 \}\ p := 0\ ;\ q := N\ \{P\}$,
en er geldt $\{ P \wedge p+1 = q \}\ b := (a = X[p])\ \{R\}$.

Met behulp hiervan vindt men een zelfde programma als in paragraaf 6.2.

Opgaven.

Deze en toekomstige opgaven zijn niet meer zo netjes geordend naar strategie.

1. Gegeven zijn gehele getallen N en M met $N \geq 0$ en $M \geq 0$ en twee integer arrays $f(i: 0 \leq i < N)$ en $g(i: 0 \leq i < M)$. f is increasing en g is decreasing.
   Schrijf een programma ter berekening van het aantal coïncidenties van f en g.

2. Gegeven zijn een integer N, $N \geq 0$, een ascending integer array $X(i: 0 \leq i < N)$ en een integer LL met $LL \geq 0$.
   Schrijf een programma dat aan integer variabele r een dusdanige waarde toekent dat voldaan wordt aan

   $$R: \quad r = (\underline{N}\ i,j : 0 \leq i \leq j < N : X[j] - X[i] \leq LL) .$$

3. De functie fusc is op $\mathbb{N}$ gedefinieerd door

   $fusc(0) = 0$, $fusc(1) = 1$ en voor $n \geq 0$: $fusc(2 \cdot n) = fusc(n)$
   $fusc(2 \cdot n + 1) = fusc(n) + fusc(n+1)$

   a. Bereken $fusc(19)$
   b. Schrijf een programma dat gegeven N, $N \geq 0$, de waarde van $fusc(N)$ berekent.

4. Gegeven zijn integer N, $N \geq 0$, en integer array $X(i: 0 \leq i < N)$ met $(\underline{A}\ i: 0 \leq i < N : X(i) > 0)$.
   Schrijf een programma ter berekening van
   $(\underline{N}\ p,q : 0 \leq p \leq N \wedge 0 \leq q \leq N : (\underline{S}\ i : 0 \leq i < p : X(i)) = (\underline{S}\ i : q \leq i < N : X(i)))$.

5. Gegeven zijn integer N, $N \geq 0$, en integer array $X(i: 0 \leq i < N)$.
   Schrijf een programma ter berekening van

   $$(\underline{N}\ p,q : 0 \leq p < q < N : X(p) = 0 \wedge X(q) = 1) .$$

6. Een rij heet van de categorie h indien
   - de rij bestaat uit één 0
   of - de rij bestaat uit een 1 gevolgd door 2 rijen van de categorie h

   (Syntactisch uitgedrukt: $\langle h \rangle ::= 0 \mid 1 \langle h \rangle \langle h \rangle$)

   Gegeven is een integer N, $N \geq 0$, en een integer array $X(i : 0 \leq i < 2N+1)$ met $(\underline{A}\, i : 0 \leq i < 2N+1 : X(i) = 0 \vee X(i) = 1)$.
   Schrijf een programma dat aan boolean variabele b een dus. danige waarde toekent dat voldaan is aan R met

   R: $b \equiv X$ is een rij van de categorie h.

7. Gegeven is een integer functie $f(i,j : i \geq 0 \wedge j \geq 0)$. f is increasing in beide argumenten.

   a. Schrijf een programma ter berekening van $(\underline{N}\, i,j : i \geq 0 \wedge j \geq 0 : f(i,j) \leq 1)$.

   b. Schrijf een programma ter berekening van $(\underline{N}\, i,j : i \geq 0 \wedge j \geq 0 : f(i,j) = 1)$.

8. Gegeven een integer N met $N \geq 0$. Schrijf een programma ter berekening van het aantal manieren waarop N als som van twee kwadraten kan worden geschreven, i.e. van $(\underline{N}\, x,y : 0 \leq x \leq y : x^2 + y^2 = N)$.

9. Gegeven zijn twee integers X en Y met $X \geq 0$ en $Y \geq 0$.
   Schrijf een programma dat $X \cdot Y$ berekent zonder gebruik te maken van de operator *.
   Wel kan gebruik gemaakt worden van mod 2 en div 2.

