Nieuwsbrief

19de jaargang december 2012 **Nummer 4**

HCC BASIC interessegroep

Inhoud

Deze uitgave kwam tot stand met bijdragen van:

Naam	*Blz*
Jan van der Linden	20

Contacten

Functie	Naam	Telefoonnr.	E-mail
Voorzitter	Jan van der Linden	071-3413679	voorz@basic-gg.hcc.nl
Secretaris	Gordon Rahman Tobias Asserstraat 6 2037 JA Haarlem	023-5334881	secr@basic-gg.hcc.nl
Penningmeester	Piet Boere	0348-473115	penm@basic-gg.hcc.nl
Bestuurslid	Titus Krijgsman	075-6145458	t.krijgsman8@upcmail.nl
Redacteur	M.A. Kurvers Schaapsveld 46 3773 ZJ Barneveld	0342-424452	m.a.kurvers@hccnet.nl
Ledenadministratie	Fred Luchsinger	0318-571187	f.luchsinger@kader.hcc.nl
Webmaster	Jan van der Linden	071-3413679	j.vd.linden@kader.hcc.nl

http://www.basic.hcc.nl

Redactioneel

Misschien hebt u de smaak te pakken na wat ik u heb laten zien in Excel 2010. Mogelijkheden met VBA die op het werkblad onmogelijk uitgevoerd kunnen worden. Ja, en als u toch die mogelijkheden wilt gebruiken dan zit er niets anders op dan wat kennis te krijgen in de functionaliteit van VBA. Een functionaliteit die u zelfs in boeken niet snel zult vinden.

QuickBASIC 4.5 is niet alleen maar een BASIC versie; het is ook actueel met de nieuwe QB64 compiler. Deze compiler is compatible met versie 4.5 maar ook met QBasic. Dankzij de nieuwe compiler kan er gewerkt worden op verschillende besturingssystemen op 32- en 64-bits. Lees meer daarover in het hoofdstuk.

Over grafisch programmeren in GW-BASIC komen we aan het einde, de appendix. Hierin zullen wat programma's, die we gehad hebben, worden uitgebreid met meer mogelijkheden. Na de appendix is het boek dan ook echt dicht. Geen zorgen, voor ieder die graag nog wat meer wil zien kan na de appendix een soort deel 2 zien, want ik ga dan zelf verder met de programma's en zal ik zelf steeds nieuwe wiskundige technieken in het tekenen laten zien.

Marco Kurvers

BASIC leren – PowerBASIC (6).

Hoofdstuk 7 – Besturingsstructuren

Zoals u eerder zag bestaat een gemeenschappelijk structuur voor een programma uit drie secties: invoer, verwerking, uitvoer. Binnen elk van deze secties zijn een reeks opdrachten die door de computer uitgevoerd worden. Soms wilt u dat een opdracht elke keer herhaald wordt, misschien elke keer op een andere waarde, maar dat u niet elke keer deze opdracht wilt schrijven. Of dat u een reeks opdrachten alleen onder bepaalde omstandigheden wilt voordoen. Bijvoorbeeld, als u merkt dat er tijdens de invoer iets niet klopt, wilt u een foutbericht weer kunnen geven. Als u de fout kunt corrigeren, wilt u terug kunnen gaan naar een eerdere positie in het programma. Daarom zijn er *besturingsstructuren* uitgevonden.

Besturingsstructuren zijn voorgeprogrammeerde functies dat het leven voor de programmeur makkelijker maakt. U kunt besturingsstructuren beschouwen als de tijdbesparende apparaten van de programmeerwereld. Net zoals bij een magnetron kooktijd, zal met het gebruik van besturingsstructuren zowel uw tijd bij het programma schrijven, en de tijd die nodig is om het programma te laten draaien, verminderen.

Alle programmeertalen gebruiken besturingsstructuren. De simpelste besturingsstructuur is *sequence* (opeenvolgend) – een statement volgend op een ander statement volgend op een ander statement, en ga zo maar door. Gegevens worden verzameld, verwerkt en aan de gebruiker gepresenteerd. Een logica die in één richting loopt. U schrijft het programma op dezelfde manier – elke coderegel wordt in die volgorde uitgevoerd.

Wel, niet veel programma's passen in deze beschrijving. Zelfs in een eenvoudig programma zou u veel items in een soortgelijke manier willen verwerken, of het willen verwerken op een bepaalde manier onder een bepaalde voorwaarde. Meer complexe besturingsstructuren zijn daarom ook nodig. Deze structuren, het GOTO statement, lussen (`FOR/NEXT`, `WHILE/WEND` en `DO/LOOP`), voorwaardelijke vertakkingen (`IF/THEN`, `IF/THEN/ELSE` en `SELECT CASE`) en onvoorwaardelijke vertakkingen (`EXIT FAR` en `EXITFAR AT`) zullen worden behandeld in dit hoofdstuk.

Wat voor de computer betreft is er geen noodzaak voor besturingsstructuren, behalve bij de zeer simpelste, waarvan één instructie na een andere volgt. De reden dat er zoveel besturingsstructuren aanwezig zijn is om mensen te helpen code te schrijven en de code te begrijpen.

De maximum aantal keren dat u deze besturingsstructuren mag nesten is 64.

GOTO

Na de reeks is de volgende eenvoudigste vorm van besturing de GOTO. De GOTO is een dinosaurus, een fossiel, voorloper van meer complexe en elegante besturingsstructuren. GOTO geeft de computer opdracht om te springen naar een ander deel van het programma en worden daar de instructies verder uitgevoerd. Een regelnummer of een label kunnen beide worden gebruikt om aan te kunnen geven waar naartoe gesprongen moet worden.

Kijk eerst eens naar een voorbeeld dat de besturingsstructuur op volgorde gebruikt om elementen in een array toe te voegen. U wilt misschien vijf elementen van een array aan een reeks toevoegen om een totaal te krijgen. U kunt code gebruiken als volgt:

```
totaal% = 0
totaal% = totaal% + item%(0)  ' eerste element van de array toevoegen
totaal% = totaal% + item%(1)  ' tweede toevoegen
totaal% = totaal% + item%(2)  ' derde toevoegen
```

```
totaal% = totaal% + item%(3)  ' en zo verder
totaal% = totaal% + item%(4)  ' totdat u het einde bereikt
PRINT "Er zijn "; totaal%; "items totaal."
```

Dit werkt prima voor een dergelijke kleine steekproef, maar wilt u echt die lijn over en 1.000 elementen van het array toevoegen en kopiëren? Bij het gebruik van GOTO kunt u hetzelfde commando meerdere malen gebruiken. Een nieuwe variabele *aantal* deelt het programma mee wanneer het klaar is:

```
maximum% = 4   ' zet waarde maximum aantal
totaal% = 0
aantal% = 0
accumuleren:
totaal% = totaal% + item%(aantal%)      ' voeg volgende waarde aan
                                        ' totaal toe
aantal% = aantal% + 1                   ' verhoog het aantal
IF aantal% <= maximum GOTO accumuleren  ' is alles geteld?
PRINT "Er zijn "; totaal%; "items totaal."
```

Als u het aantal items van de array *Item%()* wilt wijzigen, hoeft u alleen maar het *maximum%* te wijzigen en de code blijft gewoon werken. Het feit dat het gemakkelijker te lezen is als *aantal%* groter wordt, komt doordat deze code van veel coderegels bespaard blijft.

Veel programmeurs zijn sterk gekant tegen het gebruik van GOTO. Meestal zijn ze correct, maar niet vanwege een GOTO kwaad. Door slordig gebruik van GOTO (van plaats tot plaats in de code springen, eerder dan met behulp van een modulaire aanpak met functies en procedures) kunt u een programma maken dat verwarrend te volgen is en tot programmering fouten kan leiden. Bovendien bieden moderne BASICs, zoals PowerBASIC, meer elegante constructies waardoor programma's gemakkelijker te lezen zijn. Logisch, er is geen verschil tussen een correct gebruikte GOTO en een van deze meer moderne besturingsstructuren. Echter, vanwege misbruik van GOTO is er een zeer reële zin dat GOTOs het teken van slechte programmering zijn.

Lussen

Een van de meest nuttige besturingsstructuren is de eenvoudige lus. Een lus voert een instructie of een set instructies voor een bepaald aantal keren uit. We hebben laten zien hoe een GOTO gebruikt wordt om een lus na te bootsen. Maar waarom een plaatsvervanger gebruiken terwijl PowerBASIC de beste methode levert?

FOR/NEXT

Een toegewijde vorm is de FOR/NEXT lus. De twee delen van deze structuur omsluiten de statements die u wilt herhalen. De FOR statement vertelt waar te teller naar moet kijken, wat de beginwaarde is en wat de eindwaarde zal zijn. Dit definieert hoeveel keren de lus uitgevoerd zal worden. De NEXT statement bepaald het einde van de lus.

```
numkatten% = 0
' voert de lus tien keren uit bij gebruik van de waarde
' teller van 1, dan 2, dan 3 en zo verder
' kent de huidige waarde van teller toe aan de waarde van numkatten
FOR teller% = 1 TO 10
     numkatten% = numkatten% + teller%
     PRINT numkatten%; "te veel? Castreer of steriliseer uw kat!"
NEXT teller%
```

De waarde van de FOR teller kan gewijzigd worden in de lus. Als u dat doet, wees dan voorzichtig en

weet vooral zeker dat de lus het einde bereikt. Bijvoorbeeld, de volgende code zal voor altijd uitgevoerd worden:

```
numkatten% = 0
FOR teller% = 1 TO 10
     numkatten% = numkatten% + teller%
     PRINT numkatten%
     ' het volgende statement voorkomt dat teller hoger wordt dan 5
     IF teller% > 5 THEN teller% = teller% - 1
NEXT teller%
```

U kunt ook de waarde veranderen van de teller bij gebruik van het STEP sleutelwoord. Normaal telt de FOR teller elke keer met één op. Bij gebruik van STEP kunt u met elke waarde verhogen die u wilt, zelfs met een waarde waarmee u de lus achteruit kunt uitvoeren:

```
numkatten% = 0
FOR teller% = 10 TO 1 STEP -2
     numkatten% = numkatten% + teller%
     PRINT numkatten%
NEXT teller%
```

Onthoud dat in het laatste voorbeeld de lus maar vijf keer uitgevoerd wordt. Bij de laatste keer, dat het uitgevoerd wordt, zal *teller%* 2 zijn. Wanneer de STEP toename van twee afgetrokken wordt is *teller%* kleiner dan de ondergrens van een, zodat de lus stopt. Een beetje extra zorg is noodzakelijk wanneer u STEP gebruikt om ervoor te zorgen dat de lus op uw manier wordt uitgevoerd.

