Nieuwsbrief

19de jaargang juni 2012 **Nummer 2**

HCC BASIC interessegroep

Inhoud

Deze uitgave kwam tot stand met bijdragen van:

Naam	*Blz*

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

We weten natuurlijk wanneer we iets een goede kwaliteit noemen, maar wat is eigenlijk een kwantiteit? Sterker nog, de compiler herkent zelf alleen de kwaliteit en niet de kwantiteit. U kunt het al raden: de kwantiteit kunnen we vinden tijdens een programma-uitvoer.

Het deel over 3D tekenen en tekenen van vlakken in de ruimte was zeer interessant en leerzaam. In dit deel van ‘Grafisch programmeren in GW-BASIC’ ga ik het over heel wat anders hebben: tekenen met de programmeertaal LOGO. U zult versteld staan hoe een klein tekenprogrammaatje in LOGO ontzettend veel BASIC code in beslag neemt.

Om te kunnen programmeren met FreeBASIC kan ik u zeggen dat het een behoorlijke verandering is, zeker voor Liberty BASIC gebruikers, en helemaal zeker wanneer we FbEdit erbij nemen. In de vorige nieuwsbrief heb ik u daar wat van laten zien. Ik zou ook, heb ik toen gezegd, hiermee verder gaan in deze nieuwsbrief. Het kost echter veel tijd en moeite om alles over FreeBASIC en FbEdit te vinden. Daarom heb ik besloten om hier niet verder op in te gaan. Mijn excuses daarvoor.

Marco Kurvers

BASIC leren – PowerBASIC (4).

Expressies

Een expressie bestaat uit operatoren en operanden. *Operatoren* zijn symbolen of woorden, zoals het plusteken (+), die wiskundige, relationele, of logische bewerkingen (Boolean) vertegenwoordigen. Operanden zijn de hoeveelheden waarop bewerkingen worden uitgevoerd. Net als de gegevenstypen zijn er twee fundamentele typen expressies in PowerBASIC: numerieke en tekenreeksen.

String expressies

Een *string expressie* bestaat uit string constanten, string variabelen en string functies, optioneel gecombineerd met de concatenatie operator (+). String expressies geven altijd strings terug als resultaat. Voorbeelden van string expressies zijn:

```
"Katten en honden"                  ' string constante
voornaam$                           ' string variabele
voornaam$ + achternaam$             ' de concatenatie operator
LEFT$(a$ + z$,7)                    ' string functie
a$ + MID$("Katten en honden",5,3)
RIGHT$(MID$(a$ + z$,1,6),3)
```

TIP! Concatenatie wordt ook wel *samenvoegen* genoemd.

Onthoud dat flex strings en vaste-lengte strings altijd een vaste-lengte hebben, respectievelijk gedefinieerd in the corresponderende MAP of DIM statement, daardoor zal de string concatenatie met deze strings heel anders werken dan u zult denken. Bijvoorbeeld, het volgende programmafragment:

```
MAP groeten$$*40
groeten$$ = "hallo"
groeten$$ = groeten$$ + "daar"
```

voegt de vier-karakter string *daar* toe aan de 40-karakter flex string (`hallo`, gevolgd door 35 spaties), maar het resultaat is afgekapt tot 40 karakters (de voorgedefinieerde lengte van `groeten$$`), en de nieuwe toegevoegde string zal daardoor verloren gaan. Eén oplossing voor dit probleem is gebruik te maken van de RTRIM$ functie om de overtollige spaties van `hallo` te verwijderen voordat `daar` toegevoegd wordt:

```
MAP groeten$$*40
groeten$$ = "hallo"
groeten$$ = RTRIM$(groeten$$) + " daar"
```

Numerieke expressies

Numerieke expressies, niet verrassend, zijn gevormd uit numerieke constanten, variabelen en functies, optioneel gescheiden door numerieke operatoren. Numerieke expressies produceren een waarde in een van de elf numerieke types (byte, word, integer, double word, long integer, quad integer, single, double, extended precision, BCD fixed point, of BCD floating point). Voorbeelden van numerieke expressies zijn:

```
37
37/15
a
37/a
SQR (37/a)
SQR ((c + d)/a) * SIN (37/a)
```

U moet zich van bewust zijn dat numerieke expressies een uitvoer hebben in een bewerkingsvolgorde

volgens een bepaalde hiërarchie. De volgende lijst geeft de expressie evaluatievolgorde.

Machtsverheffing heeft de hoogste prioriteit, dat wil zeggen het wordt als eerste uitgevoerd; IMP heeft de laagste prioriteit, dat wil zeggen het wordt als laatste uitgevoerd.

- machtsverheffing (^)
- negatie (-)
- vermenigvuldiging (*), drijvende komma (floating point) deling (/)
- integer deling (\)
- modulo (MOD)
- optelling (+), aftrekking (-)
- relationele operatoren (<, <=, =, >=, >, <>)
- NOT, ISFALSE, ISTRUE
- AND
- OR, XOR (exclusief OR)
- EQV (gelijkwaardigheid)
- IMP (implicatie)

Als voorbeeld, de expressie `3 + 6 / 3` evalueert naar 5, niet 3. De deling heeft een hogere prioriteit dan de optelling, dus de deling operatie (`6 / 3`) zal eerst uitgevoerd worden. De compiler zal hier niet door verward raken, maar mensen helaas wel. Dus een betere programmeerstijl `3 + (6 / 3)` of `3 + 6/3` te gebruiken, met beide haakjes of de afstand tussen de intentie duidelijk te maken.

De bewerkingen van dezelfde prioriteit worden in PowerBASIC van links naar rechts uitgevoerd. Bijvoorbeeld in de expressie `4 - 3 + 6` is de aftelling (`4 - 3`) uitgevoerd voordat de optelling (`3 + 6`) wordt uitgevoerd. Dit produceert de tussentijdse expressie 1 + 6.

Bewerkingen tussen haakjes hebben de hoogste prioriteit en worden altijd als eerste uitgevoerd; binnen de haakjes wordt de voorrang standaard gebruikt. Gebruik haakjes zoals knoflook—royaal, maar niet overtollig.

Operatoren

De *numerieke operatoren* worden ingedeeld in drie grote groepen: rekenkundig, relationeel en logisch.

Rekenkundige operatoren

Rekenkundige operatoren voeren normale rekenkundige bewerkingen uit. Het volgende tabel laat de rekenkundige PowerBASIC operatoren zien in gesorteerde volgorde.

Sommige van deze operatoren hebben wat uitleg nodig. De backslash (\) geeft een integer deling. Integer delingen ronden het af tot gehele waarden zonder de restwaarden. Bijvoorbeeld, `5 \ 2` evalueert naar 2 en `9 \ 10` evalueert naar 0.

De rest van een integer deling kan worden achterhaald met de MOD (modulo) operator (MOD is geldig voor alle numerieke types). MOD is soortgelijk aan een integer deling, behalve dat het de *rest* van de deling resulteert. Bijvoorbeeld, 5 MOD 2 resulteert de waarde 1 en 9 MOD 10 resulteert de waarde 9.

Als elke operatie van een integer getal deling, of van een MOD operator, buiten het bereik van een quad integer getal is, zal er een overloopfout optreden.

Operator	Actie	Voorbeeld
^	Machtsverheffen	10 ^ 4
-	Negatie	-16
*, /	Vermenigvuldiging en drijvende komma deling	45 * 19, 45 / 19
\	Integer deling	45 \ 19
MOD	Modulo	45 MOD 19
+, -	Optelling, aftrekking	45 + 19, 45 - 19
ISFALSE,	Booleaanse onjuistheid	ISFALSE 45
ISTRUE	en juistheid	ISTRUE 19
NOT, AND,	Bit manipulatie	NOT 0, 45 AND 19
OR, XOR,	operatoren	45 OR 19, 45 XOR 19
EQV, IMP		45 EQV 19, 45 IMP 19

Bit manipulaties

Naast het maken van complexe tests hebben logische operatoren controle over de onderliggende bit patronen van hun geheel getal bewerkingen. De meest gebruikte operatoren zijn AND, OR en XOR (exclusief OR).

Onderstaande tabellen laten uit bovengenoemde operatoren voorbeelden zien wat de resultaten zijn uit de opgegeven maskers. Het masker is de vergelijkingswaarde met de eerste waarde. Logische operatoren werken altijd bitgewijs, zoals u uit de onderstaande tabellen kunt zien.

Een AND operator wijzigt niet de bits als beide bits waar of onwaar zijn, met uitzondering als een bit uit de waarde op 0 staat (onwaar) en een bit in het masker op 1 staat (waar) dan zal de bit in de resultaatwaarde ook op 0 staan, zie hieronder.

.

	1001	0111	0000	0000	&H9700	
AND	0011	1111	1111	1111	&H3FFF	(masker)
	0001	0111	0000	0000	&H1700	(resultaat)

Een OR operator wijzigt niet de bits als beide bits waar of onwaar zijn, met uitzondering als een bit uit de waarde op 1 staat (waar) en een bit in het masker op 0 staat (onwaar) dan zal de bit in de resultaatwaarde ook op 1 staan, zie hieronder.

	1001	0111	0000	0000	&H9700	
OR	1100	0000	0000	0000	&HC000	(masker)
	1101	0111	0000	0000	&HD700	(resultaat)

Een XOR operator wijzigt niet de bits als beide bits op nul staan of wanneer alleen de bits in het masker op nul staan, zie hieronder.

	1001	0111	0000	0000	&H9700	
XOR	1100	0000	0000	0000	&HC000	(masker)
	0101	0111	0000	0000	&H5700	(resultaat)

Natuurlijk zullen de resultaten uit 0 AND 1 en 1 AND 0 hetzelfde zijn, maar wat ik uit bovenstaande tabellen duidelijk wil maken is de *manipulatie*. Hier gaat het om welke bits beïnvloedt worden en welke niet. U kunt uit een 0 AND 1 al zien dat de bit in de resultaatwaarde niet beïnvloedt wordt en dus op 0 zal blijven staan. Dat is niet het geval wanneer we een OR zouden gebruiken.

XOR operaties zijn gebruikelijk in graphics, omdat zij toestaan dat een object tegen een complexe achtergrond beeldgebieden verplaatst. Trek met XOR het object eenmaal tegen de achtergrond, XOR vervolgens hetzelfde object om op dezelfde plaats te wissen, dan de achtergrond terugzetten naar de oorspronkelijke toestand.

Bit-level rekenkundige statements

Er zijn vele malen nodig in programmeren wanneer u verschillende Ja/Nee variabelen wilt bijhouden. Traditioneel zijn er twee manieren om dit te doen:

- Een aparte integer-variabele instellen voor het bijhouden van elke voorwaarde. Bijvoorbeeld: *CondensedPrint* (true als verkorte modus is ingeschakeld), *Changed* (true als inhoud van het bestand is bewerkt, maar niet is opgeslagen).
- Met de individuele bits van een enkele variabele "vlag" kunt u een hele reeks vlaggen aanwijzen. Als u bijvoorbeeld bit 7 van *CondensedPrint* hebt verwezen, zou u de staat kunnen testen of de vlag van de variabele in- of uitgeschakeld is, zodat u de huidige stand van CondensedPrint kunt bepalen.

Van de twee manieren is de tweede zuiniger met het geheugen, maar in oude BASIC stijl is er meer programmering bij betrokken en meer werk nodig van uw kant. PowerBASIC maakt dit veel makkelijker voor u met de BIT functie en statement. Gebruik, om te zien of een bit gezet is:

```
i = BIT(FlagVar, BitNum)
```

waar *FlagVar* de variabele of array is met de inhoud van een set van bits en *BitNum* het nummer is van de bit die u wilt testen.

Om individuele bits te zetten of te wissen, gebruik het BIT statement. De syntaxis is:

```
BIT {SET|RESET|TOGGLE} var, bitnum
```

De SET optie schakelt (zet op één) de aangegeven bit in, RESET wist (zet op nul) de bit en TOGGLE wisselt de waarde van de bit (als de huidige waarde één is, wordt het nul en als de huidige waarde nul is, wordt het één).

```
%CONDENSED = 1                      ' Bit posities voor
%BOLD = 2                           ' sommige vlagwaarden.
%EXPANDED = 3

' Flags? is de byte die gebruikt wordt om deze vlaggen in op te slaan.

BIT SET Flags?, CONDENSED           ' Schakel CONDENSED bit in
BIT RESET Flags?, BOLD              ' Wis BOLD bit
BIT TOGGLE Flags?, EXPANDED         ' Wissel EXPANDED bit
```

Voor grotere risico's, PowerBASIC presteert geen foutcontrole op de argumenten die u op het BIT statement toepast. U moet voorzichtig zijn niet de grenzen te overschrijden van de variabele of array die u toepast; als u het doet, zal u de inhoud van een andere variabele of array worden overschreven. Hoewel, een goed neveneffect is dat u met BIT een array kunt toepassen van elke grootte en gedurende die array toegang heeft naar de bits.

```
DIM Flags?(1:30000)
BIT TOGGLE Flags(1), 240000&
```

Rekenkundige schuif en rotatie statements

Misschien vindt u het nuttig de aantal bits te kunnen manipuleren in een enkele bewerking. PowerBASIC's SHIFT en ROTATE statements laat u dit doen.

