Nieuwsbrief

19de jaargang maart 2012 **Nummer 1**

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

In dit jaar 2012 hopen we een beter jaar te hebben. Ook al zou het best wel een zwaar jaar kunnen worden, toch moeten we proberen er met z'n allen samen doorheen te komen. Dat zal de last zeker verminderen.

Ook met programmeren kunnen we proberen weer nieuwe technieken te vinden, ideeën te bedenken en nieuwtjes te ontdekken.

De laatste tijd worden projecten niet meer alleen in BASIC geschreven. Elke programmeertaal heeft zijn voordelen en nadelen. Daarom worden ze tegenwoordig samen gebruikt. Dankzij de BASIC compilers is dit nu mogelijk.

In het onderdeel 'Grafisch programmeren in GW-BASIC' komt het laatste deel over 3D tekenen en het tekenen over vlakken in de ruimte. Hier komt het antwoord over hoe we de lijnen kunnen verbergen die we niet willen zien.

BASIC tip!
Wilt u wat meer weten over de statements en functies van Liberty BASIC? U kunt een lijst bekijken in alfabetische volgorde op de website http://www.libertybasic.nl/index1.html.

Marco Kurvers

BASIC leren – PowerBASIC (3).

Eerder was ik van plan om na twee lessen met PowerBASIC te stoppen. Wel, voor degene die graag meer over PowerBASIC wil weten heeft geluk, want ik heb besloten om toch verder te gaan.

Variabelen

Variabelen vertegenwoordigen numerieke waarden of stringwaarden. In tegenstelling tot constanten, kan de waarde van een variabele veranderen tijdens de programma-uitvoering. Net als bij labels, moeten de namen van variabelen beginnen met een letter en kunnen ze maximaal 255 letters en cijfers bevatten – hoewel u in de praktijk niet de lengte kunt overschrijden. Wees gul in de naamgeving van belangrijke variabelen. In tegenstelling tot interpretive BASIC, stelen lange variabelennamen in PowerBASIC geen run-time geheugen. De enkele precisie variabelen *EndOfMonthTotals* en *emt* vereisen beide precies vier bytes in opslagruimte. Een goede vuistregel is om variabelennamen zo kort mogelijk te houden, zodat genoeg statements op één regel passen. Veel programmeurs gebruiken één-letter variabelen voor lus tellers, zoals *i*, *j*, *k*, *l* en *x*, *y*, *z* zijn vooral favorieten. Echter kunt u ook namen zoals *aantal*, *totaal*, *index* enzovoort, voor meer duidelijkheid nemen, vooral als u geneste lussen hebt.

Variabelennamen kunnen tevens tijdelijk ingevoegd zijn. Merk op dat de tijdelijke variabelen echter ook gebruikt worden om te scheiden van de leden van een door de gebruiker gedefinieerd type. Dit voorkomt dat de naam van een variabele, die een gedeclareerd gebruiker gedefinieerd recordtype omvat, niet zal botsen met dezelfde naam. Zo is onderstaande code niet toegestaan:

```
User.First.Name$ = "Roberto"

TYPE UserType
  Account AS LONG
  AmountDue AS BCD
END TYPE
Dim User AS UserType
```

Gebruikers gedefinieerde variabele <u>User</u>
botst met stringvariabele <u>User.First.Name$</u>

PowerBASIC ondersteunt veertien soorten types: string; flex string; fixed-length string; integer; long integer; quad integer; byte; word; double word; single-, double- en extended-precision floating point; binary coded decimal (BCD) fixed point en binary coded decimal (BCD) floating point.
Er zijn drie manieren om variabelen te declareren:

- *Een variabele declareren met een bepaald type.* Gebruik het DIM statement om een variabele te declareren en gebruik de AS *type* syntaxis.
- *Een type declaratie symbool toevoegen aan de variabele naam.* Hiermee bepaalt PowerBASIC zelf het standaardtype dat bij het symbool hoort.
- *Gebruik het standaardtype.* Als u niet het type declaratie symbool toevoegt met een variabele, gebruikt PowerBASIC het standaardtype single precision. Om een ander type standaard te maken gebruikt u het DEF*type* statement, bijvoorbeeld,

```
cat# = 1.312          '    cat# is een double precision variabele
cat = 16.5            '    cat is single precision als standaard
cat% = 3              '    cat% is een integer variabele
DEFINT c              '    variabelen die met de letter c beginnen
                           zijn nu standaard integers
cats = 16             '    cats is standaard een integer variabele
```

Om meer ruimte te sparen en berekeningen sneller uit te laten voeren, is het beter om de declaratie DEFINT *a-z* in alle programma's te laten beginnen. Alle variabelen zullen dan integer worden gedeclareerd en niet meer anders. Wanneer u dit consequent doet, hebt u het %-teken achter elke integer variabele niet meer nodig, en zult u niet verward raken.

Houd in gedachten dat *cat%*, *cat&*, *cat&&*, *cat!*, *cat#*, *cat##*, *cat@*, *cat@@*, *cat$* en *cat$$* tien aparte variabelen zijn. Hoewel het gebruik, van het over en weer maken met verschillende *cat* variabelen zoals dit, legaal is; goede praktische programmering suggereert dat u complete aparte namen gebruikt voor verschillende variabelen. Het is ook altijd beter om beschrijvende en begrijpbare namen voor uw variabelen te gebruiken, in plaats van gebruik te maken van enkele letters. Het is ontzettend moeilijk een programma te debuggen waar men *x@* heeft ingevoerd in plaats van *x!* of *x#*. Stel je de verwarring eens voor als je probeert *x&&* en *x&* te onderscheiden. Als u variabelennamen gebruikt zoals *count!*, *result#*, *remain##*, *company$* en *names$$*, zult u aanzienlijk minder moeite hebben met het scheiden houden van uw variabelen en hun types.

Arrays

Het is vaak nuttig een reeks variabelen als een groep te behandelen. Hierdoor kunt u gemakkelijker bewerkingen uitvoeren. Een *array* is een groep van een tekenreeks of numerieke gegevens, gedeeld onder dezelfde variabele naam. De afzonderlijke waarden die deel uitmaken van een matrix, worden *elementen* genoemd. Een element van een matrix kan worden gebruikt in een statement of in een expressie, waar u normaal gesproken regelmatig een tekenreeks variabele of een numerieke variabele gebruikt. Met andere woorden, elk element van een matrix is zelf al een variabele.

PowerBASIC biedt verschillende statements die bewerkingen uitvoeren op een gehele matrix, zodat u kunt sorteren op de inhoud, de inhoud voor gegevens die overeenkomen met een bepaalde voorwaarde te scannen, en gegevens in te voegen of gegevens uit de bestaande structuur te verwijderen.

U kun een matrix beschouwen al een rij met vakken, genummerd van nul tot een vooraf bepaald aantal; vier in het onderstaand voorbeeld. Elke doos bevat een andere waarde, die al dan niet kan afwijken van de waarden in de andere vakken. De vakken en hun nummers worden vertegenwoordigd door haakjes en een nummer; *item%(3)* vertegenwoordigt bijvoorbeeld vak nummer drie van de matrix *item%*. Dus als de waarde in het vak nummer 3 waarde 1952 is, zou de code `total%=item%(3)` de waarde 1952 in de variabele *total%* plaatsen.

50	10	-5	1952	104
item%(0)	item%(1)	item%(2)	item%(3)	item%(4)

Een vijf element numerieke array

Bij het starten van het programma is elk element van elke numerieke array ingesteld op nul en een string array wordt ingesteld op de lege string (“”, lengte nul). Dit proces heet het initialiseren van een array.
Dimensionering van een dynamische array met DIM of REDIM wist ook elk element. Als een programma met het RUN statement later opnieuw wordt gestart, wordt deze initialisatie herhaald.

Het declareren van de naam en type van een array, wordt evenals het aantal en het organiseren van de delen ervan, uitgevoerd door het DIM statement, bijvoorbeeld:

```
DIM payments(55)
```

Hiermee maakt u een array variabele *payments* die uit 56 enkel-precisie elementen bestaat, genummerd van 0 tot 55. De array *payments* en een enkel-precisie variabele ook genaamd *payments*, zijn afzonderlijke variabelen. Als dit voor u verwarrend is, zult u begrijpen waarom het beste is om altijd verschillende variabele namen te gebruiken.

Subscripts

Individuele array elementen hebben subscripts die uit integer expressies bestaan en tussen haakjes aan de rechterkant van de matrixvariabelen staan. Bijvoorbeeld, *payments*(3) en *payments*(44) zijn twee elementen van de array *payments* die 56 elementen heeft. Normaal begint het eerste matrix element met een subscript 0, hoewel dit kan worden veranderd met het DIM of OPTION BASE statement.

Onderstaande voorbeelden werken als volgt:

```
' Dit DIM statement declareert een array met 56 elementen
' met subscripts van 0 tot 55.
DIM payments(55)
.
.
.
OPTION BASE 1

' Dit DIM statement declareert een array met 55 elementen
' met subscripts van 1 tot 55 vanwege het OPTION BASE
' statement
DIM payments(55)
```

Wanneer PowerBASIC een verwijzing vindt naar een matrix die geen corresponderende DIM instructie heeft, zal de afmeting van de matrix automatisch standaard naar 11 elementen verwijzen (subscript waarden 0 tot en met 10). Het is echter aan te bevelen de tijd te nemen om expliciet elke matrix te dimensioneren die u maakt voor uw programma. U kunt met de compiler optie **O**ption/**C**ompiler/**V**ariable Declarations/**A**rrays IDE of met de commandoregel /**ODA** het verplicht stellen dat arrays gedeclareerd moeten worden.

In PowerBASIC kunt u in plaats van een normale bovengrens, zoals dat in de interpretive BASIC versies alleen maar kon, ook een bereik van subscript waarden als een matrix opgeven. Het DIM statement

```
DIM fog(50:60)
```

creëert bijvoorbeeld een array *fog* met een matrix bestaande uit elf enkele precisie elementen genummerd van 50 tot 60.

Het statement

```
DIM clouds(50:60, 25:45)
```

creëert een twee dimensionale array *clouds* met een inhoud van 231 (11 x 21) elementen.

In PowerBASIC kunt u de afstand van een subscript declaratie in een programma nauwer structureren.

Neem bijvoorbeeld een programma die 19de-eeuwse geboortestatistieken bijhoudt. Het centrale gegevensstructuur van dit programma is een enkele precisie matrix van 100 elementen die de aantal baby-'s, geboren in elk jaar van die eeuw, bevatten. Ideaal gezien zou u een matrix gebruiken waarvan de subscripts gelijk zijn aan elk jaar waarin de geboorten heeft plaatsgevonden (bijvoorbeeld, *births*(1851) vertegenwoordigt hoeveel baby's in het jaar 1851 op de wereld kwam), zodat code als hieronder het zo eenvoudig mogelijk zou maken.

```
DIM births(1899)
FOR year = 1800 TO 1899
     PRINT "Er waren "; births(year); " baby's geboren in "; year; "."
NEXT year
```

Helaas maakt het DIM statement een 1900 element array (van 0 tot 1899), waarvan de eerste 1800 ongebruikt zijn. Traditioneel hebben BASIC programmeurs dit probleem aangepakt door de matrix te declareren als

```
DIM births(99)
```

en door te spelen met subscripts:

```
FOR year = 1800 TO 1899
     PRINT "Er waren "; births(year-1800);
     PRINT " baby's geboren in "; year; "."
NEXT year
```

Nou, ook al werken dit soort oplossingen. Het compliceert wel dingen en zorgt voor vertraging van het programma omdat er 100 keer afgetrokken wordt die daarvoor er niet waren. Het is beter een bereik te declareren, zoals dit:

```
DIM births(1800:1899)    ' array births heeft een subscript bereik
                         ' van 1800 tot 1899
FOR year = 1800 TO 1899
     PRINT "Er waren "; births(year); " baby's geboren in "; year; "."
NEXT year
```

OPTION BASE kan ook worden gebruikt om de laagste aantal elementen te bepalen van een array. Het kan waarden hebben van 0 tot 32767. Anders dan te proberen te onthouden welke is gezet die u hebt gebruikt voor een programma, is het beter om de basis met standaardwaarde 0 achterwege te laten en arrays te declareren die zelf verschillende systeemnummers nodig hebben:

```
DIM births(99)           ' Dit array heeft 100 elementen van 0 tot 99
DIM birth2(1:99)         ' Dit array heeft 99 elementen van 1 tot 99
DIM birth3(3 TO 99)      ' Dit array heeft 97 elementen van 3 tot 99
```

String arrays
De elementen van string arrays houden strings vast in plaats van getallen. Elke string kan een andere lengte hebben van 0 tot de huidige string segment lengte (maximum 32750 tekens). De totale stringruimte voor strings en string arrays, regelmatig flex en vaste lengte, is onbeperkt. Bijvoorbeeld,

```
DIM words$(50)
```

creëert een volgorde van 51 onafhankelijke stringvariabelen:

```
words$(0) = "Cathie likes cats."       'een 18 karakterstring
words$(1) = ""                         'een nul-string (lege string)
words$(2) = "Nicki is a sweet child."  'een 23 karakterstring
.
.
.
words$(50) = SPACE$(200)               'een 200 karakterstring
```

Flex string arrays zijn net zo gedimensioneerd als reguliere strings; het MAP statement wordt gebruikt om de lengte op te geven voor elk element van de array:

```
DIM cloud$$(9)
MAP cloud$$() * 12
```

Dit declareert een array van 10 strings, elk van 12 karakters lang. U kunt ook een variabele of een numerieke expressie als lengte in het MAP statement gebruiken.

