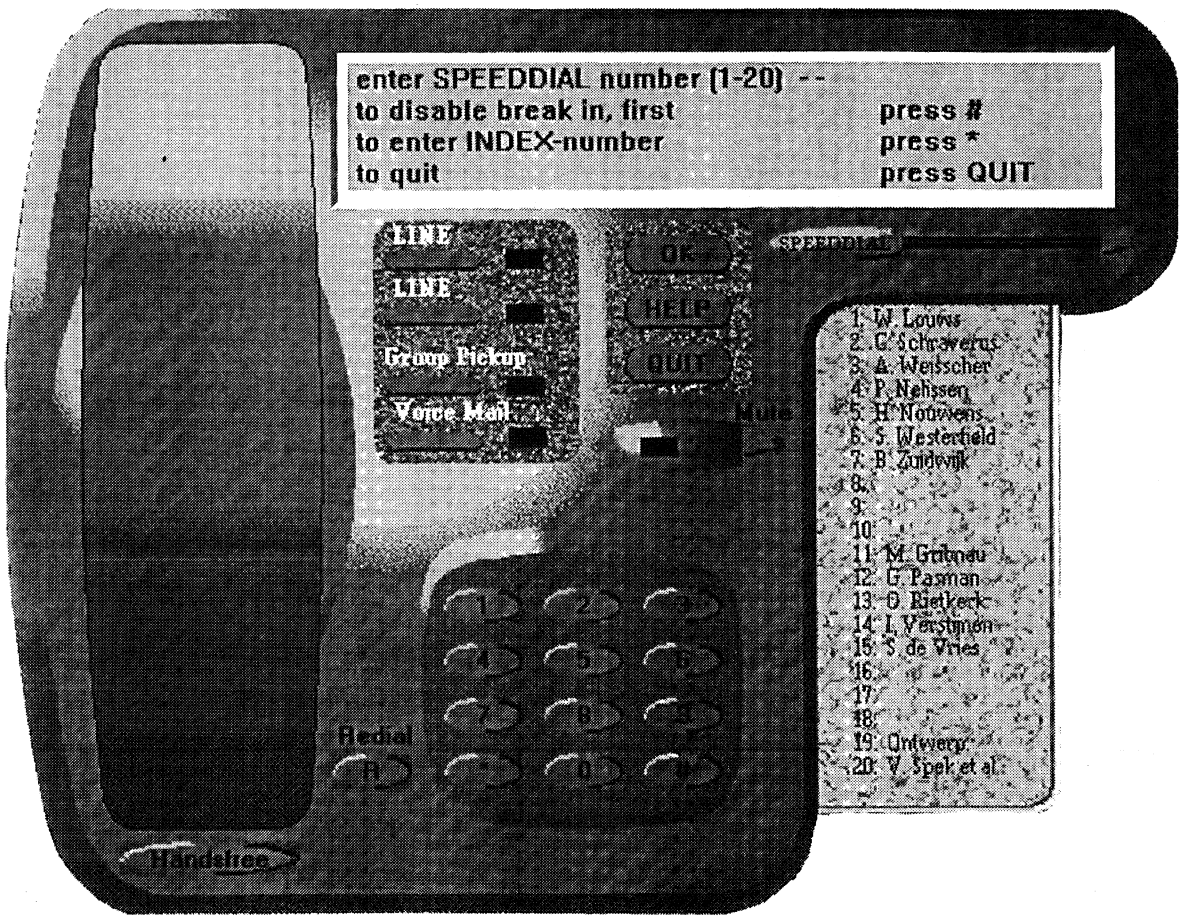
IDE132A (613)
Readernr. 12214

Informatica 1

Practicumhandleiding

November 1997

Ir. M.P.A.J. de Hoogh et al.



TU Delft
Technische Universiteit Delft

Faculteit van het Industrieel Ontwerpen
Vakgroep Produkt- en systeemergonomie

m.p.a.j. dehlooghe@io.tudelft.nl

INHOUD

1. INLEIDING

In Informatica 1 ga je leren programmeren en wel in en m.b.v. Visual Basic.
Deze practicumhandleiding bevat voldoende informatie om Informatica 1 met succes te voltooien.
Informatie in dit hoofdstuk vervangt eerdere informatie als b.v. in papieren patroon.
Eventuele verdere aanwijzingen zul je op de publicatieborden aantreffen.

De studielast bedraagt 60 SBU. Deze worden opgedeeld in 8 college-uren, 24 practicumuren en 28 zelfstudie-uren. De zelfstudie bestaat uit het bestuderen van deze practicumhandleiding, het maken van huiswerkopdrachten, en het voorbereiden van oefeningen en tentamen.

In blok 3 volg je de colleges en neem je deel aan de eerste drie oefeningen.
In blok 4 neem je deel aan de laatste drie oefeningen en het tentamen.
In onderstaand overzicht zie je alle activiteiten die van je verwacht worden.

Blok 3	**Activiteit**
Week 1	College 1. Gegevens Bestuderen practicumhandleiding Hoofdstuk 1 t/m 6, § 10.1 en 10.2
Week 2	College 2. Algoritmen Huiswerkopdrachten maken
Week 3	Oefening 1. Visual Basic Step-by-step 1 lessen 1, 2, 4 en 5 Bestuderen practicumhandleiding Hoofdstuk 7 m.u.v. § 7.4 en 7.7
Week 4	College 3. Objecten Oefening 2. Visual Basic Step-by-step lessen 6, 7, 9, 10 Bestuderen practicumhandleiding § 7.4, § 7.7, §10.3 t/m 10.5
Week 5	College 4. Te hanteren aanpak bij het maken van de eindopdracht Oefening 3. Voorbereiding eindopdracht

Blok 4	**Activiteit**
Week 1	Maken en inleveren opzet eindopdracht
Week 2 t/m 4	Oefening 4 t/m 6. Maken eindopdracht
12 maart 1998	Tentamen
	Herkansingstentamen

Colleges
De colleges ondersteunen de te leren stof. Ze vormen een aanvulling op de in de practicumhandleiding wellicht wat droog en compact beschreven stof.

Oefeningen
Aanwezigheid tijdens de oefeningen is verplicht. Neem **vooraf** contact op mij (kamer 03.41, tst. 2733) als je niet kunt zodat we kunnen kijken wanneer jij en je partner de oefening in kunnen halen. Afwezigheid zonder voorafgaande verwittiging is alleen bij aantoonbare overmacht toegestaan.

In de eerste twee oefeningen maak je kennis met de basismogelijkheden van Visual Basic aan de hand van een aantal lessen als beschreven in Microsoft's Visual Basic 5.0 Step-by-step boek. Aan het begin van de oefening krijg je een map met lessen die je aan het einde weer in moet leveren!

Tijdens de derde oefening ga je een bestaand Visual Basic programma – als beschreven in deze practicumhandleiding in § 7.4 – aanpassen. Het is belangrijk dat je dit programma van te voren heb bestudeerd en begrepen! Anders kun je snel genoeg aan de slag. Deze oefening helpt je je te realiseren wat je wel en niet begrepen hebt uit de practicumhandleiding en van Visual Basic. Het resultaat wordt weliswaar niet beoordeeld, maar benut deze oefening ten volle, ook door vragen te stellen over wat je niet snapt.

Aan het eind van de derde oefening krijg je een eindopdracht – tot het maken van een Visual Basic programma tijdens de overige drie oefeningen - uitgereikt. Het ontwerp daarvan moet je in de week voorafgaande aan de vierde oefening, dus in week 1 van blok 4, ter beoordeling inleveren. Zonder ontwerp kun je niet aan de eindopdracht beginnen! We kijken het ontwerp na en brengen zonodig correcties aan. Voor het gemaakte werkstuk krijg je een cijfer!

Tentamen

Op het tentamen wordt getoetst wat je tijdens het practicum en uit de practicum-handleiding hebt geleerd. De inhoud van de practicumhandleiding vormt de stof. Het boek 'Living with Computers' bevat in hoofdstuk 14 en 15 en in bijlage C aanvullende informatie die van pas kan komen maar niet getoetst wordt.

Eindcijfer

Het eindcijfer voor Informatica 1 is het gemiddelde van het cijfer behaald voor het tentamen en het werkstuk.

Studietrajecten

Beschik je al over enige programmeerervaring en wil je bekend raken met de object-georiënteerde mogelijkheden van Visual Basic dan raden wij je aan:

- Tijdens de derde oefening met de object-georiënteerde versie van de aan te passen applicatie aan de slag te gaan (ZEKA_O.VBP). De ontwikkeling ervan staat in § 7.7 beschreven. Bestudeer deze vooraf.
- Een partner te kiezen die ook al enige programmeerervaring heeft.
- Een object-georiënteerde versie van de te beoordelen applicatie te maken.

Heb je geen programmeerervaring, dan staat het je weliswaar vrij hetzelfde te ondernemen, maar dan loop je meer risico. Gebruik dan de derde oefening om na te gaan of je voldoende van object-oriëntatie in Visual Basic begrijpt om met succes een object-georiënteerde versie van de te produceren applicatie te kunnen vervaardigen.

Zij die een object-georiënteerde versie produceren kunnen rekenen op een extra punt bonus.

Overigens: alle informatie in de practicumhandleiding hoort bij de stof die op het tentamen wordt getoetst, dus ook object-oriëntatie ook al heb je dit niet toegepast in de eindopdracht! § 7.7 hoef je dan niet per se te bestuderen.

Schriftelijk studiemateriaal

Stukjes in deze practicumhandleiding die voorzien zijn van een doorgetrokken lijn in de kantlijn kunnen in eerste instantie bij de bestudering worden overgeslagen.

De hoofdstukken 14 en 15 en bijlage C uit het boek Living with Computer bevatten aanvullende informatie die echter niet getoetst zal worden.

2. GEGEVENS

In Living with Computers wordt veel aandacht geschonken aan de computer, computerprogramma's en de makers ervan. Aan de functie van die programma's, het verwerken van gegevens, wordt veel minder aandacht besteed.
In dit hoofdstuk besteed ik aandacht aan de basisbegrippen die met gegevens te maken hebben en probeer ik enig inzicht te verschaffen in de ontwikkelingen die tot de huidige situatie hebben geleid.

2.1 Inleiding

Gegevensopslag en gegevenstransport stonden vooral in de begindagen van de computer in dienst van de gegevensverwerking: de computer werd met name gebruikt voor wetenschappelijke toepassingen, zoals het analyseren van gegevens om er modellen uit af te leiden of deze modellen te verifiëren.
Tegenwoordig wordt de computer meer en meer gebruikt voor communicatie-doeleinden: de verwerking dient de communicatie.
Gegevensopslag en -transport vervullen een onontbeerlijke functie in communicatie.
Gegevensopslag is nodig teneinde er in een later stadium nog over te kunnen beschikken: hierbij gaat het dus om het overbruggen van tijd.
Gegevenstransport is nodig om de gegevens naar een andere plaats te brengen: hierbij gaat het dus om het overbruggen van plaats. Dit kan natuurlijk altijd mechanisch d.w.z. door fysiek transport van de gegevensdrager, maar sinds ongeveer drie generaties beschikken we ook over andere manieren waarbij het bericht zich 'zelfstandig' voortplant (verplaatst) door een medium. (Welke?)

2.2 Back to basics

Wat wordt onder gegevens verstaan? Feitelijk kan alles wat we waarnemen gegevens worden genoemd. In de praktijk echter wordt alles wat vastgelegd - opgeslagen - kan worden als gegevens beschouwd.
Gegevens worden vastgelegd op **gegevensdragers**. Bij het vastleggen wordt gebruik gemaakt van een bepaalde **gegevensvoorstelling** of -**representatie**.
De gegevensvoorstelling is **symbolisch** of **niet-symbolisch** (meer of minder natuurgetrouw). Bij de symbolische gegevensvoorstelling worden de gegevens m.b.v. symbolen (patronen zonder evidente betekenis), zgn. **tekens,** vastgelegd.
De betekenis die aan gegevens wordt toegekend wordt **informatie** genoemd. Het subtiele verschil tussen gegevens en informatie is dat informatie onze kennis doet toenemen, gegevens niet.
De betekenis van een symbolische gegevensvoorstelling - de informatie die de gegevens bevat - is gebaseerd op gemaakte afspraken. Deze afspraken worden een **taal** genoemd. De tekens die gebruikt mogen worden worden het **alfabet** van de taal genoemd. De regels aangaande de wijze waarop de tekens mogen worden gecombineerd wordt de **syntaxis** van de taal genoemd. De betekenis die aan de combinaties moet worden toegekend wordt de **semantiek** van de taal genoemd; de aanvullende betekenis die door toegevoegde gebaren ontstaat de **pragmatiek** van de taal. Alleen diegenen die de taal kennen, kunnen de informatie uit de gegevens halen.

Naast gegevensrepresentatie bij opslag vindt ook gegevensrepresentatie bij transport plaats.
Bij communicatie - het uitwisselen van gegevens - is altijd sprake van twee partijen [1]. De partij die de andere partij iets kenbaar wil maken wordt de **bron** of **zender** genoemd, de andere partij de bestemming of ontvanger. Datgene wat de zender kenbaar wil maken wordt het **bericht** of de **boodschap** genoemd. Het materiaal of de substantie die het transport van bron naar bestemming mogelijk gemaakt wordt het (communicatie)**kanaal** genoemd. Bij het transport door het communicatiekanaal kunnen afwijkingen of fouten ontstaan als gevolg van storingen of ruis die op het kanaal inwerken. Bij telecommunicatie wordt wat daadwerkelijk over het kanaal wordt getransporteerd het **signaal** genoemd.
Communicatie kan pas geslaagd worden genoemd als de ontvanger het bericht begrijpt d.w.z. in staat is de informatie die het bericht bevat te begrijpen: zender en ontvanger moeten dezelfde taal 'spreken'.

2.3 Ontwikkelingen

Alle ontwikkelingen die er op het gebied van de communicatie zijn geweest kunnen gezien worden als - geslaagde - pogingen tot het wegnemen van beperkingen die aan communicatie tot dan toe kleefden.
Het begon uiteraard met de directe communicatie tussen mensen.

Inherente beperkingen directe communicatie
Directe communicatie - via gebaren of (stem)geluid - tussen mensen kent een aantal beperkingen (in plaats en tijd), want zender en ontvanger moeten zich op hetzelfde moment binnen een bepaalde afstand van elkaar bevinden:
- Visuele overdracht (m.b.v. gebaren) is minder geschikt als zich obstakels tussen de zender en de ontvanger bevinden of als er te weinig licht is.
- Auditieve overdracht ('spraak') heeft een aanzienlijk kleiner bereik dan visuele overdracht, omdat naarmate de afstand tot de zender toeneemt, de sterkte van het signaal afneemt.

Beide vormen van directe communicatie vullen elkaar dus aan!
Veelvuldige afwezigheid van licht (in ver van de evenaar verwijderde streken) en/of de aanwezigheid van obstakels (b.v. in bergen of bossen) moet een stimulans zijn geweest voor de ontwikkeling van auditieve communicatie.
Directe communicatie op 'verschillende' tijdstippen is echter niet mogelijk: de boodschap is slechts gedurende een korte tijd beschikbaar.
Deze beperkingen kunnen alleen met behulp van externe hulpmiddelen geheel of gedeeltelijk worden opgeheven. In de loop der tijd zijn steeds betere manieren ontdekt om deze beperkingen op te heffen.

Boodschappers
Van de twee soorten beperkingen - die in plaats en tijd - is de beperking in plaats het minst stringent. Dit probleem kan het eenvoudigst - zonder extra hulpmiddelen - worden opgelost door het dichter bij elkaar brengen van zender en ontvanger. Uiteraard kost dit een bepaalde hoeveelheid moeite (energie) die men er natuurlijk wel voor over moet hebben. Wanneer echter zender en ontvanger hiertoe niet in staat kunnen of willen zijn, kan een boodschapper worden gebruikt. Dit brengt dan weer het extra risico met zich mee dat het bericht toevallig of moedwillig wordt vergeten of niet overgedragen. Het gebruik van niet-menselijke gegevensdragers (postduiven bijvoorbeeld) dringt zich hier dus pas op zodra een oplossing moet worden gevonden op de onbetrouwbaarheid van het gebruik van menselijke boodschappers!)

Schrift

Al is het gebruik van externe gegevensdragers bij het overbruggen van een (te grote) afstand gewenst, bij het overbruggen van tijd is het noodzakelijk. Zo ontstond langzamerhand geschreven taal en wel uit enerzijds het vereenvoudigen en abstraheren van afbeeldingen tot tekens en anderzijds uit het koppelen van klanken aan (dergelijke) tekens. Ook werden tekens, cijfers, bedacht voor het representeren van getallen. (Het getal 0, overigens, was aan het begin van de jaartelling nog niet 'uitgevonden'.)
Geschreven berichten kunnen voor lange(re) tijd worden bewaard en kunnen bijvoorbeeld gebruikt worden om anderen aan gemaakte afspraken te herinneren. De boodschap kan niet (in ieder geval minder gemakkelijk) verloren gaan of worden verminkt. De boodschap hoeft niet opnieuw te worden verzonden; een menselijke boodschapper is niet langer per se noodzakelijk.

Wel veroorzaakte het gebruik van geschreven taal een aantal nieuwe problemen: mensen moest worden geleerd de tekens te begrijpen: ze moesten de taal leren. De uitvinding van de boekdrukkunst moet een enorme stimulans zijn geweest tot standaardisatie èn invoering van het taalonderwijs, alhoewel er nu, bijna vijfhonderd jaar later, in onze samenleving nog mensen zijn die nooit goed hebben leren lezen of schrijven.

Code

Een **code** is, in de volksmond, de weergave van een bericht in de een of andere onbekende taal. Elke **symbolische** representatie kan als een representatie in code worden beschouwd omdat er geen sprake is van een direct - zonder kennis van de taal - te begrijpen representatie.
Het omzetten van een bericht uitgedrukt in de ene taal (de **brontaal**) in een andere taal (de **code**- of **doeltaal**) wordt coderen genoemd. Het weer uitdrukken van het bericht in de brontaal wordt **decoderen** genoemd. De tekens uit het alfabet van de codetaal worden codes genoemd. Een goed voorbeeld van een codetaal die een eeuw lang voor het telegraafverkeer is gebruikt is morse.

De morsecodes behorende bij een aantal - in het Nederland - gangbare tekens [2]					
a	.-	n	-.	1	.----
b	-...	o	---	2	..---
c	-.-.	p	.--.	3	...--
d	-..	q	--.-	4	....-
e	.	r	.-.	5	.....
f	..-.	s	...	6	-....
g	--.	t	-	7	--...
h	....	u	..-	8	---..
i	..	v	...-	9	----.
j	.---	w	.--	0	-----
k	-.-	x	-..-		
l	.-..	y	-.--		
m	--	z	--..		

De codetaal hoeft niet per se een alfabet met hetzelfde aantal tekens te hebben als de brontaal. Zo kent het morse-alfabet slechts twee tekens: . (dot of *di*(t), punt) en - (dash of *dah*, streep). Voor 2 tekens uit ons alfabet kunnen de punt en de streep worden gebruikt. Voor representatie van de overige tekens zijn twee of meer morsetekens nodig. Er is daarbij gekozen voor een codering waarin de minder frequente voorkomende tekens meer morsetekens gebruiken, zodoende ervoor zorgende dat een bericht gemiddeld uit een zo klein mogelijk aantal morsetekens zou bestaan.

(Daarnaast werd bij het verzenden van een telegraafbericht wel van afkortingen gebruik gemaakt: de eerste vorm van compressie!)
Derhalve, wanneer omzetting moet plaatsvinden naar een taal met een kleiner alfabet, dan zijn voor de codering van een teken in de brontaal (soms) meerdere tekens in de doeltaal nodig. Alle soorten gegevens kunnen (meer of minder natuurgetrouw) m.b.v. tekst worden beschreven. Voor beeld- en geluidgegevens zijn er betere - d.w.z. meer natuurgetrouwe - 'beschrijvingen' mogelijk.

Geluidgegevens
Geluiden binnen een bepaald frequentiebereik kunnen met de stem redelijk worden ge(re)produceerd (alsook worden gehoord). Natuurlijk: worden de geluiden door mensen m.b.v. muziekinstrumenten gemaakt, dan is weergave ervan met dezelfde muziekinstrumenten het meest natuurgetrouw. Tot het moment waarop geluid direct kon worden opgeslagen en weer gereproduceerd moest men zich met het beperkte - want ternauwernood geschikt voor muziek - notenschrift behelpen.
Op zeker moment werd de opslag van geluid mogelijk, eerst nog analoog en mechanisch (afleesbaar) op cilinders en platen, later analoog en magnetisch op banden en cassettes en sinds kort digitaal en optisch (afleesbaar) op CD's.
Mechanisch en elektrisch transport van het opgeslagene van zender (opnemer) naar ontvanger was nodig totdat de radio werd uitgevonden: de geluiden konden in radiogolven worden gecodeerd en door de 'ether' worden verzonden, opgevangen en weer worden gedecodeerd tot geluid. Dankzij de uitvinding van de radio (en overige geluidsapparatuur) was het niet langer noodzakelijk om geluid in notenschrift te coderen, en ook weer te decoderen. Natuurlijk, wilde men elders - anders dan door uit het hoofd naspelen - dezelfde muziek kunnen produceren, dan moesten de noten weer wel worden geproduceerd. Computers en aangesloten geluidsapparatuur kunnen hedentendage worden gebruikt om muziek uitgedrukt in noten ten gehore te brengen en - vice versa - voor het uitdrukken van gespeelde noten in het notenschrift.

Beeldgegevens
Mensen zijn zonder hulpmiddelen niet in staat beelden te (re)produceren.
Zou taal wel zo'n succes zijn geweest als we - d.w.z. ons lichaam - wel in staat waren geweest beelden natuurgetrouw te tonen? Het kost(te) immers heel wat minder tijd om de werkelijkheid in schrift te beschrijven, dan deze uit te beelden! Natuurlijk: de (visuele) werkelijkheid kan nooit zo nauwkeurig met woorden worden beschreven, als met beeld, maar dit nadeel viel in het niet bij de hogere snelheid waarmee tekst kon worden geproduceerd. Bovendien konden veranderingen gemakkelijk met tekst worden beschreven dan met (vele) afbeeldingen. Ook was niet altijd alles van belang: in tekst kon net datgene worden weergegeven dat interessant was voor de ontvanger(s). Het valt echter niet te ontkennen dat beelden vaak een grotere (emotionele) invloed hadden en hebben.
Tot ongeveer een eeuw geleden vergde het heel veel training om (op schilderijen e.d.) een goede natuurgetrouwe weergave van een beeld te reproduceren: de fotografie stelde eenieder in staat de (visuele) werkelijkheid vast te leggen. Eerst nog in zwart-wit, later ook in kleur. Eerst langzaam, later met een zodanige snelheid (m.b.v. filmapparatuur) dat de indruk van beweging kon worden gewekt. Beweging kon voorheen al helemaal niet worden vastgelegd, alleen gesuggereerd. Wel gaat bij de projectie van de vier-dimensionele werkelijkheid op twee dimensies veel verloren.
Mechanisch transport van beeldmateriaal moest plaatsvinden tot de uitvinding van de televisie(camera). (De uitvinding van de fax even buiten beschouwing latende.)
Overigens was het op dat moment nog niet mogelijk om die beelden vast te leggen (voor uitzending in een later stadium). Het duurde echter niet lang voordat naast ruimte ook tijd kon worden overbrugd: grote hoeveelheden beeldmateriaal konden magnetisch op banden worden gezet.

Relevante uitvindingen [3, 6]

	TRANSPORT	TAAL & REKENEN		BEELD & GELUID
ca. -15000				grotschilderingen (Lascaux)
ca. -3000	wiel	spijkerschrift (klei) tellen hiërogliefen voor religieuze doeleinden papyrus		
ca. -1000		Grieks alfabet (met klinkers)		
ca. -700	houten wegen (Romeinen)	hiëroglyfen voor algemeen gebruik		
ca. -450		abacus (telraam)		
ca. 105			papier (China)	
ca. 200	wegkaarten			
ca. 1000			vijfregelig notenschrift	
ca. 1450			boekdrukkunst	
1550-1600	wetgeving m.b.t. onderhoud van wegen (Engeland) zeekaarten		potlood	
1600-1650		logaritme-tabellen rekenlineaal optelmachine bedacht		
1650-1700		mechanische rekenmachine	tijdschrift, krant, post	
1700-1750	stoomschip, trein	binaire getallen [5]	dagblad	fotografisch materiaal
1750-1800	school voor weg- en waterbouw (Frankrijk) bestrating onderzeeër, ballon, luchtschip			visuele telegrafie
1800-1850	stoomlocomotieven paardentram fiets, helicopter	ponskaarten machinale drukpers stalen pen, vulpen	fotografie papier machinaal maken elektriciteit	elektrische telegrafie magische lantaarn
1850-1900	motorfiets, auto elektrische trein elektrische tram zweefvliegtuig elektrische voertuigen	typemachine kleurendruk	filmprojector filmcamera	stereoscopische camera microfilm, telefoon grammofoon röntgenapparatuur radiografie magnetische recorders
1900-1925	vliegtuig autobussen hovercraft		radar	film, televisie radio-uitzendingen
1925-1950	raket straaljager	teletypewriter balpen	communicatietheorie transistor	talkies, kleurenfilm FM, computer, stereo instant fotografie holografie, LP's
1950-1960	satellieten	FORTRAN, COBOL, Algol '60		NTSC, SECAM video(tape)recorder transistorradio
1960-1970	eerste man in ruimte	BASIC	IC's, laser magnetische schijven minicomputer rekenmachine	cassette-recorder
1970-1980	eerste man op maan	Pascal	Floppy disk micro-computers harde schijf VisiCalc	Dolby video-disk CD-audio (DAD) Walkman
1980-1990	Space Shuttle	C	Portable, IBM PC Lotus 123 CD-ROM (magneto-)optische schijven	Compact Disc CD-i

2.4 Soorten gegevensvoorstellingen

Gegevensvoorstellingen kunnen op verschillende manieren worden geklassificeerd.

Er wordt onderscheid gemaakt tussen digitale, analoge en iconische gegevensvoorstellingen [7].
Een gegevensvoorstelling is:
- **iconisch** als tekeningen, diagrammen, schaalmodellen en schetsen gebruikt worden.
- **analoog** als een natuurkundige grootheid wordt gebruikt, zodat iedere waarde daarvan binnen een bepaald interval overeenkomt met een voor te stellen veranderlijke (b.v. de stand van de wijzer van een klok voor een tijdstip, spanning voor geluidssterkte in audio-installaties);
- **digitaal** als een eindig aantal verschillende tekens (een alfabet) gebruikt wordt;

Er kan ook onderscheid gemaakt worden tussen **mens**- en **machinegerichte gegevensvoorstellingen** [4]. Een voorbeeld van een machine-gerichte voorstellingswijze is de streepjescode. Een voorbeeld van een gegevensvoorstelling die zowel mens- als machinegericht is, zijn de tekens behorende tot het OCR-lettertype (zie blz. 204 van LwC).

Literatuur

[1] D.E. Boekee en J.C.A. van der Lubbe, Informatietheorie, Delftse Uitgevers Mij., 1988, Delft.
[2] Werner W. Diefenbach, Morselehrgang für den Funkamateur, Franzis-Verlag GmbH, 1980, München.
[3] G.W.A. Dummer, Electronic inventions & discoveries (3rd revised and expanded ed.), Pergamon Press Ltd., 1983, Oxford.
[4] J. Faber (red.), Elementaire informatica: de middelen, Pandata BV, 1988, Rijswijk.
[5] A. Glaser, History of binary and other nondecimal numeration, Tomash Publishers, 1981.
[6] I. McNeil (ed.), An Encyclopedia of the History of Technology, Routledge, 1990, Londen.
[7] PBNA, poly automatiserings zakboekje (2e druk), Koninklijke PBNA, 1984, Arnhem.

3. GEGEVENS IN DIGITALE COMPUTERS

Alle informatie die we in een computer willen opslaan moet worden gedigitaliseerd d.w.z. slechts een eindig aantal waarden kan (exact) worden gerepresenteerd. Dit is geen probleem voor de representatie van tekst en – een eindige verzameling - gehele getallen, maar de digitalisatie van analoge en iconische gegevens, zoals niet-gehele getallen, geluid en beeld, introduceert fouten. Het copiëren van digitale gegevens kan echter wel zonder fouten geschieden, zodat kwaliteit van de digitalisatie bepalend is voor de kwaliteit van de digitale gegevens. De gemaakte fouten kunnen bij het rekenen met en/of bewerken van niet-exacte digitale gegevens elkaar versterken, met verder kwaliteitsverlies als gevolg. Ook dan is de kwaliteit van de oorspronkelijke digitalisering van belang: hoe hoger, hoe acceptabeler de eindkwaliteit. Beperkingen bij het bereiken van de hoogstmogelijke kwaliteit zijn de daaraan verbonden kosten en natuurlijk de beschikbare techniek. Hoe hoger de kwaliteit des te duurder is het digitalisatie-proces, des te meer opslagcapaciteit en tijd is er voor nodig en des te geavanceerdere technieken zijn er nodig. De standaards voor de representatie van niet-gehele getallen zijn inmiddels wel gezet, die voor de representatie van beeld en geluid ontwikkelen zich nog steeds.

In dit hoofdstuk concentreer ik me op de representatie van tekst, getallen, beeld en geluid in digitale computers. Allereerst bespreek ik kort de mogelijkheden van de hedendaagse computer voor het opslaan van gegevens.

Het allerkleinste geheugenelement
Elk kleinste geheugenelement van een computer kan zich (op een gegeven moment) in één van twee verschillende toestanden bevinden, die met 0 resp. 1 worden aangeduid. (Elk tweetal tekens zou voor de representatie van deze twee toestanden gebruikt kunnen worden, dus verkijk je niet op het gebruik van deze twee cijfertekens!) Zo'n geheugenelement wordt een **bit** (afkorting van ***binary digit***) genoemd. Het bit is in staat het laatste (gelijkspannings)signaal dat er op is gezet (vergelijkbaar dus met een 0 of 1) te 'onthouden', en geeft dit signaal op verzoek weer af.

Kleinste nog apart op te vragen geheugenelement
In de hedendaagse digitale computers is het niet mogelijk slechts een bit per keer uit het geheugen te halen. Opslag van gegevens vindt in groepen van 8 bits plaats. Een groep van 8 bits wordt een **byte** (een samentrekking van ***by eight***) genoemd. In een byte kunnen 256 verschillende combinaties worden opgeslagen (0000 0000, 0000 0001, 0000 0010, 0000 0011 t/m 1111 1111). (In het algemeen kunnen in k bits 2^k combinaties dus verschillende waarden worden opgeslagen.)
De **woordlengte** van de computer bepaald hoeveel bits per keer uit het geheugen wordt opgehaald. Dit is gewoonlijk een veelvoud van 8 bits, een aantal bytes dus. De practicummachines hebben een woordlengte van 32 bits, het zijn 32-bitsmachines.

Opslagmogelijkheden in digitale computers
In een digitale computer kunnen zich verschillende soorten geheugens bevinden met verschillende functies. Daarvan zijn (alleen) registers en RAM-geheugen **vluchtig** (***volatile***): de inhoud ervan gaat bij uitzetten verloren! Wat in het ROM-geheugen en op achtergrondgeheugen (schijf, diskette, CD-ROM) staat gaat dan niet verloren.

Registers bevinden zich in de CVE en worden gebruikt voor de opslag van (tussen)resultaten van uitgevoerde opdrachten. (Elke opdracht wordt in veel soorten computers uitgevoerd door het in hardware verankerde microprogramma. De functie van deze schakelingen is het dan een opdracht op te halen, te interpreteren en uit te voeren door toepasselijke signalen te versturen.)

Het **RAM-geheugen** (ook wel **primair** of **hoofdgeheugen** genoemd) bestaat uit een rij (opeenvolgende) geheugenelementen (meestal bytes). Elk geheugenelement wordt met een volgnummer aangegeven, wat een adres wordt genoemd. In dit geheugen aanwezige gegevens kunnen direct door de CVE worden opgehaald op basis van hun adres. Alleen opdrachten die zich hierin bevinden kunnen door de computer worden uitgevoerd.

Het **ROM-geheugen** is eveneens een rij geheugenelementen waarin zich o.a. opdrachten bevinden die de computer uitvoert bij het opstarten; deze geheugenelementen kunnen niet worden gewijzigd.

Geheugen, zoals harde schijven en floppy disks, waarop gegevens permanent kunnen worden opgeslagen worden **secundair** of **achtergrondgeheugen** genoemd.

Randapparatuur, als printers, beeldschermen maar ook harde schijven, zijn gewoonlijk via speciale insteekkaarten op de computer aangesloten. Deze insteekkaarten bevatten vaak ook registers en geheugenelementen. Het geheugen van de insteekkaart die aangesloten is op het beeldscherm, de zgn. videokaart, zal de (gecodeerde) inhoud van wat op het beeldscherm wordt weergegeven bevatten. Er wordt echter niet altijd van insteekkaarten gebruik gemaakt; het toetsenbord bijvoorbeeld is rechtstreeks op het moederbord van de computer aangesloten.
De registers in de insteekkaarten zullen vaak informatie bevatten die de werking ervan bepaalt. De computer communiceert via **poorten** met deze insteekkaarten.

3.1 Representatie van tekst

Een tekst (in het Nederlands, Engels, ...) kan beschouwd worden als een rij tekens. De tekens zijn afkomstig uit een eindige verzameling van tekens, uit een alfabet dus. Voor de binaire opslag van elk teken in het alfabet moet (vergelijkbaar met de morsecode) één unieke combinatie van nullen en enen worden gekozen. Hoe anders zou de tekst weer gereconstrueerd kunnen worden? Er zijn dus net zoveel verschillende combinaties (=codes) nodig als er tekens in het alfabet zitten. Dit zijn er ongeveer 90 voor ons alfabet. Als 7 bits gebruikt worden houden we toch nog ongeveer 40 combinaties over. Deze combinaties worden in de praktijk gebruikt voor **besturingskarakters**, d.w.z. combinaties die een speciale betekenis hebben voor randapparatuur als printers en beeldschermen. Natuurlijk kunnen ook meer bits gebruikt worden; acht is een logische keuze omdat de meeste computers toch alleen het beste kunnen werken met bytes.
Welk teken bij welke code hoort, kan in een tabel, de zgn. codetabel worden weergegeven. De ISO (International Standards Organization) heeft geprobeerd enige standaardisatie aan te brengen door de ontwikkeling van de 7-bits standaard **ISO-7**, waarvan ***ASCII (American Standard Code for Information Interchange***) de Amerikaanse variant is (of is het omgekeerd?).
Aangezien PC's bits in groepjes van 8 gebruiken zijn er een extra 128 combinaties die niet gedefiniëerd waren in de standaard ASCII-codetabel. Dit heeft aanleiding gegeven tot de ontwikkeling van de 8-bits standaard **ISO-8** of ***Extended-ASCII***-code.

De 8-bits-code **EBCDIC** van IBM afkomstig biedt een standaard voor dezelfde karakterset als de ISO-7, maar biedt dus ruimte aan 256 combinaties.

Teneinde een teken op het beeldscherm weer te kunnen geven moet de computer wel weten welk patroon (of vlek zo U wilt) moet worden weergegeven d.w.z. welke kleur de punten in een rechthoekig gebied op het beeldscherm moeten krijgen. Gewoonlijk bevat een computer in het ROM-geheugen van een standaard alfabet, de zgn. ***hardware character set***, de benodigde (bit)patronen. Zo'n patroon is feitelijk hetzelfde als een rasterafbeelding. Steeds vaker worden echter ook vector-afbeeldingen (zie § 3.3 Representatie van afbeeldingen) gebruikt voor de weergave van tekst. Toepassingsprogramma's, als tekstverwerkers, gaan gewoonlijk vergezeld van hele verzamelingen lettertypen, die dus voor elke letter uit een bepaalde character set een raster- of vectorafbeelding bevatten voor weergave ervan op beeldscherm of printer.

Elke karakterset kan momenteel echter niet meer dan 256 tekens bevatten, omdat elk teken in één byte wordt opgeslagen. Daarom moeten computers (of besturings-systemen) momenteel nog meerdere karaktersets, bijbehorende **codetabellen** en weergave-afbeeldingen bevatten, daar in de diverse landen diverse alfabetten worden gebruikt. Om het gebruik van meerdere codetabellen tegen te gaan is er hard gewerkt aan de acceptatie van een 16-bits karakterset, die gebruikt kan worden voor de opslag van ten hoogste 65536 tekens, ruim voldoende voor de meest gangbare alfabetten. Deze 'standaard' wordt in de wandelgangen **Unicode** genoemd en is al door een aantal bedrijven geaccepteerd. Microsoft heeft echter besloten Unicode niet te implementeren in Windows 95.

Ook de tekens die worden gebruikt voor het weergeven van getallen, de zgn. **cijfertekens** (0 t/m 9) komen voor in de codetabel. We staan er in de regel zelden bij stil dat wij getallen gewoonlijk in de vorm van rijen cijfertekens representeren. Wij beschouwen zo'n rijtje cijfertekens als hèt getal. Wij hebben weliswaar met een dergelijke tekstpresentatie leren rekenen maar voor een computer is dit inefficiënt.

Invoer van tekst in een computer

De momenteel meest gebruikte manier om tekst in een computer in te voeren is via het toetsenbord. (De invoer m.b.v. Optical Character Recognition (OCR) is in opmars.) Het toetsenbord stuurt bij elke verandering (zowel bij het indrukken als loslaten van een toets) een signaal naar de computer. Een programma in het ROM-geheugen reageert op elke verandering. Gewoonlijk zet deze bij het loslaten van een toets de code die bij die toets hoort in een buffer, de toetsenbordbuffer. In die buffer is slechts plaats voor een beperkt aantal codes; het is tevens gebruikelijk om als de buffer vol is, wat kan gebeuren als een programma niet snel genoeg codes uit de buffer leest, een geluidje te laten horen, zodat de gebruiker weet dat de buffer vol is. (Probeer het maar uit!)

Uitvoer van tekst bij een computer

Tekst wordt hetzij op een beeldscherm hetzij op een printer weergegeven.
Bevindt het beeldscherm zich in een teksttoestand (de alfanumerieke toestand) dan zal het video-geheugen alleen de codes van de tekens die weergegeven moeten worden bevatten. De bij een teken behorende afbeelding wordt dan pas bij weergeven opgehaald (b.v. uit het ROM als de hardware character set wordt gebruikt). In een grafische toestand bevat het video-geheugen de afbeelding zelf.
(DOS werkt altijd in de teksttoestand en Windows in een grafische toestand!)

Inmiddels ouderwetse printers ontvingen van de computers de codes van de tekens die moesten worden weergegeven. Voor het naar een nieuwe regel of nieuwe bladzijde gaan bevatte de tekst speciale besturingstekens, d.w.z. binaire codes die geen tekst, maar besturingsinformatie representeerden. Zo werd het gebruikelijk om ASCII-code 13 te gebruiken om naar het begin van de regel te gaan: dit teken wordt **Carriage Return** (**CR**) genoemd. ASCII-code 10 werd het teken om een printer een regel op te laten schuiven: de **Line Feed** (**LF**). Naar het begin van de volgende regel gaan kon dus worden gerealiseerd door een CR-teken en een LF-teken naar de printer te sturen. In ASCII-teksten, die met een gewone teksteditor (b.v. Window's Kladblok) kunnen worden bewerkt, staan tussen elk tweetal regels nog deze twee tekens (alhoewel de teksteditor deze niet laat zien!).
De printers van tegenwoordig zijn in staat om afbeeldingen weer te geven en kunnen dus i.p.v. de tekstcodes ook gewoon de bijbehorende afbeeldingen ontvangen. Dit houdt echter in dat meer gegevens naar de printer moet worden gezonden: het kan zinvol zijn de afbeeldingen van diverse veelgebruikte lettertypes in de printer op te slaan, zodat de computer deze niet naar de printer hoeft te sturen!

3.2 Representatie van getallen

Er werd gezocht naar een zodanige wijze van opslag, een zodanig codering van getallen dat daarmee correct en snel door computers kon worden gerekend.
Is het mogelijk om alle getallen zodanig te coderen (in een eindig aantal bits) dat ze van elkaar te onderscheiden zijn? Alle getallen, waarvan er oneindig veel zijn, kunnen natuurlijk nooit in een eindig aantal bits, waarmee dus ook maar een eindig aantal unieke combinaties te vormen is, exact worden gerepresenteerd. Dit betekent, dat moet worden gekozen welke getallen wel exact en welke niet exact zullen kunnen worden opgeslagen. In de praktijk wordt een verschillende representatie gebruikt voor de opslag van gehele resp. niet-gehele getallen.

3.2.1 Representatie van gehele getallen

Twee vragen moeten worden beantwoord: hoeveel bits moeten worden gekozen en welke gehele getallen moeten exact kunnen worden gerepresenteerd?
Laten we eerst de tweede vraag beantwoorden en even veronderstellen dat k bits worden gebruikt. Met k bits kunnen 2^k verschillende codes en dus ook precies dit aantal verschillende gehele getallen worden gerepresenteerd. Welke 2^k gehele getallen moeten worden gekozen? Welke berekeningen met gehele getallen moeten wel exact kunnen worden uitgevoerd en welke niet? Nu worden in computers vaak tellers gebruikt, die regelmatig met 1 moeten worden opgehoogd of verlaagd. Het zou heel vervelend zijn als het resultaat van deze ophoging of verlaging niet (exact) gerepresenteerd zou kunnen worden opgeslagen! De exact gerepresenteerde getallen moeten elkaar daarom opvolgen.
Dit betekent dat alle getallen in een bepaald interval wel en alle getallen buiten dat interval niet exact gerepresenteerd moeten worden. Welk interval moet worden gekozen?
In de praktijk zijn 2 soorten intervallen in gebruik. Nl. een waarbij alleen niet-negatieve getallen exact worden gerepresenteerd nl. alle gehele getallen in het interval $[0,2^{k-1}]$, en een waarbij evenveel negatieve als positieve getallen exact worden gerepresenteerd nl. die in het interval $[-2^{k-1},2^{k-1}-1]$. Natuurlijk moet dan ook nog worden bepaald welke binaire code welk geheel getal voorstelt. Het is erg belangrijk dat gemakkelijk met de binaire codes kan worden 'gerekend' als waren het getallen. Feitelijk gaat het erom ervoor te zorgen dat de codes zodanig worden gekozen, dat er een minimum aan hardware nodig is om correct met de getallen te kunnen rekenen.

Representatie als binair getal

Een mogelijke keuze is die waarbij de binaire representatie van de gehele getallen als binaire code wordt gebruikt. Immers: de **binaire representatie** van een getal is net als de binaire code een rijtje nullen en enen. De 0 en de 1 worden dan beschouwd als cijfertekens. (De beslissing de twee toestanden waarin een bit zich kan bevinden met 0 resp. 1 aan te duiden kan hieruit worden verklaard.)
Een dergelijke representatie lijkt op de manier waarop wij gehele getallen noteren. Alleen maken wij niet gebruik van twee verschillende cijfertekens maar van tien. Ons **talstelsel** heet daarom het **tientallig** (of **decimale**) talstelsel. Wordt van slechts twee verschillende cijfertekens gebruik gemaakt dan wordt van het **tweetallig** (of **binair**) talstelsel gesproken.
Een in een dergelijk talstelsel m.b.v. cijfertekens weergegeven getal is een gewogen som van de gebruikte cijfers, waarbij elk gewicht geschreven kan worden als b^e, waarin b, de **basis**, het **grondtal** van het talstelstel is, 10 bij het decimale talstelsel en 2 bij het binaire talstelsel. De **exponent** e is voor het meest rechtse cijfer gelijk aan 0, voor het op een na rechtste cijfer gelijk aan 1, enzovoorts: deze is dus een functie van de positie van het cijferteken. Vandaar dat het decimale talstelsel net als het binaire talstelsel een **positiestelsel** wordt genoemd. (Het talstelsel dat de Romeinen gebruikten was geen positiestelsel!)
Bij de weergave van een bepaald getal in het een of andere talstelsel is het gebruikelijk het grondtal rechts van het getal iets lager weer te geven. Zo stelt 1010_2 het getal 10_{10} voor (oftewel tien).

Wat is er gemakkelijker dan de binaire representatie van een geheel getal als code te gebruiken? Dit gebeurt dan ook inderdaad als het om niet-negatieve gehele getallen gaat, maar bij negatieve getallen lukt dat niet! Want: voor de representatie van negatieve getallen gebruiken wij het minteken, maar een computer kent alleen de 0 en de 1 en andere tekens zijn er niet. Dit probleem kan worden opgelost door een van de bits te gebruiken voor het teken b.v. 0 als het om een positief getal gaat en 1 als het om het negatief getal gaat. Vaak wordt het meest linkse bit daarvoor gebruikt: dit wordt dan het **tekenbit** genoemd. Dan blijven k-1 bits over om de absolute waarde van het getal op de gebruikelijke wijze op te slaan (de absolute waarde van een getal is immers positief). Als een geheel getal op deze wijze gerepresenteerd wordt opgeslagen wordt van **de teken-groottenotatie (*sign-magnitude)*** gesproken.
Andere notatiewijzen maken de manipulatie door elektronische circuits echter gemakkelijker. (Bovendien kan het getal 2^{k-1} in de teken-groottenotatie niet worden gerepresenteerd.)

De excess notatie

Een alternatief is de ***excess-*2^{k-1}** notatie. Alle getallen worden daarin binair als positieve gehele getallen gecodeerd. Het allerkleinste getal (-2^{k-1}) wordt gecodeerd als 0_2. Het op een na kleinste getal als 1_2 enzovoort. Het grootste getal ($2^{k-1}-1$) als $(2^k-1)_2$.
Zo worden de getallen -4, -3, -2, -1, 0, 1, 2 en 3 in de excess-4 notatie (k=3) gecodeerd als 000_2, 001_2, 010_2, 011_2, 101_2, 110_2 en 111_2.

De 2-complement notatie

Het meest gebruikt is echter de **2-complement notatie**.
De niet-negatieve getallen (0 t/m $2^{k-1}-1$) worden weer gewoon als 0_2 t/m $2^{k-1}-1_2$ gecodeerd.
De negatieve getallen worden verder zodanig gecodeerd dat de som van dit getal met het bijbehorende positieve getal (modulo 2^k) gelijk aan 0 is. (Modulo 2^k wil zeggen dat een eventueel k+1-ste bit op de eerste plaats wat bij de optelling kan ontstaan wordt verwijderd, omdat de representatie nu eenmaal maar k nullen of enen kan bevatten.)

(Overigens: ook bij de *excess* notatie is de som gelijk aan 0.) Het bitpatroon van het negatieve getal is gemakkelijk uit die van het bijbehorende positieve getal te construeren en omgekeerd. Verander daartoe alle nullen in enen en omgekeerd, en tel er vervolgens (binair natuurlijk) 1 bij op. (Omdat 2^k niet kan worden voorgesteld is het niet mogelijk op deze wijze -2^k te vinden!) Aan het allereerste bit is te zien of het getal negatief (1) of niet-negatief is (0).

Omdat de negatie van een getal zo gemakkelijk te bepalen is de 2-complementnotatie zo populair. Het optellen zowel als aftrekken kan immers met dezelfde schakelingen gebeuren: een getal van een ander getal aftrekken is immers niets anders dan daarbij de negatie van het getal optellen. En de negatie is gemakkelijk te bepalen!
In de 2-complementnotatie is 2 gelijk aan 010_2 en -1 gelijk aan 111_2.
Het verschil van -1 en 2 is gelijk aan -1+(-2) oftewel gelijk aan $111_2 + 110_2 = 1101_2$.
De eerste 1 zou door de computer die met k=3 werkt worden genegeerd (omdat deze niet zou kunnen worden opgeslagen) en 101_2 zou worden opgeslagen, inderdaad gelijk aan -3 in de 2-complementnotatie!

Overflow
Tellen we de binaire codes van 2 en 3 bij elkaar op (010_2 en 011_2) dan is het resultaat 101_2, maar dit stelt het getal -3 voor en niet het getal 5 (wat immers niet in het exact te representeren interval [-4, 3] ligt). Het resulterende getal is groter dan het grootste exact te representeren getal (3). In dit geval (en ook als de werkelijke uitkomst kleiner zou zijn dan het kleinste nog exact te representeren getal) wordt van ***overflow*** gesproken. *Overflow* kan bij de 2-complement notatie gemakkelijk worden ontdekt omdat dan het tekenbit verkeerd is. Als twee positieve getallen bij elkaar worden geteld moet het tekenbit gelijk aan 0 zijn, en bij de optelling van twee negatieve getallen gelijk aan 1. Een aangezien op het optreden van overflow moet worden gereageerd moet deze situatie tenminste gemakkelijk kunnen worden vastgesteld.

De keuze voor k
De keuze voor k is gedeeltelijk afhankelijk van de mogelijkheden van de computer waarop het programma draait. Computers zijn gewoonlijk alleen in staat met gehele getallen te werken waarvoor k een veelvoud is van het (kleinste) aantal bits dat voor de opslag gebruikt wordt; en dit kleinste aantal is meestal 8. De grootte van de registers bepaalt om welk veelvoud het gaat! Bij computers gebaseerd op een Intel Pentium processor kan een register 32 bits bevatten, dus met 32-bits gehele getallen kan nog zonder problemen gerekend worden (en dan ook met gehele getallen die in 8 of 16 bits worden voorgesteld). Programma's die met gehele getallen willen werken bestaande uit een groter aan bits moeten zelf aangeven hoe met die getallen moet worden gewerkt (de computer kan het niet zelf!). In de tijd dat de grootte van een PC-register nog 16 bits was (bij de Intel 80286) boden vertaalprogramma's in eerste instantie mogelijkheden aan voor het gebruik van gehele getallen met k=16. Alleen gehele getallen in het interval [-32768, 32767] kunnen dan exact worden opgeslagen, wat in die tijd meestal voldoende was voor de gebruikte gehele getallen.
Pas later boden vertaalprogramma's standaardfaciliteiten voor het gebruiken van grotere gehele getallen. Maar tot nu toe wordt het gehele getal met k=16 (***integer*** genaamd) als hèt standaardtype gehele getal beschouwd, terwijl getallen die gebruikmaken van 32 bits, ***long integers*** worden genoemd.

Invoer en uitvoer van gehele getallen bij een computer
Invoer via het toetsenbord is altijd in de vorm van tekst. Dat betekent dat het programma in eerste instantie beschikt over de tekstrepresentatie van het getal, en dit is een andere dan de binaire getal-representatie en derhalve de tekstrepresentatie moet omzetten in de bijbehorende getalrepresentatie. Het spreekt voor zich, dat,

aangezien het natuurlijk heel vaak voorkomt dat getallen moeten worden ingevoerd en verwerkt, vertaalprogramma's over faciliteiten beschikken die hiervoor zorgen zonder dat de maker van het programma zich hierover zorgen hoeft te maken. De gebruiker hoeft slechts aan te geven van welk type (geheel getal) het gegeven is dat moet worden ingevoerd en het vertaalprogramma zorgt er bij vertaling voor dat de juiste conversie plaatsvindt.
Bij uitvoer vindt conversie plaats van getalrepresentatie naar tekstrepresentatie. Meestal is het mogelijk om nog invloed uit te oefenen op de wijze waarop de tekst moet worden gevormd. Bij gehele getallen meestal alleen om het aantal tekens waaruit de tekstrepresentatie moet bestaan. Een dergelijke opmaakvoorschrift heet een ***format***.

3.2.2 Representatie van niet-gehele getallen

Het kiezen van de beste representatie voor de opslag van niet-gehele getallen is niet zo gemakkelijk als bij gehele getallen. Er zijn immers oneindig veel niet-gehele getallen in elk interval, terwijl slechts een eindig aantal getallen uit een bepaald interval kan worden gecodeerd. Daar komt nog bij dat het interval behoorlijk groot moet worden gekozen, omdat de toepassingen die van de computer gebruik moeten kunnen maken met behoorlijk grote niet-gehele getallen moeten kunen werken. Getallen die niet een eindig aantal cijfers achter de komma hebben (zoals e en de vierkantswortel uit 2) kunnen sowieso nooit exact worden gecodeerd. Het is dus mogelijk dat het resultaat van een berekening (b.v. het berekenen van boven-genoemde vierkantswortel) niet exact kan worden gecodeerd. Daarnaast zijn er meer getallen met een niet-eindige representatie in de binaire notatie dan in de decimale notatie. Zo is 1/10 niet-eindig in binaire notatie, maar wel eindig (als 0,1) in de decimale notatie!
De binaire code die dan wordt opgeslagen is dan de representatie van het dichtsbijzijnde exact voor te stellen niet-gehele getal. Er heeft dan **afronding** plaatsgevonden. De juiste keuze voor de te gebruiken representatie moet er voor zorgen dat (gemiddeld) een zo klein mogelijke afrondingsfout wordt gemaakt.
In de loop der jaren is door verschillende producenten in hun computers van verschillende representaties voor de opslag gebruik gemaakt. T.b.v. de overdracht van data alsook processen tussen verschillende computers mogelijk te maken was standaardisatie noodzakelijk. Twee standaards ontstonden: de **IEEE floating-point** standaard en de **USASI Fortran** standaard, die beide zowel beschrijven welke wijze van opslag moet worden gebruikt, als ook hoe berekeningen ermee moeten worden uitgevoerd. Het voordeel van standaards is o.a. dat computers met speciale apparatuur kunnen worden uitgerust die de berekeningen kunnen uitvoeren. Een dergelijke zogenaamde mathematische co-processor is in de moderne PC-processoren geïntegreerd maar moest nog niet zo lang geleden apart erbij worden gekocht.

Deze standaards definiëren opslag van reële getallen in 4, 8 en 10 bytes (in 32, 64 resp. 80 bits). Bij opslag in 4 bytes wordt van ***single precision*** gesproken; bij 8 bytes van ***double precision*** en bij opslag in 10 bytes van ***extended precision***.

De standaards hebben gemeen dat elk getal als **drijvende-komma-getal (floating point)** wordt opgeslagen. Een getal staat genoteerd als drijvende-komma-getal als het staat genoteerd als het produkt van teken, mantisse, en de macht van het een of andere grondtal. Een mogelijke drijvende-komma-representatie van -300 is $-1.3.10^2$. Het teken is hier dus -1, de mantisse is 3, het grondtal 10 en de macht (ook wel exponent genoemd) 2. In computers wordt meestal een ander grondtal gebruikt, meestal 2 maar in sommige 16. Het grondtal wordt echter niet opgeslagen. Dit is

immers voor die computer altijd hetzelfde. Dus alleen het teken, de mantisse en de exponent worden opgeslagen. Voor de opslag van het teken is altijd maar 1 bit nodig, omdat er slechts twee waarden zijn. De macht wordt altijd als geheel getal opgeslagen meestal in de excess-notatie wat het vergelijken van de grootte van de exponenten vergemakkelijkt. Het aantal hiervoor gebruikte bits kan variëren en is afhankelijk van het grootste getal dat men nog wil kunnen representeren: hoe meer bits hiervoor worden gebruikt, hoe groter het grootste nog te representeren getal. Echter: hierdoor nemen wel (bij gelijke k en grondtal) de afrondingsfouten toe. In de IEEE floating-point single precision standaard wordt voor de opslag van de geheeltallige exponent 8 bits gebruikt. Voor de opslag van de mantisse zijn dan 32-1-8=23 bits over.
Het grootste getal dat zo kan worden voorgesteld is $3{,}8.10^{38}$, het kleinste getal (in absolute waarde) $1{,}5.10^{-45}$. De mantisse wordt daarbij **genormaliseerd** opgeslagen. In een genormaliseerde drijvende-komma-representatie zal de mantisse altijd liggen in een bepaald interval b.v. tussen 0,1 en 1 (bij het grondtal 10). Dus 16,7 wordt genoteerd als $0{,}167.10^{2}$ en niet als 167.10^{-1} of $1{,}67.10^{1}$. Het voordeel hiervan is bij het binaire equivalent dat het eerste bit in de mantisse altijd gelijk aan 1 zal zijn en daarom niet hoeft te worden opgeslagen waardoor met een hogere nauwkeurigheid kan worden opgeslagen.
Het is niet gemakkelijk direct in te zien, wat een bepaalde opslag nou precies betekent voor de getallen die wel en niet exact kunnen worden opgeslagen. Vandaar dat gewoonlijk wordt gezegd hoeveel significante cijfers van een getal met de representatie nog exact kunnen worden opgeslagen. Bij de single precision zijn dit er 6 à 7, bij de double precision 15 à 16 en bij de extended precision 19 à 20. Het aantal significante cijfers is gelijk aan het aantal cijfers achter de komma wanneer van een genormaliseerde weergave gebruik wordt gemaakt.
Getallen tussen 0 en 1 kunnen het nauwkeurigst worden gerepresenteerd. (Waarom?)