De start-, eind- en STEP-waarden hoeven geen gehele waarden te zijn. Bijvoorbeeld, de volgende FOR/NEXT lus is volledig toegestaan:

```
FOR teller% = 0.4 TO 1.8 STEP 0.2
     PRINT teller%;
NEXT teller%
```

Wees voorzichtig dat uw lus geen afrondingsfouten vertoont wanneer u zwevende puntwaarden gebruikt. Bijvoorbeeld,

```
FOR teller! = 0 TO 2 STEP 0.1
     PRINT teller!;
NEXT teller!
```

herhaalt alleen 20 keer (de laatste gedrukte waarde is 1.9) als gevolg van het feit dat 0.1 niet precies een enkele precisie getal vertegenwoordigt. De enkele precisie weergave van 0.1 plus de geaccumuleerde waarde in *teller!* is groter dan 2.0 in de laatste iteratie, ook al lijkt het correct op papier. Dezelfde lus met gebruik van binair gecodeerde decimale zwevende puntwaarden werkt zoals verwacht, herhaalt 21 keer (de laatste gedrukte waarde is 2):

```
FOR teller@@ = 0 TO 2 STEP 0.1@@
     PRINT teller@@;
NEXT teller@@
```

FOR/NEXT lussen mogen worden genest met andere FOR/NEXT lussen of andere structuren. Bijvoorbeeld, om alle elementen van een 10x5x4 array af te drukken:

```
FOR x% = 0 TO 9
     FOR y% = 0 TO 4
```

```
            FOR z% = 0 TO 3
                 PRINT A(x%, y%, z%)
            NEXT z%
       NEXT y%
NEXT x%
```

De drie NEXT statements kunnen ook worden gecombineerd in één:

```
NEXT z%, y%, x%
```

zolang de juiste volgorde behouden blijft.

Als u een luie typist bent, voegt PowerBASIC de meest onlangs gebruikte lus variabele voor u toe:

```
FOR teller = 1 TO 10
     FOR index = 1 TO 20
          …
     NEXT
NEXT
```

impliceert `NEXT index`, gevolgd door `NEXT teller`. Als u echt vaag bent, kunt u ook deze twee combineren in één

```
NEXT,
```

wat impliceert

```
NEXT index, teller
```

Natuurlijk, deze vermindert de leesbaarheid en vergroot de kans op fouten aan de kant van de programmeur.

WHILE/WEND

Een probleem met FOR/NEXT lussen is dat u wilt weten wat de eindwaarde van *teller* zal worden voordat de lus start. Maar als u niet zeker weet wanneer uw variabele de maximum waarde zal bereiken die u wilt toestaan, kunt u bij beëindiging gebruik maken van GOTO om vanuit de lus te ontsnappen:

```
FOR teller = 1 TO 1000        ' doe dit als meeste 1000 keer
     ' voeg een willekeurig bedrag aan het totaal toe
     totaal = totaal + RND * teller
     ' laat niet het totaal boven de 1000 komen
     IF totaal > 1000 GOTO 100
NEXT teller
100  PRINT teller             ' teller zal kleiner zijn dan 1000
```

Hoewel dit geen halsmisdaad is, kan het wel voorkomen worden met behulp van de WHILE/WEND lus, waarmee gecontroleerd wordt op een voorwaarde. De actie blijft presteren voor zolang als de voorwaarde houdt. Bijvoorbeeld,

```
teller = 0
WHILE totaal < 1000
     teller = teller + 1
     totaal = totaal + RND * teller
```

```
WEND
```

De FOR/NEXT en WHILE/WEND constructies kunnen gemakkelijk bij eventuele lus vereisten, die u hebt, uitgevoerd worden. Maar voor de juiste keuze die u zou willen is er meer variatie, PowerBASIC biedt de DO/LOOP.

DO/LOOP

De DO/LOOP laat u een aantal voorwaarden voor einde instellen op het begin van de lus, het einde van de lus, of beide. De simpelste DO/LOOP ziet er uit als dit (normaal wilt u een voorwaarde instellen om te stoppen; deze lus zal oneindig uitgevoerd worden):

```
DO
     katten! = katten! + 1
LOOP
```

De DO/LOOP is altijd een keer uitgevoerd. U kunt een WHILE binnen een DO/LOOP gebruiken, maar dat converteert de DO/LOOP naar een WHILE/WEND structuur:

```
DO WHILE a < b
     {statements}
LOOP
```

die op dezelfde manier werkt als dit:

```
WHILE a < b
     {statements}
WEND
```

zo wilt u waarschijnlijk de WHILE/WEND in de eerste plaats gebruiken. In dit geval vervangt de compiler zelfs het sleutelwoord WEND voor LOOP.

U kunt ook de WHILE statement aan het einde van de lus plaatsen, maar bedenk wel dat de statements in de lus eenmaal worden uitgevoerd voordat de voorwaarden gecontroleerd worden:

```
DO
     {statements}
LOOP WHILE a < b
```

De DO/LOOP ondersteunt ook UNTIL:

```
DO
     {statements}
LOOP UNTIL a < b
```

Hier eet u uw dessert eerst voordat er gecontroleerd wordt of de klusjes wel zijn gedaan. Dit is nuttig als u niet van klussen houdt, of als u ervoor wilt zorgen dat u in ieder geval één scheur in de statements van de verwerking heeft, ongeacht welke voorwaarde op het moment van het programma. Hier is een plaats waar een klein verschil doorgaans gevaarlijk kan zijn. De plaatsing van het UNTIL statement is zeer belangrijk. Wees voorzichtig als u vertrouwd bent met zulke constructies in andere programmeertalen. Als u de UNTIL bovenaan gebruikt, voert het op dezelfde manier uit op dezelfde plaats, maar met een tegenovergestelde voorwaarde:

```
DO UNTIL a < b
     {statements}
```

```
LOOP
```

is zeker hetzelfde als

```
DO WHILE a >= b
     {statements}
LOOP
```

wat, zoals eerder is uitgelegd, hetzelfde is als een WHILE/WEND structuur. U kunt deze structuren op deze manier gebruiken als u vindt dat deze gemakkelijker zijn om te lezen. De volgende twee code-voorbeelden zijn gelijk; sommigen vinden de ene gemakkelijker te lezen, sommigen liever de andere.

```
DO WHILE (a > b) OR (c < d)
     {statements}
LOOP
```

is gelijk aan

```
DO UNTIL (a <= b) AND (c >= d)
     {statements}
LOOP
```

Belangrijke verschillen

Het is belangrijk te onthouden dat de code binnen een DO/LOOP met aan het einde haar controle *altijd* een keer zal worden uitgevoerd, overwegend dat de code binnen een WHILE/WEND, FOR/NEXT, of een DO/LOOP met aan het begin haar controle *nooit* kan worden uitgevoerd. Als de voorwaarde aanwezig is en de WHILE wordt als eerst gepasseerd, zal de WHILE lus niet worden doorlopen:

```
b = 2
a = 1
WHILE a > b
     a = a - 1
WEND
PRINT a
```

Het gedrukte antwoord zal 1 zijn. De eis dat de WHILE lus hier oplegt (dat de waarde van *a* kleiner dan of gelijk is aan *b*) is al waar, het neemt dus niet de moeite om iets te doen. De waarde van *a* is daarom onveranderd wanneer u het afdrukt aan het einde van het voorbeeld. Hoewel, als u een UNTIL in een DO/LOOP gebruikt:

```
b = 2
a = 1
DO
     a = a - 1
LOOP UNTIL a <= b
PRINT a
```

zult u zien dat er een 0 wordt geprint, omdat nu de conditie aan het eind plaatsvindt. Geen van deze voorbeelden is de ‘juiste’. Elk zal correct zijn. Het hangt er alleen van af wat u bedoelt en hoe het programma het moet doen. Houd echter wel de verschillen beperkt. Nochtans, houd in gedachten dat als er verschillen in het programma zijn, u voor veel verrassingen kan komen te staan.

Eerder eruit stappen

Er zijn momenten wanneer u het nodig vindt te reageren op de condities bij een gevarieerde uitvoerende lus. Bijvoorbeeld als u aan het zoeken bent in een groot bestand, om te kijken naar een bepaald

stukje informatie, wilt u proberen een lus op te zetten die het bestuurt. Zodra u gevonden hebt waar u naar zocht, heeft het weinig zin om verder te gaan zoeken. U kunt de IF/THEN structuur gebruiken (dat uitgelegd wordt in de volgende paragraaf) om te zien of een vergelijking is gevonden.

