Nieuwsbrief

19de jaargang september 2012 **Nummer 3**

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 Excel 2010 zal ik u laten zien hoe de besturing werkt. Belangrijke onderwerpen zijn: verwijzingen, bereiken en werken met bereiken voor gevorderden. Ook laat ik u zien hoe kringverwijzingen ontstaan.

Het deel over grafisch tekenen met LOGO en het vertalen in BASIC code was zeer interessant en leerzaam. U kon vooral merken dat alleen maar met het opgeven van andere parameterwaarden heel veel soorten tekeningen tevoorschijn kwamen. Daarbij deed ik er nog eens een schepje bovenop om u de mogelijkheid te laten zien met de *While* lus.
In dit deel laat ik u een ander onderwerp zien: Educatieve toepassingsprogramma's. Wat dat met 'Grafisch programmeren in GW-BASIC' te maken heeft, kunt u in dit deel zien.

Hoe zit het tekenen eigenlijk in elkaar? Hoe worden de tekenmaterialen gebruikt? Wat is het Canvas tekengebied? Is dat het gebied waar alles op getekend wordt, zoals de componenten? Het zijn allemaal vragen die ik zal beantwoorden met behulp van Delphi. Ook zal BASIC erbij te pas komen, maar deze tekentechniek is alleen bestemd voor het moderne Basic.NET versie, o.a. VB 2008.
Dit onderwerp kunt u ook vinden op de nldelphi pagina, zie hieronder, maar uiteraard zonder Basic, mijn tips en extra alinea's.
http://www.nldelphi.com/cgi-bin/articles.exe/ShowArticle?ID=22092

Marco Kurvers

BASIC leren – PowerBASIC (5).

Hoofdstuk 6 – Gegevensstructuren gebruiken

Arrays zijn nuttig wanneer u een lijst van verschillende variabelen wilt samenbrengen tot één eenheid. Bijvoorbeeld, tien testscores of tien studentennamen. Maar wat als u verschillende ongerelateerde gegevenstypes wilt bewaren en *die* als een eenheid wilt behandelen? Dat is nou waar gebruiker gedefinieerde types mee naar voren komen. Wanneer u een gebruiker gedefinieerd type definieert, maakt u zoals ware een sjabloon aan voor een nieuwe variabele type. Nadat u er een gecreëerd hebt, kunt u zoveel variabelen definiëren van dat nieuwe type als u wenst, en ook sinds gebruiker gedefinieerde types geassocieerd kunnen worden met een willekeurige bestandsbuffer. Dit biedt u een geheel nieuwe manier om toegang te krijgen tot uw willekeurige bestanden. Pas later in hoofdstuk 9 wordt daar uitgebreid over gesproken.

PowerBASIC biedt ook *flex strings* aan, welke u kunt gebruiken om dynamische gegevensstructuren te creëren (tijdens uitvoertijd). Flex strings worden geïntroduceerd in hoofdstuk 9, maar zullen alvast meer gedetailleerd in dit hoofdstuk uitgelegd worden.

Wat zijn gebruiker gedefinieerde types?

Een PowerBASIC gebruiker gedefinieerd type is vergelijkbaar met een C **struct** of een Pascal **record**. De elementen van een gebruiker gedefinieerd type mogen elk gegevenstype van PowerBASIC hebben, uitgezonderd variabele-lengte strings en arrays. De enige andere manier die u niet in een gebruiker gedefinieerd type kunt toepassen is een referentie naar het type die u definieert. Met andere woorden, een *StudentRecord* kan geen veld van het type *StudentRecord* hebben.

Om een idee te krijgen hoe de kracht van een gebruiker gedefinieerd type werkt, stel dan eens voor dat u een leraar bent die behoefte heeft aan een programma voor het bijhouden van gegevens van zijn studenten. Aangezien uw school een zeer krappe begroting heeft (en welke school heeft dat niet deze dagen?) wilt u het programma zelf schrijven in PowerBASIC. Van elke student in de klas hebt u de volgende informatie nodig:

- De naam van de student.
- Een studentnummer.
- Een mail adres.
- Naam en telefoonnummer van de contactpersoon voor eventuele noodgevallen.
- De relatie van de contactpersoon tot de student.

Op dit moment worden deze records bewaard in een klein bestands bakje. De informatie van elke student wordt op een 3x5” bestandskaart bijgehouden. Hoe kunt u de informatie overbrengen naar de computer?
Simpel, definieer een *StudentRecord* type die alle informatie over een enkele student zal bevatten.