Overflow en underflow

Wanneer een berekening resulteert in een in absolute zin te groot getal - de exponent wordt dat de groot en kan niet worden opgeslagen - dan wordt weer van ***overflow*** gesproken. Resulteert een berekening in een - het teken negerende - te klein getal - de exponent wordt dan te klein - dan wordt van ***underflow*** gesproken. Overflow is erger dan underflow. (Waarom?)

Invoer en uitvoer van reële getallen bij een computer

Ook hier moet de tekstrepresentatie omgezet worden in de interne getalrepresentatie (bij invoer) en omgekeerd bij uitvoer. En ook hier bieden vertaalprogramma's faciliteiten gebaseerd op het door de gebruiker gekozen type. De omzetting is hier natuurlijk wel iets moeilijker, omdat tekstrepresentaties meer tekens kunnen bevatten, b.v. een decimale punt of een macht (als het getal in de wetenschappelijke notatie wordt ingevoerd). Bovendien moeten de mantisse en exponent apart worden berekend. Als deze onderdelen niet precies tussen de grenzen van de bytes in de getalrepresentatie worden gezet wordt het construeren van de getalrepresentatie moeilijker. (Miscchien ook daarom dat de exponent bij de single precision getalnotatie precies 8 bits gebruikt!) Bij het weergeven kan de gebruiker ook weer een format gebruiken om aan te geven hoe het getal in tekst moet worden uitgedrukt.

3.3 Representatie van afbeeldingen

Hedentendage wordt bij computers ook steeds meer met afbeeldingen gewerkt. Voor het opslaan van de afbeeldingen wordt van vele verschillende representaties, formaten, gebruik gemaakt. Er kan onderscheid worden gemaakt tussen het opslaan van de afbeelding zelf of het opslaan van de opdrachten die voor het construeren van de afbeelding kunnen worden gebruikt. Bij de opslag van opdrachten wordt van een **vectorafbeelding** gesproken.
Een belangrijk voordeel van een vectorafbeelding is dat voor de opslag ervan veel minder ruimte nodig is dan voor de opslag van de direct weer te geven afbeelding zelf! De term vectorafbeelding stamt uit de tijd, dat met de computer alleen tekeningen gemaakt konden worden die uit lijnen (vectoren) bestonden (zelfs de vlakken). Er was dan een speciaal apparaat nodig om de tekening te bekijken (op de beeldschermen kon alleen tekst worden weergegeven): een plot previewer, die in staat was de lijnen exact weer te geven. Dan kon besloten worden om de afbeelding op een plotter af te drukken.
Vectorafbeeldingen bevatten allang niet meer alleen de definitie van de lijnen, maar de definitie van alle primitieven (veelvlakken, cirkels e.d.) die voor het maken van de afbeelding moet worden gebruikt.
Vectorafbeeldingen kunnen alleen op computers worden gemaakt. Vele tekenpakketten hebben hun eigen taal ontwikkeld voor de beschrijving van de vectorafbeeldingen. Om toch met de door andere tekenpakketten ontwikkelde vectorafbeeldingen te kunnen werken werden conversieprogramma's geschreven, die de afbeeldingen om konden zetten. Overigens: plotters gebruikten ook weer hun eigen taal, zodat ze alleen vectorafbeeldingen konden weergeven die uitgedrukt werden in hun taal b.v. HPGL.
Een ander voordeel van vectorafbeeldingen is, dat ze hardware-onafhankelijk zijn, d.w.z. ze kunnen tussen computers met verschillende randapparatuur worden uitgewisseld en daarop worden weerge-geven, natuurlijk mits beide computers beschikken over mogelijkheden om dit formaat te kunnen lezen en de afbeelding voor het desbetreffende randapparaat te construeren. Dit maakt vectorafbeeldingen wel weer software-afhankelijk!
Zowel de pagina van de hedendaagse printer als het oppervlak van de hedendaagse beeldschermen zijn opgebouwd uit een regelmatig raster van punten. Een dergelijk beeldscherm wordt een **raster display** genoemd. Elk beeldpunt op een raster display wordt een ***pixel*** (een samentrekking van picture en element) genoemd. Het weergeven van een afbeelding op een raster display komt er op neer dat moet worden bepaald welke eigenschappen (meestal kleur) elk pixel moet krijgen. Afbeeldingen die direct, d.i. zonder conversie, op een raster display (of printer) kunnen worden weergegeven worden raster graphics genoemd. Zo'n afbeelding wordt ook wel een **bitmap** genoemd. Het nadeel van bitmaps is, dat ze machine-afhankelijk zijn, omdat ze alleen op een bepaalde groep rasterapparaten kunnen worden weergegeven, maar software-onafhankelijk omdat ze niet meer hoeven te worden geconverteerd.
Het belangrijkste nadeel is, dat ze veel geheugenruimte in beslag nemen. De hoeveelheid is afhankelijk van de grootte van de afbeelding, d.i. het aantal punten in de afbeelding, en de hoeveelheid informatie die per punt moet worden opgeslagen. Natuurlijk hoeft niet meer informatie te worden opgeslagen dan nodig is om de afbeelding weer te geven. Voor het gemak gaan we er even van uit dat per punt alleen kleurinformatie nodig is. Het aantal verschillende kleuren dat het apparaat kan weergeven bepaalt hoeveel bits voor de opslag van de kleur nodig. Als het apparaat slechts één kleur (b.v. zwart bij een zwart/wit-printer op wit papier) kan weergeven is slechts een bit nodig (wel/geen kleur) per punt; voor de opslag van een dergelijke afbeelding die een VGA-scherm geheel zal vullen zijn dus (minimaal) 640x480x1=307.200 bits, oftewel 38.400 bytes nodig. Wanneer 256 verschillende

kleuren gelijktijdig kunnen worden weergegeven, zijn 8 bits per punt nodig, waardoor al 8 keer zo veel bytes, dus 307.200, nodig zijn. In de hoogste standaard van dit moment, **true color**, wordt drie bytes kleurinformatie per punt bewaard. Op een rasterapparaat kunnen dan in theorie 256x256x256 = 16.777.216 verschillende kleuren worden weergegeven. (Natuurlijk: een 640x480-afbeelding kan natuurlijk nooit meer dan 307.200 verschillende kleuren bevatten).

Om het geheel nog wat ingewikkelder te maken zullen sommige computersystemen (Windows PC's) slechts 256 kleuren tegelijkertijd weergeven uit een palet van alle mogelijke 16,7 miljoen kleuren; de 256 kleuren die op zeker moment weergegeven worden vormen het zogenaamde systeempalet. Waarom zou een computer slechts 256 kleuren tegelijkertijd weergeven als het er 16,7 miljoen kan weergeven? Nou, dit heeft te maken met de capaciteit van het videogeheugen! Wanneer slechts 256 verschillende kleuren tegelijkertijd zouden hoeven te worden weergegeven, is maar 1 byte nodig voor de opslag van het plaatsnummer van de kleur in het systeempalet i.p.v. 3 voor de opslag van de kleurinformatie! Het systeempalet fungeert dan als tabel waarin de werkelijk te gebruiken kleur kan worden opgezocht. In dat geval hoeft slechts 1/3 van de anders benodigde hoeveelheid informatie in het videogeheugen worden opgeslagen. Deze manier waarop de afbeelding in het videogeheugen wordt opgeslagen kan natuurlijk ook worden toegepast bij (het opslaan van) de afbeelding zelf: de afbeelding kan bestaan uit een tabel waarin de in de afbeelding gebruikte kleuren staan opgeslagen, aangevuld met voor elk punt het plaatsnummer in de tabel van de kleurinformatie, waardoor dus ook maar iets meer dan 1/3 van de hoeveelheid opslag-ruimte nodig is. Het is dus het gebrek aan voldoende geheugenruimte (en impliciet komt dit overeen met de hoeveelheid geld die men bereid is te betalen) die ervoor gezorgd heeft dat dergelijke methoden zijn ontwikkeld. De tabel waarin de feitelijk te gebruiken kleurcodes staan opgeslagen wordt een logisch palet genoemd. Elke afbeelding die op een beeldscherm wordt weergegeven, moet het besturings-systeem vragen het logische palet te realiseren d.w.z. de kleuren die moeten worden weergegeven toe te voegen aan het systeempalet zodat de afbeelding natuurgetrouw kan worden weergegeven. Als niet alle kleuren in het logische palet kunnen worden gerealiseerd (b.v. als meerdere afbeeldingen met samen meer dan 256 kleuren in hun logische paletten), dan zal het besturingssysteem naar de dichtstbijzijnde kleur zoeken! Het besturingssysteem construeert een tabel waarin voor elk element in het logische palet staat welk element uit het systeempalet daarbij hoort. Deze wordt gebruikt om de afbeelding weer te geven. In het videogeheugen komen dan uiteindelijk de volgnummers te staan zoals het besturings-systeem die uit de geconstrueerde tabel haalt.
In de loop der jaren zijn vele verschillende bitmapbestandsformaten ontstaan, alle met hun eigen eigenschappen en bestaansreden ([4]). De belangrijkste zijn:

• TIFF (Tagged Image File Format)	Wordt ondersteund door veel software-pakketten. De meeste recente versie staan compressie toe: handig voor transport.
• BMP (Bitmap)	Het is een ongecomprimeerd formaat en wordt daarom zelden gebruikt voor grote afbeeldingen. Wordt door Windows ondersteund (alsook de DIB- en WMF-formaten).
• DIB (Device Independent Bitmap)	Afbeeldingen in dit formaat kunnen op een reeks van apparaten worden weergegeven.
• GIF (Graphics Interchange Format)	Gecomprimeerd formaat ontwikkeld voor gebruik op CompuServe.
• EPS (Encapsulated PostScript File)	Komt uit de desktop publishing wereld. Slaat een afbeelding in PostScript-formaat op. Kan wel in andere formaten worden omgezet, maar niet (of moeilijk) omgekeerd. Bedoeld om naar een PostScript-printer te sturen.

• WMF (Windows Metafile Format)	Het is een vectorformaat, ondersteunt door Windows. Metafile betekent dat het zowel raster als vector graphics kan bevatten.
• HPGL (Hewlett Packard Graphics Language)	Wordt gebruikt voor uitvoer naar plotters. Raakt in onbruik, behalve bij sommige CAD-programma's.
• JPEG (Joint Photographic Experts Group)	Een recent formaat ontwikkeld voor maximale compressie.

Zoals al eerder opgemerkt worden vectorafbeeldingen meestal met de computer gemaakt. Daarnaast bestaan er natuurlijk nog veel afbeeldingen die niet met de computer (kunnen) worden gemaakt, die analoog worden vastgelegd op magnetische banden (video en film), op papier (zoals foto's). Om deze afbeeldingen de computer in te krijgen moeten ze worden gedigitaliseerd. Dit kan o.a. m.b.v. scanners, die steeds nauwkeuriger worden. Het resultaat is altijd een rasterafbeelding. Er is een trend gaande waarbij digitalisering steeds eerder in het registratieproces plaatsvindt: denk maar aan digitale foto- en videocamera's, waarbij het opgenomene direct in digitale vorm in een geheugen wordt opgeslagen.
Bij het digitaliseren van analoge informatie treden altijd afbeeldingsfouten op, omdat analoge informatie nu eenmaal niet discreet is, en alle waarden in een bepaalde bereik kunnen worden aangenomen!
Het voordeel van uitdrukking in digitale vorm is echter wel dat copiëring van digitale informatie wel altijd zonder kwaliteitsverlies kan plaatsvinden, terwijl analoge vermenigvuldiging altijd aan kwaliteitsverlies onderhevig is.
Een belangrijk probleem bij het digitaliseren van beelden is de grote hoeveelheid gegevens die daarbij ontstaan. Voor een minuut ongecomprimeerde full-screen digitale video is al 1,5 Gbyte geheugen nodig, wat zelfs niet eens op 2 hedendaagse CD-ROM's past! Niet alleen de opslag is overigens een probleem, ook het real-time ophalen en weergeven van zo'n grote hoeveelheid gegevens (met een snelheid van 25 Mbytes/s) vormt een probleem, vooral bij transport over een netwerk. Er is en wordt derhalve gezocht naar methoden om liefst met een minimum aan kwaliteitsverlies de digitale gegevens te comprimeren voor opslag en transport, en pas op het laatste moment bij weergave weer te decom-primeren. Er bestaan vele (de)compressieschema's die gebruik maken van hardware, software of beide. (Alles waarvan de standaardisatie uitgekristalliseerd is wordt t.z.t. in hardware uitgevoerd: hardware is sneller dan software.)
Videogegevens kunnen vóór (real-time) of na opslag worden gecomprimeerd. Het laatste houdt in dat voldoende snel geheugen aanwezig moet zijn. **Lossy videocompressie** - waarbij na het decomprimeren informatie is verdwenen - kan worden toegepast, met de grootst mogelijke reductie in het aantal gegevens, omdat het menselijk oog minder gevoelig is voor kleur- en schaduwvariaties dan voor de helderheid van het beeld. Compressieschema's maken derhalve gebruik van mathematische algoritmen die alle details verwijderen die normaal niet door het menselijk oog wordt verwerkt. Speciale compressieschema's die gebruikmaken van **fractals** (een afkorting voor ***fractional dimensional***) kunnen een compressieratio van 10.000: 1 bereiken, maar nemen te veel tijd in beslag om van praktisch nut te zijn bij real-time videoconferenties. Er zijn fractal-compressie-pakketten die een ratio van 2500:1 halen.
Het elimineren van overbodige informatie in opeenvolgende beeldjes wordt **interframe compressie** genoemd; van overbodige informatie in de beeldjes apart **intraframe compressie**. Beide methoden worden toegepast in compressieschema's als JPEG, MPEG (*Motion Pictures Experts Group*) en *digital video interactive* (DVI).
Een schema voor compressie en decompressie wordt een **codec** genoemd.

De basis voor vele compressietechnieken is het ***discrete cosine transform*** **(DCT)** algoritme. Het is een complexe mathematische formule die compressie bereikt door het elimineren van overtollige data in blokken beeldpunten. DCT is o.a. de basis voor JPEG, MPEG en H.261. Hiermee kunnen theoretisch compressieratio's tot 800:1 worden bereikt, alhoewel de resultaten bij een ratio van 200:1 sterk slechter worden.
Vectorquantisatie is een techniek die ten grondslag ligt aan vele **software-codecs**, zoals DVI (van Intel en IBM), QuickTime (Apple), Video for Windows (Microsoft), Indeo (Intel), Ultimotion (IBM), Cinepak, and MotiVE.
AT & T Bell Laboratories ontwikkelde de ***Wavelet***-techniek, gelijkend op DCT, waarbij blokken van variabele grootte worden gebruikt. Andere technieken worden weinig gebruikt.
Deze codecs zijn beslist niet gelijkwaardig en het kiezen van de meest geschikte is nog een hele kunst.
Standaards op het gebied van compressie van beelden worden met name ontwikkeld door JPEG, MPEG, ISO (International Standards Organization) en CCITT (International Telegraph and Telephone Consultative Committee). De JPEG- en MPEG-standaards zijn naar de groepen genoemd die zich met de ontwikkeling ervan bezig hebben gehouden**. Standaard-JPEG** is ontwikkeld voor compressie van individuele beelden, **motion-JPEG** voor compressie van video, **MPEG-1** voor compressie van digitale video in interactieve multimedia applicaties**, MPEG-2** voor toepassing in TV- en video-on-demand-uitzendingen. De **H.261** standaard is met name ontwikkeld voor gebruik bij videoconferenties.

3.4 Representatie van geluid

Ook bij het digitaliseren van geluid treden afbeeldingsfouten op, daar het geluidssignaal slechts een eindig aantal keren per seconde kan worden gemeten, en de meetwaarden digitaal en dus niet (altijd) exact worden opgeslagen. Met het toenemen van het aantal keer dat per seconde wordt gemeten (de ***sampling frequency***) en het aantal bits waarin de meetwaarde wordt opgeslagen kan de opnamekwaliteit worden vergroot. Bij Audio CD-kwaliteit wordt het geluidssignaal 44.100 keer per seconde gesampled en worden 16 bits (2 bytes) voor de opslag van de meetwaarde gebruikt. Voor één seconde stereo geluid is dan 44.100x2x2=176.400 bytes nodig; op een **CD-audio** kan ongeveer 74 minuten in deze kwaliteit staan.
Spraak kan goed worden gedigitaliseerd in 8 bits met een samplingfrequentie van 8000 keer per seconde.
Bij muziek kan naast het op bovenstaande wijze opslaan van het geproduceerde geluid natuurlijk ook gebruik worden gemaakt van het opslaan van de noten i.p.v. de tonen. Hiervoor is wel één standaard ontstaan nl. **MIDI (*Musical Instruments Digital Interface*)**, ofschoon er een aantal varianten van bestaan. Net als bij beelden is de omzetting van MIDI naar geluid wel altijd mogelijk, andersom niet.
Een geluidcompressie-standaard als **G.728** kan maximaal een compressieratio van 4:1 halen [3].

Literatuur

[1] J. Faber (red.), Elementaire informatica: de middelen, Pandata BV, 1988, Rijswijk.
[2] P.P. Silvester en D.A. Lowther, Computer Engineering: circuits, programs, and data, Oxford University Press Inc., 1989, New York.
[3] B.O. Szuprowicz, Multimedia Networking, McGraw-Hill, 1995, New York.
[4] R. Wodawski, Multimedia Madness (second edition), Sams Publishing, 1994, Indianapolis.

4. PROGRAMMATUURONTWIKKELING

Het ontwikkelen en onderhouden van commerciële software met een groot team stelt hoge eisen aan de kwaliteit van het ontwikkelproces en derhalve aan de organisatie ervan. Ten behoeve van die organisatie wordt in de praktijk een aantal fasen onderscheiden – tenminste analyse, ontwerp en realisatie – waarbij door het management vooraf wordt bepaald wanneer elke fase moet zijn afgerond en wat er dan bereikt moet zijn b.v. m.b.t. de gemaakte documentatie.
Het valt buiten het bestek van Informatica 1 om hier diepgaand op in te gaan. Enige informatie hierover is in Living with Computers te vinden. Wel is van belang te weten dat er vele methoden en technieken bestaan, al dan niet ondersteunt door geavanceerde software-tools, waarmee het ontwikkelproces wordt geformaliseerd.
Aan het ontwikkelen van een klein computerprogramma door slechts een paar personen voor niet-commerciëel gebruik b.v. in gebruiksonderzoek hoeven niet zulke extreem hoge eisen gesteld te worden. Gelukkig maar, want we hebben net genoeg tijd om jullie de basis bij te brengen en meer niet. In dit hoofdstuk concentreren we ons derhalve met name op de manier waarop een eenvoudig ontwerp – bestaande uit een niet te groot aantal algoritmen – kan worden gemaakt als basis voor wat je tijdens het practicum gaat doen.

4.1 De ontwikkeling van computerprogramma's

Het computerprogramma bevat aanwijzingen – in de vorm van opdrachten – aan de computer hoe een bepaalde taak moet worden uitgevoerd, aangezien de computer slechts tot het direct uitvoeren van een beperkt aantal – zeer eenvoudige – taken in staat is! Het laten uitvoeren van deze opdrachten heeft als gevolg dat de taak wordt uitgevoerd.

De taal van computers
Dit betekent dat het computerprogramma opdrachten moet bevatten die de computer kan uitvoeren. Elke opdracht moet daartoe in de taal die de computer kan begrijpen worden uitgedrukt: de **machinetaal**.

> De instructies die een bepaalde computer begrijpt kunnen beschouwd worden als zinnen in de taal die de computer 'spreekt': de machinetaal. Net als zinnen in de spreektaal moeten instructies in de machinetaal zich aan een aantal regels houden om begrepen te kunnen worden. Machinetaal maakt net als elke andere taal gebruik van een alfabet om zinnen in uit te drukken. Zo'n alfabet bestaat uit een eindig aantal tekens; bij de machinetaal zijn dit er altijd 2, die meestal met 0 en 1 worden aangeduid! Een verzameling regels die aangeeft hoe zinnen in een taal moeten worden opgebouwd wordt de **syntaxis** - of grammatica - van die taal genoemd. De betekenissen van de instructies d.w.z. wat het effect ervan is is vastgelegd in de semantische regels. Ook al gebruiken alle digitale computers hetzelfde digitale alfabet, afgedwongen door de twee mogelijke toestanden waarin het kleinste geheugenelement zich kunnen bevinden, ze hebben niet per se ook dezelfde syntaxis. Niet alle computers spreken dezelfde taal; d.w.z. we kunnen computers rangschikken naar de machinetaal die ze gebruiken. De verzameling van mogelijk te geven opdrachten – de **instructieset** - kan verschillen, maar ook de wijze waarop gegevens moeten worden gerepresenteerd, en adressen moeten worden

aangeduid! Hoe complexer de opdrachten kunnen zijn, hoe meer schakellogica een computer nodig heeft om elke instructie te begrijpen en hoe duurder de computer zal worden. Vandaar dat de hedendaagse digitale computer slechts in staat is een klein aantal zeer eenvoudige taken - en dus opdrachten - uit te voeren.
Er is zelfs een tendens naar gebruik van een zo klein mogelijk aantal machinetaalinstructies: deze vindt je terug in RISC-processoren. **RISC** is een afkorting ***voor Reduced Instruction Set Computer***. Instructies kunnen sneller worden begrepen en dus sneller worden uitgevoerd. Voor het uitvoeren van dezelfde taak zijn echter wel meer van dergelijke instructies nodig. De meeste PC's hebben echter nog een CISC-architectuur. **CISC** is een afkorting van ***Complex Instruction Set Computer***.)

Elke opdracht uitgedrukt in machinetaal wordt als direct uitvoerbaar beschouwd, omdat de computer die zonder verdere hulpmiddelen kan uitvoeren. Een computer-programma bestaat in principe dus uit machinetaalinstructies die door de computer kunnen worden uitgevoerd.
Nu is het om twee redenen minder handig om direct in machinetaal te programmeren. Ten eerste moet voor elk ander type computer een ander programma worden geschreven en dat is tijdrovend en ten tweede sluit machinetaal niet aan bij de menselijke grammatica.
Daarvoor zijn programma's ontwikkeld die de gebruiker in staat stellen zijn opdrachten aan de computer in een prettiger vorm uit te drukken dan in de machinetaal. In eerste instantie ontstonden er **assemblertalen** waarbij elke opdracht daarin uitgedrukt ook precies overeenkomt met één opdracht in de machinetaal, maar waarbij de binaire codes als gebruikt in de machinetaal zoveel mogelijk door tekst, zgn. mnemonische codes, vervangen waren. Zo hoefde men b.v. niet meer de code voor de operatie optellen te onthouden (b.v. 10010110) maar kon men eenvoudig volstaan met ADD.
Een programma dat opdrachten uitgedrukt in de ene taal omzet in opdrachten in een andere taal (niet per se een machinetaal) wordt een **vertaalprogramma** of **vertaler** genoemd.
Een vertaler die assembleertaalopdrachten omzet in machinetaalopdrachten wordt een **assembler** genoemd. Ook al was deze makkelijker te leren, er waren nog steeds relatief veel opdrachten nodig om een eenvoudige taak b.v. het optellen van twee getallen uit te voeren, omdat men zich ook met details (zoals het verplaatsen van de getallen naar en uit het geheugen) moest bezighouden. Bovendien was de asssembleertaal nog steeds afhankelijk van de machinetaal en was het dus niet zomaar mogelijk een programma geschreven in assembleertaal te gebruiken op een computer met een andere machinetaal. Er was dus sprake van een beperkte **overdraagbaarheid.**
Om dit nu verhelpen zijn **hogere programmeertalen** ontwikkeld zoals BASIC, FORTRAN, COBOL, C, Pascal en uiteindelijk ook Visual Basic. Het was nu mogelijk om voor de computer in een soort Engels opdrachten te maken. Deze opdrachten als zodanig werden door de computer niet begrepen. Ook hier zijn natuurlijk vertaalprogramma's nodig om een programma uitgedrukt in een hogere programmeertaal om te zetten in een machinetaalprogramma.
Machine- en assembleertalen vormen de **lagere programmeertalen**, de overige de hogere programmeertalen. Lagere programmeertalen horen altijd bij een bepaalde soort computer en zijn altijd voor algemeen gebruik d.w.z. ze kunnen gebruikt worden voor het maken van willekeurig welk programma dan ook. Hogere programmeertalen hoeven niet voor algemeen gebruik te zijn. De verschillende talen worden gebruikt voor verschillende toepassingen zoals zakelijke, didactische en wetenschappelijke toepassingen. Programmeertalen voor algemeen gebruik zijn nl. niet per se geschikt voor alle soorten toepassingen. Er zijn verschillende manieren om een taal krachtiger te maken. Het makkelijkste is het om specifieke functies toe te voegen; deze werden

dan naar toepassingsgebied gebundeld en te koop aangeboden door derden. Het bekendst zijn NAG en IMSL voor functies m.b.t. numerieke en statistische toepassingen. Tegenwoordig staan klasse-bibliotheken als de Microsoft Foundation Classes (MFC) wat meer in het middelpunt van de belangstelling.

Een onervaren computergebruiker wil op een eenvoudige manier zijn wensen kenbaar maken. Dit kan gemakkelijker via menu's en invulschermen dan met tekstopdrachten. Vandaar dat in eerste instantie allerlei applicaties alleen met menu's en invulschermen werkten. Menu's en invulschermen kunnen ervaren computergebruikers echter ergeren vooral wanneer die weten dat een opdracht via intypen veel sneller gegeven kan worden. Bovendien moet een gebruiker bij het gebruik van menu's en invulschermen steeds aanwezig zijn, wat niet altijd gewenst is, zeker als de gebruiker lang moet wachten op het resultaat. Opdrachten - uitgedrukt in tekst - kunnen vaak wel ingevoerd worden uit een door de gebruiker op te geven bestand en deze hoeft tijdens de verwerking van de opdrachten niet per se aanwezig te zijn! Door de voordelen die tekstopdrachten hebben bevatten pakketten veelal mogelijkheden tot het geven van tekstopdrachten. Dit heeft bovendien als voordeel, dat de hoeveelheid menu's en invulschermen beperkt kan blijven tot de meest gebruikte functies. Extra functionaliteit kan dan via tekstopdrachten worden gerealiseerd. De taal waarin de tekstopdrachten gesteld moeten worden hoort meestal bij het pakket en wordt een **vierde-generatietaal** genoemd, in tegenstelling tot de derde-generatietalen voor algemeen gebruik.

Programmeermodellen

In de loop der jaren zijn een viertal verschillende programmeermodellen, d.i. manieren van programmeren, ontstaan, in chronologische volgorde het procedurele (of imperatieve), functionele (of applicatieve), logische en object-georiënteerde model. Programmeertalen zijn gewoonlijk gebaseerd op een van deze modellen (of **paradigma 's**), alhoewel sporen van andere programmeerstijlen erin kunnen voorkomen (b.v. C++). De verschillende paradigma's hebben ook geleid tot de opkomst van bijbehorende ontwerp- en analysemethoden, waarvan de object-georiënteerde methoden het laatste decennium sterk in opkomst zijn, en ook steeds meer ondersteund worden met geschikte software-tools.

Ook al zijn hogere programmeertalen gemakkelijker te leren en te gebruiken dan lagere programmeertalen, dan nog is het niet gemakkelijk om voor een bepaalde taak direct een computerprogramma te schrijven in zo'n hogere programmeertaal. We denken immers in natuurlijke taal en niet in een bepaalde programmeertaal (althans de meeste van ons!) Het is daarom dan ook gebruikelijk eerst een beschrijving van de uit te voeren acties in een natuurlijke taal uit te drukken. Zo'n beschrijving wordt een **algoritme** genoemd. Elk programmeermodel geeft aanleiding tot een andere invulling van het begrip algoritme.

> Het woord algoritme is afkomstig van de naam van de schrijver van een boek over de rekenregels die wij op de lagere school hebben geleerd voor b.v. optellen en vermenigvuldigen. Die Perzische schrijver heette Abu Jafar Mohammed ibn Musa al-Khowarizmi; in de Latijnse vertalingen werd de auteur Algorism genoemd. In deze eeuw ontstaat het begrip algoritme met een meer algemene betekenis uit Algorism en een "t" geleend uit aritmetiek.

Een **procedureel algoritme** is een beschrijving van de wijze waarop een taak moet worden verricht waarbij de volgorde van uitvoeren met besturingsopdrachten als opeenvolging, selectie en herhaling wordt aangegeven. In wat volgt en in het VB-practicum wordt met algoritme steeds een procedureel algoritme bedoeld.

Een dergelijk algoritme is vergelijkbaar met een kookrecept. Wat in een kookrecept de ingrediënten zijn die worden gebruikt, zijn in een algoritme de gegevens die worden verwerkt!
Voordat je aan het maken van de eindopdracht kunt beginnen zul je eerst algoritmen moeten bedenken voor de acties die door het programma moeten worden uitgevoerd, en die ter beoordeling inleveren. Pas als deze zijn goedgekeurd mag je aan het programmeren in Visual Basic slaan. Hiermee voorkomen we dat je te hard van stapel loopt en bereiken we dat je goed nagedacht hebt over wat er van het programma verwacht wordt!

Algoritmen en computerprogramma's

Een algoritme is nog geen computerprogramma omdat er nog geen vertaal-programma's zijn die gewone taal in machinetaal kunnen omzetten.
Programmeertalen worden zodanig ontworpen dat het relatief gemakkelijk is een algoritme erin uit te drukken. Vaak wordt een algoritme eerst nog uitgedrukt in een tussenvorm, hetzij in **pseudocode**, hetzij schematisch m.b.v. b.v. **stroomschema's**. Het is echter ook mogelijk, een werkwijze die we hier zullen hanteren, een algoritme volledig uit te werken, zelfs tot in de taal waarin het programma wordt uitgedrukt. Het spreekt voor zich, dat wanneer de gebruikte pseudocode of schematische voorstelling voldoende geformaliseerd zouden zijn, er ook een vertaalprogramma voor zou kunnen worden geschreven.
Het uitdrukken van de beschrijving van de uit te voeren taak in de vorm van een algoritme behoort tot de ontwerpfase van de taak een computerprogramma te ontwikkelen. Het algoritme wordt opgesteld a.d.h.v. de specificatie d.w.z. de eisen waaraan het te ontwikkelen programma moet voldoen.
Teneinde de specificatie duidelijk, ondubbelzinnig en consistent te laten zijn is vaak een gedegen analyse van de vaak vaag en in algemene bewoordingen uitgedrukte taakomschrijving nodig. (Een taakomschrijving als 'tekstverwerken' b.v. is voldoende vaag om een analyse nodig te maken.)
Het ontwikkelen van een computerprogramma komt neer op het beschrijven van de deeltaken waaruit een uit te voeren taak bestaat, uiteindelijk op zodanige wijze dat de deeltaken wel door de computer uitgevoerd kunnen worden.
Dit betekent dat we moeten weten welke taken de computer wel kan uitvoeren en welke niet! Want, zolang een deeltaak nog niet door de computer kan worden uitgevoerd, moet daarvoor een (deel)algoritme worden geschreven! Het is dus mogelijk het algoritme te ontwikkelen door de taak op een logische manier in (een klein aantal) deeltaken op te splitsen, vervolgens deze deeltaken weer op te splitsen net zolang totdat - bij wijze van spreken - de bladeren van de boom zijn bereikt en de deeltaak niet verder hoeft te worden beschreven, omdat deze nu eenmaal al uitvoerbaar is.

De correctheid van computerprogramma's

Aan computerprogramma's kunnen diverse eisen worden gesteld, zoals eisen t.a.v. de correctheid, betrouwbaarheid, veiligheid, gebruiksgemak (comfort) en efficiëntie.
Van een computerprogramma mag tenminste worden verwacht dat het doet wat het behoort te doen, volgens de specificatie, d.w.z. zonder onnodige fouten te maken.
In het ideale geval kan uit een specificatie direct een foutloos computerprogramma worden gemaakt dat aan de specificatie voldoet. Dat is echter vaker niet dan wel het geval. Dit komt omdat het heel moeilijk is te bewijzen dat een computerprogramma correct werkt onder alle omstandigheden. Vandaar dat in de praktijk wordt volstaan met het testen van het programma. **Testen** betekent o.a. het programma uitvoeren voor alle mogelijke toegestane invoerverzamelingen en nagaan of het programma voor elk van die invoerverzamelingen de gewenste uitvoer produceert. Als de uitvoer niet correct is moet worden nagegaan waar de fout zit. Het zoeken van de fout wordt

debuggen genoemd. Door de verandering in toestand van het programma te bestuderen b.v. bij een stap voor stap uitvoeren van de instructies kan een fout worden opgespoord. Dit laatste kan van een ontwerp natuurlijk niet met een computer!

Het ontwerpen van computerprogramma's

Alhoewel een bronprogramma – met de hand – veel gemakkelijker is te maken dan het bijbehorende doelprogramma, is het vooral bij grote programma's niet gewenst om het bronprogramma direct – uit het hoofd – te maken. Gewoonlijk worden eerst een ontwerp gemaakt.
Er kunnen twee soorten ontwerpen worden onderscheiden: het **logische** of **functionele** en het **fysische** of **technische** ontwerp. Bij het maken van het logische ontwerp wordt nog geen gebruik gemaakt van (kennis van) (de (on)mogelijkheden van) de programmeertaal waarin het computerprogramma zal worden uitgedrukt en/of de te gebruiken ontwikkelomgeving: het logische ontwerp is nog taal- en omgevings-onafhankelijk.
Hierdoor is de ontwerper 'vrij' in het kiezen van te gebruiken gegevensstructuren en 'taal' waarin het ontwerp wordt uitgedrukt. Het begrip 'taal' moet hier ruim worden opgevat omdat er ook van schematische voorstellingen (b.v. **flow charts**) gebruik gemaakt kan worden. Het logische ontwerp kan desnoods geheel en al in natuurlijke taal worden uitgedrukt, gebruikelijker is het echter van de een of andere **pseudocode** gebruik te maken. Dit is een streng gedefiniëerde deelverzameling van natuurlijke taal die ondubbelzinnig is. Het is echter geen echte code in die zin dat ie niet geformaliseerd is en iedereen in principe zijn eigen pseudocode kan definiëren. Gewoonlijk is er toch wel enig verband met de uiteindelijk te gebruiken programmeertaal! In hoofdstuk 6 en 10 staan voorbeelden van een mogelijk te gebruiken pseudocode.
Op zeker moment moet het logisch ontwerp omgezet worden in een fysisch ontwerp waarin van de eigenschappen van de te gebruiken programmeertaal/-omgeving gebruik wordt gemaakt en wordt het ontwerp programmeertaal/-omgevingafhankelijk. Wanneer echter, als bij vele eenmansprogrammeerprojecten het geval is, de uiteindelijke taal en machine al bekend zijn, zal de grens tussen logisch en fysisch ontwerp vervagen en wordt een ontwerp wellicht al erg snel in een bepaalde programmeertaal uitgedrukt. Het gebruik van constructies uit een hogere programmeertaal kan - de meest eenvoudige ontwerpen buiten beschouwing latende - gewoonlijk echter pas plaatsvinden als het ontwerp voldoende gedetailleerd is.

Vooraf moet echter in een **specificatie**, vergelijkbaar met een program van eisen (PvE), worden vastgelegd wat er van het te maken computerprogramma wordt verwacht. Omdat een computerprogramma kan worden beschouwd als een systeem dat invoer omzet in uitvoer moet in een specificatie worden vastgelegd welke uitvoer bij welke invoer moet worden geproduceerd. Zowel de in- als de uitvoer stellen gegevens voor. Een dergelijke specificatie kan uit een statisch document bestaan, maar met name het te verwachten (interactief) gedrag van het programma is daarin moeilijk vast te leggen. Vandaar dat van (de user interface van) het programma vaak een prototype wordt gemaakt, om aan te geven welke dynamische eisen daaraan worden gesteld. Deze kan tevens worden losgelaten op gebruikers, die dan invloed kunnen uitoefenen op de specificatie. Het blijkt namelijk dat gebreken in en tekortkomingen aan de specificatie erg kostbaar kunnen zijn in termen van extra benodigd onderhoud in de vorm van aanpassingen.

Voordat ik bespreek wat het ontwerpen van een computerprogramma inhoudt, bespreek ik wat onder de toestand van een computerprogramma wordt verstaan, omdat dit begrip een essentiele rol speelt bij de ontwikkeling ervan.

Computerprogramma's en gegevens

Het computerprogramma dat door een vertaler wordt gemaakt, wordt in een bestand opgeslagen. Het besturingssysteem ontvangt op zeker moment het verzoek om het computerprogramma uit te voeren. Omdat alleen opdrachten die zich in het werkgeheugen van de computer bevinden kunnen worden opgehaald en uitgevoerd door de processor moet het besturingssysteem het computerprogramma in het werkgeheugen plaatsen en vervolgens de besturing van de computer aan het computerprogramma overdragen, wat gewoon betekent dat het de eerste opdracht in het computerprogramma uit laat voeren. Een computerprogramma dat wordt uitgevoerd gebruikt echter niet alleen werkgeheugen voor de opdrachten ervan maar heeft ook geheugenruimte nodig voor de opslag van gegevens die het nodig heeft. Het bestand bevat daarom niet alleen opdrachten maar ook informatie die het besturingssysteem gebruikt om te bepalen hoeveel geheugenruimte moet worden gereserveerd voor zowel de opslag van de opdrachten als van de gegevens die het programma zal beheren. Het geheugen dat gereserveerd wordt voor de opslag van de gegevens van het programma kan gewoonlijk worden opgedeeld in een aantal delen, die voor verschillende doeleinden worden gebruikt. Dit zijn:

- Het **datagebied**, voor de opslag van permanente, voor de gehele duur van de werking van het programma beschikbare, gegevens. Te denken valt b.v. aan gegevens over printerinstellingen. Gegevens hierin opgeslagen bevinden zich gedurende de werking van het programma steeds op dezelfde plaats in het geheugen.
- De **stapel (*stack*)**, voor de opslag van tijdelijke gegevens nodig voor de uitwisseling van informatie tussen de verschillende programma-onderdelen en voor de opslag van tussenresultaten van berekeningen. Gegevens hierin opgeslagen bevinden zich steeds op een positie in het geheugen die door de computer wordt bijgehouden – meestal in een speciaal daarvoor gebruikt register - en die kan worden opgevraagd in een programma.
- De ***heap***, voor de opslag van tijdelijke gegevens, die naar believen aangemaakt en vermietigd moeten kunnen worden. Te denken valt b.v. aan de opslag van de afzonderlijke regels van een document. Bij het verwijderen van een regel door een gebruiker kan het geheugen dat door die regel werd gebruikt worden vrijgegeven en hoeft dus niet voor de gehele duur van de werking van het programma worden gereserveerd. Gegevens hierin opgeslagen bevinden zich niet op vaste plaatsen aangezien bij het in gebruik nemen van een bepaald deel van de *heap* voor gebruik door een bepaald gegeven niet vooraf bekend is welke geheugenplaatsen beschikbaar zullen zijn. Het programma moet derhalve zelf bijhouden waar een bepaald op de heap geplaatst gegeven zich bevindt.

De afmetingen van het datagebied en de stapel liggen vooraf vast. Hoeveel geheugenruimte voor de heap beschikbaar zal zijn hangt af van de nog beschikbare hoeveelheid geheugenruimte in het werkgeheugen na het reserveren van de geheugenruimte voor de opdrachten, het datagebied en de stapel. Wanneer zelfs deze hoeveelheid geheugenruimte niet beschikbaar is, kan het programma niet worden ingelezen in het werkgeheugen en derhalve niet worden uitgevoerd. Het besturingssysteem zal je dan vertellen dat het programma te groot is om te worden uitgevoerd.

In de verschillende vormen waarin een computerprogramma kan bestaan, worden de gegevens die door het programma worden beheerd op een andere manier aangeduid. De maker van een machinetaalprogramma werkt rechtstreeks met het geheugen: hij moet de exacte plaats – een nummer - weten waar een bepaald gegeven zich in het geheugen bevindt. Bovendien moet de maker ervan weten hoeveel geheugen voor de opslag van het gegeven nodig is en hoe het gegeven moet worden opgeslagen.

De maker van een assemblerprogramma hoeft niet steeds deze plaats te onthouden, maar geeft een naam – een zgn. **mnemonische code** - aan de plaats waar zich een bepaald gegeven bevindt, zodat slechts die naam hoeft te worden onthouden. Ook hij moet weten hoeveel geheugen voor de opslag van het gegeven nodig is en hoe het gegeven moet worden opgeslagen.
De maker van een programma in een hogere programmeertaal geeft ook een naam aan een bepaald gegeven maar het vertaalprogramma bepaalt welke geheugenplaats(en) gebruikt zal/zullen worden. Een dergelijke groep geheugenplaats(en) wordt een **variabele** genoemd en de inhoud ervan wordt de waarde van de variabele genoemd. De maker vertelt het vertaalprogramma wat voor soort gegevens in de variabele opgeslagen zullen moeten worden – b.v. een teken, geheel of reëel getal – het gaat, waaruit het vertaalprogramma kan afleiden hoeveel geheugenruimte voor de opslag ervan nodig is en op welke wijze het gegeven moet worden gerepresenteerd. Het vertaalprogramma bepaalt welke variabelen in het datagebied, op de stapel of op de heap moeten worden opgeslagen.

De opslag van gegevens met een vaste grootte is geen probleem. Interessant wordt het pas als het gaat om de opslag van gegevens die niet steeds dezelfde grootte hebben. Denk b.v. aan de opslag van de tekens in een regel tekst, wat er niet altijd evenveel zijn. Hiemee kan op twee manieren worden omgegaan. Ten eerste is het mogelijk om een maximale grootte te veronderstellen en steeds dit aantal geheugenplaatsen te gebruiken. Alle gegevens die in het datagebied worden geplaatst hebben een dergelijke vaste grootte. Nadeel is de verspilling van geheugenplaatsen. Het alternatief is steeds net zo veel geheugenruimte te reserveren als voor de opslag nodig is. Steeds precies het aantal benodigde geheugenplaatsen reserveren is niet praktisch om dit b.v. zou betekenen, dat, elke keer als het aantal tekens in de tekstregel wordt gewijzigd, meer of minder geheugenruimte moet worden gereserveerd. Bij het langer worden van de regel moet extra – aaneensluitende - geheugenruimte worden gereserveerd. Die is wellicht niet beschikbaar, wat het verplaatsen van alle tekens in de regel naar een nieuwe geheugenlocatie met zich mee brengt. Het is daarom gebruikelijker om een beperkte hoeveelheid niet in gebruik zijnde geheugenruimte toe te staan door de opslagruimte steeds niet met één extra plaats te veranderen maar met een groter aantal b.v. 8. Het zal duidelijk zijn dat dergelijke gegevens niet in het datagebied maar op de *heap* zullen worden bewaard.

De toestand van een computerprogramma

Computerprogramma's verwerken gegevens tot informatie, d.w.z. invoergegevens worden gebruikt om uitvoergegevens te produceren. Soms moeten daarbij tussenresultaten – ook gegevens - worden berekend en bewaard. Het is de taak van de ontwerper van het computerprogramma te bepalen wat de invoer- en uitvoergegevens zijn en welke tussenresultaten er moeten worden berekend.
Alle op zeker moment door het programma beheerde gegevens – in alle gegevensgebieden – vormen wat de **toestand van het programma** worden genoemd. Tot de toestand kunnen echter naast de waarden als – onzichtbaar - opgeslagen in het werkgeheugen ook de zichtbare delen, met name van de user interface, worden gerekend.
Het is de taak van de ontwerper van het computerprogramma te bepalen welke waarden door het programma moeten worden bewaard omdat ze nodig zijn om de gewenste uitvoer te produceren. (Het is de kunst niet meer waarden te bewaren als strikt noodzakelijk is.) Op zeker moment zullen de te produceren uitvoerwaarden onderdeel zijn van de toestand en aan de omgeving kunnen worden verstrekt, waarmee het programma zijn (deel)taak heeft volbracht.

Daarnaast wordt in computerprogramma's ook gebruik gemaakt van waarden die niet veranderen b.v. wanneer een bepaalde waarde met 1 moet worden opgehoogd. Deze waarden worden dan niet opgeslagen in het geheugengebied waar variabelen worden opgeslagen, maar zijn onderdeel van de opdrachten.

Direct na het opstarten van een computerprogramma zijn, dus voordat het programma de eerste opdracht heeft uitgevoerd, is weliswaar geheugenruimte gereserveerd voor de opslag van de gegevens maar hebben deze geheugenplaatsen nog geen waarden gekregen: de toestand van het programma is nog ongedefiniëerd. Hoe komt het programma nu aan de inhoud daarvan? Een aantal elementaire acties moet daartoe beschikbaar zijn. Het moet mogelijk zijn om een waarde:

- In te lezen d.w.z. aan de gebruiker te vragen. Hogere programmeertalen bieden gewoonlijk faciliteiten aan voor het gemakkelijk inlezen van een bepaalde waarde. Overigens: invoer van het toetsenbord is in eerste instantie altijd tekst, die zonodig omgezet moet worden in de gewenste interne representatie. In Visual Basic kan een regel tekst worden ingelezen met InputBox.
- In het programma te zetten. De toe te kennen waarde moet in de programmacode aanwezig zijn of berekend kunnen worden uit andere opgeslagen waarden.
- In te lezen uit een bestand. In het practicum wordt hierop niet verder ingegaan.

Eenmaal over een aantal waarden beschikkende kunnen deze worden gebruikt om andere waarden te berekenen of te wijzigen. Geheugenplaatsen voor de opslag van waarden worden in hogere programmeertalen/-omgevingen **variabelen** genoemd. Wijzigen van de variabele betekent dan wijzigen van de inhoud van de bijbehorende geheugenplaats(en) en dus van de waarde van de variabele. Programmeertalen beschikken uiteraard over een opdracht waarmee de waarde van een variabele kan worden gewijzigd, een zgn. **toekenning.** De waarde die aan de variabele moet worden toegekend kan de vorm van een formule, in vaktermen een **expressie** genoemd, aannemen, waarin ook (de waarde van) andere variabelen en de aanroepen van functies – b.v. sinus en log - mogen voorkomen. Uitvoeren van de toekenning houdt het **evalueren** d.i. uitrekenen van de expressie in; de variabele wordt gelijk gemaakt aan de uitgerekende waarde. De waarde die de variabele voorheen had wordt dan overschreven en is niet langer bechikbaar.

Bij het ontwikkelen van een programma moet je je derhalve steeds afvragen welke uiteindelijke toestand je wilt bereiken en hoe – met welke opdrachten (en dus met welke tussentoestanden) – je deze toestand kunt bereiken. Hoe moelijker dit is d.w.z. hoe meer opdrachten hier voor nodig zijn, des te nuttiger kan het zijn je tussen-toestanden voor te kunnen stellen die zowel gemakkelijker te bereiken zijn vanuit de begintoestand, als het bereiken van de eindtoestand te bereiken. Bij het op volgorde zetten van een rij met n getallen b.v. zou je je een situatie kunnen indenken waarbij al een aantal getallen daarvan op volgorde is gezet. Daaruit een nieuwe situatie zien te bereiken waarin het aantal getallen dat al op volgorde is gezet met een is toegenomen is wel voor te stellen. Immers dit betekent dat het volgende getal op de goede plaats tussen de reeds op volgorde gezette getallen moet worden gezet. Dit betekent het bepalen van dat tweetal getallen in de rij van reeds op volgorde gezette getallen, waarvan het eerste niet groter is dan het toe te voegen getal en het tweede niet kleiner. Het toe te voegen getal kan dan op de plaats van het tweede getal worden ingevoegd. De gehele rij kan op volgorde worden gezet, door bovenstaande acties achtereenvolgens uit te voeren voor de getallen op plaats 2, 3, ..., n.
Overigens: het op deze manier op volgorde zetten van een rij getallen levert niet het snelste mogelijke programma op. Vaak zijn meerdere oplossingen mogelijk.

Het ontwerpen in meer detail

Ter voorbereiding op het uitvoeren van de eindopdracht op de computer zul jij ook een dergelijk ontwerp moeten maken. Vandaar dat ik wat dieper inga op de manier waarop een dergelijk ontwerp tot stand komt.
Gewoonlijk wordt onder het ontwerp van een computerprogramma een schriftelijke weergave van het te ontwikkelen computerprogramma verstaan, die nog niet direct uitvoerbaar is, d.w.z. er is (nog) geen ontwikkelomgeving die dat ontwerp kan uitvoeren.

> Feitelijk is een uitvoerbaar computerprogramma ook een ontwerp. Het wordt echter niet als ontwerp beschouwd, omdat het uitvoerbaar is. Een ontwerp is dus per definitie niet uitvoerbaar. Er zijn zelfs formele talen waarin een specificatie kan worden uitgedrukt, zodat feitelijk de specificatie al het ontwerp en uiteindelijk computerprogramma is. Dergelijke specificaties zijn echter niet zo gemakkelijk te lezen.
> Ideaal zou een ontwerp zijn dat direct – eventueel na vertaling – uitvoerbaar is en ook nog door leken te begrijpen is. Zo'n ontwerp zou daarom in natuurlijke taal uitgedrukt moeten worden, maar helaas zijn er nog geen ontwikkelomgevingen die natuurlijke taal kunnen verwerken tot een werkend computerprogramma. Vandaar dat het ontwerpen kan bestaan uit het maken van een of meerdere representaties van het computerprogramma die steeds formeler worden. Eerst kan uitdrukking in een natuurlijke taal plaatsvinden, vervolgens in de een of andere pseudocode. Eventueel kunnen schematechnieken worden gebruikt voor de leesbaarheid, maar deze moeten dan wel kunnen worden gecommuniceerd. Uiteindelijk dan gecodeerd in de een of andere programmeertaal, klaar om geimplementeerd te worden en getest.

Het ontwikkelen van het ontwerp

In een ontwerp wordt aangegeven hoe uit de invoer de uitvoer moet worden geproduceerd. Het is dus belangrijk dat de ontwerper precies weet welke uitvoer bij welke invoer moet worden geproduceerd. Hij moet de specificatie goed begrijpen.

Het ontwerp bestaat niet alleen uit een of meerdere algoritmen, maar ook uit een lijst van variabelen die in de algoritmen worden gebruikt waarvan de waarden de toestand van het programma zullen vormen. In de algoritmen komen actie-opdrachten voor die de toestand – de waarden van de variabelen - van het programma wijzigen en besturingsopdrachten die bepalen welke opdrachten wanneer moeten worden uitgevoerd.

Niet altijd worden de invoergegevens allemaal gelijktijdig bewaard. Zo is het heel goed mogelijk de som van een rij in te lezen getallen te berekenen en weer te geven zonder op zeker moment alle getallen in variabelen te hebben opgeslagen. Er hoeven slechts twee waarden te worden onthouden: die van het (laatst) ingelezen getal en die van de som. Voordat met inlezen wordt begonnen kan de som gelijk aan 0 worden gemaakt. Vervolgens kan dan een aantal malen een getal worden ingelezen (en opgeslagen als waarde van één en dezelfde variabele) waarmee de som wordt opgehoogd.

Er zijn echter ook situaties waarin het de voorkeur geniet over alle invoergegevens te kunnen beschikken. Om een rij getallen te sorteren (d.i. op volgorde zetten) moeten meerdere keren twee waarden met elkaar worden vergeleken. Het is dan ondoenlijk om deze waarden steeds opnieuw in te moeten lezen.

Hetzelfde geldt voor de uitvoergegevens. Bij het sorteren bevindt zich uiteindelijk de rij getallen die op volgorde is gezet, zich in zijn geheel in het geheugen en kan zo

worden weergegeven. Voor het weergeven van alle getallen in een ingelezen rij die positief zijn hoeven ook weer niet alle getallen gelijktijdig in het geheugen te staan: het is voldoende om steeds een getal in te lezen, te kijken of het positief is en deze dan weer te geven. Voor de opslag van dat ene getal is slechts één variabele nodig.

Bij het ontwerpen speelt hergebruik een belangrijke rol. Hierboven zag je al dat één variabele gebruikt werd voor de opslag van steeds het laatste ingelezen getal. Je zou kunnen zeggen: die variabele wordt hergebruikt. Niet alleen variabelen kunnen worden hergebruikt, maar ook opdrachten. In beide gevallen neemt de hoeveelheid geheugen af nodig voor de opslag van variabelen resp. opdrachten.

Een algoritme voor de berekening en weergave van de som zou kunnen zijn:

1. **Maak som gelijk aan 0.**
2. **Lees volgend getal in.**
3. **Tel bij som het ingelezen getal op.**
4. **Zijn er geen nog in te lezen getallen, geef dan de som weer, ga anders verder met 2.**

In dit voorbeeld worden de opdrachten 2, 3 en 4 hergebruikt. Zonder deze mogelijkheid tot hergebruik zou het overigens niet mogelijk zijn om van een rij getallen de som te bepalen als van te voren het aantal getallen onbekend is. Hoe zou je immers in een programma moeten aangeven wanneer het laatste getal is ingelezen: je weet immers niet hoeveel getallen moeten worden ingelezen. Je kunt dan slechts een vast aantal getallen inlezen! Hergebruik van opdrachten is dus niet alleen handig, het is soms ook onvermijdelijk!

Het bedenken van algoritmen

Allereerst: het is niet mogelijk om een strategie, techniek of methode te geven die in alle gevallen een goed ontwerp oplevert. Het ontwerpen van een computerprogramma is immers vergelijkbaar met het oplossen van een probleem. Er zijn wel algemene probleemoplossingsstrategieen maar die kunnen succes niet garanderen! Een goede – b.v. object-georiënteerde - ontwerpmethode kan je overigens wel beschermen tegen veelgemaakte fouten.

Ontwerpen bestaat niet uit het volgen van de een of andere methode maar uit leiden! Je moet bedenken hoe een bepaalde taak moet worden uitgevoerd. Bij een computerprogramma houdt dat het produceren van een bepaalde uitvoer in die nog niet voorhanden is uit een bepaalde invoer. Om een algoritme op te kunnen stellen is het noodzakelijk te weten welke acties wel en welke niet door het te maken computerprogramma uitgevoerd kunnen worden. Want dan weet ik over welke bouwstenen ik kan beschikken. (Vergelijkbaar met de uit te voeren handelingen in een recept: de maker ervan moet weten welke handelingen een lezer kan uitvoeren en welke niet!) Je moet dus weten welke acties **elementair** zijn en welke niet. Ook wordt wel van **primitieve** acties gesproken.

Maar het was juist de bedoeling om bij het maken van het logisch ontwerp geen gebruik te maken van de mogelijkheden van een bepaalde programmeeromgeving! Hoe weet ik nou welke primitieven beschikbaar zijn?
Nu, vaak is wel bekend wat voor soort programmeeromgeving gebruikt zal gaan worden en heeft de ontwerper een beeld van het soort acties dat daarin kan worden uitgevoerd. De in deze handleiding beschreven pseudocode is gebaseerd op de mogelijkheden van hogere programmeertalen als Visual Basic zonder daarbij van de syntaxis van Visual Basic uit te gaan. Alle in deze pseudocode beschreven algoritmen kunnen ook gemakkelijk in Visual Basic worden gerealiseerd.

Vooral als beginner ben je niet in staat direct te overzien hoe de primitieven gebruikt kunnen worden om het beoogde resultaat te bereiken. Geen paniek! Vaak heb je wel een idee wat het uitvoeren van een dergelijke taak inhoudt, en kun je dus een beschrijving daarvan geven in termen van een aantal uit te voeren deeltaken. Deze beschrijving vormt dan je hoofdalgoritme. Van alle taken daarin die niet met één elementaire actie zijn uit te voeren werk je verder uit in een nieuw deelalgoritme. Op zeker moment zijn alle taken beschreven in elementaire acties en ben je klaar.