8. Het gebruik van arrays.

Tot nog toe werd in onze programmateksten niet met arrays gemanipuleerd ; zij werden hoogstens geïnspecteerd. Zoals eerder vermeld vatten wij een array op als één variabele , gekenmerkt door domein en componentwaarden.
Het array x gedeclareerd door var x : array [0..4] of integer heeft domein (i : 0≤i<5) , hetgeen we ook aangeven met dom(x) = (i : 0≤i<5). Variabele x kan gewijzigd worden door het wijzigen van een componentwaarde.
Door de statement " k := 0 ; while k<>0 do begin x[k] := 0 ; k := k+1 end " wordt aan x de waarde (0,0,0,0,0) toegekend.
Wordt daarna de statement x[1] := 1 uitgevoerd dan wijzigt de waarde van x in (0,1,0,0,0).
Tot zover dit operationele beeld.

Is x een array en behoort k tot dom(x) dan noteren we het array y met $\begin{cases} y[i] = x[i] & \text{als } i \in dom(x) \setminus \{k\} \\ y[k] = E \end{cases}$, en dom(y) = dom(x)
als (x ; k : E).

Dus voor i in dom(x) geldt (x ; k : E)[i] = x[i] als i ≠ k
(x ; k : E)[i] = E als i = k .

De array-assignment regel luidt

Als $P \Rightarrow k \text{ in } dom(x) \wedge Q^{x}_{(x;k:E)}$

dan {P} x[k] := E {Q}.

Ter illustratie tonen we aan

{ X[p] = a ∧ X[q] = b }
z := X[p] ; X[p] := X[q] ; X[q] := z
{ X[p] = b ∧ X[q] = a }

Uit $(X[q]=a \wedge X[p]=b)^{X}_{(X;q:z)}$

$= \{ \text{def. } (X;q:z) \}$

$(z=a \wedge X[p]=b)$

en $(z=a \wedge X[p]=b)^{X}_{(X;p:X[q])}$

$= \{ \text{def. } (X;p:X[q]) \}$

$(z=a \wedge X[q]=b)$

en $(z=a \wedge X[q]=b)^{z}_{X[p]}$

$= \{ \text{subst.} \}$

$(X[p]=a \wedge X[q]=b)$

volgt het gestelde (gaat u dit nauwgezet na).

Gezien de moeizaamheid van het gebruik van deze regel worden veeleer afgeleide stellingen gebruikt (zoals de juist bewezene).

Overigens spreken we af dat we een dergelijke verwisseling zullen noteren als $X : \text{swap}(p,q)$, dus

$\{ p,q \text{ in } \text{dom}(X) \wedge X[p]=a \wedge X[q]=b \}$

$X : \text{swap}(p,q)$

$\{ X[p]=b \wedge X[q]=a \}$

waarbij in PASCAL $X:\text{swap}(p,q)$ bijvoorbeeld als voorgaand gecodeerd kan worden.

We lichten het manipuleren met arrays toe aan de hand van voorbeelden.

Het eerste voorbeeld dat we bekijken staat bekend onder de naam "The Dutch National Flag":

> Gegeven is een rij van N bakjes. Jeder bakje bevat één knikker met kleur rood, wit of blauw. Van elke knikker mag maar één keer de kleur geïnspecteerd worden. Elk tweetal knikkers mag van bakje verwisseld worden.
> Gevraagd wordt om met inachtname van bovenstaande regels de Nederlandse vlag te maken.

In termen van Pascal geformuleerd:

> Gegeven integer N, $N \geq 0$, en array k (i: $0 \leq i < N$) of kleur met type kleur = (rood, wit, blauw). Bewerkstellig R met
>
> $$R: (\underline{E}\ x,y : 0 \leq x \leq y \leq N : (\underline{A}\ i : 0 \leq i < x : k[i] = rood) \wedge (\underline{A}\ i : x \leq i < y : k[i] = wit) \wedge (\underline{A}\ i : y \leq i < N : k[i] = blauw))$$
>
> waarbij alleen de swap op array k mag worden toegepast en ieder array element ten hoogste één keer op kleur getoetst mag worden.