Multidimensionale arrays
Arrays kunnen één of meer dimensies hebben tot een maximum van acht. Een eendimensionaal array zoals *payments* is een simpele lijst van waarden. Een tweedimensionaal array presenteert een tabel van getallen met rijen en kolommen met informatie. Multidimensionale arrays zijn evenzo mogelijk:

```
DIM one(15)          ' een eendimensionale lijst
DIM two(15, 20)      ' een tweedimensionaal tabel
DIM three(7, 9, 1)   ' een 8 bij 10 speelbord met een driedimensie
                     ' kamer voor twee items: het type en de kleur
```

Arrays van vier tot acht dimensies zijn mogelijk, maar zijn veel moeilijker te gebruiken. U kunt definiëren:

```
DIM five(5, 5, 10, 20, 3)
```

maar het is natuurlijk beter om dit array te herdimensioneren in een minder aantal dimensies.

Het maximum aantal elementen per dimensie is 32767. De inhoud van een array moet binnen de 64K liggen (niet de aantal stringgegevens maar juist de stringhandelingen, bepaald uit een string array of een flex string array).

De array grenzen testen
PowerBASIC probeert hard in te treden om problemen te voorkomen met foute subscripts. Dat zijn subscripts die te hoog of te laag zijn voor een gegeven array. De compiler vertelt u over foute subscripts:

```
DIM big(50)
big(51) = 33
```

Dit programma zal niet gecompileerd worden omdat PowerBASIC geen code wil genereren om de 52ste element toe te staan van een 51-element array. Hoewel, als u variabelen gebruikt als subscripts zal PowerBASIC niet de gemaakte fout vinden tijdens het compileren:

```
DIM big(50), big2(50)
you = 51
big(you) = 33
```

Deze code wordt gecompileerd zonder een foutmelding. U kunt de ‘out of range’ controleren in regel 3 tijdens de uitvoer door te compileren met de **O**ptions/**C**ompiler/**E**rror Tests/**B**ounds Test optie en die in te schakelen. Als *big*(51) niet in de gedeclareerde array voorkomt en de optie Bounds Test uitstaat zal er een volgend deel in het geheugen worden gebruikt. Het is onwaarschijnlijk dat dit uw bedoeling was. Als het uw bedoeling was, moet u, wanneer u merkt dat uw grenzen niet goed zijn, uw programma vernieuwen zodat anderen makkelijker begrijpen hoe het later zal worden. Tijdens de uitvoer kunt u verwijzingen naar arrays, die nog niet zijn gedimensioneerd, of die zijn gewist, laten testen met de testoptie. Deze kunt u dan vervangen of opnieuw dimensioneren.

Samenvatting
PowerBASIC kent nog meer soorten arrays met constanten en equates, enzovoort. Teveel om op te noemen. U weet nu in ieder geval het belangrijkste van arrays: het gebruik ervan.

In de volgende nieuwsbrief laat ik u zien hoe PowerBASIC omgaat met expressies en operators.

Marco Kurvers

BASIC en andere programmeertalen.

Hoe verder we gaan met programmeren in de techniek, hoe belangrijker het wordt om te weten hoe andere programmeertalen in elkaar zitten. Er worden nu verschillende programmeertalen gebruikt voor het ontwikkelen van een applicatie. De reden is omdat elke programmeertaal zijn eigen voordelen en nadelen heeft. Heeft bijvoorbeeld BASIC voor een bepaalde methode een voordeel dan kan bijvoorbeeld Delphi er een nadeel aan hebben, en andersom.

Door alles samen te bundelen en te compileren, zal de applicatie sneller werken. Zo kunnen DLL's en objectlibrary's beter geschreven worden in C++ dan in BASIC en menu's, vensters en dialoogformulieren weer in BASIC. Maken we in BASIC een DLL dan is dat geen probleem, vooral in deze tijd niet meer, maar het nadeel is nog steeds dat BASIC iets mist waardoor een onderdeel maken in C++ beter geschikt is.

Operatoren.
In BASIC kennen we verschillende operatoren. Om gelijk met de deur in huis te vallen, ziet u hieronder een tabel met de bekende operatoren die wij gebruiken. Zie eens hoe de verschillen zijn in Delphi en in C++.

Operator		Delphi	C++
`x = n`	*toekennen*	`x := n`	`x = n`
`x = n`	*vergelijken*	`x = n`	`x == n`
`<`	*kleiner dan*	`<`	`<`
`>`	*groter dan*	`>`	`>`
`<=`	*kleiner dan of gelijk aan*	`<=`	`<=`
`>=`	*groter dan of gelijk aan*	`>=`	`>=`
`<>`	*ongelijk aan*	`<>`	`!=`
`not`	*niet*	`not`	`!`
`and`	*en*	`and`	`&&`
`or`	*of*	`or`	`||`

Zoals u in het tabel ziet, kent Delphi de meeste BASIC operatoren. Het vertalen van de code zal daarom met die twee talen minder moeilijk zijn dan met C++.
Vergeet niet dat we ook nog de vadertalen hebben: Pascal en C. Delphi ziet eruit als Pascal, maar is in feite een *Object Pascal* taal. Zo begonnen we vroeger met C dat later uitgebreid werd en de naam C++ kreeg. We hebben nu ook C-sharp *C#*.

Unaire operatoren.
De *unaire operatoren* zijn de enige operatoren die we in BASIC niet kunnen gebruiken. Toch is het handig om te weten wat het voor operatoren zijn en hoe we het in BASIC kunnen vergelijken.

De operatoren zijn: ++ en --.
In C++ kunnen we een coderegel tegenkomen die bijvoorbeeld er zo uitziet:

```
y++;       // in BASIC werkt dit als: y = y + 1
```

of zo:

```
y--;       // in BASIC werkt dit als: y = y - 1
```

In Visual Basic .NET kunnen we een andere operator gebruiken:

```
y += 1:z -= 1  ' in C++ werkt dit, verrassend, op dezelfde manier
```

Helaas kent Delphi ook geen unaire operatoren, zelfs niet de += en de -= operatoren. Dit betekent dat Visual Basic erg goed vooruit is gegaan. Meestal blijven degenen, die niet te technisch willen denken, liever op de oude BASIC manier. Tja, tegenwoordig gebruiken we toch een compiler. Wat willen we nog meer?

Er is een BASIC compiler, en dan?
We gebruiken een BASIC compiler om programma's nog sneller te laten werken dan met interpretive BASIC. Het verschil is dat een compiler eerst het programma converteert naar machinetaal en het dan pas uitvoert. Een interpreter doet het beide. Elke coderegel wordt nagekeken en wordt direct uitgevoerd. Voor de meeste structuren, zoals beslissingen en lussen, is dit een ramp. Wordt een lus 100 keer herhaald dan moet de interpreter 100 keer dezelfde code in de lus nakijken. De interpreter 'vergeet' direct in de volgende regel wat deze in de vorige regel gedaan heeft.

Het antwoord om de vraag 'en dan?' zou ik liever voor de BASIC gebruikers niet willen geven, maar toch laat ik een voorbeeld zien waarom een BASIC compiler niet helemaal doet om de juiste C++ snelheid te krijgen.

Normale en unaire operatoren in de praktijk.
Misschien hebt u wel eens coderegels gebruikt zoals hieronder samen:

```
x = y
y = y + 1
```

In C++ kunnen we het net zo gebruiken, maar de uitvoer zal echter trager zijn dan wanneer we onderstaande code gebruiken met een unaire operator:

```
x = y++;
```

De vraag is nu: waarom is deze code beter dan de bovenste twee? Voordat ik het antwoord geef, zal ik nog wat laten zien. C++ is echt een taal waarmee we allerlei kanten op kunnen:

```
x = ++y;        // wat krijgen we nu?
```

Dit is niet fout en het zal zelfs niet op dezelfde manier werken als de andere. Als we BASIC erbij halen, zien we pas het verschil met die twee:

```
y = y + 1
x = y
```

Waarom zouden we de eenregelige methode gebruiken in plaats van de tweeregelige methode?
Dat heeft allemaal te maken hoe de compiler de code compileert naar assembleertaal. De eenregelige methode zal veel minder assembleertaalregels in beslag nemen dan de tweeregelige methode, omdat het voor de compiler gewoon veel meer werk kost om het in assembleertaal om te zetten.
Dit is ook een van de redenen waarom grote spellen in C++ geschreven worden. C++ is omvangrijk, functioneel en cryptisch. De taal heeft minder sleutelwoorden dan BASIC, maar heeft wel toegang tot alle klassenbibliotheken van het systeem. Hieronder ziet u bijvoorbeeld hoe u de in- en uitvoermethoden kunt gebruiken:

```
#include <stdio.h>
```

De *h*eaderfile <stdio.h> heeft een hele bibliotheek met functies waarvan we gebruik kunnen maken met in- en uitvoer. Vergeten we de #include regel op te nemen, dan kunnen we onderstaande code niet gebruiken:

```
printf("Hallo allemaal.\nHoe gaat het met jullie?");
```

De functie printf() is aanwezig in de header. Het is dus geen C++ sleutelwoord zoals PRINT dat in sommige BASIC versies wel is.

Beslissingen en lussen.
Er bestaan verschillende soorten beslissingen en lussen. Bekijk eens onderstaand tabel:

BASIC	Delphi	C++
IF [NOT] conditie THEN … [ELSE …] [END IF]	if [not](conditie) then … [else …]	if ([!]conditie) … [else …]

SELECT CASE variabele
CASE n
…
[CASE ELSE …]
END SELECT
	case identifier: type of
	n: …
		switch (variabele)
		{
		case n: …
		[default: …]
		}

WHILE conditie … WEND	while conditie do …	while (conditie) …
DO … LOOP WHILE NOT conditie	repeat … until conditie	do { … } while (!conditie)
FOR m=n TO x … NEXT [m]	for m:=n to x …	for (m=n; m<=x; m++) …
FOR m=n TO x STEP y … NEXT [m]	n.v.t	for (m=n; m<=x; m+=y) …
FOR m=x TO n [STEP y] … NEXT [m]	for m:=x downto n … *// step statement n.v.t*	for (m=x; m>=n; m-=y) …
var=IIF(conditie, true-waarde, false-waarde)	n.v.t	var=conditie?true-waarde:false-waarde

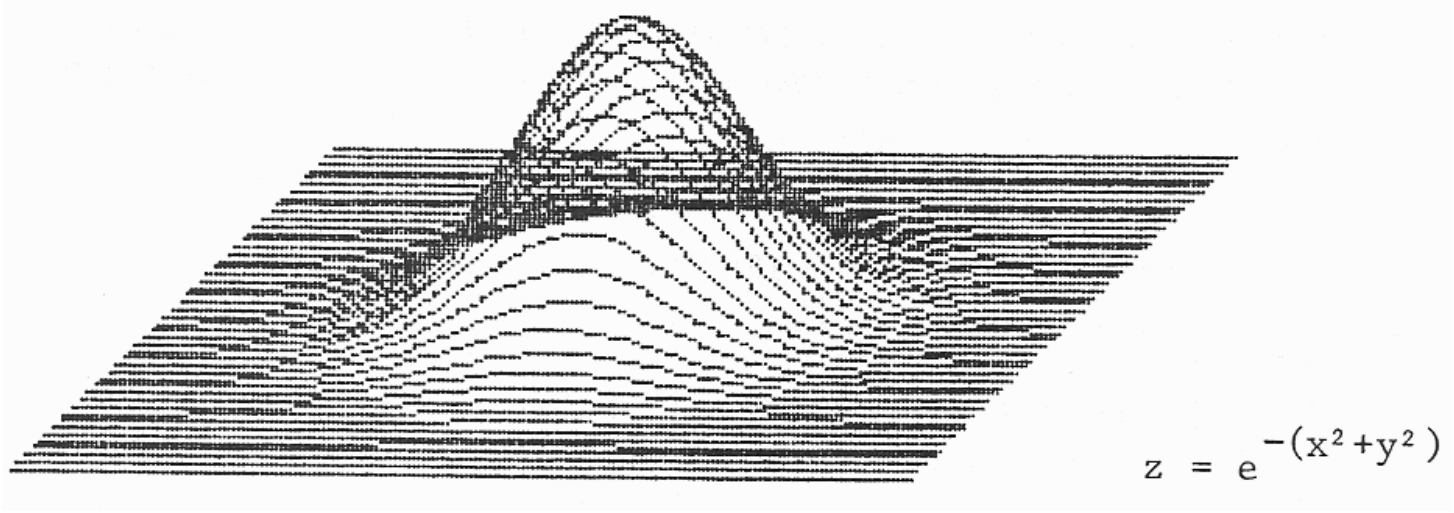
Helaas kunt u zien dat Delphi weer niet alles kent wat BASIC wel kent. Bijvoorbeeld het STEP statement dat ontbreekt en de handige IIF functie in BASIC die ook niet van toepassing is in Delphi, maar wel in C++ met de operatoren ? en :.