Zodra de test slaagt, kunt u de EXIT statement gebruiken om de lus uitvoering te stoppen en verder de gegevens te verwerken. De code zou er ongeveer als volgt uitzien:

```
FOR HuidigRec& = 1 TO NumRecords&
     ' Lees een record
     ' Vergelijk met gezochte informatie
     IF Gevonden <> 0 THEN
          EXIT FOR
     END IF
NEXT HuidigRec&
' Andere verwerking
```

U kunt dit statement gebruiken met elke lus structuur van dit hoofdstuk. Het woord gevolgd na EXIT is de naam van de structuur: EXIT DO, EXIT WHILE, EXIT FOR. U kunt het tweede sleutelwoord weglaten welke PowerBASIC vertelt dat het om de meest gangbare uitgevoerde lus gaat. Bijvoorbeeld, het vorige codefragment kan herschreven worden als dit::

```
FOR HuidigRec& = 1 TO NumRecords&
     ' Lees een record
     ' Vergelijk met gezochte informatie
     IF Gevonden <> 0 THEN
          EXIT
     END IF
NEXT HuidigRec&
' Andere verwerking
```

U kunt ook de structuur laten beginnen met de volgende iteratie zonder eerst alle statements binnen de lusstructuur uit te voeren, door gebruik te maken van het ITERATE statement. ITERATE kijkt en werkt beter dan EXIT:

```
FOR HuidigRec& = 1 TO NumRecords&
     ' Lees een record
     IF Verwijderd THEN
          ITERATE FOR
     END IF
     ' Verwerk record
NEXT HuidigRec&
```

U kunt ook het tweede sleutelwoord weglaten, zoals u dat kan doen met EXIT:

```
FOR HuidigRec& = 1 TO NumRecords&
     ' Lees een record
     IF Verwijderd THEN
          ITERATE
     END IF
     ' Verwerk record
NEXT HuidigRec&
```

IF/THEN/ELSE

Alle talen hebben het vermogen om conditioneel te vertakken. Meestal wordt dit behandeld door de IF/THEN structuur.

```
IF conditie THEN statement
```

Dit is de simpelste vorm, waar we zeggen dat *als* een gegeven conditie is waar, *dan* het statement dat er volgt zal worden uitgevoerd. Dit gevolgde statement kan uit één of meerdere statements van elk soort zijn, of het kan een GOTO zijn die naar andere statements springt en daar uitgevoerd worden. Dit is één van de manieren die laten zien waarom GOTO's zo gevaarlijk zijn. U kunt GOTO (GA NAAR), maar u weet echter niet *waar vandaan*. Dus is het moeilijk om op te sporen hoe een programma op een bepaald punt geraakt. Een oplossing voor dit probleem is dat een alternatief statement(s) moet worden uitgevoerd als de voorwaarde in de IF onwaar is. In PowerBASIC kan dit worden gedaan met de ELSE:

```
IF conditie THEN statement ELSE statement
```

Nogmaals, de statement(s) die na ELSE volgen kunnen van elk type zijn. Bijvoorbeeld:

```
IF a = b THEN PRINT a: PRINT b ELSE a = b: PRINT a
```

PowerBASIC ondersteunt ook meerdere regel- (of blok)versies afgesloten met END IF:

```
IF {conditie} THEN
     {statement 1}
     {statement 2}
ELSE
     {statement 3}
     {statement 4}
END IF
```

Als een taal geen IF/THEN blokken ondersteunt, kunt u die met GOTO's creëren. Het vorige voorbeeld is gelijk aan:

```
IF {conditie} THEN GOTO {blok1} ELSE GOTO {blok2}
```

IF/THEN/ELSE blokken kunnen ook het sleutelwoord ELSEIF hebben. Deze combineren ELSE en IF binnen het blok om meer extra geconditioneerde controles uit te kunnen voeren:

```
IF schulden = inkomen THEN
     PRINT "Zelfs het breken."
ELSEIF schulden > inkomen THEN
     PRINT "Faillissement verklaren."
ELSEIF schulden = 0 THEN
     PRINT "Vieren!"
ELSE      ' uitgevoerd als schulden < inkomen en schulden <> 0
     PRINT "Hee! Dat is mooi!"
END IF
```

De meeste GOTO's in een programma kunnen weggehaald worden door de IF/THEN blokken te restructureren. U moet heel voorzichtig zijn, omdat die voorwaarden goed vertaald moeten zijn.

SELECT CASE

In veel programma's moet u wellicht een aantal dingen doen, alles op basis van de waarde van een variabele die als een vlag fungeert. Een manier hoe u dit zou kunnen doen is gebruik te maken van een reeks GOTO statements:

```
IF x = waarde1 THEN statement1 : GOTO 300
IF x = waarde2 THEN statement2 : GOTO 300
IF x = waarde3 THEN statement3 : GOTO 300
IF x = waarde4 THEN statement4 : GOTO 300
300 ...
```

In dit voorbeeld elimineren GOTO's extra controles nadat de juiste waarde is gevonden. U kunt ook een IF/THEN/ELSEIF statementblok gebruiken:

```
IF x = waarde1 THEN
     statement1
ELSEIF x = waarde2 THEN
     statement2
ELSEIF x = waarde3 THEN
     statement3
ELSEIF x = waarde4 THEN
     statement4
ENDIF
```

PowerBASIC biedt een meer moderne en elegante aanpak voor deze structuur: de SELECT CASE statement, die logisch gezien gelijk aan een aantal GOTO's is, maar gemakkelijker te lezen is:

```
SELECT CASE dier$
     CASE "tijger"
          PRINT "Ren!"
     CASE "olifant"
          PRINT "Vraag om een ritje."
     CASE "kat"
          PRINT "Neem het thuis en voed het."
     CASE "", "?"
          PRINT "Buitenaards. Akte van de vreedzame."
END SELECT
```

Een bijkomend voordeel is dat meerdere regel-statements niet minder leesbaar in dit formaat zijn. U kunt in het SELECT CASE statement een reeks van "gevallen" die beschrijven wat te doen, wanneer aan bepaalde voorwaarden is voldaan. Soms kunt u zelfs een SELECT CASE statement gebruiken als er maar één geval is! Dit kan plaatsvinden wanneer u meerdere mogelijke waarden uitvoert. In plaats van

```
IF dier$ = "leeuwen" OR dier$ = "tijgers" OR dier$ = "beren" THEN
     PRINT "Oh, jee!"
END IF
```

kunt u schrijven

```
SELECT CASE dier$
     CASE "leeuwen", "tijgers", "beren"
          PRINT "Oh, jee!"
END SELECT
```

Deze vorm is flexibeler en gemakkelijker te wijzigen, mocht u later meer controles toe willen voegen.

Onvoorwaardelijke takken

De andere controlestructuren in dit hoofdstuk zijn beperkt op één belangrijk aspect: ze kunnen geen procedures of functies omvatten. De volgende sectie “De laatste GOTO elimineren” wordt voor speciale gevallen met behulp van vlag variabelen met lussen en voorwaardelijke takken, in plaats van GOTO’s, op die manier besproken. Er is echter nog een ander alternatief: EXIT FAR.

EXIT FAR is de beste manier om een “noodstop” op te stellen; een uitweg ongeacht hoeveel procedures of functies zijn aangeroepen. Gebruik de volgende stappen om dit te doen:

- Schrijf uw eigen noodstop routine
- Voer EXIT FAR AT uit en specificeer de label aan de routine
- Ga verder met uw programma zoals gewoonlijk
- Zodra een foutmelding verschijnt voer dan EXIT FAR uit.

Hier is een voorbeeld:

```
exit far at NoodStop

call Proc1
end

NoodStop:
     print "Fatale fout is opgetreden!"
     end

sub Proc1
     print "In Proc1"
     call Proc2
end sub

sub Proc2
     print "In Proc2"
     call Proc3
end sub

sub Proc3
     print "In Proc3"
     call Proc4
end sub

sub Proc4
     print "In Proc4"
     call Proc5
end sub

sub Proc5
     print "In Proc5"
     call Proc6
end sub
```

```
sub Proc6
     print "In Proc6"
     input "Fataal afsluiten"; jaNee$
     if ucase$(left$(jaNee$, 1)) = "J" then exit far
end sub
```

U kunt EXIT FAR AT statements nesten in meerdere lagen; EXIT FAR zal procedures beëindigen tot-dat het meeste recente uitgevoerde EXIT FAR AT statement gevonden is. PowerBASIC zal dan de procedures beëindigen totdat het gespecificeerde label in dat EXIT FAR AT statement gevonden is.

EXIT FAR maakt de call stack schoon als het de procedures beëindigd. U kunt dus veilig meer “nood-stop” routines gebruiken dan nodig is.

De laatste GOTO elimineren

Een van de “laatste stand” argumenten door voorstanders van GOTO is dat zij de code op speciale manieren “vereenvoudigen”. Het probleem dat zij beweren om het op te lossen is, wat te doen als in het midden van een reeks lange statements ontdekt wordt dat je echt niet wilt om verder te gaan met de statements:

```
     { eerste gedeelte van de verwerking }
     IF teller > totaal GOTO 1000
     { tweede gedeelte van de verwerking }
1000 ...
```

U kunt GOTO-minder programma’s schrijven die hetzelfde effect bereiken; er volgt een voorbeeld van hoe dit te doen. Zodra men aan de GOTO-minder code went, vindt men het moeilijk om te denken van de gevallen waar GOTO’s nodig zijn. Het resulteert soms in een paar extra vlaggen voor WHILE lus-sen die het hele programma omvatten. Overweeg een spel waar je voortdurend bewegingen verwerkt totdat het spel is voltooid. Het spel kan beëindigen voor verschillende redenen: de speler kan drukken op *Ctrl-Break*, er kunnen geen stappen meer mogelijk gemaakt worden of de voorwaarden op de over-winning kunnen worden bereikt. Een gestructureerde, maar onbegeerlijke GOTO’er, maakt misschien de hoofdmodule code als dit:

```
100  { speler beweging krijgen }
     IF { Ctrl-Break gedrukt } GOTO 500
     IF { beweging is niet geldig } GOTO 100      ' doe het nogmaals
     { verwerking verplaatst }
     IF { geen bewegingen meer } GOTO 300
     IF { overwinning opgedaan } GOTO 400
     GOTO 100                                     ' blijf spelen
300  { verwerking einde van spel zodra geen bewegingen over zijn }
     GOTO 600
400  { verwerking einde van spel zodra overwinning is bereikt }
     GOTO 600
500  { Ctrl-Break bericht weergegeven }
600  { haal speler uit het spel }
```