De variabelen die u als gebruiker gedefinieerde types creëert, zijn genoemd als *records* of *recordvariabelen*, want elke variabele van dat type houdt een record in, of één *set van gerelateerde informatie*. De individuele elementen zijn genoemd als *velden*. In het voorbeeld hierboven is elke set van studentinformatie gelijk aan een record, en elk deel van de informatie in dat record (als voorbeeld de achternaam) is een veld.

Gebruiker gedefinieerde types definiëren en gebruiken

De definitie van een gebruiker gedefinieerd type begint met het gereserveerd woord TYPE en eindigt met de sleutelwoorden END TYPE. Daartussen definieert u de namen en gegevenstypes van de elementen (velden) die onderdeel zijn van het nieuwe type. Als voorbeeld:
.

```
TYPE StudentRecord
     Achternaam AS STRING * 20     ' Een karakterstring van 20 posities
     Voornaam AS STRING * 15       ' Een karakterstring van 15 posities
     IDnum AS LONG                 ' Student ID, een lange integer
     Contact AS STRING * 30        ' Persoon voor contact in noodgevallen
     Contactfoon AS STRING * 14    ' Hun telefoonnummer
     ContactRel AS STRING * 8      ' Relatie van het contact tot de student
     GemiddeldResultaat AS SINGLE  ' Single-precisie percentage
END TYPE
```

Vergeet niet, de definitie van een gebruiker gedefinieerd type reserveert geen geheugen voor het opslaan van gegevens van dat type. Er wordt eerder een sjabloon voor het nieuwe type *StudentRecord* gedefinieerd. Wanneer de compiler een gedeclareerd (of gecreëerd) statement tegenkomt met een variabele van het nieuwe type, zal het “weten” hoeveel bytes van opslag voor de variabele opzij gezet moet worden.
Om dit nieuwe type te kunnen gebruiken, moet u het declareren met variabelen van dat type met het DIM statement:

```
DIM Student AS StudentRecord
```

Toegang tot de velden van een gebruiker gedefinieerd type

Om met de velden binnen een record variabele te kunnen werken, moet u de veldnaam van de variabelennaam scheiden met een punt. Hier zijn voorbeelden die de *Student* variabele gebruiken zoals deze gedeclareerd is in bovenstaande DIM statement:

```
PRINT Student.Achternaam
PRINT "Id nummer is: "; Student.IDnum
Student.Voornaam = "Bob"
Student.Achternaam = "Smith"
PRINT Student.Achternaam;", "; Student.Voornaam
```

Hoewel u in PowerBASIC nog steeds punten binnen eenvoudige variabelennamen gebruiken kunt, zou het een heel goed idee zijn om uw gebruik van de punt juist te beperken tot de toegang van de recordvariabelen. PowerBASIC kan zelf uitvinden wat u bedoeld met *User.Id*, maar wat als verschillende mensen het programma lezen? Is het een veld binnen een recordvariabele genoemd naar *User* of een simpele variabele genoemd naar *User.Id*? Waarom de dingen moeilijker maken als het makkelijker kan?

Gebruiker gedefinieerde types nesten

De velden binnen een gebruiker gedefinieerd type kunnen opmaakt worden met andere gebruiker gedefinieerde types. Net zoals een aantal Chinese dozen, zit in elke doos weer een kleinere doos, met andere woorden: u kunt een gebruiker gedefinieerd type nesten binnen een andere. Het eindresultaat is dat u gegevensstructuren creëert die een hiërarchie hebben, vergelijkbaar met de directory-structuur van uw harde schijf.

In plaats van studentennamen als twee afzonderlijke velden op te slaan, kunnen we in plaats daarvan ook wel een type *NaamRec* als volgt definiëren:

```
TYPE NaamRec
     Achter AS STRING * 20
     Voor AS STRING * 15
     Initiaal AS STRING * 1
END TYPE
```

Vervolgens kunnen we, wanneer we ons *Student* recordtype definiëren, het veld met de naam van de student als *NaamRec* definiëren:

```
TYPE StudentRecord
     Helenaam AS NaamRec
     IdNum AS LONG
     Contact AS NaamRec
     Contactfoon AS STRING * 14
     ContactRel AS STRING * 8
     GemiddeldResultaat AS SINGLE
END TYPE
```

U kunt met dit idee natuurlijk een stap verder gaan en nog meer delen van het student record als geneste records definiëren, bijvoorbeeld een *ContactRecord* of een Telef*oonRecord*, maar ik laat dat helemaal over aan u. Om toegang te krijgen tot geneste recordvelden is het simpel te weten extra puntnotaties te gebruiken, net zoals de backslash (\) gebruikt wordt om subdirectorynamen in een pad te scheiden (bijvoorbeeld C:\PB\PROGRAM). De punt binnen een recordvariabelennaam wordt gebruikt om de veldelementen van het basistype te scheiden, bijvoorbeeld:

```
StudentRecord.Helenaam
```

refereert naar het *Helenaam* veld van het type *NaamRec* binnen het *StudentRecord*, en

```
StudentRecord.Helenaam.Voor
```

refereert naar het sub-veld *Voor* binnen het *Helenaam* veld.