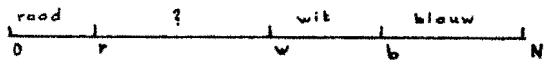
(Merk op dat zonder deze toevoeging R bevredigd kan worden door aan elk element van k de waarde blauw toe te kennen).

We introduceren integer variabelen r, w en b en kiezen als invariant

$$P: (\underline{A}i : 0 \leq i < r : k[i] = rood) \wedge (\underline{A}i : w \leq i < b : k[i] = wit) \wedge (\underline{A}i : b \leq i < N : k[i] = blauw) \wedge 0 \leq r \leq w \leq b \leq N$$

te initialiseren met r:=0 ; w:=N ; b:=N . (waar vandaan deze P?)

In onderstaand plaatje is P afgebeeld.



$P \wedge w = r \Rightarrow R$; als variante functie komt w-r in aanmerking.

Indien $w \neq r$ dan komen zowel $k[r]$ als $k[w-1]$ voor inspectie in aanmerking. We onderzoeken of de keuze verschil maakt.

- inspectie van $k[r]$:
  - rood : verhoog r (0 swaps)
  - wit : wissel met $k[w-1]$ (1 swap)
  - blauw : wissel met $k[b-1]$ en daarna met $k[w-1]$ (2 swaps)
  - gemiddeld 1 swap per repetitieslag.

- inspectie van $k[w-1]$:
  - rood : 1 swap
  - wit : 0 swaps
  - blauw : 1 swap
  - gemiddeld $^2/_3$ swap per repetitieslag.

Het programma.

```
r:=0 ; w:=N ; b:=N
{ invariant : P  ; variante functie  w-r }
;while  w<>r  do
 begin
   { P ∧ w≠r , i.h.b.  0≤ w-1 < N }
   x :=  k[w-1]
   ; if  x = rood  then  begin  k:swap(w-1,r) ; r:=r+1  end  else
    if  x = wit   then  w:= w-1    else
    if  x = blauw  then  begin  k:swap(w-1,b-1); w:=w-1 ; b:=b-1  end
   {P}
 end
```

Schrijf zelf een programma bij invariant P_1 met

P_1 : $(\underline{A}\, i : 0 \leq i < r : k[i] = \text{rood}) \wedge (\underline{A}\, i : r \leq i < w : k[i] = \text{wit}) \wedge$
$(\underline{A}\, i : w \leq i < b : k[i] = \text{blauw}) \wedge 0 \leq r \leq w \leq b \leq N.$

Als tweede voorbeeld beschouwen we het volgende probleem.

Gegeven is een integer N , $N \geq 0$, een integer array $X(i: 0 \leq i < N)$ en integer K met $K \geq 0$.
Gevraagd wordt een programma ter berekening van de maximale lengte van enige aaneengesloten deelrij van X die ten hoogste K nullen bevat.
Restrictie hierbij is dat de elementen van X maar één keer geïnspecteerd mogen worden en wel in volgorde van toenemende index. (Zo u wilt kunt u $X(i: 0 \leq i < N)$ opvatten als representatie van een sequentiële file van N waarden).

Om schrijfwerk te besparen noemen we een aaneengesloten deelrij van X met ten hoogste i nullen een *i-segment*.
We streven naar eindrelatie R met

R: r = de maximale lengte van enig K-segment van $X(i: 0 \leq i < N)$

Vervangen van N door integer variabele n leidt tot

P: r = de maximale lengte van enig K-segment in $X(i: 0 \leq i < n) \wedge 0 \leq n \leq N$

te initialiseren met $n := 0 ; r := 0$. Als variante functie ligt N-n voor de hand. We berekenen het effect van verhogen van n met 1 , onder de voorwaarde $0 \leq n < N$:

de maximale lengte van enig K-segment van $X(i: 0 \leq i < n+1)$
= { definitie max }
max (de maximale lengte van enig K-segment van $X(i: 0 \leq i < n)$,
de maximale lengte van enig K-segment van $X(i: 0 \leq i < n+1)$ eindigend in $X[n]$)
= { definitie P }
max (r , $(\underline{\text{MAX}}\ q : q \geq 0 \wedge X(i: q \leq i < n+1)$ is K-segment : $n+1-q)$)
= { definitie $\underline{\text{MIN}}$ en $\underline{\text{MAX}}$ }
max (r, $n+1 - (\underline{\text{MIN}}\ q : q \geq 0 \wedge X(i: q \leq i < n+1)$ is K-segment : $q)$)

Het tweede argument van max in de laatste regel suggereert de introductie van een nieuwe variabele a en een nieuwe invariant $P \wedge P_1$ met

P_1 : $a = (\underline{MIN}\ q : q \geq 0 \wedge X(i : q \leq i < n) \text{ is } K\text{-segment} : q)$.