Verschuiven en draaien zijn vergelijkbaar. Het verschil is dat wanneer u schuift, de geschoven bits eruit worden gegooid; wanneer u roteert, zijn de bits geroteerd aan de andere kant. Hieronder kunt u de verschillen zien:

```
1 1 0 1 1 1 0 1                 1 1 0 1 1 1 0 1
1 0 1 1 1 0 1 0                 1 0 1 1 1 0 1 1
Naar links geschoven bit        Naar links geroteerde bit
```

Relationele operatoren

Relationele operatoren staat u toe om waarden van twee expressies te vergelijken om een Booleaans resultaat te verkrijgen van TRUE of FALSE. Ze kunnen in elke numerieke expressie gebruikt worden (bijvoorbeeld, `a = (b > c) / 13)`, en de numerieke resultaten teruggegeven door relationele operatoren gebeuren in het algemeen in een IF of andere statement instructies die uiteindelijk een oordeel maken over de programmeerstroom.

Operator	Relatie	Voorbeeld
=	Gelijk aan	5 = 5
<>	Ongelijk aan	5 <> 6
<	Kleiner dan	5 < 6
>	Groter dan	6 > 5
<=, =<*	Kleiner dan of gelijk aan	5 <= 6
>=, =>*	Groter dan of gelijk aan	6 >= 5

✶ **Bij de laatste twee operatoren: waar een sterretje staat betekent dat in uw BASIC dialect deze operatoren (=< en =>) niet gebruikt kunnen worden! Controleer in de help of dit het geval is.**

Wanneer rekenkundige en relationele operatoren in een expressie worden gecombineerd, worden rekenkundige operaties altijd als eerste geëvalueerd. Bijvoorbeeld, `4 + 5 < 4 * 3` evalueert naar TRUE omdat de rekenkundige operaties (optelling en vermenigvuldiging) eerder zijn uitgevoerd dan de relationele operaties. Dit test dan de waarheid van de bewerking `9 < 12`.

Logische operatoren

Logische operatoren voeren logische (Booleaanse) operaties uit op getallen van elk type. Voordat een Boolean operatie plaatsvindt, converteert PowerBASIC automatisch drijvende komma getallen naar integers. Samen met het gebruik van de relationele operatoren, de logische operatoren staan toe complexe tests in te kunnen stellen zoals

```
IF dag > 29 AND maand = 2 THEN PRINT "Op welke planeet zijn we?"
```

AND

Deze verklaring voert een logische AND op de resultaten van de twee integer geretourneerde relationele operators. De AND operator resulteert een TRUE als beide operaties TRUE zijn. De AND operator heeft een lagere prioriteit dan de > en = relationele operatoren, dus parentheses (ronde haakjes) zijn niet nodig. Bijvoorbeeld, als de waarde van *dag* 30 is en van *maand* ingesteld is op 2, zullen beide relationele operatoren TRUE resulteren. De AND operator verklaart een logische AND op deze twee TRUE resultaten en zal daarom ook zelf een TRUE als resultaat geven.

OR

De OR (soms genoemd als inclusief OR) operator resulteert TRUE als een of beide argumenten TRUE zijn en resulteert alleen FALSE als beide argumenten FALSE zijn. Bijvoorbeeld,

```
 -1        OR        0         is        TRUE
  0        OR        0         is        FALSE
5 > 6      OR      6 < 4       is        FALSE
```

XOR
De XOR (exclusief OR) operator resulteert TRUE als de geteste waarden verschillend zijn en resulteert FALSE als ze hetzelfde zijn. Bijvoorbeeld,

```
 -1        XOR       0         is        TRUE
 -1        XOR      -1         is        FALSE
5 > 6      XOR     6 < 4       is        TRUE
```

EQV
De EQV (gelijkwaardigheid) operator is het tegenovergestelde van XOR. Het resulteert TRUE als de twee logische geteste waarden hetzelfde zijn en resulteert FALSE als ze het beide niet zijn:

```
 -1        EQV       0         is        FALSE
 -1        EQV      -1         is        TRUE
5 > 6      EQV     1 > 99      is        TRUE
```

IMP
De IMP (implicatie) operator resulteert alleen FALSE als de eerste operand is TRUE en de tweede FALSE:

```
-1   IMP  -1   is   TRUE
 0   IMP  -1   is   TRUE
 0   IMP   0   is   TRUE
-1   IMP   0   is   FALSE
```

ISTRUE
De ISTRUE operator resulteert alleen TRUE als de operand TRUE is. ISTRUE resulteert gegarandeerd -1 als een TRUE waarde, terwijl de andere operatoren elke niet-nul zijnde waarde kunnen resulteren:

```
ISTRUE     0   is   FALSE
ISTRUE    -1   is   TRUE      (-1)
ISTRUE    200  is   TRUE      (-1)
```

ISFALSE
De ISFALSE (logische negatie) operator resulteert alleen TRUE als zijn operand FALSE is. ISFALSE is soortgelijk aan de NOT operator, maar resulteert de logische TRUE negatie van de operand:

```
ISFALSE    0   is   TRUE (-1)      NOT   0   is   TRUE       (-1)
ISFALSE   -1   is   FALSE          NOT  -1   is   FALSE       (0)
ISFALSE   200  is   FALSE          NOT  200  is   TRUE      (-201)
```

Onderstaand tabel geeft een overzicht van de logische operatoren.

x	y	x AND y	x OR y	x XOR y	x EQV y	x IMP y	NOT x	ISTRUE x	ISFALSE x
T	T	T	T	F	T	T	F	T	F
T	F	F	T	T	F	F	F	T	F
F	T	F	T	T	F	T	T	F	T
F	F	F	F	F	T	T	T	F	T

Onthoud dat de logische operatoren met alle integer types kunnen werken, maar niet met de drijvende komma waarden. Als de operanden van een logische expressie buiten de grens van een quad integer (-2^{63} tot $2^{63}-1$) zijn, zal er een overflow fout verschijnen:

```
x = 2^66
IF x OR y THEN GOTO Done
```

Dit IF statement resulteert in een overlow fout wanneer *x* niet met succes geconverteerd kan worden naar een quad integer.

Strings en relationele operatoren

PowerBASIC geeft u een mogelijkheid om stringgegevens te vergelijken. String expressies kunnen getest worden voor vergelijkingen, evenals voor “groter dan” en “kleiner dan” alfabetische volgorde.

Twee string expressies zijn alleen gelijk als ze exact dezelfde karakters hebben in exact dezelfde volgorde. Bijvoorbeeld,

```
a$ = "CAT"
PRINT a$ = "CAT"; a$ = "CATS"; a$ = "cat"
```

String volgorde is gebaseerd onder twee criteria: eerst, de ASCII waarden van de karakters die ze hebben, en twee, de lengte van de strings.

Bijvoorbeeld, de letter *A* is kleiner dan de letter *B* omdat de ASCII code van *A* 65 is en is daarom kleiner dan de code van *B*, namelijk 66. Opmerking, *B* is kleiner dan *a* omdat de ASCII code van elke kleine letter *groter* is dan de corresponderende hoofdletter (precies 32 groter). Wanneer gemixte hoofdletter en kleine letter informatie vergeleken moet worden, gebruik dan de UCASE$ of LCASE$ functies om de verschillen te behouden die van invloed zijn met de test.

```
stad1$ = "Seattle"
stad2$ = "Tucson"
IF UCASE$(stad1$) > UCASE$(stad2$) THEN
     PRINT stad1$
ELSE
     PRINT stad2$
END IF
stad1$ = UCASE$(stad1$)
stad2$ = UCASE$(stad2$)
IF stad1$ > stad2$ THEN
     PRINT stad1$
ELSE
     PRINT stad2$
END IF
```

Onthoud het verschil tussen de twee statementblokken. In het eerste geval worden de stringvariabelen *stad1$* en *stad2$* geconverteerd in hoofdletters voor alleen de vergelijking, dus de eerste IF/THEN print `Tucson` uit. In het tweede geval vindt de conversie in de variabelen zelf plaats, zodat het resultaat `TUCSON` zal zijn.

De lengte is alleen belangrijk als beide strings even lang zijn of de smallere string evalueert in dit geval: smallere kleiner dan de langere. Als voorbeeld, *KAT* is kleiner dan *KATTEN*.

De ARRAY SORT en ARRAY SCAN statements geven de mogelijkheid of de kleine letters als hoofdletters als doel voor de vergelijking behandeld moet worden. U kunt ook een string opgeven die precies de sorteervolgorde bepaalt voor alle 256 ASCII tekens.

Volgend hoofdstuk

In de volgende nieuwsbrief gaan we dieper in op verschillende types, de gegevensstructuren. Onder andere wat gebruiker gedefinieerde types inhoudt

Marco Kurvers

Kwaliteit en kwantiteit.

Kom je schat? –Ja, ik kom, ik heb net me gedoucht, maar moet me nog aankleden!
Ach, dat geeft toch niet? We zijn toch met z'n tweeën!

Uit bovenstaande alinea kunnen we een conclusie maken.
Natuurlijk kunnen we ongekleed naar beneden lopen, maar we willen dat liever niet. Wat is de reden? We kunnen het wel koud hebben, hoewel dat niet erg is. De ene kan er beter tegen dan de ander.
Uit de conclusie kunnen we dus de kwaliteit vinden (zit het lichaam goed in elkaar?) en de kwantiteit (kan de persoon rond lopen zonder het koud te hebben?).

De kwaliteit is dus dat het voor een gezond mens niet uitmaakt hoe hij/zij door het huis loopt. Je kunt je gewoon bewegen.
De kwantiteit is echter dat je het zelf helemaal niet fijn vindt om zo rond te lopen; zoals gezegd: je kunt het wel koud hebben.

De kwaliteit en kwantiteit kunnen we ook in programmacode vinden. De leverancier moet een goede kwaliteit aan software kunnen leveren. Dat betekent, de software moet goed werken zonder fouten. Helaas kan het gebeuren dat er toch *schoonheidsfoutjes* ontstaan, foutjes die alleen naar voren komen tijdens het gebruik van de software. Dus niet alleen de bouwstenen van een programma moeten goed in elkaar zitten, ook moet er op het gebruik van het programma worden gelet. Dit noemen we de *kwantiteit* van het programma.

De kwaliteit en kwantiteit hebben dus niet dezelfde betekenis. Om meer hierover te weten, gaan we de kwaliteit en kwantiteit in BASIC uitzoeken.

De kwaliteit

Stel, we gaan een NAW programma maken. We moeten, zeg maar van onze "baas", er voor zorgen dat het programma de juiste onderdelen heeft. De onderdelen kunnen zijn:

- berichtvenster
- invoerformulier voor de NAW gegevens
- recordstructuur om de NAW gegevens in op te slaan
- een datastructuur om alle records in op te slaan, deze moet naar een bestand weggeschreven kunnen worden
- hoofdvenster
- hoofdmenu

- werkbalk(en)
- overzicht of gegevenstabel om alle NAW gegevens op het scherm te kunnen zien
- mogelijkheid om af te drukken

Als we dit allemaal hebben met de code erbij, kunnen we het programma starten.

Perfect! Het programma werkt precies zoals het gemaakt moest worden. Klaar om voor de gebruikers op de markt te brengen.
Ho, wacht even! Het programma moet eerst getest worden.

De kwantiteit

Maar het programma wordt juist goed gecompileerd, dus zitten er geen fouten in. Helaas is dat niet het geval. De schoonheidsfoutjes, zoals ik eerder al opnoemde, worden niet door de compiler gezien. Meestal worden die foutjes veroorzaakt door verkeerde gebruikersinvoer.

Kunnen wij als programmeur er wat tegen doen? Jazeker, we moeten ervoor zorgen dat de code strenger werkt en niet te soepel in elkaar zit. Elke keer als de code een actie onderneemt, moet het resultaat ervan gecontroleerd worden. Net zo goed dat iemand liever niet ongekleed door het huis loopt, vanwege dat die het koud heeft.

