Programmeren ja of nee, en zo ja in welke taal?

H. Hermsen
OW & OC, RU Utrecht

Summary

The goal of teaching computer literacy to pupils in secondary schools is often formulated as "learning about computers and about the application of computers in daily life". In many existing courses the subject-matter programming, primarily in connection with the solution of mathematical problems, is over emphasized. When programming a computer a pupil can indeed learn a lot about how a computer actually works. An important question is whether the widespread use of the language BASIC is the best choice for this purpose.
Understanding of the use of computers in society can however only be achieved in a programming course by choosing an appropriate set of exercises. Another good but almost never used possibility to introduce an application is letting the pupil use one on the computer at hand.
In this article the reader is informed about what programming actually is and the role this activity plays in practice. This will lead to some conclusions about the above mentioned problems.

Een toenemend aantal scholen in het voortgezet onderwijs schaft een of meer microcomputers aan. Wegens de bescheiden beschikbare middelen zijn het meestal eenvoudige apparaten (1), in aantal te klein voor een bevredigend klassikaal gebruik door de leerlingen.
De toepassing van de computer in de school is vaak *multifunctioneel*. Het gedeeltelijk automatiseren van de schoolorganisatie is vaak de doorslaggevende reden om tot aanschaf over te gaan. Toepassing in de bestaande vakken wordt wel gewenst maar komt in de praktijk nog weinig voor. De lessen waarin de computer, onderwijskundig gezien zijn belangrijkste rol speelt, worden gegeven in een nieuw vak dat veelal de naam *informatica* of *computerkunde* draagt.

De stof voor dit vak bestaat meestal uitsluitend uit het leren programmeren in de programmeertaal BASIC. De opgegeven oefeningen zijn vaak wiskundig van aard. Het gebruik van de taal BASIC is het gevolg van de beperkingen van de beschikbare apparatuur, die andere activiteiten vrijwel uitsluiten. Een belangrijke oorzaak voor de wijze van lesgeven is het enthousiasme van de betrokken (wiskunde)docent. Een enthousiasme dat gelijkenis vertoont met dat van de computerhobbyist. Ten onrechte wordt verondersteld dat het programmeren de belangrijkste (creatieve) menselijke activiteit met computers is.

Is deze grote nadruk op het programmeren zinvol voor het voorbereiden van de leerling op de geautomatiseerde informatiemaatschappij van (nu en) de toekomst? Als het leren programmeren zin heeft, moet daarvoor dan de taal BASIC gekozen worden?

In dit artikel zal getracht worden deze vragen te beantwoorden. Het vak dat tot doel heeft de leerling algemene ontwikkeling op het gebied van de informatica te verschaffen zal, in navolging van het ministerie van Onderwijs, *burgerinformatica*(2) genoemd worden.

Computers verwerken algoritmen

Wat doen computers voor ons? Een aantal antwoorden op deze vraag ligt voor de hand:

- *Gegevens bewaren*
  Deze kunnen ingevoerd, gewijzigd en opgevraagd worden. Dit alles heeft tot doel ons informatie te verschaffen. Voorbeelden zijn de Girodienst en teletekst.
- *Gegevens verwerken*
  Gegevens worden in een afgesproken vorm en volgorde ingevoerd. Na verwerking ontstaat een interpreteerbaar uitvoerresultaat. De automatische verwerking van CITO-toetsen is een voorbeeld hiervan.

- *Teksten verwerken*
  Geschreven teksten kunnen ingevoerd, bewaard en gecorrigeerd worden. Tenslotte kan een gewenste opmaak verzorgd worden (het dagblad).