Als u zelf programma's wilt omzetten, maak dan eerst voor uzelf een tabel met een BASIC kolom. Op die manier hebt u altijd een prima overzicht. Op internet kunt u ook veel oplossingen vinden over de statements en functies van andere programmeertalen. Mocht het niet lukken, stuur mij dan het probleem en ik help u mee het op te lossen.

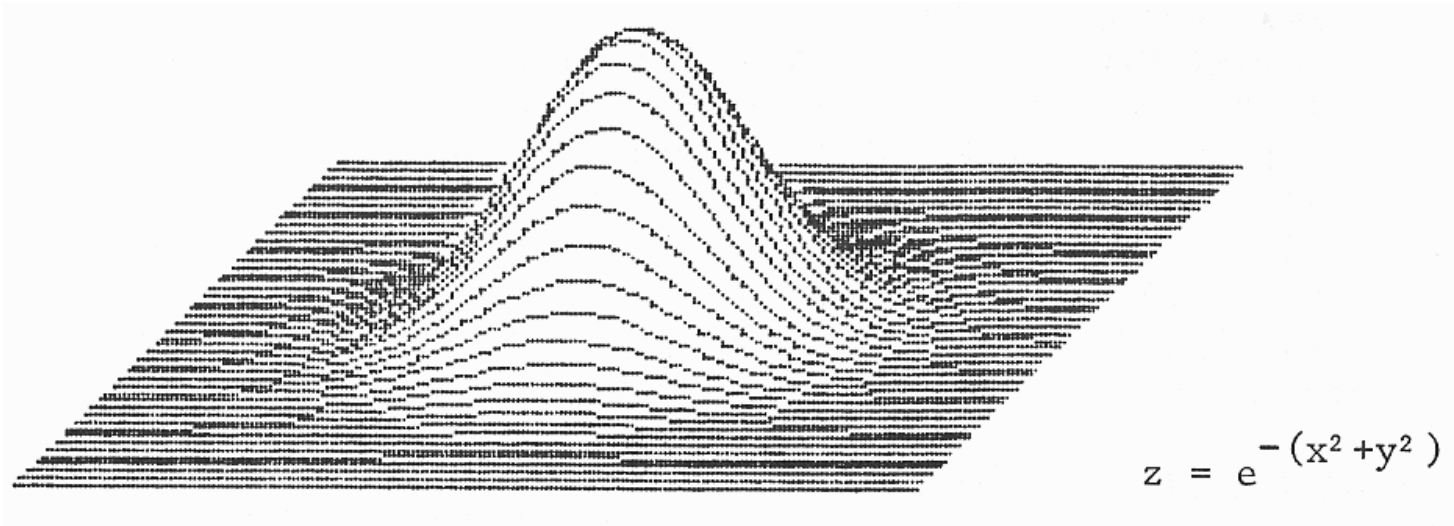
Marco Kurvers

Grafisch programmeren in GW-BASIC (10).

In de vorige nieuwsbrief hadden we Programma 28. Hieronder ziet u nogmaals de ‘foute’ tekening waar we natuurlijk niet tevreden over zijn en de ‘goede’ tekening waar we nog steeds het antwoord op de derde vraag moeten hebben.



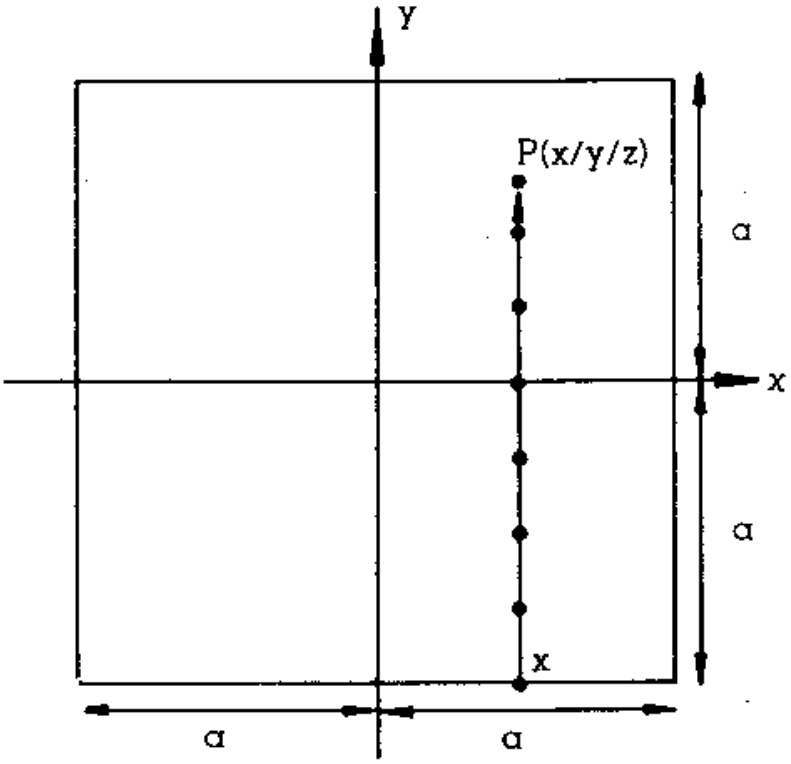
Hoe krijgen we de goede tekening voor elkaar?



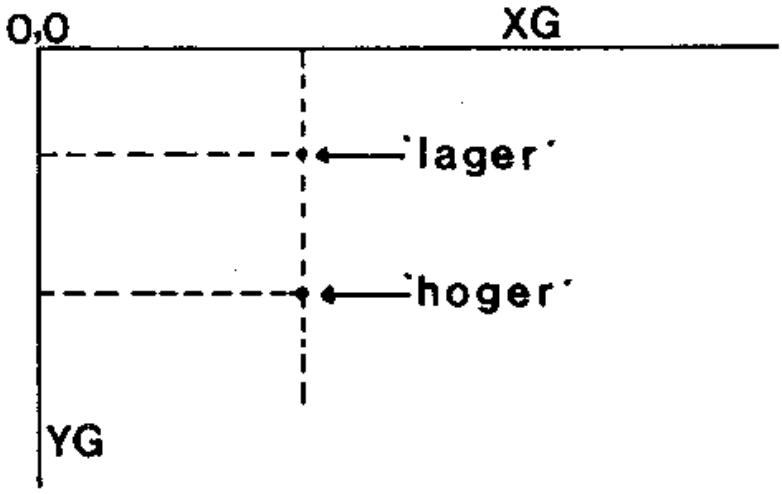
Antwoord op de derde vraag.
Bekijk de onderstaande tekening. Een punt P(x,y,z) van het te tekenen vlak is alleen dan zichtbaar wanneer:

a. het punt ‘hoger’ ligt dan alle voorgaande punten met dezelfde x-coördinaat of
b. het punt ‘lager’ ligt dan alle voorgaande punten met dezelfde x-coördinaat.

Vanzelfsprekend geldt dit ook voor de op het beeldscherm geprojecteerde punten P(XG,YG).



Bedenk bij de navolgende uiteenzetting dat de coördinaten (XG,YG) van een beeldschermpunt 'gemeten' worden ten opzichte van de linkerbovenhoek van het beeldscherm, de oorsprong van ons coördinatenstelsel.



Een punt, in beeldschermcoördinaten uitgedrukt, ligt dus 'hoger 'dan een ander punt (bij dezelfde XG-waarde) als de YG-waarde groter is en ligt 'lager 'als de YG-waarde kleiner is.

We gaan als volgt te werk. We gebruiken twee arrays (H1(XG) en H2(XG)) waarin we voor alle XG-coördinaten in ons vlak de respectievelijk kleinste en grootste YG-waarde bewaren. Vinden we voor een punt met een bepaalde XG-waarde een bijbehorende YG-waarde die òf kleiner is dan H1(XG) òf groter is dan H2(XG), dan is het punt zichtbaar en wordt H1(XG) of H2(XG) gelijk gemaakt aan deze nieuwe kleinste of grootste YG-waarde. Geldt voor een bepaalde XG dat H1(XG)<YG<H2(XG), dan is het punt (XG,YG) niet zichtbaar. H1(XG) en H2(XG) zijn te zien als twee horizonnen.

Als voor een bepaald punt $P_1(x,y,z)$ uit het vlak het beeldpunt P1(XG,YG) berekend is, gaan we kijken of YG kleiner dan of gelijk aan H1(XG) is (YG<=H1(XG)). Is dit zo dan is het punt P1 op het beeldscherm zichtbaar. Nu zetten we de vlag F1 op 1 en we maken H1(XG) gelijk aan YG (H1(XG)=YG)). Is dit niet het geval (YG>H1(XG)), dan is P1 onzichtbaar en blijft de vlag F1 op 0 staan.

Nu gaan we naar het volgende punt $P_2(x,y,z)$ uit het vlak (we verhogen x met dx). Opnieuw berekenen we het beeldpunt P2(XG,YG). Weer wordt gekeken of YG<=H1(XG) en zonodig wordt de vlag F2 op 1 gezet. De punten P1 en P2 worden alleen dan door een rechte lijn (lijntje) verbonden als beide vlaggen F1 en F2 de waarde 1 hebben, dus als F1xF2=1.

Ditzelfde doen we voor de tweede 'horizon' H2(XG). We kijken of YG>=H2(XG), ….., enzovoorts.

Omdat we bij een vaste y-waarde de x-waarde steeds met dx ophogen (bijvoorbeeld dx=3) en hiermee steeds twee buurpunten P1(XG,YG) en P2(XG,YG) berekenen slaan we als het ware een aantal punten over waarvoor geen H1(XG)- en H2(XG)-waarden berekend worden. Door lineaire interpolatie tussen de horizonwaarden (H1(XG) en H2(XG)) van twee buurpunten zouden we de horizonwaarden voor de hiertussen liggende punten kunnen berekenen. We hebben deze waarden wel nodig, omdat het kan voorkomen dat, als we y met dy verhogen, we XG-waarden vinden waarvoor nog geen H1(XG)- en H2(XG)-waarden berekend zijn.

Een programma dat de hierboven geschetste werkwijze zou volgen zou, in BASIC geschreven, veel te langzaam zijn. Om de tekensnelheid acceptabel te houden, dat wil zeggen niet langer dan 10 minuten tekenen voor één figuur, passen we de volgende vereenvoudigingen toe:

1. We berekenen alleen de onderste (voor de kijker de bovenste) horizon H1(XG).
2. We passen geen lineaire interpolatie toe bij het berekenen van de array-waarden in H1(XG). Alle beeldpunten in een interval ter lengte dx krijgen dezelfde H1(XG)-waarde.

Als we de stapgrootte dx maar klein genoeg kiezen (een goede waarde is dx=3), krijgen we ondanks deze twee simplificaties toch acceptabele tekeningen. (Bekijk de volgende tekeningen en oordeelt u zelf.) Het niet tekenen van 'voor de kijker' onzichtbare punten komt tot stand door het volgende stukje programma:

```
F1=0:L=INT(XG/DX)
IF YG<=H(L) THEN F1=1:H(L)=YG
```

Laten we dit met een voorbeeld illustreren. Stel we berekenen van een punt P(x,y,z) in het ruimtelijke vlak de projectie op het beeldscherm. Dit gebeurt door eerst met x en y de z-waarde te berekenen. (In programma 29 krijgt Y in regel 270, X in regel 290 en Z in regel 1000 een waarde.) Als de x-, y- en z-waarde van een punt in het vlak berekend zijn, wordt dit punt op het beeldscherm geprojecteerd door het berekenen van de twee coördinaten XG en YG. (In programma 29 gebeurt dit in de regels 300 en 310.) Stel nu dat als beeldpunt het punt (XG=40 en YG=126) berekend is. Het programma zal de vlag F1 of F2 op nul zetten en de waarde L=INT(XG/DX) berekenen. Voor XG=40 en DX=3 wordt deze waarde INT(40/3), dus L=13. Dit betekent in feite dat de drie punten met XG=39, XG=40 en XG=41 dezelfde horizonwaarde L=13 opleveren. Mocht nu eerder in H(13) als horizonwaarde de waarde 132 opgeslagen zijn, dan is YG<=H(L) (regel 360 of 390) waar, want 126<=132 is waar, en is het punt (40,126) zichtbaar. Op dit moment wordt de vlag F1 of F2 gezet en wordt YG=126 de nieuwe horizonwaarde bij L=13.