Als we de resultaten die ontstaan uit acties willen controleren, kunnen we de volgende kwantiteit-foutjes vinden:

- er is tekst ingevoerd, maar het moesten cijfers zijn;
- rekenkundige problemen, zoals het tweede cijfer is per ongeluk een 0 in een deling;
- de invoer van het telefoonnummer is niet gelijk aan 10 cijfers;
- de dialoogformulieren, zoals het NAW invoerformulier, zijn niet *modaal*;
- de opmaak ziet er niet goed uit (het programma kan er slordig uitzien, ook al werkt dit naar behoren).

En zo kunnen we wel doorgaan, tot we zeker weten dat we alles goed getest hebben. Het testen van de programma's is voor programmeurs erg lastig. Vooral in deze tijd, want met de handige IDE hebben we snel een programma in elkaar gezet. Daardoor zal de kwantiteit meer worden en kost het testen van programma's meer tijd. Vroeger, toen we nog geen IDE hadden, was dit echter andersom.

Bèta versies

Dit is ook de verklaring waarom er zoveel *bèta versies* bestaan. Een bèta versie is een programma dat normaal werkt, de kwaliteit goed is, maar waarvan de kwantiteit slecht is.
Bèta programma's zijn meestal gratis zodat de gebruiker het programma kan proberen. De maker van het programma vraagt dan ook om reacties van de gebruiker, zodat de maker weet wat er verder gebeuren moet.

Onze eigen code

Hoe kunnen we een dergelijk foutje vinden? De compiler ziet zelf niet wanneer er per ongeluk door 0 wordt gedeeld.
We gaan dat eens bekijken. Er zijn twee variabelen: A en B die de waarden hebben uit twee invoervakken.

```
Dim A As String, B As String
A = InvoerA.Text
B = InvoerB.Text
Print A / B
```

We kennen wel de BASIC taal, dus u ziet ook dat die code niet zal werken. We kunnen ook niet A en B declareren als Integers, want de Text eigenschap geeft een string terug. Om toch te kunnen delen, moeten we de invoer converteren naar getalwaarden.

```
Dim A As Integer, B As Integer
A = Val(InvoerA.Text)
B = Val(InvoerB.Text)
Print A / B
```

Het lijkt erop dat de code nu goed werkt. Als we dit in een formulier plaatsen met twee invoervakken, zal de kwaliteit goed zijn en lijkt de kwantiteit ook geen problemen te hebben.

Toetsen we in het InvoerA vak het getal 25 in en in het InvoerB vak het getal 5 dan zal in het Debug venster, of gewoon op het formulier als u een ander BASIC dialect gebruikt, het getal 5 verschijnen. Ook de kwantiteit lijkt in orde te zijn, maar voer in InvoerB nu eens een 0 of een letter in. Nu verschijnt er een foutmelding: Division by zero.
Het lijkt erop dat plotseling het programma niet meer goed werkt. We moeten nu niet meer naar de kwaliteit kijken, maar naar de kwantiteit. Met andere woorden, er zit een schoonheidsfoutje in de code.

Ook is het vreemd dat we die fout krijgen als we iets anders dan een cijfer invoeren, zoals een letter. Helemaal is het apart dat de fout niet verschijnt als we in het eerste invoervak een letter invoeren. Hoe komt dat?

De veroorzaker is de *Val* functie die een alfanumerieke waarde moet converteren naar een numerieke waarde. Dat gebeurt ook wel, maar de functie heeft een probleem:

Zodra er in de invoer iets anders staat dan een cijfer geeft de functie een 0 terug.

De Val functie kan ook drijvende komma getallen uit de invoervakken teruggeven. U moet dan wel een punt gebruiken tijdens de invoer.

```
Val("5") = 5
Val("7.8") = 7,8
Val("7,8") = 7
Val("A") = 0
```

Onthoud, als u al gevorderd bent en u kent al de converteer datatypes, merk dan eens op dat u met een `CDbl()` geen punt kunt gebruiken met getallen invoeren, zoals u hieronder het verschil ziet met `Val()`.

```
CDbl("5") = 5
CDbl("7.8") = 78              De punt verdwijnt!
CDbl("7,8") = 7,8
CDbl("A") =                   Type mismatch error!
Oplossing: CDbl(Val("A"))     Aanbevolen:
                              gebruik een If … Then statement
```

Om er voor te zorgen dat de variabelen A en B niet de waarde 0 krijgen, en ook dat we geen ‘Type mismatch error’ melding krijgen, moeten we zelf code toevoegen die dat controleert. We krijgen dan onderstaande code:

```
Dim A As Double, B As Double
If Val(InvoerA.Text) <> 0) Then A = CDbl(InvoerA.Text)
If Val(InvoerB.Text) <> 0) Then
```

```
        B = CDbl(InvoerB.Text)
        Print "Deling is: " + Str$(A / B)
    Else
        Print "Delen door 0 kan niet!"
    End If
```

U ziet al dat voor een goede kwantiteit heel wat controle code bijgekomen is. Voordat we de invoer toekennen aan de numerieke variabelen, controleren we met Val() eerst of er wel een getal is ingevoerd om een 'Type mismatch' te voorkomen.
De tweede controle doet dat ook, maar er is een Else regel bij die netjes een melding geeft als er een 0 is ingevoerd of een niet-numeriek teken.

Het is heel belangrijk de kwantiteit van het programma goed te benutten. Veel controlecode is beter dan helemaal geen controlecode. Wat voor stoms we ook intoetsen, de compiler zal het nooit en te nimmer weten. Ook de programmeur niet. Maak daarom uw eigen programma's niet te kaal. Onderschep deze foutjes!

On Error Goto *label*
Er is nog een andere manier om fouten te onderscheppen. Sommige BASIC dialecten hebben deze mogelijkheid.

Voordat er wat gedaan kan worden, zoals een berekening uitvoeren of een bestand inladen, moet er tijdens de uitvoer gecontroleerd worden of het wel allemaal goed blijft gaan. We kunnen dit doen met een *On Error Goto …* statement. Dit statement moet eerst getypt worden voordat we de actie intypen. De *label* is de verwijzing naar een regel met daaronder code die alleen uitgevoerd wordt wanneer er een fout ontstaat. On Error Goto zorgt ervoor dat er meteen naar die label gesprongen wordt.

We moeten echter niet vergeten om na de actie de error te sluiten. Dit wordt gedaan door een 0 op te geven als label, dus: On Error Goto 0.
Vergeet ook niet voor de label regel een Exit Sub of een Exit Function te gebruiken, anders zal altijd de code na de label uitgevoerd worden; ook als er helemaal geen fout is onderschept.

Onderstaande code laat een voorbeeld zien. Wat rood staat aangegeven is de kwantiteit van de subroutine. Zonder de rode regels kan het ook werken zolang er een geldige bestandsnaam is ingevoerd en of het lezen van het bestand wel goed gaat.

```
Sub Test()
    Dim Naam As String, Tekst As String
    If Len(Invoer.Text) > 0 Then
        Naam = Invoer.Text
        On Error Goto BestandsFout
        Open Naam For Input As #1 Len = 255
        Input #1, Tekst
        Close #1
        On Error Goto 0
        Print Tekst
    Else
        Print "Geen bestandsnaam ingevoerd!"
    End If
    Exit Sub
    BestandsFout:
        Print "Het bestand bestaat niet of is ongeldig!"
End Sub
```

Neem dus geen risico. Bescherm de code zodat u een goede kwantiteit hebt, want alleen maar een goede kwaliteit is niet voldoende.

Laat zoveel mogelijk berichten weergeven als er iets niet goed gaat. Ook de gebruiker zou graag willen weten waarom het bestand niet werkt, waarom er niet geprint kan worden, waarom de invoer ongeldig is, enzovoort.

Marco Kurvers

Grafisch programmeren in GW-BASIC (11).

In dit nieuwe hoofdstuk gaan we het over heel wat anders hebben dan 3D tekenen en tekenen van vlakken in de ruimte: **Turtle-graphics en LOGO-simulatie**.

LOGO is vooral door de turtle-graphics beroemd geworden. Er is geen andere programmeertaal waarmee zo eenvoudig zeer moeilijke grafische figuren gemaakt kunnen worden dan met LOGO. In dit hoofdstuk zullen we vijf LOGO programma's in BASIC vertalen. Deze BASIC programma's bevatten wel 25 tot 30 regels, terwijl LOGO hiervoor slechts 3 of 4 regels nodig heeft.

"Eerst LOGO, daarna andere programmeertalen"

is het motto van vele informatici en pedagogen die zich met de invoering van de informatica op scholen bezighouden.

Wanneer men deze stelling wil begrijpen, moet men niet alleen de taal LOGO en haar mogelijkheden, maar ook de omgeving waarin LOGO ontwikkeld werd goed kennen. Een korte historische terugblik lijkt daarom op zijn plaats.

Het schrijven van computerprogramma's is een intellectuele prestatie en een creatief proces. Aan de wijze waarop wij programma's ontwikkelen, herkennen wij direct onze manier van denken. Aan het Massachusetts Institute of Technology (MIT) heeft men al in het begin van de zestiger jaren een speciale programmeertaal ontwikkeld, met behulp waarvan men kunstmatige intelligentie ging bestuderen. Men noemde deze taal LISP, een afkorting van LISt-Processing.

In LISP werden programma's geschreven die een met kunstmatige zintuigen uitgeruste elektromechanische muis in staat stelden een uitweg uit ieder willekeurig doolhof te vinden. In LISP worden programma's ontwikkeld die de computer tot een medespeler met leervermogen maakt. Hoe meer spelletjes de computer tegen een menselijke tegenspeler speelt, des te beter wordt hij. Goede zetten van de tegenstander onthoudt hij en legt hij vast in zijn geheugen, terwijl hij het eigen slechte zetten voor toekomstige spelletjes uit zijn geheugen wegveegt. Een 'afvalproduct ' van deze studies aan het MIT zijn de moderne schaakprogramma's.

LOGO is een hoog ontwikkeld dialect van de LISP-taal. Seymour Papert, leerling van de bekende onderzoeker Jean Piaget en medewerker aan het MIT, heeft in twaalf jaar tijd LOGO ontwikkeld. Jean Piaget heeft in zijn bekende boek *Hoe kinderen leren* de kinderlijke denkstructuren laten zien. Hieruit blijkt dat tekenen en knutselen tot de eerste creatieve handelingen van kinderen behoren. Wij zullen zien hoe LOGO deze bezigheden ondersteunt.

Amerikanen gaan door voor een volk dat graag experimenteert. Het was hun echter duidelijk, dat jonge kinderen, wij denken dan aan zes- tot dertienjarigen, niet in staat zijn een programmeertaal zoals BASIC, laat staan Pascal, te leren en algoritmen voor numerieke, niet-numerieke of grafische problemen te schrijven. Daarom heeft men een 'kinderlijke' taal gemaakt (waarachter echter een imponerende software schuilgaat), waarmee het kind met gemak tekeningen kan maken.

Wordt het LOGO-systeem ingeschakeld, vroeger meestal met een 64K-microcomputer, met bijbehorende software, dan verschijnt in het midden van het beeldscherm een kleine driehoek, waarvan de top naar boven wijst. De kinderen noemen die driehoek 'turtle', het Engelse woord voor schildpad. Met behulp van eenvoudige bevelen kan het kind de schildpad willekeurig over het beeldscherm laten rondlopen. En al naar gelang het wenselijk is laat de turtle wel of geen spoor na.

De volgende LOGO opdrachten zijn beschikbaar:

FORWARD	100	=	100 passen vooruit
BACK	50	=	50 passen achteruit
RIGHT	90	=	draaiing van 90^0 met de klok mee
LEFT	45	=	draaiing van 45^0 tegen de klok in
PENUP		=	haal pen van papier
PENDOWN		=	zet pen op papier
HIDETURTLE		=	haal Turtle van het beeldscherm

Normaal gesproken geldt de toestand "PENDOWN". Als we bijvoorbeeld de hoofdletter F op het scherm willen tekenen, kan dat in LOGO als volgt:

```
FORWARD 100     RIGHT 90    FORWARD 50
RIGHT 90    PENUP       FORWARD 50      PENDOWN
RIGHT 90    FORWARD 50
HIDETURTLE
```

Voor 100, 50, 90 en 45 kunnen ook andere waarden gekozen worden.