U kunt gebruiker gedefinieerde types zo diep mogelijk nesten als u wilt, zolang de gehele gebruikte naam, die naar een veld refereert, binnen de maximum lengte van 255 karakters ligt. In de praktijk echter, zou u waarschijnlijk niet buiten twee of ten hoogste drie niveaus willen nesten. Bovendien zou het verder onhandiger en lastiger worden om het te onthouden, en u hebt meer kans om typefouten te maken.

Arrays gebruiken van gebruiker gedefinieerde types

U kunt arrays creëren van gebruiker gedefinieerde types net zoals u arrays kunt creëren van integers of strings of elk ander PowerBASIC gegevenstype, bijvoorbeeld:

```
DIM Klas(1:30) AS StudentRecord
```

Om toegang te krijgen tot de elementen van de *Klas* array kunt u subscripts gebruiken, net zoals u doet met elke andere array. Als voorbeeld is het derde student record *Klas(3)*. De scheidingspunt en de veldnaam vervolgen het array subscript:

```
PRINT Klas(3).Helenaam.Voor
```

drukt de voornaam af van de derde student in de klas array. Denk hieraan: de array is gemaakt uit elementen van het type *StudentRecord*, dus het subscript behoort tot de naam van de variabele als één geheel.

Gebruiker gedefinieerde types gebruiken met procedures en functies

Procedures en functies kunnen gebruiker gedefinieerde types beheren als elk ander gegevenstype. In deze sectie gaat het over de volgende onderwerpen:

- velden toepassen als argumenten
- records toepassen als argumenten
- record arrays toepassen als argumenten

Velden toepassen als argumenten

Velden in gebruiker gedefinieerde types die als PowerBASIC types gebouwd zijn (INTEGER, WORD, STRING, enzovoort) kunnen toegepast worden in procedures en functies als simpele variabelen. Bijvoorbeeld, geef de gebruiker gedefinieerd type *PatiëntRecord* als volgt:

```
TYPE PatiëntRecord
     Helenaam AS STRING * 32
     BedragSchuld AS DOUBLE
     IdNum AS LONG
END TYPE
DIM Patiënt AS PatiëntRecord
```

en u kunt een procedure *PrintStatement* gebruiken

```
SUB PrintStatement(Id AS LONG, BedragTotSchuld AS DOUBLE)
     ' toegang Id en BedragTotSchuld
     …
END SUB
```

als zo:

```
CALL PrintStatement(Patiënt.IdNum, Patiënt.BedragSchuld)
```

Records toepassen als argumenten

U kunt ook uw procedures en functies schrijven die argumenten accepteren van gebruiker gedefinieerde types. Dit is zeer geschikt als u meerdere argumenten nodig hebt; anders dan een lange argumentenlijst te moeten hebben. U kunt een enkel gebruiker gedefinieerd type toepassen, bijvoorbeeld het *PatiëntRecord* gebruiker gedefinieerd type, beschreven in het vorige deel van deze sectie. U kunt uw *PrintStatement* procedure als volgt schrijven:

```
SUB PrintStatement(Patiënt AS PatiëntRecord)
     ' toegang Patiënt.IdNum en Patiënt.BedragSchuld
     …
END SUB
```

Vervolgens roept u *PrintStatement* aan als zo:

```
CALL PrintStatement(Patiënt)
```

Record arrays toepassen als argumenten

Procedures en functies accepteren ook arrays van records net zoals ze arrays van andere types accepteren. Als u bijvoorbeeld een array *Patiënten* heeft, kunt u een functie schrijven die uit elke patiënt record de inhoud van het BedragSchuld veld optelt. De totale optelling wordt dan door de functie als resultaat teruggegeven:

```
FUNCTION TotaalBedragSchuld(Patiënten() AS PatiëntRecord) AS DOUBLE
     DIM totaal AS DOUBLE

     totaal = 0
     FOR I = LBOUND(Patiënten) TO UBOUND(Patiënten)
          totaal = totaal + Patiënten(I).BedragSchuld
     NEXT

     TotaalBedragSchuld = totaal
END FUNCTION
```