De regels

```
210 DIM H(320)
220 FOR L=0 TO 320
230    H(L)=1000
240 NEXT L
```

(neem 640 in plaats van 320 als u voor DX de waarde 1 kiest)

zorgen ervoor dat, voordat het tekenen begint, het hele scherm 'zichtbaar' is. Dit is zo als de onderste horizon (voor de kijker de bovenste horizon) de onderrand van het beeldscherm is. De waarde 1000 geeft in feite een horizon die 'ver onder' het beeldscherm ligt. Op de volgende pagina staat het programma dat alle voor ons onzichtbare punten ook niet tekent.

Met W=45^0, K=0,5, A=3 en K1=80 krijgt u de grafiek van $z = e^{-(x^2+y^2)}$ zoals die een paar pagina's eerder is getekend; de niet-zichtbare punten die achter de hoed liggen zijn inderdaad niet te zien!

Hoe kleiner we DX en DY maken, des te gedetailleerder zal de figuur worden. Maar ook ‘des te langer zal het tekenen duren’. De combinatie DX=3 en DY=5 is een goed compromis tussen de tekensnelheid en de mate van gedetailleerdheid.

```
100 'programma 29       GRAFIEK VAN Z=F(X,Y),
110 '                   MET VERBORGEN LIJNEN
120 CLEAR ,19202 : SCREEN 105,,3,3
130 CLS: KEY OFF
140 INPUT"PROJECTIEHOEK IN GRADEN(45-135)";W
150 INPUT"VERKLEININGSFACTOR      (0.5-1)";K
160 INPUT"RECHTERGRENS VOOR X     ( > 0 )";A
170 INPUT"VERGROTINGSFACTOR       (30-80)";K1
180 U=320 : V=160 : H=.5 : RD=4*ATN(1)/180
190 C=K*COS(W*RD) : S=K*SIN(W*RD)
200 DX=3 : DY=5 : AF=A/210
210 DIM H(320)
220 FOR L=0 TO 320
230     H(L)=1000
240 NEXT L
250 CLS
260 FOR YY= -210 TO 210 STEP DY
270    Y=YY*AF
280    FOR XX= -210 TO 210 STEP DX
290       X=XX*AF: GOSUB 1000
300       XG=INT(U+XX+C*YY+H)
310       YG=INT(V-S*YY-Z+H)
320       IF YG >= 0 AND YG <= 320 THEN 340
330       PRINT "FOUTE VERGROTINGSFACTOR":STOP
340       IF XX > -210 THEN 380
350       F1=0  : L=INT(XG/DX)
360       IF YG <= H(L) THEN F1=1 : H(L)=YG
370       X1=XG : Y1 = YG : GOTO 440
380       F2=0  : L=INT(XG/DX)
390       IF YG <= H(L) THEN F2=1 : H(L)=YG
400       X2=XG : Y2=YG
410       IF F1*F2 <> 1 THEN 430
420       LINE (X1,Y1) - (X2,Y2), 1
430       X1=X2 : Y1=Y2 : F1=F2
440    NEXT XX
450 NEXT YY
460 A$=INKEY$: IF A$="" THEN 460
470 CLS: KEY ON: END
480 '
1000 Z=K1*EXP(-(X*X + Y*Y))
1010 RETURN
```

Dit programma is een algemeen programma voor het tekenen van draai symmetrische ruimtelijke vlakken. Als u andere figuren wilt maken, verander dan alleen de functie in de subroutine 1000 en kies mogelijk andere waarden voor de variabelen.

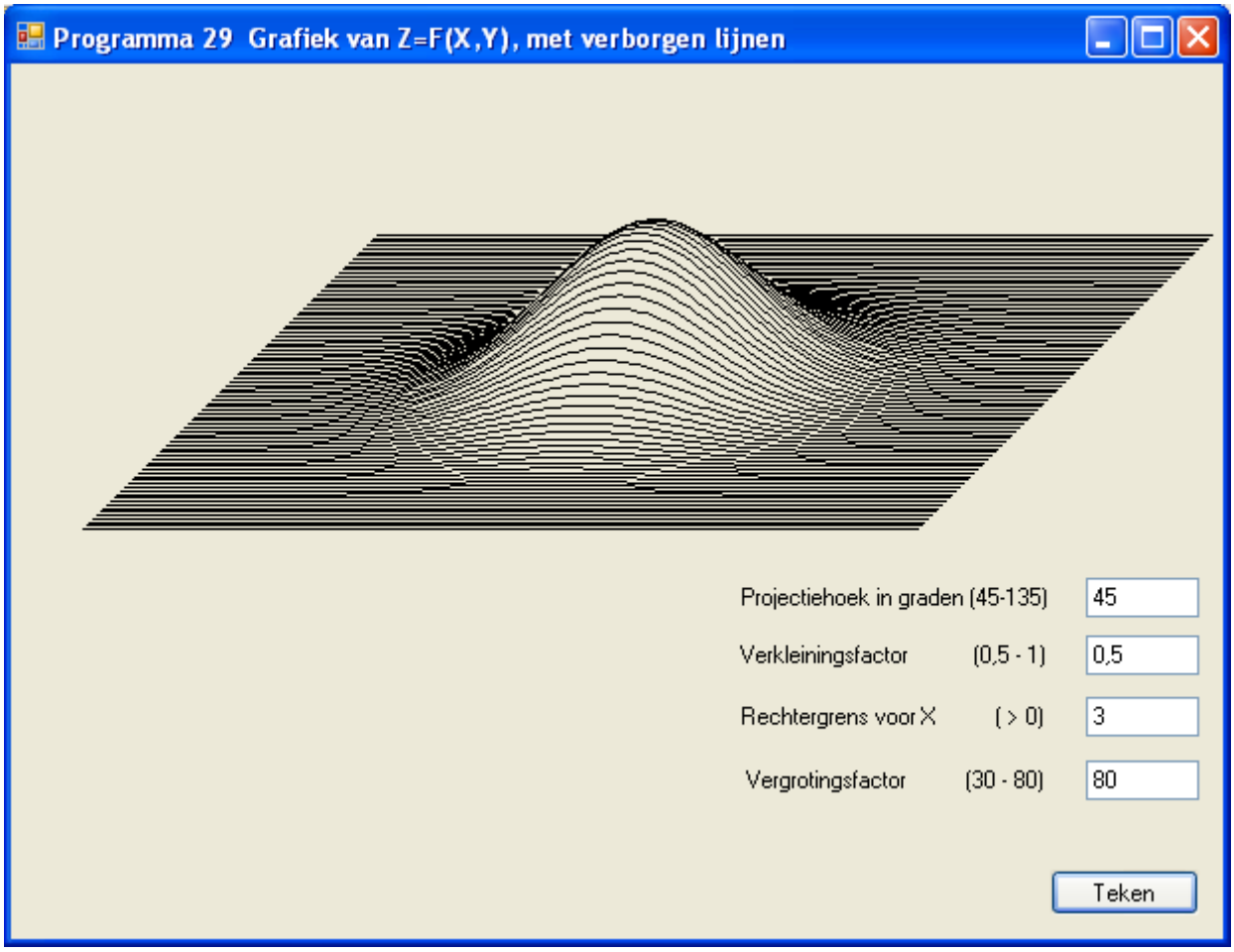
```
Public Class frmProg29
    Private Const U As Integer = 320
    Private Const V As Integer = 160
    Private Const H As Double = 0.5

    Private Function GraphFunc(ByVal intK1 As Integer, _
                               ByVal dblX As Double, _
                               ByVal dblY As Double) As Double
        Return intK1 * Math.Exp(-(dblX * dblX + dblY * dblY))
    End Function

    Private Sub frmProg29_Paint(…, ByVal e As …PaintEventArgs) …
        If Not (txtW.Text() = "" And txtK.Text() = "" And _
                txtA.Text() = "" And txtK1.Text() = "") Then
            Dim W As Integer = CInt(txtW.Text())
            Dim K As Double = CDbl(txtK.Text())
            Dim A As Integer = CInt(txtA.Text())
            Dim K1 As Integer = CInt(txtK1.Text())
            Dim RD As Double = 4 * Math.Atan(1) / 180
            Dim C As Double = K * Math.Cos(W * RD)
            Dim S As Double = K * Math.Sin(W * RD)
            Dim DX As Byte = 3, DY As Byte = 5
            Dim AF As Double = A / 210
            Dim arrayH(320) As Integer
            Dim F1 As Byte = 0, F2 As Byte = 0
            Dim X1 As Integer = 0, X2 As Integer = 0
            Dim Y1 As Integer = 0, Y2 As Integer = 0
            Dim bStop As Boolean = False
            Dim L As Integer
            For L = 0 To 320
                arrayH(L) = 1000
            Next
            For YY As Integer = -210 To 210 Step DY
                Dim Y As Double = YY * AF
                For XX As Integer = -210 To 210 Step DX
                    Dim X As Double = XX * AF
                    Dim Z As Double = GraphFunc(K1, X, Y)
                    Dim XG As Integer = Int(U + XX + C * YY + H)
                    Dim YG As Integer = Int(V - S * YY - Z + H)
                    If YG >= 0 And YG <= 320 Then
                        If XX > -210 Then
                            F2 = 0
                            L = Int(XG / DX)
                            If YG <= arrayH(L) Then
                                F2 = 1
                                arrayH(L) = YG
                            End If
                            X2 = XG : Y2 = YG
                            If Not (F1 * F2 <> 1) Then
                                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                            End If
                            X1 = X2 : Y1 = Y2 : F1 = F2
                        Else
                            F1 = 0
                            L = Int(XG / DX)
                            If YG <= arrayH(L) Then
                                F1 = 1
                                arrayH(L) = YG
                            End If
                            X1 = XG : Y1 = YG
                        End If
                    Else
                        bStop = True
                        Exit For
                    End If
                Next
```

```
                If bStop Then Exit For
            Next
        End If
    End Sub

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



Probeer uit te zoeken hoe ik de GW-BASIC code veranderd heb naar Visual Basic 2008. Zoals u in eerdere programma's hebt kunnen zien, vermijd ik de sprongen achter de THEN statements en zorg ik ervoor dat de structuren zo aangepast kunnen worden dat de GOTO statements niet meer nodig zijn, bijvoorbeeld het gebruiken van ELSE statements. Het kan ook gebeuren dat de condities omgekeerd beslist moeten worden. Om dat te kunnen doen moet u gebruik maken van een NOT voor een conditie.

De variabele H wordt door Visual Basic niet geaccepteerd om deze als een array te gebruiken, vandaar dat ik een variabele arrayH als array gebruik. Zorg ervoor dat u dus array variabelennamen en normale variabelennamen verschillend houdt.

Om u nog meer te laten genieten over het tekenen van vlakken in de ruimte, zal ik nog meer functies met de opgegeven waarden laten zien die verbluffende voorbeelden weergeven.

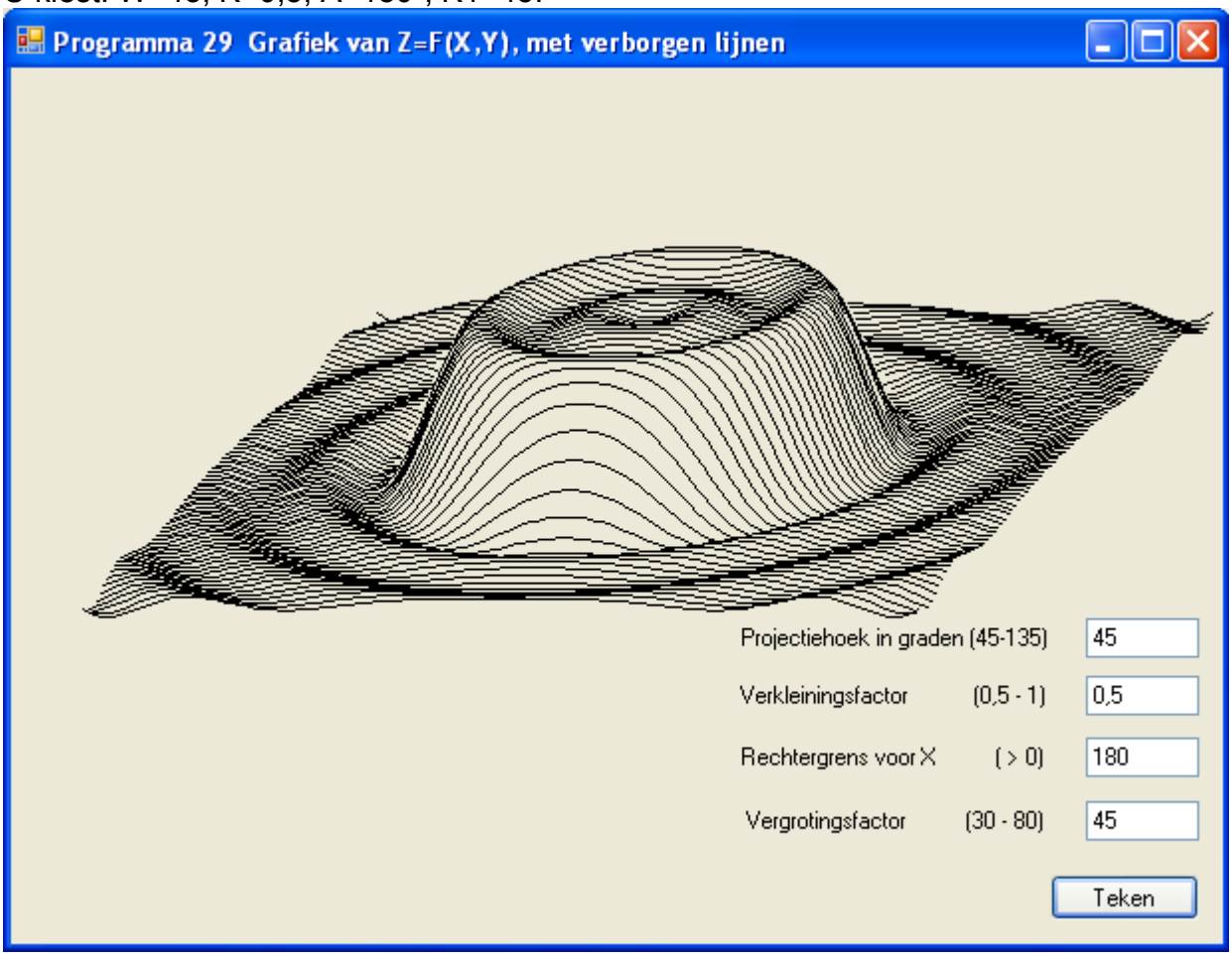
Hier komen ze.