Deze werkwijze heeft een naam: het is een **verdeel-en-heers-strategie.** Je gebruikt hem elke dag bij het bedenken wat je die dag zal doen. In eerste instantie zal dat b.v. zijn: douchen, aankleden, ontbijten, college lopen, lunchen, een practicum-oefening doen, avondeten, leren en uitgaan. Elke activiteit bestaat echter uit een heleboel door je lichaam uit te voeren acties. Ik vermoed dat je lichaam op het signaal college te lopen niet begrijpend zal reageren; het lichaam kan immers alleen maar bewegen (of niet natuurlijk!). Daarom moet elke activiteit net zolang in deelactiviteiten worden opgedeeld totdat de uit te voeren lichamelijke acties overblijven waarover niet meer hoeft te worden nagedacht, die dus 'automatisch' kunnen worden uitgevoerd. De niet direct uit te voeren handelingen worden als 'abstract' ervaren; de wel direct uit tevoeren handelingen zijn 'concreet'. Bij het uitwerken van het ontwerp neemt het abstractieniveau af totdat een bechrijving in concreet uit te voeren acties overblijft. Bij het kiezen van deeltaken speelt ervaring dus een belangrijke rol.
Bij het maken van een ontwerp gebruik je in eerste instantie gewoon de natuurlijke taal. Zo ontstaat een algoritme dat - per definitie - niet direct uitvoerbaar is in een computer of in te voeren in een beschikbare ontwikkelomgeving. Voordeel van een algoritme is natuurlijk wel dat deze gecommuniceerd kan worden met anderen die daarvoor geen programmeertaal hoeven te kennen, maar die wel hun licht over de juistheid ervan kunnen laten schijnen.
Het is verstandig om het algoritme eerst te vertalen naar een tussenvorm die wat formeler is dan de natuurlijke taal en wel op een programmeertaal lijkt, maar het nog niet is. Uitdrukking van een algoritme in een dergelijke tussenvorm wordt pseudocode genoemd. Het hoeft niet altijd te gaan om tekst alleen, ook figuren kunnen daarbij worden gebruikt. Bekende gestandaardiseerde schema-technieken zijn o.a**. flow charts** (zie boek) **en programma-structuur-diagrammen** ofwel **PSD's**. (Het maken van dergelijke schema's komt in het college aan bod.) Hierbij worden een aantal verschillende soorten opdrachten met verschillende figuren weergegeven. PSD's hebben verschillende figuren voor alle verschillende besturingsopdrachten behalve sprongopdrachten, terwijl flow charts eigenlijk alleen beschikken over een aparte figuur voor een selectie en alle opeenvolging de vorm van sprongopdrachten aanneemt. Ook al verdienen PSD's de voorkeur vanwege de noodzaak tot netjes programmeren, vooral voor beginnelingen is het gemakkelijker om flow charts te maken, en gebruik te maken van sprongopdrachten, en daarin de selecties en lussen te herkennen.
Een dergelijke schematechniek formaliseert de notatie van - met name - de verschillende soorten besturingsopdrachten maar nog niet de inhoud ervan - de actie of bewering zelf. Deze kan nog steeds in natuurlijke taal worden uitgedrukt. Wel dwingt een dergelijke schematechniek ons ertoe de volgorde waarin de opdrachten dienen te worden uitgevoerd op te geven. Een dergelijke schematechniek is derhalve alleen geschikt voor algoritmen waarbij de volgorde moet worden opgegeven: de zgn. **procedurele** of **imperatieve algoritmen**. Deze vormen wel de bulk van alle momenteel in gebruik zijnde algoritmen.

Door gemeenschappelijke kenmerken van hogere - procedurele - programmeertalen op te sporen is het ook mogelijk een formalisering van de inhoud van de opdrachten te

realiseren. Zoals al eerder is vastgesteld maken de (besturings)opdrachten gebruik van de toestand van het programma, d.w.z. de verzameling van actuele waarden die het programma nodig heeft om zijn werk te kunnen doen. Zo'n toestand is te vergelijken met de toestand waarin b.v. een warme-dranken-automaat zich bevindt tijdens het verwerken van het verzoek van een klant. Wat een dergelijke automaat doet is - terecht - afhankelijk van wat de klant tot dan toe heeft gedaan. Zo'n automaat zal pas leveren zodra het bestelde betaald is. De automaat moet dus (o.a.) 'weten' hoeveel er betaald is. Hoe kan zo'n apparaat dat weten? Net als wij mensen dingen weten dankzij ons geheugen, moet ook zo'n automaat over een geheugen beschikken waarin (o.a.) staat opgeslagen wat er tot nu toe betaald is. Het geheugen moet groot genoeg zijn om alle informatie te bewaren die op willekeurig welk moment nodig is om de gewenste functionaliteit te realiseren. Meestal kan een geheugenplaats niet voor willekeurig welk gegeven worden gebruikt, maar zullen bepaalde gegevens op vaste plaatsen staan. Hoe kan de automaat immers weten waar bepaalde informatie staat. Hooguit door dan toch weer ergens op een vaste plaats in zijn geheugen te onthouden waar die informatie staat; hierdoor neemt echter alleen maar de hoeveelheid geheugen toe en dat is meestal niet nodig!

Belangrijk bij de formalisering van het algoritme is dus vaststellen welke gegevens onthouden - en dus bewaard - moeten worden door het programma om zijn werk te kunnen doen. Een eenvoudige stelregel is: elke waarde die je meer dan één keer in het programma nodig hebt moet worden bewaard! En er moet dan dus een variabele worden gedeclareerd waarin die waarde wordt opgeslagen. De waarden die de variabelen kan aannemen bepalen welk type je moet kiezen.
Elk zelfstandig naamwoord in een opdracht kan een grootheid zijn en dus een variabele zijn.

Samenvatting
Het schrijven van een computerprogramma wordt ook wel **programmeren** genoemd. Dit is echter een naam die de lading niet meer dekt. Programmeren in enge zin is nl. de aktiviteit waarbij het ontwerp van een computerprogramma wordt omgezet in een vorm die door de computer kan worden uitgevoerd. Dit omzetten wordt **coderen** genoemd.
Het uitvoeren van een computerprogramma wordt ook wel **draaien (*run*)** genoemd. De computer waarop het programma zal moeten gaan draaien wordt ook wel **doelmachine** genoemd; met name wanneer het computerprogramma op een andere machine wordt ontwikkeld (wat met name bij commerciëel gemaakte applicaties het geval zal zijn).
Programmeren echter betreft niet alleen het specifieke coderen, maar ook het ontwerpen van het computerprogramma. Een computerprogramma wordt gewoonlijk niet direct in een (hogere) programmeertaal uitgedrukt. Meestal wordt een beschrijving ervan in gewone taal gemaakt. Dit algoritme kan vervolgens in de een of andere pseudocode en gebruikmakende van de een of andere schematechniek formeler worden uitgedrukt. Het zo ontstane ontwerp kan dan relatief gemakkelijk worden omgezet in een uitvoerbaar computerprogramma (dit heet coderen). De maker van het ontwerp moet weten welke primitieve acties en gegevensstructuren beschikbaar zijn!
Bij het uitdrukken van een in natuurlijke taal opgesteld algoritme in pseudocode moeten de variabelen worden gekozen. Het type van de variabelen hoeft dan nog niet gekozen te worden. Kennis van de mogelijkheden van programmeertalen in het algemeen is nodig om de pseudocode te kunnen maken en gebruiken en bij benadering te weten welke basisbewerkingen mogelijk zijn. Overigens: als een bepaalde bewerking achteraf niet direct beschikbaar blijkt te zijn dan kan daarvoor natuurlijk altijd een algoritme worden opgesteld.

4.2 Gegevensstructuren

Een aantal bij elkaar horende gegevens vormen wat een **gegevensstructuur** wordt genoemd. Als een gegevensstructuur een eerste, tweede, ... en laatste gegeven bevat spreken we van een **rij**. Zo is een byte een rij (van 8) bits.
Als geen opeenvolgende elementen zijn te onderscheiden, of als de volgorde er niet toe doet, dan spreken we van een **verzameling** b.v. de kaarten in de hand van een kaartspeler.

Een gegeven dat niet als opgebouwd uit andere gegevens wordt beschouwd, als ondeelbaar, en dus uit één waarde bestaat, wordt een **enkelvoudige (*simple*)** waarde genoemd, anders een **samengesteld (*compound*)** gegeven.
Of een bepaald gegeven als enkelvoudig of samengesteld wordt beschouwd hangt helemaal af van de aanschouwer. Zo beschouwt een computer niet de bit maar een woord als het kleinste ondeelbare en dus enkelvoudige gegeven. Een combinatie van dergelijke woorden voor de opslag van b.v. een reeel getal wordt door de computer als samengesteld gegeven beschouwd, maar door de gebruiker van een hogere programmeertaal als enkelvoudig, ondeelbaar, gegeven, omdat de afzonderlijke waarden waaruit het reële getal is opgebouwd, mantisse of exponent, niet langer beschikbaar zijn voor manipulatie. Dit maakt al duidelijk, dat de mogelijkheden bepalen of een verzameling gegevens als samengesteld of enkelvoudig moeten worden beschouwd. Elke programmeertaal biedt de gebruiker ervan een aantal, als ondeelbaar te beschouwen, standaard gegevenstypen aan voor de opslag van b.v. tekens, gehele en reële getallen en zelfs tekst, waarop een aantal standaard operaties kan worden uitgevoerd. De opgave van de naam en het type van een variabele wordt een **declaratie** genoemd. De declaratie stelt de ontwikkelomgeving in staat te controleren of geldige bewerkingen op de variabele worden uitgevoerd, d.w.z. of wel een juiste waarde in de variabele wordt gezet, waardoor dergelijke fouten tijdig kunnen worden opgespoord.

> Is tekst, bestaande uit een aantal tekens, een enkelvoudig of samengesteld gegeven?
> Als rij tekens beschouwd natuurlijk als samengesteld gegeven, maar in veel ontwikkelomgevingen is het mogelijk om de tekst als één waarde te gebruiken en op te slaan. Voor de gebruiker van die ontwikkelomgeving is het dus een enkelvoudig gegeven, voor de maker van de ontwikkelomgeving een samengesteld gegeven, omdat hij de afzonderlijke tekens moet opslaan! In Visual Basic is het mogelijk tekst als één waarde te beschouwen. Gebruiken van afzonderlijke tekens is dan weer niet goed mogelijk.

Tijdens het ontwerpen van een computerprogramma kun je zelf bepalen welke gegevensstructuren je wilt gebruiken; bij het bedenken ervan zou je je niet moeten hoeven beperken tot gegevensstructuren die gemakkelijk geimplementeerd kunnen worden. Ontwikkelomgevingen evolueren omdat ze voor steeds meer toepassingsgebieden – met hun eigen specifieke gegevensstructuren – geschikt willen zijn.

Het is niet voldoende om alleen variabelen van dergelijke standaardtypen te kunnen declareren. Het moet ook mogelijk zijn om gegevens tot nieuwe gegevensstructuren te kunnen combineren. Welke mogelijkheden hiertoe moet een ontwikkelomgeving aanbieden? Dit hangt helemaal af van het soort toepassingen die met de ontwikkelomgeving zullen worden ontwikkeld. In de loop der jaren zijn het aantal toepassingsgebieden sterk toegenomen, wat zowel tot nieuwe, als enkelvoudig te beschouwen, standaard gegevenstypen heeft geleid, als tot de manier waarop deze tot gegevensstructuren konden worden gecombineerd. In de begindagen werden met name wetenschappelijke toepassingen ontwikkeld, waarin veel met vectoren en

matrices moest kunnen worden gewerkt. Dit hield in dat het praktisch was om de elementen van een vector of matrix in een programmeertaal op eenzelfde wijze te kunnen aanduiden als gebruikelijk was in de wetenschappelijke traditie nl. met een naam met een of twee indices als onderschrift: zo stelt t_i element i van vector t voor en $a_{i,j}$ het element in kolom j van rij i in matrix a voor. Het aanbrengen van onderschriften in de opdrachten was toen echter nog niet mogelijk (en is het nog steeds niet) vandaar dat een andere notatie moest worden gekozen. In een taal als Pascal worden de indices achter de naam tussen vierkante haken geplaatst b.v. a[i,j], in C en Basic tussen ronde haken b.v. a(i,j). Dit betekende dat het mogelijk moest zijn een vector of matrix met één enkele gemeenschappelijke naam aan te geven. Het moest mogelijk zijn om aan te geven dat een variabele een vector of matrix was. Daartoe konden in ontwikkelomgevingen op zeker moment zgn. **arrays** worden gedeclareerd. Werd een variabele als array gedeclareerd, dan moest worden aangegeven van welk type elk element in de array zou zijn en wat de eerste en hoogste te gebruiken index zou mogen zijn. Zo kan in Visual Basic een variabele waarin de elementen van de 10x20-matrix A moet kunnen worden opgeslagen worden gedeclareerd met: Dim A(1 To 10,1 To 20) As Double. Tussen de haakjes staat voor elke dimensie van de array de index van het eerste en hoogste element vermeld. De matrix A wordt hier opgeslagen in een array met twee dimensies, een twee-dimensionale array. Het element in rij i van kolom j van A kan in Visual Basic worden aangegeven als A(i,j). In totaal zal A dus 200 elementen van het type Double – voor de opslag van een groot reëel getal – kunnen bevatten.
Het aantal dimensies en het type elementen dat in een array kan worden opgeslagen is vrij te kiezen. Voor de opslag van een vector/matrix kies je natuurlijk een een/twee-dimensionale array. Natuurlijk kan een matrix ook in een een-dimensionale array worden opgeslagen, maar dit wordt gewoonlijk alleen gedaan bij speciale matrices, zoals boven-/benedendriehoeksmatrices of *sparse* matrices, omdat dat geheugen-ruimte spaart.
Als toegift kunnen gewoonlijk ook arrays van meer dan twee dimensies worden gedeclareerd. Te denken valt b.v. aan een array waarin voor een heel jaar de gemiddelde dagtemperaturen voor een tabel van lengte- en breedte-graden moet kunnen worden opgeslagen, waarvoor een drie-dimensionale array goed gebruikt kan worden.

Daarna kwamen de administratieve toepassingen. In dergelijke toepassingen wordt veel gewerkt met entititeiten, zoals personen en orders en rekeningen, met een aatal attributen, als namen, geslacht, rekeningnummer, niet noodzakelijk van hetzelfde type. Natuurlijk was het mogelijk de afzonderlijke attributen behorende bij een rij personen in aparte arrays op te slaan, maar het verdient de voorkeur om alle gegevens in eenzelfde gegevensstructuur op te kunnen slaan. Hiertoe voegden ontwikkelomgevingen de mogelijkheid tot het definiëren van een zgn. **recordtype** toe. De declaratie van een recordtype houdt het declareren van de afzonderlijke attributen, zgn. **velden**, (en dus ook van de – mogelijkerwijs verschillende – gegevenstypen ervan) in. Hier is dus sprake van de mogelijkheid zelf een nieuw gegevenstype te declareren. In Visual Basic doe je dit met het Type-statement (zie § 7.2). Arrays kunnen in VB niet als nieuw type worden gedeclareerd. Het is dus niet mogelijk een array met elementen te declareren die ook weer array's, terwijl het wel mogelijk is een array met elementen te declareren die records zijn.

In eerste instantie konden bij de zelf te definiëren arrays en recordtypen geen operaties die op gegevens van dat type konden worden toegepast worden gedefiniëerd, zoals bij de standaardtypes mogelijk was. ('Optellen' d.i. achter elkaar zetten van twee teksten kon met de +-operator.) Sommige programmeertalen, b.v. Ada, maakten het declareren van een **abstract data type** mogelijk, waarbij naast

de datavelden ook de operaties die op variabelen van dat type konden worden uitgevoerd opgegeven konden worden. Voor een computerprogramma dat b.v. gemakkelijk met complexe getallen moet kunnen werken is het handig een abstract datatype te kunnen definiëren waarin niet alleen het reële en imaginaire deel van het complexe getal kunnen worden opgeslagen maar ook de gebruikelijke binaire operatires als optellen, aftrekken, vermenigvuldigen en delen kunnen worden gedefiniëerd, die dan in het programma kunnen worden gebruikt als betrof het een standaardtype.
Visual Basic kent geen mogelijkheden tot het definiëren van abstracte datatypen.

De apotheose in de mogelijkheden van ontwikkelomgevingen tot het zelf definiëren van gegevensstructuren is de mogelijkheid in object-georiënteerde ontwikkel-omgevingen om **klassen** te declareren, vergelijkbaar met een abstract datatype, maar met aanvullende mogelijkheden b.v. tot het bundelen van een aantal klassen in een klasse-hiërarchie met als belangrijkste voordeel hergebruik van code.
De gegevens in de klasse gedeclareerd staan opgeslagen in ***members***; de operaties die op objecten van de klasse kunnen worden uitgevoerd worden, worden **methoden** of ***member functies*** genoemd. In tegenstelling tot variabelen van andere soorten datatypes, wordt nog geen geheugenruimte gereserveerd t.g.v. de declaratie van een variabele van een bepaalde klasse. Overigens: dan wordt niet langer van een variabele gesproken maar van een **object**! (zie ook hoofdstuk 7)

Representatie van gegevensstructuren

Gegevensstructuren bestaan tijdens de ontwikkeling van een computerprogramma, net als het programma zelf, op verschillende niveau's.
Een computerprogramma kan in diverse vormen bestaan:

- als een verzameling algoritmen in het logisch ontwerp (het ontwerp);
- als een bronprogramma in een fysisch ontwerp (de sourcecode);
- als een uitvoerbaar (doel)programma (de executable).

In elk van die vormen komen gevensstructuren voor, op een steeds hoger abstractieniveau.

Logische gegevensstructuren

De gegevensstructuren die in het logische ontwerp worden gebruikt worden logische gegevensstructuren genoemd; welke dit zijn wordt slechts begrensd door de fantasie van de ontwerper die de gegevens structureert in rijen, lijsten, verzamelingen, tabellen, bomen, stapels, records enzovoorts.

Fysische gegevensstructuren

De gegevens die in het fysisch ontwerp gebruikt worden worden fysische gegevensstructuren genoemd. Het zijn de mogelijkheden van de te gebruiken ontwikkelomgeving die naast een aantal enkelvoudige gegevenssoorten het organiseren, samenvoegen van gegevens in gegevensstructuren toestaat. Bovendien is het gewenst dat een dergelijke ontwikkelomgeving de gebruiker toestaat zelf gegevensstructuren te definiëren.
Bij het omzetten van een logisch ontwerp in een fysisch ontwerp moeten voor gekozen logische besturingsstructuren bijbehorende fysische gegevensstructuren worden gekozen. De gekozen fysische besturingsstructuur wordt de implementatie van de logische besturingsstructuur genoemd.
Zelfs de meest toepasselijke fysische besturingsstructuur waarover een bepaalde ontwikkelomgeving beschikt kan nog verre van ideaal zijn. Zo heeft het implementeren van een lijst in een array het nadeel, dat tussenvoegen en verwijderen van elementen uit de lijst niet gemakkelijk is.Het geheugen in een computer bestaat uit geheugenelementen met opeenvolgende adressen, meestal bytes, waarin waarden

kunnen worden opgeslagen. Meestal moeten meerdere bytes gelijktijdig worden opgehaald voor verwerking. De inhoud als getal beschouwende, is de inhoud van het geheugen te beschouwen als een (lange) rij gehele getallen. Voor het verwerken van (rijen) gehele getallen is een computer dus met name geschikt. Wanneer het om het verwerken van niet-gehele getallen gaat of andere gegevensstructuren dan rijen is een afbeelding van de denkbeeldige gegevensstructuur naar de organisatie van het geheugen noodzakelijk!

Echt of nep?

Bepaalde eigenschappen of entiteiten die tot de hardware schijnen te behoren, maar in werkelijkheid door een combinatie van hardware en software worden nagebootst, worden **virtuele** of **conceptuele** eigenschappen of entiteiten genoemd. (Denk b.v. aan de entiteit 'bestand', die door het besturingssysteem wordt gebruikt, en waarbij de indruk wordt gewekt dat het een hardware-entiteit is, terwijl de hardware alleen sporen, tracks, en cylinders kent! Ook de 'directory' is een virtuele entiteit, door het besturingssysteem geïntroduceerd.)
Een ander voorbeeld is de situatie waarin een machine meerdere machines lijkt te zijn door gebruik te maken van ***time-slicing*** (het cyclisch toekennen van procestijd aan de diverse gebruikers). En zelfs het feit dat een computer in staat is de woorden in een programmeertaal te begrijpen door een ***interpreter*** te gebruiken.
Dergelijke, ten onrechte aan de computer toegekende, eigenschappen of erin voorkomende entiteiten worden in het leven geroepen om de gebruiker in staat te stellen op een hem meer vertrouwde wijze met de computer om te gaan. Een bestand is immers vergelijkbaar met een document, en een directory met een folder of map.

Hetzelfde geldt voor gegevens. Gebaseerd op de toepassing en de manier van omgang met gegevens organiseren we deze gegevens in b.v. rijen, vectoren, verzamelingen, lijsten, tabellen, matrices, stapels, bomen, wachtrijen en noem maar op. Ze zijn in die zin nep dat een computer ze moet representeren in de rij geheugenelementen die het ter beschikking heeft en dus niet over een equivalente opslagmogelijkheid beschikt.
Bij de omzetting van het ontwerp van een computerprogramma in een bronprogramma dat door een bepaalde ontwikkelomgeving moet worden verwerkt moeten voor de in het ontwerp gebruikte gegevensstructuren, die ook wel **logische** gegevensstructuren worden genoemd, bijbehorende in de ontwikkelomgeving beschikbare gegevensstructuren worden gekozen, **fysische** gegevensstructuren. Zijn in een bepaalde ontwikkelomgeving b.v. geen faciliteiten aanwezig om direct met verzamelingen te werken maar wel met rijen, dan moet elke verzameling in een rij worden opgeslagen. Men zegt dan: de verzameling wordt als rij geimplementeerd. Uiteindelijk moet elke ontwikkelomgeving de aanwezige fysische gegevensstructuren implementeren op een digitale computer die alleen over één lange rij geheugen-elementen beschikt! Het – relatieve – gemak waarmee rijen op een computer kunnen worden geimpementeerd betekent dat ontwikkel-omgevingen altijd wel over gegevensstructuur beschikken waarin een rij gegevens kan worden opgeslagen. (In Visual Basic is dit de een-dimensionale array).

Organisatiestructuur van primair geheugen

Het geheugen van de computer is georganiseerd als een rij opeenvolgende geheugenelementen van een vaste grootte, gewoonlijk bytes. De inhoud van zo'n geheugenelement is een binaire code, die zowel als de representatie van een teken, als van een - klein - binair getal kan worden beschouwd, maar de betekenis ervan wordt eraan toegekend door het programma dat met de binaire codes werkt, en dus door de maker van het programma, en niet door de computer zelf.

De woordlengte van een computer, het aantal bits in de registers van de CVE, bepaalt met welke aantallen geheugenelementen er gelijktijdig kan worden gewerkt. De computer stelt gewoonlijk een aantal instructies ter beschikking die ervan uitgaan, dat een woord een - evt. niet-negatief - geheel getal voorstelt. Andere diensten dan het uitvoeren van de elementaire bewerkingen op gehele getallen stelt de computer standaard niet beschikbaar. Dergelijke - niet standaard beschikbare - diensten moeten verricht worden door meerdere instructies! Voor de berekening van een vermenigvuldiging van twee reële getallen zijn zo al honderden machinetaalinstructies nodig. Het is een enorme verspilling om deze instructies elke keer opnieuw te moeten laten bedenken. Distributie van dergelijke diensten is dan ook een natuurlijk gevolg. Dit heeft geleid tot het ontstaan van hogere programmeertalen waarin het opvragen van dergelijke diensten - relatief - gemakkelijk is.

Aanduiding van gegevens

Hoe werk je in een computerprogramma nu met een bepaald gegeven?
In de machinetaal wordt alleen met geheugenelementen gewerkt en de adressen daarvan; van elk gegeven moet het adres worden gebruikt! (Gegevens kunnen niet van adres veranderen!)
In een assemblertaal kunnen adressen een naam worden gegeven; elk gegeven kan nu met zijn naam worden aangeduid. Er bestaan alleen geheeltallige gegevenssoorten die rechtstreeks door de computer kunnen worden gebruikt (woorden, bytes).
In hogere programmeertalen zijn de adressen door namen vervangen, èn kunnen de gegevens van andere soorten zijn dan door de computer rechtstreeks kunnen worden gebruikt. Elk gegeven wordt - omdat de waarde ervan variabel is - een variabele genoemd.

Evolutie van vertaalprogramma's

In de evolutie van vertaalprogramma's zijn op dit gebied twee trends te ontdekken: toename van de beschikbare standaardtypes en toename van de mogelijkheden tot het zelf definiëren van types.
Een standaardtype omhelst niet alleen een bepaalde - interne - representatie, maar tevens diensten die met betrekking tot gegevens van dit type (operaties) kunnen worden uitgevoerd.
Een standaardtype voor de opslag van tekst stelt de gebruiker b.v. in staat twee teksten met elkaar te vergelijken, of te concateneren (aan elkaar te plakken).
Een dergelijke uitbreiding is natuurlijk pas rendabel als er voldoende behoefte aan bestaat.
De standaardtypes vormen de basis voor nieuwe - door de gebruiker - zelf te definiëren typen. Daarbij kan gedacht worden aan varianten van de enkelvoudige standaardtypen (zoals deelbereiken en opsommingstypen) of aan samengestelde typen, zoals de verzameling (*set*) of array.
Een **deelbereiktype** is een type met waarden in een deelbereik van de waarden die een standaardtype kan aannemen. Andere waarden mag een gegeven van dat deelbereiktype niet aannemen en het vertaalprogramma controleert dit. Dit bevordert de betrouwbaarheid van het programma door een verbeterde controle op de juistheid van de waarden van de gegevens.
Bij een **opsommingstype** geldt hetzelfde, alleen mogen voor de verschillende waarden zelfgekozen namen worden gekozen; b.v. rechthoek, cirkel, driehoek etc. Dit verbetert de leesbaarheid van het programma.
Een **verzameling** bestaat uit een aantal - ongeordende - elementen. Deze elementen moesten in eerste instantie van hetzelfde type zijn. In tegenstelling tot een verzameling is een array juist een geordende verzameling elementen d.w.z. de elementen hebben een nummer en zijn via dit nummer beschikbaar.

4.3 De ontdekking van algoritmen

Het maken van een algoritme komt overeen met het oplossen van een probleem. Ieder van ons beschikt wel in meer of mindere mate over probleemoplossingsvermogen. (Het is overigens beter te spreken van probleemoplossende vermogens: er zijn immers ook verschillende soorten intelligentie.)
Voor het oplossen van problemen in het algemeen is wel een algoritme op te stellen, maar dit is zo abstract dat je daar niet veel aan zult hebben. De omschrijving ervan komt nog het meest overeen met de vallen-en-opstaan-strategie (*trial-and-error*): een toch beslist niet ideale werkwijze. Wel kan op deze manier een beter begrip van het probleem worden verkregen; maar het blijft een zwaktebod!

Beter is het hier aan te geven welke strategieën gebruikt kunnen worden om het probleem op te lossen.
Natuurlijk zijn er de algemene probleemoplossingsstrategieen. Een ervan hebben we zelfs al toegepast: de **verdeel-en-heersmethode**. Deze methode houdt het opdelen van het probleem in deelproblemen in, net zolang totdat we van elk deelprobleem de oplossing weten, waarmee het probleem is opgelost. Een dergelijke aanpak wordt ook wel **stapsgewijze verfijning** genoemd. (Stapsgewijze verfijning wordt beschouwd als een top-down-methode omdat deze van algemeen naar specifiek gaat.) Maar: van welke deelproblemen kennen we nu de oplossing?

Een voorbeeld
Je hebt geleerd hoe je een variabele een andere waarde moet geven. Stel dat je twee variabelen genaamd Ene en Andere hebt wiens waarden moeten worden verwisseld; d.w.z. Ene moet de huidige waarde van Andere krijgen en Andere de huidige waarde van Ene. Aan Ene de waarde van Andere geven en vervolgens aan Andere de waarde van Ene geven heeft niet het gewenste resultaat: beide variabelen zullen gelijk zijn aan de oorspronkelijke waarde van Andere. Waarschijnlijk kom je er pas na vergelijking met een situatie uit de praktijk van alledag achter hoe de verwisseling kan worden gerealiseerd. Stel, nl. dat je de inhoud van twee flessen wilt verwisselen! Juist, door eerst de inhoud van de ene fles in een derde fles te gieten. Er is dus een derde variabele nodig. Als dit de variabele Hulp is, kan een algoritme voor de verwisseling gemakkelijk worden opgesteld:

Maak Hulp gelijk aan Ene
Maak Ene gelijk aan Andere
Maak Andere gelijk aan Hulp

Na het opstellen van dit algoritme is verdere detaillering niet meer nodig. Dit betekent dan ook, dat enig inzicht in de mogelijkheden van de programmeertaal waarin het programma uiteindelijk zal worden uitgedrukt nodig is om vast te kunnen stellen wanneer voldoende detaillering is aangebracht. Zonder voldoende kennis omtrent de mogelijke opdrachten in de programmeertaal is het in voldoende detail opstellen van een algoritme moeilijk zo niet onmogelijk.
Kortom: een tweede algemene oplossingsstrategie is het **gebruik maken van analogieën**. Zo is het b.v. mogelijk een sorteeralgoritme te bedenken (voor het in oplopende volgorde zetten van een rij) gebruikmakende van de methode die we gebruiken als we een rij kaarten op volgorde zetten. (Dit sorteeralgoritme levert overigens niet het snelste algoritme op!)
Naast deze algemene probleemoplossingsstrategieën zijn er nog een strategieën die in het bijzonder bij het ontwerp - of liever: de ontdekking - van algoritmen gebruikt

kunnen worden. Deze lossen ook niet echt het probleem op maar helpen je - zoals dat heet - '**een voet tussen de deur te krijgen**'.
Vaak weet men niet waar te beginnen. In dat geval zou ik willen zeggen: begin maar ergens en zie maar waar het schip strandt (als het al strandt). Dit is de strategie die in het dagelijks leven voor zowel aanzienlijke hoeveelheden geluk als ongeluk zorgt en dit geldt uiteraard ook bij de ontwikkeling van een algoritme. In beide gevallen geldt tevens dat een behoorlijk inzicht in waar te beginnen de kans op succes aanzienlijk zal vergroten. En dat is nou juist de kennis waarover een beginneling niet beschikt. Tsja, als alles faalt kun je er nog altijd de meester bij roepen. Maar bedenk: je kunt alleen vertrouwen in je eigen aanpak krijgen als je de aanpak zelf hebt bedacht en gezien hebt dat deze werkt! Overigens: de praktijk leert dat de oplossing voor een probleem niet altijd gevonden wordt door aanhoudende geestelijke inspanning: soms schiet je ineens iets te binnen als je je met heel andere zaken bezighoudt (maar wacht niet te lang).
Een dergelijke strategie kan b.v. worden toegepast bij het bepalen van het maximum van een rij getallen. Het heeft weinig zin je het hoofd te breken over welke het is! Nou goed, doe maar even alsof het eerste getal het grootste getal is. In de meeste gevallen zal dit natuurlijk niet kloppen, o.a. als het tweede getal nog groter is. Als dit het geval is, dan kan het tweede getal het grootste getal zijn. Nou, ga daar dan maar van uit. Elk (volgend) getal wordt met het tot dan toe grootste getal vergeleken. Als alle getallen in de rij onderzocht zijn zal het tot dan toe gevonden grootste getal ook daadwerkelijk het grootste getal zijn.
Dit algoritme maakt gebruik van inductie nl. de wetenschap dat het maximum van n getallen gelijk is aan het maximum van het maximum van n-1 getallen en het (resterende) n-de getal (voor elke n>0). Het maximum van 1 getal kennen we: dat is dat getal zelf. Zodra het maximum van een aantal getallen verkregen is kan het maximum van een rij met één extra element gemakkelijk worden bepaald. Hetzelfde principe kan worden toegepast bij het bepalen van de som van n getallen: die is immers gelijk aan de som van de som van n-1 getallen en het resterende n-de getal (voor elke n>0) (en de som van 0 getallen is 0)!
Twee vergelijkbare strategieën zijn: **het zoeken van een overeenkomstig probleem** dat gemakkelijker op te lossen is of waarvan je de oplossing kent, en **het gebruiken van concrete voorbeelden**. Bij elke geslaagde vorm van induktie wordt de eerste strategie toegepast. De tweede strategie kan b.v. worden toegepast als we naar een manier zoeken om een rij getallen op volgorde te zetten nl. door eerst eens te kijken hoe we een rij van b.v. 3 getallen op volgorde zouden zetten. Hieruit zou dan een manier kunnen worden afgeleid voor het op volgorde zetten van een rij met een willekeurig aantal getallen. Natuurlijk is het hierbij uitkijken geblazen: het speciale geval kan wel eens te speciaal zijn!
Een andere aanpak: **terugwerken vanuit de oplossing**. Hierbij ga je uit van de uitvoer die het programma moet produceren en probeer je van daaruit te bepalen welke opdrachten nodig zijn om deze uitvoer te produceren. Soms is het zelfs mogelijk naar de oplossing te raden en van daaruit verder te gaan. (Ook deze aanpak hebben we toegepast bij het bepalen van het maximum.) Bij een dergelijke werkwijze kan speciale aandacht aan de toestand van het programma worden besteed.
Er zijn nog meer algemeen toepasbare oplossingsstrategieën maar die leiden meestal tot zeer ingewikkelde algoritmen en die zullen we daarom hier niet behandelen.

5. GEGEVENS IN COMPUTERPROGRAMMA'S

Eerder is al genoemd dat de computer eigenlijk niets anders doet dan bepaalde gegevens omzetten in andere gewenste gegevens. Door zijn grote rekenkracht kan een computer dit zeer snel. Net als de mens maakt de computer gebruik van invoer van gegevens alvorens daarmee gewerkt kan worden. Computerprogramma's die geen invoer en uitvoer hebben, hebben natuurlijk maar een zeer beperkt toepassingsgebied. Een programma moet in staat zijn op een gemakkelijke manier informatie op te vragen en te verstrekken. De op te vragen informatie kan vooraf zijn klaargezet (b.v. in een bestand), of tijdens de werking van het programma, interactief dus, aan een gebruiker worden gevraagd.
Informatie kan tijdens de werking van het programma worden opgeslagen voor nadere bestudering (achteraf) of direct worden getoond. De verzameling van door het programma bijgehouden gegevens op zeker moment wordt ook wel de **toestand** van het programma genoemd.

Zo moet de computer - op zeker moment - precies weten waar een bepaald gegeven zich bevindt om het te kunnen gebruiken. De maker moet een naam en locatie geven aan een bepaalde geheugenplaats en dan weet de computer hoe die geheugenplaats bereikt kan worden. De computer kan dan vervolgens gegevens in die geheugenplaats zetten en de gegevens daarin zonodig wijzigen. Daarnaast is het ook gebruikelijk om aan te geven wat voor gegevens in die opslagplaats worden gezet bv. tekst of getallen. Zo'n geheugenplaats heet een **variabele** en het geven van een naam en het type aan zo'n variabele heet **declareren**. Hoe dit precies in zijn werk gaat, komt aan bod in de oefeningen.
Geef de variabele altijd een toepasselijke naam b.v. dezelfde als de naam van de grootheid die erin opgeslagen wordt (bv. tijd). In de meeste ontwikkelomgevingen mogen deze namen geen spaties bevatten en mogen ze niet met een cijfer beginnen. Natuurlijk hoef je je daar tijdens het ontwerp niet aan te conformeren.

Het type bepaalt welke waarden in de variabele mogen worden opgeslagen. Dit stelt de ontwikkelomgeving in staat tijdens het testen vast te stellen of een poging tot het opslaan van een ongeldige waarde wordt ondernomen. Daarnaast bepaalt het type welke bewerkingen op de variabele mogen worden uitgevoerd: twee teksten kunnen immers niet bij elkaar worden opgeteld, twee getallen wel! De ontwikkelomgeving kan dit al bij het vertalen - ook wel **compileren** genoemd - vaststellen. Tenslotte stelt het type de ontwikkelomgeving in staat te bepalen hoeveel geheugen nodig is.
Het is mogelijk - dat verschilt per taal - om niet te hoeven declareren. Dan zijn er verschillende aanpakken denkbaar. Het vertaalprogramma kan uitgaan van een vast type of uit de naam van de variabele afleiden van welk type de variabele is. Of het programma veronderstelt geen vast type en er wordt tijdens het draaien van het programma bepaald hoeveel geheugenruimte voor de opslag nodig is afhankelijk van de waarde die moet worden opgeslagen. Dan kan ook pas tijdens het draaien bepaald worden waar de waarde in het geheugen moet worden opgeslagen en moet dus naast de waarde ook steeds de plaats waar de variabele zich bevindt worden opgeslagen. Deze plaats kan weer wel op een vaste plaats worden bewaard. Zo'n variabele wordt een **wijzer (*pointer*)** genoemd. (Variabelen in Visual Basic kunnen niet van het type *pointer* worden gedeclareerd.) Variabelen die zich op een vaste plaats bevinden worden **statische variabelen** genoemd, anders worden het **dynamische variabelen** genoemd.

Zodra je een variabele hebt gedeclareerd kun je er waarden in opslaan: een variabele komt immers overeen met een stukje geheugengebied (een geheugendoosje). Het is echter gebruikelijker om niet te spreken van een waarde in een variabele opslaan maar van 'een variabele een waarde geven' of 'een waarde toekennen aan een variabele'. Ontwikkelomgevingen beschikken gewoonlijk over een aparte actie-opdracht om aan variabelen een waarde toe te kennen: de **toekenning** (***assignment***).
In de declaratie heb je de variabele een naam gegeven. In een toekenning moet je zowel de naam van de variabele opgeven als de waarde die die variabele moet krijgen. Wil je de variabele *bedrag* de waarde 2 geven dan kan dat in natuurlijke taal b.v. met 'geef bedrag de waarde 2' of 'maak bedrag gelijk aan 2' of desnoods 'ken aan bedrag de waarde 2 toe'. Natuurlijk kan ook een verkorte notatie worden gebruikt (b.v. 'bedrag=2'). Nadeel is dat de betekenis ervan dan niet altijd meer direct duidelijk is. 'Bedrag=2' zou dan dus wel steeds als 'geef bedrag de waarde 2' moeten worden uitgesproken (of een van de andere langere omschrijvingen). Het is-teken in de verkorte notatie van de toekenning wordt ook wel het **toekenningsteken** genoemd. (Niet in alle talen wordt eenzelfde toekenningsteken gebruikt!)
Variabelen zullen niet alleen gebruikt worden (als hierboven) om er constante waarden (b.v. -2, 3.14 of de tekst 'Hallo wereld') aan toe te kennen: het moet ook mogelijk zijn om er waarden aan toe te kennen die afhankelijk zijn van de toestand d.w.z. van de waarden van variabelen op dat moment.

Een eenvoudig voorbeeld is dat waarbij de waarde van een variabele moet worden opgehoogd. Zo zal elke keer als de gebruiker van een warme-dranken-automaat een muntje inwerpt het ontvangen bedrag met de waarde van dat muntje moeten worden opgehoogd. Wordt het bedrag in centen bijgehouden en werpt de gebruiker een stuiver in dan zal de variabele waarin het ontvangen bedrag wordt bijgehouden met 5 moeten worden opgehoogd. Dit zou kunnen worden omschreven als 'verhoog bedrag met 5'; in de vorm van een toekenning: 'maak bedrag gelijk aan bedrag plus 5' ofwel 'bedrag=bedrag+5'. Rechts van het is-teken staat de nieuwe waarde die bedrag moet krijgen als (eenvoudige) funktie van de waarde die bedrag op dat moment heeft (oftewel de 'oude' waarde van bedrag).

Aan een variabele mag elke waarde worden toegekend die in die variabele kan worden opgeslagen. Het type van de variabele - als in de declaratie vermeld - bepaalt welke waarden in de variabele kunnen worden opgeslagen (zie verder Hoofdstuk 8 Gegevens in Visual Basic).
Het kiezen van een type stelt het vertaalprogramma in staat te controleren of de waarde die aan een variabele wordt toegekend wel toegestaan is. Omdat dit helpt bij het voorkomen van fouten kan niet voldoende de noodzaak van het kiezen van het juiste type worden benadrukt!
Links van het toekenningsteken zal altijd een variabele moeten staan. Rechts van het toekenningsteken moet iets staan dat een waarde voorstelt. Dit kan een constante waarde zijn, b.v. 2 in 'bedrag=2', maar ook een formule, b.v. bedrag+5 in 'bedrag=bedrag+5'. In de praktijk wordt een dergelijke formule een **expressie** genoemd. Omdat een expressie ook een constante waarde kan zijn, moet datgene dat rechts van het toekenningsteken staat altijd een expressie zijn, waarvan de waarde bij uitrekening aan de variabele links van het toekenningsteken mag worden toegekend.
Overigens: in de praktijk spreekt men van **evaluatie** van een expressie i.p.v. over uitrekenen.

In een expressie kunnen **operatoren** b.v. rekenkundige als +, -, / en * voorkomen, maar ook **functies**. Functies zijn stukjes programma die één enkele waarde als resultaat retourneren. Er kan onderscheid gemaakt worden tussen functies die door

de ontwikkelomgeving beschikbaar worden gesteld, de zgn. **standaardfuncties**, b.v. goniometrische functies als cosinus en tangens, of abs() die de absolute waarde van een getal retourneert, en zelf-gemaakte functies b.v. voor de berekening van de afstand tussen twee lijnen. Alle gebruikte operatoren moeten expliciet worden weergegeven: de wortel $(-b+ b^2-4ac)/(2a)$ van de vierkantsvergelijking ax^2+bx+c moet worden geschreven als (-b+(b^2-4*a*c)^0.5)/(2*a), veronderstellende dat a ongelijk aan 0 is en b^2-4ac niet kleiner dan 0 is (en bovendien dat ^ het teken voor machtsverheffen is in de gebruikte ontwikkelomgeving).
Expressies komen niet alleen rechts van het toekenningsteken voor. Ze kunnen overal worden gebruikt waar een waarde mag worden gebruikt. Een bewering b.v. is hetzij waar, hetzij onwaar. Een bewering kan daarom als een expressie worden beschouwd met als waarde waar of onwaar. Een expressie waarvan de waarde hetzij waar, hetzij onwaar kan zijn, wordt een **logische** (of **boolese**) **expressie** genoemd. Rechts van het toekenningsteken kan ook een logische expressie staan, mits de variabele waaraan de waarde wordt toegekend de waarden waar en onwaar kan aannemen. Veel programmeertalen kennen het standaardtype **boolean** (naar Boole, de bedenker van de tweetallige logica), en variabelen van dit type kunnen alleen de waarden waar en onwaar hebben.

Functies hebben vaak ook invoer nodig: argumenten, in computerjargon: parameters. B.v. pi/4 in cos(pi/4). Wanneer een functie in een expressie voorkomt, wordt tijdens het evalueren van de expressie de functie uitgevoerd. Functie plus parameters in een expressie vormen een functie-aanroep. Een functie wordt uitgevoerd na aanroep. De parameters van een functie-aanroep worden ook wel **actuele parameters** genoemd: een functie kan immers op verschillende momenten met verschillende parameters worden aangeroepen.
Het zijn de expressies - en de functies die daarin voorkomen - die het (reken)werk doen in een programma. D.m.v. toekenningen kunnen dan die waarden worden opgeslagen die elders - in expressies - nodig zijn. Wanneer een waarde verder niet gebruikt wordt hoeft deze niet te worden opgeslagen en hoeft alleen berekend te worden. Zo zal de waarde van een bewering meestal alleen op dat moment nodig zijn en is het dus niet nodig deze vooraf te berekenen en in een variabele op te slaan. Vaak moet men een afweging maken tussen herberekenen en opslaan. Herberekenen kost (reken)tijd, opslaan kost geheugen. Voor welke oplossing gekozen wordt is afhankelijk van de omgeving waarin de applicatie moet draaien. Dit betekent echter wel dat al tijdens het ontwerpen met de uiteindelijke omgeving rekening moet worden gehouden. Dit is vervelend omdat het gewenst is het ontwerp zo universeel mogelijk te houden. Desalniettemin zal dan een keuze moeten worden gemaakt en moeten worden ingeschat wat het meest ongewenst is: lange rekentijd of geheugenbezetting.

Zo is het heel wel mogelijk een koffie-automaat elke keer na de ontvangst van een muntstuk het ontvangen bedrag te laten berekenen. Dit betekent echter wel dat onthouden moet worden welke muntstukken tot nu toe zijn ontvangen. In dit geval kost het steeds opnieuw berekenen ook nog eens meer geheugenruimte en is het wel erg gemakkelijk om de juiste keuze te maken: ophogen van het ontvangen bedrag na ontvangst van elk muntstuk. Bovendien hoeft dan de waarde van het muntstuk zelf niet te worden onthouden, want die is alleen van belang tijdens het ophogen van het ontvangen bedrag en anders niet. Zo is het heel wel denkbaar dat er een aparte functie is die de waarde van het ontvangen muntstuk als resultaat heeft. Laten we deze functie *OntvangenMunt* noemen. Deze functie kan worden aangeroepen in een expressie, omdat *OntvangenMunt* slechts één waarde als resultaat heeft. Het aanroepen van *OntvangenMunt* én het ophogen van bedrag kan dan in één toekenning plaatsvinden: 'bedrag= bedrag+OntvangenMunt()'. (Tussen de haakjes

moeten de actuele parameters van de aanroep staan, maar omdat *OntvangenMunt* geen invoer heeft staat er niets tussen.)
Wordt de waarde van *OntvangenMunt* dan nergens opgeslagen? Jawel, er is een soort van kladgeheugen, **stapel** of ***stack*** genoemd dat wordt gebruikt voor de opslag van tijdelijke informatie zoals actuele parameters van de aanroepen van functies maar ook van tussenresultaten van berekeningen in expressies. Na uitvoering van de functie of berekening van de expressie zijn eventuele parameters of tussenresultaten (anders dan het resultaat van de functie en de waarde van de expressie) niet meer nodig en kan de daardoor gebruikte geheugenruimte worden vrijgegeven. Zo wordt met een stapel op een eenvoudige wijze **hergebruik van geheugen** gerealiseerd.
Ook variabelen - die een vaste plaats hebben - kunnen hergebruikt worden, d.w.z. je kunt ze een andere functie geven. Het kiezen van een toepasselijke naam kan dan lastig zijn. Het is verwarrend om in een variabele met de naam *snelheid* een waarde op te slaan die een afstand voorstelt. Het wijzigen van de naam van een variabele tijdens de uitvoering van het programma is nu eenmaal niet mogelijk: op dat moment wordt niet meer met de naam van die variabele gewerkt maar met de plaats waar deze zich in het geheugen bevindt! De naam is dus slechts een label die er aan gegeven wordt ten behoeve van en tijdens het maken van het programma. Het vertaalprogramma zorgt voor de conversie naar geheugenplaatsen! (Het machinetaalprogramma werkt niet met variabele(name)n maar met locaties!)

De naam van een variabele moet dus met zorg worden gekozen. Natuurlijk is het mogelijk om een nietszeggende naam te kiezen maar dat beperkt weer de leesbaarheid van het programma. Vandaar dat het de voorkeur geniet variabelen alleen dan te hergebruiken als dat de leesbaarheid van het programma niet nadelig beïnvloedt.
Zo is het mogelijk om beide wortels van de vierkantsvergelijking achtereenvolgens in dezelfde variabele op te slaan. Noem die variabele dan niet grootstewortel of kleinstewortel maar gewoon wortel.

Samenvatting
In hogere programmeertalen kunnen stukjes geheugen - variabelen - worden benoemd, zodat het niet langer nodig is te weten welk stukje geheugen wordt gebruikt, zelfs niet of steeds hetzelfde stukje geheugen wordt gebruikt. Bij het ontwikkelen van een computerprogramma mag je zelf bepalen welke variabelen je nodig hebt. Aanmelding via een declaratie is niet alleen - bij de meeste ontwikkelomgevingen - noodzakelijk maar ook zeer gewenst. Ga na welke gegevens het programma nodig heeft om zijn functie naar behoren te kunnen vervullen d.w.z. definiëer - liefst vooraf - de toestand van het programma en bepaal daarna welke opdrachten nodig zijn om die toestand op de juiste wijze te wijzigen totdat de gewenste uitvoer van het programma deelverzameling van de toestand is.

6. OPDRACHTEN IN ALGORITMEN

Om in staat te zijn de gewenste uitvoer te produceren moet een computerprogramma waarden bewaren. De verzameling van waarden die door het programma wordt bijgehouden wordt de toestand van het programma genoemd. Deze verzameling hoeft niet op elk moment evengroot te zijn. Het is de taak van de ontwerper van het programma na te gaan welke waarden op zeker moment moeten worden onthouden om het programma op de juiste manier te laten werken. In hogere programmeertalen worden de geheugenplaatsen waarin bepaalde waarden moeten worden opgeslagen met een – bij voorkeur toepasselijke - naam aangegeven en kan die naam in de opdrachten worden gebruikt om de waarde aan te geven waarmee moet worden gewerkt. Omdat de waarde, opgeslagen in een of meerdere geheugenplaatsen, kan worden veranderd, worden deze stukjes geheugen variabelen genoemd. Het is aan de ontwerper te bedenken welke variabelen het programma nodig heeft en hoe deze een toepasselijke naam kan worden gegeven. Gewoonlijk zal hiervoor de naam van het soort gegeven of grootheid worden gebruikt waarvan de waarde in de variabele wordt opgeslagen (b.v. snelheid of verplaatsing).

> Wees niet te zuinig met het bedenken van variabelen. Heb je maar even het idee dat een bepaalde waarde nog elders in het programma nodig is, onthoud die waarde dan in een variabele!

Programmeertalen moeten mogelijkheden verschaffen om de waarden te kunnen veranderen. Per definitie zijn het de actie-opdrachten die hiertoe mogelijkheden verschaffen.
Daarnaast moeten programmeertalen – en dus ook programmeeromgevingen die op zo'n taal gebaseerd zijn – over opdrachten beschikken waarmee de besturing kan worden geregeld d.w.z. wanneer welke opdracht moet worden uitgevoerd. Zo is het zinvol te kunnen voorkomen dat een ongeldige bewerking wordt uitgevoerd b.v. een deling door nul. Hiertoe moet een programmeertaal beschikken over besturings-opdrachten.
De mogelijkheden van een programmeertaal vormen de bouwstenen voor de ontwerper. Verschillende programmeertalen hebben verschillende mogelijkheden, waardoor deze meer of minder geschikt zijn voor het maken van bepaalde toepassingen. Programmeertalen – met name die voor ons van belang zijn – verschillen echter steeds minder van elkaar. Programmeeromgevingen die bepaalde programmeertalen implementeren bieden extra faciliteiten aan die hen meer of minder geschikt maken voor bepaalde toepassingen. Er moet b.v. onderscheid worden gemaakt tussen programmeeromgevingen die geschikt zijn voor het – snel – maken van prototypes – gespecialiseerde prototyping tools - en programmeeromgevingen die geschikt zijn voor het maken van productieversies zgn. productie tools. De laatste zijn gewoonlijk moeilijker maar bieden meer mogelijkheden.
In dit hoofdstuk komt niet een bepaalde programmeertaal of –omgeving aan bod. Eerder wordt er ingegaan op algemene faciliteiten die elke programmeertaal/omgeving zal moeten bieden. Er wordt dan ook niet van een specifieke syntaxis gebruik gemaakt, maar van een pseudocode, d.i. een deelverzameling van de Nederlandse taal voor de beschrijving van de opdracht. Een dergelijke pseudocode maakt het mogelijk een ontwerpidee vorm te geven zonder dat daarbij de keuze voor een bepaalde programmeertaal/omgeving al vast ligt. Deze pseudocode lijkt daarbij voldoende op het Nederlands om communicatie erover gemakkelijk mogelijk te maken.

6.1 Actie-opdrachten

Dit zijn alle opdrachten die de programmatoestand wijzigen. Hier zullen we er alle opdrachten onder verstaan die de waarde van variabelen wijzigen.
Twee soorten kunnen worden onderscheiden: opdrachten die waarden inlezen d.w.z. aan een gebruiker vragen, en opdrachten die de waarde wijzigen in het programma zelf (zonder tussenkomst van een gebruiker). Een opdracht die een waarde inleest wordt een **lees**- of **invoeropdracht** genoemd.
De overige actie-opdrachten zijn onder te verdelen in opdrachten die waarden veranderen die niet direct door de gebruiker kunnen worden waargenomen en waarden die waarden veranderen die wel door de gebruiker zijn waar te nemen - en dus onderdeel van de user interface zijn. De eerste soort – die interne variabelen wijzigen – opdracht wordt een **toekenning** (***assignment***) genoemd. De tweede soort opdracht wordt een **uitvoeropdracht** genoemd. Een uitvoeropdracht naar het beeldscherm wijzigt immers de waarde van het beeldscherm, een geheugengebied dat gewoonlijk niet tot de toestand van het programma wordt gerekend omdat het beschikbaar is voor alle actieve programma's en mogelijkheden tot gebruik ervan door het besturingssysteem van de computer worden aangeboden.
Er is nog een andere soort actie-opdrachten, dit zijn de **aanroepen** (Engels: ***calls***). Het gaat dan niet om aanroepen van functies, want die retourneren een waarde en waar moet die gelaten worden?, maar van aparte stukjes programma, die **procedures** worden genoemd. Hieraan wordt in hoofdstuk 7 aandacht besteed.

6.2 Besturingsopdrachten

Besturingsopdrachten bepalen – aan de hand van de toestand – wanneer opdrachten moeten worden uitgevoerd (en wanneer niet). Ze bepalen welke opdrachten moeten worden uitgevoerd, wanneer en hoe vaak, kortom: de volgorde waarin de opdrachten moeten worden uitgevoerd.
Lagere programmeertalen kennen meestal maar een soort besturingsopdracht: de **sprongopdracht**. Een sprongopdracht vertelt het programma wat de volgende uit te voeren opdracht moet zijn. Een sprongopdracht is daarbij alleen nodig als de volgende uit te voeren opdracht niet de eerstvolgende opdracht is: de computer voert anders nl. automatisch de eerstvolgende opdracht - die op het eerstvolgende adres in het werkgeheugen staat - van het programma uit.
Sprongopdrachten zijn er in twee soorten: **onvoorwaardelijk** en **voorwaardelijk**. Bij onvoorwaardelijke sprongopdrachten wordt er altijd gesprongen, bij voorwaardelijke wordt er alleen gesprongen als de toestand aan een bepaalde voorwaarde voldoet.
Zonder sprongopdrachten kan een programma nooit veel voorstellen: er kan geen rekening gehouden worden met speciale situaties. Zo zal een programma niet kunnen voorkomen dat geprobeerd wordt een niet toegestane actie - als b.v. delen door nul - uit te voeren. En ook niet of wat ingevoerd wordt wel aan de eisen voldoet!
Zijn er naast de sprongopdracht nog andere besturingsopdrachten noodzakelijk? Aangezien lagere programmeertalen vaak geen andere besturingsopdrachten bevatten zal het antwoord wel 'nee' moeten zijn! Inderdaad, met sprongopdrachten alleen kan elke gewenste volgorde van uit te voeren opdrachten worden bepaald. Overigens: als machinetalen alleen van sprongopdrachten gebruik maken betekent dat, dat alle besturingsopdrachten in hogere programmeertalen vertaald moeten worden in sprongopdrachten. Dit is opvallend daar het expliciet gebruik van sprongopdrachten in een hogere programmeertaal ten strengste wordt afgeraden.
Welke andere besturingsopdrachten komen er dan zoal in hogere programmeertalen voor? Dit zijn de selectie en de herhaling.

6.2.1 De selectie

Een selectie-opdracht stelt het programma in staat te kiezen tussen uitvoeren van één van twee (groepen) opdrachten. Een selectie wordt ook wel een test genoemd. Welke van de twee opdrachten wordt uitgevoerd hangt af van een **bewering** die wel of niet waar is. Zo'n bewering wordt ook wel een **conditie** of **voorwaarde** genoemd.

6.2.1.1 Voorbeeld van een selectie

> **Als de noemer gelijk aan nul is, meld dan dat de deling niet kan worden uitgevoerd, en voer anders de deling uit.**

De bewering is: de noemer is gelijk aan 0. Als dit waar is, moet de opdracht meld dat de deling niet kan worden uitgevoerd worden uitgevoerd; als de bewering niet waar is, d.w.z. als de noemer niet gelijk aan nul is, dus ongelijk aan nul blijkt te zijn, dan moet de opdracht voer de deling uit worden uitgevoerd.
Natuurlijk is het ook mogelijk dat er in één van de twee gevallen niets moet worden gedaan. Dan is het gebruikelijk om de bewering zo te formuleren dat de opdrachten die eventueel wel moeten worden uitgevoerd worden uitgevoerd als de bewering waar is. Er is dan geen anders-gedeelte en spreekt men van een als-dan-selectie i.p.v. een als-dan-anders-selectie.

6.2.1.2 Voorbeeld van een als-dan-selectie

Als het eerste van twee getallen altijd kleiner moet zijn dan het andere dan kan dit m.b.v. een als-dan-selectie als volgt worden gerealiseerd:

> **Als het eerste getal groter is dan het tweede getal, verwissel dan de twee getallen.**

(Een als-dan-anders-selectie kan ook m.b.v. twee als-dan-selecties worden gerealiseerd (hoe?), maar dit is omslachtig en daarom ongebruikelijk!)
Het is mogelijk dat een opdracht moet worden uitgevoerd als meerdere beweringen gelijktijdig waar zijn of als tenminste een van een aantal beweringen waar is. In het eerste geval zal tussen de beweringen het woord 'en' staan, in het tweede geval het woord 'of'. Een bewering die uit meerdere beweringen bestaat wordt een **samengestelde bewering** genoemd.

6.2.1.3 Voorbeeld van samengestelde en-beweringen

Voor het bepalen van het grootste van een drietal getallen, kunnen de volgende als-dan-anders-opdrachten worden gebruikt:

> **Als het eerste getal groter is dan het tweede <u>en</u> derde getal, dan is het eerste getal de grootste, anders als het tweede getal groter is dan het eerste <u>en</u> derde getal, dan is het tweede getal het grootste, en in alle andere gevallen is het derde getal het grootste.**

(Overigens: dit kan ook overzichtelijker, zonder samengestelde bewering (hoe?))