U kunt de functie op de volgende manier aanroepen:

```
DIM Patiënten(1:100) AS PatiëntRecord
…
PRINT "Totale bedrag schuldig:"; TotaalBedragSchuld(Patiënten())
```

Opslag benodigdheden

U kunt de totale benodigde opslag van een variabele van een gebruiker gedefinieerd type bepalen door gebruik te maken van de LEN functie. Houd echter wel rekening mee dat u de naam van een variabele van het type, waar u meer over wilt weten, toepast en niet de naam van het type zelf (zoals gedefinieerd in het TYPE statement). Onthoud dat de naam in het TYPE statement alleen gebruikt wordt als een sjabloon. Om de benodigdheden voor een student record te bepalen, gebruik:

```
RecordLengte = LEN(Student)
```

Om de opslag benodigdheden voor een dynamische array van *Student* records te bepalen:

```
DO
     INPUT "Hoeveel records wilt u: "; NumRecords
     BenodigdGeheugen = LEN(Student) * NumRecords
     IF BenodigdGeheugen > FRE(0) THEN
          PRINT "Niet genoeg geheugen om de array te creëren!"
          PRINT "Voer een kleiner getal in alstublieft."
          NumRecords = 0
     END IF
LOOP UNTIL NumRecords > 0
```

Het adres van een recordvariabele, zoals teruggegeven wordt door de VARPTR functie, is het adres in geheugen van de eerste byte van de gegevens in het record. U kunt ook het startadres van de velden binnen het record terugkrijgen met de VARPTR functie door de volledige veldnaam, als voorbeeld *Sudent.IdNum*, toe te passen.

Unies

Als u ooit geprogrammeerd hebt in Pascal of C, bent u wellicht bekend met het concept van een unie. Een unie is in sommige gevallen vergelijkbaar met een gebruiker gedefinieerd type. Beide hebben gegevensvelden die samengesteld kunnen worden met elk PowerBASIC gegevenstype, inclusief records en andere unies, en ze worden op dezelfde manier gedefinieerd. Het grootste verschil tussen gebruiker gedefinieerde types en unies is dat elk veld binnen een unie dezelfde geheugenlocatie in beslag neemt als van alle andere. Laten we eens een voorbeeld bekijken. De volgende definitie creëert een unie genoemd als *Locatie* en een locatievariabele genoemd als *SchermLoc*:

```
TYPE HoLa
     Ho AS BYTE
     La AS BYTE
END TYPE

UNION Locatie
     WordVeld AS WORD
     ByteVeld AS HoLa
END UNION

DIM SchermLoc AS Locatie

INPUT "Voer een waarde in voor SchermLoc: ", SchermLoc.WordVeld
PRINT "De waarde van WordVeld is:"; SchermLoc.WordVeld
PRINT "De waarde van de hoge byte is:"; SchermLoc.ByteVeld.Ho
PRINT "en de waarde van de lage byte is:"; SchermLoc.ByteVeld.La
```

Wanneer u het veld *SchermLoc.WordVeld* gebruikt, leest u de volledige inhoud van de unie als een geheel getal. Aan de andere kant: wanneer u verwijst naar *SchermLoc.ByteVeld.Ho*, verwijst u naar de hoge byte van *SchermLoc*.

Dynamische structuren creëren met flex strings

Gebruiker gedefinieerde types en unies (TYPE en UNION) zijn zeer geschikt voor de meeste gegevensstructuren die u wilt creëren. Omdat ze gedefinieerd zijn tijdens de compileertijd, zijn ze heel snel voor gebruik en u kunt ze gemakkelijk gebruiken om random-access of binaire bestanden in te kunnen lezen en weg te kunnen schrijven.

Soms wilt u echter meer flexibiliteit dan de types en de unies u geven. Omdat ze gedefinieerd zijn tijdens de compileertijd, moet u de grootte specificeren als een constante (meestal een letterlijke). Bijvoorbeeld, hier is het gebruikte type in de vorige sectie in gebruiker gedefinieerde types:

```
TYPE StudentRecord
     Achternaam AS STRING * 20     ' Een karakterstring van 20 posities
     Voornaam AS STRING * 15       ' Een karakterstring van 15 posities
     IDnum AS LONG                 ' Student ID, een lange integer
     Contact AS STRING * 30        ' Persoon voor contact in noodgevallen
     ContactFoon AS STRING * 14    ' Hun telefoonnummer
     ContactRel AS STRING * 8      ' Relatie van het contact naar de student
     GemiddeldResultaat AS SINGLE  ' Single-precisie percentage
END TYPE

DIM Student AS StudentRecord
```

Dit type moet volstaan voor de meeste doeleinden, maar wat als u het nodig hebt om het aan te passen? U kunt niet een gebruiker gedefinieerd type tijdens de uitvoertijd aanpassen; zodra u het compileert is het ingesteld. Het is wel mogelijk om flex strings aan te passen.