```
1000 R=SQR(X*X+Y*Y)*RD
1010 Z=K1*(COS(R)-COS(3*R)/3+COS(5*R)/5-COS(7*R)/7)
1100 RETURN
```

Plaats onderstaande functie in Programma 29 en zorg ervoor dat u deze functie aanroept op de plaats waar u de vorige functie aanriep. Doe dat ook met de volgende functies.

```
Private Function GraphFuncV1(ByVal dblRD As Double, _
                             ByVal intK1 As Integer, _
                             ByVal dblX As Double, _
                             ByVal dblY As Double) As Double
    Dim R As Double = Math.Sqrt(dblX * dblX + dblY * dblY) * dblRD
    Return intK1 * (Math.Cos(R) - Math.Cos(3 * R) / _
                    3 + Math.Cos(5 * R) / 5 - Math.Cos(7 * R) / 7)
End Function
```

U kiest: W=45, K=0,5, A=180^0, K1=45.

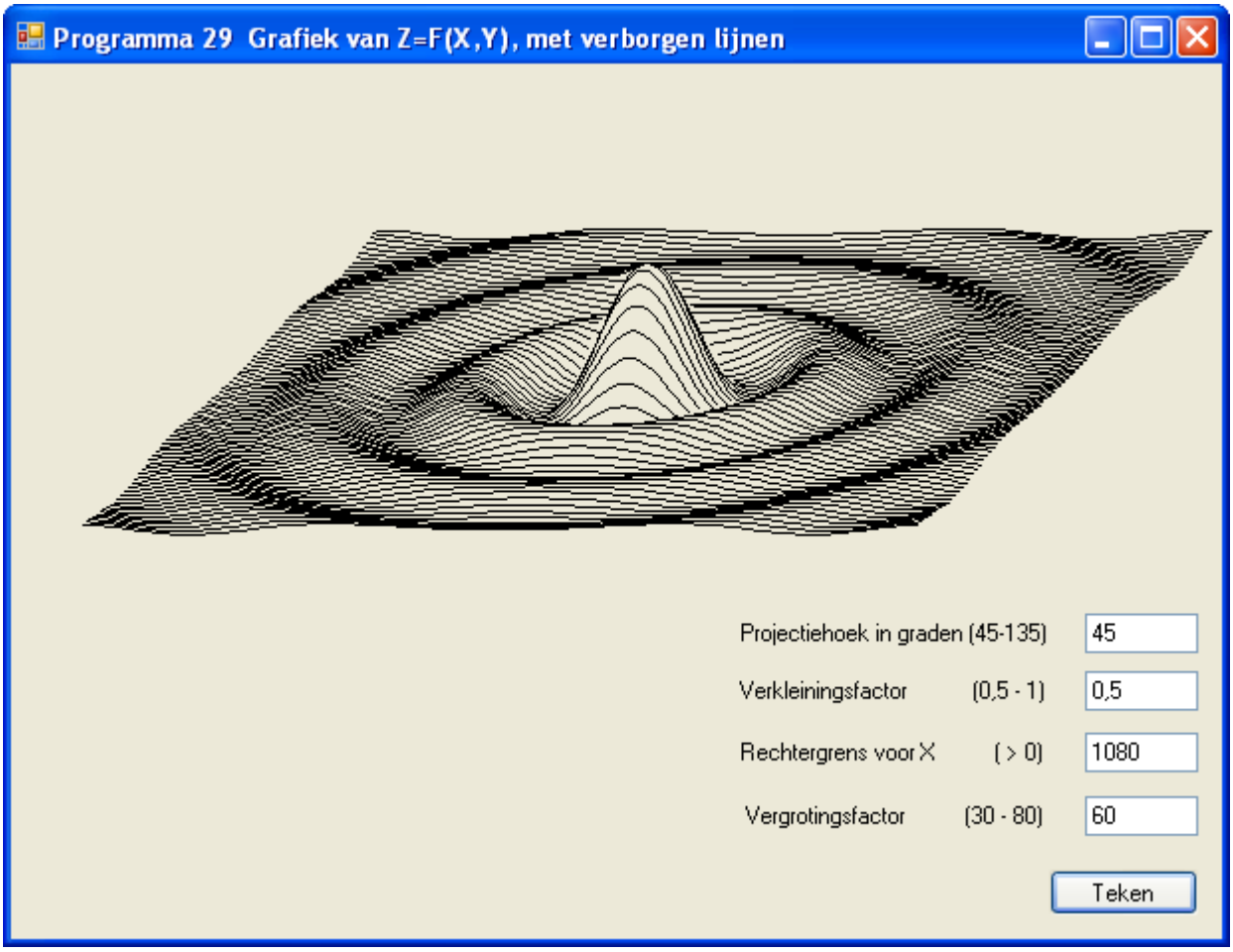


$$z = \cos(r) - \frac{\cos(3r)}{3} + \frac{\cos(5r)}{5} - \frac{\cos(7r)}{7}$$

```
1000 R=SQR(X*X+Y*Y)*RD
1010 IF R=0 THEN Z=K1 : RETURN
1020 Z=K1*SIN(R)/R
1100 RETURN
```

```
Private Function GraphFuncV2(ByVal dblRD As Double, _
                             ByVal intK1 As Integer, _
                             ByVal dblX As Double, _
                             ByVal dblY As Double) As Double
    Dim R As Double = Math.Sqrt(dblX * dblX + dblY * dblY) * dblRD
    Dim Z As Double = 0
    If R = 0 Then
        Z = intK1
    Else
        Z = intK1 * Math.Sin(R) / R
    End If
    Return Z
End Function
```

U kiest: W=45, K=0,5, A=1080, K1=60.



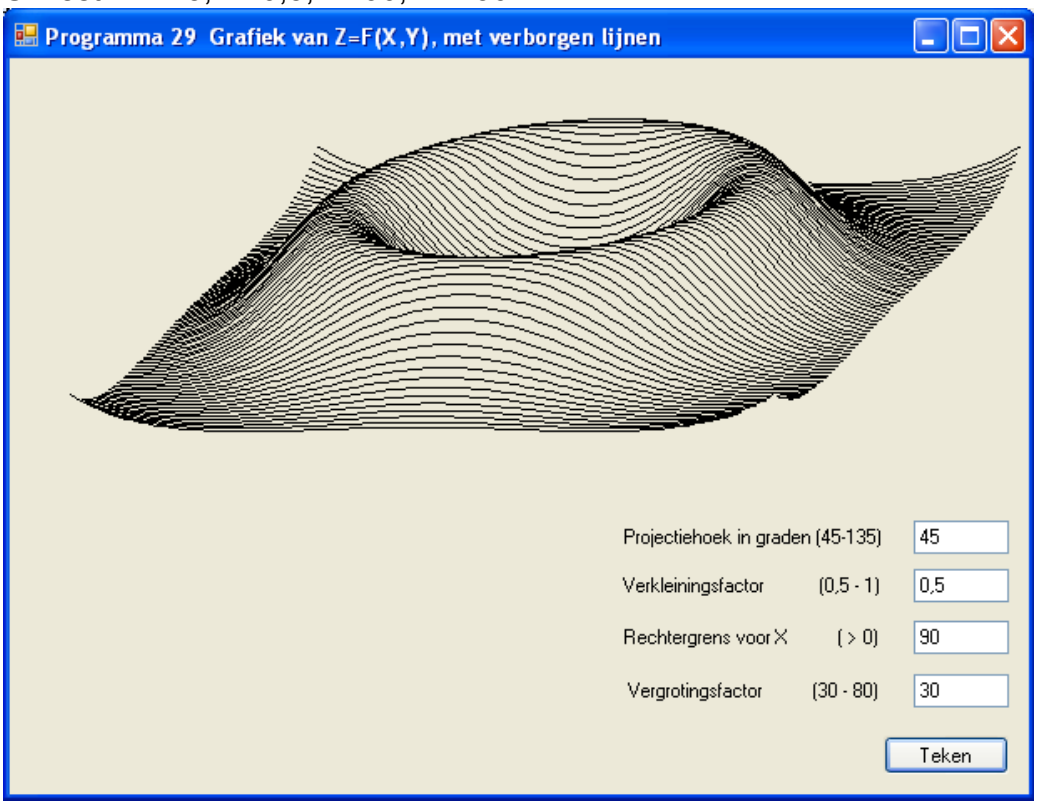
$$z = \frac{\sin(r)}{r}$$

```
1000 R=SQR(X*X+Y*Y)
1010 Z=K1*EXP(-COS(R/16))
1100 RETURN
```

Maak de subroutine als functie in Visual Basic zonder parameter RD. De rest is hetzelfde.

```
Dim R As Double = Math.Sqrt(dblX * dblX + dblY * dblY)
Return intK1 * Math.Exp(-Math.Cos(R / 16))
```

U kiest: W=45, K=0,5, A=90, K1=30.



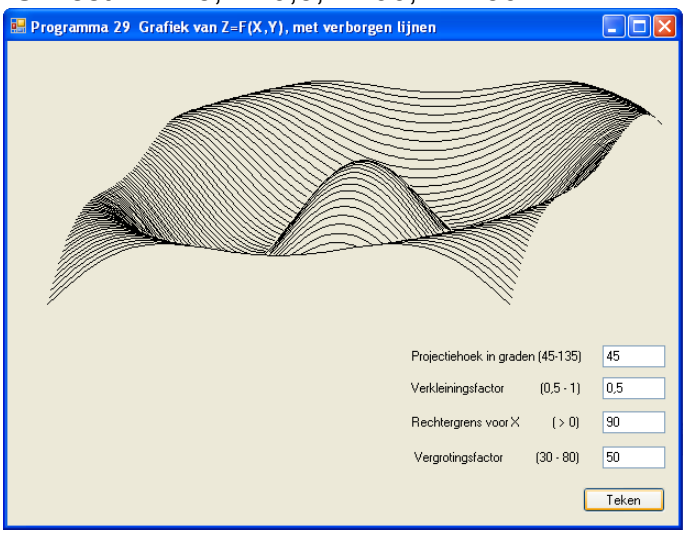
$$z = \exp(-\cos(\frac{r}{16}))$$

Voor de volgende twee tekeningen geldt:

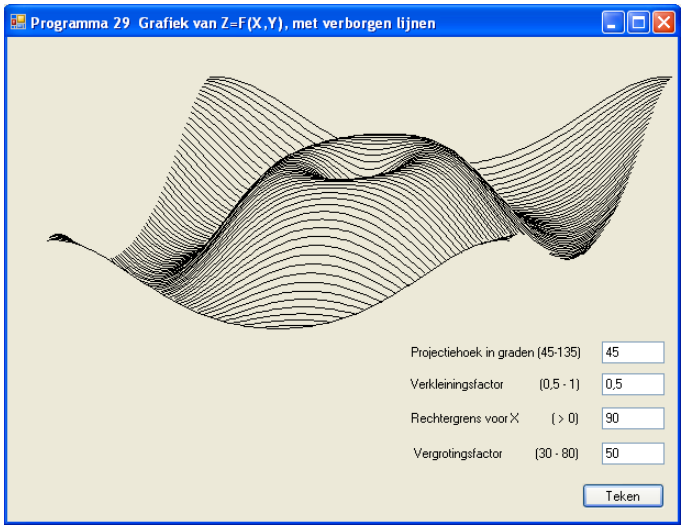
```
1000 R=SQR(X*X+Y*Y)
1010 Z=K1*COS(R/16) 'RESPECT. Z=K1*SIN(R/16)
1100 RETURN
```