We krijgen dan het volgende programma.

```
n:=0 ; r:=0 ; a:=0      {P ∧ P1}
;while n<>N do
 begin
    "bewerkstellig P1 (n+1)-n"
    ;r:= max (r, n+1-a)
    {(P ∧ P1) (n+1)-n}
   ;n:=n+1
    {P ∧ P1}
 end
```

Blijft ons nog over "bewerkstellig $P_1{}^{n}_{n+1}$" .
Er geldt

$$P_1 \wedge 0 \leq n < N \wedge X[n] \neq 0 \Rightarrow P_1{}^{n}_{n+1} .$$

en

$$P_1 \wedge 0 \leq n < N \wedge X[n] = 0 \wedge K > 0 \Rightarrow (\underline{MIN}\ q : q \geq 0 \wedge X(i : q \leq i < n+1) \text{ is } K\text{-segment} : q) = (\underline{MIN}\ q : q \geq 0 \wedge X(i : q \leq i < n) \text{ is } (K-1)\text{-segment} : q) .$$

en

$$P_1 \wedge 0 \leq n < N \wedge X[n] = 0 \wedge K = 0 \Rightarrow (n+1) = (\underline{MIN}\ q : q \geq 0 \wedge X(i : 0 \leq i < n+1) \text{ is } K\text{-segment} : q) .$$

Blijkbaar dienen we voor $K > 0$ de invariant $P \wedge P_1$ uit te breiden met

P_2 : $a1 = (\underline{MIN}\ q : q \geq 0 \wedge X(i : 0 \leq i < n) \text{ is } (K-1)\text{-segment} : q)$.

Uit de overeenkomst van P_1 en P_2 leiden we af dat voor $K > 1$ de invariant $P \wedge P_1 \wedge P_2$ uitgebreid dient te worden met P_3 etcetera.

We lossen dit probleem op door het invoeren van een integer array $a(i: 0 \leq i < K+1)$ en breiden de oorspronkelijke invariant P uit tot $P \wedge Q$ met

Q: $(\underline{A}\, i : 0 \leq i < K+1 : a[i] = (\underline{MIN}\, q : q \geq 0 \wedge X(j : q \leq j < n) \text{ is } i\text{-segment} : q))$.

Dan geldt $Q \wedge 0 \leq n < N \wedge X[n] \neq 0 \Rightarrow Q^{n}_{n+1}$ en verder

$0 \leq n < N \wedge Q \wedge X[n] = 0$
$\Rightarrow$ { definitie Q }
$(\underline{A}\, i : 0 < i < K+1 : a[i-1] = (\underline{MIN}\, q : q \geq 0 \wedge X(j : q \leq j < n+1 \text{ is } i\text{-segment} : q)) \wedge$
$(n+1) = (\underline{MIN}\, q : q \geq 0 \wedge X(j : q \leq j < n+1) \text{ is } 0\text{-segment} : q)$.

Teneinde niet het array a "een plaats op te hoeven schuiven" in het geval $X[n] = 0$, en zo in een orde $K * N$ algoritme te belanden, vatten we a op als "cyclisch array", i.e. wijzigen we Q in Q' met

Q': $0 \leq h < K \wedge$
$(\underline{A}\, i : 0 \leq i < K : a[i \oplus h] = (\underline{MIN}\, q : q \geq 0 \wedge X(j : q \leq j < n) \text{ is } i\text{-segment} : q))$

waarbij $\oplus$ staat voor de optelling modulo $(K+1)$.