De vraag had echter een diepere bedoeling. Genoemd werden een aantal *toepassingen* van de computer, een aantal *functies* die het apparaat voor ons kan vervullen. De toepassingen zijn echter geen eigenschap van de computer zèlf.
De functies van de computer worden bepaald door *programma's*. Wanneer er in het vervolg sprake is van een functie of toepassing van de computer wordt bedoeld: een functie of toepassing gerealiseerd door programma's die door de computer *verwerkt* worden.
De term *software* wordt gebruikt om het totaal van voor een computer beschikbare *programmatuur* aan te geven.
Software als noodzakelijke aanvulling op de *hardware*.
De hardware is de verzameling apparaten zoals die er staat, zoals je die, stuk voor stuk en al dan niet met beleid, uit elkaar zou kunnen halen.
De software is in gecodeerde vorm opgeslagen op magnetische gegevensdragers, zoals magneetbanden of schijven. De gegevensdragers kunnen "beschreven" of "gelezen" worden door aan de computer gekoppelde "aftast"apparatuur.
De computer kan direct met software "gevoed" worden via een aangesloten schijfeenheid, die voorzien is van een schijf of flexibel schijfje. Kopieën van de software behoren ook op een veilige plek opgeborgen te worden, bijvoorbeeld in een brandvrije kluis. Er zou immers iets mis kunnen gaan met een veelgebruikte gegevensdrager.

We wagen ons even aan een vergelijking met geluidsapparatuur. De hardware is daar de versterker, de luidsprekers, de platenspeler en het cassettedeck. De software is dan het op bandjes of platen vastgelegde (gecodeerde) geluid. De *functies* van de geluidsapparatuur zijn ons het genot te verschaffen van de meest uiteenlopende – van pop tot klassiek – soorten muziek. Deze funties worden door de hardware vervuld vanwege diens eigenschap de gecodeerde melodieën in luchttrillingen, dus geluid, om te kunnen zetten.
De *melodieën* die de computer voor ons speelt worden dus bepaald door de programma's waaruit de software bestaat. Zo kan men op dezelfde computer een spelletje spelen (pop) maar ook een personeelsadministratie voeren (klassiek).
Melodieën betekenen in dit verband het gedrag dat de computer vertoont aan de *buitenwereld* via bijvoorbeeld beeldscherm, printer of een noodzakelijke menselijke activiteit aan een toetsenbord. Een uitgevoerde functie kan ook een niet direct zichtbaar effect hebben. Bijvoorbeeld wijzigingen die aangebracht worden in een verzameling gegevens op een gegevensdrager.
Om misverstanden te voorkomen: ook de gekoppelde "aftasters", al dan niet van een gegevensdrager voorzien, moeten we beschouwen als *buitenwereld*. Met computer bedoelen we uitsluitend het apparaat dat de programma's *verwerkt* ("afspeelt" in onze vergelijking).

We komen nu terug op de vraag die aan het begin werd gesteld: "wat doet een computer voor ons?"
Uit het voorafgaande zal duidelijk zijn dat het antwoord zal moeten worden: het uitvoeren van in programma's vastgelegde opdrachten. Software bestaat uit *programmatuur*, dat wil zeggen verzamelingen van functioneel bij elkaar horende computerprogramma's. Op hun beurt zijn programma's reeksen gecodeerde opdrachten voor de betreffende computer. Een reeks gecodeerde opdrachten die de computer een deeltaak van een zekere functie laat verrichten noemen we *algoritme*.

De conclusie uit dit alles is: *computers zijn apparaten die algoritmen kunnen verwerken*.
Het begrip *algoritme* heeft overigens een ruimere betekenis; een computerprogramma is er een voorbeeld van.

Meer over algoritmen

Het woord algoritme komen we reeds tegen in een, uit de twaalfde eeuw stammende, Latijnse vertaling van een Arabische verhandeling uit het jaar 850 over de algebra. Het betekende daar zoiets als: rekenmethode.

Voor de geïnteresseerde: in de moderne logica worden de begrippen algoritme en *effectieve procedure* door elkaar gebruikt. (Een voorbeeld hiervan is het met behulp van waarheidstabellen bepalen of een formule van de propositionele logica een tautologie, en dientengevolge een theorema van deze logica is.)
Een precieze definitie van het begrip is overigens niet makkelijk te geven. Alvorens aan de hand van enkele voorbeelden het begrip te verduidelijken volgt nog een (vage) omschrijving geïnspireerd door een poging tot definitie in een publikatie van Prof. E.W. Dijkstra (3).