Dit is oké en werkt, maar de volgende code elimineert de vraag naar GOTO’s en maakt de logische stroom duidelijker (let op het gebruik van equates om de leesbaarheid te verhogen):

```
%WAAR = -1
%ONWAAR = 0
```

```
%DOORGAAN = 0
%BREAKGEDRUKT = 1

%GEENBEWEGINGENMEER = 2
%OVERWINNING = 3

' hier ligt het vlees van het spel...
WHILE vlag = %DOORGAAN
     goedebeweging = %ONWAAR
     WHILE NOT goedebeweging AND vlag <> %BREAKGEDRUKT
          CALL spelerbewegingkrijgen(goedebeweging, beweging)
          IF { Ctrl-Break gedrukt } THEN vlag = %BREAKGEDRUKT
     WEND
     { verwerk beweging met de volgende code }
     IF vlag <> %BREAKGEDRUKT THEN
          IF { geen bewegingen meer } THEN vlag = %GEENBEWEGINGENMEER
          IF { overwinning opgedaan } THEN vlag = %OVERWINNING
     END IF
WEND
' ...vallen naar hier wanneer het spel op enige wijze eindigt.
' Nu moet u sommige speciale verwerkingen doen,
' gebaseerd op hoe het spel eindigde.
SELECT CASE vlag
     CASE %BREAKGEDRUKT
          { Ctrl-Break bericht weergegeven }
     CASE %GEENBEWEGINGENMEER
          { verwerking einde spel zodra geen bewegingen over zijn }
     CASE %OVERWINNING
          { verwerking einde spel zodra overwinning is bereikt }
END SELECT
{ haal speler uit het spel }
```

Deze code elimineert alle labels en is nu ordelijker. In het eerste programma is het moeilijk om de modulaire structuur te zien. De GOTO's verwarren de grens tussen de spelverwerking en het hanteren van het eindspel. Met de vlaggen en WHILE statements valt het programma meer in twee belangrijke secties. Ook met behulp van equates voor de vlaggen is het programma zelf-documenterende. Als u er nog steeds niet van overtuigd bent, komt u waarschijnlijk met een andere manier hoe het spel beëindigd moet worden, misschien met een tijdslimiet. Met welke versie van het programma zou u liever willen werken?

Samenvatting

Lussen en conditionele vertakkingen gebruiken maakt uw programmeerleven veel gemakkelijker. Het is niet alleen gemakkelijker om met deze structuren programma's te schrijven, het is ook gemakkelijker voor u (of iemand anders) deze programma's later te handhaven. Deze structuren zijn bedrieglijk simpel; u kunt een enorme hoeveelheid programmeringscontrole eruit knijpen. Zo, nu u deze krachtige hulpprogramma's onder de riem hebt, kunt u leren hoe u ze samenkoppelt binnen logische modules, of structuren, zoals subroutines, functies, procedures en de nieuwe en bestaande units.

In de volgende nieuwsbrief zal ook daarom een nieuw hoofdstuk verschijnen, hoofdstuk 8. Dat hoofdstuk zal over 'Modulair programmeren' gaan.

Marco Kurvers

Op twee gronden van Excel.

In dit deel van Excel gaan we het over de twee gronden hebben: de voorgrond en achtergrond. Hoe gebruiken we ze en hoe kunnen we deze gronden samen bundelen?

De voorgrond en achtergrond

Werken met de Office programma's zonder macro's is de voorgrond. In Excel is dat alleen maar het lint en de werkmap. De werkmap is een soort document, zoals we in Word kennen, maar kan uit meerdere werkbladen bestaan, standaard zijn dat er altijd drie.

Zodra we de macro's inschakelen, zoals opnemen of een project gaan maken, gebruiken we de achtergrond. Gebruikt u alleen maar de macro's, maar opent u niet de VBA editor? Dan is natuurlijk de vraag voor u of we het dan over de voorgrond of de achtergrond hebben. Hoewel alleen maar met het opnemen en uitvoeren van macro's de voorgrond nodig is, moeten we toch spreken van de achtergrond. De reden is dat er in uw document VBA code gebruikt wordt. U dient dan ook uw bestand op te slaan met macro-ondersteuning. Excel documenten zonder deze macro-ondersteuning hebben dan ook een heel ander bestandstype.

Bestandsformaten

Excel 2007 en 2010 kennen zes belangrijke bestandstypen. Onderstaand tabel laat twee van de zes bestandstypen zien met de beschrijvingen.

Type	Extensie	Beschrijving
Excel werkmap	.xlsx	In dit bestandstype wordt alle informatie in een Excel werkmap in het XML-formaat opgeslagen. Binaire bestanden zoals illustraties worden ook in de structuur opgenomen en in XML gelabeld. Vervolgens wordt het geheel in het ZIP-formaat gecomprimeerd. Als u dit type bestand van een onbekende uitgever toegezonden krijgt, kunt u het rustig openen omdat er geen macro's in aanwezig kunnen zijn. Dit bestandstype behoort daarom tot de voorgrond.
Excel werkmap met macro's	.xlsm	In dit bestandstype kunnen macro's worden opgeslagen. Wilt u dus opgenomen macro's of code uit de VBA editor opslaan, dan zult u dit type moeten kiezen in het dialoogvenster **Opslaan als**. Tevens dient u goed na te gaan als u van een onbekende uitgever een .xlsm-bestand ontvangt, of het veilig is! Dit bestandstype hoeft niet alleen tot de achtergrond te behoren, het kan ook beide zijn. In feite is dat ook zo. Later daarover meer.

De overige vier bestandstypen kunnen macro's bevatten maar hebben ieder hun eigen doel. Zo is een type **.xlsb** alleen nodig als u gigantische grote werkbladen nodig heeft. Dit bestandsformaat maakt gebruik van alle rijen (1.048.576) en kolommen (16.384), die samen een werkblad bestaande uit een astronomisch aantal van 17.179.869.184 cellen vormen. Omdat het een binair bestandsformaat is, kan het ertoe leiden dat door een paar foute bits het hele bestand onbruikbaar kan worden. Het voordeel is wel dat een dergelijk bestand veel sneller geopend wordt.

Als u macro's gebruikt en u slaat uw bestand op met de extensie **.xls**, dan kan het vervelend zijn dat niet alle macro's zullen werken zoals verwacht. Immers, het lint bestond bijvoorbeeld nog niet in eerdere versies van Excel.

Macro's en VBA projecten

Als u macro's gebruikt op de voorgrond en u maakt geen gebruik van de VBA editor, dan weet u nu dat de achtergrond automatisch erbij te pas komt. Immers, macro's bevatten code en dat betekent dat het bestand met de extensie **.xlsm** opgeslagen moet worden.

De vraag is nu, waar worden de macro's in bewaard? In een apart macrobestand? In een macrolijst? De macro's die u opneemt komen in geen van beide mogelijkheden terecht. Ze komen terecht in een **VBA module**. U kunt de macro-module vergelijken met een Visual Basic 6.0 module. De macro's zien er ook uit als normale subroutines.

Zodra de eerste macro bewaard wordt, maakt VBA een module aan en wordt er een subroutine met de macronaam aangemaakt. De module komt in het VBA project van de werkmap waar u mee bezig bent. Als u dus met de werkbladen bezig bent, is er al een VBA project. U bent er echter niet mee bezig, en mocht u het document bewaren onder het bestandstype **.xlsx** dan gaat het VBA project verloren. Let op, dat is dus alleen als u geen gebruik maakt van macro's. Bevat het VBA project één of meerdere macro-modules, dan is het niet meer mogelijk de werkmap met bestandstype zonder macro's op te slaan.

Macro's starten

Macro's kunnen op verschillende manieren worden gestart.

- In het dialoogvenster **Macro**. De macro die u kiest wordt in de lijst geselecteerd, daarna klikt u op **Uitvoeren**.
- Door een toetsencombinatie. Deze toetsencombinatie kan gekozen worden tijdens het opnemen van de macro. De toetsencombinatie bestaat altijd uit een toets samen met de **Ctrl**-toets. Denk erom dat u goed nadenkt welke toetsencombinatie u wilt gebruiken, omdat binnen de Excel-interface ook vele toetsencombinaties gedefinieerd zijn. Bijvoorbeeld **Ctrl** + b is een toetsencombinatie om de celinhoud vet te zetten. Als u deze combinatie toekent aan een macro dan gaat die vóór de oorspronkelijke toekenning!
- Door gebruik van objecten. U kunt een macro aan een afbeelding of een knop toekennen. De macro wordt dan gestart als u op de afbeelding of knop klikt.
- Het is mogelijk zelf lint-items te definiëren en er macro's aan toe te kennen.
- Macro's kunnen ook aan gebeurtenissen gekoppeld worden. Deze treedt automatisch op zodra er een bepaalde toestand gewijzigd wordt. Er treedt bijvoorbeeld een event op als een werkmap geopend wordt. Een macro die aan deze gebeurtenis wordt gekoppeld, wordt elke keer dat het optreedt automatisch uitgevoerd.
- Via de werkbalk **Snelle toegang**. Het is mogelijk deze werkbalk, die linksboven te vinden is, uit te breiden met uw macro's. Zo is het mogelijk een compleet VBA project via de snelle toegangsbalk te starten in plaats van via een lint-item of een gebeurtenis in het werkblad.

Meer over het VBA project

Goed, dat zijn dus de macro's. U kunt er veel mee, dat is één ding dat zeker is. Toch ontbreken er mogelijkheden op de voorgrond die op de achtergrond wel aanwezig zijn. We weten dat een selectie maken gemakkelijk in een macro opgenomen kan worden, maar zoals we gewoon op de voorgrond geen selecties kunnen nesten (*zie vorige nieuwsbrief*), zo is dat ook niet mogelijk als we macro's gaan gebruiken. De enige manier om het nesten van selecties toch in een macro te kunnen bewaren, is gebruik te maken van meer functionaliteit: VBA.

Via het tabblad **Ontwikkelaars** kunt u de VBA editor openen. De editor bestaat uit:

- Een verkenner waar minstens één project of meer projecten in staan. Elk project is te vergelijken met een werkmap op de voorgrond.
- Het codevenster. Hier staat alle VBA code van het gekozen onderdeel uit de projectverkenner.
- Het VBA menu met de werkbalk(en).
- Het **Eigenschappen**-venster. Hiermee kunnen we eigenschappen bewerken van de gekozen objecten, zoals een werkblad.

- Eventueel het **Direct**-venster. Als we wat willen weten over de waarde van een functie of variabele, kunnen we – tijdens het debuggen van het project – de waarde controleren, of we voeren zomaar een expressie of functie uit.