Omdat LOGO een vertolkend systeem is, wordt elke opdracht (na het geven van RETURN) direct uitgevoerd. Zo op het oog lijkt het slechts leuk speelgoed! Laten we nu eens een LOGO programma bekijken; liever spreken we van een procedure:

```
TO VIERKANT
REPEAT 4 (FORWARD 75        RIGHT 90)
END
```

Het LOGO systeem weet dat tussen TO en END een procedure (een stuk programma) gedefinieerd wordt. De procedure tussen TO en END (in ons voorbeeld VIERKANT genoemd) maken we zelf. Zouden we bovenstaande procedure intoetsen, dan zal op het beeldscherm een vierkant met zijden ter lengte 75 getekend worden. De programmeurs onder u kennen vast de opdracht REPEAT waarmee een herhalingsstructuur geprogrammeerd kan worden.

Als we de procedure VIERKANT in het LOGO systeem ingevoerd hebben, kunnen we hier ook gebruik van maken. LOGO onthoudt alle procedures die wij zelf invoeren. We kunnen dan ook later nieuwe procedures ontwikkelen waarin we gebruik maken van eerder ingevoerde procedures (zoals VIERKANT). Nu volgt een procedure met een parameter:

```
TO VIERKANT:ZIJDE
REPEAT 4 (FORWARD:ZIJDE    RIGHT 90)
END
```

Toetsen we nu VIERKANT 150 **<RETURN>** in, dan zal LOGO een vierkant met zijden van 150 tekenen. Niets verhindert u een procedure in te bedden in een volgende procedure en die weer in een volgende en die weer ….., enzovoorts. Alleen de beschikbare geheugenruimte is hierbij een remmende factor. Bekijk het volgende LOGO programma.

LOGO programma nr. 1

```
TO VIERKANTPATROON
REPEAT 8 (FORWARD 20        LEFT 45      VIERKANT 75)
END
```

Dit eenvoudige drieregelige LOGO programma tekent een patroon van acht vierkanten (zie later de afbeelding). We zullen dit programma straks in BASIC vertalen. Het zal blijken dat hiervoor enige wiskundige en programmeer-technische kennis nodig is. Het LOGO programma kan door een leerling van de lagere school gemaakt worden, terwijl het BASIC programma dat deze acht vierkanten tekent slechts door scholieren van de hoogste klassen van het voortgezet onderwijs kan worden gemaakt. Het wordt nog moeilijker als we intikken:

```
TO STROOK:AANTAL
REPEAT:AANTAL (FORWARD 100      VIERKANTPATROON)
END
```

Tikken we vervolgens in STROOK 5 **<RETURN>** dan maken we heel eenvoudig een fraai ornament. Probeer dit maar eens in BASIC of Pascal. LOGO wordt pas echt interessant als we van de mogelijkheid gebruik maken dat een procedure zichzelf aanroept (recursiviteit). De turtle-graphics berusten op het principe dat we procedure parameters geven en dat deze procedure zichzelf steeds met andere parameterwaarden aanroept. Bekijk de volgende procedure:

```
TO VEELHOEK:ZIJDE:HOEK
FORWARD:ZIJDE  LEFT:HOEK
VEELHOEK:ZIJDE:HOEK          ← hier roept de procedure zichzelf aan
END
```

Als we nu VEELHOEK 200 144 intikken, zal het LOGO systeem een regelmatige vijfpuntige ster tekenen. We zullen echter op de STOP toets moeten drukken (vergelijkbaar met de Escape toets of het venster sluiten) om het tekenen te stoppen. Brengen we in deze procedure een stopmechanisme aan en laten we de procedure steeds de waarde van ZIJDE veranderen, dan krijgen we de welhaast bekendste LOGO procedure:

LOGO programma nr. 2

```
TO TURTLE:ZIJDE:GROEI:HOEK
FORWARD:ZIJDE  LEFT:HOEK
MAKE:ZIJDE:ZIJDE+:GROEI
IF:ZIJDE>200 THEN STOP
TURTLE:ZIJDE:GROEI:HOEK
END
```

Als er een dubbelepunt voor een LOGO woord staat, weet het LOGO systeem dat het om een naam van een variabele gaat en niet om de naam van een procedure of een LOGO opdracht. In de bovenstaande LOGO procedure zien we hoe de procedure zichzelf steeds met een nieuwe waarde voor ZIJDE aanroept. Ook dit programma vertalen we in BASIC. Hiermee kunt u elke denkbare rechtlijnige turtle grafiek maken. U hoeft alleen de waarden van de invoerparameters (ZIJDE, GROEI en HOEK) te veranderen.

Het derde LOGO programma tekent een reeks vierkanten in spiraalvorm. Het zijn bekende figuren in het ‘land van de computergraphics’.

LOGO programma nr. 3

```
TO VIERKANTSPIRAAL:ZIJDE:HOEK
IF:ZIJDE>150 THEN STOP
VIERKANT:ZIJDE   LEFT:HOEK
VIERKANTSPIRAAL:ZIJDE+4:HOEK
END
```

Ook dit LOGO programma vertalen we in BASIC. Met deze drie programma’s hebt u niet alleen met LOGO maar ook met de ‘turtle-graphics’ kennis gemaakt.

Natuurlijk zijn we in LOGO niet gebonden aan ‘rechte lijnen’. In LOGO hebben we de beschikking over alle wiskundige functies en bewerkingen. Alle programma’s uit het boek, waar het ‘Grafisch programmeren in GW-BASIC’ vandaan komt, kunnen in LOGO geschreven worden. Zij zouden dan vast korter, overzichtelijker en begrijpelijker geweest zijn. In LOGO kunnen we heel eenvoudig met lijsten en tabellen werken. In LOGO hoeven we de variabelen niet te declareren. Achter een naam in LOGO kan gewoon een getal, een string, een n-dimensionaal veld en nog veel meer schuil gaan. Ook kan het de naam van een procedure zijn. We hoeven ons niet te bekommeren om het reserveren van geheugenruimte (DIM opdracht in BASIC), noch om het aangeven van het type (real, integer, character, boolean, enz.); het LOGO systeem regelt dit zelf.

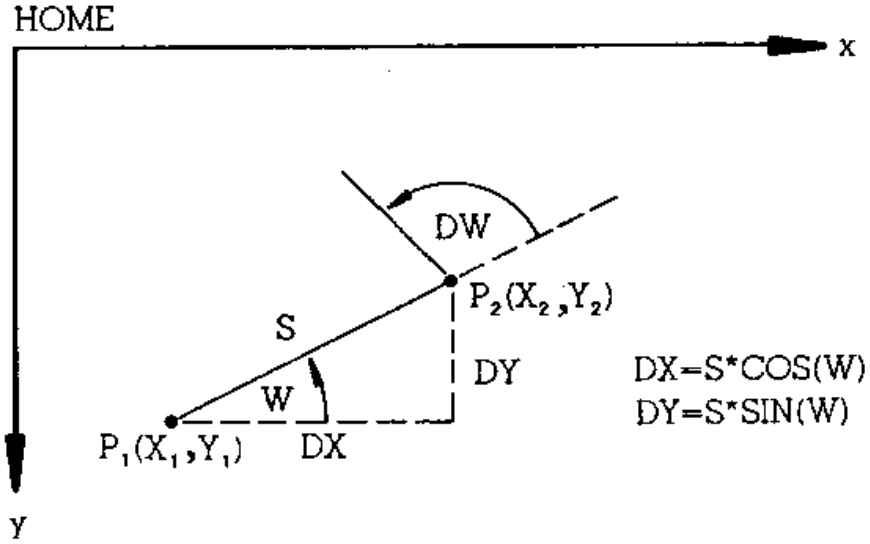
Wie in LOGO programmeert leert automatisch netjes programmeren. Begrippen als procedure, recursief, herhaling, toekenning, worden op een natuurlijke manier aangeleerd. Met een paar opdrachten is een kind al in staat ingewikkelde figuren te tekenen. Met LOGO degraderen we de computer niet tot een programmeerbare zakrekenmachine, maar halen we er alles uit wat erin zit.

Vertaling LOGO programma nr. 1 in BASIC

De kern van elk LOGO-graphics programma wordt gevormd door twee opdrachten:

FORWARD (respectievelijk BACK)	:ZIJDE
LEFT (respectievelijk RIGHT)	:HOEK

Hoe vertalen we deze opdrachten in BASIC? Bekijk hiertoe de volgende figuur.

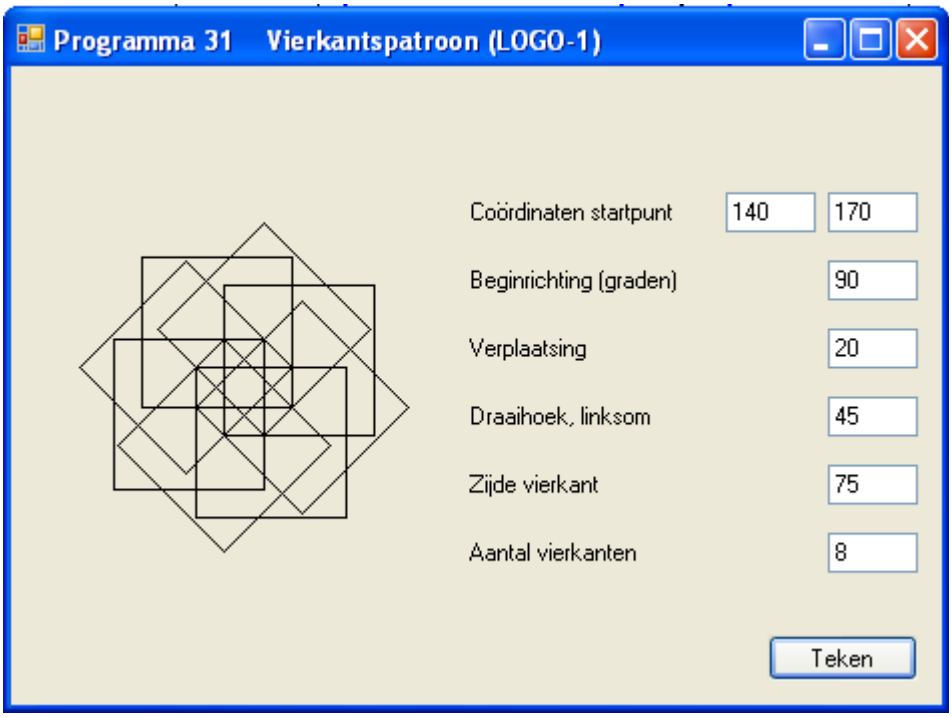


De pen staat in punt $P_1(X_1,Y_1)$ en moet in de richting van hoek W over een afstand S verplaatst worden. In $P_2(X_2,Y_2)$ aangekomen moet de PEN (eigenlijk de turtle) een hoek van DW^0 linksom maken.
De volgende BASIC opdrachten voeren dit uit:

```
INPUT "X1,Y1,W"; X1,Y1,W
H=0.5 : RD=π/180 : W1=W*RD
X2=INT(X1+S*COS(W1)+H)
Y2=INT(Y1-S*SIN(W1)+H)
LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
X1=X2 : Y1=Y2
W=W+DW : IF W>=360 THEN W=W-360
W1=W*RD
```

Deze opdrachten komt u in alle volgende BASIC programma's tegen.
Meer theorie is niet nodig! De programma's moeten met bovenstaande informatie gelezen kunnen worden.

Experimenteer met programma 31 LOGO-1 vierkantspatroon. Met W=90^0, S1=20, DW=45, S2=75 en N=8 krijgen we de onderstaande tekening.