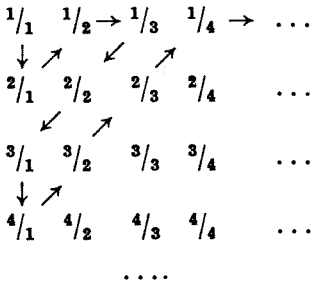
Een algoritme is een beschrijving van een gedragspatroon uitgedrukt in termen van een gedefinieerde (of intuïtief duidelijke), eindige verzameling beschreven (primitieve) handelingen waarvan aangenomen kan worden dat ze uitgevoerd kunnen worden.

Als eerste voorbeeld nemen we de algoritme van Euclides waarmee we de grootste gemene deler (GGD) van twee getallen, zeg n en m, kunnen bepalen.
De randvoorwaarden zijn:
n, m, $\in \{1, 2, 3, 4, 5, 6, \ldots\}$ en $n \geqq m$.

De algoritme, als we de beschreven stappen in volgorde uitvoeren, luidt dan:
1. Geef x de waarde van n.
   Geef y de waarde van m.
2. Laat R(x/y) de rest van de deling x/y voorstellen.
3. Als R(x/y) nul is dan mogen we stoppen. De huidige waarde van y is dan de grootste gemene deler van m en n.
4. (Kennelijk geldt hier $R(x/y) \neq 0$)
   Geef x de waarde van y.
   Geef y de waarde van R(x/y) uit 2.
   Ga nu naar 2 met deze nieuwe waarden.

Het uitvoeren van dit proces voor bijvoorbeeld de getallen 60 en 51 leidt achtereenvolgens tot:
GGD(60,51) = GGD(51,9) = GGD(9,6) = GGD(6,3) = 3.

Het is duidelijk dat het met de algoritme beschreven proces na een eindig aantal stappen tot een resultaat zal leiden.
Deze laatste eis wordt in het algemeen aan een "echt" algoritme gesteld. Een algoritme dat niet stopt wordt ook wel *pseudo*-algoritme genoemd.
Het tweede voorbeeld is de pseudo-algoritme die Cantor gebruikte in het bewijs dat de aftelbaarheid van de rationele getallen moest aantonen (4).
Eerst wordt met een plaatje duidelijk gemaakt hoe we de rationele getallen in een matrix moeten plaatsen. In feite is dit al een pseudo-algoritme op zich.



Vervolgens wordt verteld hoe er bij het aftellen gehandeld moet worden. Het motto is: Begin links-boven en volg de pijlen. Wandel door de matrix volgens een weg, waarvan het eerste deel aangegeven is. Wanneer we bij een getal zijn aangeland vragen we ons af of we de waarde ervan reeds afgeteld hebben. Zo ja, dan vervolgen we onze wandeling zonder meer. Zo nee, dan plaatsen we eerst het getal als laatste element in een groeiende deelrij van afgetelde rationele getallen. Het begin van die rij is dan:

1, 2, ½, ⅓, 3, 4, 3/2, ⅔, ¼.

Beide algoritmen appelleren aan de intuïtie van de lezer. De verwachting is dat de gegeven instructies bij iedereen tot eenzelfde resultaat zal leiden. Het begrijpen van de rekenmethode voor de GGD of het bewijs van de stelling komt neer op het mentaal uitvoeren van de in de algoritmen vastgelegde handelingen.
Een algoritme is zo een (formeel of informeel) wiskundig communicatiemiddel tussen mensen. Een algoritme kan daarnaast een communicatiemiddel met een machine zijn.
Wanneer een algoritme een beschrijving is van een proces dat op mechanische (of elektronische) wijze uitgevoerd moet worden op een machine, spreken we van programma.
Een programma is geschreven in een zekere programmeertaal. Zo'n programmeertaal moet dan de eigenschap hebben een "... *gedefinieerde ..., eindige verzameling beschreven primitieve handelingen...*" te zijn.
Een voorbeeld van een programma is het volgende in de programmeertaal ECOL geschreven algoritme voor de berekening van n! voor n lopend van 0 tot 100:

```
FUN fac(n)
 ALS n = 0 DAN
  uitkomst := 1              ;0!
 ANDERS
  uitkomst := n*fac(n-1)
 ALSEND
FUN := uitkomst

START
 i := 0
 ZOLANG i < 101
  REGEL := i,"! = ",fac(i) ;druk af
  i := i + 1               ;verhoog i
 HERHAAL
KLAAR
```