Toon zelf aan

$\{ 0 \leq n < N \wedge Q' \wedge X[n] = 0 \}\ h := (h + K) \bmod (K+1);\ a[h] := n+1\ \{ Q'^{n}_{n+1} \}$.

Op de volgende pagina vindt het ongeannoteerde eindresultaat. Geef zelf de annotaties.

Geef een orde N programma voor dit probleem zonder invoeren van een array in het geval dat de array-elementen van X meer dan eens geïnspecteerd mogen worden.

```
n:=0 ; r:=0 ; h:=0 ; v:=0 ; while v <> K+1 do begin a[v]:=0 ; v:=v+1 end
{P ∧ Q'}
; while n <> N do
 begin
   if X[n]=0 then begin h:= (h+K) mod (K+1) ; a[h]:=n+1 end
  ; w:= n+1 - a[(h+K) mod (K+1)]
  ; if r<w then r:=w
  ; n:=n+1
 end
```

Opgaven.

1. Gegeven is een integer N, $N \geq 0$, en een integer array $X(i: 0 \leq i < N)$ met
   $(\underline{A}\, i : 0 \leq i < N : X[i]=0 \vee X[i]=1)$.
   Schrijf een programma dat R bewerkstelligt met
   R: $X(i: 0 \leq i < N)$ is ascending
   waarbij de swapoperatie de enige toegestane operatie op X is.

2. "Minimale segmentsom".

   Gegeven zijn een integer N met $N \geq 0$ en een integer array $X(i: 0 \leq i < N)$.
   Voor $0 \leq i \leq j \leq N$ is de segmentsom $S(i,j)$ gedefinieerd door
   $S(i,j) = (\underline{\Sigma}\, k : i \leq k < j : X[k])$

   Schrijf een programma dat aan integer variabele r een dusdanige waarde toekent dat voldaan is aan

   R: $r = (\underline{MIN}\, i,j : 0 \leq i \leq j \leq N : S(i,j))$

3. Gegeven is een integer N, $N \geq 0$, een integer array $X(i: 0 \leq i < N)$ en een integer K met $K \geq 1$.
   Een rij $t(i: 0 \leq i < K)$ heet een telrij indien
   $0 \leq t(0) \wedge t(K-1) < N \wedge (\underline{A}\, i : 0 < i < K : t(i-1) < t(i)) \wedge (\underline{A}\, i : 0 \leq i < K : X[t(i)] = i)$.
   Schrijf een programma ter berekening van het aantal telrijen van X.

4. "Langste flank".

Gegeven zijn een integer N met $N \geq 1$ en een integer array $X(i: 0 \leq i < N)$.
Een segment $X(i: p \leq i < q)$ met $0 \leq p \leq q \leq N$ heet een flank indien
$(\underline{A}\, i : p < i < q : X[i-1] \leq X[i]) \vee (\underline{A}\, i : p < i < q : X[i-1] \geq X[i])$.

Schrijf een programma ter berekening van de maximale lengte van enige flank van X.

5. "Longest Upsequence".

Gegeven zijn een integer N met $N \geq 1$ en een integer array $X(i: 0 \leq i < N)$.
Voor elke s met $0 \leq s \leq N$ kan men deelrijen van X vormen door een willekeurig (N-s)-tal elementen uit de rij X weg te laten en de volgorde van de resterende s elementen te handhaven. Zo'n deelrij ter lengte s die ascending is heet een upsequence ter lengte s.

Schrijf een programma ter berekening van de maximale lengte van enige upsequence van X.

6. "Gebalanceerde segmenten".

Gegeven zijn een integer N met $N \geq 0$ en een integer array $X(i: 0 \leq i < N)$ met $(\underline{A}\, i : 0 \leq i < N : X[i] = 0 \vee X[i] = 1)$.
Een segment $X(i: p \leq i < q)$ met $0 \leq p \leq q \leq N$ heet gebalanceerd indien
$(\underline{N}\, i : p \leq i < q : X[i] = 0) = (\underline{N}\, i : p \leq i < q : X[i] = 1)$

Schrijf een programma ter berekening van de maximale lengte van enig gebalanceerd segment van X.