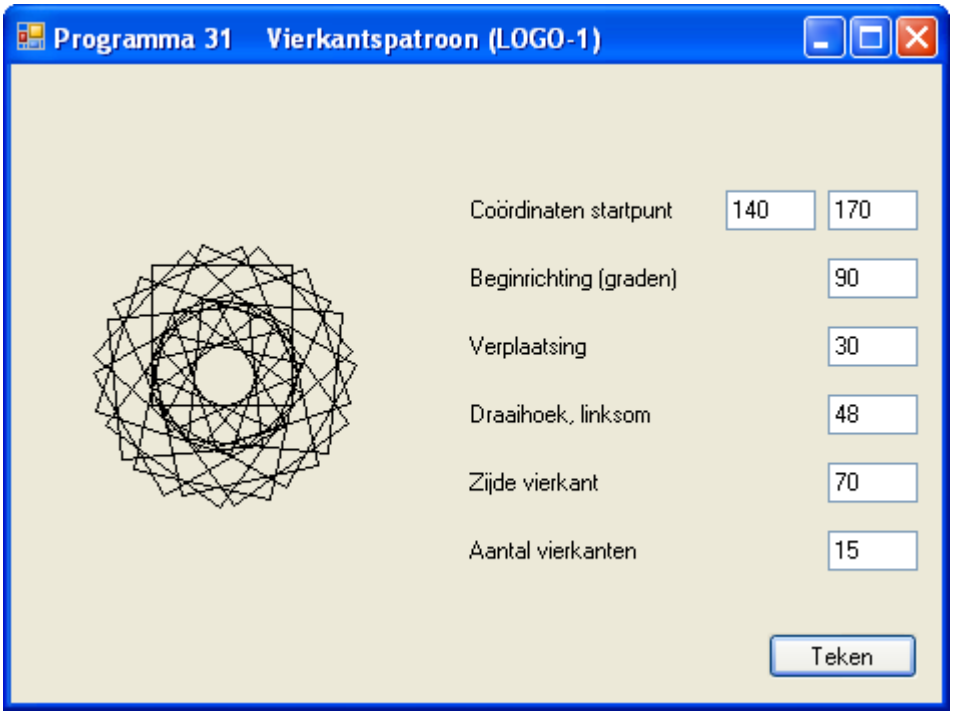

```
100 ' programma 31   VIERKANTSPATROON(LOGO-1)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "COORDINATEN STARTPUNT "; X1,Y1
150 INPUT "BEGINRICHTING (GRADEN)"; W
160 INPUT "VERPLAATSING          "; S1
170 INPUT "DRAAIHOEK,LINKSOM     "; DW
180 INPUT "ZIJDE VIERKANT        "; S2
190 INPUT "AANTAL VIERKANTEN     "; N
200 PI=4*ATN(1)
210 H=.5: RD=4*ATN(1)/180: W1=W*RD
220 CLS
230 FOR J=1 TO N
240    X2=INT(X1+S1*COS(W1)+H)
250    Y2=INT(Y1-S1*SIN(W1)+H)
260    LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
270    X1=X2: Y1=Y2: AX=X1: AY=Y1
280    W=W+DW : IF W >= 360 THEN W=W-360
```

```
290     W1=W*RD
300     FOR K=0 TO 2
310        X2=INT(X1+S2*COS(W1+K/2*PI)+H)
320        Y2=INT(Y1-S2*SIN(W1+K/2*PI)+H)
330        LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
340        X1=X2: Y1=Y2
350     NEXT K
360     LINE (FNX(X1),Y1) - (FNX(AX),AY),1
370     X1=AX: Y1=AY
380 NEXT J
390 A$=INKEY$: IF A$="" THEN 390
400 CLS: KEY ON: END
```

```
Public Class frmProg31
    Private Const H As Double = 0.5

    Private Sub btnTeken_Click(…, …) Handles btnTeken.Click
        Refresh()
    End Sub

    Private Sub frmProg31_Paint(…, ByVal e As …PaintEventArgs) …
        If txtX1.Text() <> "" And txtY1.Text() <> "" And _
               txtW.Text() <> "" And _
               txtS1.Text() <> "" And txtDW.Text() <> "" And _
               txtS2.Text() <> "" And txtN.Text() <> "" Then
            Dim X1 As Integer = CInt(txtX1.Text())
            Dim Y1 As Integer = CInt(txtY1.Text())
            Dim W As Integer = CInt(txtW.Text())
            Dim S1 As Integer = CInt(txtS1.Text())
            Dim DW As Integer = CInt(txtDW.Text())
            Dim S2 As Integer = CInt(txtS2.Text())
            Dim N As Integer = CInt(txtN.Text())
            Dim RD As Double = Math.PI / 180
            Dim W1 As Double = W * RD
            For J As Integer = 1 To N
                Dim X2 As Integer = Int(X1 + S1 * Math.Cos(W1) + H)
                Dim Y2 As Integer = Int(Y1 - S1 * Math.Sin(W1) + H)
                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                X1 = X2 : Y1 = Y2
                Dim AX As Integer = X1, AY As Integer = Y1
                W += DW
                If W >= 360 Then W -= 360
                W1 = W * RD
                For K As Integer = 0 To 2
                    X2 = Int(X1 + S2 * Math.Cos(W1 + K / 2 * Math.PI) + H)
                    Y2 = Int(Y1 - S2 * Math.Sin(W1 + K / 2 * Math.PI) + H)
                    e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                    X1 = X2 : Y1 = Y2
                Next
                e.Graphics.DrawLine(Pens.Black, X1, Y1, AX, AY)
                X1 = AX : Y1 = AY
            Next
        End If
    End Sub
End Class
```

W=90, S1=30, DW=48, S2=70 en N=15 geeft de onderstaande tekening.

Als DW een deler is van 360, is N gelijk aan 360/DW. Kiest u voor DW een willekeurige waarde, neem dan voor N een groot getal, bijvoorbeeld 100, en breek het tekenen met de stoptoets (of normaal gesproken met de Escape toets of door te klikken op het kruisje) af. De ‘kinderen’ doen dat in LOGO ook op deze manier.



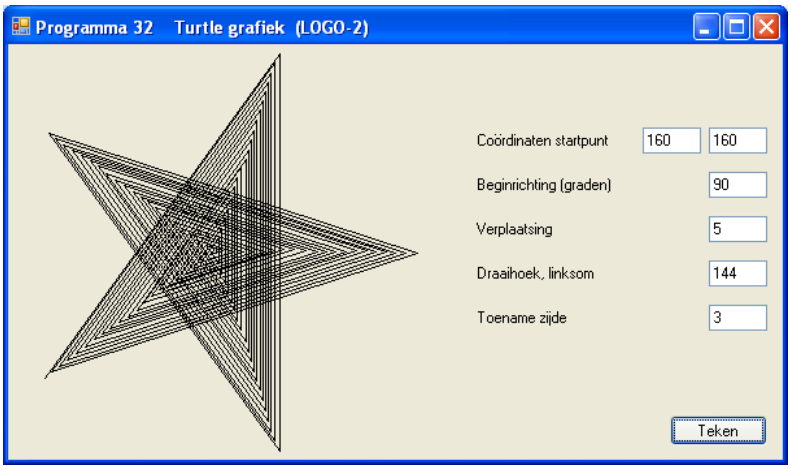
Vertaling LOGO programma nr. 2 in BASIC

Het programma lijkt sterk op het vorige en behoeft daarom geen commentaar.

```
100 '    programma 32   TURTLE-GRAFIEK(LOGO-2)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "COORDINATEN STARTPUNT "; X1,Y1
150 INPUT "BEGINRICHTING (GRADEN)"; W
160 INPUT "VERPLAATSING          "; S
170 INPUT "DRAAIHOEK,LINKSOM     "; DW
180 INPUT "TOENAME ZIJDE         "; DS
190 H=.5
200 RD=4*ATN(1)/180 : W1=W*RD
210 CLS : LOCATE 11,1
212 PRINT USING "W : ####"; W
214 PRINT USING "S : ####"; S
216 PRINT USING "DW: ####"; DW
218 PRINT USING "DS: ####"; DS
220 X2=INT(X1+S*COS(W1)+H)
230 Y2=INT(Y1-S*SIN(W1)+H)
240 WHILE X2>=0 AND X2<=320 AND Y2>=0 AND Y2<=320
250      LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
260      X1=X2: Y1=Y2
270      W=W+DW : IF W >= 360 THEN W=W-360
280      W1=W*RD: S=S+DS
290      X2=INT(X1+S*COS(W1)+H)
```

```
300       Y2=INT(Y1-S*SIN(W1)+H)
310 WEND
320 A$=INKEY$: IF A$="" THEN 320
330 CLS: KEY ON: END
```

Hieronder volgen voorbeelden uit dit programma. Het laatste voorbeeld is echter niet het voorbeeld zoals het zou moeten wezen. Volgens het boek had de waarde van DS=0 moeten zijn. Helaas bleef het programma hangen. Ik heb daarom maar waarde 1 gekozen. Kunt u wel achterhalen waarom waarde 0 niet werkt? Probeer het eens uit. Ik zal u wel de originele tekening laten zien na deze voorbeelden.