We zien een definitie van een functie fac(n) die als waarde n! zal opleveren. De functie roept zichzelf aan: n*fac(n-1) hetgeen overeenkomt met de eigenschap dat n! = n * (n-1)!. De mogelijkheid van zichzelfaanroepende functies wordt recursie genoemd. Het valt buiten het kader van dit artikel in te gaan op het grote belang van deze manier van uitdrukken.
In het hoofdprogramma, tussen START en KLAAR wordt opdracht gegeven de resultaten voor i lopend van 0 tot 100 af te drukken.

Onder informatici bestaan (niet altijd geheel gelijkluidende) opvattingen over wat de eigenschappen van een programmeertaal moeten zijn wil het een goede beschrijver van algoritmen genoemd kunnen worden. Verder zijn er methoden ontwikkeld om algoritmen te ontwerpen en van een algoritme te bewijzen dat het een correcte werking heeft. Het is zeer wel mogelijk, en gebeurt zelfs in cursussen, de kunst van het maken van algoritmen met behulp van een programmeertaal te leren, zonder deze programma's ooit aan een computer aan te bieden. Een correctheidsbewijs is voldoende garantie voor een goede werking. Een controle daarvan op een echte machine zou kinderachtig zijn. Waarmee hier overigens niet uitgesproken is, dat dit dan de beste aanpak zou zijn.

Programmeertalen, algoritmiek en aanverwante onderwerpen zijn in de nog jonge informaticawetenschap kennisgebieden waarop al een ruime theorie voorhanden is, die zich snel uitbreidt.
Er bestaan zelfs wel opvattingen dat algoritmisch denken identiek is aan het bedenken van oplossingen voor een grote klasse problemen. De ontwerptechnieken voor algoritmen moeten dan beschouwd worden als een middel om een sluimerende, reeds aanwezige, denktrant tot het bewustzijn te laten doordringen.
Een directe consequentie hiervan is dat het zinvol is het algoritmetiseren vroeg aan te leren en tot de algemene ontwikkeling van mensen te laten behoren.
Het gelijkschakelen van probleemgericht denken en algoritmisch denken is speculatief en gaat mijns inziens veel te ver. Creativiteit, inspiratie zo u wilt, speelt, evenals bij het bedrijven van de wiskunde, een grote rol bij het bedenken en maken van programma's.

Deze creativiteit zouden we kunnen beschouwen als de benaming van een denkproces waarvan we de werking vooralsnog niet kunnen doorgronden. Daarom ook niet zondermeer het predikaat algoritmisch kunnen geven.
Ik beschouw een algoritme dan ook uitsluitend als een (formeel) communicatiemiddel tussen mensen of tussen mensen en machines; de methodiek van het ontwerpen van algoritmen als een op gezette tijden zelfgekozen hulpmiddel om, op zich grillig verlopende, gedachten en invallen te stroomlijnen met het doel een overzichtelijk programma met een lage foutkans te verkrijgen.

Zoals in een later stadium duidelijk zal worden, behoort het programmeren niet tot de dagelijkse activiteiten van een computergebruiker. Het argument dat het maken van algoritmen (in de vorm van programma's) aansluit bij ons wezenlijke denken bestrijd ik.
Heeft het dan nog enige zin het programmeren op te nemen in het vak burgerinformatica? Toch wel. Straks zal aangegeven worden waarom.

Wat is programmeren?