9. Naschrift.

In deze cursus hebt u kennis gemaakt met een aantal basisbegrippen en basistechnieken van het programmeren. De mogelijkheid om over programma's te kunnen redeneren zonder een beroep te hoeven doen op de uitvoering ervan door een computer is een van de belangrijkste elementen van deze cursus.

Het spreekt vanzelf dat deze cursus maar een klein gedeelte van het vakgebied programmeren bestrijkt. Programmeertechnieken zoals divide and conquer, dynamic programming, recursion, backtracking, hashing en dergelijke, kunnen wellicht in een volgende cursus aan de orde komen.

Als besluit geven we enkele praktische wenken die van nut zijn bij het onderwijs in de informatica.

- Behandel de onderdelen apparatuur, gebruik van apparatuur en programmeren strikt gescheiden.

  Bij vermenging hiervan worden leerlingen geconfronteerd met beeldschermen, toetsenborden, login-procedures, editors, compilers en wat dies meer zij. Daarnaast wordt en passant nog iets over programmeren verteld. Het is bijzonder moeilijk om vanuit zo'n situatie nog fundamentele kennis over het programmeren bij te brengen.

- Maak een bewuste keuze voor een programmeertaal en kies daarvan alleen het noodzakelijke.

  Er zijn programmeertalen zoals Fortran en Basic waarin het moeilijk is om correcte programma's te schrijven. Als basis voor programmeeronderwijs zijn dergelijke talen ongeschikt.

  In deze cursus hebben wij ons beperkt tot een paar statements uit Pascal. Dan is het al moeilijk genoeg. Hetzelfde geldt voor datastructuren.

- Vermijd antropomorfie.

  Computers zijn niet intelligent, ook niet kunstmatig intelligent. Zij zijn niet vriendelijk en staan niet te wachten. Begrippen als toestand en toestandsovergang zijn niet moeilijk en kunnen aan tal van voorbeelden geïllustreerd worden. Men denke aan wasautomaten, liften, koffieautomaten etcetera.

- Werk netjes en zorgvuldig.

  Veel fouten in programma's zijn het gevolg van slordig en onzorgvuldig werken. Wen leerlingen er aan om netjes te schrijven en zorgvuldig te formuleren.
  Niet alles hoeft op elk niveau even formeel. Maar informeel betekent niet slordig, onvolledig of fout.

- Gebruik apparatuur waarvoor zij bedoeld is.

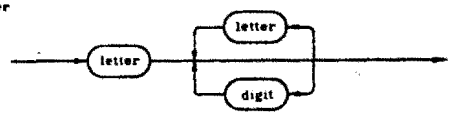
  Wordt op school een computer aangeschaft voor programmeeronderwijs, gebruik dit apparaat dan niet voor leerlingenadministratie, roosteradministratie etcetera.

Wij wensen u succes toe.

Appendix A.