Figuur 1 laat het lint zien met het tabblad *Ontwikkelaars*. Met de knop *Visual Basic* opent u de editor.

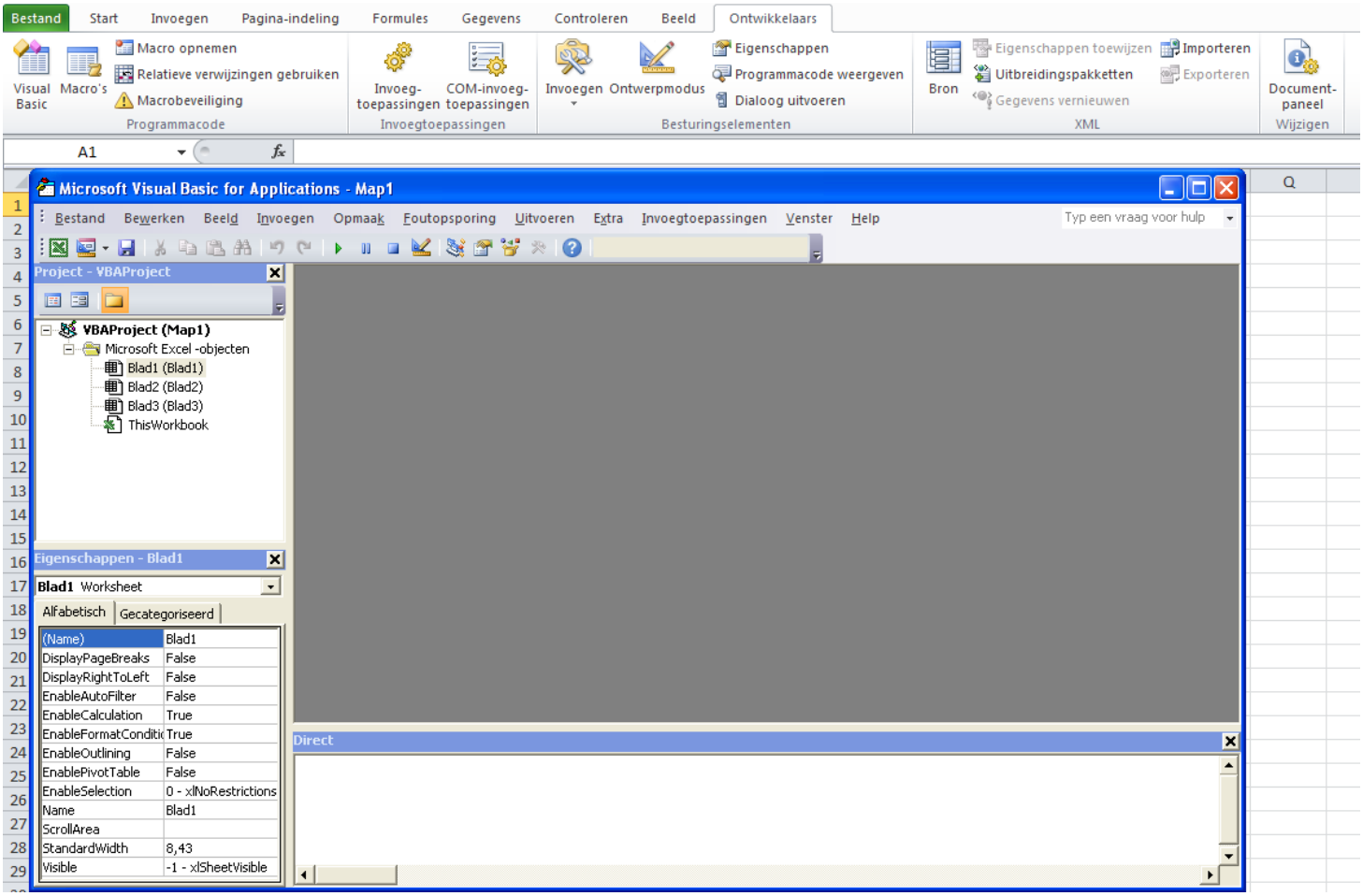
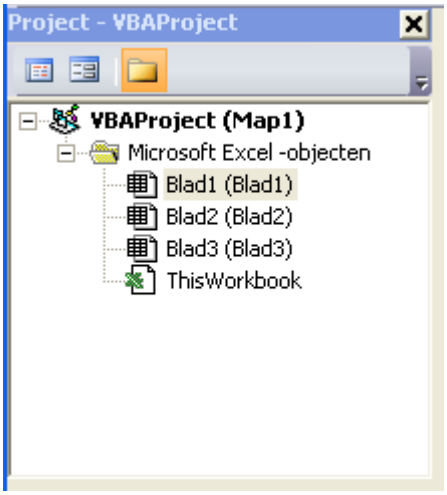


Fig. 1 Via het tabblad Ontwikkelaars kunnen we met de knop Visual Basic de editor openen. Met de Macro's knop openen we alleen de macrolijst waarmee we de macro's kunnen bewerken en uit kunnen voeren.

Aan de linkerkant is de projectverkenner aanwezig. Figuur 2 laat deze groter zien.



Een VBA project ziet er standaard uit als een lege werkmap. De projectnaam **Map1** is de titel van de werkmap.

De werkbladen ziet u hier als objecten. Werkblad **Blad1** is dus de naam van het object. De naam die tussen haakjes staat is de titel van het werkblad. Wijzigt u op de voorgrond de titel Blad1 in een andere titel, dan zal alleen wat tussen haakjes staat veranderen. Welke titel u ook opgeeft, in VBA zult u altijd met de standaard objectnamen Blad1, Blad2, Blad3, …, Blad*x* moeten werken.

Het ThisWorkbook is de eigenlijke map van het project. Er kan dan ook maar één zo'n workbook in zitten. U kunt het niet wijzigen of verwijderen. Hiermee kunt u het hele project besturen.

Fig. 2 Een lege werkmap ziet er op de achtergrond in de VBA editor er standaard zo uit.

Figuur 3 laat het venster zien met de eigenschappen van het gekozen object of module uit de projectverkenner.

Als u wat kiest uit de projectverkenner, zal het eigenschappenvenster worden vernieuwd met de eigenschappen en waarden van het gekozen object.

U ziet hier dat het object **Blad1** van het type **Worksheet** is. Zoals u in de vorige nieuwsbrief heeft gezien, kunt u met dit object werken en de gegevens in de cellen bewerken.

Als u de eigenschappen langsgaat, stuit u plots op de eigenschap **Name** met de gegeven objectnaam **Blad1**. Eerder heb ik u verteld dat het onmogelijk is om objectnamen van de werkbladen te wijzigen, maar waarom staat hier dan de eigenschap? Wat gebeurt er als we de naam gaan veranderen?

Laten we het eens proberen. Verander **Blad1** in **Blad9** en druk op Enter.

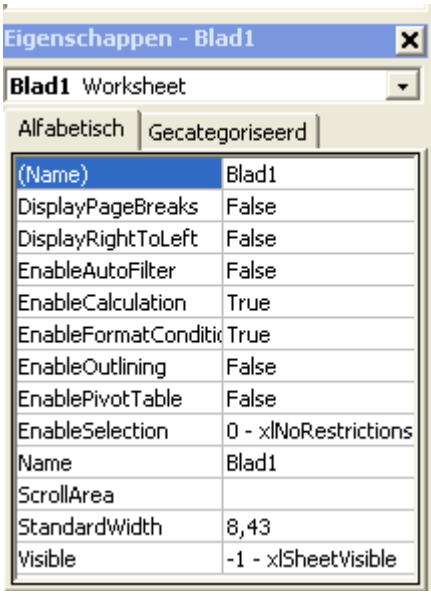


Fig. 3 Eigenschappenlijst

Gelijk zien we in de projectverkenner dat bij het eerste blad de naam **Blad9** tussen de haakjes staat en dat *niet* daadwerkelijk de naam zelf veranderd is. Er staat nog steeds *Blad1*!

Probeer deze dan eens: verander **Blad1** in **Blad2**. Visual Basic komt nu met een foutmelding:

Mocht de melding te klein zijn, hieronder staat het nog een keer:

Kan de naam van een blad niet wijzigen in de naam van een ander blad, een opgeroepen objectbibliotheek of een werkmap die wordt gebruikt door Visual Basic.

Wat we ook proberen, VBA weet donders goed ons op de vingers te tikken, mochten we zulke dingen willen proberen, en dat geldt niet alleen in de ontwikkelfase (de IDE), maar ook als we gebruik gaan maken van de editor.

Onderaan in de projectverkenner ziet u **ThisWorkbook**. Als u er op dubbelklikt, wordt de editor geopend. Het lijkt erop alsof we een module geopend hebben, maar ThisWorkbook is eigenlijk geen module, althans geen module waar we normaal onze eigen macro's in zetten. Het ThisWorkbook is een *klassemodule*. Dit is het hart van de Excel-map waar we mee werken. Telkens, als we een map openen, wordt er een event gestart. Dat event is een zeer belangrijke event. De subroutine hiervan is:

```
Private Sub Workbook_Open()
     ' wat wilt u hier doen?
End Sub
```

Het is mogelijk om de commentaar te beantwoorden met code zoals:

- het starten van een VBA project;
- het starten van een dialoogformulier;
- het instellen en uitvoeren van werkblad configuraties, zoals ingeplande takenlijsten.