Programma's waarin algoritmen voor computers zijn vastgelegd zijn een gevolg van menselijke activiteit. Een programmerende gebruiker, specialist of niet, behoeft programma's, zou dit al mogelijk zijn, niet te coderen in een vorm die direct door de computer te verwerken is.
Direct te verwerken programma's bestaan in feite uit reeksen getallen. Voor een belangrijk deel zijn dit de coderingen van (simpele) opdrachten voor de computer. Deze zijn onderdeel van de zogenoemde *machinetaal* van de computer.
Ook treffen we gegevens, dat wil zeggen code's voor getallen en letters, aan, die in het programma zijn opgenomen.
Een aangeschafte computer beschikt in het algemeen minstens over software, die een programmeur in staat stelt programma's te maken in een voor mensen bruikbare taal.
Programmeren is dan in de eerste plaats het bedenken en opschrijven van een programmatekst volgens de voorschriften van de gebruikte taal. Deze tekst moet dan ingevoerd worden en kan vervolgens met behulp van *vertaal*software omgezet worden in direct te verwerken code. Tenslotte kan opdracht gegeven worden de vertaalde versie van het programma door de computer te laten verwerken.
Voor de volledigheid: een programmatekst wordt soms ook wel *direct* verwerkt door middel van *interpretatie*software. Dit is gebruikelijk bij de taal BASIC.
Interpretatiesoftware vertaalt niet naar een apart te verwerken code maar voert de in het programma vastgelegde opdrachten onmiddellijk uit.
Verwerking door middel van interpretatie is trager dan verwerking van een vertaalde versie. De opdrachten in de programmatekst moeten immers steeds opnieuw herkend worden alvorens ze door middel van een actie verwerkt worden. Bij vertaalde programma's heeft die herkenning al plaatsgevonden. De vertaalde versie kan zonder meer door de computer verwerkt worden.
Voor grote en/of veel gebruikte programma's verdient vertaling de voorkeur.

Om het allemaal nog wat ingewikkelder te maken: er bestaan ook mengvormen van vertaling en interpretatie. Zo komt het wel voor dat een programma vertaald wordt naar een eenvoudig te interpreteren code, die bedacht is door de ontwerper van de vertaalsoftware. We kunnen deze code opvatten als de machinetaal van een *virtuele* (niet bestaande, zelf bedachte) computer. Verwerking van een programma is dan de interpretatie van de vertaalde versie op een echte computer. De virtuele computer wordt op die manier op de echte nagebootst.

Software die algemene taken voor een gebruiker vervult zoals het bewaren van ingevoerde (programma)-teksten, het oproepen van een vertaler of interpretator en het ter verwerking aanbieden van een vertaald programma, noemen we het *bedrijfssysteem* (operating system) van de computer. Het bedrijfssysteem verzorgt als het ware de interne organisatie van de hardware en maakt in eerste aanleg de computer en zijn randapparaten voor de mens hanteerbaar.
Een belangrijke taak van het operating system kan zijn een daarvoor geschikte computer te verdelen onder verschillende *gelijktijdige* gebruikers.
Naast het bedrijfssysteem is er de eerder genoemde vertaalsoftware, die het mogelijk maakt programmatuur te ontwikkelen voor de toepassingen van de computer in de automatisering.

Programmeertalen worden wel verdeeld in *hoge* en *lage* talen. Een lage taal (ook wel *assembleer*taal genoemd) is machine-gericht, staat dicht bij de code's van de machinetaal en vormt een nog net voor mensen hanteerbare mogelijkheid algoritmen voor de computer te beschrijven.
Een hoge taal, zoals bijvoorbeeld PASCAL, is mensgericht en kenmerkt zich door:

- Een kernachtige uitdrukkingsmogelijkheid.
  Enkele regels programmatekst kunnen uitdrukking geven aan iets waarvoor honderden regels assembleertaal-opdrachten nodig zijn.

- De potentiële geschiktheid voor het beschrijven van overzichtelijke en leesbare algoritmen, mits aan een aantal schoonheidseisen is voldaan.

- De te verwachten werking van gebruikte taalconstructies worden, machine-onafhankelijk, binnen een taaldefinitie beschreven. Specifieke eigenschappen van een computer waarop de programma's verwerkt kunnen worden, blijven hierbij dus buiten beschouwing.

- Directe uitwisseling van programma's tussen verschillende merken computers behoort tot de mogelijkheden.
  Hierbij moet wel aangenomen worden dat de gebruikte taal voldoende gestandaardiseerd is (PASCAL wel, BASIC zeker niet) en er op die verschillende computers vertaal- of interpretatie software voor de taal aanwezig is.