Voorbeeld A

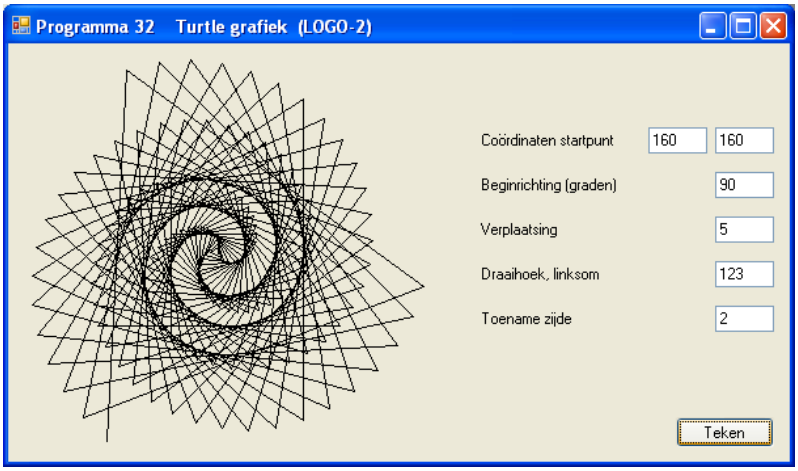
X1 = 160, Y1 = 160

W = 90

S = 5

DW = 144

DS = 3



Voorbeeld B

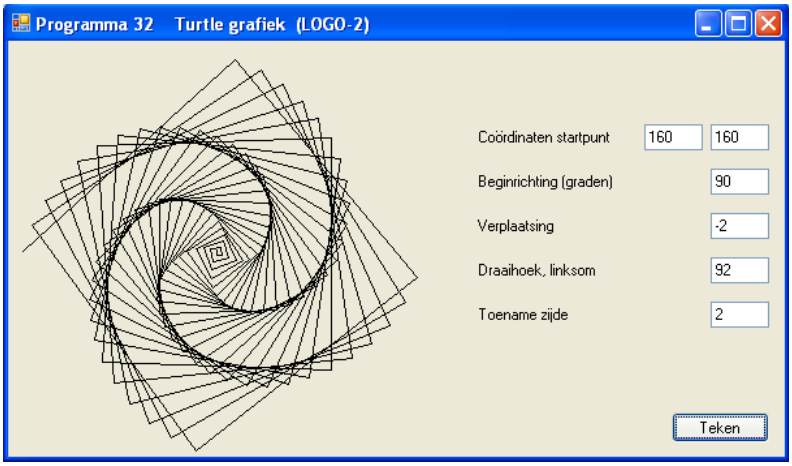
X1 = 160, Y1 = 160

W = 90

S = 5

DW = 123

DS = 2



Voorbeeld C

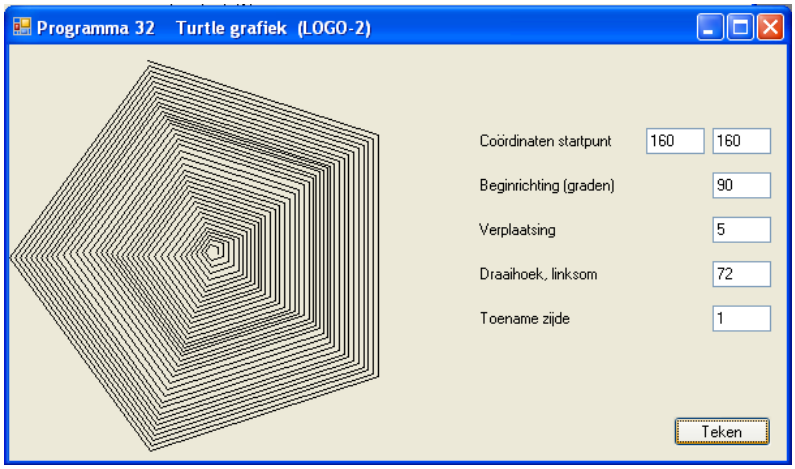
X1 = 160, Y1 = 160

W = 90

S = -2

DW = 92

DS = 2



Voorbeeld D

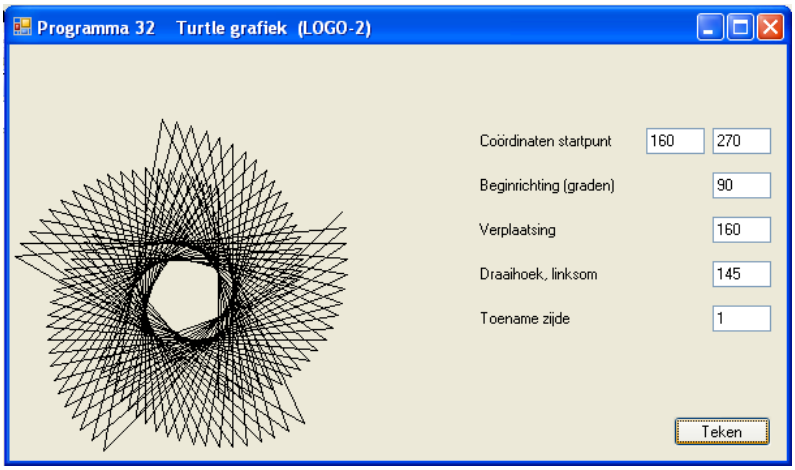
X1 = 160, Y1 = 160

W = 90

S = 5

DW = 72

DS = 1



Voorbeeld E

X1 = 160, Y1 = 270

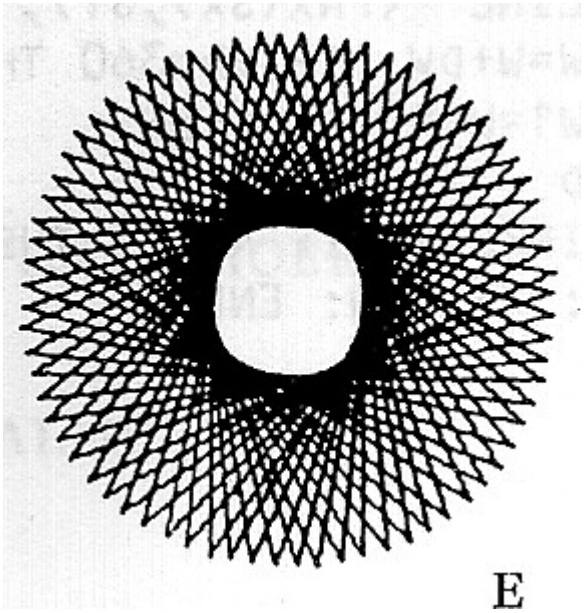
W = 90

S = 160

DW = 145

DS = 1

Hoe Voorbeeld E eigenlijk had moeten zijn met DS = 0, ziet u hier.



X1 = 160, Y1 = 270

W = 90

S = 160

DW = 145

DS = 0

Maar zoals ik al zei: helaas lukte het niet om het voorbeeld precies zo te krijgen.

Hieronder ziet u nog de vertaling van het programma in Visual Basic 2008.

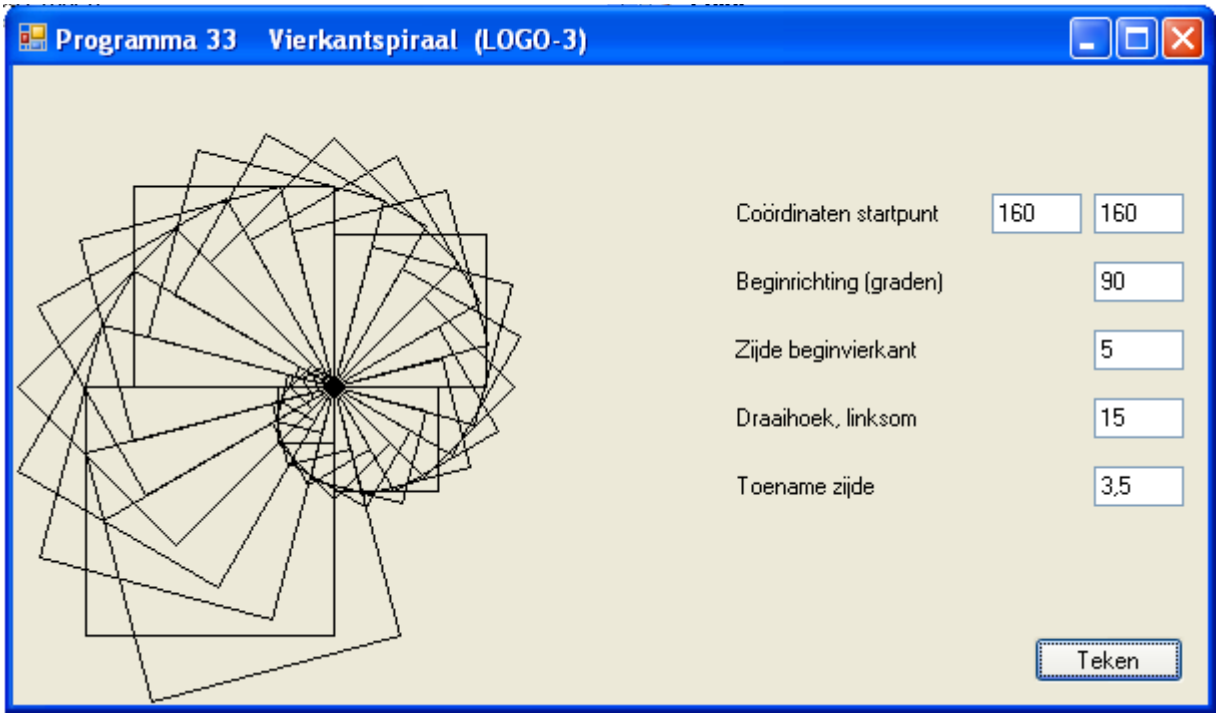
```
Public Class frmProg32
    Private Const H As Double = 0.5

    Private Sub btnTeken_Click(…, …) …
        Refresh()
    End Sub
```

```
    Private Sub frmProg32_Paint(…, ByVal e As …PaintEventArgs) …
        If txtX1.Text() <> "" And txtY1.Text() <> "" And _
               txtW.Text() <> "" And _
               txtS.Text() <> "" And txtDW.Text() <> "" And _
               txtDS.Text() <> "" Then
            Dim X1 As Integer = CInt(txtX1.Text())
            Dim Y1 As Integer = CInt(txtY1.Text())
            Dim W As Integer = CInt(txtW.Text())
            Dim S As Integer = CInt(txtS.Text())
            Dim DW As Integer = CInt(txtDW.Text())
            Dim DS As Integer = CInt(txtDS.Text())
            Dim RD As Double = Math.PI / 180
            Dim W1 As Double = W * RD
            Dim X2 As Integer = Int(X1 + S * Math.Cos(W1) + H)
            Dim Y2 As Integer = Int(Y1 - S * Math.Sin(W1) + H)
            While X2 >= 0 And X2 <= 320 And Y2 >= 0 And Y2 <= 320
                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                X1 = X2 : Y1 = Y2
                W += DW
                If W >= 360 Then W -= 360
                W1 = W * RD : S += DS
                X2 = Int(X1 + S * Math.Cos(W1) + H)
                Y2 = Int(Y1 - S * Math.Sin(W1) + H)
            End While
        End If
    End Sub
End Class
```

Vertaling LOGO programma nr. 3 in BASIC

```
100 '  programma 33  VIERKANTSPIRAAL(LOGO-3)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "COORDINATEN STARTPUNT  "; SX,SY
150 INPUT "BEGINRICHTING (GRADEN) "; W
160 INPUT "ZIJDE BEGINVIERKANT    "; S
170 INPUT "DRAAIHOEK,LINKSOM      "; DW
180 INPUT "TOENAME ZIJDE          "; DS
190 H=.5: PI=4*ATN(1): RD=PI/180: W1=W*RD
200 CLS : PSET (FNX(SX),SY),1
210 WHILE VLAG=0
220    FASE=0: X=SX: Y=SY
230    FOR I=2 TO 4
240       X=INT(X+S*COS(W1+FASE)+H)
250       IF X<0 OR X>320 THEN VLAG=1:GOTO 340
260       Y=INT(Y-S*SIN(W1+FASE)+H)
270       IF Y<0 OR Y>320 THEN VLAG=1:GOTO 340
280       LINE -(FNX(X),Y),1
290       FASE=FASE+PI/2
300    NEXT I
310    LINE -(FNX(SX),SY), 1
320    W=W+DW: IF W>=360 THEN W=W-360
```

```
330    W1=W*RD: S=S+DS
340 WEND
350 A$=INKEY$: IF A$="" THEN 350
360 CLS: KEY ON: END
```



Om bovenstaand programma vertaald in Visual Basic 2008 op een Windows formulier te krijgen, viel niet mee. Het bleek dat enige aanpassingen in de code vereist waren om de spiraal te kunnen tekenen. Er is bijvoorbeeld een nadeel dat het *Graphics* object geen tekenmethode kent om vanuit een voorgaande lijn verder te tekenen: *Line –(x1,y1),1*, zodat ik een oplossing moest vinden. Kijk eens naar het VB programma en zie de aanpassingen die ik moest maken.

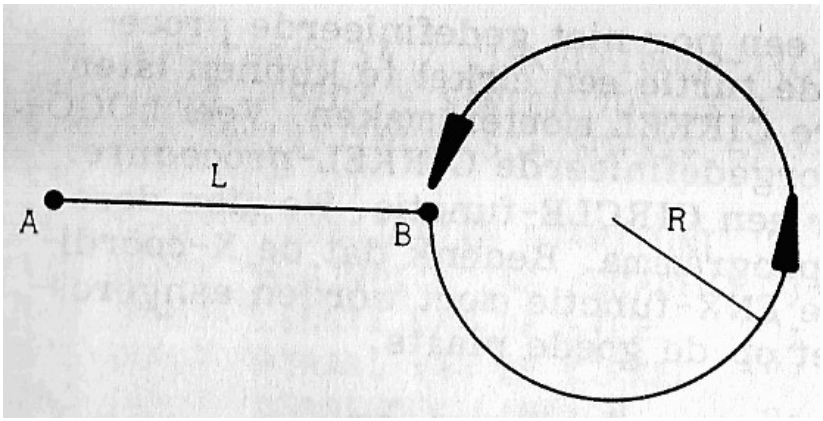
```
Public Class frmProg33
    Private Const H As Double = 0.5

    Private Sub btnTeken_Click(…, …) …
        Refresh()
    End Sub

    Private Sub frmProg33_Paint(…, ByVal e As …PaintEventArgs) …
        If txtSX.Text() <> "" And txtSY.Text() <> "" And _
               txtW.Text() <> "" And _
               txtS.Text() <> "" And txtDW.Text() <> "" And _
               txtDS.Text() <> "" Then
            Dim SX As Integer = CInt(txtSX.Text())
            Dim SY As Integer = CInt(txtSY.Text())
            Dim W As Integer = CDbl(txtW.Text())
            Dim S As Integer = CInt(txtS.Text())
            Dim DW As Integer = CDbl(txtDW.Text())
            Dim DS As Double = CDbl(txtDS.Text())
            Dim RD As Double = Math.PI / 180
            Dim W1 As Double = W * RD
            Dim Vlag As Boolean = False
            Dim X, Y, XN, YN As Integer
            e.Graphics.DrawLine(Pens.Black, SX, SY, SX, SY)
```

```
            While Not Vlag
                Dim Fase As Double = 0
                X = SX : Y = SY
                For I As Byte = 2 To 4
                    XN = Int(X + S * Math.Cos(W1 + Fase) + H)
                    If XN < 0 Or XN > 320 Then
                        Vlag = True
                        Exit For
                    End If
                    YN = Int(Y - S * Math.Sin(W1 + Fase) + H)
                    If YN < 0 Or YN > 320 Then
                        Vlag = True
                        Exit For
                    End If
                    e.Graphics.DrawLine(Pens.Black, X, Y, XN, YN)
                    Fase += Math.PI / 2
                    X = XN : Y = YN
                Next
                If Not Vlag Then
                    e.Graphics.DrawLine(Pens.Black, XN, YN, SX, SY)
                    W += DW : If W >= 360 Then W -= 360
                    W1 = W * RD : S += DS
                End If
            End While
        End If
    End Sub
End Class
```

Vergelijk deze drie programma's eens met de overeenkomstige LOGO programma's. In BASIC moeten we zelf alle graphic-software schrijven, terwijl deze in LOGO als machineroutines aanwezig is.

Het 4de en 5de LOGO programma

Tot slot van dit hoofdstuk geven we nog twee BASIC programma's waarmee de turtle behalve rechte wegen ook kromme wegen kan bewandelen. Dit voegt een nieuw element aan de turtle-graphics toe. De LOGO structuur zullen we stapsgewijs verfijnen. Als uitgangspunt nemen we de onderstaande figuur. Deze bestaat uit een lijnstuk L, met daaraan vast een cirkel met straal R. We noemen deze figuur voor het gemak even 'de steelpan'.