Er is nog een event waar hetzelfde mee gedaan kan worden, namelijk als we geen map willen openen maar een nieuw blad in de map willen toevoegen. De subroutine is:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
     ' wat wilt u, eventueel met het Sh object, doen?
End Sub
```

Het voordeel van deze subroutine is dat de bovengenoemde mogelijkheden per blad gekozen kunnen worden en niet voor de hele map. U kunt bijvoorbeeld ervoor kiezen in het tweede blad een zoekformulier te starten, terwijl het eerste blad alleen de gegevenslijst bevat waarmee in gewerkt kan worden. Dit is een zeer geschikte situatie die voor werkzaamheden, zoals voorraadbeheer en boekhouding, gunstig zijn.

In de volgende nieuwsbrief laat ik u meer zien over de ThisWorkbook klassemodule en laat ik u ook zien hoe we eigen dialoogformulieren kunnen ontwikkelen en in Excel kunnen starten.

Marco Kurvers

QB64, de nieuwe QuickBASIC compiler.

Met de programmeertaal BASIC wordt nog steeds geprogrammeerd, of het nou BASIC is zonder de IDE of met de IDE. Ieder heeft zijn eigen techniek en ieder gebruikt zijn eigen BASIC versie om ermee te kunnen werken.
Dat we vroeger veel werkten met QuickBASIC 4.5 betekent nog niet dat we hiermee klaar zijn. Moeten we speciaal overstappen naar de moderne versies? Helemaal niet. Maar dat betekent niet dat QuickBASIC stil is blijven staan.

QuickBasic nog steeds actueel met de nieuwe QB64 compiler

QB64 is een moderne versie van de programmeertaal BASIC die het mogelijk maakt om QuickBasic 4.5 en QBasic programma's te gebruiken in Windows XP, Vista, Windows 7, Linux en Mac OSX. De compiler werkt op 32- en 64-bit computers.
De taal heeft veel nieuwe mogelijkheden zoals stereo geluid, verbeterde grafische mogelijkheden en nog veel meer. Met QB64 zijn snelle en betrouwbare programma's te maken.

QB64 is een compiler (C++ emitter) met een editor (IDE). Er wordt gestreefd naar 100% compatibiliteit met QuickBasic 4.5. De bedenker en ontwikkelaar van QB64 is Galleon. Hij heeft er lang en hard aan gewerkt.

QB64 streeft niet alleen naar 100% compatibiliteit met QuickBasic 4.5 maar ook naar het uitbreiden van de mogelijkheden van QBasic om tegemoet te komen aan de hedendaagse behoeften, zonder de filosofie achter Basic geweld aan te doen. Nieuwe statements beginnen vaak met een underscore (_) om het compatible te houden met de oude QB-broncode.

Een uitzondering hierop is het verbeterde TIMER-statement. Dit verschilt alleen van het originele TIMER-statement door een nauwkeurigheidsparameter. Dat verandert de compatibiliteit niet. Andere verbeteringen zijn het gebruik van het volledige geheugen, TCP/IP ondersteuning, het laden en afspelen van geluidbestanden en het laden en opslaan van afbeeldingen. Absolute en interrupt calls zijn ingebouwd in QB64. En voor al deze mogelijkheden zijn geen include-bestanden of libraries nodig.

Op het moment is de QB64 community actief in het ontwikkelen van QB64 documentatie (Wiki), het doorgeven van foutrapporten, ideeën, voorbeelden en het helpen van beginners in het QB64 forum.

De community helpt ook Galleon, de ontwikkelaar van QB64, met het oplossen van problemen met de compiler.

De naam QB64 houdt in dat de compiler op 64-bits computers werkt, maar 64-bit adressering is nog niet in QB64 geïmplementeerd. De compiler is nog 32-bits. Het is wel de bedoeling van de ontwikkelaar om 64-bits in QB64 te implementeren. Vanaf versie 0.9 werkt QB64 op 64-bit Linux-systemen.

QB64 is van origine gemaakt in QB 4.5 en is daarmee gecompileerd. Dit was de oplossing voor het geheugenprobleem van QB 4.5 en de latere QBX's. Vanaf versie 0.872 is de broncode in een *.bas bestand vrijgegeven. Wie goed in programmeren is kan zelf nieuwe eigenschappen toevoegen en deze doorgeven aan Galleon op de forums. Deze zal het implementeren daarvan overwegen. Ook gebruikers kunnen vragen om extra opties of suggesties doen ter verbetering van QB64 op het forum. Suggesties voor C++ code en BASIC code zijn hier ook welkom.

Galleon en de community hopen dat u plezier zal hebben met het programmeren in QB64!
Om hulp hoeft u slechts te vragen.

The QB64 community

Het programma is geheel gratis te downloaden van de website: QB64.net
Er is een forum, een online of downloadbare handleiding (reference) en een wiki stie.

Bron: QB64.net/wiki
Artikel: Jan van der Linden

Grafisch programmeren in GW-BASIC (appendix).

In deze appendix laten we zien hoe een aantal programma's uitgebreid kunnen worden om leuke effecten te bereiken. Ook geven we nog een aantal tekeningen, dat u met de programma's kunt maken. Tot slot volgt dan nog een betere versie van het bolprogramma

Programma 4

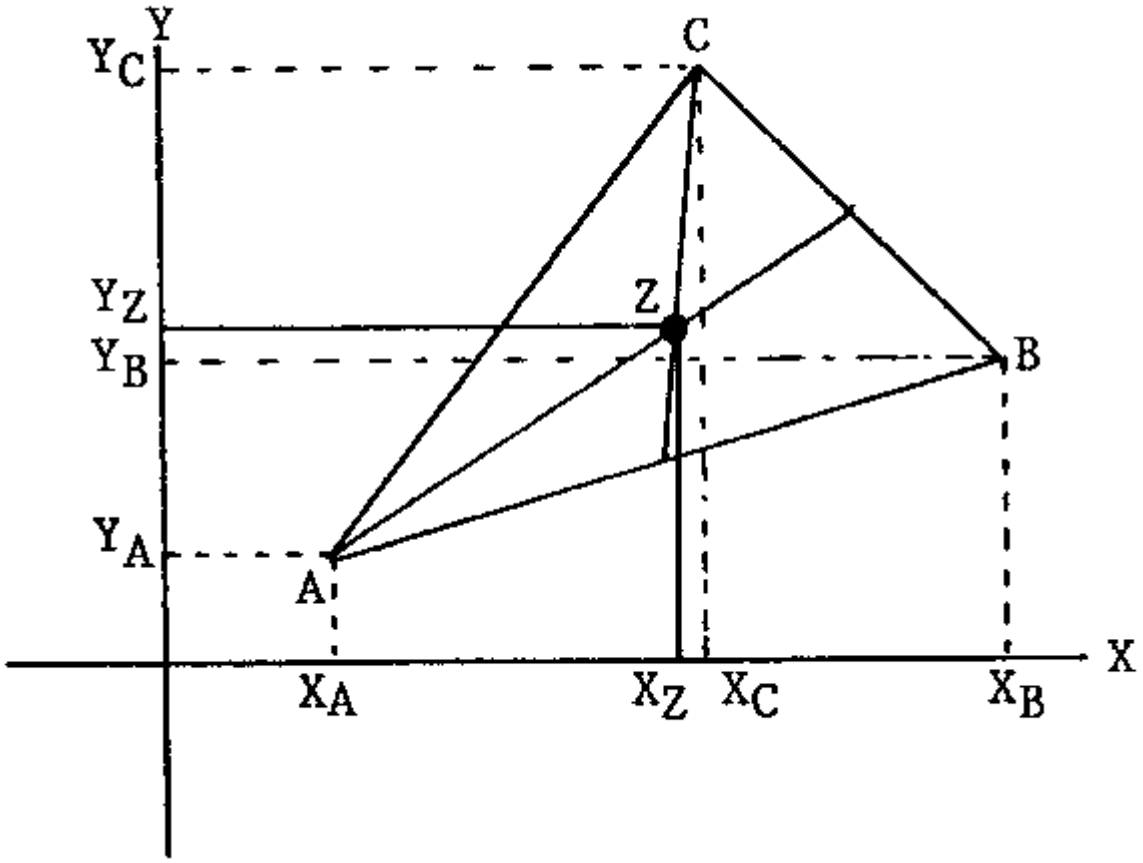
In de tekening, die programma 4 oplevert, zien we dat tussen een zeshoek en de daarbinnen liggende zeshoek zes driehoeken ontstaan. We gaan deze driehoeken om en om 'inkleuren'. We doen dit niet alleen voor de driehoeken tussen de buitenste zeshoek en de eerste ingeschreven zeshoek, maar ook bij de driehoeken tussen de eerste ingeschreven zeshoek en de tweede ingeschreven zeshoek, voor de driehoeken tussen de tweede ingeschreven zeshoek en de derde ingeschreven zeshoek, enzovoorts. Er ontstaan dan drie 'zwarte' spiraalvlakken.

Als we een vlak, in dit geval een driehoek, dat geheel omsloten wordt door lijnen, willen inkleuren, moeten we in dat vlak een punt lokaliseren. De schermcoördinaten van dat punt gebruiken we dan in de PAINT opdracht om het vlak waarin dat punt ligt te kleuren (zwart, of een andere kleur). Omdat er een groot aantal driehoeken gekleurd moet worden, moeten we ook een groot aantal coördinaten van punten in die driehoeken berekenen. Deze punten moeten op een systematische manier in de FOR-NEXT lus (regels 200-320) bepaald worden. Als we in het X-Y vlak een driehoek ABC tekenen, waarvan de coördinaten van de hoekpunten (X_A,Y_A); (X_B,Y_B) en (X_C,Y_C) zijn (zie de tekening op pagina 22), dan weten we uit de meetkunde dat de coördinaten van het zwaartepunt (Z) van de driehoek gelijk zijn aan

$$\left(\frac{X_A + X_B + X_C}{3}, \frac{Y_A + Y_B + Y_C}{3}\right).$$

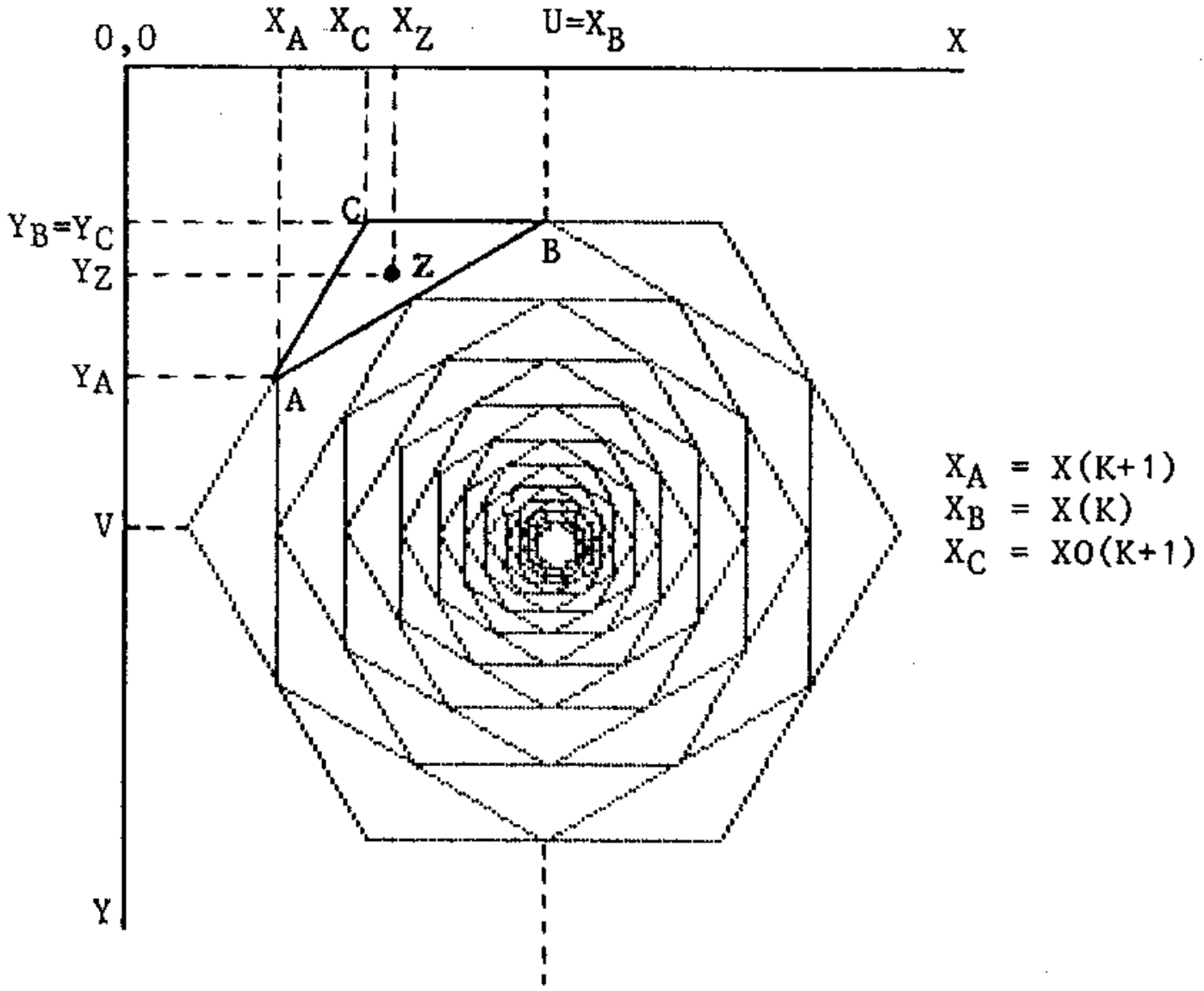
Het zwaartepunt van een driehoek is het snijpunt van de drie lijnen die elk een hoekpunt met het midden van de daar tegenoverliggende zijde verbindt. In de tekening is Z het zwaartepunt en er geldt dus dat

$$X_Z = \frac{X_A + X_B + X_C}{3} \text{ en } \quad Y_Z = \frac{Y_A + Y_B + Y_C}{3}.$$



$X_Z = 1 / 3 (X_A + X_B + X_C)$
$Y_Z = 1 / 3 (Y_A + Y_B + Y_C)$

We kiezen in ons programma steeds het zwaartepunt van de driehoek als inwendig punt. Om de coördinaten hiervan te kunnen berekenen moeten we dus de drie x- en de drie y-coördinaten van de hoekpunten weten. We hebben in de onderstaande tekening, die door programma 4 getekend wordt, één driehoekje als voorbeeld genomen. Van dit driehoekje zijn de hoekpunten A en B twee hoekpunten van de ingeschreven driehoek, terwijl hoekpunt C een hoekpunt is van de daarvoor getekende (in ons voorbeeld de buitenste) zeshoek..
We moeten dus steeds van twee zeshoeken de coördinaten van alle zes hoekpunten ter beschikking hebben. In het onderstaande programma hebben we, om de coördinaten van de hoekpunten van de zojuist getekende zeshoek te kunnen onthouden voordat het nieuwe berekend worden, twee arrays extra opgenomen, en wel XO(6) en YO(6) (van XOud en YOud). In regel 295 maken we XO(J) en YO(J) gelijk aan de coördinaten van de zojuist getekende zeshoek (X(J) en Y(J)), vlak voordat X(J) en Y(J) gelijk worden gemaakt aan de coördinaten van de hoekpunten van de volgende zeshoek (MX(J) en MY(J)) in regel 300.



We zien vervolgens in de regels 233 en 234 hoe we de schermcoördinaten van het zwaartepunt berekenen uit de twee hoekpunten van de nieuwe zeshoek (X(K), X(K+1) en Y(K) en Y(K+1)) en één hoekpunt van de daarvoor getekende zeshoek (XO(K+1) en YO(K+1)).
In de FOR-NEXT lus van regel 232 nemen we de stapgrootte 2 om steeds één driehoek over te slaan. Natuurlijk hoeven we dit inkleuren pas te doen als de tweede zeshoek getekend is, vandaar de IF N=1 THEN 240 in regel 231. Het inkleuren gebeurt tenslotte in regel 235 met PAINT (FNX(X),Y),1. Denk erom dat ook hier FNX(X) in plaats van X genomen wordt, anders wordt een verkeerd vlak gekleurd.

Het aangepaste programma en het resultaat staan op de volgende pagina.