```
Dim R As Double = Math.Sqrt(dblX * dblX + dblY * dblY)
Return intK1 * Math.Cos(R / 16) 'Respect. Return intK1 * Math.Sin(R / 16)
```

U kiest: W=45, K=0,5, A=90, K1=50.



$$z = \cos(\frac{r}{16})$$



$$z = \sin(\frac{r}{16})$$

Met dit programma hebben we talrijke andere grafieken gemaakt. Elke keer als u een ‘leuke functie’ tegenkomt, kunt u meteen de daarbij horende draai symmetrische figuur maken.

Tot slot volgt nog een programma voor het tekenen van een fraaie viertoppige grafiek. U bent hem misschien wel eens in een computertijdschrift tegengekomen. Om de snelheid op te voeren is de sub-routine 1000 in het hoofdprogramma opgenomen. U kunt dit natuurlijk ook in alle andere programma's doen.

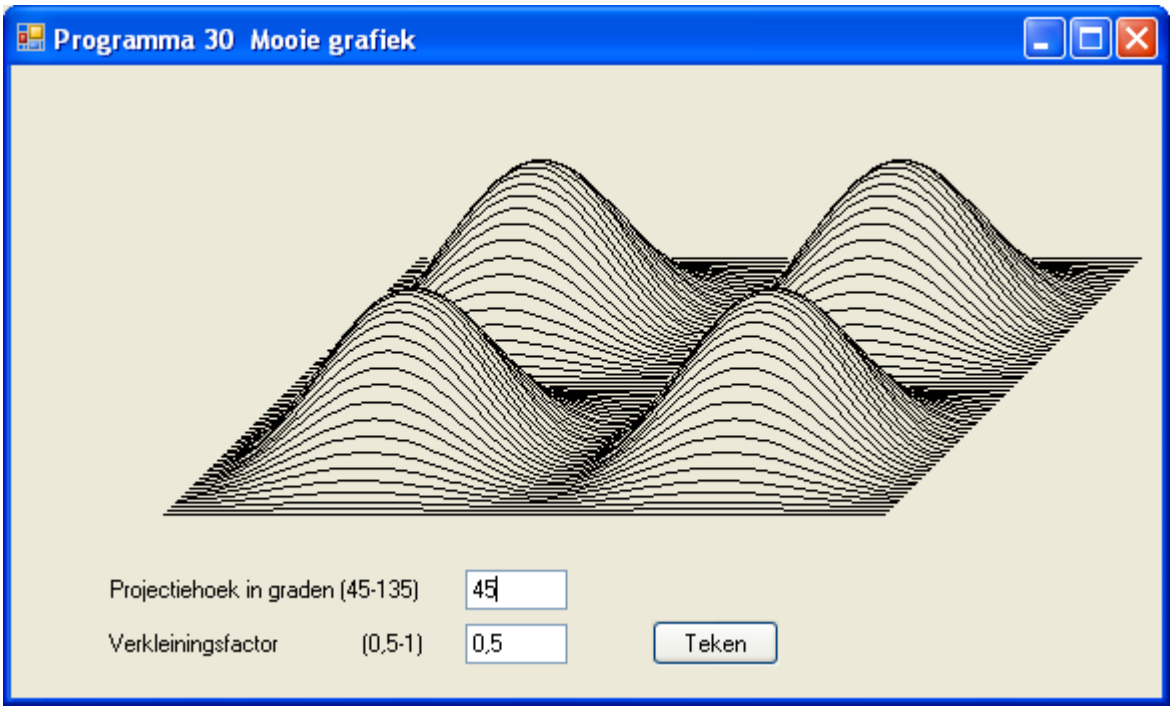
```
100 '            programma 30      MOOIE FUNCTIE
110 CLEAR ,19202 : SCREEN 105,,3,3
120 CLS: KEY OFF
130 INPUT "PROJECTIEHOEK IN GRADEN (45-135)"; W
140 INPUT "VERKLEININGSFACTOR       (0.5-1)"; K
150 U=320 : V=160 : H=.5 : RD=4*ATN(1)/180
160 C=K*COS(W*RD) : S=K*SIN(W*RD)
170 DX=3 : DY=5 : K1=20
180 PI=4*ATN(1)
190 DIM H(320)
200 FOR L=0 TO 320 : H(L)=1000 : NEXT L
210 CLS
220 FOR YY= -180 TO 180 STEP DY
230    M1=COS(YY*2*PI/180 - PI) + 1
240    FOR XX= -180 TO 180 STEP DX
250       M2=COS(XX*2*PI/180-PI)+1: Z=K1*M1*M2
260       XG=INT(U+XX+C*YY+H):YG=INT(V-S*YY-Z+H)
270       IF XX > -180 THEN 310
280       F1=0 : L=INT(XG/DX)
290       IF YG <= H(L) THEN F1=1 : H(L)=YG
300       X1=XG : Y1=YG : GOTO 360
310       F2=0 : L=INT(XG/DX)
320       IF YG <= H(L) THEN F2=1 : H(L)=YG
330       X2=XG : Y2=YG
340       IF F1*F2=1 THEN LINE (X1, Y1)-(X2,Y2),1
350       X1=X2 : Y1=Y2 : F1=F2
360    NEXT XX
370 NEXT YY
380 A$=INKEY$: IF A$="" THEN 380
390 CLS: KEY ON: END
```

```
Public Class frmProg30
    Private Const U As Integer = 320
    Private Const V As Integer = 160
    Private Const H As Double = 0.5

    Private Sub frmProg30_Paint(…, ByVal e As …PaintEventArgs) …
        If Not (txtW.Text() = "" And txtK.Text() = "") Then
            Dim W As Integer = CInt(txtW.Text())
            Dim K As Double = CDbl(txtK.Text())
            Dim RD As Double = 4 * Math.Atan(1) / 180
            Dim C As Double = K * Math.Cos(W * RD)
            Dim S As Double = K * Math.Sin(W * RD)
            Dim DX As Byte = 3, DY As Byte = 5
            Dim K1 As Integer = 20
            Dim arrayH(320) As Integer
            Dim F1 As Byte = 0, F2 As Byte = 0
            Dim X1 As Integer = 0, X2 As Integer = 0
            Dim Y1 As Integer = 0, Y2 As Integer = 0
            Dim L As Integer = 0
            For L = 0 To 320
                arrayH(L) = 1000
            Next
            For YY As Integer = -180 To 180 Step DY
```

```
                Dim M1 As Double = Math.Cos(YY * 2 * Math.PI / 180 - Math.PI) + 1
                For XX As Integer = -180 To 180 Step DX
                    Dim M2 As Double = Math.Cos(XX * 2 * Math.PI / 180 - Math.PI) + 1
                    Dim Z As Double = K1 * M1 * M2
                    Dim XG As Integer = Int(U + XX + C * YY + H)
                    Dim YG As Integer = Int(V - S * YY - Z + H)
                    If XX > -180 Then
                        F2 = 0 : L = Int(XG / DX)
                        If YG <= arrayH(L) Then
                            F2 = 1 : arrayH(L) = YG
                        End If
                        X2 = XG : Y2 = YG
                        If F1 * F2 = 1 Then
                            e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                        End If
                        X1 = X2 : Y1 = Y2 : F1 = F2
                    Else
                        F1 = 0 : L = Int(XG / DX)
                        If YG <= arrayH(L) Then
                            F1 = 1 : arrayH(L) = YG
                        End If
                        X1 = XG : Y1 = YG
                    End If
                Next
            Next
        End If
    End Sub

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



$$z = (\cos(\frac{2\pi x}{180} - \pi) + 1)(\cos(\frac{2\pi y}{180} - \pi) + 1)$$

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

FreeBASIC en de editor FbEdit.

FreeBASIC is een project dat uit een set gecreëerde tools bestaat, geconstitueerd met een compiler, een GNU gebaseerde assembler, een linker en met archieven, en ondersteunt uitvoerbare library's, inclusief een software gebaseerde grafische library. Tegenwoordig ondersteunt de compiler het bouwen voor i386 gebaseerde architecturen in DOS, Linux, Windows en Xbox platformen. Het project bevat ook dunne bindingen (header files) voor wat populaire 3D persoonlijke library's zoals de C runtime library.

Over FreeBASIC is er veel informatie te vinden op de website http://www.freebasic.net.

FreeBASIC is alleen maar een compiler, verder niets. Gelukkig is er ook een editor geschreven, nagenoeg in FreeBASIC, met een IDE.
Omdat de editor geschreven is in FreeBASIC, bevat ook FbEdit alle tools zoals hierboven staan aangegeven. In eerste instantie lijkt de code veel op Liberty BASIC vanwege de manier hoe de eventcode gemaakt moet worden (werken met handles), maar wat we vooral in FreeBASIC vinden is de C structuur. Net zoals we met *include* een library moeten invoegen in C, zo moeten we dat ook doen in FreeBASIC.

Een '*Hallo Wereld*' met FreeBASIC.
FreeBASIC is dus alleen maar de compiler. We moeten dus zelf voor een teksteditor zorgen om FreeBASIC code in te kunnen voeren, als we dus zonder FbEdit willen programmeren.

We hebben voldoende aan het Kladblok. Open deze en voer onderstaande regel in:

```
Print "Hallo Wereld!"
```

Kies eerst als type 'Alle bestanden'.
Sla dan het bestand op met de extensie '.bas', bijvoorbeeld: 'Hallo.bas'.
Als u niet eerst het type veranderd in 'Alle bestanden' dan is de kans groot dat Kladblok het bestand opslaat als 'Hallo.bas.txt' en dat is niet de bedoeling. Dit is ook een handige tip als u databestanden voor uw programma's moet opslaan. In de volgende nieuwsbrief kom ik daar op terug.

Zorg ervoor dat FreeBASIC op uw computer staat en door de Command Prompt (console scherm) bekend is. Ga naar de directory waar het .bas bestand opgeslagen is en voer onderstaande opdracht in:

```
fbc Hallo.bas
```

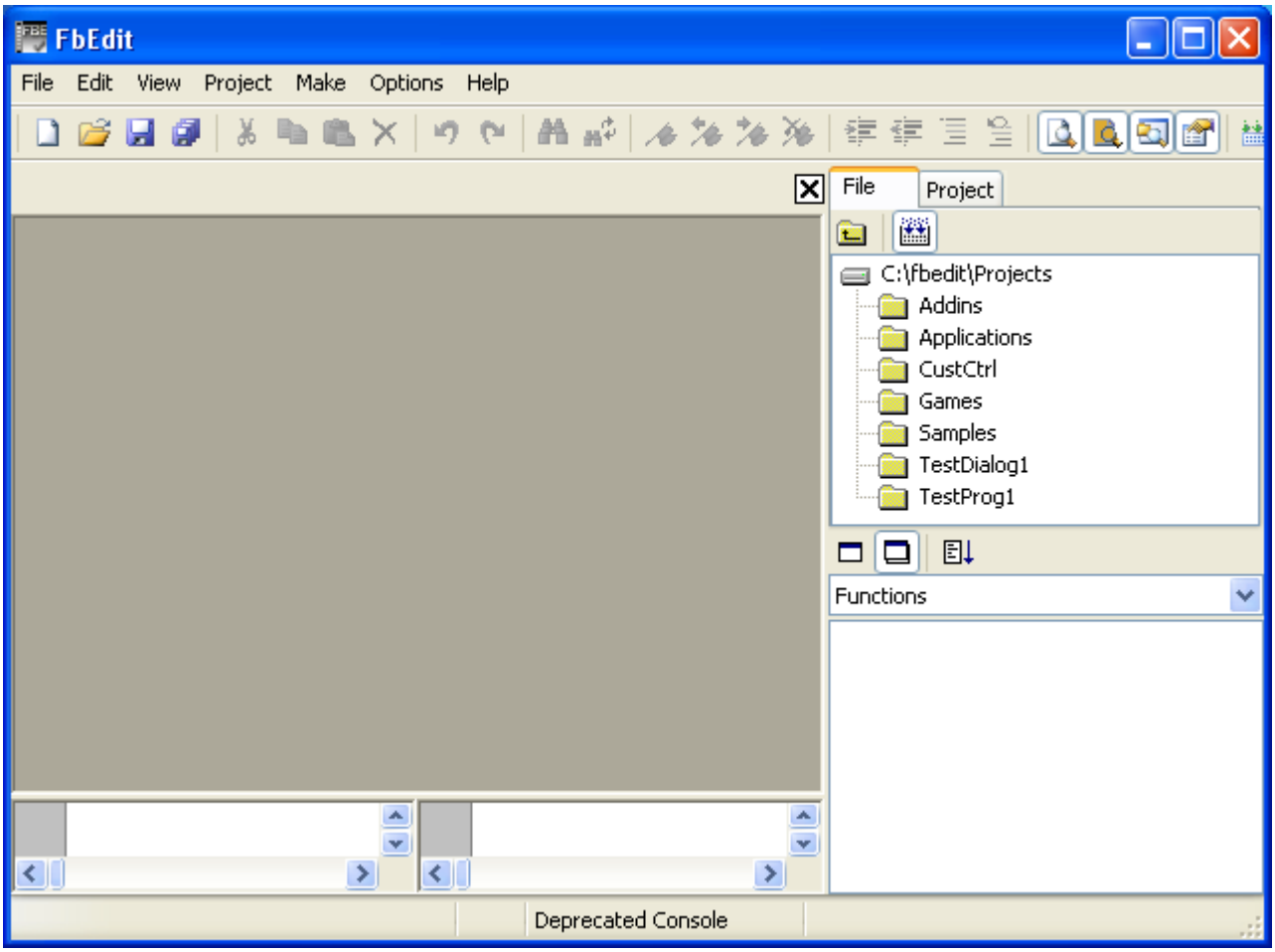
De drie letters *fbc* heten FreeBasicCompiler. De compiler zal dan ook in dezelfde map als waar Hallo.bas staat de code compileren en het uitvoerbare Hallo.exe bestand creëren.