Als de turtle de steelpan tekent, begint hij in A. Hij legt vervolgens een afstand L in positieve x-richting af en komt in B. Hier draait hij 90^0 met de klok mee en legt daarna de omtrek van de cirkel af tot hij weer in B uitkomt. Daar draait hij weer 90^0 met de klok mee en kijkt dan recht vooruit naar het punt A. Met deze steelpan kunnen we twee soorten tekeningen maken. Laten we deze mogelijkheden bekijken:

LOGO programma 4

```
TO CIRKELFIGUUR-1:LIJNSTUK:STRAAL:HOEK
STEELPAN:LIJNSTUK:STRAAL
LEFT:HOEK
CIRKELFIGUUR-1:LIJNSTUK:STRAAL:HOEK
END
```

LOGO programma 5

```
TO CIRKELFIGUUR-2:LIJNSTUK:STRAAL:HOEK
REPEAT 4 (STEELPAN:LIJNSTUK:STRAAL   LEFT 90)
LEFT:HOEK
CIRKELFIGUUR-2:LIJNSTUK:STRAAL:HOEK
END
```

Deze twee LOGO programma's kunnen nog niet uitgevoerd worden, want de procedure STEELPAN is nog niet in het LOGO systeem gedefinieerd. We gaan de structuur verfijnen:

```
TO STEELPAN:LIJNSTUK:STRAAL
FORWARD:LIJNSTUK     RIGHT 90
CIRKEL:STRAAL
RIGHT 90
END
```

Ook in deze procedure zien we een nog niet gedefinieerde procedure, CIRKEL, opduiken. Om de turtle een cirkel te kunnen laten maken zullen we deze procedure CIRKEL moeten maken. Veel LOGO versies bevatten al zo'n voorgedefinieerde CIRKEL procedure. BASIC beschikt ook over een CIRCLE procedure. We zien deze in regel 280 van het volgende programma. Bedenk dat de X-coördinaat van het middelpunt met de FNX functie moet worden aangeroepen, anders komt de cirkel niet op de goede plaats. We weten al dat dit tegenwoordig in BASIC niet meer van toepassing is en we de FNX functie achterwege kunnen laten.

```
CIRCLE (FNX(U),V),  R,   1,   0,   360*RD,   1/1.55
            ↑       ↑    ↑    ↑       ↑         ↑
     coördinaten   straal  |   begin  eind      correctiefactor
     middelpunt            |   hoek   hoek
                           |
                gewone afdrukkleur
```

Als we echt LOGO willen nadoen, moeten de cirkels getekend worden vanuit het uiteinde van de steel (zie punt B in tekening op de vorige pagina). De CIRCLE procedure begint juist met tekenen aan de andere kant van de cirkel.

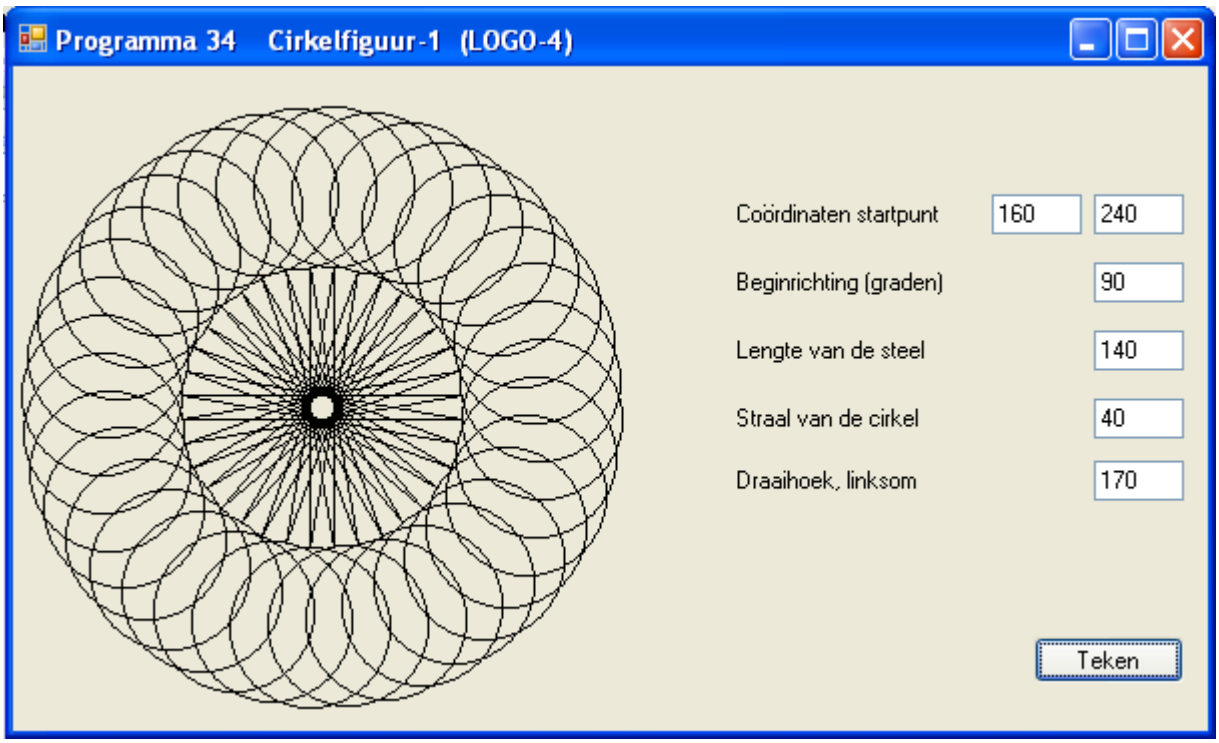
Als we het 'echt' willen doen moeten we de regels 280 t/m 300 vervangen door:

```
AX=X2 : AY=Y2
REM CIRKEL M(U,V) EN STRAAL R
U=INT(AX+R*COS(W1)+H)
V=INT(AY-R*SIN(W1)+H)
X1=AX : Y1=AY
FOR WW=(W1-PI) TO (W1+PI) STEP PI/32
   X2=INT(U+R*COS(WW)+H)
```

```
      Y2=INT(V-R*SIN(WW)+H)
      LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
      X1=X2:Y1=Y2
   NEXT WW
   W=W+DW : IF W>=360 THEN W=W-360
   W1=W*RD : X1=AX : Y1=AY
```

Vertaling van LOGO programma nr. 4 in BASIC

```
100 'programma 34   CIRKELFIGUUR-1(LOGO-4)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "COORDINATEN STARTPUNT  "; SX,SY
150 INPUT "BEGINRICHTING (GRADEN) "; W
160 INPUT "LENGTE VAN DE STEEL    "; L
170 INPUT "STRAAL VAN DE CIRKEL   "; R
180 INPUT "DRAAIHOEK,LINKSOM      "; DW
190 RD=4*ATN(1)/180: W1=W*RD: H=.5
200 CLS
210 X1=SX : Y1=SY
220 WHILE X2<>SX OR Y2<>SY
230    X2=INT(X1+L*COS(W1)+H)
240    U= INT(X2+R*COS(W1)+H)
250    Y2=INT(Y1-L*SIN(W1)+H)
260    V= INT(Y2-R*SIN(W1)+H)
270    LINE  (FNX(X1),Y1)-(FNX(X2),Y2),1
280    CIRCLE(FNX(U),V),R,1,0,360*RD,1/1.55
290    W=W+DW: IF W>=360 THEN W=W-360
300    W1=W*RD: X1=X2: Y1=Y2
310 WEND
320 A$=INKEY$: IF A$="" THEN 320
330 CLS: KEY ON: END
```



Om het programma te vertalen in Visual Basic 2008, heb ik de CIRCLE opdracht aan moeten passen. Het *Graphics* object kent geen Circle methode waardoor ik gebruik moest maken van de methode *DrawEllipse*. Het probleem is echter dat de methode niet vanuit het middelpunt, maar vanuit een boundary rectangle tekent. Er moet dus een linker bovenhoek en een rechter benedenhoek worden bepaald.

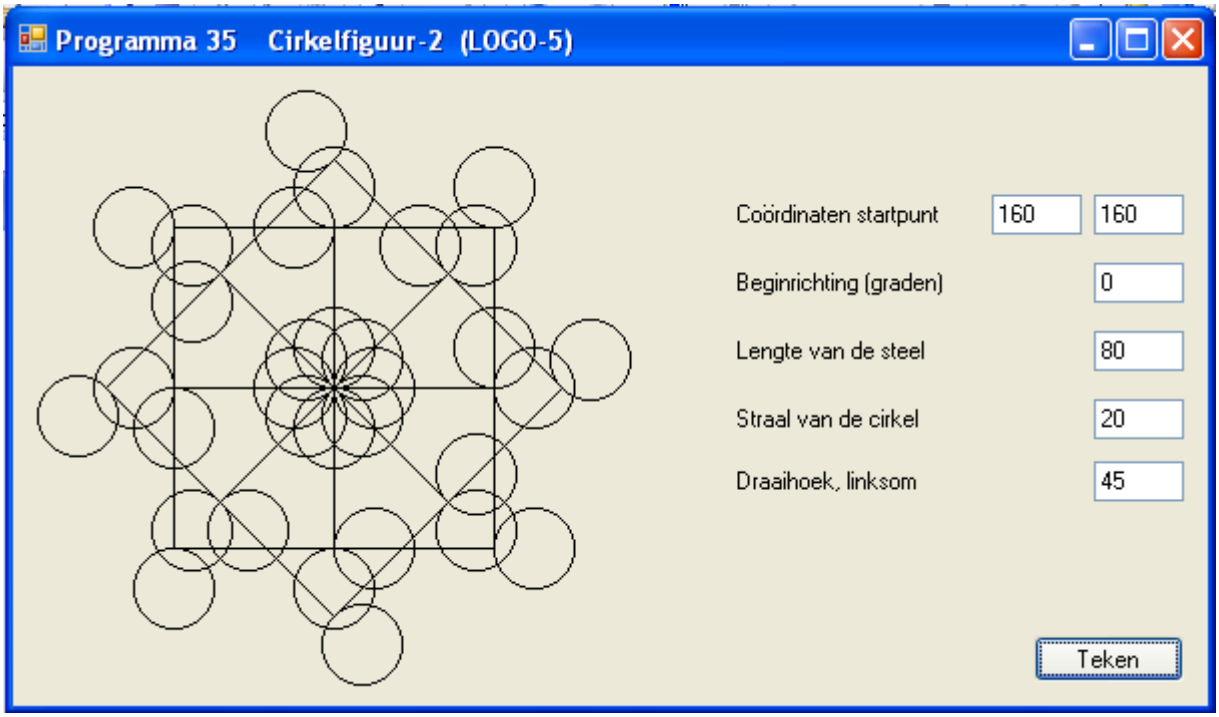
Denk eraan dat ik bij *DrawEllipse* de eerste parameter heb weggelaten. Deze staat in drie punten. Verander de parameter net als bij *DrawLine* in **Pens.Black**.
Dit heb ik gedaan om het programma op deze pagina overzichtelijk te houden. Zie het VB programma hieronder.

```
Public Class frmProg34
    Private Const H As Double = 0.5

    Private Sub btnTeken_Click(…, …) Handles btnTeken.Click
        Refresh()
    End Sub

    Private Sub frmProg34_Paint(…, …PaintEventArgs) Handles MyBase.Paint
        If txtSX.Text() <> "" And txtSY.Text() <> "" And _
               txtW.Text() <> "" And _
               txtL.Text() <> "" And txtDW.Text() <> "" And _
               txtR.Text() <> "" Then
            Dim SX As Integer = CInt(txtSX.Text())
            Dim SY As Integer = CInt(txtSY.Text())
            Dim W As Integer = CDbl(txtW.Text())
            Dim L As Integer = CInt(txtL.Text())
            Dim DW As Integer = CDbl(txtDW.Text())
            Dim R As Integer = CInt(txtR.Text())
            Dim RD As Double = Math.PI / 180
            Dim W1 As Double = W * RD
            Dim X1, Y1, X2, Y2, U, V As Integer
            X1 = SX : Y1 = SY
            While X2 <> SX Or Y2 <> SY
                X2 = Int(X1 + L * Math.Cos(W1) + H)
                U = Int(X2 + R * Math.Cos(W1) + H)
                Y2 = Int(Y1 - L * Math.Sin(W1) + H)
                V = Int(Y2 - R * Math.Sin(W1) + H)
                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                e.Graphics.DrawEllipse(…, U - R, V - R, R * 2, R * 2)
                W += DW : If W >= 360 Then W -= 360
                W1 = W * RD : X1 = X2 : Y1 = Y2
            End While
        End If
    End Sub
End Class
```

U kunt ook zien dat de beginhoek en eindhoek niet meer nodig zijn, maar in GW-BASIC moeten deze parameters wel opgegeven worden. Dit is net zo het geval bij de correctiefactor.