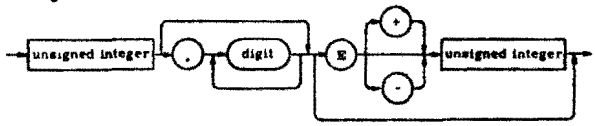
PASCAL

identifier

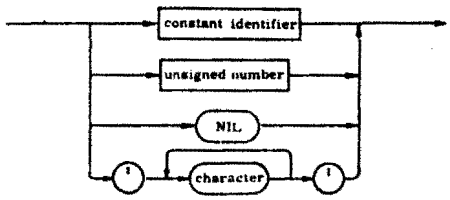


unsigned integer

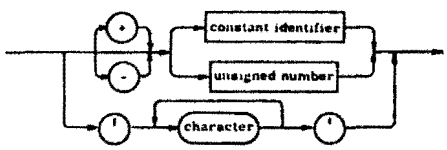
unsigned number



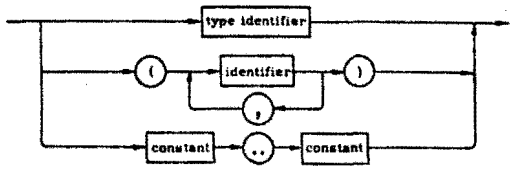
unsigned constant



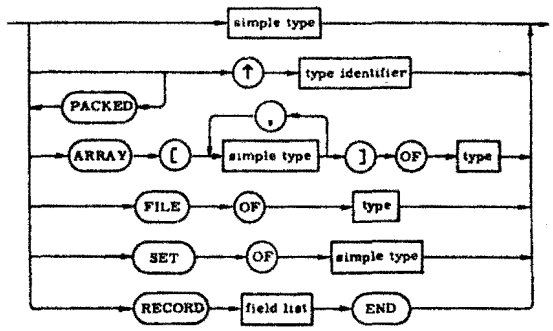
constant



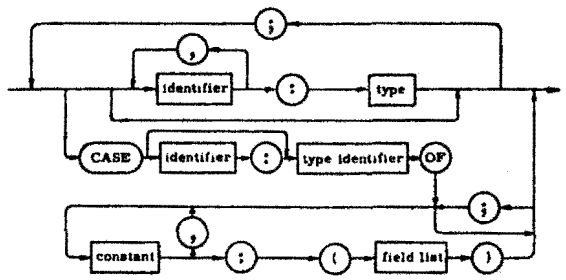
simple type



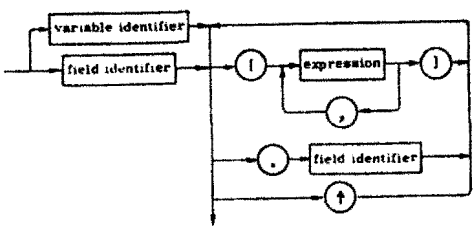
type



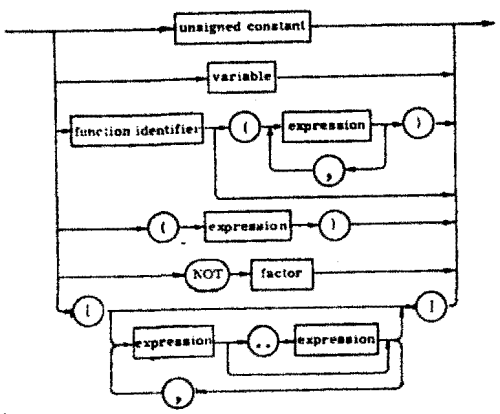
field list



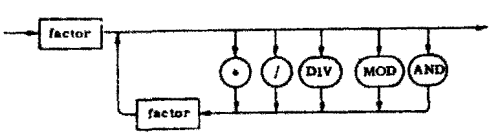
variable



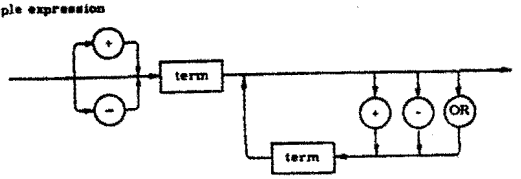
factor



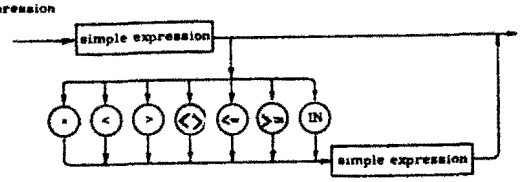
term



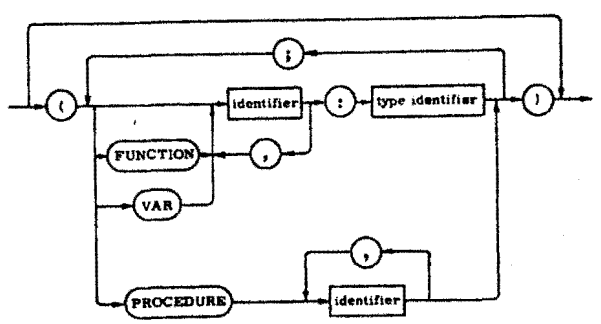
simple expression

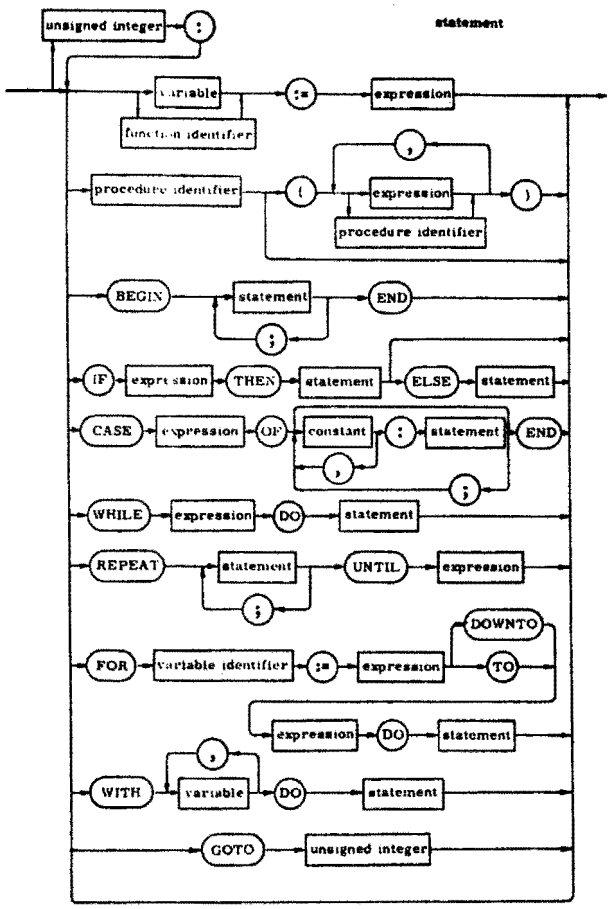


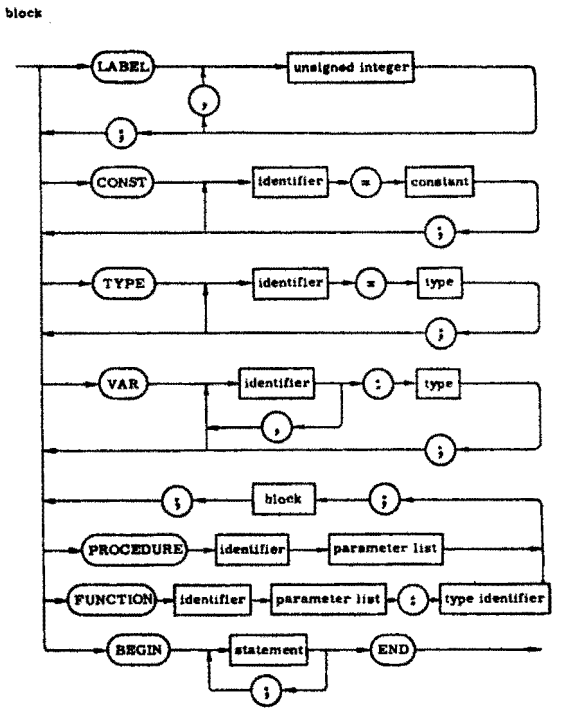
expression

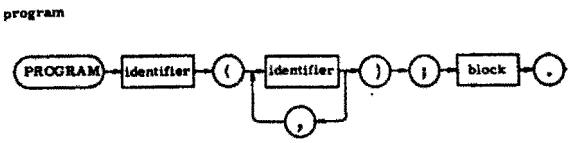


parameter list




statement
unsigned integer
:
variable
function identifier
:=
expression
procedure identifier
(
,
expression
)
procedure identifier
BEGIN
statement
;
END
IF
expression
THEN
statement
ELSE
statement
CASE
expression
OF
constant
:
statement
,
;
END
WHILE
expression
DO
statement
REPEAT
statement
;
UNTIL
expression
FOR
variable identifier
:=
expression
DOWNTO
TO
expression
DO
statement
WITH
,
variable
DO
statement
GOTO
unsigned integer


block
LABEL
unsigned integer
,
;
CONST
identifier
=
constant
;
TYPE
identifier
=
type
;
VAR
identifier
:
type
,
;
;
block
;
PROCEDURE
identifier
parameter list
FUNCTION
identifier
parameter list
:
type identifier
BEGIN
statement
;
END


program
PROGRAM
identifier
(
identifier
,
)
;
block
.