6.2.1.4 Voorbeeld van een samengestelde of-bewering

Voor het bepalen van het produkt van twee getallen, kan de volgende als-dan-anders-opdracht worden gebruikt:

> **Als het eerste <u>of</u> het tweede getal gelijk aan 0, is het produkt gelijk aan 0, anders is het produkt gelijk aan het eerste getal maal het tweede getal.**

Hier is de selectie-opdracht niet per se noodzakelijk, omdat het produkt, zelfs als een van de twee getallen gelijk aan 0 is, toch wel goed berekend zou worden. Wel kan het programma sneller worden door de selectie te gebruiken, omdat de vermenigvuldiging

niet echt hoeft te worden uitgevoerd als een van de twee getallen nul is. (Wel neemt daardoor de omvang van het programma toe!)
De woorden 'en' en 'of' die in samengestelde beweringen worden gebruikt worden **logische** (of **boolese**) **operatoren** genoemd in tegenstelling tot de rekenkundige operatoren zoals + en -. Er is nog een derde logische operator nl. 'niet'. Deze maakt het mogelijk om opdrachten uit te laten voeren als een bepaalde bewering juist niet waar is.

6.2.1.5 Voorbeeld van een niet-bewering

Het produkt kan ook als volgt worden berekend:

> **Als het eerste en het tweede getal niet gelijk aan 0 zijn, dan is het produkt gelijk aan het eerste maal het tweede getal, en anders gelijk aan 0.**

Samengestelde en- en of-beweringen kunnen m.b.v. niet altijd in elkaar worden omgezet. Zo is 'a en b' identiek aan 'niet ((niet a) of (niet b))' en 'a of b' identiek aan 'niet ((niet a) en (niet b))' voor willekeurige beweringen a en b. Omdat 'niet (niet a)' gelijk is aan 'a', zal 'niet (a en b)' identiek zijn aan '(niet a) of (niet b)'. In een aantal gevallen kunnen deze **regels van de Morgan** van pas komen.
De groep opdrachten die moet worden uitgevoerd als een bewering waar is, is het dan-gedeelte. De groep die moet worden uitgevoerd als de bewering niet waar is, wordt het anders-gedeelte genoemd. Een selectie-opdracht in een als- of anders-gedeelte wordt een **geneste selectie-opdracht** genoemd.

6.2.1.6 Voorbeeld van een geneste selectie-opdracht

> **Als het eerste getal niet groter is dan het tweede getal, en als dan het tweede getal niet groter is dan het eerste getal, meld dan dat beide getallen gelijk zijn.**

Door welke samengestelde selectie is deze selectie te vervangen? Door welke niet-samengestelde?

6.2.2 De herhaling

Met een herhaling is het mogelijk een groep opdrachten meerdere malen uit te laten voeren. De herhalingsopdracht bestaat óf uit een opgave van het aantal keer dat de opdrachten moeten worden uitgevoerd óf uit de voorwaarde waaraan voldaan moet zijn opdat de opdrachten moeten worden uitgevoerd. Herhalingsopdracht en bijbehorende te herhalen opdrachten worden ook wel een **lus** genoemd. In de regel zijn de begrippen herhaling en lus synoniem.
De te herhalen opdrachten worden de **romp** (***body***) van de lus genoemd of ook wel de opdrachten in/binnen de lus.
Een herhalingsopdracht die vermeldt hoe vaak de opdrachten moeten worden uitgevoerd wordt een **onvoorwaardelijke herhaling** genoemd; wordt een voorwaarde vermeld, dan wordt van een **voorwaardelijke herhaling** gesproken.

6.2.2.1 Voorbeeld van een onvoorwaardelijke lus

Voor het bepalen van het grootste van een aantal, zeg 3, in te lezen getallen:

> **Doe drie keer: als het de eerste keer is, lees dan het grootste getal in; en lees anders het getal in. Als dit getal groter is dan het grootste getal (tot nu toe), dan is dit getal het grootste getal.**

Hoe had het gebruik van de selectie kunnen worden vermeden?

6.2.2.2 Voorbeeld van een voorwaardelijke lus

Voor het bepalen van het grootste van een aantal in te lezen positieve gehele getallen, waarbij de gebruiker met een niet-positief getal aangeeft dat alle getallen verwerkt zijn:

> **Doe totdat het ingelezen getal niet positief is: lees een getal in; als het de eerste keer is dan is het ingelezen getal het grootste getal; anders is het ingelezen getal het grootste getal als het groter is dan het grootste getal tot nu toe.**

Hoe had het gebruik van de selectie kunnen worden voorkomen?
Klopt de werkwijze ook als de gebruiker de eerste keer een niet-positief getal invoert? Waarom klopt deze werkwijze niet meer als het kleinste van een aantal positieve getallen moest worden bepaald? (Veronderstel wel dat grootste door kleinste en groter door kleiner is vervangen!)

In het voorbeeld hierboven ging het om een voorwaardelijke lus waarbij pas na afloop van uitvoering werd gekeken of de opdrachten nog een keer moeten worden uitgevoerd. Een dergelijke lus wordt een **post-test-lus** (***post test loop***), in tegenstelling tot een **pre-test-lus** (***pre test loop***) waarbij vooraf wordt getest of de voorwaarde waar is en het dus mogelijk is dat de groep opdrachten niet eens één keer wordt uitgevoerd. Omdat de opdrachten in een post-test-lus wel altijd tenminste een keer worden uitgevoerd, kun je aan de hand van het minimaal aantal keer dat de opdrachten in de lus uitgevoerd dienen te worden afleiden of je een post-test-lus of pre-test-lus moet maken.

Onvoorwaardelijke lussen kunnen gemakkelijk omgezet worden in voorwaardelijke lussen. De iteratie-opdracht 'doe 3 keer:' kan immers heel eenvoudig worden geschreven als 'doe zolang het aantal keer dat de opdrachten zijn uitgevoerd kleiner is dan 3' of 'doe totdat de opdrachten 3 keer zijn uitgevoerd'. Een dergelijke omzetting is natuurlijk altijd mogelijk, ongeacht het aantal keer dat de opdrachten moeten worden uitgevoerd. In feite worden onvoorwaardelijke opdrachten door het vertaalprogramma altijd omgezet in equivalente voorwaardelijke opdrachten. Waarom dan nog onderscheid maken? Nou, onvoorwaardelijke lussen komen zo vaak voor dat de meeste hogere programmeertalen er een aparte notatie voor gebruiken. Overigens: hogere programmeertalen implementeren die faciliteiten die in beschrijvingen van taken veel voorkomen. De mogelijkheden in hogere programmeertalen vormen dus de neerslag van de natuurlijke wijze van beschrijving van de uit te voeren taken. Alhoewel het niet zo evident is waarom je het aantal keer zou bijhouden dat de opdrachten in de lus uitgevoerd zijn, komt het in de praktijk zo vaak voor dat de opdrachten in de lus dienen te weten hoe vaak ze al zijn uitgevoerd, dat hogere programmeertalen je gewoonlijk verplichten dit aantal of een daarmee verband houdend geheel getal bij te houden. Binnen de lus kan dat getal dan worden gebruikt om verschillen tussen uitvoeringen van de opdrachten te realiseren (zie de uitwerkingen in de volgende paragraaf).

6.3 Pseudocode

Voor de beschrijving van algoritmen zullen we gebruikmaken van een bepaalde pseudocode. Deze biedt een functionaliteit zoals die bij hogere programmeertalen gebruikelijk is.

De **als-dan-anders**-opdracht is beschikbaar als algemene **selectie**-opdracht.
De **herhaal-totdat** of **zolang**-opdracht wordt gebruikt voor **voorwaardelijke**, de **doe-voor**-opdracht voor **onvoorwaardelijke** lussen. Omdat in deze pseudocode de opdrachten in het dan- of anders-gedeelte van de selectie en in de zolang-lus niet worden afgesloten, wordt door het inspringen van alle die opdrachten aangegeven welke opdrachten tot welke besturingsstructuur behoren (zie uitwerkingen). Voor de conformiteit zullen we steeds inspringen, ook als is dan niet per se noodzakelijk.

De **toekenning** (maak {variabele} gelijk aan {formule}) is er om variabelen, waarin één enkele waarde staat, een waarde te geven. Wanneer een ander algoritme moet worden uitgevoerd, kan dat door voor de naam van de uit te voeren algoritme en eventuele actuele parameters het woord voer te zetten, en het woord uit erachter.

Daarnaast zullen we ook over een standaard invoer- en uitvoerfunctie moeten beschikken.
De meest elementaire invoerfunctie is het opvragen van de waarde van een variabele; de variabele waarvan de waarde moet worden ingelezen moet als uitvoerparameter in de aanroep van deze functie, die ik hier **lees** zal noemen, worden vermeld.
De meest elementaire uitvoeractie is het tonen van de een of andere waarde; de waarde die moet worden getoond moet als invoerparameter in de aanroep van deze functie, die ik **schrijf** zal noemen, worden vermeld. Omdat vaak meerdere waarden achtereenvolgens moeten worden getoond, is het handig af te spreken dat meerdere waarden gescheiden door een komma kunnen worden getoond met één schrijf-opdracht: 'schrijf a,b' is dus equivalent met 'schrijf a' en 'schrijf b'. Eenzelfde afspraak maken we voor voor het inlezen: 'lees a,b' is equivalent met 'lees a' gevolgd door 'lees b'.

Onze pseudocode beschikt over alle genoemde selectie- en herhalingsvarianten voor de bepaling van de volgorde waarin opdrachten worden uitgevoerd, over de toekenning voor het onthouden van een waarde (in een variabele), over een functie lees voor het vragen van de waarde van een variabele aan de gebruiker, over een functie schrijf voor het tonen van een waarde aan de gebruiker, en over een mechanisme voor het uitvoeren van andere (in pseudocode uitgedrukte) algoritmen. Daarnaast moet bij elk algoritme worden vermeld wat de parameters en wat de lokale variabelen. Van elke parameter moet worden aangegeven of het een waarde- of variabele-parameter is. Bovendien moeten in de aanroep de actuele parameters in een volgorde staan die overeenkomt met die in de lijst van formele parameters.

Algoritmen uitgedrukt in deze pseudocode kunnen gemakkelijk in een gangbare hogere programmeertaal, waaronder Visual Basic, worden omgezet.

Hieronder wordt voor elk van de tot nu besproken voorbeelden aangegeven hoe een algoritme kan worden uitgedrukt in onze pseudocode. Het herkennen van de benodigde variabelen is daarbij van cruciaal belang. Vandaar dat daaraan in het bijzonder aandacht zal worden besteed. Het kiezen van het type van de variabelen echter hoeft hier echter nog niet plaats te vinden, omdat een geschikte keuze samenhangt met de door de ontwikkelomgeving beschikbaar gestelde standaardtypen en de mogelijkheden daarvan.

6.3.1 Voorbeelden selectie

6.3.1.1 Voorbeeld van een selectie (vervolg)

Zelfstandige naamwoorden die in dit algoritme voorkomen zijn 'noemer', 'nul' en 'deling'. Deze komen in aanmerking om in variabelen te worden opgeslagen. De noemer is een (onbekende) getalwaarde die in een variabele genaamd noemer opgeslagen zou kunnen zijn. 'Nul' is weliswaar een waarde maar omdat altijd dezelfde waarde wordt gebruikt hoeft deze niet in een aparte variabele worden gebruikt. Aangezien in een programma ook gewoon (constante) waarden mogen worden gebruikt hoeft 'nul' niet in een variabele te staan. Dan 'deling'. Het gaat hier om het wel of niet uitvoeren van de deling d.w.z. de een of andere teller delen door de noemer. De teller kan een variabele zijn, maar dat hoeft niet. Om daadwerkelijk de deling uit te kunnen voeren moeten we weten wat de teller is: een variabele, een waarde of een formule. Veronderstel nu even dat er een variabele genaamd teller is waarin de teller staat opgeslagen. Dan komt het uitvoeren van de deling neer op het berekenen van teller/noemer. De waarde van deze berekening kan b.v. aan een variabele worden toegekend of worden getest of getoond. Wanneer de waarde van de deling alleen maar hoeft te worden getoond is geen variabele nodig om de waarde in op te staan. Ons algoritme wordt dan:

```
als noemer=0, dan
        schrijf "Delen kan niet, want de noemer is gelijk aan 0."
anders
        schrijf teller," gedeeld door ",noemer," is: ",teller/noemer,"."
```

In de bewering NOEMER=0 is het =-teken geen toekenningsteken maar de is-gelijk-aan-operator. Als kleiner-dan-operator wordt gewoonlijk < gebruikt, als groter-dan-operator >, als ongelijk-aan-operator <>, als kleiner-dan-of-gelijk-aan-operator <= en als groter-dan-of-gelijk-aan-operator >=.
Tekst die letterlijk moet worden weergegeven wordt hier - net als in Visual Basic - tussen dubbele aanhalingstekens geplaatst. Dit moet omdat de vertaler in staat moet zijn onderscheid te maken tussen variabelen en (letterlijke) tekst.

6.3.1.2 Voorbeeld van een als-dan-selectie (vervolg)

Er is sprake van twee getallen waarvan het ene kleiner moet zijn dan het andere (na uitvoering van de te bedenken opdrachten). Verwisselen houdt in dat daar waar in het geheugen het ene getal staat het andere komt te staan en omgekeerd. Dat is hetzelfde als ervoor zorgen dat de ene variabele de waarde van de andere krijgt en omgekeerd. Noem de ene variabele b.v. **ene** noemen en de andere variabele **andere**. Het verwisselen kan niet in één opdracht in de pseudocode worden gedaan. Met de opdrachten **maak ene gelijk aan andere** en **maak andere gelijk aan ene** kan ook niet het gewenste effect worden bereikt: met de eerste toekenning kan **ene** gelijk gemaakt worden aan de waarde van **andere**. Daarna hebben **ene** en **andere** dezelfde waarde; het effect van uitvoeren daarna van de tweede toekenning is dan ook nihil: **andere** en **ene** hebben immers al dezelfde waarde. Er moet een manier worden bedacht om de oorspronkelijke waarde van **ene** even ergens op te slaan, vervolgens ene gelijk aan andere te maken en andere gelijk aan de waarde die ene oorspronkelijk had. Daarvoor is een extra variabele nodig waarin we de oorspronkelijke waarde van **ene** kunnen opslaan (en dus van hetzelfde type). Noem deze b.v. **hulp**. Het algoritme kan dan worden geschreven als:

```
als ene>andere, dan
        maak hulp gelijk aan ene
        maak ene gelijk aan andere
        maak andere gelijk aan hulp
```

6.3.1.3 Voorbeeld van samengestelde en-beweringen (vervolg)

Hier is sprake van drie getallen waarvan de grootste moet worden bepaald. De drie variabelen waarin de drie getallen opgeslagen zijn noem ik hier **een**, **twee** en **drie**. De voorwaarden zullen wel niet zoveel problemen geven ook al kan b.v. de eerste voorwaarde niet als een>twee en >drie worden geschreven, maar wel als **een>twee en een>drie**. Hetzelfde geldt voor de andere voorwaarden.
In het algoritme wordt beweerd dat de eerste de grootste is als aan de eerste voorwaarde voldaan wordt. Aan die bewering hebben we op zich niets. We moeten dit resultaat op de een of andere manier registreren of - als het niet onthouden hoeft te worden - aan de gebruiker kenbaar maken.
Registreren is onthouden d.w.z. een bepaalde variabele een zodanige waarde te geven, dat (later) onderscheid kan worden gemaakt tussen de verschillende mogelijke resultaten. Hier zul je willen onthouden welke van de drie variabelen de grootste is of de grootste waarde zelf. Het hangt van de context af wat nodig is (misschien wel allebei)! Wil je (later) weten welke van de drie de grootste is, dan kun je b.v. in een variabele hetzij de waarde 1, 2 of 3 op te slaan: 1 als **een** de grootste is, 2 als **twee** de grootste is en 3 als **drie** de grootste is. Als het er niet om gaat welke de grootste is maar wat de grootste is moet de grootste waarde worden onthouden, b.v. in de variabele **grootste**. In dat geval krijgen we de volgende pseudocode:

```
als een>twee en een>drie, dan
        maak grootste gelijk aan een
anders
        als twee>een en twee>drie, dan
                maak grootste gelijk aan twee
        anders
                maak grootste gelijk aan drie
```

Als zowel had moeten onthouden welke van de drie variabelen de grootste is en wat de grootste waarde is dan zijn natuurlijk twee variabelen nodig (met verschillende namen)! Als je onthouden had, b.v. in een variabele **nummergrootste**, welke van de drie de grootste was, dan moet je opnieuw testen om te bepalen welke de grootste is:

```
als nummergrootste=1, dan
        maak grootste geljk aan een
anders
        als nummergrootste=2, dan
                maak grootste gelijk aan twee
        anders
                maak grootste gelijk aan drie
```

(Als de drie getallen in een rij waren opgeslagen, b.v. als **getallen(1)**, **getallen(2)** en **getallen(3)**, dan zou dit algoritme kunnen worden vervangen door één opdracht: **maak grootste gelijk aan getallen(nummergrootste)**, een veel korter algoritme.)

6.3.1.4 Voorbeeld van een samengestelde of-bewering (vervolg)

Er is sprake van twee getallen waarvan het produkt moet worden uitgerekend. Zowel de twee getallen als het produkt kunnen in variabelen opgeslagen worden b.v. resp in **getal1**, **getal2** en **produkt**:

```
als getal1=0 of getal2=0, dan
        maak produkt gelijk aan 0
anders
        maak produkt gelijk aan getal1 maal getal2
```

6.3.1.5 Voorbeeld van een niet-bewering (vervolg)

Het algoritme in pseudocode uitgedrukt met dezelfde variabelen als daarnet:

```
als niet getal1=0 en niet getal2=0, dan
        maak produkt gelijk aan getal1 maal getal2
anders
        maak produkt gelijk aan 0
```

De voorwaarde kan natuurlijk ook genoteerd worden als: **getal1<>0 en getal2<>0**. Maar waarom staat er geen **niet getal1=0 en getal2=0**? Dat heeft met prioriteiten te maken! Om te bepalen in welke volgorde operatoren moeten worden toegepast heeft elke operator een bepaalde prioriteit. Zo bepaalt Meneer Van Dalen wacht Op Antwoord dat Machtsverheffen de hoogste prioriteit heeft, dan Vermenigvuldigen, Delen, Optellen en Aftrekken. (Dit ezelsbruggetje vertelt ons echter niet dat Vermenigvuldigen en Delen dezelfde prioriteit hebben.)
De prioriteit van de logische operatoren verschilt per vertaalprogramma (misschien dat er daarom geen ezelsbruggetje voor bestaat). Bij Visual Basic hebben de vergelijkingsoperatoren (=,<,<>,>,<= en >=) de hoogste prioriteit, dan niet, dan en en dan of. **Niet getal1=0 en getal2=0** is dus feitelijk **Niet (getal1=0) en getal2=0** en dus equivalent met **getal1<>0 en getal2=0**, terwijl we **getal1<>0 en getal2<>0** willen. Wat wel goed lijkt maar het niet is, is **Niet (getal1=0 en getal2=0)**! Immers, dit is ook waar als slechts een van de twee getallen gelijk aan 0 is, maar ook dan moet het produkt direct gelijk aan 0 worden gemaakt. Wel goed is: **niet (getal1=0 of getal2=0)**! In het algoritme heb ik zo voorkomen dat er haakjes zouden moeten worden gezet!

6.3.1.6 Voorbeeld van geneste selectie-opdrachten (vervolg)

Noem de variabelen waarin de twee getallen staan opgeslagen **getal1** resp. **getal2**. De pseudocode:

```
als niet getal1<getal2, dan
        als niet getal1>getal2, dan
                schrijf "de getallen zijn gelijk"
```

De voorwaarden kunnen worden vervangen door één voorwaarde (en de selecties dus door één selectie): **niet getal1<getal2 en niet getal1>getal2**, of ook: **getal1>=getal2 en getal1<=getal2**.
Het is gebruikelijk om de opdrachten binnen een besturingsopdracht in te laten springen: je kunt dan gemakkelijk zien welke opdrachten onder welke voorwaarden uitgevoerd moeten (blijven) worden.

6.3.2 Voorbeelden lussen

6.3.2.1 Voorbeeld van een onvoorwaardelijke lus (vervolg)

We zouden net als voorheen drie verschillende variabelen kunnen gebruiken. Dit heeft echter als nadeel dat het van het aantal keer dat ingelezen is afhangt welk getal moet worden ingelezen. Gemakkelijker is het om steeds dezelfde variabele in te laten lezen; natuurlijk is dit alleen mogelijk als de ingelezen waarde niet hoeft te worden onthouden: als de volgende keer wordt ingelezen, dan wordt de huidige waarde overschreven! Laten we deze variabele maar gewoon **getal** noemen.
Ook hier besluit ik de grootste waarde op te slaan in de variabele **grootste**.

Het algoritme is zo opgesteld, dat het noodzakelijk is te moeten kunnen vaststellen of het om het eerste in te lezen getal gaat. Omdat er alleen maar onderscheid tussen de eerste en overige keren gemaakt hoeft te worden is het hiervoor niet per se nodig om te onthouden de hoeveelste keer het betreft: we kunnen een variabele, zeg **eerstekeer**, gebruiken die we vooraf gelijk aan **true** maken. In de lus testen we dan of **eerstekeer** gelijk aan **true** is. Zo ja, dan maken we **eerstekeer** gelijk aan **false**, zodat alle volgende keren **eerstekeer** (nog) steeds gelijk aan **false** zal zijn. De opdrachten in de lus moeten echjter precies drie keer worden uitgevoerd en het zal noodzakelijk zijn om bij te houden hoe vaak de opdrachten in de lus al zijn uitgevoerd; daarvoor is dan wel een teller nodig. Laten we deze **teller** noemen. En we laten **teller** lopen van 1 tot (en met) 3. Testen of het de eerste keer is, is dan hetzelfde als testen of **teller** gelijk aan 1 is: het is dan dus niet nodig om een aparte variabele als **eerstekeer** te gebruiken.

Het algoritme wordt dan:

```
doe voor teller=1 tot 3 ' d.w.z. voor teller gelijk aan 1, 2 en 3
    lees getal
    als teller=1, dan
        maak grootste gelijk aan getal
    anders
        als getal>grootste, dan
            maak grootste gelijk aan getal
```

Wie goed naar de selecties kijkt, zie dat in beide gevallen dezelfde opdracht moet worden uitgevoerd nl. **maak grootste gelijk aan getal**. Is er nou niet een manier om de voorwaarden te combineren? Ja, die is er! **Grootste** moet gelijk aan getal worden gemaakt als **teller** gelijk aan 1 is, maar ook als **getal>grootste**. Het lijkt alsof de samengestelde voorwaarde **teller=1 en getal>grootste** moet zijn, maar dan is niet zo! De voorwaarde die we moeten maken moet waar zijn als aan minstens één van de twee voorwaarden voldaan is; dit is het geval als van of gebruik gemaakt wordt. (Een en-samenstelling is alleen waar als beide voorwaarden waar zijn!) De juiste voorwaarde is dus: **teller=1 of getal>grootste**. Herschrijf zelf het algoritme!

In bovenstaand algoritme hebben we net gedaan alsof teller 'automatisch' elke keer na het uitvoeren van de opdrachten in de lus met 1 wordt opgehoogd en gekeken werd of deze nog niet groter dan 3 was. Zoals al eerder gezegd, programmeertalen bevatten aparte mogelijkheden voor het op deze gemakkelijke manier formuleren van een onvoorwaardelijke lus. Het vertaalprogramma zorgt voor de vertaling in een equivalente voorwaardelijke lus. Het ophogen en vergelijken gebeurt dus wel, maar dit hoeft niet expliciet te worden opgegeven. Een equivalent algoritme met een onvoorwaardelijke lus:

```
maak teller gelijk aan 1
zolang teller<=3
        lees getal
        als teller=1 of grootste>getal, dan
                maak grootste gelijk aan getal
        maak teller gelijk aan teller+1
```

6.3.2.2 Voorbeeld van een voorwaardelijke lus (vervolg)

Het verschil met de onvoorwaardelijke lus is, dat we vooraf niet weten hoe vaak de opdrachten in de lus moeten worden uitgevoerd; de opdrachten in de lus hebben daarop invloed!
Als we besluiten het aantal keer niet te tellen dat er een getal is ingelezen dan hebben we toch een aparte variabele waarin wordt bijgehouden of het wel of niet de eerste keer is. (Nu kunnen we **eerstekeer** toch eens gebruiken!)
Een mogelijk pseudocode:

```
Maak eerstekeer gelijk aan true
herhaal
        lees getal
        als eerstekeer of getal>grootste, dan
                maak grootste gelijk aan getal
                maak eerstekeer gelijk aan false
totdat getal<=0
```

Het is gebruikelijk om bij een herhaal-totdat-lus de opdrachten tussen herhaal en totdat te zetten, terwijl bij een zolang-lus de opdrachten na zolang staan; dit opdat het duidelijk is dat bij een herhaal-totdat-lus de voorwaarde pas wordt berekend na uitvoering van de opdrachten i.p.v. ervoor! Het is dan duidelijk dat de opdrachten altijd tenminste een keer worden uitgevoerd.
Door gebruik te maken van de samengestelde selectie wordt eerstekeer ook weer opnieuw gelijk aan false gemaakt als een nieuw grootste getal wordt ingelezen, terwijl het eigenlijk alleen maar de eerste keer hoeft: het kan dus vaker gebeuren dan eigenlijk nodig is. De samengestelde selectie maakt echter de code dusdanig eenvoudiger, dat we dat eventueel teveel uitvoeren van **maak eerstekeer gelijk aan false** op de koop toenemen.
Als de gebruiker de eerste keer een niet-positief getal invoert, zal grootste gelijk worden aan dit niet-positieve getal! Dat heeft dan als voordeel, dat achteraf gemakkelijk na te gaan is of er wel getallen zijn ingevoerd: als grootste niet-positief is zijn geen getallen ingevoerd en is er dus ook geen grootste getal! Er zou dus achter kunnen staan:

```
als grootste<=0, dan
        schrijf "Er is geen grootste getal: geen getallen ingelezen."
anders
        schrijf "Het grootste getal is: ",grootste,"."
```

7. ORGANISATIE IN COMPUTERPROGRAMMA'S

Het computerprogramma met de meest eenvoudige organisatie bestaat uit een stel opdrachten die allemaal toegang hebben tot alle in het programma gebruikte variabelen. Het programma bevat dan één blok waarin de variabelen worden gedeclareerd en één blok waarin de opdrachten staan en komt daardoor overeen met één enkel algoritme. Als het programma wordt gestart, wordt voor de variabelen geheugenruimte gereserveerd, en deze geheugenruimte blijft voor de hele levensduur van het programma beschikbaar voor de opslag van de waarden van die variabelen. Een dergelijke eenvoudige organisatie is echter te beperkt om ontwerpen die gewoonlijk meerdere abstractieniveau's beslaan – b.v. ontstaan door de toepassing van stapsgewijze verfijning - gemakkelijk in uit te kunnen drukken.
In de loop der jaren zijn een aantal organisatie-principes toegepast. Hier bespreek de organisatie in subroutines – de basis voor programma's gebaseerd op het procedurele programmeerparadigma – en organisatie in klassen – de basis voor object-georiënteerde programma's.
Hier bespreek ik eerst de organisatie van de opdrachten in subroutines, omdat het begrip subroutine in alle paradigma's voorkomt – niet per se altijd onder dezelfde naam – en de basisfaciliteit is voor het opdelen van het computerprogramma in afzonderlijke onderdelen.

7.1 Organisatie in subroutines

Ontwikkelomgevingen stellen gebruikers ervan in staat een aantal opdrachten te bundelen in een procedure (of functie). Een dergelijke bij elkaar behorende verzameling opdrachten met een bepaalde functionaliteit - of het nou een functie of een procedure is - wordt een **subroutine** genoemd.
Er zijn verschillende redenen om opdrachten in een subroutine samen te bundelen. Een belangrijke reden is hergebruik: wanneer in een programma bepaalde opdrachten vaker dan eens moeten worden uitgevoerd, dan is het zinvol deze in een procedure onder te brengen. Hierdoor neemt de totale hoeveelheid code af, want de opdrachten hoeven maar een keer, nl. in de declaratie van de procedure worden opgenomen. Hier wordt ook van een declaratie gesproken, omdat het vermelden van de opdrachten nog niet betekent dat ze worden uitgevoerd: ze worden pas uitgevoerd als de subroutine wordt aangeroepen. Teneinde aanroepen van een subroutine mogelijk te maken moet de declaratie - naast de opdrachten zelf - minstens de naam van de subroutine bevatten; deze kan dan worden aangeroepen in een call door vermelding van de naam van de uit te voeren subroutine.
Een subroutine kan beschouwd worden als een programma binnen een programma - een systeem binnen een systeem. Hij kan zijn eigen toestand bezitten, heeft zijn eigen in- en uitvoer. In- en uitvoer van een subroutine worden **parameters** genoemd. De variabelen die alleen binnen de subroutine gebruikt kunnen worden – en dus niet door andere subroutines – worden **lokale variabelen** genoemd.
Subroutines zijn er in twee soorten: functies en procedures. **Functies** worden altijd alleen in formules gebruikt, omdat ze een waarde – het **resultaat** van de functie - retourneren. **Procedures** retourneren geen antwoord en kunnen derhalve niet in formules worden gebruikt, maar moeten als afzonderlijke opdrachten worden opgegeven.
Lees voor een nadere toelichting de volgende drie paragrafen.

BOX 7.1 In- en uitvoer in Visual Basic

Voor het aan de gebruiker vragen van de waarde van een variabele kent Visual Basic de functie **InputBox$**. Deze retourneert altijd de tekst die de gebruiker heeft ingetypt:

```
strNaam = InputBox$("Uw naam s.v.p.")
```

Inputbox$ maakt gebruik van een modaal venster voor het opvragen van de tekst. Modaal wil zeggen dat zolang het venster zichtbaar is geen andere vensters geactiveerd kunnen worden. Dit komt erop neer dat het programma pas verder gaat met de volgende opdracht als het modale venster door de gebruiker is gesloten.
Voor het tonen van de waarde van een variabele maakt Visual Basic gebruik van de procedure **MsgBox**. Aan MsgBox moet (minstens) de weer te geven tekst worden doorgegeven b.v.

```
MsgBox "De door U opgegeven naam was " & strNaam
```

Ook MsgBox gebruikt hiervoor een modaal venster. Zie voor de online Help bij VB.
InputBox en MsgBox zijn asynchroon: het programma moet wachten op de gebruiker. Deze zijn dus met name geschikt voor door het programma gecontroleerde in- en uitvoer, d.w.z. op het moment dat het programma dat wil. Daarnaast is in Visual Basic ook door de gebruiker bepaalde in- en uitvoer mogelijk via op vensters aangebrachte controls als text boxen (invoer van tekst) en labels (uitvoer van tekst). Bij invoer via een text box kan de gebruiker nog andere taken in het programma tijdens het invullen van de tekst uitvoeren; dit geeft een gebruiker meer vrijheid te bepalen wanneer hij wat doet: de volgorde van invoer staat niet bij voorbaar vast en moet door de ontwerper bewust worden geregisseerd.

7.1.1 Hoe een subroutine invoer betrekt

Een subroutine heeft toegang tot alle variabelen die in het programma buiten de subroutines gedeclareerd zijn, d.w.z. kan in elke subroutine worden gebruikt en gewijzigd. Dergelijke variabelen worden **globale variabelen** genoemd.
Dit heeft echter een groot nadeel. Als een subroutine rechtstreeks van deze variabele(name)n gebruik maakt kan deze subroutine niet gemakkelijk in een ander computerprogramma worden gebruikt omdat daarin dezelfde variabele(name)n moeten voorkomen met dezelfde betekenis. Een dergelijke subroutine is daardoor niet gemakkelijk her te gebruiken, een aanzienlijke beperking vooral als het om commercieel ontwikkelde computerprogramma's gaat waar verregaand hergebruik tot een aanzienlijke kostenreductie kan leiden.
Bovendien is een dergelijke subroutine niet gemakkelijk aan te passen, omdat deze op soms onoverzichtelijke wijze gebruik maakt van de toestand van het programma. Het verdient daarom de voorkeur een subroutine niet direct gebruik te laten maken globale variabelen. Maar hoe komt de subroutine dan aan zijn invoer? Via **parameters**! Een cosinusfunctie - beschikbaar gesteld door een ontwikkelomgeving - kan immers in alle ermee gemaakte applicaties zonder probleem worden gebruikt omdat een invoerwaarde als argument – in vaktermen als actuele parameter – wordt opgegeven b.v. cosinus(2). Een ander bijkomend voordeel van deze aanpak is, dat de subroutine dan met verschillende invoer kan werken. Het is niet langer alleen maar een manier om minder code nodig te hebben (in dat geval zou de subroutine geen parameters nodig hebben), het wordt dan mogelijk de werking ervan te laten bepalen door de opgegeven parameters. Dat is pas echt hergebruik. Dit betekent echter wel dat de programma-ontwikkelaar in staat moet zijn te ontdekken wat opdrachten voor gemeenschappelijke structuur hebben; het hoeven niet langer meer precies dezelfde opdrachten te zijn die zinvol in één subroutine kunnen worden ondergebracht.

7.1.2 Hoe een subroutine uitvoer afgeeft

Net zoals een subroutine parameters gebruikt om invoer uit te betrekken, zo kan het van parameters gebruik maken om uitvoer af te geven. Er is echter een duidelijk verschil tussen invoer- en uitvoerparameters. Invoerparameters leveren waarden aan een subroutine, terwijl uitvoerparameters waarden moeten kunnen krijgen. En het enige in een computerprogramma dat een waarde kan krijgen is een variabele! Derhalve moeten de uitvoerparameters van een subroutine altijd variabelen zijn. Dit betekent dat de actuele parameter behorende bij een uitvoerparameter een variabele moet zijn, het mag dus nooit een waarde zijn (zoals die uit een formule komt)! Invoerparameters worden ook wel **waardeparameters** genoemd, omdat de actuele parameter als waarde wordt beschouwd (en dus wel een formule mag zijn); uitvoerparameters worden **variabele-parameters** genoemd, omdat de actuele parameter een variabele moet zijn. In de declaratie van de subroutine moeten niet alleen de parameters worden opgegeven, maar ook van elke parameter of het om een waarde- of variabele-parameter gaat.
Een subroutine kan ook één waarde als resultaat afgeven. In dat geval wordt de subroutine een **functie** genoemd; een subroutine zonder resultaat wordt een **procedure** genoemd. Omdat een resultaatwaarde wordt afgegeven mag de aanroep van een functie alleen in een formule voorkomen. Het uitrekenen - evalueren in vaktermen - van de formule betekent het uitvoeren van de functie en het resultaat is de in het evalueren te gebruiken waarde.
Natuurlijk kan een subroutine ook gobale variabelen direct wijzigen maar dit is - zoals al eerder beargumenteerd - minder wenselijk want dit leidt tot verminderde herbruikbaarheid en aanpasbaarheid.

7.1.3 Hoe een subroutine zijn eigen toestand bijhoudt

Een subroutine heeft ook een eigen toestand. Dit zijn de waarden van de bij/in/binnen de subroutine gedeclareerde variabelen, de zgn. **lokale variabelen**. Deze kunnen alleen door de subroutine gebruikt worden, niet door andere subroutines. Het verschil met parameters van de subroutine is, dat bij aanvang van uitvoering van de opdrachten in de subroutine lokale variabelen nog geen waarde hebben, terwijl de waardeparameters al de waarde hebben gekregen die in de aanroep staat vermeld, en variabeleparameters de waarden die er elders in het programma aan zijn gegeven! Een lokale variabele mag dus pas worden gebruikt in een formule als zij een waarde heeft gekregen.

> Omdat dergelijke lokale variabelen alleen tijdens het uitvoeren van de subroutine gebruikt worden, kunnen ze na uitvoering van de subroutine weer worden verwijderd d.w.z. de geheugenplaatsen die ze bezetten, kunnen weer worden vrijgegeven voor gebruik door andere variabelen. Gewoonlijk gebeurt dit m.b.v. de eerder genoemde ***stack***, waarop tijdelijke variabelen worden opgeslagen.

De totale toestand van een programma, op zeker moment, bestaat niet alleen uit de waarden van de globale variabelen, maar ook uit de waarden van alle lokale variabelen van subroutines die op dat moment worden uitgevoerd. Dit lijkt het geheel onoverzichtelijker te maken, maar het tegendeel is waar. Immers: bij de controle van de werking van een aparte subroutine hoeft alleen rekening te worden gehouden met de lokale toestand - de waarden van lokale variabelen en parameters - die gewoonlijk veel kleiner is dan de totale toestand. Wanneer het contact met de omgeving uitsluitend via parameters plaatsvindt hebben we een subroutine waarvan de werking veel gemakkelijker te testen is dan wannneer de interface met de omgeving onduidelijk is.

7.1.4 Hoe een subroutine gedeclareerd wordt

In computerprogramma's moet het mogelijk zijn om een subroutine af afzonderlijke eenheid op te geven, te declareren in vaktermen. Om de opdrachten in de subroutine te kunnen laten uitvoeren moet de subroutine een naam krijgen en moet worden opgegeven of het om een functie of procedure gaat. Daarnaast moet van elke in de subroutine gebruikte variabele worden opgegeven of het om een (waarde- of variabele-) parameter, lokale of globale variabele gaat. Elke variabele moet een – natuurlijk toepasselijke - naam worden gegeven, teneinde in de opdrachten te kunnen aanduiden. De parameters worden formele parameters genoemd, omdat tijdens de declaratie van de subroutine nog niet bekend is welke actuele parameters bij uitvoering van de subroutine zullen worden gebruikt. In de opdrachten van de subroutine – de **romp** (***body***) in vaktermen - gebruik je de naam van de formele parameter uit de parameterlijst als was het een lokale variabele. Een waardeparameter is vergelijkbaar met een lokale variabele, met dit verschil dat een waardeparameter wel al een waarde heeft – nl. die van de bijbehorende actuele parameter – vóórdat de opdrachten in de romp worden uitgevoerd en een lokale variabele nog niet. De waardeparameter kan in de romp een andere waarde worden gegeven maar dit heeft geen invloed op de doorgegeven waarde: een waarde, immers, kan niet worden gewijzigd, een variabele wel. Een variabeleparameter verschilt aanzoenlijk van een waardeparameter of lokale variabele, omdat dit geen copie van de waarde van de actuele parameter is, maar de actuele parameter zelf is, wat dan ook een variabele moet zijn. Dit komt neer op het vervangen van de naam van de formele parameter door die van de actuele parameter in de romp vóórdat de opdrachten daarin worden uitgevoerd: er wordt dus met de ingevoerde variabele gewerkt, die dan ook kan worden gewijzigd. Dit maakt het mogelijk om resultaten aan de omgeving af te geven. Een bekend voorbeeld is een subroutine die de waarde van een variabele inleest. In de aanroep wordt de naam van variabele vermeld waarvan de waarde moet worden ingelezen door de subroutine.
Door aan een dergelijk subroutine de variabele door te geven, kan de ingelezen waarde in die variabele worden geplaatst.

7.2 Organisatie van gegevens

Niet alleen de opdrachten moeten kunnen worden georganiseerd, ook de gegevens. Daarom dient een goede programmeeromgeving te beschikken over mogelijkheden om gegevens te bundelen in zgn. gegevensstructuren (zie voor details § 4.2). Een adequate programmeeromgeving stelt de gebruiker in staat zijn eigen gegevensstructuren te definiëren.

BOX 7.2 Gegevensstructuren in Visual Basic

In Visual Basic is het mogelijk gegevens op twee basismanieren te bundelen:
1. Voor gegevens van hetzelfde type in **arrays**, d.w.z. als rij (een-dimensionaal), tabel (twee-dimensionaal), kubus, ...
2. Voor gegevens van een verschillende type in **records**.

Een belangrijk verschil tussen beide is, dat een variabele, waarin meerdere gegevens van hetzelfde type moeten kunnen worden opgeslagen direct als array kan worden gedeclareerd. Voor de opslag van de bezetting van een schaakbord:

```
Dim SchaakBord(1 To 8,1 To 8) as Integer.
```

Terwijl, voordat in een variabele meerdere gegevens van verschillende soorten moeten kunnen worden opgeslagen, eerst de naam en het type van elk gegeven dat daarin moet kunnen worden opgeslagen, moet worden opgegeven in een Type-declaratie (moet in een module worden geplaatst). B.v.

```
Type tStudent
    ' de velden moeten net als variabelen worden gedeclareerd
    Naam As String * 30
    Geslacht as Boolean
    Leeftijd As Integer
    Studienummer As String * 6
End Type
Dim Studenten() As tStudent ' voor de opslag van de studentendata
```

De haakjes achter studenten geven aan dat het hier om een array gaat waarvan de dimensies nog niet bekend zijn! Deze kunnen in een later stadium met het ReDim-statement worden opgegeven. Ook is te zien dat de beide soorten gegevensstructuren kunnen worden gecombineerd! (In een recordstructuur kunnen ook arrays worden gedeclareerd).

Een **array** is een fysische gegevenssstructuur, terwijl een rij een logische gegevensstructuur is. Alle gegevensstructuren die in een bepaalde programmeer-omgeving beschikbaar zijn zijn fysische gegevensstructuren. Structuren die in het domein bestaan van het probleem – zoals b.v. een lijst studenten of de velden van een schaakbord – zijn logische gegevensstructuren. Bij het kiezen van de te gebruiken programmeeromgeving bij het implementeren van het ontwerp moeten voor de logische gegevensstructuren een geschikte fysische gegevensstructuur worden gekozen.

Het is niet moeilijk in te zien dat voor de opslag van de lijst studenten een array kan worden gebruikt waarvan elk element een recordstructuur heeft. Voor de opslag van de bezetting van de velden van een schaakbord kan ook eenvoudig een twee-dimensionale array worden gebruikt waarbij in elk element het nummer van het stuk wordt opgeslagen. Het opslaan van een lijst studenten in een array heeft echter als nadeel, dat, wanneer daaraan de gegevens van een bepaalde student op een bepaalde – niet- laatste plaats – moeten worden toegevoegd, b.v. om ervoor te zorgen

dat de studienummers op volgorde blijven, dan moeten eerst alle gegevens van alle studenten die daarna in de array staan worden opgeschoven, om een element te krijgen waarin de nieuwe gegevens kunnen worden geplaatst, wat nadelige gevolgen heeft voor de prestatie van het programma! De keuze van een geschikte fysische gegevensstructuur maakt dus ook af van de manier waarop de gegevens zullen worden gebruikt.
Het valt buiten het bestek van Informatica 1 in te gaan op logische gegevensstructuren die veel voorkomen en hoe deze het beste kunnen worden geïmplementeerd. Ik zal er slechts een paar noemen nl. de rij, de lijst, de boom, de tabel, de wachtrij.

Gewoonlijk wordt een ontwerp eerst gemaakt zonder acht te slaan op de te gebruiken fysische gegevensstructuren. Een dergelijk ontwerp wordt een logisch ontwerp genoemd. Pas zodra de keuze van de te gebruiken programmeeromgeving wordt gemaakt moet het logisch ontwerp in een fysisch ontwerp worden omgezet waarin geschikte fysische gegevensstructuren die optimaal zijn voor de logische gegevensstructuren die in het ontwerp voorkomen.

BOX 7.3 Gegevensstructuren in Visual Basic (vervolg)

Visual Basic beschikt slechts over de array en de record als fysische gegevensstructuren. Maar hiermee kan worden volstaan voor de implementatie van de meest gebruikte logische gegevensstructuren, als is het het absolute minimum. De recordstructuur kan niet worden uitgebreid tijdens de werking van het programma. Een array kan echter wel groter worden gemaakt en daardoor extra elementen bevatten, dan daarvoor. Daartoe moet bij de declaratie van de arrayvariabele de grootte van de array niet worden opgegeven, zoals in Box 7.2 gebeurde voor de lijst studenten. Aan de array **studenten** kan gemakkelijk een element worden toegevoegd met: **ReDim Preserve studenten(Ubound(studenten)+1)**.
De functie Ubound retourneert het nummer van het laatste element in de array dat na uitvoering met deze opdracht dus 1 groter is geworden. Het keyword Preserve is nodig om de gegevens die al in **studenten** aanwezig waren te behouden.

7.3 Organisatie van subroutines in modules

Hergebruik van opdrachten is erg belangrijk bij de ontwikkeling van een computerprogramma: hoe minder opdrachten het programma bevat, hoe gemakkelijker het programma te testen en te onderhouden is. Het is bovendien onzinnig om een aantal opdrachten die een bepaalde bewerking uitvoeren, b.v. voor het roteren van een figuur, op alle plaatsen waar deze in het programma dienen te worden uitgevoerd te moeten herhalen. De programmeeromgeving moet de programmeur in staat stellen opdrachten te bundelen en deze, wanneer de programmeur dit maar wil, te laten uitvoeren. In de praktijk komt dit erop neer, dat de gebundelde opdrachten een naam krijgen, en dat het vermelden van de naam ervan betekent dat de opdrachten behorende bij die naam moeten worden uitgevoerd. Een dergelijke groep opdrachten wordt dan een subroutine genoemd (zie ook § 7.1). (Het vermelden van de naam van de subroutine wordt het aanroepen van de subroutine genoemd en betekent dat de subroutine wordt uitgevoerd.)

Als voor het maken van het ontwerp van het computerprogramma van een top-down-benadering gebruik is gemaakt, en elk algoritme daardoor niet per se allemaal direct uitvoerbare opdrachten bevat, en het ontwerp derhalve uit meerdere algoritmen kan bestaan, kan elk algoritme in een aparte subroutine worden beschreven. De in dat algoritme voorkomende niet-direct uitvoerbare

opdrachten zullen dan als subroutine gerealiseerd kunnen worden, waardoor de organisatie van het programma overeenkomt met die van het ontwerp, wat de omzettting van ontwerp naar programma aanzienlijk vergemakkelijkt. Het is de taak van de ontwerper een hiërarchie van taken die door het programma dienen te worden uitgevoerd te ontdekken en te ontwerpen, en voor elk ervan een geschikt algoritme te ontwikkelen. Hoe een taak moet worden uitgevoerd hoeft niet verder te worden beschreven, als het een taak betreft die al door de programmeeromgeving kan worden uitgevoerd, zoals b.v. het wijzigen van de waarde van een variabele (met een toekenning) op het inlezen van variabelen van een bepaald type. Kennis van de primitieve taken die door een computerprogramma in een bepaalde taal kan worden uitgevoerd is derhalve noodzakelijk. (De belangrijkste primitieve taken die met Visual Basic kunnen worden uitgevoerd staan in het vorige hoofdstuk besproken.)

Wanneer het alleen maar ging om het herkennen van zich herhalende patronen van precies dezelfde opdrachten dan was deze taak nog niet zo moeilijk en kon zelfs worden geautomatiseerd. Het komt echter veelvuldig voor dat het niet om precies dezelfde opdrachten gaat maar om varianten waarin alleen de te gebruiken gegevens veranderen! Het zal toch immers niet steeds dezelfde figuur zijn die moet worden geroteerd?! Zo is het niet ondenkbaar dat in een computerprogramma herhaaldelijk de inhoud van twee verschillende variabelen moeten worden verwisseld (zie Box 7.4). De te gebruiken variabelen worden dan aan de subroutine doorgegeven door ze als actuele parameters in de aanroep te vermelden. De beschrijving van de subroutine vermeld een lijst van invoergegevens en benoemt deze en gebruikt deze als ware bekend om welke variabelen het gaat.

Ter herinnering

Dit zijn de formele parameters. Daarnaast is het mogelijk om variabelen te declareren die alleen binnen de procedure actief zijn d.w.z. te gebruiken zijn: deze worden lokale variabelen genoemd. Deze zijn alleen tijdens de uitvoering van de subroutine beschikbaar. Lokale variabelen die gedurende de hele werking van de subroutine beschikbaar zijn, worden ook statische variabelen genoemd, alhoewel ze dus niet gedurende de totale werking van het programma beschikbaar zijn!

(Visual Basic gebruikt echter het keyword Static voor het declareren van lokale variabelen waarvan de waarde tussen opeenvolgende aanroepen van de subroutine moet worden bewaard; het gaat hier dan dus om globale variabelen die echter niet in andere delen van het programma kunnen worden gebruikt!)

BOX 7.4 Subroutine ter verwisseling van twee getallen

Een eenvoudig voorbeeld is de verwisseling van twee getallen. Verwisseling ervan kan met de volgende opdrachten in Visual Basic (met achter het aanhalingsteken (het commentaarteken in VB) de bijbehorende pseudocode):

```
t=a ' maak t gelijk aan a
a=b ' maak a gelijk aan b
b=t ' maak b gelijk aan t
```

waarbij t een hulpvariabele is waarin de waarde van a wordt bewaard zodat die nog bekend is als b de waarde daarvan moet krijgen. Moet op een ander moment in het programma de inhoud van de variabelen **e** en **f** worden verwisseld, dan kan dat met:

```
t=e
e=f
f=t
```

waarin t weer als hulpvariabele voor de verwisseling fungeert.

Box 7.4 Subroutine ter verwisseling van twee getallen (vervolg)

De werkwijze is in beide gevallen hetzelfde, maar de opdrachten zijn niet precies gelijk. In het ene geval gaat het om de variabelen **a** en **b**, in het andere geval om de variabelen **e** en **f**. Het zou fijn zijn als het mogelijk was om de toch identieke bewerking van het verwisselen van twee getallen maar één keer in het programma te hoeven specificeren en daarna te kunnen laten uitvoeren met verschillende soorten invoer, in het ene geval met **a** en **b** en in het andere geval met **e** en **f**. In het ene geval zal ik aan de subroutine die het verwisselen uitvoert **a** en **b** als invoer willen doorgeven, in het andere geval **e** en **f**.
Dit betekent dat het mogelijk moet zijn een stukje code te schrijven met zijn eigen in- en uitvoer. Als invoer accepteert het te schrijven stukje code twee variabelen, als uitvoer – in dit geval – dezelfde variabelen, met hun inhoud verwisselt. Het probleem is echter, dat dat stukje code niet weet wat de namen van de variabelen weet die het als invoer doorkrijgt. Toch kan code worden geschreven (en in machinetaal worden omgezet) die de gewenste functie vervult. Daartoe moet je je realiseren dat onderscheid gemaakt moet worden tussen de code zoals die wordt opgegeven en de code zoals die moet worden uitgevoerd. Op het moment dat de code wordt bedacht hoeft niet bekend te zijn welke variabelen precies opgegeven zullen worden wanneer de code moet worden uitgevoerd. In de specificatie kun je de in- en de uitvoer namen geven als betrof het variabelen engebruiken als betrof het bekende variabelen. De namen die je aan de in- en uitvoergegevens geeft vermeld je in de parameterlijst. Hier zijn dat er twee en ik noem deze ene en andere. De code die de verwisseling uitvoert wordt dan:

```
t=ene
ene=andere
andere=t
```

Dit stukje code moet voorzien worden van een beschrijving van de parameterlijst. In een algoritme zou je gewoon een lijstje maken met alle in- en uitvoergegevens die de subroutine gebruikt:

```
Ene      variabele-parameter     na afloop gelijk aan andere
andere   variabele-parameter     na afloop gelijk aan ene
t        lokale variabele        voor opslag waarde ene
```

Uitvoeren van Verwissel op a en b kan dan overal met:

```
Call Verwissel(a,b)
```

Tussen de haakjes staan de actuele parameters, die dus de actuele in/uitvoergegevens te gebruiken in de uitvoering voorstellen. Voor ene zal bij uitvoering dan a worden gebruikt, voor andere b. Het woord Call mag in Visual Basic worden weggelaten.

De globale variabelen in het programma kunnen beschouwd worden als de lokale variabelen van het programma: ze zijn alleen binnen het programma actief. Of andersom: de lokale variabelen binnen een subroutine kunnen als de globale variabelen van het programmaatje dat de subroutine is worden beschouwd. De gegevens die in het werkgeheugen van de computer staan, kunnen als de globale variabelen van de computer worden beschouwd, die verdwijnen als de computer wordt uitgezet. De gegevens die op het achtergrondgeheugen van de computer staan; ze zijn echter resistent: ze blijven (gewoonlijk voldoende lang) bestaan ook al staat de computer niet aan!

Het kunnen bundelen van opdrachten tot subroutines die als mini-programma's fungeren met hun eigen lokale toestand is een belangrijk onderdeel van alle zichzelf-respecterende programmeeromgevingen en het ontbreken van een dergelijke faciliteit is ondenkbaar in de hedendaagse programmeeromgevingen. Het is belangrijk in te zien dat de subprogramma's gemakkelijker te maken en te onderhouden zijn omdat ze gemakkelijker te begrijpen en doorgronden, omdat ze klein aantal gegevens gebruiken en beheren.

BOX 7.5 Procedures en functies in Visual Basic

Procedures beginnen met het keyword **Sub**; functies met het keyword **Function**. Van het antwoord van de functie moet het gegevenstype worden opgegeven aan het einde van de regel.

Een procedure voor de verwisseling van twee gehele getallen:

```
Sub Verwissel(intEne as integer, intAndere as integer)
Dim intHulp as integer
    intHulp = intEne
    intEne = intAndere
    intAndere = intHulp
End Sub
```

Een functie voor het optellen van twee reële getallen:

```
Function Som(ByVal intEerste as double, _
                ByVal intTweede as Double) As Double
    Som = intEerste + intTweede ' resultaat toekennen aan naam
End Function
```

Elke programmeeromgeving moet minimaal over bovengenoemde organisatie-faciliteiten beschikken. Het staat de gebruiker toe subroutines te maken die afzonderlijk kunnen worden getest en aangezien deze kleiner zijn dan het totale programma is dat gemakkelijker. Als alle onderdelen naar behoren werken, werkt het hele programma ook naar behoren. Er is echter een probleem, dat het testen na aanpassing aanzienlijk bemoeilijkt. Elke subroutine kan in principe elke globale variabele wijzigen, zonder dat hierop controle wordt uitgeoefend. Bij het aanpassen van een bepaalde subroutine is het dan noodzakelijk alle subroutines die dezelfde globale variabelen gebruiken opnieuw te testen. Dit is uiteraard een ongewenste situatie. Het verdient dan ook de voorkeur om subroutines niet direct van globale gegevens gebruik te laten maken! Dit kan worden bereikt door alle benodigde gegevens als parameter aan de subroutine door te geven! Alhoewel dit het probleem verplaatst naar de aanroepende subroutine, zorgt dit er wel voor dat de subroutine direct in andere programma's kan worden gebruikt, aangezien zich in de subroutine geen namen van globale gegevens voorkomen die in dat programma gedeclareerd zijn! Afzonderlijk testen wordt dan ook mogelijk.
De globale gegevens kunnen worden beschouwd als gemeenschappelijke gegevens van de subroutines. Nu is het in de regel zo, dat niet al die gegevens door alle subroutines gebruikt worden. Dan is het mogelijk subroutines in groepen in te delen die globale gegevens gebruiken die alleen in de huidige groep gebruikt worden. Het zou fijn zijn als de globale gegevens die alleen door een bepaalde groep subroutines worden gebruikt lokaal te maken voor die groep subroutines, d.w.z. te garanderen dat ze alleen door subroutines in die groep gebruikt kunnen worden, en nooit door

subroutines in andere groepen. Bij het testen na aanpassingen hoeven alleen de subroutines in de groep waarin de te testen subroutine opnieuw te worden getest omdat die alleen gegevens delen.
Deze overwegingen hebben ertoe geleid, dat veel programmeeromgevingen het bundelen van subroutines toestonden in functionele eenheden, die units (Pascal) of modules (Modula-2) of packages (Ada of SmallTalk) werden genoemd en in aparte bestanden worden opgeslagen. Deze bevatten gewoonlijk twee delen: een interface-gedeelte en een implementatie-gedeelte. In het interface-gedeelte wordt alles opgegeven waarover andere eenheden moeten kunnen beschikken: echte globale gegevens en de subroutines die door andere eenheden moeten kunnen worden opgeroepen. In het implementatie-deel wordt de werking van alle subroutines beschreven; alle gegevens die alleen door de subroutines in de unit moeten kunnen worden gebruikt en overige subroutines die alleen door andere subroutines in de unit kunnen worden aangeroepen, en waarvan de interface – waarin de manier waarop de subroutine moet worden aangeroepen wordt beschreven – niet beschikbaar gesteld wordt in het interface-gedeelte.
Gegevens kunnen dan minimaal op drie niveau's worden gebruikt:

1. **Globaal**: gegevens die overal in het programma kunnen worden gebruikt;
2. **eenheid-lokaal**: gegevens die beschikbaar zijn voor alle subroutines in een eenheid;
3. **lokaal**: gegevens die alleen gebruikt kunnen worden door opdrachten binnen één subroutine.