```
100 '          programma 4    INGESCHREVEN ZESHOEKEN
105 '                         MET SPIRAAL
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 DIM X(6),Y(6),MX(6),MY(6),XO(6),YO(6)
140 R=160: U=160: V=160: H=.5: W=60*4*ATN(1)/180
150 FOR J=0 TO 6
160     W1=J*W
170     X(J)=INT(U+R*COS(W1)+H)
180     Y(J)=INT(V-R*SIN(W1)+H)
190 NEXT J
200 FOR N=1 TO 20
210   FOR J=0 TO 5
220     LINE (FNX(X(J)),Y(J))-(FNX(X(J+1)),Y(J+1)),1
230   NEXT J
231   IF N=1 THEN 240
232   FOR K=1 TO 5 STEP 2
233     X=INT((X(K)+X(K+1)+XO(K+1))/3)
234     Y=INT((Y(K)+Y(K+1)+YO(K+1))/3)
235     PAINT(FNX(X),Y),1
236   NEXT K
240   FOR K=0 TO 5
250     MX(K)=INT((X(K)+X(K+1))/2+H)
260     MY(K)=INT((Y(K)+Y(K+1))/2+H)
270   NEXT K
280   MX(6)=MX(0) : MY(6)=MY(0)
290   FOR J=0 TO 6
295     XO(J)=X(J) : YO(J)=Y(J)
300     X(J)=MX(J) : Y(J)=MY(J)
310   NEXT J
320 NEXT N
330 A$=INKEY$: IF A$="" THEN 330
340 CLS: KEY ON: END
```

Nu is het de vraag hoe ik de tekening op een Windows-form hebben kunnen krijgen. Hier heb ik veel naar moeten zoeken voor een oplossing om een soortgelijke uitvoer te krijgen als in GW-BASIC. Het probleem was de PAINT opdracht die vanaf VB 6.0 niet meer bestaat. Er werden andere methoden voor ontworpen om soortgelijke vlakken te kunnen vullen. Deze methoden, zoals de *FillRectangle* en de *FillRegion* methoden, vullen wel gebieden, maar echter met vaste randcoördinaten en niet wat PAINT altijd deed; vullen vanaf de opgegeven punt en alles inkleuren totdat het hele vlak ingekleurd is. We weten dat ook maar één lek in de rand ervoor zorgde dat alles ingekleurd werd. Met de nieuwe methoden zijn we van dat probleem af, maar we kunnen nu niet meer zomaar een vlak vullen vanaf een opgegeven punt. Dat is jammer.

Toch heb ik de oplossing gevonden. Met de methode *FillPolygon* kunnen door middel van opgegeven hoekpunten allerlei vlakken ingekleurd worden. Kijk maar eens naar hetzelfde programma in Visual Basic 2008 code en zie de wijzigingen.

```
Public Class frmProg4
    Const R = 160, U = 160, V = 160, H = 0.5

    Private Sub frmProg4_Paint(…, ByVal e As ….PaintEventArgs) …
        Dim X(6), Y(6), MX(6), MY(6), XO(6), YO(6) As Integer
        Dim W As Single = 60 * Math.PI / 180
        For J As Integer = 0 To 6
            Dim W1 As Single = J * W
            X(J) = Int(U + R * Math.Cos(W1) + H)
            Y(J) = Int(V - R * Math.Sin(W1) + H)
        Next
        For N As Integer = 1 To 20
            For J As Integer = 0 To 5
                e.Graphics.DrawLine(Pens.Black, X(J), Y(J), _
                                                X(J + 1), Y(J + 1))
            Next
            If N <> 1 Then
                For K As Integer = 1 To 5 Step 2
                    Dim PointA As Point = New Point(X(K + 1), Y(K + 1))
                    Dim PointB As Point = New Point(X(K), Y(K))
                    Dim PointC As Point = New Point(XO(K + 1), YO(K + 1))
                    Dim Driehoek As Point() = {PointA, PointB, PointC}
                    e.Graphics.FillPolygon(Brushes.Black, Driehoek)
                Next
            End If
            For K As Integer = 0 To 5
                MX(K) = Int((X(K) + X(K + 1)) / 2 + H)
                MY(K) = Int((Y(K) + Y(K + 1)) / 2 + H)
            Next
            MX(6) = MX(0) : MY(6) = MY(0)
            For J As Integer = 0 To 6
                XO(J) = X(J) : YO(J) = Y(J)
                X(J) = MX(J) : Y(J) = MY(J)
            Next
        Next
    End Sub
End Class
```

Nu ik de hoekpunten uit de berekende coördinaten neem en deze in Point objecten bewaar, hoef ik niet meer de twee formules uit te voeren met X en Y om het zwaartepunt te krijgen. De drie hoekpunten moesten opgeteld worden en daarna worden gedeeld door 3. Door nu direct de hoekpunten te gebruiken en deze aan FillPolygon mee te geven, is het zwaartepunt niet meer nodig. Zorg er wel voor dat u na het aanmaken van de hoekpunten eerst alles in een vlak opslaat, zoals ik deze de Driehoek heb genoemd. FillPolygon vraagt namelijk om een dynamische Point array. U kunt dus de leukste figuurlijke vlakken ontwerpen dankzij de Point objecten.

Tip! Wilt u niet de vlakken vullen? Gebruik dan de methode *DrawPolygon*.

Programma 36

We gaan de kaart van Zwitserland verfraaien met de Zwitserse vlag.
Eerst geven we het gewijzigde programma.