De eigenlijke "werking" van een computer wordt weergegeven door de algoritmen die we, bij voorkeur in een hoge programmeertaal, voor deze computer kunnen schrijven.
Voor het formuleren van overzichtelijke algoritmen kan het beste gebruik gemaakt worden van programmeertalen die hiervoor ontworpen zijn. Helaas zijn dan de vele BASIC varianten minder geschikte kandidaten.
Programmeren is een creatieve bezigheid, die door de vele computerhobbyisten als een uitstekende tijdspassering wordt beschouwd. Ook in cursussen over de computer en zijn toepassingen komt het zelf programmeren aan de orde. In de *echte* praktijk komt deze activiteit echter vrij weinig voor en wordt zelfs meer en meer op de achtergrond gedrongen.

De praktijk

Buiten de school doet de computer in snel tempo zijn intrede in het maatschappelijk leven. Nog niet eens zo heel lang geleden was het werken met de computer geconcentreerd in de diverse computercentra van grote bedrijven en instellingen. Gebruikers moesten op papier en volgens een vast protocol hun wensen aan de computerafdeling kenbaar maken.
De eerste decentralisatie ontstond door verdere ontwikkeling van datacommunicatietechnieken en de mogelijkheid een computer onder verschillende gelijktijdige gebruikers te verdelen. Computergebruikers konden door middel van een terminal, op of in de buurt van hun "werkplek", rechtstreeks met de computer communiceren en hun wensen op deze wijze uitgevoerd krijgen.
Snel na de decentralisatie van de communicatie met een computer werd de computer zèlf gedecentraliseerd. Dit kon vanwege de steeds kleinere afmetingen van de apparatuur dankzij de ontwikkelingen op het gebied van de micro-elektronica.
De lage produktiekosten zorgden ervoor dat de computer betaalbaar werd voor nieuwe gebruikers: scholen, kleine bedrijven, hobbyisten en tenslotte de gebruiker van de huiscomputer. Elke burger wordt op een of andere manier wel met de computer geconfronteerd.
Dit alles moest hand in hand gaan met een ontwikkeling het werken met computers uit de specialistische hoek te halen. De daarvoor ontwikkelde software stelt een gebruiker in staat zonder te programmeren met de computer te kunnen werken.
Daarmee is overigens niet gezegd dat dit werk dan altijd gereduceerd kan worden tot een simpele bediening. De eenvoud hiervan is sterk afhankelijk van de complexiteit van een gebruikte toepassing.
In de praktijk bestaat het werken met de computer meestal uit het in dialoog treden met het computersysteem via een beeldscherm en toetsenbord. Deze dialoog wordt gestuurd door programma's (dus algoritmen). Wanneer een gebruiker een gewenste functie heeft "uitgesproken" wordt deze door (andere) algoritmen uitgevoerd.
Deze activiteit kan beschouwd worden als *interactief programmeren* zonder overigens zelf een programma te schrijven. Er wordt naar gestreefd de dialoog met de computer zo "menselijk" mogelijk te maken. We spreken in dit verband van *gebruikersvriendelijke software.* Het omschrijven van een gewenste functie vraagt echter een nauwkeurige en niet mis te verstane boodschap. De conventies waaraan hiervoor voldaan moeten worden vormen een afspiegeling van de computer als algoritmenverwerker, zij het op een logisch hoog niveau.
Een gebruiker moet beschikken over een mentaal beeld van de gebruikte functies van de computer. Hij of zij moet verder de conventies kennen waaraan de dialoog moet voldoen.
Hoe complexer de gebruikte toepassing hoe noodzakelijker het begrip over de eigenlijke werking van de computer.

We komen hiermee op de noodzaak mensen in algemene zin kennis te verschaffen over het apparaat waarmee ze werken. Dat wil zeggen laten ervaren hoe de computer geprogrammeerd kan worden.
Hoewel soms niet strikt noodzakelijk voor de gebruiker van eenvoudige toepassingen, is de ontmythologiserende werking van het zelf naar de hand zetten van de computer ook voor haar of hem van belang.
Een door sommigen wel gepropageerde strikte scheiding tussen een programmerende elite en de *gewone* computergebruiker is uit een oogpunt van efficiency wel aardig, maar mag niet leiden tot een geforceerde kenniskloof.
Het is tevens de vraag of we er goed aan doen het werken met de computer, waar dat kan, op te splitsen in zeer simpele deeltaken. Op deze wijze ontstaat een moderne vorm van lopende bandwerk, waarvan we het geestdodende karakter kunnen voorspellen.