BOX 7.6 Gebruiksgebied van gegevens in Visual Basic

Een Visual Basic programma kent twee soorten eenheden:

1. De form, die een venster definiëert met alle benodigde functionaliteit daarvan;
2. De module, met bij elkaar horende subroutines die niet bij een venster (hoeven te) horen.

Gegevens die alleen lokaal dienen te worden gebruikt moeten met het DIM-statement worden gedeclareerd. Met het PUBLIC- of GLOBAL-statement kunnen gegevens worden gedeclareerd die ook door andere forms en modules kunnen worden gebruikt. Wanneer van Public hetzij Global gebruik gemaakt worden wordt verderop uitgelegd.

BOX 7.7 Automatische procedures in Visual Basic forms

Een form en elke control daarop moet in staat te reageren op handelingen die een gebruiker verricht ten aanzien van dat form of die control. Derhalve herkent elke form en control een aantal events (gebeurtenissen) zoals het klikken erop of het bewegen met de muis erover. Als een dergelijk event optreedt wordt een subroutine uitgevoerd, die daarom een event-handler wordt genoemd. Visual Basic stelt je in staat ook te reageren op het optreden van elk event. Hiertoe stelt Visual Basic eventprocedures ter beschikking. De naam ervan begint met de naam van het control of Form (als het een form betreft!) gevolgd door de naam van het event gescheiden door een liggend streepje b.v. FORM_CLICK. Deze naam ligt vast en dient niet te worden gewijzigd. De eventprocedures kunnen worden gevonden in het Code venster.

7.4 Voorbeeld: Een Zeer Eenvoudige Koffie-Automaat

Het in dit voorbeeld gemaakte ontwerp hoef je niet te kunnen reproduceren, maar wel begrijpen.

In dit voorbeeld wordt een applicatie ontworpen die een zeer eenvoudige koffie-automaat simuleert.
Het gaat hierbij nadrukkelijk niet om de kwaliteit van de user interface van zo'n apparaat, maar om de werk- en denkwijze die gebruikt wordt bij het tot stand komen van een computerprogramma die de werking van een dergelijk apparaat nabootst. Aan de user interface zelf wordt daarom – en daarvoor mijn welgemeende excuses – minimale aandacht besteedt.

In deze paragraaf wordt alleen die code getoond die voor het begrijpen van de basisbeginselen van van belang is en op dit moment begrepen kan worden. Op de computers die in het practicum worden gebruikt kunnen in de directory die door de leiding gebruikt wordt de applicatie worden gevonden die het resultaat is van het hier gemaakte ontwerp.

De hier gebruikte aanpak moet ook in het practicum worden opgevolgd.
Het ontwerp bestaat uit het ontwerp van de user interface (de *look*) en de functionaliteit, het gedrag (de *feel*). Het ontwerp van de user interface kan eenvoudig bestaan uit een tekening waarop de onderdelen, en een beschrijving ervan staan vermeld. In de beschrijving van elk onderdeel moet worden aangegeven waar deze voor dient d.w.z. wat de functie van het onderdeel is.
De organisatie van de te realiseren functionaliteit wordt bij een dergelijke user interface door de user interface afgedwongen: elke specifieke activiteit van de gebruiker met betrekking tot een afzonderlijk te onderscheiden onderdeel veroorzaakt een reactie van het gesimuleerde apparaat. De keuze de functionaliteit van het programma te organiseren in hoofdsubroutines, waarbij elke hoofdsubroutine de reactie van het apparaat op een gebruikersactie is, is een vanzelfsprekende. De verdere opdeling is een ontwerpkeuze die niet gemakkelijk te maken is. Wanneer de hoofdsubroutines geen gemeenschap-pelijke code bezitten is een verdere opdeling niet noodzakelijk. Deze kan overigens wel gewenst zijn wanneer dit bevordelijk is voor de leesbaarheid en begrijpbaarheid van de algoritmen.

Vraag je daarom bij het maken van de algoritmen af wat het algoritme moet doen d.w.z. welke functies het algoritme moet vervullen en niet hoe het deze moet vervullen. Of een functie wel of niet gemakkelijk kan worden geïmplementeerd – b.v. door direct geschikbaar te zijn in de te gebruiken omgeving – moet daarbij buiten beschouwing worden gelaten. Dit hangt immers mede af van de te gebruiken programmeeromgeving die in dit stadium vaak nog niet bekend is! Het gaat in eerste instantie om de logica van de oplossing en niet om de specifieke fysische verschijningsvorm waarin het ontwerp gegoten moeten worden om uitgevoerd te kunnen worden d.w.z. de mogelijkheden van de te gebruiken programmeertaal en – omgeving. Overigens is kennis van de mogelijkheden die alle geschikte programmeeromgevingen gemeen hebben wel een pre, omdat op grond daarvan – min of meer - kan worden vastgesteld of een ontwerp klaar is. Het is overigens niet onmogelijk dat voor de realisatie van een bepaalde functie als beschreven in een ontwerp in een bepaalde programmeeromgeving die op het moment van het maken van een ontwerp nog niet vast stond meerdere opdrachten nodig zijn. Als tijdens het maken van het ontwerp de ontwerper weet welke programmeeromgeving gebruikt zal gaan worden en dus welke mogelijkheden deze heeft, dan kan de ontwerper precies bepalen wanneer elke functie in het ontwerp met precies één opdracht kan worden

gerealiseerd en de implementatie gemakkelijk is. Het is echter beter dat het ontwerp niet beïnvloed wordt door kennis omtrent de implementatie-omgeving omdat dit de vorm waarin de oplossing moet worden gegoten vastlegt, waardoor wellicht een niet-optimaal ontwerp ontstaat. Het valt overigens niet te voorkomen dat de vorm waarin het ontwerp wordt gegoten beinvloed wordt door het soort programmeeromgeving dat zal worden gebruikt. Wanneer een ontwerper ervaring heeft met het implementeren van ontwerpen is deze bekend met de manier waarop de meest elementaire functies kunnen worden gecombineerd om bepaalde abstracte functies te realiseren. Een dergelijke ontwerper zal de neiging hebben zijn ontwerp uit te drukken in termen van functies, waarvan de implementatie hem bekend is.

Een absolute beginner kent alleen de meest elementaire basisfuncties zoals die door de taal worden gedefiniëerd, en heeft daardoor veel meer dan de ervaren ontwerper de neiging een oplossing direct in termen van de mogelijkheden van dat beperkte(re) assortiment uit te drukken. Het bedenken van een algoritme dat direct van die meest elementaire, en dus kleine, bouwstenen gebruikt kan daardoor echter een onoplosbare complexiteit krijgen. Een beginnende ontwerper moet zich daarom niets aantrekken van zijn gebrek aan ervaring en zichzelf toestaan functies te beschrijven die niet direct zijn te implementeren, d.w.z. waarvoor niet een enkele opdracht in de te gebruiken omgeving bestaat. Hierbij is een beginnend ontwerper zelfs in het voordeel,omdat hij onbevooroordeeld is, en niet beinvloed wordt in de richting van bepaalde – bekende – oplossingen te werken.

Het ontwerp is af, zodra deze op zodanige wijze is uitgedrukt, dat elke functie met een enkele opdracht in de te gebruiken programmeeromgeving kan worden beschreven! Elke functie die niet met een opdracht beschreven kan worden moet in een apart algoritme verder worden uitgewerkt. Elk algoritme komt in de uiteindelijke code in een aparte subroutine te staan.
Deze aanpak – het geleidelijk aan in meer detail uitwerken van algoritmen – wordt **stapsgewijze verfijning** genoemd en is een voorbeeld van een top-down-ontwerpmethode. Bij de tegenhanger, de bottom-up-methode, worden juist basisfuncties gecombineerd tot nieuwe basisfuncties, die dan weer worden gecombineerd tot weer nieuwe functies op een volgend abstractieniveau. Gebruikmaken van de bottom-up-methode is alleen dan mogelijk wanneer er een duidelijk beeld bestaat van benodigde functies. Gedacht kan worden aan de functies die user interface elementen moeten kunnen vervullen. Zelfs voordat een project van start gaat kan worden bedacht dat bepaalde functies nodig zijn die moeten worden gebouwd, omdat ze nodig zijn maar niet direct beschikbaar zijn! Het succes van een bepaalde programmeeromgeving wordt met namen bepaald door het inzicht van de ontwerpers in de bouwstenen die gebruikers ervan nodig zouden kunnen hebben. Zij moeten daartoe een beeld hebben van het soort applicaties dat met de omgeving gebouwd moet kunnen worden en de faciliteiten die daarvoor nodig zijn!

Voordat aan het ontwerpen kan worden begonnen moet je goed weten wat er verwacht wordt van de te bouwen applicatie: er moet een definitie worden gemaakt. Resultaat van deze definitie is een specificatie van de te bouwen applicatie: een beschrijving van de werking van de applicatie die zo volledig, ondubbelzinnig en consistent mogelijk is. Een dergelijke beschrijving kan met behulp van schema's of tekeningen worden verduidelijkt.

7.4.1 De specificatie

De zeer eenvoudige koffie-automaat moet opeenvolgende gebruikers in staat stellen:

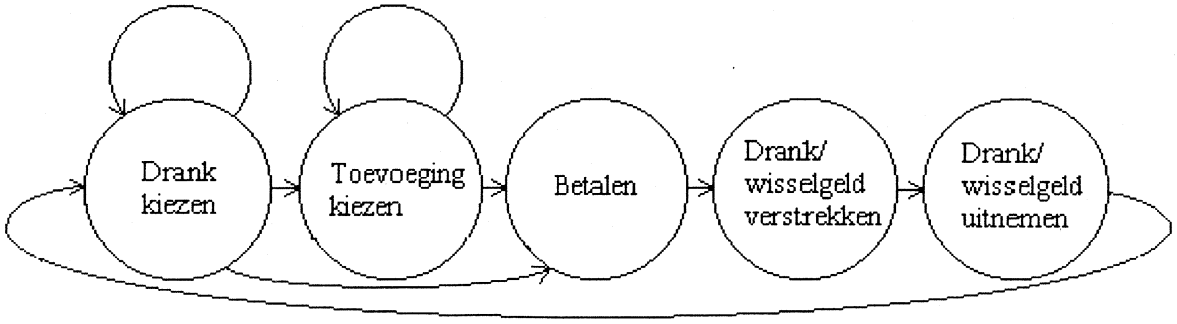
1. Een keuze te maken uit een aantal beschikbare dranken;
2. Een keuze te maken uit een aantal – mogelijkerwijs per drank – verschillende toevoegingen;
3. De drank te laten betalen;
4. Het drankje en eventueel wisselgeld te verstrekken;
5. Het drankje en eventueel het wisselgeld te laten verwijderen;

(Niet alleen de functies moeten worden opgenoemd maar ook de volgorde waarin deze moeten worden uitgevoerd! Feitelijk komt in de specificatie al tot uitdrukking welke – te onderscheiden - functies het apparaat moet kunnen vervullen, en kan dus als een ontwerpactie worden beschouwd, alhoewel het hier niet een-twee-drie duidelijk is welke alternatieven er zijn!)

De hier genoemde functies moeten door de gebruiker in de aangegeven volgorde worden uitgevoerd:

- pas nadat een drank gekozen is, kunnen toevoegingen worden gekozen;
- pas nadat een drank is gekozen, kan worden betaald;
- zodra een toevoeging is gekozen of al wat betaald is, mag geen andere drank meer worden gekozen;
- zodra betaald is, mag geen toevoeging meer worden gekozen;
- zodra de drank betaald is, wordt deze en eventueel wisselgeld verstrekt;
- zodra de drank/het wisselgeld verstrekt is, mag deze worden uitgenomen;
- pas nadat de drank en eventueel wisselgeld uitgenomen is, mag een volgende gebruiker dit proces doorlopen.

Een en ander kan vermoedelijk in een diagram duidelijker worden gemaakt! In de diagram worden de verschillende acties met cirkels aangegeven. Met gerichte lijnen wordt aangegeven wat de mogelijke volgende actie is. Een dergelijk diagram geeft veel duidelijker dan een tekst aan welke mogelijkheden het systeem moet bieden.



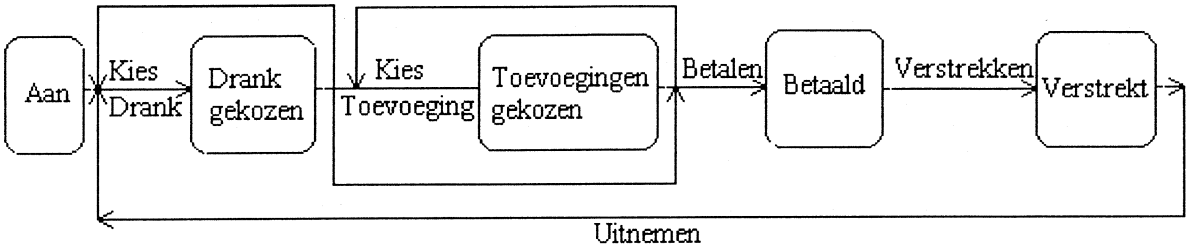
Figuur 7.1 Specificatiediagram van de zeer eenvoudige koffie-automaat

Het aan- cq. uitzetten van de zeer eenvoudige koffie-apparaat is hierin buiten beschouwing gelaten. In het diagram is steeds met pijlen aangegeven welke volgende acties uitgevoerd kunnen worden. Zo kan een gebruiker na het kiezen van een drank (opnieuw) een drank kiezen, gaan betalen, of toevoegingen kiezen. In het diagram is echter niet aangegeven welke toevoegingen beschikbaar zijn wanneer welke dranken zijn gekozen.

De acties die herhaald kunnen worden zijn in dit diagram zeer duidelijk aangegeven, maar betekenen verschillende dingen:

- De gebruiker kan één drank kiezen zolang nog geen toevoeging is gekozen.
- De gebruiker kan een aantal toevoegingen kiezen zolang niet met betalen is begonnen.
- Na het uitnemen van de drank/het wisselgeld kan opnieuw een drank worden gekozen.

Er bestaan vele soorten diagramtechnieken voor het weergeven van een specificatie. In bovenstaand diagram zijn de acties (processen) als knoop aangegeven. Wanneer naast de pijlen de gegevens zouden worden gezet die door het proces wordt voortgebracht, ontstaat een data flow diagram.
Het is echer ook mogelijk om de acties als pijl weer te geven als in onderstaand zgn. state transition diagram (STD).



Figuur 7.2 State transition diagram van de zeer eenvoudige koffie-automaat

In de afgeronde rechthoeken staat de toestand vermeld waarin de koffie-automaat zich op dat moment bevindt. Een en ander dient voor zich te spreken!

Vervuiling van de specificatie met ontwerpkeuzes

De specificatie moet beschrijven WAT er van het apparaat verwacht wordt, d.w.z. de eisen die aan het apparaat worden gesteld, en niet HOE het apparaat deze moet vervullen. Wanneer in de specificatie – eventueel impliciet - van een bepaalde volgorde waarin de functies moeten worden vervuld wordt uitgegaan, dan bevat de specificatie al een aantal ontwerpbeslissingen. Bij het opstellen van de specificatie van een bestaand apparaat, is het moeilijk te voorkomen dat kennis m.b.t. de werking van dergelijke apparaten gebruikt wordt bij het opstellen van de specificatie. Ontwerpbeslissingen van anderen die een dergelijk apparaat hebben bedacht komen dan in de specificatie terecht en dus ook de fouten die daarbij gemaakt zijn. U bent gewaarschuwd!
Het is daarom van belang dat vooraf zo veel mogelijk alternatieve gebruikswijzen worden bedacht, b.v. in de vorm van scenario's, die dan door de opdrachtgever en – liefst ook - toekomstige gebruikers worden geevalueerd.
Anders zal pas tijdens het ontwerpen ontdekt worden dat er bij het opstellen van de specificatie niet-bedoelde ontwerpkeuzes zijn gemaakt. Zo ligt in de specificatie als hierboven getoond al vast, dat eerst de drank wordt gekozen en dan de extra's. Pas daarna wordt betaald. Hoe meer energie al in het maken van het ontwerp is gestoken, hoe meer weerstand er zal zijn om opnieuw te beginnen met ontwerpen.
Opvallend is overigens dat de (pijlen in de) getoonde diagrammen alleen mogelijk zijn wanneer al ontwerpkeuzes zijn gemaakt m.b.t. de volgorde waarin de functies moeten worden vervuld, en toch als middel tot specificatie worden beschouwd! Een zuivere specificatie zal slechts een opsomming van de te vervullen functies zijn zonder de indruk te wekken dat die in een bepaalde volgorde zullen moeten worden uitgevoerd. Dat die indruk vaak wel wordt gewekt, komt door de volgorde waarin de functies worden opgesomd gebaseerd op de werking van vergelijkbare apparaten als bekend bij de opsteller van de specificatie.

7.4.2 Het ontwerpen

Het moeilijkste is altijd waar te beginnen. Het eindresultaat, een computerprogramma dat onze zeer eenvoudige koffie-apparaat nabootst, is bekend, maar dit vertelt ons niet zo veel over hoe.
Hoe realistisch moet en kan een dergelijke simulatie zijn? Een computer is, om maar een voorbeeld te noemen, immers niet in staat om koffie te schenken; het is hooguit in staat dit proces uit te beelden, maar zelfs dat is niet verplicht: het zou kunnen volstaan met een mededeling die meldt dat de drank is verstrekt!

Eigenlijk zou in de specificatie vastgelegd moeten zijn waarvoor de simulatie gebruikt gaat worden.
Daaruit kan dan worden afgeleid hoe realistisch de simulatie moet zijn. Dan is het zaak na te gaan met welke programmeeromgeving deze natuurgetrouwheid kan worden bereikt. Voor een enigszins realistische nabootsing is het toch minimaal vereist dat de user interface van het apparaat getoond kan worden en dat deze te verwachten interactie faciliteert d.w.z. mogelijk maakt. Is dit mogelijk, dan kunnen we eerst de user interface ontwerpen en vervolgens de interactie, bestaande uit de toepasselijke reactie van de applicatie op gebruikersacties. Er zijn hedentendage vele programmeeromgevingen, waaronder specialistische prototyping tools, die het mogelijk maken de user interface min of meer natuurgetrouw in een venster na te bootsen, waaronder de programmeeromgeving die tijdens het practicum wordt gebruikt, Visual Basic. In Visual Basic kunnen op een venster controls als knoppen (command button), voor het weergeven van tekst (labels) en plaatjes (image of picture box) worden geplaatst voor de communicatie met de gebruiker.
In ons geval bestaat het ontwerp dan ook uit de user interface die bepaalt welke gebruikersacties mogelijk zijn, en de beschrijving van de reactie van de applicatie op elke van de mogelijke gebruikers-acties. Eerst bedenken we de user interface, dan de rest.

7.4.2.1 Ontwerpen van de user interface

Om de user interface te kunnen maken, moeten we echter wel weten welke acties een gebruiker moet kunnen verrichten. In dit eenvoudige voorbeeld zal het niet moeilijk zijn hier achter te komen. In theorie voert de gebruiker een aantal taken uit teneinde bepaalde doelstellingen te verwezenlijken. De doelstellingen bepalen de taken. Uit de taken moeten de acties worden afgeleid. De doelstelling van de gebruiker hier is het verkrijgen van een bepaalde drank. Allereerst moet de gebruiker bepalen of dit doel met de koffie-automaat kan worden bereikt. (De eerste taak van de koffie-automaat is dus de eventuele gebruiker op de hoogte te stellen van de beschikbare dranken en hoe deze kunnen worden verkregen, d.w.z. welke taken moeten worden uitgevoerd om een bepaald doel te bereiken.)

Het is niet ondenkbaar dat een koffie-automaat op verzoek een drank gratis verstrekt, en dit kan een instelbare optie zijn voor een te bedenken ontwerp, maar hier gaan we ervan uit dat het om een koffie-automaat gaat die meerdere dranken kan verstrekken tegen vergoeding. Aan bepaalde dranken daarvan kunnen – gebaseerd op principes van goede smaak – stoffen ter verhoging van de smaak worden toegevoegd. Een taakanalyse leert ons derhalve, dat de volgende taken door de gebruiker moeten worden verricht met de te ontwerpen koffie-automaat:

1. Het selecteren van een dranksoort.
2. Het selecteren van een aantal toevoegingen.
3. Het betalen van het bestelde.
4. Het uitnemen van het bestelde en eventueel wisselgeld.

(Hier hebben we de door taken afgeleid uit de specificatie, waarin de functies die de koffie-automaat moet kunnen vervullen, staan beschreven.)
De koffie-automaat moet de gebruiker via de user-interface in staat stellen deze taken uit te voeren. De taken geven aanleiding tot de definitie van acties die moeten worden uitgevoerd om de taak uit te voeren. Per taak moet worden bepaald welke acties dit zijn.

Het selecteren van een dranksoort
De automaat is in staat meer dan één drank te verstrekken. Daarom moet de gebruiker een keuze kunnen maken uit een aantal beschikbare dranken. In de praktijk maakt een gebruiker zijn keuze kenbaar door de knop in te drukken met daarop de naam van de drank die dan wordt verstrekt. Vandaar dat we op de user-interface voor elke mogelijk te verstrekken drank een knop zullen aanbrengen. We moeten bepalen welke dranken kunnen worden gekozen; dit bepaalt immers het aantal knoppen en de tekst op de knoppen. Normaal staat dit in de specificatie; aangezien het hier slechts om een voorbeeld gaat, en niet om de simulatie van een daadwerkelijk te bouwen automaat, moeten we als ontwerper zelf een keuze maken.Wel moeten tenminste 2 dranken gekozen kunnen worden. We kiezen voor 4 dranken: koffie, thee, warm water en chocolademelk.

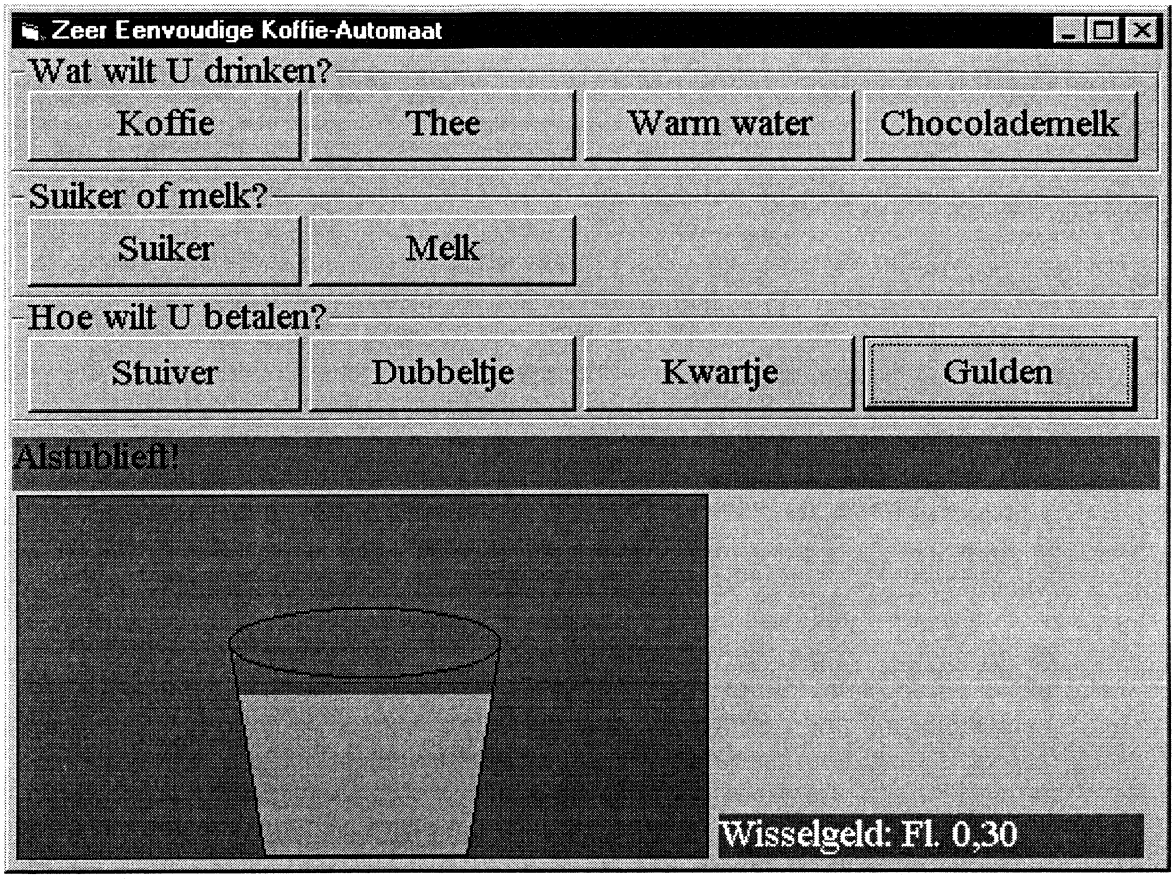
Het selecteren van een aantal toevoegingen
Ook hier geen in de specificatie vastgelegde toevoegingen; we kiezen de gebruikelijke extra's: suiker en melk, die ook via knoppen kunnen worden toegevoegd.

Het betalen van het bestelde
Betalen kan met echt geld (munt of papier) of met een card. Hier kiezen we voor betalen met echt geld. Omdat het om kleine bedragen gaat is, kiezen we voor betalen met muntgeld. In een realistische simulatie zul je afbeeldingen van muntstukjes naar een gleuf kunnen slepen en erin kunnen laten vallen. Minder realistisch – maar sneller realiseerbaar - is voor elke munt een knop op de interface te zetten waarop gedrukt moet worden om aan te geven dat de munt wordt 'ingeworpen'. In de gekozen oplossing bevat de interface vier knoppen: stuiver, dubbeltje, kwartje en gulden.

Het uitnemen van het bestelde en eventueel wisselgeld
Echt uitnemen is er natuurlijk niet bij. Hooguit kan de afbeelding van een bekertje met drank versleept worden naar een plaats 'buiten' het apparaat. Hetzelfde geldt voor het wisselgeld. Het eenvoudigst te realiseren is een klik op de afbeelding van bekertje en wisselgeld. In de hier ontwikkelde versie wordt inderdaad een bekertje getekend, waarop geklikt moet worden om uitnemen aan te geven. Het wisselgeld wordt daarentegen niet echt getekend, maar gewoon in tekst getoond; ook hierop moet worden geklikt. Tsja, hoe realistischer, hoe meer tijd het kost; aangezien in dit voorbeeld de interactie centraal staat doen we wat water bij de wijn!

Overige interface-aspecten
Tot nu toe hebben we ons beziggehouden met taken die de gebruiker moet kunnen uitvoeren. Hieruit kon worden afgeleid welke elementen de user-interface tenminste moest bevatten. (Alhoewel we ervan uit zijn gegaan dat elke knop maar voor een functie gebruikt kan worden, waardoor meer knoppen nodig zijn dan het absolute minimum!) Tot nu toe hebben we 4 knoppen voor invoer van de dranksoort, 2 knoppen voor invoer van de extra's, 4 knoppen voor betalen met muntjes, een afbeelding van een bekertje met inhoud en een tekst met wisselgeld.

Welke elementen moet de interface nog meer bevatten? Wel, alhoewel het de voorkeur geniet dat de gebruikswijze evident is kan het geen kwaad de gebruiker te vertellen wat er van hem/haar verwacht wordt. Vandaar dat op de gemaakte interface een regel tekst kan worden weergegeven met een gebruiksaanwijzing in wat we de display zullen noemen. Natuurlijk is het de vraag of deze wel gelezen wordt of kan worden (door bepaalde personen en/of onder bepaalde omstandigheden) maar met overwegingen van ergonomische aard houden we ons hier niet bezig. Dus: er mag niet van worden uitgegaan dat de aanwijzing wordt opgevolgd: het ontslaat de automaat niet van de verplichting de gebruiker tegen zichzelf te moeten beschermen. Een en ander geeft aanleiding tot de volgende interface.



7.4.2.2 Ontwerpen van de interactie

Nu begint het echte werk, nl. bedenken wat er moet gebeuren als de gebruiker wat doet. Eerst even een lijstje maken van taken en bijbehorende acties waarop gereageerd moet worden.

Taak	Actie(s)
Kiezen van een drank	Klikken op een van de 4 knoppen met een dranksoort erop
Kiezen van een toevoeging	Een aantal keer klikken op een van de 2 knoppen met een extra erop
Betalen	Klikken op een aantal van de 4 knoppen met de naam van een muntstuk erop, totdat er meer betaald is dan er betaald moest worden
Uitnemen van het bestelde en eventueel wisselgeld	Klikken op het getoonde bekertje en eventueel op de tekst waarin het wisselgeld vermeld staat

Elke reactie bestaat uit een aantal – geen, één of meer - opdrachten die in essentie de toestand van het programma zodanig wijzigen dat een juiste werking wordt verkregen. Die toestand bestaat uit de waarden van alle variabelen die het programma bijhoudt. Behalve die in de interface hebben we er nog geen. Bij het ontwerpen van de reactie moeten we ons afvragen: moet er een bepaalde waarde worden onthouden, en zo ja, in welke variabele? Bestaat die variabele nog niet, dan moet die een naam worden gegeven. (Bij het implementeren betekent dit dat deze moet worden gedeclareerd, maar tijdens het ontwerpen hoeft dat nog niet!) In een toekenning kan dan de waarde aan die variabele worden gegeven. Natuurlijk kan het wel voorkomen dat we bepaalde waarden wel berekenen maar niet onthouden en later toch merken dat we deze nodig hebben, dan zal het ontwerp moeten worden aangepast, maar dat is niet zo erg. Geen paniek dus.

Reactie op het Kiezen van een drank

Welke waarde(n) zou(den) hierdoor moeten worden onthouden? Misschien de gekozen drank? Hebben we die waarde dan nog ergens anders nodig? Op dit moment is dat niet te zeggen. Misschien om te kunnen bepalen hoeveel straks betaald moet worden. Maar wat betaald moet worden kan ook nu direct worden bepaald! Hebben we de gekozen drank verder nog ergens voor nodig? Misschien niet en dan kunnen we in een later stadium de variabele waarin de waarde staat opgeslagen van de gekozen drank weer verwijderen. Maar ervan uitgaande dat we elders in het programma nog moeten weten welke drank gekozen is, besluiten we in een variabele op te slaan welke drank is gekozen. Het is gemakkelijk om deze variabele een naam te geven b.v. GekozenDrank. (Alhoewel er op zich niets op tegen is om de naam van een variabele uit meerdere woorden te laten bestaan, kies ik er hier voor, in overeenstemming met de eisen die er in programmeeromgevingen aan de namen van variabelen worden gesteld, geen spaties in de naam voor te laten komen. Teneinde de leesbaarheid te bevorderen begin ik elk 'woord' in de naam met een hoofdletter, conform de Hongaarse notatie.) Maar welke waarde moett dan in die variabele worden opgeslagen voor een bepaalde gekozen drank? De specifieke waarde doet er eigenlijk niet toe, als het maar een verschillende waarde is voor elke andere drank, zodat elders in het programma aan de waarde te zien is welke drank is gekozen. Ikzelf zou een nummer gebruiken (1 voor koffie, 2 voor thee, enzovoorts.), maar tekst is misschien natuurlijker: "koffie", "thee", etc. (De dubbele aanhalingstekens worden hier in overeenstemming met het gebruik in VB gebruikt, om aan te geven dat het om een tekst gaat en niet om de naam van een variabele!) Samengevat betekent dit dat als volgt gereageerd zou kunnen worden op

Het klikken op de Koffie-knop (versie 1)

maak GekozenDrank gelijk aan "koffie"

Het klikken op de Thee-knop (versie 1)

maak GekozenDrank gelijk aan "thee"

Het klikken op de Warm water-knop (versie 1)

maak GekozenDrank gelijk aan "warm water"

Het klikken op de Chocolademelk-knop (versie 1)

maak GekozenDrank gelijk aan "chocolademelk"

Ook de interface behoort theoretisch tot de toestand van het programma en dus moet ook worden nagedacht op wijzigingen die daarin moeten worden aangebracht. Feit is namelijk dat de gebruiker de waarde van GekozenDrank niet kan zien en dus niet weet of de knop goed is ingedrukt. Om onzekerheid en dus ongemak te voorkomen aan de kant van de gebruiker kan het geen kwaad deze te vertellen welke drank als gekozen wordt beschouwd. Dit kan b.v. een lichtje zijn (op een echte automaat in ieder geval), maar ook een mededeling op een display. Voor dit laatste kiezen we hier. Maar hoe geen we dit dan aan? Welke variabele moeten we dan veranderen? Ja, met kennis van Visual Basic zouden we weten dat het om de Caption-property van een label gaat, maar bij het ontwerpen hoeven we dat niet te weten. Eenvoudiger is het gewoon te zeggen welke tekst in het display moet worden weergegeven, dan moet de persoon die de implementatie schrijft maar bedenken hoe dit te realiseren. (Als dit principe ook was toegepast op het onthouden van de gekozen drank, dan hadden we daar gewoon kunnen schrijven : onthoud dat koffie/thee/warm water/chocolademelk als drank gekozen is.)

Natuurlijk is het mogelijk om de gebruiker te vertellen welke drank gekozen is, maar we willen de gebruiker ook laten weten wat hem die keuze kost en we hebben maar één display waarop slechts een beperkt aantal tekens kan worden weergegeven (alhoewel niet in onze specificatie vermeld). Laten we er hier voor kiezen om de gebruiker alleen te vertellen hoeveel dit hem kost. Die opdracht zou dus kunnen luiden in elk van de vier gevallen:

Vertel de gebruiker hoeveel (koffie/thee/warm water/chocolademelk)* hem kost

Of iets concreter:

Vermeld in het display hoeveel de gekozen drank kost

Deze laatste opdracht geeft al aan dat een bepaalde tekst in het display moet worden weergegeven, maar nog niet precies welke. Om welke tekst gaat het nou precies? Uiteindelijk moet er iets komen te staan in de trant "Te betalen: fl. x,xx." Voor x,xx moet dan het bedrag worden ingevuld dat de gekozen drank kost. Het programma moet derhalve weten hoeveel elke drank kost opdat de juiste tekst kan worden weergegeven. Kost koffie 25 cent, thee 15 cent, warm water 10 cent en chocolademelk 50 cent, dan krijgen we de volgende opdrachten als reactie op het aanklikken:

Het klikken op de Koffie-knop (versie 2)

maak GekozenDrank gelijk aan "koffie"
vermeld in het display "Te betalen: fl. 0,25."

Het klikken op de Thee-knop (versie 2)

maak GekozenDrank gelijk aan "thee"
vermeld in het display "Te betalen: fl. 0,15."

Het klikken op de Warm water-knop (versie 2)

maak GekozenDrank gelijk aan "warm water"
vermeld in het display "Te betalen: fl. 0,10."

Het klikken op de Chocolademelk-knop (versie 2)

maak GekozenDrank gelijk aan "chocolademelk"
vemeld in het display "Te betalen: fl. 0,50."

We zijn ervan uitgegaan dat er wanneer de drank wordt gekozen geen tegoed is. Dit is alleen mogelijk als er nog geen toevoegingen zijn gekozen. Dit is een redelijk uitgangspunt, want het is onwaarschijnlijk dat een gebruiker eerst de hoeveelheid toe te voegen suiker en melk kiest en daarna pas een drank kiest. Maar niets weerhoudt de gebruiker ervan eerst op de extra- ja zelfs op de betaal-knoppen te drukken, ook al is dit op dat moment niet toepasselijk. Met een dergelijk handelen moet het programma ook rekening houden – de automaat moet dat ook. Er moet voorkomen worden dat de automaat dan fouten maakt. Vroeger zouden ze zeggen dat de gebruiker tegen zichzelf beschermd moet worden, maar dan ga je eigenlijk van uit dat de gebruiker iets verkeerds doet, dus het is beter te zeggen dat de automaat tegen zichzelf beschermd moet worden. Nu is het in de praktijk niet mogelijk te voorkomen dat op een bepaalde knop wordt gedrukt, en het is ook zeker niet mogelijk om een knop onzichtbaar te maken zoals dat bij de simulatie op een computer wel kan. Er zijn twee alternatieve aanpakken: de knop uitzetten, zodat de reactie die gewoonlijk wordt vertoond op het moment dat de knop ingedrukt mag worden niet optreedt, of, in de reactie rekening houden met de mogelijkheid dat de knop is ingedrukt op een ongeldig tijdstip. Het uitzetten van een knop is verreweg het gemakkelijkst binnen een computerprgramma omdat de knop er dan vanuit kan gaan dat deze terecht is ingedrukt en er met ongeldig gebruik geen rekening hoeft te worden gehouden, maar of het op een echter te bouwen automaat ook gemakkelijker is weet ik niet. Hier kies ik er echter voor bepaalde groepen knoppen uit te schakelen, en je zult wel merken dat dat nog een hele klus is. Hier betekent dat, dat op het moment dat een drank kan worden gekozen, de extra- en betaalknoppen uit moeten staan. Maar ze moeten ook weer op het juiste moment worden aangezet nl. nadat een drank gekozen is. Dit betekent dus dat als aanvullende acties zowel de betaal- als extra-knoppen moeten worden aangezet. Dit geldt voor alle betaalknoppen maar niet voor alle extra-knoppen veronderstellende dat niet bij alle dranken dezelfde toevoegingen mogelijk zijn, alhoewel sommigen dat als betutteling van de gebruiker zouden zien. Laten we maar doen alsof deze keuze hier ter illustratie wordt gemaakt. Aan koffie en thee mag suiker worden toegevoegd, aan koffie, thee en chocolademelk melk. Dit geeft de volgende algoritmen voor de reactie van het klikken op de vier drankknoppen:

Het klikken op de Koffie-knop (versie 3)

```
maak GekozenDrank gelijk aan "koffie"
vermeld in het display "Te betalen: fl. 0,25."
zet de betaal-knoppen aan
zet de suiker- en melkknop aan
```

Het klikken op de Thee-knop (versie 3)

```
maak GekozenDrank gelijk aan "thee"
vermeld in het display "Te betalen: fl. 0,15."
zet de betaalknoppen aan
zet de suikerknop aan
```

Het klikken op de Warm water-knop (versie 3)

```
maak GekozenDrank gelijk aan "warm water"
vermeld in het display "Te betalen: fl. 0,10."
zet de betaalknoppen aan
```

Het klikken op de Chocolademelk-knop (versie 3)

```
maak GekozenDrank gelijk aan "chocolademelk"
vemeld in het display "Te betalen: fl. 0,50."
zet de betaalknoppen aan
zet de suikerknop aan
```

(Hoe het aanzetten in zijn werk moet gaan staat hier verder niet vermeld, en eigenlijk is het ook meer de taak van de implementator. Maar de ontwerper moet wel weten dat het kan (en niet per se hoe)!)

Reactie op het Kiezen van een toevoeging

Op de suiker- cq. melk-knop kan geklikt worden nadat een drank is gekozen. Bij het schrijven van de reactie hoeven we er geen rekening mee te houden dat de knop niet op het juiste moment wordt ingedrukt: dan staat ie immers uit en kan niet worden ingedrukt!

Wanneer een extra-knop wordt ingedrukt betekent dit dat aan deze extra aan de te leveren drank moet worden toegevoegd. Het is realistisch te veronderstellen dat hieraan kosten zijn verbonden. Het door de gebruiker te betalen bedrag neemt daardoor toe met een bepaald bedrag. Neem voor een portie suiker 5 cent en voor een portie melk(poeder) 10 cent. Een denkbaar algoritme voor de suiker-knop is dan:

Klikken op de suiker-knop (versie 1)

Verhoog het te betalen bedrag met 5 cent
Vermeld het te betalen bedrag in het display

Alleen: we weten het te betalen bedrag niet! Ja, het staat wel in het display, maar het is niet bijgehouden in een variabele! Natuurlijk kunnen we (in Visual Basic) het te betalen bedrag uit het display halen, maar dit is (in Visual Basic) veel moeilijker dan het tegoed gewoon onthouden. Iets makkelijker lijkt het om het te betalen bedrag af te leiden uit de gekozen drank, en die is wel onthouden, maar dit is geen goed idee, want dit werkt alleen de eerste keer als een extra is gekozen, en daarna niet meer (waarom?).
Vandaar dat een variabele zal moeten worden gebruikt, b.v. TeBetalen waarin zal worden onthouden wat de gebruiker moet betalen! In het algoritme voor de koffie/thee/warm water/chocolademelk-knop moet de opdracht Sla 0,25/0,15/0,10/0,50 op in TeBetalen worden toegevoegd. (De resulterende algoritmen worden hier niet herhaald.) Het algoritme voor de suiker-knop wordt dan:

Klikken op de suiker-knop (versie 2)

maak TeBetalen gelijk aan TeBetalen + 0,05
vermeld de tekst "Te betalen: fl. " gevolgd door
het te betalen bedrag in het display

Deze laatste opdracht staat nog niet volledig uitgewerkt. Het is overigens (in Visual Basic) niet moeilijk om twee teksten achter elkaar te plakken (met de operator &). Het is echter geen taak van de ontwerper om te bedenken hoe dit moet gebeuren.

Moet er verder nog iets aan de interface veranderen? Moeten er geen knoppen worden aan en uitgezet? Jawel, zodra een toevoeging is gekozen, mag geen drank meer worden gekozen. Dus:

Klikken op de suiker-knop (versie 3)

maak TeBetalen gelijk aan TeBetalen + 0,05
vermeld de tekst "Te betalen: fl. " gevolgd door
het te betalen bedrag in het display
zet de drankknoppen uit

Voor de melk-knop kan een vergelijkbaar algoritme worden opgesteld.

Reactie op Betalen

Zodra een drank is gekozen, kan de gebruiker gaan betalen. Door het indrukken van een betaal-knop neemt het te betalen bedrag af met het bedrag dat de knop representeert! Om redenen die nog duidelijk zullen worden gemaakt, kies ik ervoor om niet TeBetalen te verlagen, maar een variabele waarin wordt bijgehouden hoeveel betaald is, met de naam Betaald, op te hogen! Zodra Betaald groter is dan TeBetalen heeft de gebruiker betaald en kan de drank en eventueel wisselgeld worden verstrekt en worden uitgenomen. Hier is een voorlopig algoritme voor de stuiver-knop:

Klikken op de stuiver-knop (versie 1)

```
maak Betaald gelijk aan Betaald + 0,05
als Betaald groter dan of gelijk is aan TeBetalen, dan
        verstrek drank en/of wisselgeld
anders
        vermeld "Te betalen: fl. " gevolgd door
                TeBetalen-Betaald in het display
```

Dit algoritme is echter nog niet af! Op het moment dat de eerste keer betaald wordt, staan de extra-knoppen en mogelijkerwijs ook de drankknoppen nog aan! Het is redelijk om zodra het betalen begint de gebruiker niet meer toe te staan een andere drank of toevoegingen te kiezen (alhoewel de drank dan nog niet verstrekt is, dus misschien gaan we te ver!). Dit betekent dat de eerste keer dat betaald wordt de drank- en extra-knoppen moeten worden uitgezet. Alleen aan Betaald kunnen we zien dat dat het geval is, die is dan gelijk aan 0! Dus:

Klikken op de stuiver-knop (versie 2)

```
als Betaald = 0, dan
        zet dan de drank- en extra-knoppen uit
maak Betaald gelijk aan Betaald + 0,05
als Betaald groter dan of gelijk is aan TeBetalen, dan
        verstrek drank en/of wisselgeld
anders
        vermeld "Te betalen: fl. " gevolgd door
                TeBetalen-Betaald in het display
```

(Overigens: het is ook mogelijk om de drank- en extra-knoppen altijd uit te zetten, maar dan doen we in een aantal gevallen te veel, ofschoon de gebruiker van de applicatie er waarschijnlijk niets van zal merken, maar een dergelijk aanpak is minder efficient dan ik zou willen! Het voordeel van altijd uitzetten is natuurlijk wel dat we Betaald niet langer nodig hebben!)

Het algoritme lijkt nu af, maar we hebben nog helemaal niet beschreven wat we verstaan onder het verstrekken van drank en/of wisselgeld! Wat betekent dit voor de interface en de variabelen? Dit moet ook nog worden uitgewerkt in een algoritme maar dit hoeft niet in hetzelfde algoritme! Hiervoor maken we een apart algoritme, dat in Visual Basic in een aparte subroutine zal worden geplaatst.

Verstrekken betekent het verstrekken van drank en eventueel wisselgeld. Drank moet altijd worden verstrekt. Wisselgeld alleen als er meer betaald is dan betaald moest worden. We moeten echter ook nagaan wat er ter voorbereiding van het uitnemen moet gebeuren. Met wat nadenken vooraf kunnen we dat hier al ontdekken, i.p.v. later. Ten eerste moeten geen van de knoppen nog ingedrukt kunnen worden: er mag niet meer betaald worden! De extra- en drankknoppen zijn al uitgezet, nu moeten nog de betaalknoppen worden uitgezet. Het uitnemen houdt in dat zowel de drank moet worden uitgenomen, als eventueel wisselgeld. Zodra beide zijn uitgenomen, moet de

automaat dan weer in de begintoestand worden geplaatst. Het uitnemen kan op twee manieren geschieden: in een verplichte volgorde of in een door de gebruiker bepaalde volgorde. Het laatste verdient de voorkeur. Na het uitnemen van drank cq. wisselgeld moet de automaat na kunnen gaan of nu beide zijn uitgenomen. Dit controleren kan het gemakkelijkst plaatsvinden aan de hand van in variabelen opgeslagen waarden, één variabele waarin bijgehouden wordt of de drank al is uitgenomen en één waarin wordt bijgehouden of het wisselgeld al is uitgenomen. Deze variabelen noem ik DrankUitgenomen en WisselgeldUitgenomen, waarin de waarden Waar/Onwaar moeten kunnen worden opgeslagen: het zijn logische variabelen.
Een mogelijk algoritme:

Verstrek drank en eventueel wisselgeld

zet de betaalknoppen uit
verstrek drank
maak DrankUitgenomen gelijk aan Onwaar
als Betaald>TeBetalen, dan
    verstrek wisselgeld
    maak WisselgeldUitgenomen gelijk aan Onwaar
anders
    maak WisselgeldUitgenomen gelijk aan Waar

Het een waarde geven van Drankuitgenomen en WisselgeldUitgenomen kan ook in de algoritmen van 'verstrek drank' en 'verstrek wisselgeld'. Maar WisselgeldUitgenomen moet wel op Waar worden gezet als er geen wisselgeld uit te nemen is en dat zou dan weer wel hier moeten plaatsvinden, vandaar dat de algoritmen 'verstrek drank' en 'verstrek wisselgeld' niets meer doen dat wat ze beloven!
De moeilijkheid is nu wat nu precies onder het sec verstrekken van drank en wisselgeld verstaan moet worden! Al eerder is besloten een uit te nemen drank te tekenen, en de hoeveelheid uit te nemen wisselgeld alleen in de vorm van tekst te tonen. Welke interface-elementen dit precies moeten zijn doet nu nog niet ter zake, alleen dat ze beschikbaar zijn en dat de gebruiker er b.v. op kan klikken om het uitnemen na te bootsen. In het kader van dit voorbeeld is het niet nodig hier verder aan te geven hoe precies het verstrekken in zijn werk gaat. Op de computer kun je de implementatie vinden (ZEKA.VBP), en kun je bekijken hoe het verstrekken is gedaan. Het verstrekken van het wisselgeld is overigens heel eenvoudig aangepakt: slechts door in een label het wisselgeldbedrag te vermelden.

Reactie op uitnemen

De volgende stap is een algoritme te bedenken voor het uitnemen van drank en wisselgeld. We mogen daarbij niet uitgaan van een vaste volgorde! Dit betekent dat zowel na uitnemen van drank en wisselgeld gecontroleerd moet worden of beide zijn uitgenomen. Zo ja, dan moet de automaat weer in zijn begintoestand worden gebracht, d.w.z. teneinde een nieuw verzoek te kunnen afhandelen! Zo nee, dan moet verder nog niets gebeuren! Dus:

Klikken op de verstrekte drank (versie 1)

maak DrankUitgenomen gelijk aan Waar
verwijder de afbeelding van de verstrekte drank
als WisselgeldUitgenomen Waar is, dan
    breng de automaat weer in zijn begintoestand

Klikken op het verstrekte wisselgeld (versie 1)

maak WisselgeldUitgenomen gelijk aan Waar
verwijder het wisselgeld
als DrankUitgenomen Waar is, dan
    breng de automaat weer in zijn begintoestand

Dit lijkt goed, en dat is het ook, maar het kan nog iets beter. Immers: als WisselgeldUitgenomen Waar is, dan hoeft DrankUitgenomen niet per se gelijk aan Waar te worden gemaakt, want het wisselgeld is al uitgenomen en dan is het voldoende de automaat in zijn begintoestand te brengen. In die toestand maakt het nl. niet uit wat de waarde van DrankUitgenomen of WisselgeldUitgenomen is, omdat deze bij het verstrekken op de juiste beginwaarde worden gezet. Het is overigens net zo goed mogelijk om DrankUitgenomen en WisselgeldUitgenomen bij het in de begintoestand brengen van de automaat gelijk aan Onwaar te maken; dan hoeft alleen WisselgeldUitgenomen gelijk aan Waar te worden gemaakt als er geen wisselgeld hoeft te worden verstrekt! Overigens: het verwijderen van de afbeelding of het wisselgeld moet wel altijd gebeuren, omdat de gebruiker anders bij het eerste wat hij uitneemt niet ziet dat het uitnemen gelukt is.

Klikken op de verstrekte drank (versie 2)

```
verwijder de afbeelding van de verstrekte drank
als WisselgeldUitgenomen Waar is, dan
        breng de automaat weer in zijn begintoestand
anders
        maak DrankUitgenomen gelijk aan Waar
```

Klikken op het verstrekte wisselgeld (versie 2)

```
verwijder het wisselgeld
als DrankUitgenomen Waar is, dan
        breng de automaat weer in zijn begintoestand
anders
        maak WisselgeldUitgenomen gelijk aan Waar
```

Rest nog het in de begintoestand brengen van de automaat. Hiertoe moeten we nagaan in welke toestand de automaat zich momenteel bevindt en in welke toestand deze zich moet bevinden bij aanvang. Eerst maar de toestand van de interface. Na het uitnemen staan alle knoppen uit, maar een eventuele wisselgeldtekst staat er nog, alsook de afbeelding van de drank! Ook in de display staat nog een tekst weergegeven. In de begintoestand zijn is geen afbeelding zichtbaar en ook geen wisselgeld-tekst; en kan in het display de tekst "Kies een drank." staan. Bovendien moeten van de knoppen de drankknoppen aan staan! De rest kan blijven zoals ie is. Dus:

De automaat in de begintoestand brengen

```
plaats "Kies een drank." in de display
zet de drankknoppen aan
verwijder de wisselgeldtekst
verwijder de afbeelding van de getoonde drank
```

Het laatste wat we ons moeten afvragen is of de automaat de eerste keer ook in de goede begintoestand bevindt. De interface-maker moet er dan voor hebben gezorgd dat alle knoppen behalve de drankknoppen uit staan, en dat de tekst "Kies een drank." In het display zichtbaar is, dat geen wisselgeldtekst zichtbaar is, en dat geen afbeelding van een verstrekte drank zichtbaar is.

7.4.3 De implementatie

7.4.3.1 Implementatie van de interface

De eerder getoonde interface hadden we natuurlijk al stiekem met Visual Basic gemaakt. Het venster heeft de naam frmZEKA gekregen, met als titel "Een Zeer Eenvoudige Koffie-Automaat". De drankknoppen zijn in een control array met de naam cmdDrank geplaatst met als index 0, 1, 2 en 3. De extra-knoppen in de control array cmdExtra met als index 0 en 1 en de betaalknoppen in de control array cmdMunt met als index 0, 1, 2 en 3. De display heeft de naam lblDisplay, de picture box waarin een afbeelding van de uit te nemen drank wordt getoond de naam pctDrank en de label waarin het wisselgeldbedrag zal worden geschreven lblWisselgeld.

> Opmerking: de eerste drie letters van de naam van het control geven aan om wat voor soort control het gaat (lbl staat voor label, cmd voor command button, pct voor picture box). Dit doen we opdat we snel weten om wat voor soort object het gaat. We verwachten dat je in het practicum hetzelfde doet.

We moeten ervoor zorgen dat bij het opstarten alle knoppen behalve de drankknoppen uitstaan. Dit doen we door de Enabled-property van alle knoppen die uit moeten staan gelijk aan False te maken in het Properties Window (dus niet met code!). Aanzetten hoeft niet want bij verstek staat de Enabled-property op True. Op deze manier uitzetten betekent dat de knop zichtbaar blijft, maar dat aan de knop te zien is, dat deze uit staat, en dat op indrukken niet zal worden gereageerd. Wordt op een knop gedrukt met de muis, dan treedt het Click-event van die knop op, tenminste als de knop aan staat. De reactie op het indrukken van de knop moeten we vermelden in de bijbehorende eventprocedure, waarvan de naam altijd begint met de naam gevolgd door een liggend streepje (_) gevolgd door de naam van het event.

Het is verstandig vóór het schrijven van de eventprocedures, eerst alle te gebruiken variabelen te verzamelen en te declareren. Declareren doe je in het declaratie-gedeelte van het venster met het Dim-statement. In een declaratie moeten we ook opgeven om wat voor variabele het gaat, d.w.z. van welk type deze is. Welk type je kiest hangt af van het soort waarden dat in die variabele zal worden opgeslagen. Wil je in een variabele alleen (kleine) gehele getallen opslaan, dan kies je als type Integer (Engels voor geheel). Wil je tekst in een variabele opslaan dan neem je als type String. Wil je Waar/Onwaar in een variabele opslaan dan kies je als type Boolean. Alleen gebruik je in Visual Basic niet de woorden Waar en Onwaar maar de Engelstalige equivalenten True en False.