Start nu het uitvoerbare bestand en de onderstaande regel zal worden weergegeven:

```
Hallo Wereld!
```

Zoals het nu lijkt: niet veel bijzonders! Inderdaad, er lijkt ook niet veel anders te zijn met FreeBASIC. Het is daardoor mogelijk om ook gewone BASIC programma's te schrijven, bijvoorbeeld QuickBASIC. FreeBASIC accepteert normale BASIC statements zoals Print, Dim en If…Then. In de volgende nieuwsbrief laat ik voor beginners en ervaren BASIC gebruikers meer zien wat we voor code kunnen gebruiken in FreeBASIC.

FbEdit, de IDE editor.
Alleen maar de FreeBASIC compiler gebruiken kan lastig zijn en irritaties geven. Het is beter om FbEdit te gebruiken. Een geïntegreerde ontwerp omgeving (Integrated Development Environment, IDE). De compiler ondersteunt meerdere BASIC dialecten met een eigenaardige control-flow die eruit ziet als een C structuur. Hieronder ziet u de editor.

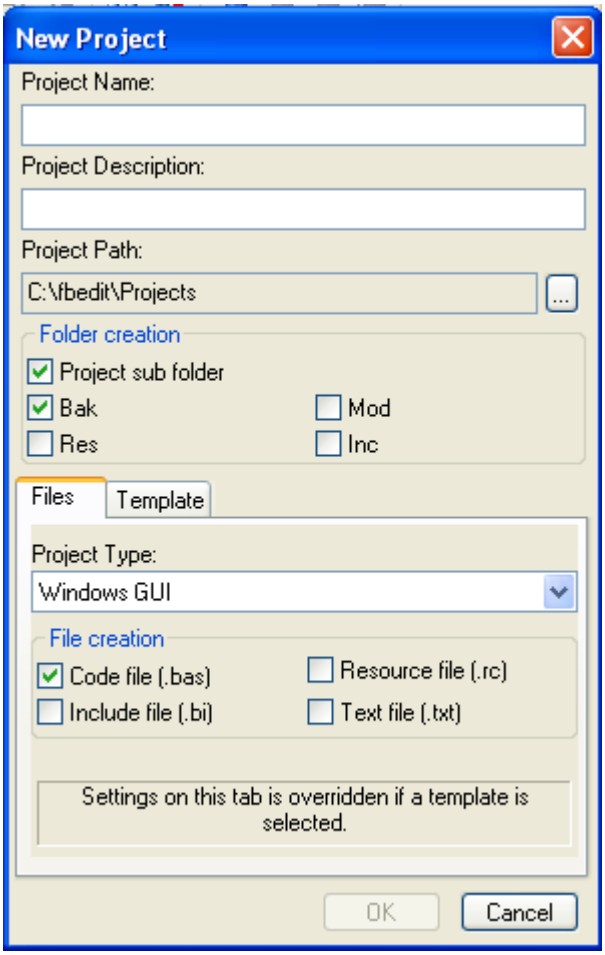


FbEdit heeft al kant-en-klare projecten die u voor uw eigen projecten kunt gebruiken. Deze zijn aanwezig in de *File*-lijst die aan de rechterkant staat. Zo is er al een Alien project aanwezig in de map Games. De code die zover voor het project nodig is staat er allemaal bij.

Elk FreeBASIC project bestaat tenminste uit de volgende bestanden:
- Bas in deze bestanden komen alle bestuurbare codeblokken
- Bi hier worden alle constanten, (object)typen en (gedeelde) variabelen gedeclareerd de letter *i* staat voor *include*
- fbp dit is het feitelijk projectbestand
- Rc Rc staat voor Resources; hier worden alle (dialoog)formulieren ontwikkeld

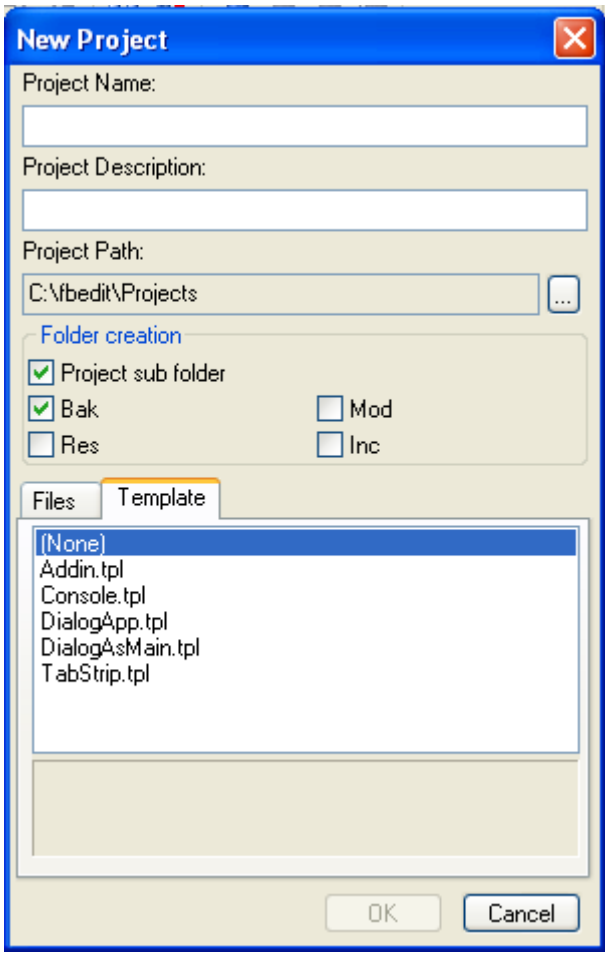
Zouden we direct zo'n project openen, dan zou u met een open mond en verbaasd zien hoe FreeBASIC eruit ziet. Niet te filmen, zou u zeggen. Resultaat: een C geraamte aangekleed met BASIC code!

Kies File – New Project.
Onderstaand venster verschijnt.



De FreeBASIC compiler vraagt veel gegevens aan u voordat er een nieuw project gemaakt kan worden.

- Project Name:
  Voer hier een naam in hoe het project moet heten.
- Project Description:
  Voer hier een omschrijving in om wat voor project het gaat.
- Project Path:
  Hier kunt u de map kiezen waar het project in opgeslagen moet worden. Klik op de 'drie punten' knop aan de rechterkant als u een andere map wilt.
- Folder creation
  Vink hier aan welke onderdelen u in het project wilt gebruiken. De twee belangrijkste zijn al aangevinkt. Hieronder staan de omschrijvingen:
  - Project sub folder
    Zorgt voor een eigen project map in de gekozen Project Path.
  - Bak
    Maakt backup bestanden aan van het project in een backup map.
  - Res
    Maakt een Resource map aan voor al uw resources, zoals de formulieren.
  - Mod
    Maakt een Module map aan voor al uw module bestanden.
  - Inc
    Maakt een Include map aan voor al uw include bestanden, zoals de (object)typen.



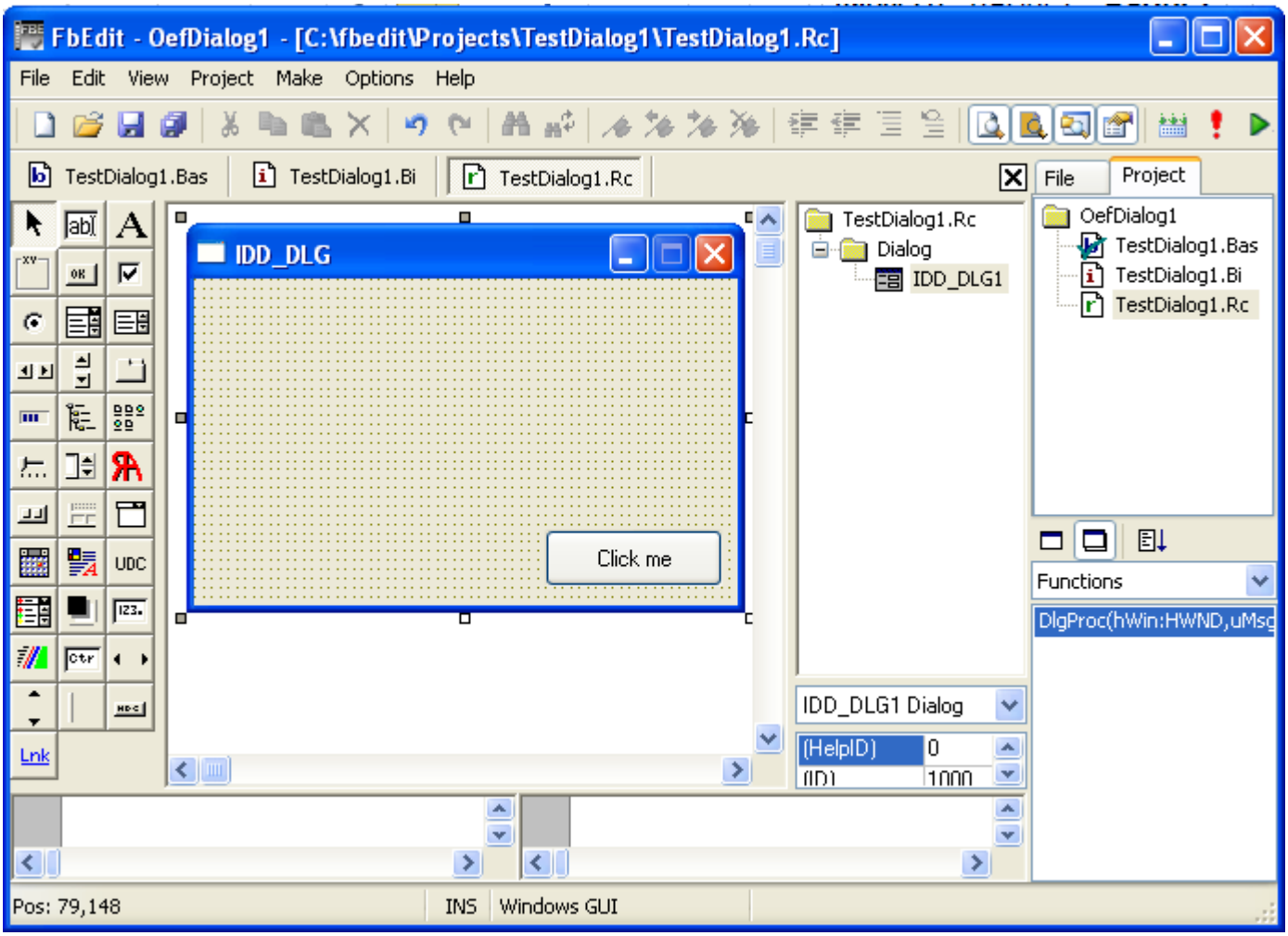
Tabblad Files

Hier kunt u kiezen wat voor type project u wilt maken. Er is een hele lijst vol met soorten projecttypes, maar normaal gesproken gebruiken we de *Windows GUI*, de standaard applicatie.
Vink ook de onderdelen aan die u in het project wilt maken. De extensies van de onderdelen zijn op de vorige pagina al besproken. Er staat echter een waarschuwing bij, onderaan de opties:

Instellingen op dit tabblad zullen overschreven worden als u een template kiest.