Conclusies

Aangezien al het gebruik van de computer bepaald is door op de computer werkende algoritmen, is het zelf programmeren een hulpmiddel tot een beter begrip van door de computer uitgevoerde functies.
Vrijwel iedereen wordt geconfronteerd met toepassingen van de computer. Enige kennis van de algoritmiek in het algemeen en het programmeren in het bijzonder hoort dan ook in de algemene vorming thuis.
Het werkt zeer verhelderend voor een leerling de computer door een eigen gemaakt programma iets uit te laten voeren dat volledig door haar of hem bepaald is.
Omdat programmeren het beste kan gebeuren in talen die een goede uitdrukkingsmogelijkheid hebben voor het beschrijven van algoritmen, is BASIC een slechte kandidaat voor onderwijskundige doeleinden.

PASCAL, ELAN, ECOL, COMAL en LOGO liggen meer voor de hand. Dat dit niet meteen tot iets zeer ingewikkelds leidt bewijzen de experimenten met de taal LOGO, gebruikt door 6- tot 12-jarigen (5).
Met nadruk wordt gesproken over *enige* kennis van het programmeren. Een belangrijk doel van de lessen burgerinformatica zal beslist ook het bijbrengen van kennis over "dagelijkse" toepassingen van de informatica moeten zijn.

Met dit laatste wordt dan niet het leren van wat feitjes uit een boek bedoeld.
Aangezien er vele toepassingen van de computer zijn die een verre van triviale bediening vergen (b.v. het werken met geavanceerde systemen voor gegevensbeheer) zijn er in principe zinvolle mogelijkheden om op dit terrein actief ervaringen met de computer op te doen.
Het is van groot belang een evenwicht te vinden tussen het leren programmeren en het overdragen van kennis over de toepassingen van de informatica.
Een probleem hierbij is dat het vergaren van enige kennis van het programmeren het leren van een programmeertaal impliceert. Dit kan een relatief tijdrovende zaak zijn.
Het verdient dan ook aanbeveling de programmeeropdrachten zo te kiezen dat ze reeds als toelichting op de te behandelen toepassingen kunnen dienen. Het vervaardigen van een dergelijk programma kan de leerling bijvoorbeeld inzicht verschaffen in een geautomatiseerd vliegtuigboekingssysteem.
Tot besluit het bedoelde programmaatje:

```
START
 lees klantgegevens
 haal vluchtgegevens
 boek zomogelijk
STOP

SUB lees klantgegevens
 vlucht := LEES           ;vluchtnummer
 tijd := LEES             ;aantal dagen vooruit
 aantal := LEES           ;gewenst aantal stoelen
END

SUB haal vluchtgegevens
 RIJ (1:tijd) vrije stoelen
 vrije stoelen := BESTAND(vlucht)
END

SUB boek zomogelijk
 ALS vrije stoelen(tijd) < aantal DAN
   REGEL := "slechts plaatsen voor",vrije stoelen(tijd)
 ANDERS
   vrije stoelen(tijd) := vrije stoelen(tijd) - aantal
   BESTAND(vlucht) := vrije stoelen
   REGEL := "geboekt voor ",aantal
 ALSEND
END

KLAAR
```

Literatuur

(1) Ontleend aan gegevens van het Rijks Inkoop Bureau.
(2) Deze term is gebaseerd op formuleringen van Prof. Davidse. Zie de *Nota Onderwijs en Informatietechnologie*, uitg. Staatsdrukkerij te Den Haag.
(3) Prof. Dr. E.W. Dijkstra, *A short introduction to the art of programming*, uitg. Technische Hogeschool Eindhoven, EWD 316.
(4) Uit S.C. Kleene, *Introduction to metamathematics*, Wolters-Noordhoff Publishing Company and North-Holland Publishing Company, 1971.
(5) Zie o.a. Pinxteren c.s. in *De school met den computer*, (Intermediair 42, 22 oktober 1982.