Een lijstje van de variabelen is snel gemaakt: GekozenDrank, TeBetalen, Betaald, DrankUitgenomen en WisselgeldUitgenomen. In GekozenDrank wordt de naam van de gekozen drank opgeslagen, dit is een tekst, vandaar dat we als type ervoor String kiezen. TeBetalen en Betaald zijn bedragen; dit kunnen variabelen van het type Single zijn, waarin reële getallen kunnen worden opgeslagen, maar door in centen te gaan rekenen, kunnen we ze van het type Integer laten zijn. DrankUitgenomen en WisselgeldUitgenomen zijn beslissingsvariabelen die we van het type Boolean laten zijn. Gebruiken we dezelfde conventie als bij de naamgeving van de controls dan geeft dit de volgende declaraties (te plaatsen in het declaratiegedeelte van het frmZEKA):

```
Dim strGekozenDrank As String
Dim intTeBetalen As Integer
Dim intBetaald As Integer
Dim blnDrankUitgenomen As Boolean
Dim blnWisselgeldUitgenomen As Boolean
```

7.4.3.2 Implementatie van de interactie

Reactie op het Kiezen van een drank

Een van de reacties van de drankknoppen is het aanzetten van de betaalknoppen. Deze opdracht is niet direct met één Visual Basic-opdracht te realiseren en kan in Visual Basic het beste als procedure worden gerealiseerd, met een toepasselijke naam zodat de implementatie lijkt op het ontwerp, d.w.z. dezelfde architectuur heeft. Het aanzetten van de betaalknoppen zou b.v. in de procedure ZetBetaalknoppenAan kunnen geschieden met de opdrachten

```
cmdMunt(0).Enabled=True ' aanzetten van de Stuiver-knop
cmdMunt(1).Enabled=True ' aanzetten van de Dubbeltje-knop
cmdMunt(2).Enabled=True ' aanzetten van de Kwartje-knop
cmdMunt(3).Enabled=True ' aanzetten van de Gulden-knop
```

en dat werkt prima, maar het is nog beter om gebruik te maken van het feit, dat we deze knoppen in een control array hebben ondergebracht elk met een eigen index (=plaatsnummer). We kunnen een onvoorwaardelijke lus gebruiken om hetzelfde te doen. Dan hebben we wel een extra variabele nodig om over de waarden die de index aan kan nemen te tellen. Dit mag een lokale variabele zijn, die dus buiten de procedure niet gebruikt mag worden, maar dat is niet erg. Laten we deze intTeller noemen van het type Integer dus. Bovenstaande opdrachten zijn dan te vervangen door:

```
Dim intTeller As Integer
    For intTeller=0 To 3
        cmdMunt(intTeller).Enabled = True ' aanzetten betaalknop intTeller
    Next intTeller
```

met dezelfde werking. Het is belangrijk in te zien, dat de waarde van de teller gebruikt wordt om verschillen in de opeenvolgende opdrachten op te vangen. In een lus gaat het dus niet altijd om per se om dezelfde opdrachten, maar kan het – en zal het ook vaak - om opdrachten gaan die iets van elkaar veranderen. De truc is het dan te ontdekken waarin ze verschillen en dan de tellervariabele te gebruiken om het verschil te realiseren.
Zouden we bovenstaande opdrachten in de procedure ZetBetaalknoppenAan plaatsen, dan zouden we nog een andere procedure nodig hebben om de knoppen uit te zetten met daarin de opdrachten:

```
Dim intTeller As Integer
    For intTeller=0 To 3
        cmdMunt(intTeller).Enabled = False ' uitzetten betaalknop intTeller
    Next intTeller
```

Hierin is True door False vervangen, meer niet. Nu is er een gemakkelijke manier om beide procedures door slechts één procedure te vervangen, nl. door aan Enabled toe te kennen waarde als parameter aan die procedure door te geven en die dan i.p.v. True of False te gebruiken. Moeten de drankknoppen worden aangezet, dan geef je als actuele parameter True door, bij uitzetten geef je False door.
Het enige verschil tussen deze opdrachten en de andere is het woordje True. Die procedure zouden we ZetDrankknoppen kunnen noemen, en die zou er als volgt uitzien:

```
Sub ZetDrankknoppen(ByVal blnAanOfUit As Boolean)
Dim intTeller As Integer
    For intTeller=0 To 3
        cmdDrank(intTeller).Enabled=blnAanOfUit ' zetten knop intTeller
    Next intTeller
End Sub
```

Tussen de haakjes staat de fomele parameter, hier met de naam blnAanOfUit. Tijdens het specificeren van ZetDrankknoppen is natuurlijk nog niet bekend met welke actuele waarde ZetDrankknoppen zal worden aangeroepen. De formele parameter fungeert als variabele die de waarde die aan de procedure wordt doorgegeven krijgt in de aanroep. Binnen de procedure kan met deze formele parameter worden gewerkt als was het een gewone (lokale) variabele. Het keyword Byval geeft aan dat blnAanOfUit een waarde-parameter is d.w.z. de beginwaarde van blnAanOfUit is de waarde die aan de procedure in de aanroep wordt doorgegeven. Is de bijbehorende actuele parameter de naam van een variabele, dan wordt deze als expressie beschouwd (en dus geëvalueerd d.i. uitgerekend), zodat de waarde van de variabele aan de procedure wordt doorgegeven, en niet de variabele zelf.
Wordt ByVal weggelaten, dan wordt de formele parameter een variabele-parameter genoemd, en moet een variabele aan de procedure worden doorgegeven, die dan door de procedure kan, maar niet per se hoeft te, worden gewijzigd. In de procedure stelt de formele parameter dan niet langer een lokale variabele voor, maar een externe variabele. Een waarde-parameter stelt daarentegen een lokale variabele voor wiens beginwaarde in de aanroep staat, die ook wel kan worden gewijzigd, maar de doorgegeven waarde verandert nooit!
De procedure ZetDrankknoppen kan aan het venster worden toegevoegd met Tools|Add Procedure. Maak deze procedure Private, want deze wordt immers alleen door frmZEKA gebruikt!
Na deze uitweiding die ons de procedure ZetDrankknoppen heeft opgeleverd, zijn we in staat de reactie van de drankknoppen te maken. Door te dubbelklikken op een van de knoppen komen we in het skelet van de eventprocedure terecht. Er staat Sub cmdDrank_Click(Index As Integer) boven, ongeacht op welke van de drankknoppen we dubbelklikken. Hoe weten we nou om welke knop het gaat? De reactie verschilt immers! Antwoord: de waarde van Index is de index van de ingedrukte knop. Druk je de Koffie-knop in, dan zal Index gelijk aan 0 zijn, druk je de ChocoladeMelk-knop in dan zal Index gelijk aan 3 zijn. We moeten de waarde van Index gebruiken om onderscheid te maken tussen de knoppen. Dit kan met:

```
If Index=0 Then
    strGekozenDrank="Koffie": intTeBetalen=25
    lblDisplay.Caption="Te betalen: fl. 0," & intTeBetalen & "."
    ZetBetaalknoppen(True)              ' aanzetten betaal-knoppen
    cmdExtra(0).Enabled=True            ' aanzetten melk-knop
    cmdExtra(1).Enabled=True            ' aanzetten suiker-knop
ElseIf Index=1 Then
    strGekozenDrank="Thee": intTeBetalen=15
    lblDisplay.Caption="Te betalen: fl. 0," & intTeBetalen & "."
    ZetBetaalknoppen(True)              ' aanzetten betaal-knoppen
    cmdExtra(1).Enabled=True            ' aanzetten suiker-knop
ElseIf Index=2 Then
    ... ' enzovoorts
```

(Wil je meedere opdrachten op één regel kwijt zet er dan een dubbele punt tussen. Gebruik zeker het keyword And niet, dit is een operator zodat er toch slechts sprake is van een enkele opdracht, die dan waarschijnlijk niet meer doet wat je wilt! Commentaar, achter enkele aanhalingstekens, werkt verduidelijkend, en ook met lege regels kunnen opdrachten worden gescheiden die minder met elkaar te maken hebben.)
Het kan echter met minder opdrachten: in alle gevallen moeten de betaal-knoppen worden aangezet. Het is dus mogelijk deze opdracht uit de If-Then-ElseIf-opdracht te halen. Het maakt niet uit of ie ervoor of erachter wordt gezet, omdat de volgorde waarin de(ze!) opdrachten moeten worden uitgevoerd er niet toe doet.
Dan rijst al gauw de vraag of het met nog minder opdrachten kan! En dat kan. Maar alleen door meer variabelen te gebruiken! Alle overige opdrachten zijn immers afhankelijk van de waarde van Index. Dit is de waarde die strGekozenDrank en intTeBetalen krijgen, en de extra-knoppen die moeten worden aangezet. De manier waarop strGekozenDrank en intTeBetalen moeten worden ingesteld is geen eenvoudige functie van Index, het gaat immers om een aantal onsamenhangende waarden. Deze zullen derhalve moeten worden onthouden. Dit kan in een array waarin voor elke mogelijke waarde van Index de bijbehorende drank resp. het te betalen bedrag kan worden opgeslagen. Aangezien Index de waarden 0, 1, 2 en 3 kan aannemen kunnen we de array's declareren met 0 als kleinst mogelijke index en 3 als grootst mogelijke index. Laten we de array voor de opslag van de 4 dranksoorten strDrankType en de aray voor de opslag van de te betalen centen intDrankKosten noemen. We voegen de volgende declaraties toe aan het declaratiegedeelte:

```
Dim strDrankType(0 To 3) As String
Dim intDrankPrijs(0 To 3) As Integer
```

(Er mag ook (3) i.p.v. (0 To 3) worden neergezet: als de kleinst mogelijke index gelijk aan 0 is, mag deze, en het keyword To, worden weggelaten uit de declaratie!)
Deze variabelen moeten wel een waarde krijgen voordat ze kunnen worden gebruikt. Dit hoeft slechts eenmalig te gebeuren. Het meest geschikte tijdstip is dat wanneer het form wordt ingelezen, geladen. Dan wordt de Form_Load-eventprocedure aangeroepen. Dubbelklikken op het form zelf (niet op een control) doet het skelet van de Form_Load-eventprocedure verschijnen. Zorg dat er staat:

```
Sub Form_Load()
    ' de namen van de beschikbare dranken
    strDrankType(0)="Koffie": strDrankType(1)="Thee"
    strDrankType(2)="Warme melk": strDrankType(3)="Chocolademelk"

    ' de prijzen van de beschikbare dranken
    intDrankKosten(0)=25: intDrankKosten(1)=15
    intDrankKosten(2)=10: intDrankKosten(2)=50
End Sub
```

Hoe zit het nu met het aan cq. uitzetten van de extra-knoppen en hun relatie met Index? Nu, melk moet kunnen worden toegevoegd als de gebruiker koffie koos d.i. als Index gelijk aan 0 is. Suiker als de gebruiker koffie, thee of chocolademelk koos, dus als Index gelijk aan 0, 1 of 3 is. De Enabled-property van de melk-knop, cmdExtra(1).Enabled, moet gelijk aan True worden gemaakt als Index gelijk aan 0 is, anders gelijk aan False. Dat kan heel gemakkelijk als je je realiseert, dat de waarde van Index=0 juist de waarde True heeft als Index gelijk aan 0 is, en anders False. Makkelijker dan testen of Index 0, 1 of 3 is, is testen of Index ongelijk aan 2 is met de ongelijk-operator <>.

De opdrachten die we eerder nodig hadden kunnen nu worden vervangen door:

```
Sub cmdDrank_Click(Index As Integer)
    strGekozenDrank=strDrankType(Index)
    intTeBetalen=intDrankPrijs(Index)
    lblDisplay.Caption="Te betalen: fl. 0," & intTeBetalen & "."
    ZetBetaalKnoppen(True)
    cmdExtra(0).Enabled=(Index=0)
    cmdExtra(1).Enabled=(Index<>2)
End Sub
```

(De haakjes om Index=0 en Index<>2 zijn niet nodig maar bevorderen de leesbaarheid!)

Reactie op het Kiezen van toevoegingen

Dit eenmaal wetende is het een stuk makkelijker om te bedenken wat er moet gebeuren om ook een minimum aan regels code nodig te hebben voor het klikken op de extra-knoppen. Hiertoe moet een extra array worden gedeclareerd waarin de van een toevoeging wordt opgeslagen. Noem deze intExtraPrijs en voeg de volgende declaratie toe aan het Declaratie-gedeelte:

```
Dim intExtraPrijs(0 To 1) As Integer
```

en de volgende opdrachten aan de Form_Load-eventprocedure:

```
intExtraPrijs(0)=5: intExtraPrijs(1)=10
```

eventueel van commentaar voorzien! Dan voldoet:

```
Sub cmdExtra_Click(Index As Integer)
    intTeBetalen=intTeBetalen+intExtraPrijs(Index)
    lblDisplay.Caption="Te betalen: fl. " & (intTeBetalen\100) & "," & _
                        Format$(intTeBetalen Mod 100,"00") & "."
    ZetDrankknoppen(False)
End Sub
```

waarin ZetDrankknoppen kan op dezelfde wijze als ZetBetaalknoppen is gemaakt en het liggend streepje aangeeft dat de rest van de opdracht op de volgende regel staat. Hierin valt onmiddellijk de expressie op die voor het weergeven van het te betalen bedrag zorgt. Bij het indrukken van de drankknoppen stond er nog gewoon "Te betalen: fl. 0," & intTeBetalen. Toen was het te betalen bedrag nog kleiner dan één gulden, nu niet meer! Daarom moet, om het aantal guldens in intTeBetalen te pakken te krijgen intTeBetalen geheeltallig door 100 worden gedeeld met \ en moet, om het aantal resterende centen in intTeBetalen te pakken te krijgen intTeBetalen Mod 100 worden uitgerekend. Om ervoor te zorgen dat 5 als 05 wordt weergegeven gebruiken we de Format$-functie waaraan als tweede actuele parameter de format string "00" wordt doorgegeven om ervoor te zorgen dat het getal altijd met twee cijfers zal worden weergegeven.

Reactie op Betalen

De reactie op het indrukken van een betaal-knop moet in de eventprocedure cmdMunt_Click worden geplaatst. Ook hier stelt Index weer de index van de ingedrukte knop voor, 0 bij de stuiver-knop, 1 bij de dubbeltje-knop, enz. Het zal nu wel duidelijk zijn dat we weer naar verschillen gaan kijken die bestaan tussen de uit te voeren code voor de verschillende waarden van Index. We zien dan dat er alleen verschil is tussen de waarde van het muntstuk waarmee Betaald moet worden opgehoogd. De vier waarden plaatsen we daarom weer in een array, die we b.v. intMuntWaarde zullen noemen en in het declaratiegedeelte gedeclareerd kan worden met:

```
Dim intMuntWaarde(0 To 3) As Integer
```

En in Form_Load kan worden ingesteld op:

```
intMuntWaarde(0)=5: intMuntWaarde(1)=10
intMuntWaarde(2)=25: intMuntWaarde(3)=100
```

De onder de betaal-knoppen te plaatsen code wordt dan:

```
If intBetaald=0 Then
   ZetDrankKnoppen(False)
   ZetExtraKnoppen(False)
End If
intBetaald=intBetaald+intMuntWaarde(Index)
If intBetaald>=intTeBetalen Then
   VerstrekDrankEnWisselgeld
Else
   lblDisplay.Caption="Nog te betalen: fl. " & _
                      (intTeBetalen-intBetaald)\100 & "," & _
                      Format$((intTeBetalen-intBetaald) Mod 100,"00") & "."
End If
```

waarin ZetExtraKnoppen op vergelijkbare wijze werkt ZetDrankKnoppen en ZetBetaalKnoppen. In de laatste opdracht wordt intTeBetalen-intBetaald twee keer uitgerekend. Een keer berekenen was ook mogelijk geweest, maar dan hadden we de waarde ervan in een variabele moeten opslaan. Dat is overigens niet erg, maar het gaat slechts om een eenvoudige berekeningen die snel genoeg kan worden uitgevoerd. De afweging opslaan (onthouden) of (opnieuw) uitrekenen zul je wel vaker moeten maken. Onthouden van globale variabelen kost extra geheugen, maar van lokale variabelen alleen tijdens de werking van de subroutine waarin ze voorkomen, dus dat is niet zo erg. Uitrekenen kost altijd tijd, en vooral als dat veel tijd is kan het programma te langzaam worden. Voor het practicum maakt het niet zo veel uit welke keuze je maakt, want het gaat om eenvoudige berekeningen en programma's die niet per se snel hoeven te zijn of veel geheugen gebruiken.

```
Sub VerstrekDrankEnWisselgeld()
   ZetBetaalKnoppen(False)
   blnDrankUitgenomen=False
   If intBetaald>intTeBetalen Then
         VerstrekWisselgeld
         blnWisselgeldUitgenomen=False
   Else
         blnWisselgeldUitgenomen=True
   End If
   VerstrekDrank
End Sub
```

VerstrekDrank wordt hier niet verder uitgewerkt. VerstrekWisselgeld wordt

```
lblWisselgeld.Caption = "Wisselgeld: fl. 0," & _
                        Format$(intBetaald-intTeBetalen,"00") & "."
```

We hoeven niet te delen: er is altijd minder dan een gulden wisselgeld!

Reactie op Uitnemen

Klikken op de verstrekte drank:

```
Sub pctDrank_Click()
    pctDrank.Cls ' verwijderen van de afbeelding
    If blnWisselgeldUitgenomen=True Then
        ZetAutomaatInBeginStand
    Else
        blnDrankUitgenomen=True
    End If
End Sub
```

Klikken op het verstrekte wisselgeld:

```
Sub lblWisselgeld_Click()
    lblWisselgeld.Caption=""          ' leegmaken wisselgeldlabel
    If blnDrankUitgenomen=True Then ' drank al uitgenomen
        ZetAutomaatInBeginStand
    Else ' onthouden dat het wisselgeld is uitgenomen
        blnWisselgeldUitgenomen=True
    End If
End Sub

Sub ZetAutomaatInBeginStand()
    lblDisplay.Caption="Kies een drank."   ' gebruiker verzoeken
    ZetDrankknoppen(True)                  ' drankknoppen aanzetten
    pctDrank.Cls                           ' roep Cls aan voor leegmaken
End Sub
```

7.4.3.3 Testen

Voordat we het programma op gebruikers loslaten moeten we eerst een zekere mate van vertrouwen hebben in de juiste werking van het programma. Zelfs in het zorgvuldigst ontworpen programma kunnen nog fouten en onvolkomenheden voorkomen. Testen betekent het programma gebruiken teneinde fouten en onvolkomenheden in de werking te ontdekken. Liefst testen we alle manieren waarop het programma kan worden gebruikt, d.w.z. kijken of de reactie van het programma op alle mogelijke gebruikswijzen klopt. Dit is voor grote programma's vaak ondoenlijk. Zelfs voor een klein programma als in dit voorbeeld ontworpen is het onmogelijk om alle mogelijke gebruikersacties te testen. Het aantal keer dat een gebruiker b.v. een drank kan kiezen is al oneindig (1, 2, 3, etc.). Op basis echter van kennis van de werking van het programma kan aannemelijk worden gemaakt, dat het niet nodig is meer dan twee keer opeenvolgend een drank te kiezen om te kijken of het te betalen bedrag niet per ongeluk wordt opgehoogd.
Naast kennis omtrent de werking van het programma kan ook gebruik worden gemaakt van kennis m.b.t. de te verwachten gebruikswijze van het programma! Dit kan immers betekenen dat niet alle mogelijkheden van het programma worden benut. Het is dan niet zinvol mogelijkheden van het programma te testen die toch niet worden gebruikt! Twee kanttekeningen:

1. Kennis omtrent het programma is altijd vermeende kennis. Je loopt het risico dat je vooroordelen hebt over de werking zodat je bepaalde testen overslaat, waarbij je net bepaalde fouten mist.
2. Hetzelfde geldt voor kennis van de gebruiker. Wie zegt dat de gebruiker niet een bepaalde gebruikswijze zal kiezen moet daar wel zeker van zijn. Bovendien: 'de' gebruiker bestaat niet!

Soms kan volstaan worden met gewoon maar met het programma werken en waarnemen wat er gebeurt. Dat is hier ook gebeurd en een aantal onvolkomenheden zijn aan het licht gekomen. Er zijn vaak verschillende manieren om de fout of onvolkomenheid weg te nemen. Wanneer opdrachten moeten worden toegevoegd moeten die daar worden geplaatst waar ze het toepasselijkst zijn. Meestal kan uit meerdere plaatsen worden gekozen, elk met hun specifieke voor- en nadelen. Er zijn niet echt regels voor aan te geven. Door ervaring ontstaat een bepaald gevoel voor waar zo'n opdracht thuishoort.

Onvolkomenheid m.b.t. het uitnemen

De gebruiker kan op pctDrank en lblWisselgeld blijven klikken ook op andere momenten. De Enabled-property ervan is immers voortdurend True! Dat geeft soms ongewenste effecten (ga na welke!) en dit moet worden voorkomen. Een dergelijke tekortkoming, die een logische fout wordt genoemd, dient bij het testen van het programma te worden ontdekt en gecorrigeerd!
Het makkelijkste is het om de Enabled-properties aan het begin gelijk aan False te maken en gelijk aan True na het verstrekken en gelijk aan False als het wisselgeld resp. de drank wordt uitgenomen. Het op True zetten kunnen we in VerstrekDrank resp. VerstrekWisselgeld plaatsen. Op False zetten bij het reageren op het klikken op het wisselgeld cq. de afbeelding.

Onvolkomenheid m.b.t. het betalen

Deze fout zou niet aan het licht komen als maar steeds een drankje wordt uitgenomen. Het blijkt nl. dat intBetaald steeds groter wordt. Immers: intBetaald wordt in cmdMunt_Click alleen maar steeds opgehoogd; ergens in het programma moet intBetaald weer op 0 worden gezet. Er zijn verschillende plaatsen waar dit kan worden gedaan. Het kan in ZetAutomaatInBeginStand. Maar het kan ook elke keer als opnieuw een drank is gekozen gelijk met het een startwaarde geven van intTeBetalen. Mijn voorkeur gaat uit naar ZetAutomaatInBeginStand aangezien hier datgene moet gebeuren dat de automaat klaar maakt voor een volgende klant. Daarin zult U dan ook de opdracht intBetaald=0 aantreffen. Er staat duidelijk aangegeven dat het om een toevoeging gaat. Ook is een duidelijke identificatie toegevoegd, hier een datum, zodat aangebrachte wijzigingen later gemakkelijk terug te vinden zijn.
Gaat het daarna goed? Ja, maar dit komt, omdat Visual Basic alle getalvariabelen een startwaarde 0 geeft. Dit betekent dat intBetaald na het opstarten gelijk aan 0 is, zoals gewenst. Was dat niet het geval dan zou intBetaald bij het opstarten wellicht niet 0 kunnen zijn (dat hangt af van de min of meer toevallige inhoud van de geheugenruimte die door intBetaald wordt gebruikt), en zou het programma soms wel en niet goed werken! Er is dus veel voor te zeggen om intBetaald bij het kiezen van een drank op 0 te zetten, dat is altijd veilig. Natuurlijk is het ook mogelijk om intBetaald in Form_Load gelijk aan 0 te maken, dan hebben we het probleem ook opgelost. Dit doen we niet omdat we Visual Basic kennen, maar een beetje gevaarlijk is het wel!

Onvolkomenheden m.b.t. mededelingen

Wanneer de gebruiker betaald heeft, blijft het laatste nog te betalen bedrag in het display staan, totdat wisselgeld en drank zijn uitgenomen. Het zal vele gebruikers niet duidelijk zijn wat er dan van hen verwacht wordt. De gebruiker moet wachten met het uitnemen van de drank en eventueel wisselgeld tot deze verstrekt is/zijn. Het is zinvol een dergelijke mededeling in het display te zetten. Tevens is het zinvol na het uitnemen van drank zonodig aan te geven het wisselgeld uit te nemen, of andersom. Toevoegingen die hiervoor zorgen tref je aan in VerstrekDrankEnWisselgeld en pctDrank_Click en lblWisselgeld_Click.

7.5 Het leven van variabelen

Wanneer variabelen gedurende de gehele werking van het programma beschikbaar zijn, spreekt men van **statische variabelen**. Variabelen die tijdens de werking van het programma worden aangemaakt en vernietigd worden worden **dynamische variabelen** genoemd. Voor de creatie en vernietiging van dynamische variabelen bevatten programmeeromgevingen speciale opdrachten. Van welk type zijn dergelijke variabelen dan als ze gedeclareerd worden? In een aantal programmeeromgevingen is dat het type pointer (wijzer). Na declaratie van een dergelijke pointervariabele bevat deze nog geen verwijzing naar een waarde, is leeg zou je kunnen zeggen. In de opdracht tot aanmaken van een variabele tijdens de werking van het programma, een dynamische variabele dus, moet de pointervariabele die naar die variabele zal verwijzen moeten worden vermeld. De pointervariabele zelf kan dus best een statische variabele zijn. De waarde van de dynamische variabele kan via de pointervariabele worden gezet en gebruikt. De dynamische variabele kan weer worden vernietigd met een speciale opdracht daartoe, weer door vermelding van de pointervariabele daarin, waarna de pointervariabele weer leeg zal zijn.
Het voordeel van dynamische variabelen zal duidelijk zijn: tijdens de werking van het programma kan indien nodig nieuwe geheugenruimte voor de opslag van gegevens worden aangevraagd en gebruikt zolang als nodig is. Denk b.v. maar eens aan de regels in een document. Het aantal en de inhoud ervan veranderen tijdens de werking van de tekstverwerker steeds: er is dus steeds een wisselend aantal geheugenplaatsen nodig voor de opslag van het document. Tijdens de werking van het programma worden er regelmatig nieuwe regels aangemaakt en korter/langer gemaakt.

BOX 7.7 Dynamische variabelen in Visual Basic

Visual Basic kent – volgens sommigen gelukkig - geen pointervariabelen. Het is derhalve niet mogelijk zelf tijdens de werking van een Visual Basic programma nieuwe variabelen te creëren. Wel kent Visual Basic dynamische array's: tijdens de werking van het programma kunnen elementen aan het einde van de array worden toegevoegd of verwijderd. Ook kan de array met het **Erase**-statement helemaal leeg worden gemaakt.
Controls, zoals command buttons (knoppen) en lists (lijsten), kunnen echter wel - als uitbreiding van een control array - worden gecreëerd (met **Load**) en vernietigd (met **Unload**). (Dit zul je tijdens het practicum overigens niet doen!) Hetzelfde geldt voor de forms. Een form wordt automatisch aangemaakt wanneer een eigenschap van het form voor het eerst een waarde wordt gegeven of een methode van het form voor het eerst wordt aangeroepen. Het aanmaken kan ook zelf ter hand worden genomen door gebruik te maken van het Load-statement. Een form kan worden vernietigd met Unload. Dit gebeurt automatisch bij het sluiten van het venster. De controls die tijdens het ontwerpen van de interface op een form worden gezet worden bij het laden van het form automatisch door Visual Basic aangemaakt.

Naar een dynamische variabele kunnen vele verwijzingen bestaan. Een dynamische variabele mag eigenlijk pas worden vernietigd als er geen verwijzingen meer naar bestaan. Een goede programmeeromgeving zorgt hier voor. Het vernietigen van een verwijzing resulteert mag pas dan in de vernietiging van de dynamische variabele resulteren als het de enige verwijzing is. De programmeur moeten echter wel steeds zelf de verwijzingen (de inhoud van de pointers) vernietigen.
Een goede programmeeromgeving helpt zijn gebruikers door een dynamische variabele pas te vernietigen als alle verwijzingen ernaar zijn verdwenen en niet eerder. Een dergelijke faciliteit wordt **garbage collection** genoemd.

7.6 Organisatie in klassen

Ongeveer 10 jaar geleden werd een manier bedacht om programma's te organiseren niet in subroutines maar in **klassen** te organiseren. Een klasse is vergelijkbaar met een samengesteld gegevenstype, met dit verschil dat het niet alleen velden voor de opslag van waarden bevat, maar ook de op variabelen van dit gegevenstype toegestane bewerkingen (**operaties**). Dergelijke variabelen worden gewoonlijk **objecten** genoemd. Net zoals het mogelijk is variabelen van een bepaald gegevenstype te declareren, zo is het mogelijk objecten van een bepaalde klasse te declareren. Elk object behoort tot een klasse en wordt een **instantie** van die klasse genoemd.

Een programma bestaat daarbij op zeker moment uit een of meerdere samenwerkende objecten met elk hun eigen proces – een situatie die vaak beter overeenkomt met een realistisch model van de met het computerprogramma te simuleren werkelijkheid - en niet langer uit één enkel proces. Wanneer meerdere objecten tegelijkertijd actief kunnen zijn, eist het coordineren van de activiteiten van deze parallelle processen meer van de ontwerper. Overigens kunnen programma's waarin op elk moment slecht één proces actief zal zijn, ook heel goed m.b.v. objecten worden geimplementeerd.

Ontwikkelomgevingen die de maker in staat stellen een computerprogramma als een verzameling samenwerkende objecten te definiëren, en die bovendien nog een aantal – nog te beschrijven – eigenschappen bezitten worden **object-georiënteerd** genoemd.

Objecten zijn dynamische variabelen die naar wens kunnen worden aangemaakt en dan vernietigd. Net als alle variabelen zijn ze van het een of andere type, alleen wordt dat type een klasse genoemd. De declaratie van een klasse lijkt erg op de declaratie van de eerder besproken units en packages. Het bevat een interface met de declaratie van vrij toegankelijke gegevens en de definitie van beschikbare subroutines, en een implementatiegedeelte met lokale, niet-beschikbare gegevens en de implementatie van tenminste de in de interface vermelde subroutines. De in de implementatie genoemde – en voor de omgeving verborgen - variabelen worden echter **velden** (***fields***) of ***members*** genoemd, terwijl de – voor de omgeving - beschikbare subroutines **methods** of ***member*** **functies** worden genoemd. Subroutines die niet beschikbaar zijn, en dus niet in de interface worden genoemd worden ***utility*** **functies** genoemd.
Een klasse bevat daarnaast echter ook de definitie en specificatie van een method, waarmee objecten van die klasse kunnen worden aangemaakt. Een dergelijke method wordt een **constructor** genoemd. In sommige programmeeromgevingen kunnen meerdere constructors worden gedefiniëerd.
Tenslotte moet een klasse de specificatie van een method bevatten voor het vernietigen van een object van die klasse. Een dergelijke method wordt een **destructor** genoemd. Dit is er altijd maar één. Het is gebruikelijk om daarin de door het object gebruikte geheugenruimte vrij te geven.
De klasse definiëert zodoende zowel de toestand van elke instantie ervan, als het gedrag ervan.
Een klasse kan beschouwd worden als een gietmal voor het maken van objecten.

Een object-georiënteerde programmeeromgeving stelt speciale opdrachten ter beschikking voor het aanmaken en vernietigen van objecten vergelijkbaar met die voor dynamische variabelen, alleen moet hierbij niet de naam van de pointervariabele worden genoemd maar de naam van het object.

Voordat een object kan worden aangemaakt moet deze worden gedeclareerd. In tegenstelling tot statische variabelen betekent declareren van een object niet automatisch ook dat er geheugenruimte voor wordt gereserveerd: daartoe moet de aanmaakopdracht worden gegeven. Bij het aanmaken van het object wordt de constructor van de klasse aangeroepen; bij vernietigen van het object de destructor!

BOX 7.8 Objecten en klassen in Visual Basic

Feitelijk heb je al met objecten gewerkt in Visual Basic zonder het te weten: elk form is namelijk een object! Maar van welke klasse is zo'n form dan? Een form dat je tekent is eigenlijk ook een klasse! Het is de gietmal waaruit een form kan worden gemaakt. Bij het opstarten van de gemaakte applicatie maakt Visual Basic een exemplaar van de formklasse aan. Het is dus ook mogelijk om meerdere daarvan aan te maken! Hiervoor kent Visual Basic de **New**-operator: een nieuw exemplaar, zeg NogEenForm1, van Form1 kan worden aangemaakt met de opdracht: **Set NogEenForm1 = New Form1**. Vergeet dan niet NogEenForm1 te declareren met **Dim NogEenForm1 As Form1**.
NogEenForm1 kan vervolgens worden getoond met de opdracht **NogEenForm1.Show of NogEenForm1.Visible = True**, die ook het laden van het form voor zijn rekening neemt. Vernietigen van NogEenForm1 kan dan met: **Set NogEenForm1 = Nothing**.
Het is mogelijk zelf klassen te definiëren in Visual Basic. Een klasse definiëer je in een klasse-module. Maak je een klasse-module aan dan zijn er direct twee methoden beschikbaar om in te vullen nl. **Initialize** en **Terminate**. Dit zijn resp. de constructor en destructor van de klasse. De constructor wordt uitgevoerd als met de **New**-operator een object van die klasse wordt aangemaakt. De destructor als aan de objectvariabele de waarde **Nothing** wordt toegekend.
Na aanmaken van een object kunnen de methoden ervan worden aangeroepen en de publieke velden – de velden die met **Public** zijn gedeclareerd in de klassemodule – worden gebruikt.
Een methode roep je aan door de aanroep te beginnen met de naam van het object gevolgd door een punt gevolgd door de naam van de methode en de evt. parameterlijst. Roteren van een rechthoek over 45 graden zou dan b.v. kunnen met de opdracht **figRechthoek.Rotate(45)** ervan uitgaande dat het object FIGRECHTHOEK gedeclareerd is als **Dim figRechthoek As Figuur** en aangemaakt met **Set figRechthoek = New Figuur**.
Een publiek veld gebruik je op dezelfde wijze als een veld van een recordtype, door de naam van het veld achter de naam van het object te zetten met een punt ertussen. Stel je wilt weten onder welke hoek de gemaakte figuur geroteerd is. Als de hoek opgeslagen staat in het veld intRotatieHoek, dan is figRechthoek.intRotatieHoek de hoek waaronder figRechthoek geroteerd is.

Een klasse is dus zowel een speciaal soort unit als een speciaal soort gegevensstructuur (vgl. het beschouwen van licht als deeltje of als golf).
Een programma organiseren in klassen i.p.v. modules heeft een aantal voordelen waarvan ik er twee noem:

1. De velden zijn automatisch verborgen, ingekapseld (*encapsulated*) zoals dat heet, en hier hoeft niet over nagedacht te worden zoals dat bij modules het geval is.
2. Objecten zijn vaak gemakkelijk te herkennen in het domein van de toepassing, wat het gemakkelijk maakt om het programma op te bouwen uit objecten waarvan een equivalent in de 'echte wereld' bestaat, zoals een figuur in een grafische en een bankrekening in een financiele toepassing.

Data-inkapseling (***encapsulation***) zoals dat moet worden aangeleerd bij de organisatie in modules is bij het organiseren in klassen veel natuurlijker omdat de klassen toestand en gedrag van 'echte' objecten voorstellen. De maker hoeft slechts na te gaan welke objecten in het domeingebied voorkomen, wat gewoonlijk niet moeilijk is, en vervolgens de toestand en het gedrag in een klasse te beschrijven. Dit is meestal gemakkelijker dan het opsplitsen van de werking van het programma in functionele eenheden, waarbij de gemaakte keuze altijd een mate van willekeur bevat. Bij de organisatie in klassen wordt je gedwongen tot het maken van een goede keuze, terwijl dat bij de organisatie in modules niet vanzelfsprekend is en moet worden aangeleerd.
Kunnen gebruiken van klassen en objecten maken een natuurlijker ontwerpen mogelijk. Ze dwingen data-inkapseling af. Ze bevorderen hergebruik van code en maken programma's gemakkelijker aanpasbaar. Daarnaast maken ze een efficiënter ontwerpen mogelijk als nog een aantal extra mogelijkheden aanwezig zijn in de programmeeromgevingen. Efficiënter in de zin dat minder code nodig is dan anders. Dit kan worden bereikt als de programmeeromgeving de mogelijkheid tot **overerving** (***inheritance***) biedt. Een dergelijke programmeeromgeving staat het definiëren van subklassen toe. Een klasse kan worden gespecialiseerd door de definitie van een **subklasse**. Een subklasse bevat alle eigenschappen en het gedrag van de klasse waarvan het is afgeleid – de **superklasse** – en voegt daar nog eigen specifieke eigenschappen en gedrag aan toe.

Een eenvoudig voorbeeld is een hiërarchie van klassen bestaande uit de superklasse voertuig, diens subklassen gemotoriseerd en niet-gemotoriseerd voertuig, en daarvan de subklassen auto, motorfiets, step, fiets, driewieler, eenwieler enzovoorts. Een ander voorbeeld is die van de superklasse figuur, en de subklassen driehoek, rechthoek, vierkant, cirkel, ellips enzovoorts. Alle (soorten) figuren hebben een middelpunt en kunnen getekend, verwijderd, verplaatst en geroteerd worden (bij een cirkel overigens zonder betekenis).

In een programmeeromgeving met overerving kan dan het gemeenschappelijke gedrag van een aantal soorten objecten in een superklasse worden ondergebracht. Dit heeft dan als voordeel dat dit gedrag maar een keer beschreven hoeft te worden, en niet voor elke klasse objecten apart, waardoor minder code hoeft te worden geschreven met de bijbehorende voordelen. Methoden die in de superklasse voorkomen, kunnendoor de subklassen opnieuw worden gedefiniëerd, alhoewel dit niet altijd nodig is. Het verplaatsen van een figuur gebeurt altijd op dezelfde manier, met dezelfde methode zou je kunnen zeggen, nl. verwijderen, middelpunt verplaatsen en weer opnieuw tekenen. Die hoeft dan door de subklassen niet opnieuw te worden gedefiniëerd. Het tekenen echter wel want driehoeken en rechthoeken moeten beslist niet op dezelfde manier te worden getekend. Het tekenen moet dus wel opnieuw worden gedefiniëerd, het verplaatsen echter niet. Zodra b.v. een rechthoek de opdracht krijgt zichzelf te verplaatsen, wordt de methode tot verplaatsen van een figuur aangeroepen als beschreven in de klasse figuur. Deze zoekt dan uit om wat voor soort figuur het gaat en roept dan de juiste methode voor het (opnieuw) tekenen van dat soort figuur aan. De mogelijkheid om tijdens het draaien van het programma na te gaan welke methode moet worden aangeroepen afhankelijk van de klasse waartoe het object behoort, wordt **polymorfisme** ('meervormigheid') genoemd.

Polymorfisme is dus nodig om ervoor te zorgen dat een bepaalde methode – zoals het verplaatsen van een object – slecht eenmaal hoeft te worden gedefiniëerd. Zonder polymorfisme is hergebruik van code uit superklassen in een klasse-hiërarchie niet mogelijk.

In een programmeeromgeving zonder overerving moet meer code worden geschreven om dezelfde functionaliteit te realiseren omdat elke methode die klassen gemeen hebben in elke klasse moet worden gedefiniëerd. Wel kan – als polymorfisme aanwezig is – een methode worden aangeroepen zonder dat vooraf – voor het draaien van het programma – bekend hoeft te zijn op welke type object het moet worden toegepast, zolang als het object dat wordt gebruikt maar over die methode beschikt. Dit kan dan echter pas tijdens het draaien van het programma worden vastgesteld. Polymorfisme kan ook via het gebruik van **interfaces** worden gerealiseerd; dit is de aanpak zoals die in Visual Basic en Java wordt aangeboden. Aan een methode die objecten van een bepaalde interfaceklasse accepteert kunnen dan ook objecten worden doorgegeven die deze interface implementeren. Op die manier kunnen b.v. klassen die de interfaceklasse voertuig implementeren ook aan methoden die als invoerparameter objecten van het type voertuig accepteren ook objecten van die klasse worden doorgegeven. Het voordeel van interfaces boven vererving is dat een klasse meerdere interfaceklassen kan implementeren, terwijl bij vererving het moeilijk te realiseren is dat een subklasse meerdere superklassen heeft.

BOX 7.9 Object-oriëntatie in Visual Basic

Visual Basic mag (nog) niet als een volledig object-georiënteerde programmeeromgeving worden beschouwd omdat het niet over vererving beschikt. Polymorfisme kan wel via interfaces worden gerealiseerd.
In de vorige box is al beschreven hoe klassen en objecten moeten worden gemaakt. Er werd verteld dat publieke velden in een klasse op dezelfde manier konden worden gebruikt als de velden in een record. Een belangrijk nadeel van een dergelijke aanpak is dat geen controle op de juistheid van de waarde van het veld kan worden uitgeoefend. Dit is geen probleem als het b.v. om een tekst gaat die uit een willekeurig aantal tekens kan bestaan, maar wel als b.v. een temperatuur in een bepaald bereik moet worden opgegeven (b.v. altijd minstens gelijk aan het absolute nulpunt). Om controle op de juistheid van het gebruik mogelijk te maken kent Visual Basic properties ('eigenschappen'). Properties zien er naar de gebruiker ervan uit als velden, en ze kunnen op dezelfde manier worden gebruikt, maar in de klasse-module worden ze met een drietal speciale methoden beschreven. Dit zijn de Let- en Set-methode die worden aangeroepen als een property een waarde worden toegekend, en de Get-methode die wordt uitgevoerd als de waarde van een property moet worden afgegeven. De Set-methode wordt aangeroepen als de property een object is i.p.v. een waarde. De velden die gebruikt worden voor de opslag van de met de eigenschap bedoelde gegevens kunnen best heel anders heten. Een eenvoudig voorbeeld is dat waarbij een temperatuur moet worden gezet. De naam van de property is b.v. Temperatuur. Maar de waarde wordt bijgehouden in het veld intGraden door in de klasse-module te zetten (Properties kunnen met Add Procedure worden toegevoegd aan een klasse-module):

```
Private sngGraden As Single
Public Property Let Temperatuur(ByVal nNewValue As Single)
   sngGraden = nNewValue
End Property

Public Property Get Temperatuur() As Single
   Temperatuur = sngGraden
End Property
```

BOX 7.9 Object-oriëntatie in Visual Basic (vervolg)

De temperatuur van b.v. het object Huis instellen kan dan met:

```
Huis.Temperatuur = 20
```

En weergeven met:

```
MsgBox "De temperatuur in huis is momenteel " & _
        Huis.Temperatuur & " graden."
```

Het is mogelijk een klasse een event te laten genereren m.b.t. het object, net zoals dat het geval is bij controls en forms (wat overigens ook objecten zijn). In de klasse-module kan een event worden gedeclareerd met het Event-keyword en worden gegenereerd m.b.v. de standaardprocedure RaiseEvent. Daarachter komt dan de naam van het event te staan en eventuele parameters. Zo is het denkbaar dat bij een verandering van de temperatuur een event wordt opgewekt dat deze verandering doorgeeft:

```
RaiseEvent TemperatuurVerandering, sngGraden
```

Dit event moet dan in het declaratiegedeelte van de klasse-module worden gedeclareerd met:

```
Public Event TemperatuurVerandering(ByVal Verandering as Single)
```

Om nu een event Huis_TemperatuurVerandering te krijgen moet Huis gedeclareerd zijn met het WithEvents-keyword in de declaratie b.v. **Private WithEvents Huis As ....** Zet je deze declaratie in het declaratiegedeelte van een form, dan zal een event Huis_TemperatuurVerandering te vinden zijn in het Code-venster van het form waarin code kan worden geschreven.

Tenslotte is het nog mogelijk om events toe te voegen aan forms. Declareer deze met het keyword Event in het declaratiegedeelte van b.v. Form1 b.v. **Event NieuweTemperatuur**. Plaats in het declaratiegedeelte van de klasse-module de declaratie **Private WithEvents mForm1 As Form1** en voeg propertyprocedures toe die mForm1 instellen en opvragen. In het Code-venster van de klasse-module is nu het event mForm1_NieuweTemperatuur te vinden en de reactie van de klasse kan daarin worden geplaatst. Vergeet niet een parameter toe te voegen waarin de nieuwe temperatuur komt te staan. Vervolgens kan na het aanmaken van het object van die klasse in de Form_Load eventprocedure en het zetten van mForm1, het event NieuweTemperatuur worden gegenereerd – voor het doorgeven van een nieuwe temperatuur aan het object - in Form1 daar waar gewenst met RaiseEvent:

```
RaiseEvent NieuweTemperatuur, sngNieuweTemperatuur
```

Uiteraard kan dit ook rechtstreeks worden bereikt via propertyprocedures voor de temperatuur.

7.7 Een Zeer Eenvoudige Koffie-Automaat (Revisited)

Eerder is de simulatie van een zeer eenvoudige koffie-automaat gemaakt. Hierbij werd geen gebruik van objecten gemaakt. Het is echter logisch de werking van de automaat op te delen in user interface en interne werking. Dit betekent dat goed moet worden nagedacht over welk deel van toestand en gedrag nu tot de interne werking behoort en welk tot dat van de user interface. De interne werking realiseer ik in een klasse, de user interface in het form. Wanneer de opdeling naar behoren plaatsvindt, kunnen bepaalde aspecten van user interface en klasse gemakkelijker worden gewijzigd, zonder dat de ander hoeft te worden aangepast. (Dit is overigens niet helemaal waar, want het toevoegen van een extra te leveren drank betekent dat zowel de interface als de klasse moet worden aangepast, maar logisch: de klasse en de interface vormen samen de koffie-automaat!)

De definitie van een klasse gebeurt in Visual Basic door een klasse-module aan te maken met Project|Add Class Module. Deze geven we de naam clsZEKA. De klassemodule bevat nu al twee voorgedefiniëerde methoden nl. Class_Initialize en Class_Terminate die worden uitgevoerd als een instantie van de klasse wordt aangemaakt resp. verwijderd. (Helaas accepteert de Initialize-procedure geen parameters, zodat voor het doorgeven van bepaalde startgegevens een aparte methode moet worden gedefiniëerd.)

Laten we beginnen met na te gaan waar de gedeclareerde variabelen thuishoren. Alle variabelen die met de interne werking te maken hebben komen in aanmerking om als velden in de klasse te worden gedeclareerd. Dit zijn alle variabelen, inclusief de arrays, met uitzondering van blnDrankUitgenomen en blnWisselgeldUitgenomen. Het is wel mogelijk om ook het bijhouden van blnDrankUitgenomen en blnWisselgeldUitgenomen tot de interne werking te laten behoren, maar het is niet nodig want het is feitelijk de interface die die informatie nodig heeft, om te bepalen of alle goederen zijn uitgenomen.
De velden in de klasse zullen hetzij met Public of Private moeten worden gedeclareerd i.p.v. met Dim. Velden die met Public worden gedeclareerd kunnen rechtstreeks door het object dat van die klasse is gemaakt worden gebruikt en dus – zonder controle – worden gewijzigd. Het principe van informatie-verbergen vereist echter dat de toestand van een object alleen door methoden van de klasse kunnen worden gewijzigd. Door velden met en als Private te declareren zijn ze niet langer direct beschikbaar voor het object. Deze velden zijn dan alleen nog maar met opdrachten die in de klasse staan te wijzigen. Ook hier kiezen we voor deze benadering. We voegen derhalve de volgende declaraties aan de klassemodule toe (b.v. door ze uit het form te knippen):

```
Private strDrankType(3) As String      ' de beschikbare dranken
Private intDrankPrijs(3) As Integer    ' de prijzen van de dranken
Private intExtraPrijs(1) As Integer    ' de prijzen van de extra's
Private intMuntWaarde(3) As Integer    ' de waarde van de munten

Private intBetaald As Integer          ' wat betaald is
Private intTeBetalen As Integer        ' wat betaald moet worden
Private strGekozenDrank As Integer     ' naam van de gekozen drank
```

(Het type tPunt behoort tot de interface, omdat het daarin wordt gebruikt om het vullen van het bekertje te simuleren.)

De volgende stap is lastiger. Welk gedrag, tot nu toe in de interface opgeslagen, moet naar de klasse worden overgeheveld, en welke eigenschappen moeten in de klasse worden gedeclareerd?! En op welke events in de klasse moet door de interface worden gereageerd?

Er zijn twee manieren waarop een object de waarde van een van zijn Private-velden kan gebruiken. Hetzij door een methode aan te roepen die in de klasse als Public gedefiniëerd is (Met en als Private gedeclareerde methoden zijn per definitie niet beschikbaar buiten de klasse!), of door gebruik te maken van een eigenschap *(property).* Een eigenschap wekt naar buiten de indruk een veld te zijn, maar in de klasse moeten voor elke property een property Let-procedure worden gedefiniëerd om de eigenschap een waarde te kunnen geven, en een property Get-procedure om de waarde van de eigenschap op te kunnen halen. De waarde van de eigenschap kan in een veld opgeslagen staan maar dit hoeft niet. Alle communicatie zou in theorie via methoden plaats kunnen vinden maar het gebruik van eigenschappen is vaak natuurlijker voor het wijzigen of aanspreken van de toestand van een object.

Om te kunnen bepalen welke eigenschappen en methoden in de klasse moeten worden gedefiniëerd bepalen we eerst welke informatie-uitwisseling er tussen de interface en de klasse plaats zou kunnen vinden. Er zijn twee aanpakken denkbaar. Zo is het mogelijk alleen het hoogst noodzakelijke door de interface te laten doen: het uit/aanzetten van de knoppen, het doen van mededelingen en het verstrekken van drank en wisselgeld. De interface hoeft dan helemaal geen weet te hebben van de toestand van de klasse; is de slaaf van de klasse, die alle intelligentie van de automaat bevat. Het is dan de klasse die aan de interface doorgeeft welke mededelingen moeten gedaan. De klasse hoeft niet te weten – en kan ook beter niet weten – hoe de interface dit doet. En de interface hoeft niet te weten om welke informatie het gaat! Alle informatie die de interface nodig heeft kan als parameter in een RaiseEvent-aanroep aan de interface worden doorgegeven: er zijn dan geen eigenschappen nodig.

In het andere geval kent de interface wel de eigenschappen van de klasse en weet hoe deze te gebruiken. Krijgt de klasse b.v. opdracht weer te geven hoeveel er betaald moet worden, dan vraagt het aan het object hoeveel er betaald moet worden, door de waarde van de eigenschap TeBetalen op te vragen. Hoe deze informatie aan de gebruiker moet worden getoond bepaalt de interface. Het kiest b.v. de taal waarin dit moet worden meegedeeld. De interface maakt dan gebruik van kennis m.b.t. de gebruiker.

Het is logischer om kennis omtrent de gebruiker in de interface te plaatsen en niet in de klasse. Vandaar dat we hier kiezen voor de tweede aanpak. Dan moet worden bepaald welke informatie de interface nodig heeft van het object. Het is niet moelijk om in te zien dat het hier om alle in de niet object-georiënteerde versie van de automaat gedeclareerde variabelen, met uitzondering van de arrays, blnDrankuitgenomen en blnWisselgelduitgenomen, dus intBetaald, intTeBetalen en strGekozenDrank. Laten we voor het opvragen van de waarden ervan de properties Betaald, TeBetalen en Drank gebruiken. Betaald en TeBetalen zullen alleen door frmZEKA worden opgevraagd en niet gezet zodat we daarvoor geen Propert Let-procedure nodig hebben. Deze kunnen gewoon in het Codevenster van de klasse worden ingetypt, maar ook via Tools|Add Procedure als Public Property worden toegevoegd. De Property Let-procedures van Betaald en TeBetalen zijn overbodig en mogen worden verwijderd. Voeg daartoe aan de klassemodule toe:

```
Property Get Betaald() As Integer
    Betaald=intBetaald
End Property

Property Get TeBetalen() As Integer
   TeBetalen=intTeBetalen
End Property

Property Get Drank() As String
    Drank=strGekozenDrank
End Property

Property Let Drank(ByVal strDrank As String)
    strGekozenDrank=strDrank
End Property
```

Daar waar in de code van frmZEKA intBetaald staat komt straks een verwijzing naar de property Betaald te staan. Maar daartoe moeten we wel een object van het type clsZEKA declareren in frmZEKA die een verwijzing naar de koffie-automaat gaat bevatten. Laten we deze ckaZEKA noemen en in het declaratiegedeelte van frmZEKA als volgt declareren:

```
Dim ckaZEKA As clsZEKA
```

ckaZEKA kan pas worden gebruikt als deze wordt aangemaakt met New. Dit had in de declaratie gekund (met Dim ckaZEKA As New clsZEKA), maar hier kiezen we ervoor ckaZEKA aan te maken in Form_Load met:

```
Set ckaZEKA=New clsZEKA
```

Set en New zijn verplichte Visual Basic keywords. Het object ckaZEKA heeft een property Betaald, die met ckaZEKA.Betaald kan worden aangeduid. Nu kunnen we intBetaald, intTeBetalen resp. strGekozenDrank overal in frmZEKA vervangen door ckaZEKA.Betaald, ckaZEKA.TeBetalen resp. ckaZEKA.Drank.

Maar voordat we dat doen zullen we eerst eens bekijken welke code naar de klasse moet verhuizen en welke kan blijven. Het klikken, tekenen en weergeven van mededelen vindt natuurlijk altijd m.b.t. de interface plaats. Alle invoer moet aan het object worden doorgegeven, en wel door een methode ervan aan te roepen, en alle uitvoer die het object wenst moet aan de interface worden doorgegeven, en wel door een event op te wekken waarop de interface dan toepasselijk kan reageren.

Laten we beginnen met het opsplitsen van de reactie op het kiezen van een drank. De gebruiker klikt op een van de vier drankknoppen en de eventprocedure cmdDrank_Click wordt aangeroepen. Aan het object moet worden doorgegeven welke drank is gekozen. Nu is de Caption van de ingedrukte knop gelijk aan de naam van de gekozen drank zodat de naam van de gekozen drank aan het object kan worden opgegeven door de Drank-property gelijk te maken aan de Caption van de ingedrukte knop. Meer hoeft er hier niet te gebeuren. We weten immers – in principe – niet of de toekenning zal lukken. Het kan zijn dat we een waarde proberen toe te kennen die niet toegestaan is. We verwachten van de automaat dat die een event opwekt als de toekenning is gelukt. frmZEKA moet wel weten hoe dit event heet. Laten we dit event

DrankGekozen noemen. Voordat een dergelijk event beschikbaar kan komen in frmZEKA moeten er twee dingen gebeuren: het event moet in de klasse met het Event-keyword worden gedeclareerd en aan de declaratie van ckaZEKA moet het keyword WithEvents worden toegevoegd. Daarom plaatsen we de volgende regel in het declaratiegedeelte van de klassemodule:

```
Event DrankGekozen()
```

Bovendien moet, wanneer de property Drank met succes wordt ingesteld dit event worden opgewekt. De opdracht daartoe voegen we toe aan de Property Get-procedure van Drank:

```
RaiseEvent DrankGekozen
```

En wijzigen we de declaratie van ckaZEKA, in het declaratiegedeelte van frmZEKA, in:

```
Dim WithEvents ckaZEKA As clsZEKA
```

In cmdDrank_Click kunnen we volstaan met de toekenning

```
ckaZEKA.Drank=cmdDrank(Index).Caption
```

Natuurlijk: de rest van de reactie moet nu gecopieerd worden naar de eventprocedure ckaZEKA_DrankGekozen, die als het goed is nu beschikbaar is. Maar in de eerstvolgende opdracht wordt intTeBetalen gelijk gemaakt aan intDrankPrijs(Index). Zowel intTeBetalen als intDrankPrijs zijn hier niet beschikbaar, maar wel in de klasse! Dit betekent, dat in de Property Get-procedure van Drank ook intTeBetalen moet worden ingesteld, alleen beschikken we niet over Index, want die is alleen bekend in cmdDrank_Click. Omdat we met een eigenschap werken en niet met een methode kunnen we maar één waarde aan ckaZEKA doorgeven en dat moet een tekst zijn. Het is overigens geen probleem om Index aan ckaZEKA.Drank toe te kennen. Visual Basic weet dat Drank een tekst is en zal Index automatisch omzetten in een tekst (0 wordt "0", 1 wordt "1", etc.) Zelfs als VB dit niet deed, dan was het nog wel mogelijk door de conversiefunctie CStr te gebruiken (Cstr(0) is gelijk aan "0", etc.). Door niet de Caption van de knop maar Index door te geven, moet de Property Get-procedure van Drank echter wel worden aangepast. We krijgen niet langer de naam van de drank door maar het nummer van de knop, maar niet als getal maar als tekst. Dan is strGekozenDrank=strDrank geen goede opdracht meer, omdat dan strgekozenDrank gelijk wordt aan "0", "1", ... i.p.v. "Koffie", "Thee", ... Dan moet strDrankType worden gebruikt om weer de goede tekst in strGekozenDrank te krijgen! De opdracht strGekozenDrank=strDrankType(strDrank) accepteert VB echter niet omdat strDrank een tekst is, en geen getal. Deze keer zet VB strDrank niet automatisch in een getal om, dus moet er een conversiefunctie worden gebruikt. In dit geval is dit Cint die van een tekst het bijbehorende getal maakt.
Zou het dan niet makkelijker zijn om het type van de parameter strDrank van String te veranderen in Integer? Dit is alleen mogelijk als Drank een Integer-property zou zijn, d.w.z. als ook het resultaat van de Property Let-procedure van het type Integer is. Dan zou Drank niet langer een String-property zijn maar een Integer-property. Met behulp van strDrankType is het gemakkelijk om de naam van de drank bij een bepaald dranknummer te bepalen in de Property Let-procedure van Drank en die in strGekozenDrank op te slaan. Het is echter iets moeilijker om uit strGekozenDrank weer opnieuw het nummer te bepalen, dat de Property Get-procedure moet retourneren. Dan is het nodig te testen wat strGekozenDrank is, en als deze Koffie is 0, als deze Thee is 1, etc. als resultaat te retourneren. Dit is alles best mogelijk, maar dan is het wellicht gemakkelijker om niet langer de naam van de drank te onthouden maar het nummer. Natuurlijk moet dan ook de naam strGekozenDrank worden veranderd in intGekozenDrank! Maar dit is echter absoluut geen probleem! Zo hoeft er

aan frmZEKA absoluut niets te veranderen omdat deze via de Property Drank met het object communiceert, en die helemaal niet weet in welk veld de gekozen drank nu precies bewaard wordt en van welk type dat veld is. Dit is een van de voordelen van informatie-verbergen die een klasse van nature biedt.

Goed, eenmaal besloten dat we niet de naam van de drank maar het nummer door geven, met behoud van de propertynaam Drank, moeten de property procedures van Drank een beetje aangepast worden tot:

```
Propert Get Drank() As Integer
    Drank=intGekozenDrank 'resultaat is het nummer van de gekozen drank
End Property

Property Let Drank(ByVal intDrank As Integer)
    intGekozenDrank=intDrank
    intTeBetalen=intDrankPrijs(intDrank) ' dit is nu erg gemakkelijk
    RaiseEvent DrankGekozen
End Property
```

Vergeet niet de declaratie van strGekozenDrank te veranderen in:

```
Private intGekozenDrank As Integer
```

En in frmZEKA de opdracht ckaZEKA.Drank=cmdDrank(Index).Caption te wijzigen in:

```
ckaZEKA.Drank=Index
```

In de eventprocedure ckaZEKA_DrankGekozen kan dan worden gezet:

```
lblDisplay.Caption="Te betalen: fl. 0," & ckaZEKA.TeBetalen & "."
Call ZetBetaalknoppen(True)
cmdExtra(0).Enabled=(ckaZEKA.Drank=0)
cmdExtra(1).Enabled=(ckaZEKA.Drank<>2)
```

(We kunnen niet langer cmdExtra(0).Enabled=(Index=0) schrijven, omdat Index in deze eventprocedure niet bekend is, maar gelukkig is ckaZEKA.Drank gelijk aan het nummer van de drank dat we hier nodig hebben).

Goed, nu dat in orde is gaan we kijken hoe op het indrukken van de extra-knoppen moet worden gereageerd. Het ophogen van de property TeBetalen is de taak van het object. Dit ophogen zou via het toekennen van Index aan een property kunnen gebeuren b.v. een property met de naam Extra. Laten we het voor de verandering is met een methode doen. Laten we deze GekozenExtra noemen, waaraan Index als actuele parameter wordt doorgegeven. In cmdExtra_Click komt dan te staan:

```
ckaZEKA.GekozenExtra Index
```

(Index mag ook tussen haakjes worden gezet. Dan is misschien duidelijker dat het om de aanroep van een methode gaat!) frmZEKA verwacht dat het object het event ExtraGekozen opwekt als reactie.