Vertaling LOGO programma nr. 5 in BASIC

```
100 'programma 35    CIRKELFIGUUR-2(LOGO-5)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "COORDINATEN STARTPUNT  "; X1,Y1
150 INPUT "BEGINRICHTING (GRADEN) "; W
160 INPUT "LENGTE VAN DE STEEL    "; L
170 INPUT "STRAAL VAN DE CIRKEL   "; R
180 INPUT "DRAAIHOEK,LINKSOM      "; DW
190 RD=4*ATN(1)/180: W1=W*RD: H=.5
200 CLS
210 FOR J=1 TO 4
220    X2=INT(X1+L*COS(W1)+H)
230    U =INT(X2+R*COS(W1)+H)
240    Y2=INT(Y1-L*SIN(W1)+H)
250    V =INT(Y2-R*SIN(W1)+H)
260    LINE  (FNX(X1),Y1)-(FNX(X2),Y2),1
270    CIRCLE(FNX(U),V),R,1,0,360*RD,1/1.55
280    W=W+90: IF W>=360 THEN W=W-360
290    W1=W*RD: X1=X2: Y1=Y2
300 NEXT J
310 W=W+DW: IF W>=360 THEN W=W-360
320 W1=W*RD
330 A$=INKEY$: IF A$="" THEN 210
340 CLS: KEY ON: END
```



Het bovenstaand programma tekent door totdat er op een toets gedrukt wordt. Voor Visual Basic 2008 is het beter om het tekenen niet oneindig door te laten gaan, vooral niet omdat na zoveel keren tekenen er geen cirkel meer bijkomt. Ik heb daarom in onderstaand VB programma een *While* lus gemaakt die tien keer de *For* lus start. Dat is voldoende om bovenstaande tekening op het formulier te krijgen, en hij ziet er precies zo uit als de tekening in het boek.

```
Public Class frmProg35
    Private Const H As Double = 0.5

    Private Sub btnTeken_Click(…, …) Handles btnTeken.Click
        Refresh()
    End Sub

    Private Sub frmProg35_Paint(…, …PaintEventArgs) Handles MyBase.Paint
        If txtX1.Text() <> "" And txtY1.Text() <> "" And _
               txtW.Text() <> "" And _
               txtL.Text() <> "" And txtDW.Text() <> "" And _
               txtR.Text() <> "" Then
            Dim X1 As Integer = CInt(txtX1.Text())
            Dim Y1 As Integer = CInt(txtY1.Text())
            Dim W As Integer = CDbl(txtW.Text())
            Dim L As Integer = CInt(txtL.Text())
            Dim DW As Integer = CDbl(txtDW.Text())
            Dim R As Integer = CInt(txtR.Text())
            Dim RD As Double = Math.PI / 180
            Dim W1 As Double = W * RD
            Dim X2, Y2, U, V As Integer
            Dim T As Byte = 1
            While T <= 10
                For J As Byte = 1 To 4
                    X2 = Int(X1 + L * Math.Cos(W1) + H)
                    U = Int(X2 + R * Math.Cos(W1) + H)
                    Y2 = Int(Y1 - L * Math.Sin(W1) + H)
                    V = Int(Y2 - R * Math.Sin(W1) + H)
                    e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                    e.Graphics.DrawEllipse(…, U - R, V - R, R * 2, R * 2)
                    W += 90 : If W >= 360 Then W -= 360
                    W1 = W * RD : X1 = X2 : Y1 = Y2
                Next
                W += DW : If W >= 360 Then W -= 360
                W1 = W * RD
                T += 1
            End While
        End If
    End Sub
End Class
```

Met andere waarden voor deze variabelen kunt u de mooiste turtle-plaatjes maken.

Tot nu toe waren dit allemaal voorbeeldprogramma's om een overzicht te kunnen geven hoe we grafisch kunnen tekenen. In de volgende nieuwsbrief gaan we op een hele andere manier grafisch tekenen, met educatieve toepassingsprogramma's.

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

BASIC nieuws, tips en oplossingen.

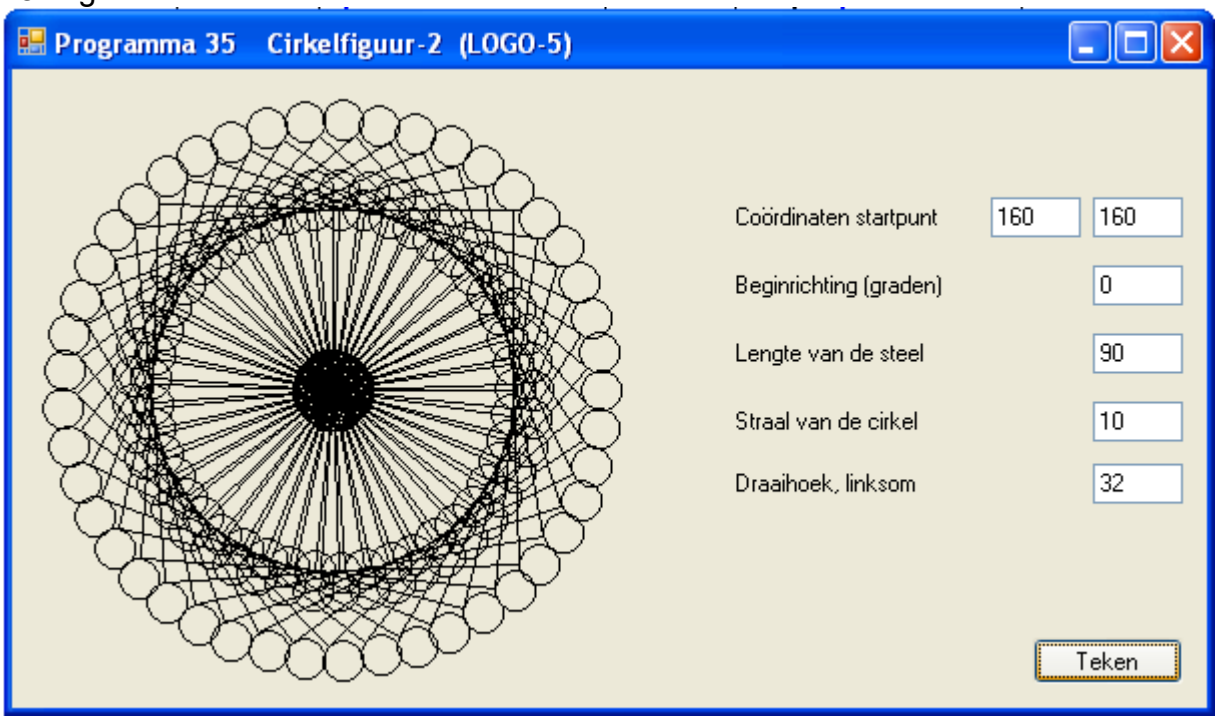
Voor BASIC nieuws, tips en oplossingen wil ik even terug naar het hoofdstuk over ‘Grafisch programmeren in GW-BASIC’.

Interessanter
De tekeningen worden steeds mooier, maar de programma’s worden steeds moeilijker. Ik heb veel tijd nodig gehad om de GW-BASIC code te vertalen naar Visual Basic 2008.

Toch worden ze interessanter, want de invoermogelijkheden worden steeds uitgebreider. Uit één programma kunnen we meerdere soorten tekeningen maken door alleen maar andere parameters op te geven.

Achtergronden en Mandala tekeningen
De tekeningen krijgen ook steeds meer een textuur-achtig effect. Dit geeft het voordeel dat we ze kunnen gebruiken als achtergronden voor formulieren en webpagina’s. De texturen beginnen ook een Mandala karakter te krijgen. Zulke tekeningen hebben een effect op een voorwerp. Een voorbeeld van een Mandala tekening is een effect met allemaal vierkanten, maar de volledige tekening kan eruit zien alsof het lijkt dat het getekend is met cirkels.

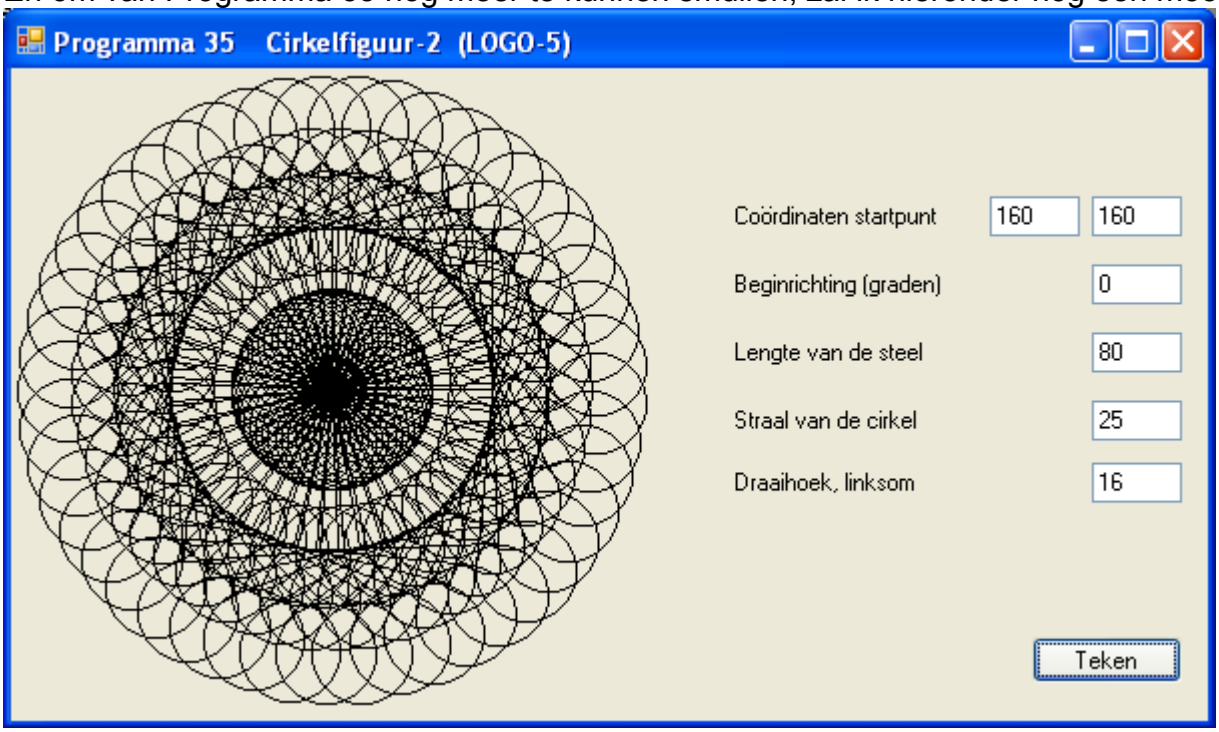
Maar ik heb ook uit Programma 35 van het hoofdstuk *Grafisch programmeren in GW-BASIC* een leuke tip. U kunt veel experimenteren met verschillende parameters, maar bekijk eens onderstaande tekening.



Niet alleen met de opgegeven waarden heb ik deze voor elkaar gekregen, maar het viel mij op dat ook het *While* statement een grote rol speelt. Door een ruime waarde te geven voor variabele T kan elke invoer voor mooie tekeningen zorgen.
Verander in Programma 35 het *While* statement in: `While Not T <= 64`.
Als u kleinere waarden voor variabele T geeft, is de kans groot dat de tekening niet compleet getekend wordt.

U kunt het *While* statement ook in GW-BASIC gebruiken, in plaats van het programma oneindig te laten werken totdat er op een toets gedrukt wordt.

En om van Programma 35 nog meer te kunnen smullen, zal ik hieronder nog een mooiere laten zien.



Kies voor deze tekening: `While T <= 224`

Het resultaat is dan ook: hoe kleiner de draaihoek is, hoe kleiner de cirkels zijn. Maar dan zullen er ook meer cirkels getekend moeten worden, zodat de variabele T een hogere waarde moet hebben.

Het appendix over het grafisch programmeren in GW-BASIC
Na het volgende hoofdstuk *Educatieve toepassingsprogramma's* komt er een appendix. Er komen dan extra voorbeeldprogramma's van alle hoofdstukken. Er worden ook wat bekende voorbeeldprogramma's getoond met uitgebreidere code.

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 19:00 uur tot 21:00 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	Marco Kurvers	do. t/m zo.	19-22	(0342) 424452	m.a.kurvers@hccnet.nl
Office					
Web Design, met XHTML en CSS