Als u de instellingen van dit tabblad wilt gebruiken, zorg er dan voor dat er geen template is gekozen. Hiernaast ziet u nogmaals het 'New Project' formulier, maar dan met de tabblad *Template*.
Elke template heeft een eigen File creation, vandaar die waarschuwing.

Een TestDialog1 project.
Hieronder kunt u een geopend project zien genaamd ‘TestDialog1’. Deze staat kant-en-klaar rechts in de File-lijst. Om in de ontwikkelomgeving te komen, klikt u op ‘TestDialog1.Rc’ of als alle tabbladen van het project aanwezig zijn, kunt u ook op het tabblad ‘TestDialog1.Rc’ klikken.



Als u dubbelklikt op ‘TestDialog1.Bi’ of u kiest het tabblad, dan ziet u het codevenster van het formulier verschijnen met onderstaande code.

```
#define IDD_DLG1 1000
#define IDC_BTN1 1001

Dim Shared hInstance As HMODULE
```

Is dit BASIC? Ja en nee. Het is C versus BASIC, oftewel BASIC in C. De twee #define regels zijn *compiler directives*. Hiermee worden ID sleutels voor dialoogformulieren (IDD) en componenten (IDC) ingesteld. Elke dialoog of component heeft zijn eigen unieke ID sleutel.

De Dim regel definieert een instantie naar de hoofdmodule (HMODULE). Het sleutelwoord Shared zorgt ervoor dat de instantie gedeeld kan worden, anders zal de objectvariabele alleen in dit codevenster zichtbaar zijn.

Alles moet aan Windows als berichten doorgegeven worden zodat er actie ondernomen kan worden mocht de gebruiker bijvoorbeeld op de knop klikken, of het formulier vergroten of verkleinen. Zie de code hieronder. Klik op het tabblad ‘TestDialog1.Bas’.

Lange coderegels die hier niet op een regel passen moet u niet afbreken.

```
/'
     Dialog Example, by fsw

     compile with:  fbc -s gui dialog.rc dialog.bas

'/

#Include Once "windows.bi"

#Include "TestDialog1.bi"

Declare Function DlgProc(ByVal hWin As HWND, ByVal uMsg As UINT, ByVal
wParam As WPARAM, ByVal lParam As LPARAM) As Integer

'''
''' Program start
'''

     ''
     '' Create the Dialog
     ''
     hInstance=GetModuleHandle(NULL)
     DialogBoxParam(hInstance, Cast(ZString Ptr,IDD_DLG1), NULL, @DlgProc,
NULL)
     ''
     '' Program has ended
     ''

     ExitProcess(0)
     End

'''
''' Program end
'''
Function DlgProc(ByVal hWin As HWND,ByVal uMsg As UINT,ByVal wParam As
WPARAM,ByVal lParam As LPARAM) As Integer
     Dim As Long id, Event, x, y
     Dim hBtn As HWND
     Dim rect As RECT

     Select Case uMsg
          Case WM_INITDIALOG
               '
          Case WM_CLOSE
               EndDialog(hWin, 0)
               '
          Case WM_COMMAND
               id=LoWord(wParam)
               Event=HiWord(wParam)
               Select Case id
                    Case IDC_BTN1
                         EndDialog(hWin, 0)
                         '
```

```
            End Select
        Case WM_SIZE
            GetClientRect(hWin,@rect)
            hBtn=GetDlgItem(hWin,IDC_BTN1)
            x=rect.right-100
            y=rect.bottom-35
            MoveWindow(hBtn,x,y,97,31,TRUE)
            '
        Case Else
            Return FALSE
            '
    End Select
    Return TRUE

End Function
```

Elke formulier zal in de Resource lijst terecht komen. Het enige wat u hoeft te doen is, alle formulieren en componenten in het Bi codevenster definiëren, zie pagina 26. U moet de unieke nummers, dat elk formulier en elke component krijgt, in de *#define* regels plaatsen. Op pagina 26 ziet u dat het dialoogformulier IDD_DLG1 is gedefinieerd met nummer 1000. Stel dat u in de Resource lijst een formulier toevoegt, dan zal het tweede formulier de waarde 1100 krijgen.
FbEdit zal niet automatisch voor u de code bijwerken. U moet zelf voor het tweede dialoogformulier een nieuwe #define regel toevoegen. Onderstaande nieuwe regel wordt dan:

```
#define IDD_DLG2 1100
```

Plaatst u er een component op zoals een label en eventueel ook nog een editbox erbij dan moeten onderstaande #define regels worden toegevoegd.

```
#define IDC_STC1 1101
#define IDC_EDT1 1102
```

Wijzig daarna niet meer zelf de nummers. Ze moeten overeenkomen met de unieke nummers die in de object explorer van de componenten te vinden zijn.

Maar laten we niet teveel hier op hameren. Eerst zou u wel eens willen weten wat FreeBASIC ons te bieden heeft. Daarom zal ik in de komende nieuwsbrieven veel laten zien over FreeBASIC.
Wilt u toch meer weten over de IDE editor, bekijk dan eens onderstaande website:

http://www.oby.ro/rad_asm/fbedit/index.html.

U kunt daar de editor ook downloaden.

Marco Kurvers

BASIC, Delphi en de control flow.

Met de control flow kunnen we elk programma goed gestroomlijnd structureren en leesbaar maken. We kunnen nu al met BASIC gestructureerd programmeren en we ‘denken’ dat we dan het programma prima geschreven hebben.
Hoe ver BASIC ook gekomen is, het blijft een vieze taal! Niet mee eens? Als u vindt dat BASIC juist een zeer goede gestructureerde taal is, dan vind ik dat prima. In ieder geval hoeft u met BASIC listings niet bij Delphi programmeurs aan te kloppen. Ze hoeven het woord BASIC maar één keer te ho-

ren en de antwoorden zijn gelijk: ‘Ga weg met je BASIC!’, ‘Het is een niet gestroomlijnde taal!’, ‘Ach BASIC, daar maak je geen goede applicaties mee!’

We hebben Liberty BASIC, FreeBASIC, Visual Basic 6 en Visual Basic .NET. Ik heb deze eruit gekozen, want met deze vier kan de beste gestructureerde code geschreven worden. Hoewel PowerBASIC zeer krachtig is en net als FreeBASIC ook compiler directives kent, is het geen object gestructureerd BASIC dialect.

In dit hoofdstuk wil ik eens Delphi naar voren halen, want Delphi biedt een mogelijkheid waar een BASIC programmeur over zou dromen. Ook al lijken, gezien uit het tabel in het hoofdstuk ‘BASIC en andere programmeertalen.’, de mogelijkheden minder goed dan wat BASIC heeft, toch missen we een belangrijk aspect in BASIC waardoor de Delphi code er zo ‘keurig’ uitziet.

Oh, is de BASIC code rommelig dan?!
Ja en nee, het ligt eraan *hoe* u uw programma schrijft. Echter, in Delphi kunnen we dat zelf niet meer bepalen, en dat komt doordat de control flow, dus de codestructuur, strenger door de compiler gecontroleerd wordt. We kunnen bijvoorbeeld niet zomaar ergens een variabele gaan declareren. In Delphi geldt daar een bepaalde plaats voor. In BASIC maakt het niet uit waar we een variabele declareren. Conclusie: we krijgen een gebouw met een dak van onderen en de deur van boven of de kamer van onderen, het dak in het midden en de deur van boven. BASIC zal deze verkeerde control flow niet zien, de compiler van Delphi wel. Dat is waarom zij (de programmeurs) BASIC een vieze taal vinden.

De verschillen.
Het is niet de bedoeling hele Delphi projecten te laten zien en daarvan een vertaling naar BASIC eronder te zetten. Ik wil u als BASIC lezer niet het gevoel geven dat de nieuwsbrief plotseling deels Delphi is geworden. Ik wil u alleen maar laten zien wat voor control flow codeblokken er in Delphi zijn en hoe zulke blokken in BASIC eruit zien. Wat wilt u liever, hier het alleen over BASIC hebben en dat u naar een converteermethode op internet moet zoeken of dat ik u het beide hier kan uitleggen? Natuurlijk, de tweede keuze is fijner.

Bovendien wil ik u echt vertellen dat tegenwoordig zo’n uitleg erg belangrijk is omdat meerdere programmeertalen samen de applicatie vormen en (haast) niet meer met één programmeertaal gedaan wordt.

Variabelen, toekenningen, beslissingen en lussen.
Vindt u code op Internet, in een boek of waar dan ook, die geschreven is in Delphi en u zou het in BASIC willen hebben, dan kan onderstaand tabel u misschien helpen.

! Wat rood staat aangegeven is optioneel.

Delphi	**BASIC**
`var breedte: Integer;`	`Dim breedte As Integer`
`var decGetal: Real;`	`Dim decGetal As Double`
`A := 10;`	`A = 10`
`var Tekst: string[30];`	`Dim Tekst As String * 30`
`LTekst := LeftStr(Tekst, 5);`	`LTekst = Left$(Tekst, 5)`
`MTekst := MidStr(Tekst, 6, 3);`	`MTekst = Mid$(Tekst, 6, 3)`
`RTekst := RightStr(Tekst, 3);`	`RTekst = Right$(Tekst, 3)`

`var`
`  arrVar: array[1..10] of`
`  Integer;`
	`Dim arrVar(10) As Integer`

`arrVar[4] := 50;`	`arrVar(4) = 50`
`Inc(I, 2);`	`I = I + 2`
`Dec(I, 2);`	`I = I - 2`
`c := Succ('F'); // result: 'G'`	`c = Chr$(Asc("F") + 1)`

`c := Pred('F'); // result: 'E'`	`c = Chr$(Asc("F") - 1)`
`n := Succ(m); // result: m + 1`	`n = m + 1`
`n := Pred(m); // result: m - 1`	`n = m - 1`
`if (n > m) and (m = 0) then` `  m := 1;`	`If n > m and m = 0 Then m = 1`

`if (conditie) then`
`begin`
`  // statements`
`end else`
`begin`
`  // statements`
`end;`
	`IF conditie THEN BEGIN`
	`  REM statements  Commodore 128`
	`BEND: ELSE BEGIN  BASIC V7.0`
	`  REM statements`
	`BEND`
	`If conditie Then`
	`  ' statements  Andere BASIC`
	`Else  versies`
	`  ' statements`
	`End If`

`for I := 1 to 20 do`
`  J[I] := I * 5;`
	`For I = 1 To 20`
	`  J(I) = I * 5`
	`Next I`

`for I := 20 downto 1 do`
`  J[I] := x * 40 + y;`
	`For I = 20 To 1 Step -1`
	`  J(I) = x * 40 + y`
	`Next I`

`while (n < m) do`
`begin`
`  // statements`
`  n := n + 1;`
`end;`
	`Do While n < m`
	`  ' statements`
	`  n = n + 1`
	`Wend Alleen oude versies, VB6 en VBA`
	`End While Alleen VB.NET versies`
	`Loop Alle versies, alleen met Do While`

`repeat`
`  Inc(n, 1);`
`  // statements`
`until (n >= m);`
	`Do`
	`  n = n + 1`
	`  ' statements`
	`Loop Until n >= m`

`function GetNaam(index:`
`  Integer):`
`  string;`
`var vnaam: typenaam;`
`begin`
`  // statements`
`  result := strWaarde`
`end;`
	`Function GetNaam(ByVal index As Integer)`
	`  As String`
	`  Dim vnaam As typenaam`
	`  ' statements`
	`  GetNaam = strWaarde Alleen oude versies, VB6 en VBA`
	`  Return strWaarde Alleen VB.NET versies`
	`End Function`

`procedure WatAnders(index:`
`  Integer;`
`  var s: string);`
`var vnaam: typenaam;`
`begin`
`  // statements`
`  s := strWaarde`
`end;`
	`Sub WatAnders(ByVal index As Integer, _`
	`  ByRef s As String)`
	`  Dim vnaam As typenaam`
	`  ' statements`
	`  s = strWaarde`
	`End Sub`

Wilt u meer weten? U kunt mij vragen sturen over gebruik van meerdere programmeertalen en hoe we de code samen tot een applicatie kunnen brengen. U kunt mij ook vinden in De Meern (1ste week), in Amersfoort (3de week) en in Barneveld (2de en 4de weken) per maand.

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.
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.hcc-net.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	Jan v.d. Linden				j.vd.linden@kader.hcc.nl
QuickBasic					
Visual Basic voor Windows	Jeroen v. Hezik	ma. t/m zo.	19-21	(0346)	j.a.van.hezik@kader.hcc.nl
Visual Basic .NET	Marco Kurvers	do. t/m zo.	19-22	214131 (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