Aan ckaZEKA moet de methode GekozenExtra en het event ExtraGekozen worden toegevoegd. GekozenExtra moet als Public worden gedeclareerd omdat deze immers van buiten de klasse aangeroepen moet kunnen worden, en ziet er dan als volgt uit:

```
Public Sub GekozenExtra(ByVal intExtra As Integer)
    intTeBetalen=intTeBetalen+intExtraPrijs(intExtra)
    RaiseEvent ExtraGekozen
End Sub
```

Vergeet niet het event toe te voegen aan de klasse-module:

```
Event ExtraGekozen()
```

Vervolgens zal in frmZEKA de eventprocedure ckaZEKA_ExtraGekozen waarheen de overige opdrachten in cmdExtra_Click kunnen worden verplaatst. Vergeet niet intTeBetalen te vervangen door ckaZEKA.TeBetalen.

Vervolgens komt cmdMunt_Click aan bod. Ook hier zullen we weer een methode gebruiken, die we de naam MuntBetaald zullen geven. De opdracht

```
ckaZEKA.BetaaldeMunt Index
```

kan achter het blok worden geplaatst waarin de drank- en extra-knoppen worden uitgezet. Uiteraard moet in dat blok intBetaald worden vervangen door ckaZEKA.Betaald. frmZEKA wacht vervolgens op het event MuntBetaald waarin de overige opdrachten kunen worden geplaatst, waarin weer de variabelen door hun property-equivalent zijn vervangen. Het toevoegen van event en eventprocedure is nu een makkie. Alle overige procedures kunnen, afgezien van de vervanging van variabelen door hun property-equivalenten, hetzelfde blijven. Alleen de opdracht ckaZEKA.Betaald=0 in ZetAutomaatInBeginStand kan niet worden gegeven, omdat er geen Property Let-procedure van Betaald is! Die kunnen we wel maken, maar het is logischer een methode te maken waarmee de automaat in de beginstand kan worden gezet die dan het veld intBetaald gelijk aan 0 maakt, en via het opwekken van een event laat weten dat de automaat in de beginstand moet worden gezet. Laten we die methode ZetInBeginStand noemen en het event Opstarten. Omdat ZetAutomaatInBeginStand dan nog maar één opdracht, nl. de aanroep van ZetInBeginStand, zal bevatten, kunnen we alle aanroepen van ZetAutomaatInBeginStand vervangen door aanroepen van ZetInBeginStand. Alle overigen opdrachten in ZetAutomaatInBeginStand verplaatsen we naar de eventprocedure ckaZEKA_Opstarten, waarna ZetAutomaatInBeginStand kan worden verwijderd.

De overige opdrachten die in Form_Load staan plaatsen we in de constructor Class_Initialize van clsZEKA zodat deze worden uitgevoerd zodra ckaZEKA wordt aangemaakt. Tenslotte plaatsen we de opdracht Set ckaZEKA=Nothing in de Form_Unload eventprocedure om ckaZEKA te verwijderen.

Hiermee is de opsplitsing voltooid. Het project is op het netwerk te vinden onder de naam ZEKA_O.VBP.

8. GEGEVENS IN VISUAL BASIC

In Visual Basic staan gegevens opgeslagen in constanten, variabelen of properties.

8.1 Properties

Objecten kunnen properties hebben. Alle controls die je op forms aanbrengt en de forms zelf zijn objecten. Daarnaast is het mogelijk om zelf klassen te definiëren, en binnen elke klasse eigen properties. Properties die alleen-lezen zijn kun je niet van waarde veranderen, maar alleen (in expressies) gebruiken.
Veranderen van een property doe je (meestal) in een toekenning. Properties kunnen echter ook veranderen door bepaalde activiteiten van Visual Basic. Zo zullen uiteraard de hoogte en breedte van een form veranderen als je het form tijdens het draaien van het programma een andere grootte geeft.

8.2 Constanten

Je slaat een gegeven in een constante op als de waarde ervan niet zal veranderen maar misschien wel op meerdere plaatsen in je applicatie wordt gebruikt. Het grote voordeel is dan dat je slechts eenmaal de waarde ervan hoeft op te geven, en aanpassing dus gemakkelijk is.

8.3 Variabelen

In een variabele kun je een of meerdere waarden opslaan. Wil je meerdere waarden gelijktijdig bewaren dan gebruik je een array of- een recordvariabele.
Een waarde kan alleen in een toekenning worden gewijzigd. Die door je in een toekenning door links van het toekenningsteken de naam van de variabele die de te wijzigen waarde bevat te vermelden en rechts van het toekenningsteken een formule die als uitgerekend de nieuwe waarde bevat. Hierdoor verdwijnt de vorige waarde die de variabele had. De waarde van een variabele verandert alleen als je dat zelf wilt.

8.4 Aanmaken van constanten en variabelen

Constanten en variabelen moeten worden aangemaakt, oftewel gedeclareerd, voordat je ze kunt gebruiken. (Properties hoeven niet nog eens te worden gedeclareerd omdat het aanbrengen van het bijbehorende object op het form al een soort declaratie is.)
Constanten declareer je met het Const-statement:

Const {naam constante}={waarde}

Variabelen declareer je meestal met het Dim-statement:

Dim {naam variabele} As {type}

Na declaratie heeft een variabele nog geen waarde: tekst- (string) resp. getalvariabelen (integer, single) zijn gelijk aan "" resp. 0.

8.5 Naamgeving van constanten en variabelen

Zoals je ziet moet je in de declaratie van een constante of een variabele de naam ervan invullen (en dus zelf bedenken). Kies een naam die toepasselijk is, die iets te maken heeft met de functie die de variabele gaat vervullen, de soort waarden die de variabele gaat bevatten. Een variabele die telt hoeveel getallen in een rij groter dan 0 zijn zul je eerder herkennen als je deze intAantal noemt dan als je deze intIkVerzinMaarWat noemt. Overigens: Visual Basic gebruikt alleen de eerste 32 tekens in de naam; die moeten dus uniek zijn. Bovendien mogen in de naam van een constante of variabelen alleen letters, cijfers en liggende streepjes voorkomen; het eerste teken mag echter geen cijfer zijn. Kies ook geen gereserveerd woord (als do of end) of naam van een property als naam want daar kan Visual Basic niet tegen. Zoals als je al zag begon ik de naam van de variabele met een afkorting van het type. Laat de naam van variabelen van het type Integer beginnen met int, van het type string met str enzovoorts. Je kunt dan gemakkelijk herkennen van welk type de variabele is.

8.6 De waarde van constanten

Daarnaast moet in de declaratie van een constante ook de waarde worden opgegeven. Dit kunnen alleen getallen (incl. True en False) en tekst (tussen dubbele aanhalingstekens) zijn.

8.7 Keuze van het gegevenstype

In de declaratie van een variabele moet het type worden opgegeven. Bedenk dus vooraf wat voor soort waarden je zult opslaan in die variabele. Zijn het gehele getallen kies dan als type Integer, zijn het niet-gehele getallen kies dan Single of Double. Is het een tekst (rij tekens) kies dan het type String.

8.8 Opslaan van een aantal waarden in één variabele

Wanneer in de variabele meerdere gegevens van hetzelfde type moeten worden opgeslagen, moet je de variabele als array declareren. Een array is een rij waarbij elk element van die rij een eigen (plaats)nummer heeft, vergelijkbaar met de huizen in een straat. Zo'n nummer wordt index genoemd. De nummering is altijd aaneengesloten, zodat je alleen het kleinste en grootste nummer hoeft op te geven in de declaratie.

8.8.1 Declaratie en gebruikswijze van arrays met een vast aantal elementen

Zo kan een rij die 10 gehele getallen moet gaan bevatten gedeclareerd worden met

```
Dim intGetallen(1 To 10) As Integer
```

Het eerste gehele getal in deze rij geef je aan met intGetallen(1), het tweede met intGetallen(2), enz. Wil je een dergelijke array alleen in een subroutine (kunnen) gebruiken, dan moet je die in de subroutine met Static i.p.v. Dim declareren. Dit betekent echter wel, dat de inhoud van de array tussen opeenvolgende keren dat de subroutine wordt uitgevoerd behouden blijft, terwijl de inhoud anders verloren gaat.

8.8.2 Declaratie en gebruikswijze van arrays met een variabel aantal elementen

Weet je vooraf niet hoeveel elementen de rij zal gaan bevatten dan hoef je nog niet op te geven welke nummers de elementen zullen krijgen; zet dan in de declaratie gewoon niets tussen de haakjes:

```
Dim intGetallen() As Integer
```

Het instellen van het aantal elementen tijdens de werking van de applicatie doe je met het Redim-statement. Voor een rij (een-dimensionale array) heeft deze de vorm:

```
ReDim {naam variabele}({laagste index} To {hoogste index})
```

Voor {laagste index} en {hoogste index} mag ook een expressie worden gebruikt: hier kan dus ook een variabele worden gebruikt. Het deel {laagste index} To kan worden weggelaten als de laagstmogelijke index 0 moet zijn.
Elke keer als Redim op bovenstaande wijze wordt aangeroepen wordt de rij leeggemaakt d.w.z. getalelementen worden 0, tekstelementen "". Dit kan worden voorkomen door ReDim te vervangen door ReDim Preserve. Dan blijven de waarden van de elementen die tot dan toe in de array zaten behouden.
Wanneer de array een n-dimensionale array (n>1) is moeten tussen de haakjes n bereiken worden opgegeven gescheiden door komma's. Zo kan de twee-dimensionale array intTabel b.v. worden gedimensioneerd met:

```
ReDim intTabel (-1 To 1, 3) As Integer
```

Bovenstaande matrix ziet er zo uit:

	0	1	2	3
-1	-	-	-	-
0	-	-	-	-
1	-	-	-	-

Nadat voor de eerste keer met ReDim het aantal dimensies is gekozen kan het aantal dimensies niet meer worden gewijzigd.
Een twee-dimensionale array wordt ook wel een tabel (of matrix) genoemd. De eerste index wordt dan om toepasselijke redenen ook wel rij-index genoemd, de tweede index de kolom-index.

Van elke dimensie kan altijd de laagste resp. grootste index worden opgevraagd met de functie LBound resp. UBound. Als eerste parameter moet de naam van de array worden opgegeven. Als tweede parameter moet de dimensie worden opgegeven; als deze gelijk aan 1 is, mag deze worden weggelaten). Zo zal LBound(intTabel) gelijk zijn aan -10 en UBound(intTabel,2) gelijk aan 15.
Wanneer een array niet meer nodig is kan deze worden verwijderd met Erase b.v. Erase intTabel. Daarna kan deze met ReDim wel weer worden gedimensioneerd.

8.9 Opslaan van een aantal gegevens van een verschillend type in een variabele

Wanneer de variabele gegevens van een verschillend type moet kunnen bevatten, dan zul je zelf een nieuw type moeten declareren; dit kan alleen in een module. Informatie hierover kun je in hoofdstuk 7 vinden en in de VB Help onder Type.

8.10 Bruikbaarheidsgebied van constanten en variabelen

Constanten en variabelen kunnen op verschillende plaatsen worden gedeclareerd. Binnen een subroutine (Sub of Function) gedeclareerd zijn ze alleen in de subroutine bekend (niet daarbuiten). Dit zijn de **subroutine-lokale declaraties**.

Binnen een form gedeclareerd (in het (general|declarations)-gedeelte) zijn ze alleen in dat form bekend (maar dan wel in alle subroutines). Dit zijn **de formulier-globale declaraties**.

Wil je een constante of variabele in meerdere forms kunnen gebruiken, dan moet deze worden gedeclareerd in een module. Een module kan alleen code (en declaraties) bevatten, en kun je aanmaken door File|New module uit te voeren.
Constanten daarin gedeclareerd in de vorm

Global Const {naam constante} = {waarde}

zijn in alle forms te gebruiken. Laat je de term Global weg dan kennen alleen de subroutines in het module deze constante. Variabelen moeten gedeclareerd worden in de vorm

Global {naam variabele} As {type}

om overal te worden gekend. Dit zijn de **applicatie-globale declaraties**.

8.11 Bruikbaarheidsgebied van properties

Bij het gebruiken van een property moet ook de naam van het object worden vermeld. Een uitzondering hierop vormen de properties van het form wanneer deze binnen dat form worden gebruikt! De Caption van Form1 kan binnen Form1 gewoon met Caption worden aangegeven.
Maar properties van controls en forms kunnen ook buiten het form waarin ze voorkomen worden gebruikt! Hiertoe is het noodzakelijk om ook nog eens de naam van de form te vermelden. Wil je b.v. de property BackColor van het object lblUitleg in form frmUitleg ergens in form frmStart gebruiken, dan moet je deze property aanduiden met frmUitleg.lblUitleg.BackColor.

8.12 De waarden die in variabelen kunnen worden opgeslagen

Het gegevenstype bepaalt welke waarden in een variabele kunnen worden opgeslagen. Omdat een digitale computer waarden opslaat in een eindig aantal geheugenelementen waarin slechts een eindig aantal verschillende binaire codes kunnen worden opgeslagen kunnen ook slechts een eindig aantal waarden worden opgeslagen in variabelen van welk type dan ook.

8.12.1 Gehele getallen sla je op in variabelen van het type Integer of Long.

In variabelen van het type Integer kan (op een PC) alleen elke waarde van -32768 t/m 32767 worden opgeslagen. In de meeste gevallen zal dit toereikend zijn. Moet een andere (grotere of kleinere) gehele waarde in een variabele worden opgeslagen dan kun je de variabele als Long declareren: deze kan elke waarde van -2.147.483.648 t/m 2.147.483.647 bevatten. Heb je nog grotere getallen nodig dan moet je het gehele getal in een variabele van het type Single of Double opslaan. Sla gehele getallen alleen op in variabelen van het type Long, Single of Double als het niet anders kan.

8.12.2 Reële getallen sla je op in variabelen van het type Single of Double.

Reële getallen waarvan de absolute waarde bij benadering ligt tussen $1,4*10^{-45}$ en $3,4*10^{38}$ kunnen in variabelen van het type Single worden opgeslagen. In variabelen van het type Double kunnen reële getallen worden opgeslagen waarvan de absolute waarde bij benadering ligt tussen $4,94*10^{-324}$ en $1,79*10^{308}$. Slechts een eindig aantal reële getallen kan exact worden gerepresenteerd. Van reële getallen opgeslagen als Single worden slechts ca. 6 cijfers opgeslagen: 0,123456 en 123456 zijn exact te representeren, maar 0,123456789 wordt afgerond tot 0,123457 opgeslagen en 123456789 tot 12345800. Van reële getallen opgeslagen als Double zijn ca. 15 cijfers nauwkeurig voor het echte getalkraken onontbeerlijk.

8.12.3 Tekst sla je op in variabelen van het type String.

Hierin kunnen maximaal 32767 tekens worden opgeslagen. Weet je van te voren het aantal tekens dan kun je als type String * {aantal tekens} gebruiken.

9. OPDRACHTEN IN VISUAL BASIC

9.1 Soorten opdrachten

VB kent - net als de meeste hogere programmeertalen - de volgende soorten opdrachten:

Hoofdsoort	Deelsoort	In VB	Functie
Toekenning			Een variabele of property een (nieuwe) waarde geven.
Aanroep		call	Uitvoeren van de opdrachten in deelprogramma (procedure of methode)
Besturing	Selectie	If-then-else	Als aan de voorwaarde wordt voldaan wordt de ene groep opdrachten uitgevoerd, anders de andere groep opdrachten.
	Onvoorwaardelijke lus	For-next	Vast aantal keer uitvoeren van een aantal opdrachten.
	Voorwaardelijke pre-test lus	do while-loop	Een aantal opdrachten uitvoeren zolang aan de voorwaarde wordt voldaan.
	Voorwaardelijke post-test-lus	do-loop until	Een aantal opdrachten uitvoeren totdat aan een bepaalde voorwaarde is voldaan

9.1.1 De toekenning

Eigenlijk is er maar een soort opdracht die wat doet: dat is de toekenning.
Een toekenning gebruik je als je een property of variabele een (andere, nieuwe) waarde wilt geven. Overal waar je daarna die variabele/property gebruikt wordt de laatste waarde die je die variabele/property hebt gegeven gebruikt.
Een toekenning heeft de volgende vorm:

{naam property/variabele} = {formule}

B.v. om intLeeftijd de waarde 16 te geven.

intLeeftijd = 16

Het is-gelijk teken stelt hier het toekenningsteken voor.
Links daarvan moet de naam van een property of variabele staan. Als het een property betreft moet de volledige naam daarvan worden weergegeven d.w.z. de naam van het object met die property, een punt gevolgd door de naam van property zelf (b.v. **lblLeeftijd.backcolor**).
Rechts van het toekenningsteken mag een formule staan; de property/variabele krijgt de waarde die deze formule voorstelt. In deze formule kunnen ook gewoon weer properties en variabelen worden gebruikt, maar ook functies (b.v. het gebruik van de functie Str$ in **lblLeeftijd.caption = Str$(intLeeftijd)**).
In de informatica wordt zo'n formule een **expressie** genoemd.

9.1.2 Besturingsopdrachten: de selectie

De selectie-opdracht maakt het mogelijk een of meerdere opdrachten te laten uitvoeren afhankelijk van een bepaalde voorwaarde. Hiertoe beschikt Visual Basic over de **If-Then-(Else)**-opdracht die één van de volgende twee vormen kan aannemen:

```
If {voorwaarde} Then {opdracht} Else {opdracht}
```

of

```
If {voorwaarde} Then
    {opdracht(en)}
Else
    {opdracht(en)}
End If
```

Als aan de voorwaarde voldaan wordt, dan worden de opdrachten tussen **Then** en **Else** uitgevoerd, anders die tussen **Else** en **End If**. Als tussen **Else** en **End If** geen opdrachten staan kan **Else** worden weggelaten.
Een eenvoudig voorbeeld waarbij lblMelding.Caption afhankelijk van intLeeftijd wordt bepaald:

```
If intLeeftijd>=50 Then
    lblMelding.Caption="U heeft Sarah of Abraham gezien."
Else
    lblMelding.Caption="U heeft Sarah of Abraham nog niet gezien."
End If
```

Als het geslacht bekend is, is het ook mogelijk om lblMelding.Caption nog nauwkeuriger te bepalen. In het volgende voorbeeld is intEenVrouw True als de bedoelde persoon een vrouw en False al deze een man is:

```
If intLeeftijd>=50 Then
   If intEenVrouw Then
      lblMelding.Caption="U heeft Sarah gezien."
   Else
      lblMelding.Caption="U heeft Abraham gezien."
   End If
Else
   If intEenVrouw Then
      lblMelding.Caption="U heeft Sarah nog niet gezien."
   Else
      lblMelding.Caption="U heeft Abraham nog niet gezien."
   End If
End If
```

Elke If-Then(-Else)-opdracht die meerdere regels gebruikt moet met End If worden afgesloten!
Zoals je ziet kunnen If-Then(-Else)-opdrachten ook binnen andere If-Then-(Else)-opdrachten worden gezet: dit wordt **nesten** genoemd. (Door netjes een aantal spaties in te springen hou je overzicht.)
Bovendien hoef je niet **intEenVrouw=True** neer te zetten, want True is True en False is niet True!

9.1.2.1 Combineren van voorwaarden

Wil je dat bepaalde opdrachten alleen worden uitgevoerd als aan twee voorwaarden voldaan wordt zet dan **And** tussen de voorwaarden. Zo wordt in het volgende voorbeeld lblMelding.Caption alleen gelijk aan "U bent een tiener." gemaakt als zowel intLeeftijd >= 10 als intLeeftijd <= 19:

```
If intLeeftijd>=10 And intLeeftijd<=19 Then
    lblMelding.Caption="U bent een tiener."
End If
```

Wil je dat bepaalde opdrachten alleen worden uitgevoerd als aan de ene of de andere voorwaarde voldaan wordt zet dan **Or** tussen de voorwaarden:

```
If intLeeftijd<10 Or intLeeftijd>19 Then
    lblMelding.Caption="U bent geen tiener."
End If
```

And en **Or** worden logische operatoren genoemd. Gebruik ze alleen om voorwaarden te combineren en vooral niet om opdrachten van elkaar te scheiden (In VB doe je dat met een :).
Wanneer meer dan twee voorwaarden moeten worden gecombineerd is het belangrijk te weten welke operator de hoogste prioriteit heeft: **And**. (Zo heeft * een hogere prioriteit dan +: 3+4*5=23 en niet 35.) Plaats zonodig haakjes.
Wanneer op elke waarde van een en dezelfde variabele verschillend moet worden gereageerd en het gaat hier om vele waarden dan is het zinvol i.p.v. van een aantal If-Then-Else-opdrachten gebruik te maken van de **Select Case**-opdracht (zie ook Visual Basic's online help):

```
Select Case intLeeftijd
    Case 0:          lblMelding.Caption="U bent een pasgeborene."
    Case 1 To 20:    lblMelding.Caption="U bent een kind."
    Case 21 To 64:   lblMelding.Caption="U bent een volwassene."
    Case 65 To 99:   lblMelding.Caption="U bent een senior."
    Case Else
        lblMelding.Caption="U bent meer dan een eeuw oud."
End Select
```

9.1.3 Besturingsopdrachten: herhalingen (lussen)

In theorie kunnen twee soorten herhalingen worden onderscheiden: de voorwaardelijke herhaling en de onvoorwaardelijke herhaling.
Bij een onvoorwaardelijke herhaling weet je van te voren hoe vaak opdrachten uitgevoerd moeten worden. B.v. stop 2 keer een suikerklontje in de thee.
Bij voorwaardelijke herhalingen moeten opdrachten worden uitgevoerd zolang of totdat aan een bepaalde voorwaarde wordt voldaan. B.v. stop suikerklontjes in de thee totdat de suiker boven de thee uikomt.
Onvoorwaardelijke herhalingen worden in Visual Basic gerealiseerd met de For-Next-opdracht. Deze heeft de algemene vorm:

```
For {tellervariabele}={startwaarde} To {eindwaarde}[ Step {stap}]
    {opdracht(en)}
Next [{naam variabele}]
```

(Wat tussen vierkante haken staat mag worden weggelaten; het Step-gedeelte alleen als stap gelijk aan 1 is.) Startwaarde, eindwaarde en stap zijn formules die gehele getallen moeten voorstellen. De opdrachten worden uitgevoerd voor waarden van de

variabele gelijk aan startwaarde, startwaarde+stap, startwaarde+2 *stap, etc. zolang de waarde van de variabele maar kleiner is dan die van eindwaarde: als de startwaarde groter/kleiner is dan de eindwaarde wanneer de stapgrootte positief/negatief is, worden de opdrachten in de lus 0 keer uitgevoerd!
In de opdrachten in de lus (tussen For en Next) kan de waarde van de tellervariabele natuurlijk worden gebruikt zodat niet steeds precies hetzelfde wordt gedaan.

Voorwaardelijke herhalingen kent Visual Basic in 4 soorten:
- Do While-Loop,
- Do-Loop Until,
- Do Until-Loop en
- Do-Loop While.

We bespreken hier alleen de eerste twee. **Do While-Loop**-opdrachten hebben de vorm:

```
Do While {voorwaarde}
        {opdracht(en)}
Loop
```

De opdrachten worden uitgevoerd zolang aan de voorwaarde voldaan wordt. Dit betekent dat de opdrachten ervoor moeten zorgen dat op zeker moment niet meer aan de voorwaarde voldaan zal zijn, anders zit je in een oneindige lus d.w.z. het programma komt nooit uit de lus en kan niet meer zelf stoppen!

Do-Loop Until-opdrachten hebben de vorm:

```
Do
        {opdracht(en)}
Loop Until {voorwaarde}
```

De opdrachten worden uitgevoerd totdat aan de voorwaarde voldaan wordt. Ook hier is het dus noodzakelijk dat de opdrachten ervoor zorgen dat op zeker moment aan de voorwaarde voldaan zal worden.

Toepassingen van lussen

Het gebruik van lussen is soms wel, soms niet noodzakelijk. Stel: je jongere broertje of zusje moet - hoe kinderachtig ook - een aantal strafregels schrijven met steeds dezelfde tekst. Een dergelijk routinematig karweitje kan toch veel makkelijker door de computer worden gedaan! Het heeft natuurlijk weinig zin dan elke te schrijven regel met een aparte Print-opdracht weer te geven; dan zou je nog steeds alle tekst in moeten voeren. Is er geen gemakkelijkere manier? Jazeker, die is er: nl. met een lus b.v als volgt:

```
For intKeer = 1 To 100
    Print "Ik heb het wel gedaan, maar het was niet mijn schuld..."
Next intKeer
```

Het gevolg is dat 100 keer onder elkaar de tussen aanhalingstekens staande tekst met een twijfelachtig waarheidsgehalte op je form wordt weergegeven.
Natuurlijk is het dan voor de lezer ervan moeilijk vast te stellen of het aantal wel klopt. Dus, als je dan toch gehoorzaam bent, kun je ook wel afdrukken de hoeveelste regel het is. maar dat zou wel betekenen dat steeds iets anders moet worden geprint, en kan dat dan wel? Jazeker! De variabele intKeer zal achtereenvolgens de waarden 1, 2, 3, ..., 100 aannemen. De waarde die intKeer heeft is binnen de lus (dat wat tussen de eerste en laatste regel van de **For-Next**-opdracht staat) beschikbaar en kan dus worden geprint. Natuurlijk willen we nog een spatie tussen de weergegeven waarde van intKeer en de tekst, zodat de volgende For-Next-opdracht het gewenste werk doet:

```
For intKeer=1 To 100
    Print CStr(intKeer) & _
            " Ik heb het wel gedaan, maar het was niet mijn schuld..."
Next intKeer
```

(Het liggend streepje kan gebruikt worden om een opdracht over meerdere regels te verdelen, niet alleen hier, maar ook in Visual Basic.)
Kortom: het is mogelijk dat wat binnen de lus gebeurt afhankelijk te maken van de waarde die de lusteller (want zo wordt intKeer genoemd) op dat moment heeft. Zo kun je toch verschillende dingen laten doen, ja zelfs totaal verschillende dingen want binnen lussen mag je ook gewoon weer If-Then(-Else)-opdrachten gebruiken.
Goed, in bovenstaande gevallen was het nog mogelijk - alhoewel onhandig - om het gebruik van een lus te omzeilen (nl. door alle (100) opdrachten één voor één in te typen). De start- en eindwaarde waarvoor de acties binnen de lus moesten worden uitgevoerd waren daarbij immers steeds constant d.w.z. gelijk voor elke uitvoering van het programma waardoor het dus steeds om een vast aantal uit te voeren acties ging, die desnoods ook één voor één zouden kunnen worden ingetypt. Het komt echter in de praktijk regelmatig voor dat een aantal acties niet per se een vast aantal keer maar een variabel aantal keer moet worden uitgevoerd, afhankelijk van de waarden die een of meerdere variabelen op dat moment bezitten. Dan is het niet meer ('eenvoudig') mogelijk om het gebruik van een lus te omzeilen, omdat je immers niet precies meer weet hoe vaak de acties moeten worden uitgevoerd.
Stel b.v. dat je de som van een aantal getallen moet bepalen maar dat je bij het schrijven van het programma niet weet hoeveel getallen dat zullen zijn. Wel weet je b.v. dat het aantal waar het om gaat de waarde van de variabele intAantal is en dat de getallen staan opgeslagen in intGetallen(1) t/m intGetallen(intAantal). De som kan eenvoudig met de volgende lus worden berekend:

```
intSom=0
For intKeer=1 To intAantal
    intSom=intSom+intGetallen(intKeer)
Next intKeer
```

Het is gemakkelijk te zien dat dit klopt voor alle waarden die intAantal kan aannemen. Als intAantal gelijk aan 0 is wordt de lus niet uitgevoerd, omdat de startwaarde (1) dan kleiner is dan de eindwaarde (0). Na uitvoering van de lus zal intSom nog steeds gelijk aan 0 zijn, zoals het hoort.
Als intAantal gelijk aan 1 is wordt de opdracht in de lus maar één keer uitgevoerd, nl. voor intKeer = 1, en zal intSom na uitvoering van de lus gelijk aan intGetallen(1) zijn, ook zoals het hoort. En als intKeer gelijk aan 2 is wordt intSom eerst gelijk aan 0 + intGetallen(1) gemaakt (intKeer = 1) en vervolgens gelijk aan zichzelf (intGetallen(1)) + intGetallen(2) dus gelijk aan intGetallen(1) + intGetallen(2), de som van de eerste twee getallen uit intGetallen, precies zoals het hoort.
Hier is ook goed te zien hoe voordelig het kan zijn om bij elkaar horende gegevens in een array op te slaan. Had je geweten hoe de som van de waarden opgeslagen in de variabelen intGetal1, intGetal2, intGetal3 t/m intGetal100 op een net zó gemakkelijke manier had kunnen worden berekend? Die is er niet want het nummer aan het einde is onderdeel van de naam, en geen index waarvoor een formule kan worden gebruikt!
Bovenstaande uitleg zou je ervan hebben moeten overtuigen dat het gebruik van lussen niet alleen voordelig kan zijn maar soms ook onvermijdelijk.
Er kunnen echter situaties optreden die niet met For-Next-lussen kunnen worden behandeld. Deze treden op wanneer het aantal keer dat de acties in de lus moeten worden uitgevoerd niet vooraf d.w.z. vóór uitvoering van de lus (zelfs niet in de vorm van een formule) bekend is!

Denk b.v. maar eens aan het bepalen van de som van een een aantal getallen totdat de som groter dan 100 is. Weet je van te voren hoe vaak je het volgende getal bij de som op moet tellen? Dat hangt immers af van de waarde van de getallen. In zo'n geval heb je een voorwaardelijke herhaling nodig. Omdat we in dit geval willen doorgaan met optellen totdat de som groter dan 100 is, is een Do-Loop Until-opdracht op z'n plaats. Maar let op! Als intGetallen allemaal negatieve getallen bevat zal de som nooit groter worden dan 100 en zal de lus nooit worden beëindigd. Dat zou betekenen dat je programma niet zal eindigen en dat je eeuwig aan het toetsenbord gekluisterd zal moeten blijven zitten - je ergste nachtmerrie. De lus moet niet alleen eindigen als de som groter dan 100 is maar ook als alle getallen gebruikt zijn! De rij bevat immers altijd maar een eindig aantal getallen. Maken we weer gebruik van de rij intGetallen(1) t/m intGetallen(intAantal) dan werkt de volgende lus:

```
intSom=0
intKeer=1
Do
    intSom=intSom+intGetallen(intKeer)
    intKeer=intKeer+1
Loop Until intSom>100 Or intKeer>intAantal
```

Hier moet je er wel zelf voor zorgen dat intKeer de juiste waarde heeft. De eerste keer dat de opdrachten in de lus worden uitgevoerd moet intSom gelijk worden aan intGetallen(1). Dus moet intKeer dan gelijk aan 1 zijn. Dit kan alleen worden bewerkstelligd door intKeer vóór de lus gelijk aan 1 te maken. De tweede keer dat de opdrachten in de lus worden uitgevoerd moet intSom met intGetallen(2) worden opgehoogd; dit betekent dat intKeer dan gelijk aan 2 moet zijn. intKeer was gelijk aan 1, moet 2 worden dus moet er 1 bij op worden geteld.
Je zou kunnen denken dat de voorwaarden met And i.p.v. Or hadden moeten worden verbonden. Dan zou de lus pas eindigen als zowel intSom>100 als intKeer>intAantal waar zijn, terwijl er juist gestopt moet worden zodra aan de ene of aan de andere voorwaarde (of beide) is voldaan.

9.1.4 Aanroepen van methodes

Objecten (controls en forms) hebben **eigenschappen (*properties*)** en herkennen **gebeurtenissen (*events*)**. Daarnaast zijn met sommige **objecten methoden (*methods*)** verbonden. Zo kent een list box de methoden AddItem voor het toevoegen van een element, RemoveItem voor het verwijderen van een element en Clear voor het leegmaken van de lijst. De aanroep van een methode heeft de volgende vorm:

{object}.{methode} {argumenten}

De argumenten - ook wel actuele parameters genoemd - worden met komma's van elkaar gescheiden. Sommige methoden hebben geen argumenten (zoals Clear). Soms hoeven niet per se alle argumenten opgegeven te worden. Dan wordt voor de ontbrekende een verstekwaarde (default) genomen. Zo kunnen in een aanroep van AddItem twee parameters worden opgegeven; de eerste moet de toe te voegen tekst voorstellen; de tweede de plaats (in de lijst) waar de tekst moet worden toegevoegd. Als de tweede parameter ontbreekt, dan wordt de tekst automatisch aan het einde van de lijst toegevoegd.

9.1.5 Aanroepen van procedures

In tegenstelling tot methoden - die meegeleverd zijn bij de objecten - moeten procedures altijd wel worden gedeclareerd.
De declaratie ervan heeft de vorm:

```
Sub {naam procedure} ({formele parameters})
{declaraties van lokale variabelen}
   {opdrachten}
End Sub
```

De eerste procedures die je in Visual Basic tegenkwam zijn de eventprocedures, die (automatisch) worden aangeroepen als het bijbehorende event optreedt. De naam ervan heeft dan de vorm {object}_{event}.
De naam van een eventprocedure - alsook de formele parameters - ligt vast. De eerste regel van een eventprocedure zul je dan ook nooit hoeven te veranderen. Daarnaast is het mogelijk eigengemaakte procedures toe te voegen. Hiervan moet je wel zelf de naam en formele parameters kiezen. Je kunt aan een form een procedure toevoegen m.b.v. het pull-down menu Tools|New Procedure...; het kan echter ook door de eerste regel ervan in te typen op een willekeurige plaats in het Code Window die niet tot een andere procedure behoort, b.v. in het (general)-gedeelte of na de End Sub van een eventprocedure.
De formele parameters vormen de invoergegevens die aan de procedure dienen te worden doorgegeven in de aanroep. Een procedure aanroepen betekent opdracht geven tot het uitvoeren van de opdrachten in de procedure. Omdat de formele parameters in de opdrachten binnen de procedure worden gebruikt moeten deze in de aanroep worden opgegeven. Een heel eenvoudig voorbeeld is een procedure die de waarden van twee variabelen onderling verwisselt:

```
Sub Verwissel(sngEne As Single, sngAndere As Single)
Dim sngHulp As sngSingle
   sngHulp=sngEne      ' even de waarde van sngEne bewaren in sngHulp
   sngEne=sngAndere    ' sngEne gelijk maken aan sngAndere
   sngAndere=sngHulp   ' sngHulp bevat de oude (originele) sngEne
End Sub
```

Tijdens de creatie van deze procedure is al wel aangegeven wat de invoer resp. uitvoer van de procedure is, nl. sngEne en sngAndere. Deze parameters worden formele parameters genoemd omdat ze nog geen waarde hebben, deze pas krijgen in de aanroep. Ze kunnen in de declaratie van de procedure echter al wel worden gebruikt alsof ze bekend zijn. De pas bij aanroepen uit te voeren opdrachten kunnen worden beschreven in termen van de namen van de parameters, zonder dat de waarden daarvan op dat moment bekend hoeven te zijn.
De procedure Verwissel wordt elke keer uitgevoerd als deze wordt aangeroepen. Een procedure roep je in Visual Basic aan door de naam van de procedure in te typen gevolgd door de voor de formele parameters te gebruiken gegevens. Deze gegevens worden de actuele parameters genoemd. Je moet altijd net zoveel actuele parameters in de aanroep gebruiken als er formele parameters in de declaratie voorkomen.
Variabele sngX kan de waarde krijgen van sngY en omgekeerd met de opdracht:

```
Call Verwissel(sngX,sngY)
```

9.1.6 Aanroepen van functies

Naast procedures kunnen in Visual Basic ook functies - op vergelijkbare wijze - worden gedeclareerd. Functies zijn in programmeertalen te vergelijken met wiskundige functies: je stopt er een getal in en krijgt er een ander getal voor terug. Standaardfuncties zijn sinus, cosinus en dergelijke. Functiedeclaraties hebben de vorm:

```
Function {naam functie} ([{formele parameters}]) As {gegevenstype}
{declaraties van lokale variabelen}
  {opdrachten}
End Function
```

Uit de declaratie kan al een belangrijk verschil met een procedure worden afgeleid: een functie heeft een resultaat (van het aangegeven type). Omdat een functie een resultaat heeft mag de aanroep van een functie alleen in een formule (b.v. rechts van het toekenningsteken, in een voorwaarde of als actuele parameter) worden gebruikt en kan dus **nooit** als afzonderlijke opdracht in de code staan. Dus b.v. wel sngY = Rnd(), maar nooit Rnd, want Rnd is een functie en geen procedure!
Een eenvoudige functie is die de som van twee getallen als resultaat retourneert:

```
Function SomVan(sngEne As Single, sngAndere As Single) As Single
    SomVan=sngEne+sngAndere
End Function
```

Wanneer we nu sngZ gelijk willen maken aan de som van sngX en sngY, dan bereiken we dat met de toekenning:

```
sngZ=SomVan(sngX,sngY)
```

SomVan kan nu echter alleen de som van twee variabelen retourneren, maar niet de som van twee waarden. Door vóór de formele parameters het sleutelwoord ByVal (afkorting van **by value**) te plaatsen maken we kenbaar dat de actuele parameters als expressies moeten worden beschouwd. De formele parameters worden dan waarde-parameters genoemd (zonder het sleutelwoord ByVal wordt een formele parameter een variabele-parameter genoemd). De declaratie van SomVan wordt dan:

```
Function SomVan(ByVal sngEne As Single, _
                ByVal sngAndere As Single) As Single
    SomVan=sngEne+sngAndere
End Function
```

Met SomVan kan nog steeds de som van sngX en sngY op dezelfde wijze bepaald worden. Het is nu echter ook mogelijk voor de actuele parameters formules in te vullen. Zo zal de opdracht

```
sngZ=sqr(SomVan(sngX*sngX,sngY*sngY))
```

sngZ gelijk maken aan de vierkantswortel uit de som van de kwadraten van sngX en sngY oftewel de lengte van de schuine zijde van een rechthoekige driehoek met rechte zijden met lengte sngX en sngY.
Het lijkt erop of het gebruik van ByVal alleen maar voordelen heeft. Er is echter een beperking. Zelfs als de actuele parameters variabelen zijn dan nog kunnen deze niet door de subroutine worden gewijzigd: ze worden immers als expressies, waarden, beschouwd. Zouden dus de formele parameters van Verwissel als waarde-parameters zijn gedeclareerd dan zou de aanroep: Call Verwissel(sngX,sngY) de waarden van sngX en sngY ongemoeid laten. Maak daarom alleen gebruik van waarde-parameters als je ook formules in de aanroep wilt gebruiken.

9.2 Formules, voorwaarden en operatoren

Formules en voorwaarden worden in hogere programmeertalen expressies genoemd. Voorwaarden kunnen alleen maar logische expressies zijn, d.w.z. de waarde ervan is True of False. Voorwaarden komen voor in besturingsopdrachten.
Formules komen hetzij in een toekenning rechts van het = teken voor (b.v. intProdukt = intX * intY) of tussen de ronde haken waar de index(en) van het te gebruiken element van de array moeten worden vermeld. In formules pas je functies en operatoren toe.
Voorbeelden van functies zijn cos (voor het berekenen van cosinus) en sqr (voor het berekenen van een vierkantswortel). De argumenten waarop operatoren worden losgelaten worden operanden genoemd.

9.2.1 Operatoren in VB

Rekenkundige operatoren	
+	voor het berekenen van een som of het achter elkaar plakken van teksten
&	voor het achter elkaar plakken van teksten
-	voor het berekenen van een verschil (of als teken b.v. in -intAantal)
*	voor het berekenen van een produkt
/	voor het berekenen van een quotiënt (35 / 8 = 4,375)
\	voor het uitvoeren van een geheeltallige deling (35 \ 8 = 4)
Mod	voor het berekenen van de rest na deling (37 Mod 8 = 5, want 37/8=4 rest 5)
Abs	voor het bepalen van de absolute waarde van een getal
^	voor machtsverheffen (2^8=256)

Vergelijkingsoperatoren	
=	is gelijk aan
<	kleiner dan
<=	kleiner dan of gelijk aan
>	groter dan
>=	groter dan of gelijk aan
<>	ongelijk aan

Logische operatoren	
Not	gelijk aan True als de operand False is, en andersom
And	gelijk aan True als beide operanden True zijn, anders False
Or	gelijk aan True als minstens één van beide operanden True is, anders False

Functies kunnen standaardfuncties zijn, maar ook zelfgemaakte functies. Als de functie parameters heeft moeten tussen de actuele parameters tussen ronde haken achter de functienaam worden vermeld. Een voorbeeld van een standaardfunctie die geen parameters heeft is Rnd, die als resultaat een willekeurig getal in [0,1) geeft. Belangrijke functies die je wellicht nodig zult hebben.

Naam	**Resultaat**
Chr	het teken behorende bij de ASCII-code (zo is Chr(97) "a").
Asc	de ASCII-code behorende bij het teken dat is opgegeven (zo is Asc("a") 97).
CStr	de tekenrepresentatie van het getal dat is opgegeven (CStr(123) is "123").
Val	het getal zijnde de het getal dat het argument voorstelt (zo is Val("123") 123).

9.3 Tekst

Tekst sla je op in variabelen van het type String. Kies als type String * {aantal tekens} als je van te voren weet hoeveel tekens je tekst gaat bevatten. Voor {aantal tekens} moet je dan zelf een aantal invullen.

```
Dim strNaam as String
```

of

```
Dim strAdres as String * 24
```

9.3.1 Het aantal tekens in een tekst

Het aantal tekens dat een tekst bevat wordt de lengte van de tekst genoemd. De lengte van een variabele van het type String kun je opvragen met de standaardfunctie Len. Als strNaam van het type String is, is Len(strNaam) dus op elk moment de lengte van strNaam. Je kunt Len ook gebruiken voor het bepalen van het aantal tekens dat in een bepaalde property staat, tenminste als die property tenminste van het type String is. Zo is het aantal tekens dat de Caption property van label lblUitleg bevat gelijk aan Len(lblUitleg.Caption).
Zo zal het vaak voorkomen dat je wilt vaststellen of een tekstvariabele wel tekens bevat b.v. If Len(strNaam)>0 Then...

9.3.2 Stukjes tekst gebruiken

Als je een deel van een tekstvariabele wilt gebruiken heb je de functie Mid$ nodig. Deze functie heeft 3 parameters: de eerste is de tekstvariabele, de tweede het nummer van het eerste teken en de derde het aantal tekens. Als strTekst gelijk is aan "hetzelfde" is Mid$(S,3,3) gelijk aan "tze". Het is belangrijk dat de tweede en derde parameter toegestane waarden zijn. Zo is het niet toegestaan dat de derde parameter negatief is. In de praktijk betekent dit sommige Mid$ pas gebruikt mogen worden als aan bepaalde voorwaarden voldaan is.
Tussen twee haakjes: Geef je in de aanroep geen derde parameter op dan pakt VB automatisch de rest van de tekst. Gemakkelijk als je de rest van de tekst nodig hebt. Als de eerste parameter van Mid$ gelijk is aan 1 kun je beter gebruikmaken van de functie Left$. Deze heeft dezelfde functie als Mid$ alleen hoeft de 1 niet te worden vermeld en daarom heeft Left$ slechts twee parameters. Er is ook een functie Right$ waarmee je gemakkelijk een gedeelte aan het einde van de tekst kunt te pakken krijgen: zo is Right$(strTekst,2) gelijk aan "de" als strTekst gelijk is aan "hetzelfde".

9.3.3 Stukjes tekst vervangen

Ook dit doe je met Mid$, alleen moet je Mid$ dan links van het toekenningsteken gebruiken. Ik weet het: links van het toekenningsteken mag je toch alleen variabelen (geen expressies) gebruiken? Nou, dit is dan de uitzondering op de regel. Dus Mid$(strTekst,4,3)="" maakt strTekst gelijk aan "hetde" als strTekst van te voren "hetzelfde" was. Je kunt met Mid$ dus ook stukjes tekst uit tekstvariabele verwijderen. Hier kun je Mid$ niet door Left$ of Right$ vervangen!

9.3.4 Teksten achter elkaar plakken

Dit wordt ook wel concateneren genoemd. Die doe je met &. Dus "het" & "zelfde" is hetzelfde als "hetzelfde". Heel handig als je getallen in een tekst wilt invoegen:
"U bent " & Str(intLeeftijd) & " jaar oud.".

9.4 Conversie en weergave

VB kent een aantal standaardfuncties voor conversie van en naar getallen. Met de functie Val kan van elke tekstvariabele een getal worden gemaakt. Dit werkt ook als het geen getal voorstelt! Als je er zeker van wilt zijn dat de conversie alleen wordt uitgevoerd als de tekst ook daadwerkelijk een getal voorstelt gebruik dan CInt, CLng, CSng of CDbl of een andere conversiefunctie (afhankelijk van gewenste type van het resultaat).
Met de functie CStr (resultaat is van het type String maak je van een getal een tekst. Van elk teken wordt steeds de ASCII-code d.w.z. het volgnummer in de ASCII-tabel opgeslagen. De functie Asc geeft je dit volgnummer. Zo is Asc("A") gelijk aan 65. Anderszijds kan ook het teken dat bij een bepaalde ASCII-code hoort worden verkregen nl. met de Chr-functie. Zo is Chr(65) gelijk aan "A".

Wanneer je wilt dat een tekst over meerdere regels in een label of text box komt te staan moet je weten welke tekens tussen de tekst in twee opeenvolgende regels moeten staan. Dit zijn Chr(13) en Chr(10). Dus de tekst "Welkom" & Chr(13) & Chr(10) & "bij" & Chr(13) & Chr(10) "mijn programma" wordt op drie regels weergegeven en "Welkom bij mijn programma" op één regel. In het Properties Window kun je deze speciale tekens niet invoeren: dit moet dus altijd met code. Overigens kent Visual Basic 5 ook een heleboel voorgedefiniëerde constanten die allemaal met de letters vb beginnen. De hierbovengenoemde regelscheider kent VB als vbCrLf en in je programma kun je dus ook schrijven: "Welkom" & vbCrLf & "bij" & vbCrLf & "mijn programma".

Wil je een getal, een tekst, een datum, een tijdstip op een speciale manier weergeven, gebruik dan Format($). Zie voor details weer de online VB Help.

9.5 Foutafhandeling

Als tijdens het draaien van een applicatie een fout optreedt gaat VB op zoek naar de meest recent gedefiniëerde foutafhandelaar (error handler). Als je er zelf geen opgegeven hebt, wordt uiteraard VB's eigen foutafhandelaar gestart. Deze geeft gewoonlijk een boodschap op het scherm weer die aangeeft wat voor fout is opgetreden. In zo'n geval zal de applicatie in de regel worden afgebroken. Door zelf een foutafhandelaar te maken kan voorkomen worden dat in situaties waarin correctie wel degelijk mogelijk is de applicatie vroegtijdig wordt beëindigd. Een foutafhandelaar maak je kenbaar m.b.v. een On Error-opdracht. Er zijn een aantal algemene vormen waarvan de volgende het meest gebruikt wordt (althans door mij):

On Error Goto {label}

Voor {label} moet de naam van een label worden gekozen die in dezelfde subroutine voorkomt. Als er een fout optreedt worden de opdrachten volgende op het label uitgevoerd.

```
Function Deling(ByVal sngX As Single, ByVal sngY As Single, _
                                    intFout As Integer) As Single
    intFout=0
    On Error Goto DelenMislukt
    Deling=sngX/sngY
    Exit Function
DelenMislukt:
    intFout=Err
    MsgBox Error$(Err)
    Exit Function
End Function
```

De functie in dit voorbeeld heeft als resultaat het quotiënt van sngX en sngY, maar het gaat o.a. fout als de noemer, d.i. sngY, gelijk aan 0 is of als sngY zo klein is dat het resultaat niet een Single kan worden opgeslagen. Het is niet moeilijk met een If-Then-opdracht te voorkomen dat abusievelijk door 0 wordt gedeeld, maar het is een stuk moeilijker rekening te houden met een te kleine waarde van de noemer, omdat deze afhangt van de waarde van de teller sngX.
Als een fout optreedt bij het delen dan worden de opdrachten volgende DelenMislukt uitgevoerd. VB houdt een foutcode bij, die opgevraagd kan worden met de standaardfunctie **Err** (van het type Integer). Deze is altijd ongelijk aan 0, zodat als intFout gelijk aan 0 is de deling is gelukt en anders mislukt. De omschrijving van de fout kan worden opgevraagd door de aanroep van de functie **Error$** (van het type String) met als actuele parameter de foutcode.
Het effect van de opdracht **Exit Function** is dat het uitvoeren van de functie wordt beëindigd. Een dergelijke opdracht moet vóór DelenMislukt staan, omdat anders, terwijl het delen gelukt is, toch de opdrachten achter DelenMislukt worden uitgevoerd, wat niet de bedoeling is. De foutafhandelaar kan eindigen met **Exit Function** (**of Exit Sub** als het om een procedure gaat) maar ook met **Resume** (opnieuw proberen de mislukte opdracht uit te voeren (b.v. na correctie van sngY)) of met **Resume Next** (verder gaan met de opdracht volgende op diegene die mislukt is). De foutafhandelaar blijft actief tot de subroutine klaar is, of totdat deze door een nieuwe foutafhandelaar wordt vervangen, of totdat hij inactief wordt gemaakt met **On Error Goto 0**.
(Tussen twee haakjes: Visual Basic kent ook nog een aantal andere Exit-faciliteiten. Met **Exit For** kan een For-Next-lus worden verlaten; met **Exit Do** elke **Do-Loop**-lus. Het gebruik van deze faciliteiten, in het bijzonder van de **Exit For**, is echter af te raden.)

9.6 Overzicht functies, procedures en methoden [VBLR]

Categorie	Actie	Functie/Procedure/Methode
Arrays	Wijzigen verstek laagste index	Option Base
	Declareren en initialiseren	Dim, Global, ReDim, Static
	Laagste en hoogste index opvragen	LBound, UBound
	Opnieuw initialiseren	Erase, ReDim
Bepalen van moment van uitvoeren	Springen	GoSub...Return, GoTo, On Error, On ... GoSub, On ... GoTo
	Eindigen of pauzeren van het programma	DoEvents, End, Stop, Unload
(Besturing)	Herhaling (lus)	Do ... Loop, For ... Next, While ... Wend
	Beslissingen nemen	If ... Then ... Else, Select Case
Conversie van	ANSI waarde naar string	Chr, Chr$
	Datum naar seriëel nummer	DateSerial, DateValue
	Decimaal getal naar ander talstelsel	Hex, Hex$, Oct, Oct$
	Nummer naar string	Format, Format$, Str, Str$
Conversie van	Het ene datatype naar het andere	CCur, CDbl, CInt, CLng, CSng, CStr, CVar, CVDate, Fix, Int
	Seriëel nummer naar datum	Day, Month, Weekday, Year
	Seriëel nummer naar tijdstip	Hour, Minute, Second
	String naar ASCII-waarde	Asc
	String naar nummer	Value
	Tijdstip naar seriëel nummer	TimeSerial, TimeValue
Copiëren, knippen en Plakken	Gebruik het Clipboard object	Clear, GetData, GetFormat, GetText, SetData, SetText
Datum/Tijdstip	Huidige tijdstip en datum opvragen	Date, Date$, Now, Time, Time$
	Huidige datum en tijdstip instellen	Date, Date$, Time, Time$
	Een proces timen (b.v. t.b.v. een bepaalde tijd wachten)	Timer
Dynamic Data Exchange (DDE)	Gebruiken van een VB-applicatie als DDE-cliënt	LinkExecute, LinkPoke, LinkRequest
	Een VB-applicatie gebruiken als DDE-server	LinkSend
Foutafhandeling	Foutboodschappen opvragen	Error$
	Foutcodes opvragen	Err, Erl
	Zelf een run-time error genereren	Error
	Foutafhandelaar maken	On Error, Resume
I/O (bestanden)	Een bestand openen (cq. aanmaken)	Open
	Een bestand sluiten	Close, Reset
	Weergave besturen	Spc, Tab, Width #

Categorie	Actie	Functie/Procedure/Methode
	Een bestand copiëren naar een ander	FileCopy
I/O bestanden	Informatie over een bestand opvragen	EOF, FileAttr, FileDate, FileLen, FreeFile, Loc, LOF, Seek
	Schijven of directories beheren	ChDir, ChDrive, CurDir, CurDir$, MkDir, RmDir
	Bestanden beheren	Dir, Dir$, Kill, Lock ... Unlock, Name
	Lezen uit een bestand	Get, Input, Input #, Input $, Line Input #
	Bestandsattributen instellen/opvragen	GetAttr, SetAttr
	Bestandspositie instellen	Seek
	Schrijven naar een bestand	Print #, Put, Write #
Grafisch	Coördinatenstelsel veranderen	Scale, ScaleMode, ScaleWidth, ScaleHeight, ScaleLeft, ScaleTop
	Grafische inhoud verwijderen	Cls
	Vormen tekenen	Circle, Line, PSet
	Tekst weergeven	Print
	Grootte van tekst opvragen	TextHeight, TextWidth
Grafisch	Inlezen/schrijven van plaatje van/naar bestand	LoadPicture, SavePicture
Grafisch	Werken met kleur	Point, QBColor, RGB
Objecten manipuleren	Rangschikken van forms of controls op het scherm	Arrange, Zorder
	Kiezen van een control	SetFocus
	Dialoogvenster tonen	InputBox, Inputbox$, MsgBox
	Slepen en laten vallen (drag and drop)	Drag
	Tonen of verbergen van forms	Show, Hide
	Laden of ontladen van objecten	Load, Unload
	Verplaatsen of veranderen van de grootte van objecten	Move
	Afdrukken van een form	PrintForm
	Bijwerken van scherm	Refresh
	Toevoegen of verwijderen van items uit een lijst in een list en combo box	AddItem, RemoveItem
Bewerkingen	Algemene functies	Exp, Log, Sqr
	Operatoren	-, +, /, *, \, Mod, ^
	Willekeurige getallen genereren	Randomize, Rnd
	Absolute waarde opvragen	Abs
	Het teken van een expressie opvragen	Sgn
	Numerieke conversie	Fix, Int
	Goniometrische functies	Atn, Cos, Sin, Tan
Afdrukken	Lay-out	Scale, Spc, Tab, TextHeight, TextWidth

Categorie	Actie	Functie/Procedure/Methode
Afdrukken	Printer besturen	EndDoc, NewPage
	Afdrukken	Print, PrintForm
Procedures	Een procedure aanroepen	Call
	Een verwijzing naar een externe procedure (in een DLL) opgeven	Declare
	Een subroutine declareren	Function ... End Function, Sub ... End Sub
	Een subroutine beëindigen	Exit Function, Exit Sub
Strings (tekst)	Waarde geven	Let
	Twee strings vergelijken	StrComp
	Conversie naar kleine of hoofdletters	LCase, LCase$, UCase, Ucase$
	Maken van strings van zichzelf herhalende karakters	Space, Space$, String, String$
	Lengte van een string opvragen	Len
Strings	Op speciale wijze weergeven van een string	Format, Format$
	Een string uitlijnen	Lset, RSet
	Strings manipuleren	InStr, Left, Left$, LTrim, LTrim$, Mid, Mid$, Right, Right$, RTrim, RTrim$, Trim, Trim$
	String vergelijkregels instellen	Option Compare
	Werken met ASCII- en ANSI-waarden	Asc, Chr$
Variabelen en	Variabelen of constanten declareren	Const, Dim, Global, Static
constanten	Informatie over een variant opvragen	IsDate, IsEmpty, IsNull, IsNumeric, VarType
	Declaratie afdwingen	Option Explicit
	Verstekdatatype instellen	Deftype
Diversen	Omgevingsvariabelen opvragen	Environ, Environ$
	Toetsaanslagen genereren	SendKeys
	Commandoregelargumenten opvragen	Command, Command$
	Laten uitvoeren van nog niet uitgevoerde gebeurtenissen	DoEvents
	Uitvoeren van andere programma's	AppActivate, Shell
	Een geluid produceren	Beep
	Objecttype opvragen	Is

9.7 Systeemobjecten en andere speciale objecten

Van de systeemobjecten worden hier alleen de belangrijkste properties en methoden genoemd. Voor meer informatie verwijs ik U naar de online Help van Visual Basic.

App
De directory waarin het EXE-bestand staat staat opgeslagen in de property Path. De naam van de EXE staat opgeslagen in de property EXEName.