```
100 'programma 36     KAART VAN ZWITSERLAND MET WAPEN
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 K=2 : H=.5 : V=300
150 READ X,Y
160 X1=INT(K*X+H) : Y1=INT(V-K*Y+H)
170 READ X,Y
180 WHILE X <> 0
190    X2=INT(K*X+H) : Y2=INT(V-K*Y+H)
200    LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
210    LINE (FNX(X1),Y1+1)-(FNX(X2),Y2+1),1
220    X1=X2 : Y1=Y2
230    READ X,Y
240 WEND
300 A$=INKEY$: IF A$="" THEN 300
310 CLS: KEY ON: END
320 DATA 69,108,71,107,70,104,75,104,76,106
330 DATA 80,107,81,104,86,105,91,108,94,107
340 DATA 101,106,100,108,105,108,106,110,101,110
350 DATA 98,112,102,117,108,118,112,112,114,115
360 DATA 118,110,128,110,139,102,145,103,146,95
370 DATA 142,86,144,78,154,77,154,72,163,67
380 DATA 168,68,173,76,177,73,174,59,177,56
390 DATA 177,52,171,51,167,56,161,50,165,43
400 DATA 166,34,162,34,157,42,143,36,139,48
410 DATA 136,45,133,48,132,40,122,23,125,15
420 DATA 122,11,119,12,114,20,116,25,100,35
430 DATA 102,45,94,42,88,35,90,28,79,15
440 DATA 75,15,66,19,60,14,52,12,48,13
450 DATA 37,29,39,36,37,40,39,43,29,45
460 DATA 16,38,18,33,13,29,6,28,6,32
470 DATA 11,34,13,40,10,45,12,53,25,63
480 DATA 26,73,30,73,48,94,42,94,46,102
490 DATA 54,102,53,99,61,98,63,102,69,108
500 DATA 0,0
```


```
Public Class frmProg36

    Private Const K = 2
    Private Const H = 0.5
    Private Const V = 300
    Private Kaart() As Byte = _
        { _
            69, 108, 71, 107, 70, 104, 75, 104, 76, 106, _
            80, 107, 81, 104, 86, 105, 91, 108, 94, 107, _
            101, 106, 100, 108, 105, 108, 106, 110, 101, 110, _
            98, 112, 102, 117, 108, 118, 112, 112, 114, 115, _
            118, 110, 128, 110, 139, 102, 145, 103, 146, 95, _
            142, 86, 144, 78, 154, 77, 154, 72, 163, 67, _
            168, 68, 173, 76, 177, 73, 174, 59, 177, 56, _
```

```
            177, 52, 171, 51, 167, 56, 161, 50, 165, 43, _
            166, 34, 162, 34, 157, 42, 143, 36, 139, 48, _
            136, 45, 133, 48, 132, 40, 122, 23, 125, 15, _
            122, 11, 119, 12, 114, 20, 116, 25, 100, 35, _
            102, 45, 94, 42, 88, 35, 90, 28, 79, 15, _
            75, 15, 66, 19, 60, 14, 52, 12, 48, 13, _
            37, 29, 39, 36, 37, 40, 39, 43, 29, 45, _
            16, 38, 18, 33, 13, 29, 6, 28, 6, 32, _
            11, 34, 13, 40, 10, 45, 12, 53, 25, 63, _
            26, 73, 30, 73, 48, 94, 42, 94, 46, 102, _
            54, 102, 53, 99, 61, 98, 63, 102, 69, 108, _
            0, 0 _
        }
    Private Sub frmProg36_Paint(…, ByVal e As ….PaintEventArgs) …
        Dim X2, Y2 As Integer
        Dim KaartP(Int((Kaart.Count - 1) / 2)) As Point

        Dim X As Integer = Kaart(0)
        Dim Y As Integer = Kaart(1)

        Dim X1 As Integer = Int(K * X + H)
        Dim Y1 As Integer = Int(V - K * Y + H)
        KaartP(0) = New Point(X1, Y1)
        Dim P As Integer = 1

        X = Kaart(2) : Y = Kaart(3)
        Dim Data As Integer = 4
        While X <> 0
            X2 = Int(K * X + H) : Y2 = Int(V - K * Y + H)
            e.Graphics.DrawLine(Pens.White, X1, Y1, X2, Y2)
            e.Graphics.DrawLine(Pens.White, X1, Y1 + 1, X2, Y2 + 1)
            X1 = X2 : Y1 = Y2
            X = Kaart(Data) : Y = Kaart(Data + 1)
            KaartP(P) = New Point(X1, Y1)
            P += 1
            Data += 2
        End While
        e.Graphics.FillPolygon(Brushes.White, KaartP)
        e.Graphics.FillRectangle(Brushes.Black, 130, 150, 60, 20)
        e.Graphics.FillRectangle(Brushes.Black, 150, 130, 20, 60)
    End Sub
End Class
```

Het zal duidelijk zijn dat de in dit boek gegeven programma's onbeperkt uitgebreid kunnen worden. Ze kunnen vast ook verbeterd worden; de invoerwaarden zouden bijvoorbeeld gecontroleerd kunnen worden, zodat het programma niet ergens halverwege afbreekt door een foutieve schermcoördinaat. Wij hebben dat allemaal niet gedaan, omdat we de programma's klein wilden houden, waardoor alle aandacht op de tekentechniek gericht blijft.

Bron: IBM- en GW-BASIC graphics van Academic Service
Tekst overname, tips en veranderingen: Marco Kurvers

Cursussen

Liberty Basic:
Cursus en naslagwerk, beide met voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Qbasic:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

QuickBasic:
Cursusboek en het lesvoorbeeld op diskette, € 11,00 voor leden. Niet leden € 13,50.

Visual Basic 6.0:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Basiscursus voor senioren, Windows 95/98,
Word 97 en internet voor senioren, (geen diskette). € 11,00 voor leden. Niet leden € 13,50.

Computercursus voor iedereen: tekstverwerking met Office en eventueel met VBA, Internet en programmeertalen, waaronder ook Basic, die u zou willen leren.
Elke dinsdag in buurthuis Bronveld in Barneveld van 19:00 uur tot 21:00 uur. Kosten € 5,00 per week.
Meer informatie? Kijk op 'http://www.i-t-s.nl/rdkcomputerservice/index.php' of neem contact op met mij.

Computerworkshop voor iedereen; heeft u vragen over tekstverwerking of BASIC, dan kunt u elke 2de en 4de week per maand terecht in hetzelfde buurthuis Bronveld in Barneveld van 19:30 uur tot 21:30 uur. Kosten € 2,00.
Meer informatie? Kijk op 'http://www.buurthuisbronveld.nl' of neem contact op met mij.

Software

Catalogusdiskette,	€ 1,40 voor leden. Niet leden € 2,50.
Overige diskettes,	€ 3,40 voor leden. Niet leden € 4,50.
CD-ROM's,	€ 9,50 voor leden. Niet leden € 12,50.

Hoe te bestellen

De cursussen, diskettes of CD-ROM kunnen worden besteld door het sturen van een e-mail naar penm@basic-gg.hcc.nl en storting van het verschuldigde bedrag op:

ABN-AMRO nummer 49.57.40.314
HCC BASIC ig
Haarlem

Onder vermelding van het gewenste artikel. Vermeld in elk geval in uw e-mail ook uw adres aangezien dit bij elektronisch bankieren niet wordt meegezonden. Houd rekening met een leveringstijd van ca. 2 weken.
Teksten en broncodes van de nieuwsbrieven zijn te downloaden vanaf onze website (http://www.basic.hccnet.nl). De diskettes worden bij tijd en wijlen aangevuld met bruikbare hulp- en voorbeeldprogramma's.
Op de catalogusdiskette staat een korte maar duidelijke beschrijving van elk programma.

Alle prijzen zijn inclusief verzendkosten voor Nederland en België.

Vraagbaken

De volgende personen zijn op de aangegeven tijden beschikbaar voor vragen over programmeerproblemen. Respecteer hun privé-leven en bel alstublieft alleen op de aangegeven tijden.

Waarover	Wie	Wanneer	Tijd	Telefoon	Email
Liberty Basic	Gordon Rahman	ma. t/m zo.	19-23	(023) 5334881	grahman@planet.nl
MSX-Basic	Erwin Nicolai	vr. t/m zo.	18-22	(0516) 541680	basic@lordthanatos.com
PowerBasic CC	Fred Luchsinger	ma. t/m vr.	19-21		f.luchsinger@kader.hcc.nl
QBasic, QuickBasic	Jan v.d. Linden				j.vd.linden@kader.hcc.nl
Visual Basic voor Windows	Jeroen v. Hezik	ma. t/m zo.	19-21	(0346) 214131	j.a.van.hezik@kader.hcc.nl
Visual Basic .NET	Marco Kurvers	do. t/m zo.	19-22	(0342) 424452	m.a.kurvers@hccnet.nl
Basic algemeen, zoals VBA Office	Marco Kurvers	do. t/m zo.	19-22	(0342) 424452	m.a.kurvers@hccnet.nl
Web Design, met XHTML en CSS