Clipboard
Voor het run-time ophalen van een plaatje van het clipboard (klembord) gebruik je de methode GetData, voor het plaatsen van een plaatje (b.v. een picture property) in het clipboard gebruik je de methode SetData. Voor het ophalen van een tekst gebruik je GetText en voor het plaatsen van een tekst gebruik je SetText. Het resultaat van de functiemethode GetFormat is True als een item van het type opgegeven in de actuele parameter voorkomt in het Clipboard, en anders False. Bevat het klembord tekst, dan zal Clipboard.GetFormat(1) gelijk aan True zijn en Clipboard.GetFormat(2) gelijk aan False, omdat de 1 aangeeft te kijken of er tekst op het klembord staat, en 2 of er een bitmap op het klembord staat.

Debug
Met de Print-methode kun je ernaar schrijven b.v. Debug.Print "Foutje, bedankt!". Bevindt je applicatie zich in de Break-mode dan kun je het Debug-window oproepen met het pull-down menu Window|Debug en kun je de waarden van variabelen opvragen (typ een vraagteken gevolgd door de naam van de variabele) of de applicatie beëindigen (door End in te typen). (Druk daarna wel op de Enter-toets.)

Printer
Voor de communicatie met de printer is er het Printer-object. Deze maakt het mogelijk een (grafische) pagina te beschrijven (vergelijkbaar met een picture box). Schrijven naar de af te drukken pagina gebeurt met de methode Print. Afdrukken gebeurt met de methode EndDoc. Daarnaast kunnen natuurlijk ook het lettertype (FontName), de lettergrootte (FontSize) en een aantal andere weergave-eigenschappen worden ingesteld. Vanaf Visual Basic 5 kunnen ook afbeeldingen worden afgedrukt!

Screen
Belangrijke properties zijn Width en Height resp. de breedte en de hoogte van het scherm (in twips). Tenslotte is er nog MousePointer, waarmee voor het hele scherm de muiscursor kan worden ingesteld b.v. op 11 (zandloper) of 0 (alle zichtbare objecten bepalen de muiscursor). Het aantal twips per pixel in horizontale resp. verticale richting staat opgeslagen in TwipsPerPixelX resp. TwipsPerPixelY.

Overige speciale objecten
De controls in elke form komen in een array voor, de Controls-verzameling. De Controls-verzameling heeft een property, genaamd Count, die het aantal controls aangeeft die de aangegeven form bevat. Met deze verzameling is het mogelijk om b.v. de positie van alle controls in een form te wijzigen.
Ook heeft VB een Forms-verzameling waarin alle geladen (!) forms in de applicatie voorkomen; de eerste is Forms(0), de laatste Forms(Forms.Count-1). Omdat het om geladen forms moet gaan, en het aantal geladen forms tijdens het draaien van de applicatie kan veranderen, is het niet altijd mogelijk te veronderstellen dat een bepaalde Form altijd dezelfde plaats heeft in de Forms-verzameling.

Dialogen
Voor het inlezen van de waarde van een variabele kan InputBox$ worden gebruikt. Voor het weergeven van tekst MsgBox. Daarnaast kunnen op een gemakkelijke manier een aantal standaard Windows-dialoogvensters worden getoond nl. die voor het opvragen van de naam van een bestand, van een lettertype, van een kleur en van een printer. Hiertoe moet een common diloag control op het form worden gezet! Zie verder de online Help van VB.

10. HET VISUAL BASIC PRACTICUM

Het practicum Visual Basic heeft als hoofddoel je de basisbeginselen van het programmeren bij te brengen. Bovendien krijg je inzicht in de mogelijkheden van Visual Basic voor het maken van applicaties t.b.v. produktsimulaties en gebruiksonderzoek, voordat het produkt daadwerkelijk gemaakt wordt. Helaas kunnen we je in het practicum slechts laten kennismaken met die onderdelen van Visual Basic die je nodig hebt tijdens het practicum.

In de eerste drie oefeningen maak je kennis met Visual Basic. De eindopdracht, die tijdens de derde oefening wordt uitgereikt, bestaat er uit, dat je een applicatie maakt met Visual Basic gebaseerd op een ontwerp bestaande uit algoritmen, die je van te voren ter goedkeuring inlevert.

De colleges worden gegeven als voorbereiding op het practicum. Op de colleges worden voorbeelden besproken en worden kleine opdrachten gegeven, en wat grotere opdrachten als huiswerk. Werk serieus aan deze opdrachten want ze zorgen ervoor dat je beslagen ten ijs komt wanneer je de eindopdracht moet gaan maken. Het college over Algoritmen is erg belangrijk, omdat je dan kennis maakt met het maken van algoritmen.

De manier waarop (het ontwerp van) de Zeer Eenvoudige Koffie-Automaat werd gemaakt was enigszins informeel. Hier geef ik aan hoe het in te leveren ontwerp eruit moet zien. Eerst bespreek ik en laat zien hoe een algemeen algoritme eruit moet zien.

10.1 Representatie van een algemeen algoritme

Opbouw van een algoritme
Een algoritme bestaat uit een naam, een omschrijving van de werking van het algoritme, (een lijst van) variabelen en opdrachten. De naam van het algoritme moet toepasselijk worden gekozen; uiteraard moet de omschrijving duidelijk zijn.
In de lijst van variabelen moeten alle variabelen worden genoemd die in de opdrachten worden gebruikt.

De variabelen
Van elke variabele moet de naam worden weergegeven, de categorie waartoe deze behoort en wat de functie/gebruikswijze van de variabele is. De naam moet toepasselijk zijn gekozen b.v. de naam van de grootheid waarvan het de waarde zal bevatten. Een variabele kan tot de volgende categorieën behoren:

- invoer
- uitvoer
- globaal invoer
- globaal uitvoer
- lokaal

Variabelen die tot de in- of uitvoercategorie behoren worden ook wel formele parameters genoemd. Het verschil tussen in- en uitvoerparameters is, dat invoerparameters niet door het algoritme worden gewijzigd, terwijl uitvoerparameters wel worden gewijzigd. (Invoerparameters worden als waarde-, uitvoerparameters als variabele-parameters geïmplementeerd.)

Variabelen die niet als in- cq. uitvoer door het algoritme worden gebruikt, maar wel gebruikt worden in andere delen van het programma vormen de globale variabelen. Het is nuttig onderscheid te maken tussen globale variabelen die alleen gebruikt worden zonder te worden gewijzigd en globale variabelen die daadwerkelijk worden gewijzigd. Bij aanpassingen aan het ontwerp achteraf is dan snel te zien welke globale variabelen het algoritme aanpast cq. gebruikt. Het verdient overigens de voorkeur geen gebruik te maken van dergelijke globale variabelen, en alle uitwisseling via parameters plaats te laten vinden; met name wanneer een algoritme wordt ontwerpen om ook in andere programma's moet kunnen worden gebruikt.
Variabelen die het algoritme nodig heeft om zijn werking naar behoren te kunnen vervullen moeten als lokale variabelen worden gedeclareerd. Een dergelijke variabele, d.w.z. de waarde ervan, is alleen beschikbaar binnen het algoritme. De naam ervan mag hetzelfde zijn als de naam van niet door het algoritme gebruikte variabelen alhoewel dit verwarring kan geven. Het is dan overigens niet langer mogelijk om een andere variabele – uit een ander algoritme -met dezelfde naam te gebruiken!

Uitvoeren van een algoritme
Een algoritme kan worden uitgevoerd vanuit een – meestal ander – algoritme door deze aan te roepen. In de aanroep moeten de door het algoritme te gebruiken in- en uitvoerparameters worden opgegeven. De plaats van een actuele parameter in deze lijst bepaalt voor welke formele parameter deze zal worden gebruikt tijdens de uitvoering. Als het om de actuele parameter behorende bij een invoerparameter hoort, dan wordt de actuele parameter als expressie beschouwd, en wordt de waarde ervan als te gebruiken startwaarde van de formele parameter gebruikt. Een actuele parameter behorende bij een uitvoerparameter moet de naam van een variabele zijn, die bij uitvoering van het algoritme wordt gesubstitueerd voor de formele parameter. (Bij het opstellen van een algoritme weet je natuurlijk nog niet welke variabele dit zal zijn, maar je kunt toch al een algoritme opstellen door de formele parameter te benoemen. Het specifieke mechanisme dat wordt gebruikt om bij uitvoering ervoor te zorgen dat de juiste variabele wordt gebruikt is niet van belang!)

Opdrachten
De opdrachten beschrijven hoe het algoritme uit de invoer de uitvoer bepaalt. De opdrachten vallen in de volgende categorieën:

Opdrachtsoort	**Werking/Functie**	**Notatie**
Toekenning	Inhoud (=waarde) van een variabele wijzigen	maak {variabele} gelijk aan {expressie}
Invoer (lezen)	Inhoud van variabele door lezen (=aan gebruiker vragen) wijzigen	lees {variabelen}
Uitvoer (schrijven)	Mededelen aan gebruiker	Schrijf {tekst}
Aanroep	(opdrachten in) ander algoritme uitvoeren	voer {algoritme} {lijst actuele parameters} uit
Selectie	Opdracht(en) voorwaardelijk wel/niet uitvoeren	als {voorwaarde}[[,] dan] {opdrachten} [anders {opdrachten}]
Onvoorwaardelijke herhaling	Opdrachten een 'vast' aantal keer uitvoeren	doe voor {tellervariabele}={start} tot {eind} [met stap {stapgrootte}] {opdrachten}
Voorwaardelijke herhaling	Opdrachten een conditioneel aantal keer uitvoeren	zolang {voorwaarde}[doe] {opdrachten}

Voor tussen accolade staande delen moet worden ingevuld wat tussen de accolades omschreven is (de accolades dan wel weglaten).
Voor {variabele} moet steeds de naam van een variabele worden ingevuld die één waarde heeft. Dit kan dus ook een bepaald element van een rij of veld van een record zijn.
Een element van een rij moet worden omschreven als 'element {elementexpressie} van {rijvariabele}' of (in pseudocode) als '{rijvariabele}({elementexpressie})'.
Een veld van een recordvariabele als 'veld {veld} van {recordvariabele}' of (in pseudocode) met '{recordvariabele}.{veld}'.
Tussen vierkante haken staande delen zijn optioneel.
Omdat voorwaardelijke herhalingen niet worden afgesloten moet de opdrachten die dan herhaald moeten worden uitgevoerd, ingesprongen worden weergegeven. Hetzelfde geldt voor de opdrachten die in het dan- en anders-gedeelte van een selectie staan; zelfs wanneer op het dan-gedeelte een anders-gedeelte volgt.

Tips voor het opstellen van een algoritme

- Maak altijd eerst een kladversie.
- Gebruik daarbij twee vellen papier: één voor de lijst met variabelen, één voor de overige gegevens.
- Denk goed na over wat er van het algoritme wordt verwacht.
- Omschrijf dit beknopt en leidt hieruit een goede naam af. Schrijf deze op.
- Leid uit de omschrijving af welke invoer- en uitvoergegevens het algoritme zal hebben. (Ik ga er van uit dat er geen globale variabelen worden gebruikt!) Een algoritme dat als eerste zal worden uitgevoerd, heeft geen in-/uitvoerparameters en zal gegevens in moeten lezen of zelf een startwaarde geven.
- Op dit moment zal het nog niet mogelijk zijn om te bepalen welke variabelen lokaal dienen te worden gebruikt. Tijdens het bepalen van de opdrachten zal duidelijk moeten worden welke waarden moeten worden bewaard; voor elke te bewaren waarde moet een lokale variabele aan de lijst met variabelen worden toegevoegd. Voor elke onvoorwaardelijke herhaling is een tellervariabele nodig! Het is niet nodig voor elke onvoorwaardelijke herhaling een aparte tellervariabele te gebruiken; het is vaak mogelijk een eerder gebruikte tellervariabele nog eens te gebruiken, als die op dit moment niet gebruikt wordt!
- Ga na wanneer de eindtoestand bereikt zal zijn en hoe die kan worden bereikt. Probeer van de een of andere oplossingsstrategie gebruik te maken. Probeer vast te stellen welke (tussen)toestanden bereikt moeten worden om het einddoel dichterbij te brengen en hoe die bereikt kunnen worden. Zelfs al weet je niet precies hoe uit de ene toestand de andere bereikt moet worden dan nog zul je in staat zijn te omschrijven wat er moet gebeuren. Dit zou dan in een nog te schrijven algoritme kunnen worden beschreven. Werk eerst de delen waarvoor je wel een oplossing ziet, stort je daarna op de nog niet uitgewerkte delen.
- Voeg aan de opdrachten commentaar toe waarin de (bereikte) toestand wordt omschreven.
- Zet de opdrachten voldoende ver uit elkaar zodat nog opdrachten tussen kunnen worden gevoegd.
- Controleer de werking van de kladversie niet alleen met met eenvoudige, doch generieke invoer, maar ook met speciale invoer.
- Maak tenslotte een officiële versie in het net om in te leveren voorzien van je na(a)m(en).

10.2 Voorbeelden van algemene algoritmen

In deze paragraaf geef ik een aantal algoritmen, waarvan er een aantal op het college zullen worden toegelicht.

Het berekenen van de som van twee getallen (1)

Naam	:	SomTweeGetallen
Functie	:	retourneren som van eerste twee parameters in derde parameter
Variabelen	:	A invoer het eerste getal
		B invoer het tweede getal
		Som uitvoer de som van de twee getallen
Opdrachten	:	Maak Som gelijk aan A plus B (in VB: Som=A+B)
Toelichting	:	De waarde van een variabele kan in een toekenning met links van het toekenningsteken de naam van de variabele en rechts van het toekenningsteken een expressie (formule) wiens waarde aan de variabele moet worden gegeven. Omdat Som in het algoritme een waarde krijgt, moet deze als uitvoerparameter worden opgegeven.

Het berekenen van de som van twee getallen (2)

Naam	:	SomTweeGetallen
Functie	:	Retourneren van de som van twee getallen
Variabelen	:	A invoer het eerste getal
		B invoer het tweede getal
Opdrachten	:	Maak SomTweeGetallen gelijk aan A plus B (in VB: SomTweeGetallen=A+B)
Toelichting	:	Een uitvoerparameter kan worden uitgespaard door de som als resultaat te retourneren.

Het berekenen van de inhoud van een bol

Naam	:	InhoudBol
Functie	:	de inhoud van een bol met straal r als resultaat retourneren
Variabelen	:	R invoerparameter de straal van de bol
Opdrachten	:	Maak InhoudBol gelijk aan 16 maal de arctangens van 1 maal R tot de macht 3 gedeeld door 3 (in VB: InhoudBol=16*atan(1)*R^3/3)
Toelichting	:	De inhoud van een bol met straal R is $(4/3)\pi R^3$. Voor π kan natuurlijk een benadering worden gebruikt b.v. 3,14159, maar we kunnen ook 4 maal de arc tangens van 1 gebruiken als benadering voor π. We kunnen InhoudBol een resultaat laten retourneren door de te retourneren waarde aan InhoudBol, de naam van het algoritme toe te kennen. Een algoritme dat een waarde retourneert zal als functie worden geimplementeerd. Ik veronderstel dat er een operator ^ is die kan machtsverheffen met een hogere prioriteit dan het delen met /, anders zou de opdracht niet goed zijn.

Het verwisselen van de inhoud van twee variabelen

```
Naam        : VerwisselInhoud
Functie     : Het verwisselen van de inhoud van twee variabelen
Variabelen  : A   uitvoerparameter   na uitvoering gelijk aan de (begin)waarde van b
              B   uitvoerparameter   na uitvoering gelijk aan de (begin)waarde van a
              T   lokaal             voor het opslaan van de (begin)waarde van a
Opdrachten  : Maak T gelijk aan A    ' opslaan van beginwaarde A in T
              Maak A gelijk aan B    ' A de waarde van B geven, nu ongelijk aan T!
              Maak B gelijk aanT     ' B gelijkmaken aan de beginwaarde van A in T!
Toelichting : Beide variabelen moeten als uitvoerparameter worden opgegeven, want ze
              veranderen van waarde! Er is een extra variabele nodig om daarin de waarde
              van de ene variabele op te slaan. Vervolgens is het mogelijk om die variabele
              gelijk te maken aan de waarde van de andere variabele, die dan vervolgens
              aan de beurt is om de oorspronkelijke waarde te krijgen.
```

Het op volgorde zetten van drie variabelen

```
Naam        : ZetOpVolgorde
Functie     : Het op volgorde zetten van drie variabelen
Variabelen  : A   uitvoerparameter   na afloop gelijk aan de kleinste waarde
              B   uitvoerparameter   na afloop gelijk aan de middelste waarde
              C   uitvoerparameter   na afloop gelijk aan de grootste waarde
Opdrachten  : als B kleiner dan A, dan
                    Voer verwissel(A,B) uit        ' nu is: A<=B
              als C kleiner dan B, dan
                    Voer verwissel(B,C) uit        ' B<=C, maar misschien is B < A
                    als B kleiner dan A, dan
                          Voer verwissel(A,B) uit
Toelichting : Na uitvoering moet A kleiner dan of gelijk aan B zijn en B kleiner dan of gelijk
              aan C (ook te noteren als: A<=B<=C). De aanpak houdt in dat er eerst voor
              wordt gezorgd dat A<=B. De derde waarde, in C, moet dan ook nog op de
              goede plaats worden 'tussengevoegd'.
              Opvallend is dat geen extra lokale variabele nodig zijn. Ja, Verwissel heeft er
              wel een, nl. T, maar daar hoeft ZetOpVolgorde geen weet van te hebben.
```

Het bepalen van het maximum van drie getallen

```
Naam        : MaximumDrieGetallen
Functie     : Retourneren van de grootste van drie waarden
Variabelen  : A   invoer   het eerste getal
              B   invoer   het tweede getal
              C   invoer   het derde getal
Opdrachten  : als A>B, dan ' volgorde CBA of BCA of BAC
                    als A>C dan ' volgorde CBA of BCA
                          maak MaximumDrieGetallen gelijk aan A
                    Anders ' volgorde BAC
                          Maak MaximumDrieGetallen gelijk aan C
              Anders ' volgorde CAB of ACB of ABC
                    als B>C, dan ' volgorde CAB of ACB
                          maak MaximumDrieGetallen gelijk aan B
                    anders ' volgorde ABC
                          maak MaximumDrieGetallen gelijk aan C
Toelichting : Natuurlijk is het ook mogelijk na te gaan welke van de zes mogelijke volgordes
              waar is, maar dan staan er in totaal 12 vergelijkingen in de opdrachten en 6
              toekenningen, terwijl 3 vergelijkingen en 4 toekenningen voldoende is!
```

Bepalen of twee rechthoeken elkaar overlappen

Naam	:	Overlappend
Functie	:	Retourneert True als de twee rechthoeken elkaar overlappen, anders False
Variabelen	:	L1 invoer X-coordinaat linkerbovenhoek rechthoek 1 B1 invoer Y-coordinaat linkerbovenhoek rechthoek 1 R1 invoer X-coordinaat rechterbenedenhoek rechthoek 1 O1 invoer Y-coordinaat rechterbenedenhoek rechthoek 1 L2 invoer X-coordinaat linkerbovenhoek rechthoek 2 B2 invoer Y-coordinaat linkerbovenhoek rechthoek 2 R2 invoer X-coordinaat rechterbenedenhoek rechthoek 2 O2 invoer Y-coordinaat rechterbenedenhoek rechthoek 2
Opdrachten	:	als R1<L2 of R2<L1 of B1>O2 of B2>O1, dan Maak Overlappend gelijk aan False ' geen overlap Anders Maak Overlappend gelijk aan True ' wel overlap
Toelichting	:	Hier moet enige inspanning worden geleverd om te bepalen welke informatie aangeleverd moet worden van de rechthoeken; moeten dit de twee tegenover elkaar gelegen hoekpunten zijn of een hoekpunt en de lengte en de breedte? Dan is het ook nog mogelijk om de gegevens van de rechthoeken in een gegevensstructuur te plaatsen en dus slechts twee invoerparameters te hoeven aanleveren. Bepalen of de twee rechthoeken elkaar overlappen is relatief gemakkelijk ervan uitgaande dat de twee tegenoverelkaargelegen hoekpunten (linkerbovenhoek en rechterbenedenhoek) gegeven zijn! De opdracht kan worden gevonden door te bepalen wanneer de rechthoeken elkaar juist niet overlappen. **Bedenk zelf hoe met één opdracht zou kunnen worden volstaan!**

Het bepalen van de som van een rij getallen

Naam	:	SomRijGetallen
Functie	:	Retourneert de som van een rij getallen
Variabelen	:	R invoer de rij getallen N invoer het aantal te gebruiken getallen in de rij Index lokaal de index van het eerste nog niet gebruikte rij-element Som lokaal de som
Opdrachten	:	Maak Som gelijk aan 0 Doe voor Index=1 tot N Maak Som gelijk aan Som plus element Index van R ' nu is Som gelijk aan de som van de eerste Index elementen Maak SomRijGetallen gelijk aan Som
Toelichting	:	Een variabel aantal elementen kan niet op dezelfde manier worden opgeteld als twee getallen nl. in één toekenning, omdat niet bekend is hoeveel plus-tekens moeten worden gebruikt. Een onvoorwaardelijke lus is dan het aangewezen element. Zie hoe de te bereiken tussentoestand hier het hebben berekend van de som van de eerste Index elementen. Nadat de som is berekend voor Index = N, zijn we klaar en kan Som als resultaat worden geretourneerd.

Het inlezen van en weergeven van het gemiddelde van een rij getallen (1)

```
Naam        : ToonGemiddelde
Functie     : Inlezen van een rij getallen en er het gemiddelde van weergeven
Variabelen  : R     lokaal  voor de opslag van maximaal 100 getallen
              N     lokaal  het aantal ingelezen getallen
Opdrachten  : Maak N gelijk aan 0 ' nog geen getallen ingelezen
              Herhaal
                    Maak N gelijk aan N plus 1
                    Lees element N van R
              Totdat N gelijk aan 100 of element N van R gelijk aan 0
              ' het aantal ingelezen elementen is 1 kleiner als laatste gelijk aan 0
              als element N van R gelijk is aan 0, dan
                    Maak N gelijk aan N-1
              als N groter is dan 0, dan
                    Schrijf "Het gemiddelde is: " & SomRijGetallen(R,N)/N
              Anders
                    Schrijf "Geen getallen  ingevoerd!"
Toelichting : In veel omgevingen moet van een rij al vooraf worden aangegeven hoeveel
              elementen deze zal kunnen bevatten. Hier koos ik ervoor om maximaal 100
              getallen in de rij te kunnen opslaan: Dit is dan dus ook het grootste aantal
              getallen waarvan het gemiddelde kan worden berekend.
              De elementen worden een voor een ingelezen. Om klaar te staan voor het
              inlezen van het volgende element moet het plaatsnummer van het te lezen
              element, als opgeslagen in N, met 1 worden opgehoogd. Hier wordt
              verondersteld dat de gebruiker door het opgeven van 0 aangeeft dat er geen
              verdere getallen meer hoeven te worden ingelezen.
              Na het uitvoeren van de Doe-totdat-herhaling, een voorwaardelijke lus, zal het
              aantal te gebruiken elementen 1 kleiner dan N zijn als het laatste ingelezen
              getal gelijk aan 0 was.
```

Het inlezen van en weergeven van het gemiddelde van een rij getallen (2)

```
Naam        : TonenGemiddelde
Functie     : Weergeven van het gemiddelde van een rij in te lezen getallen
Variabelen  : Getal lokaal  ingelezen getal
              N     lokaal  aantal ingelezen getallen
              Som   lokaal  som van de ingelezen getallen
Opdrachten  : Maak N gelijk aan 0 ' nog geen getallen ingelezen
              Herhaal
                    Lees Getal
                    Maak N gelijk aan N+1
                    Maak Som gelijk aan Som plus Getal
              Totdat Getal=0
              als N groter dan 1, dan
                    Schrijf "Het gemiddelde is: " & Som/(N-1)
              Anders
                    Schrijf "Geen getallen ingevoerd!"
Toelichting : Hier wordt het gemiddelde weergegeven van een – vooraf – onbekend aantal
              getallen door de som direct te berekenen. Het is niet erg om Getal bij Som op
              te tellen als ie 0 is, Som verandert daardoor niet! Wel is het aantal ingelezen
              getallen altijd 1 kleiner dan N!
```

Het bepalen van het aantal woorden in een tekst

Naam	: AantalWoordenIn		
Functie	: Retourneren van het aantal woorden in een tekst		
Variabelen	: Tekst	invoer	de tekst
	Index	lokaal	de index van het teken dat we willen bekijken
	Aantal	lokaal	aantal gevonden woorden in de tekst

```
Opdrachten : Maak Aantal gelijk aan 0      ' nog geen woorden aangetroffen
             ' bepaal van elk teken in Tekst of het het eerste teken van een woord is
             Doe voor Index=1 tot de lengte van Tekst
                    ' is het teken op positie Index het eerste teken van een woord?
                    als het teken op plaats Index van Tekst ongelijk aan " " is _
                       en het teken op plaats Index-1 van Tekst is een spatie of _
                      Index=1, dan
                           Maak Aantal gelijk aan Aantal+1 ' een nieuw woord!
             Maak AantalWoordenIn gelijk aan Aantal ' resultaat AantaWoorden
```

Toelichting : In dit geval is het niet nodig dat Tekst een uitvoerparameter is, want deze wordt niet gewijzigd. Het is weer mogelijk om een resultaat te retourneren.
Hier ziet U hoe een onvoorwaardelijke herhaling (Doe voor ...) wordt gebruikt om over alle tekens in de tekst te 'lopen'. Als tellervariabele wordt Index gebruikt. Het inspringen wordt gebruikt om aan te geven welke opdrachten zich binnen de herhaling en welke zich erbuiten bevinden. De laatste opdracht bevindt zich buiten de herhaling!
In dit algoritme wordt ervan uitgegaan, dat het aantal tekens in Tekst bekend is (de lengte van Tekst) of gemakkelijk kan worden bepaald. Bezit de straks te gebruiken omgeving hiertoe geen makkelijke manier b.v. een functie die als resultaat het aantal tekens in een tekst retourneert, dan zal alsnog zelf een algoritme hiervoor moeten worden geschreven! Hetzelfde geldt voor het teken op plaats Index.
Voor het spatieteken wordt " " gebruikt, twee dubbele aanhalingstekens met daartussen een spatie. Dit is in overeenstemming met Visual Basic, waar constante tekst tussen dubbele aanhalingstekens moet worden geplaatst om het te onderscheiden van namen van variabelen.
Een teken is pas het begin van een nieuw woord als het teken dat ervoor staat een spatie is of als er geen teken voorstaat. Het zou niet goed zijn als elk teken ongelijk aan een spatie als het begin van een woord wordt gezien!
Het is overigens ook mogelijk steeds het vorige teken te onthouden en na te gaan of dit wel of niet een spatie is. Door voor de herhaling deze variabele gelijk aan een spatie te maken hoeft dan niet meer getest te worden of Index gelijk aan 1 is.

De grootste gemene deler van twee positieve getallen

Naam	: Ggd		
Functie	: Retourneert de grootste gemene deler van twee positieve getallen		
Variabelen	: A	invoer	het grootste van de twee getallen
	B	invoer	het kleinste van de twee getallen

```
Opdrachten : als B gelijk aan 1 is, dan
                    Maak Ggd gelijk aan A
             Anders
                    Maak Ggd gelijk aan Ggd(B,A Mod B)
```

Toelichting : Dit is een algoritme, dat zichzelf aanroept, een zgn. **recursief** algoritme.
Het hier gebruikte algoritme is vernoemd naar Euclides, omdat die het geschreven schijnt te hebben. Recursiviteit kan hele elegante oplossingen opleveren. Overigens: recursiviteit kan ook m.b.v. herhalingen worden gerealiseerd. **Bedenk zelf de niet-recursieve versie van dit algoritme.**

10.3 Representatie van algoritmen in Visual Basic

Om bovenstaande algoritmen om te zetten in werkende Visual Basic programma-onderdelen is kennis van Visual Basic nodig. Je moet weten:

- Elk algoritme moet hetzij in een aparte Sub (procedure) of in een Function (functie) worden gezet. Een algoritme dat een resultaat retourneert wordt een Function.
- De naam van het algoritme wordt de naam van de Sub/Function.
- De parameterlijst van de Sub/Function moet alle parameters bevatten. Vóór elke invoer-parameter moet het keyword **ByVal** worden geplaatst, om aan te geven dat het om een waardeparameter gaat! Wanneer de parameter een rij is, moet achter de naam van de rij een haakje openen en een haakje sluiten worden geplaatst; de rij moet als variabele-parameter worden opgegeven, dus zonder ByVal ervoor!
- Alle lokale variabelen moeten met het Dim-statement worden gedeclareerd binnen de Sub/Function, bij voorkeur aan het begin, voor de opdrachten.
- Het verdient de voorkeur van alle variabelen het type op te geven dat aangeeft welke soort waarden erin mogen worden opgeslagen. Kies het type Integer voor de opslag van gehele getallen, Single of Double voor de opslag van reële getallen, String*1 voor de opslag van een teken, en String voor de opslag van tekst. Zet achter de naam van een rijvariabele tussen ronde haken een bereik van toegestane indices (b.v. 1 To 100 voor elementen 1 t/m 100). Plaats achter de parameterlijst van een functie het type van het resultaat dat wordt geretourneerd. Gloable variabelen kun je met het Dim-statement declareren in het declaratie-gedeelte. Hier kunnen ook constanten en eigen recordtypen (alleen in een module) worden opgegeven.
- Elders in deze handleiding kun je zien hoe bepaalde soorten opdrachten moeten worden gerepresenteerd in Visual Basic.
- Een element van een rij geef je aan met '{rij}({elementexpressie}); een veld van een recordvariabele met '{recordvariabele}.{veld}'.
- De makkelijkste manier om de waarde van een variabele in te lezen is met de functie InputBox$ (zie de online VB Help). Het resultaat ervan is altijd tekst. Visual Basic zorgt wel voor een eventuele omzetting, maar je kunt ook een van de omzettingsfuncties (Cint, Csng, Cdbl) gebruiken.
- De makkelijkste manier om de gebruiker iets mee te delen is met de procedure MsgBox (zie de online VB Help). Minimaal moet een tekst aan MsgBox ter weergeving worden gegeven.
- Dergelijke algemene subroutines kunnen het beste in een Visual Basic module worden geplaatst. Visual Basic voert als eerste altijd de Sub Main uit. Dit betekent, dat een van de algoritmen Main moet heten en de opdrachten moet bevatten, die als eerste moeten worden uitgevoerd en die voor het aanroepen van alle andere subroutines zorgen.

Tijdens een van de colleges zal het computerprogramma PSD & Code worden gedemonstreerd dat in staat is uit dergelijke algoritmen Visual Basic code te genereren. PSD & Code kan gebruikt worden om het verband tussen algoritme en bijbehorende code te gaan begrijpen, waardoor het vervolgens gemakkelijker wordt gemaakte algoritmen in Visual Basic om te zetten. Het maken van de algoritmen echter blijft een proces dat niet geautomatiseerd kan worden!

Ter illustratie hier de uitwerking van de laatste twee voorbeelden.

Inlezen van een rij getallen en tonen van het gemiddelde

```
Dim RijGetallen(1 To 100) As Double ' declaratie van de rij van max. 100 getallen
Function SomVanRijGetallen(R() As Double, N As Integer) As Double
Dim Index As Integer
    For Index=1 To N
        Som=Som+R(Index)
    Next Index
    SomVanRijGetallen=Som
End Function
Sub Main()
Dim N As Integer
    ' inlezen van de rij getallen
    N=0
    Do
        N=N+1
        RijGetallen(N) = CDbl(InputBox$("Typ getal #" & N & _
                                        "in (0 om te stoppen).","",0)
    Loop Until N=100 Or RijGetallen(N)=0
    ' zorgen dat N ook echt het aantal ingelezen getallen is
    If RijGetallen(N)=0 Then N=N-1 ' If-Then mag op één regel (geen End If nodig)
    If N>0 Then
        MsgBox "Het gemiddelde is: " & SomRijGetallen(RijGetallen,N)/N
    Else
        MsgBox "Geen getallen ingevoerd!"
    End If
End Sub
```

Het bepalen van het aantal woorden in een tekst

```
Function AantalWoordenIn(ByVal Tekst As String) As Integer
Dim Aantal As Integer: Dim Index As Integer
    Aantal=0
    For Index=1 to Len(Tekst)
        If Mid$(Tekst,Index,1)<>" " And _
           (Index=1 Or Mid$(Tekst,Index-1,1)=" ") Then
            Aantal=Aantal+1
        End If
    Next Index
    AantalWoordenIn=Aantal ' resultaat retourneren!
End Function
Sub Main() ' testprogramma!
Dim IngetypteTekst As String
    Do
        IngetypteTekst=InputBox$("Typ een tekst in (geen om te stoppen).")
        MsgBox "De tekst bevat " & AantalWoordenIn(IngetypteTekst) & "woorden."
    Loop Until Len(IngetypteTekst)=0
End Sub
```

10.4 De eindopdracht

Moderne Windows-programma's zijn event-gestuurd. Vensters bevatten gewoonlijk meerdere invoergevoelige elementen en gebruikersacties wekken een reactie van het programma op. Het programma kan er echter niet langer vanuit gaan in welke volgorde de gebruikersacties precies zullen optreden en het programma moet daarom zorgvuldiger worden ontworpen dan ee traditioneel programma waarin precies vaststond wanneer de gebruiker invoer mocht plegen.
In een event-gestuurd programma, als met Visual Basic kunnen worden gemaakt, veroorzaakt elke mogelijke gebruikersactie de uitvoering van een deel van het programma dat daarom ***event handler*** wordt genoemd. Het is moeilijk om te spreken van één programma. Een dergelijke programma is niet langer gebaseerd op één hoofdalgoritme, maar op een aantal algoritmen voor *event handlers*, elk een reactie op een gebruikersactie uitgevoerd op een user interface element. Elk algoritme moet een naam krijgen of een omschrijving waaruit blijkt op welke gebruikersactie wordt gereageerd.
Uit de specificatie moet daarom eerst de user interface worden ontworpen. Hieruit kan een lijst van te ontwerpen event handlers worden opgesteld. Vervolgens kan dan voor elke event-handler een algoritme worden opgesteld. **Duidelijk moet worden vermeld op welke event het algoritme een reactie is!**

De eindopdracht omhelst het schrijven en implementeren van dergelijke algoritmen. De user interface alsmede de lijst met gebruikersacties waarop gereageerd moet worden hoeft niet te worden geschreven. De namen in deze lijst zijn de namen van de te ontwikkelen algoritmen. **Elk algoritme mag niet meer dan een bladzijde beslaan. Wel mogen daarin weer andere algoritmen worden aangeroepen (die ook op één maximaal bladzijde omschreven staan). Geef in de omschrijving van de werking van elk algoritme aan vanuit welk(e) algoritme(n) dit algoritme wordt aangeroepen!**

Maak bovendien een aparte lijst van alle globale variabelen (inclusief de functie die ze in het programma vervullen).

In de lijsten met variabelen zullen ook properties van user interface elementen aangeduid worden. Geef daarvan als categorie 'property invoer' of 'property uitvoer' (als de property van waarde verandert in het algoritme!). Het is niet nodig om ook nog aan te geven dat ze globaal zijn!

Plaats op elk ingeleverd vel je namen, studienummers en groepscode.

Aangezien het in de eindopdracht om het simuleren van een product met knoppen die niet echt uitgezet kunnen worden (anders dan wanneer het product uitstaat), is het niet toegestaan de knoppen uit te zetten op de manier zoals in deze handleiding gebruikt. Je zult een andere manier moeten gebruiken. Het gemakkelijkst is het om daarvoor één variabele te gebruiken wiens waarde bepaalt hoe gereageerd moet worden. Een dergelijke variabele wordt ook wel een statusvariabele of vlag genoemd: hij informeert (vlagt) over de toestand waarin het apparaat zich bevindt.

10.5 Opslaan van de Visual Basic applicatie

Een Visual Basic applicatie wordt een project genoemd.
Elk project bestaat uit een aantal forms en modules (tot nu toe heb je alleen met forms te maken gehad). De namen ervan en de namen van de bestanden waarin ze staan opgeslagen staan opgeslagen in het projectbestand, een soort inhoudsopgave. Gaat het projectbestand verloren, dan ben je alleen de inhoudsopgave kwijt, meer niet.

Zorg ervoor dat je nooit je forms (en modules) kwijtraakt. De informatie en aanwijzingen hieronder kunnen je daarbij helpen.

Sla je project regelmatig (b.v. elk kwartier) zowel op je eigen schijf (H:) als op de lokale schijf (C:) op.

Een form bevat zowel een venster als bijbehorende code. (Deze code bestaat weer uit declaraties van constanten, variabelen, van zelfgemaakte functies en procedures, als ook de zelfgemaakte code behorende bij event-procedures). Een module bevat alleen code. Het verschil tussen de code in een form en de code in een module is dat de code in een module beschikbaar is voor alle forms in het project, terwijl de code in een form alleen binnen de form bekend is en niet in andere forms.
Elke form en module in je project wordt bij het opslaan in een apart bestand geplaatst. Voor het bestandsbeheer van Visual Basic is **elke form en module** gewoon een file, en **kan** dus **met Save File (As) worden opgeslagen**, met Add File worden toegevoegd aan het project en met Remove File uit je project worden verwijderd. Overigens: verwijderen betekent niet dat het bestand op schijf verwijderd wordt: het form of de module wordt eenvoudigweg uit het project verwijderd; het zal daarna nog gewoon op schijf staan.
Omdat je in het VB-practicum waarschijnlijk niet met modules te maken krijgt beschrijf ik hier alleen de situatie voor forms. De omgang met modules - moet uit het bovenstaande blijken - is (vrijwel) identiek.

Het opslaan van de projectgegevens doe je m.b.v. Save Project of Save Project As. Als alle bestanden waarin de forms en de projectgegevens moeten worden opgeslagen bekend zijn worden alle forms en de projectgegevens opgeslagen in de huidige bestanden.
Als het project forms bevat die nog niet zijn opgeslagen dan zal Visual Basic je vragen om de namen van de bestanden waaronder deze forms moeten worden opgeslagen. Visual Basic kiest zelf een naam en geeft deze de extensie .FRM. Je mag de voornaam veranderen en het pad, maar **verander nooit en te nimmer de extensie .FRM**. Nadat alle forms zijn opgeslagen vraagt Visual Basic je om de naam van het bestand waarin de projectgegevens moeten worden opgeslagen. Deze geeft Visual Basic automatisch de extensie .VBP. Ook hier geldt: **verander nooit en te nimmer de extensie .VBP!** Overigens: de projectgegevens bevatten o.a. de namen van de bestanden waarin de forms staan opgeslagen. Als je dus een form in een ander bestand plaatst (met File Save As) dan moet je ook altijd daarna het project nog een keer opnieuw opslaan teneinde ervoor te zorgen dat de projectgegevens kloppen. Ga je andersom te werk dan klopt de inhoudsopgave niet meer!

Samenvatting Visual Basic 5.0 Step by Step, les 1 [VBSbS]

Om	Doe
Visual Basic op te starten	Klik op het Visual Basic programma-icoon of kies Visual Basic onder Programs.
De omschrijving van de functie van een knop te zien	Plaats de muiscursor boven de knop.
Een bestaand project te openen (in te lezen)	Kies File|Open Project.
Een nieuw project beginnen	Kies File|New Project.
Een programma uitvoeren (draaien)	Klik op de startknop in de knoppenbalk of druk op F5.
Verplaatsen van de toolbox (met beschikbare controls)	Druk de muisknop in boven de titelbalk ervan en sleep de toolbox naar een andere plaats.
Eigenschappen (*properties*) wijzigen	Open eerst zonodig het Properties Window; selecteer vervolgens het element waarvan je de eigenschappen wilt wijzigen; klik vervolgens op de te wijzigen eigenschap en verander de waarde ervan (meerdere manieren mogelijk).
Weergeven van het Projectvenster	Klik op de Project Explorer button op de knoppenbalk en dubbelklik dan op de titelbalk van het Projectvenster.
Een user interface (venster) maken	Zet controls (uit de toolbox) op het form, en pas hun eigenschappen naar wens aan. Het form en de meeste controls zijn gemakkelijk met de muis kleiner of groter te maken.
Een object verplaatsen	Versleep het object.
De afmetingen van een object wijzigen	Selecteer het object, en versleep het vierkantje aan de kant die je wilt veranderen.
Een object (control) verwijderen	Selecteer het object en druk op de Del-toets.
Het Codevenster openen	Dubbelklik op het Form of op een van de controls, of klik op het Codevenster-icoon linksboven in het Projectvenster.
Programmacode invoeren (schrijven) (om tijdens het uitvoeren van het programma te laten uitvoeren als een bepaalde gebeurtenis plaatsvindt)	Typ toepasselijke opdrachten in in de *event handler* (tussen Sub en End Sub) die wordt uitgevoerd als de bijbehorende gebeurtenis (als het klikken op een knop) plaatsvindt.
Een programma opslaan (op schijf)	Kies File|Save Project of klik op de Save Project-knop op de knoppenbalk. (Sla het project op op je eigen (H:) schijf!)
Van je applicatie een *executable* maken	Kies File|Make *filename.exe*.
Visual Basic afsluiten	Kies File|Exit of druk op de Sluiten-knop.
Een project opnieuw ophalen	Kies File|Open Project of kies het bestand in de lijst van recente appliaties.

Samenvatting Visual Basic 5.0 Step by Step, les 2 [VBSbS]

Om	Doe
Een *text box* (invulveld) aan te maken	Klik op de TextBox control in de toolbox; druk de linkermuisknop in boven het form, verplaats de muis, laat de muisknop los waar de andere hoek van de text box moet komen.
Een *command button* (knop) maken	Klik op de CommandButton control in de toolbox; doe verder hetzelfde als boven.
Een eigenschap (*property* tijdens het draaien van de applicatie (*run-time*) (laten) wijzigen (dit kan alleen met code!!)	Verander de waarde van de eigenschap in een toekenning waarin je de eigenschap een nieuwe waarde geeft, b.v. txtInvoer.Text="Verander mij!"
Een plaatje run-time inlezen	Roep de *LoadPicture*-functie aan in code met de naam van het bestand als actuele parameter en ken het resultaat aan de Picture-property van een image of picture box object toe, b.v. imgPlaatje.Picture=LoadPicture({bestand}).
Een groep van verbonden *option buttons* (radioknoppen) maken	Plaats deze in/op een object dat als container dienst kan doen, zoals een form, een picture box, een frame of een panel.
Toevoegen van *items* (regels) aan een list box	Voeg opdrachten toe die de AddItem-methode aanroepen b.v. {listbox}.AddItem {tekst} Plaats deze opdrachten in de Form_Load eventprocedure als je wilt dat de lijst na het opstarten al direct regels bevat.
Opstarten van andere applicaties vanuit een Visual Basic applicatie	Plaats een OLE control op je form; selecteer vervolgens het applicatie-object in het Insert Dialog dialoogvenster dat dan verschijnt.
Weergeven van bestaande databases in je applicatie	Plaats een data control op je form om een database te kunnen doorlopen; bind dan het data-object aan een object dat database-records kan weergeven, zoals een text box.
Records in een database aanpassen	Geef de database in je applicatie weer. Bewerk het record run-time in de text box(en) ; klik op een pijl in de data control om de wijziging op te laten slaan.
ActiveX controls installeren (toevoegen aan de toolbox)	Kies Projects|Components en klik op de Controls tab. Selecteer de toe te voegen ActiveX controls, en klik op OK.

Samenvatting Visual Basic 5.0 Step by Step, les 4 [VBSbS]

Om	Doe
Een variabel te declareren	Typ in de code achter Dim de naam van de variabele in gevolgd door As en het gegevenstype (b.v Integer, String, Double).
De waarde van een variabele wijzigen	Ken een nieuwe waarde aan de variabele toe met de toekenningsoperator (=) b.v. strLand="Nederland"
De waarde van een variabele vragen aan de gebruiker van je applicatie	Gebruik de InputBox$-functie b.v. strNaam=InputBox$("Wat is Uw naam?")
Tekst aan de gebruiker van de applicatie tonen in een dialoogvenster	Gebruik de MsgBox-procedure b.v. MsgBox "Tot de volgende keer!",,"Einde"
Declareren van een constante	Typ in het declaratie-gedeelte van het form achter Const de naam van de constante gevolg door de te gebruiken waarde achter een =-teken b.v. Const MijnNaam="Programma"
Maken van een formule (=expressie)	Koppel variabelen of (constante) waarden aan elkaar met een van de zeven binaire operatoren (Mod,+,-,/,\,*,^) en gebruik het resultaat in een toekenning, een voorwaarde of als actuele waardeparameter.
Combineren van tekst	Gebruik de concatenatie-operator (&) b.v. strNaam="Piet" & " de " & strAchternaam
Tekst naar getalrepresentatie omzetten	Gebruik de Val-functie b.v. dblPi=Val("3.1415926535897932")
Een functie gebruiken	Voeg de functie met daarachter tussen haakjes de benodigde actuele parameters (argumenten) toe aan een formule b.v. dblSchuineZijde=Sqr(dblX^2+dblY^2)
De volgorde van uitrekenen (evalueren) van een formule bepalen	Gebruik haakjes b.v. 5+3*4/2 is gelijk aan 11 (5+3)*4/2 is gelijk aan 16

Samenvatting Visual Basic 5.0 Step by Step, les 5 [VBSbS]

Om	Doe
Een logische expressie te maken	Gebruik een vergelijkingsoperator (<,<=,=,>,>=,and,or) tussen twee waarden b.v. a<b
Een selectie uit te voeren	Gebruik een If-Then-(Else-) of Select-Case-opdracht met bijbehorende expressies en keywords.
Meerdere vergelijkingen in een logische expressie combineren	Gebruik een logische operator (And, Or, Not, of Xor) tussen de vergelijkingen.
De Debug toolbar te tonen	Kies View|Toolbars|Debug.
Break mode op te roepen voor het debuggen	Klik op de Break-knop in de Debug toolbar of plaats een stop-opdracht in de code waar je Break mode wilt oproepen.
Eén regel code te laten uitvoeren	Klik op de Step Into-knop.
Een variabele te onderzoeken in het Codevenster	Selecteer de variabele die je wilt onderzoeken met de muis en klik dan op de Quick Watch knop in de Debug toolbar.
Een *watch*-expressie verwijderen	Klik op de expressie in het Watchesvenster en klik op Delete.

Samenvatting Visual Basic 5.0 Step by Step, les 6 [VBSbS]

Om	Doe
Voer een groep opdrachten een (vast) aantal keren uit	Plaats de opdrachten tussen een For- en Next-opdracht in een lus b.v. ' de tafel van 8 For i=1 To 10 Print "8*" & i & "=" & (8*i) Next i
Geef tekst weer op het form	Gebruik de Print-methode b.v. frmVenster.Print "Dit is een venster"
Een bepaalde rij van waarden gebruiken	Gebruik het Step-keyword b.v. For i=2 To 10 Step 2 MsgBox i & " is een even getal." Next i
Een For-Next-lus eerder beëindigen	Gebruik de opdracht Exit For.
Een groep opdrachten uitvoeren totdat aan een bepaalde voorwaarde is voldaan	Plaats de opdrachten tussen een Do- en Loop-opdracht b.v. Do strAntwoord=InputBox("Nog eens?") Loop Until strAntwoord="Nee"
Een oneindige lus voorkomen	Zorg dat de voorwaarde in de zolang/totdaty-lus op zeker moment False/True wordt.
Een groep opdrachten uitvoeren op vaste tijdstippen	Plaats de opdrachten in de Timer-eventprocedure van een op het form geplaatst Timer-object. De opdrachten worden uitgevoerd als de Enabled-property True is na elke Interval milliseconden.
De tekst in de titelbalk van het form instellen	Geef de Caption-property van het form een waarde.

Samenvatting Visual Basic 5.0 Step by Step, les 9 [VBSbS]

Om	Doe
Een venster verbergen	Maak de Visible-property gelijk aan False of voer de Hide-methode uit
Reageren op het optreden van runtime fouten	Plaats code hiertoe achter een label aan het einde van de Sub of Function: dit is de error handler. Het nummer van de fout die optrad is de waarde van de Err-functie. Een omschrijving is te verkrijgen met Error$(Err). De error handler treedt in werking na de opdracht On Error Goto {label}, en kan met On Error Goto 0 worden uitgezet.
Doorgaan na een fout	Maak gebruik van de Resume, Resume Next of Resume {label}-opdrachten (zie VB Help).
Een Sub/Function direct verlaten	Gebruik de Exit Sub/Function-opdracht (plaats deze voor de error handler label om te voorkomen dat deze altijd wordt uitgevoerd, ook als er geen fout optreedt).
Een nieuwe module aanmaken	Kies Project|Add Module.
Een nieuwe module opslaan	Kies File|Save {module} As (na selectie).
Een module verwijderen uit een project	Kies Project|Remove {module} (na selectie).
Een bestaande module toevoegen	Kies Project|Add File.
Een *public* (overal bruikbare) variabele maken	Gebruik i.p.v. Dim Public.
Een *public* sub/functie maken	Alle subs/functies in modules zijn *public* (in tegenstelling tot subs/functions in forms).
Een zelf-gedefiniëerde sub/function aanroepen	Plaats achter de naam van de sub/function de te gebruiken actuele parameters met komma's van elkaar gescheiden. Gebruik een sub-aanroep altijd als losse opdracht en een functie-aanroep in een expressie. De actuele parameters in een functie-aanroep moeten tussen haakjes worden geplaatst.
Zelf een Sub/Function maken	Direct door achter de End Sub/Function de kop van de Sub/Function in te typen. Of via Tools|Add Procedure.

Samenvatting Visual Basic 5.0 Step by Step, les 10 [VBSbS]

Om	Doe
Objecten in een collectie te verwerken	Schrijf een For Each...Next lus die elk element van de collectie verwerkt b.v. Dim Ctrl As Control For Each Ctrl In Controls Ctrl.Visible=False ' onzichtbaar maken Next Ctrl

Alle objecten in de Controls collectie die geen TextBox zijn naar rechts bewegen	Dim Ctrl As Control
	For Each Ctrl In Controls
	If Not (TypeOf Ctrl Is TextBox) Then
	Ctrl.Left=Ctrl.Left+200
	End If
	Next Ctrl

Eem publieke array met een vaste grootte aanmaken	Voor de opslag van 10 werknemernamen: Public Employees(9) As String
Een dynamische array aanmaken	Declareer de array maar geef geen bereik van te gebruiken elementen op tussen de haakjes: Dim Temperatures() As Double Geef de grootte op in een ReDim-opdracht: ReDim Temperatures(Days)
Een array aanmaken in een Sub/Function	Hetzij als dynamsiche array: Dim Employees() As String ' zelf dimensioneren: ReDim Employees(9) ' 10 elementen Of met: Static Employees(9) As String ' waarden blijven tussendoor behouden!
De verstekwaarde van de kleinste array-index gelijk aan 1 maken	Option Base 1
Het canvas van een form leegmaken	Cls
De elementen in een array doorlopen	Schrijf een For..Next lus die een tellervariabele gebruikt om elk element van de array te adresseren. B.v. For intIndex=1 To intAantalGetallen dblTotaal=dblTotaal+dblGetallen(intIndex) Next intIndex
De cursor op het form positioneren (voor printen op het form of tekenen)	Stel de CurrentX- en CurrentY-properties van het form in op het aantal twips van de linkerkant resp. bovenkant van het form b.v. CurrentX=200: CurrentY=100

Veel gebruikte eigenschappen in Visual Basic [VBLR]

Naam	Komt voor in	Gebruik
AutoSize	Label,Picture Box	Als True, dan passen de afmetingen zich aan aan de inhoud (Caption of Picture-property).
BackColor	Alle visuele objecten	De achtergrondkleur van de control. Bij command buttons heeft deze geen effect! Bij labels ook geen effect als de BackStyle-property op Transparant (doorschijnend) staat.
Caption	Form, Label, Command Button, Check Box, Panel, Frame, Option Button	Een tekst die op het object verschijnt. In een form in de titelbalk. Wordt in overige objecten getoond in de voorgrondkleur (zie ForeColor).
DragMode	Alle visuele objecten	Als 0, dan begint een drag-and-drop operatie direct als de muisknop op het object wordt ingedrukt. Als gelijk aan 1 pas nadat de Drag-methode wordt aangeroepen.
Enabled	Alle visuele objecten	Als True, dan kan het object worden geactiveerd. Een command button laat zien of ie *enabled* of *disabled* is, maar vele andere objecten niet!
ForeColor	Alle visuele objecten	De voorgrondkleur: de kleur waarin tekst en lijnen worden weergegeven.
Height, Width	Alle visuele objecten	De hoogte/breedte van het object in eenheden van de container waarin het object zit. Forms, panel, picture boxen kunnen als container dienstdoen d.w.z. er kunnen controls op geplaatst worden. Onzichtbaar maken van de container maakt ook alle controls in de container onzichtbaar! Screen.Width/Height is de breedte/hoogte van het scherm (in twips).
Image	Image Box, Picture Box	Het getekende in het object (inclusief plaatje).
Index	Vele controls	Het plaatsnummer in de control array waar deze control onderdeel van uitmaakt.
Interval	Timer	Het aantal milliseconden tussen twee opeenvolgende uitvoeringen van de Timer-eventprocedure.
Left, Top	Vele visuele objecten	De positie van het object in zijn container. Voor een form de positie (in twips) op het scherm.
List()	Alle list en combo boxen	De array met de tekst van alle regels in het object opgeslagen. Het aantal regels staat in de ListCount-property en het eerste element heeft de index 0! Het nummer van een eventueel geselecteerde regel staat in de ListIndex-property (is anders –1).
Name	Alle objecten	De naam van het object zoals die in de code moet worden gebruikt! Kan tijdens het draaien van de applicatie niet worden gewijzigd.
Picture	Form, Image en Picture Box	Het plaatje dat in het object wordt getoond. Wat er op het object getekend wordt (niet in de image box!) wordt erop getekend. Samen vormen ze de Image-property.

Naam	Komt voor in	Gebruik
Tag	Alle objecten	Een tekst die vrij te gebruiken is (b.v. om een identificatie in op te slaan)
Text	Text box	De weergegeven tekst. Het Change-event treedt op als de tekst verandert.
Value	Scrollbars	De waarde van de schuif.
Visible	Alle visuele objecten	Als True, is het object zichtbaar (en alle objecten die het bevat), anders is het onzichtbaar (en dus ook alle objecten erop).
WordWrap	Label	Als True, dan worden woorden die niet passen op volgende regels weergeven, anders niet.

Veel gebruikte events in Visual Basic [VBLR]

Naam	Treedt op in	Treedt op
Activate	Form	Als het venster actief wordt.
Change	Vele objecten	Als de inhoud (Caption, Text, Value) verandert.
Click	Visuele objecten	Als erop geklikt wordt terwijl het object *enabled* is.
DblClick	Visuele objecten	Als op het object dubbelgeklikt wordt. Het vervelende is vaak wel dat dan ook een Click-event optreedt.
DeActivate	Form	Als het venster de-actief wordt: een goede plek om het venster onzichtbaar te maken met de Hide-methode of door de Visible-property gelijk aan False te maken.
DragDrop	Visuele objecten	Als de gebruiker een versleept object (de source) op het object laat vallen.
DragOver	Visuele objecten	Als de gebruiker tijdens het verslepen over dit object beweegt.
KeyPress	Form, Text box	Als de gebruiker een gewone toets indrukt en loslaat.
Load	Form	Als het form wordt aangemaakt (dit gebeurt maar één keer in het leven van een form).
MouseDown, MouseUp	Visuele objecten	Als de gebruiker een muisknop indrukt/loslaat boven het object
MouseMove	Visuele objecten	Als de gebruiker de muiscursor erboven beweegt.
Resize	Form, Picture box	Als het object van grootte verandert. Treedt ook na het laden van het form op.
Timer	Timer	Als sinds de vorige keer dat dit event optreedt er, als opgeslagen in de Interval-property, milliseconden zijn verstreken (mits de Enabled-property True is).

Veel gebruikte methoden in Visual Basic [VBLR]

Naam	Komt voor in	Gebruik/Functie
AddItem	Combo/List Box	Voor het toevoegen van een item (tekstregel).
Clear	Combo/List box	Voor het leegmaken.
Cls	Form, Picture box	Voor het verwijderen van het erop getekende.
Drag	Visuele objecten	Voor het beginnen, stoppen met slepen.
Move	Visuele objecten	Voor het verplaatsen van het object.
Print	Form, Printer, Picture box, Debug	Voor het weergeven van tekst op de huidige positie als door de CurrentX- en CurrentY-positie aangegeven.
Refresh	Visuele objecten	Voor het bijwerken van de weergave van het object, terwijl code nog wordt uitgevoerd.
RemoveItem	Combo/List box	Voor het verwijderen van een item.
SetFocus	Visuele objecten	Voor het actief maken van het object.

Bronvermelding

VBSbS Michael Halvorson, Microsoft Visual Basic step by step, Microsoft Press, 1997, ISDN 1-57231-435-4.

VBLR Microsoft Corporation, Microsoft Visual Basic 3.0 Language Reference, Microsoft Corporation, 1993, DB51411-0593.