Flex strings aanpassen

Stel dat u de gebruiker zelf de lengte van elk van de leden in de gegevensstructuur wilt laten definiëren. U hebt de code zoals dit:

```
configbestand = freefile

open "CONFIG.DAT" for input as #configbestand
     input #configbestand,    lenStudentAchternaam
     input #configbestand,    lenStudentVoornaam
     input #configbestand,    lenStudentContact
     input #configbestand,    lenStudentContactFoon
     input #configbestand,    lenStudentContactRel
close #configbestand
```

Dan gebruikt u die waarden om een flex string structuur te con-structureren:

```
map StudentFlex$$ * lenStudentAchternaam + _
                    lenStudentVoornaam + _
                    4 + _
                    lenStudentContact + _
                    lenStudentContactFoon + _
                    lenStudentContactRel + _
                    4, _
     lenStudentAchternaam as StudentAchternaam$$, _
     lenStudentVoornaam as StudentVoornaam$$, _
     4 as StudentIDnum$$, _
     lenStudentContact as StudentContact$$, _
     lenStudentContactFoon as StudentContactFoon$$, _
     lenStudentContactRel as StudentContactRel$$, _
     4 as StudentGemiddeldResultaat$$

studentBestand = freefile
open "STUDENTS.DAT" for input as #studentBestand

do until eof(studentBestand)
     input #studentBestand, StudentFlex$$
     print rtrim$(StudentVoornaam$$); " ";
     print rtrim$(StudentAchternaam$$); " ";
     print "(ID #"; cvl(StudentIDnum$$);
     print ") had een resultaatgemiddelde van";
     print cvs(StudentGemiddeldResultaat$$);
loop
```

Nadelen aan flex strings

Zoals altijd, met flexibiliteit komt compromis: er zijn twee belangrijke nadelen aan het gebruik van flex strings met vergelijking tot gebruiker gedefinieerde typen:

- *Snelheid.*
  Omdat flex strings gedefinieerd zijn in uitvoertijd, is er een kleine hoeveelheid verwerkingstijd bij betrokken. Gebruiker gedefinieerde types zijn statisch - gedefinieerd in compileertijd - waardoor er geen overhead is.
- *Gebruikersgemak.*
  Flex strings moeten gedefinieerd worden met gebruik van het MAP statement. Het definiëren van de flex string structuur elementen is veel meer gecompliceerder dan het definiëren van een gebruiker gedefinieerd type. Ook in een flex string structuur bestaan alleen flex strings, dus het invoegen van numerieke waarden betekent dat u string-naar-numeriek en numeriek-naar-string conversiefuncties moet gebruiken (*CVx* en *MKx$* respectievelijk). U kunt direct numerieke waarden aan numerieke members van een gebruiker gedefinieerd type toekennen.

In de volgende nieuwsbrief laat ik u zien hoe we besturingsstructuren in PowerBASIC kunnen gebruiken.

Marco Kurvers

Besturingstechniek in Excel 2010

In dit deel over de besturingstechniek in Excel 2010 ga ik het hebben over:

- Verwijzingen;
- Werken met de Sheet en Range objecten.

Verwijzingen

De vorige keer heb ik het ook over verwijzingen gehad. Het is bijzonder handig om een waarde uit de ene cel te laten verwijzen naar een andere cel, zonder gebruik te hoeven maken van kopiëren en plakken.

Kijk, dat is handig, dan laten we ze toch in vervolg naar elkaar verwijzen!
Maar als u teveel gebruik maakt van deze mogelijkheid, kunt u vreemde verrassingen tegenkomen:

- kringverwijzingen, een probleem dat vaak voorkomt;
- afhankelijkheid, de cellen gaan zich met elkaar 'bemoeien'.

Het uitroepteken '!'

We kunnen alleen verwijzen naar cellen door gebruik te maken van het uitroepteken. Blijven we in het zelfde werkblad dan is het uitroepteken niet nodig. Als we in cel A1 de waarde 10 intoetsen en we geven in A2 op:

```
=Blad1!A1
```

dan zal in cel A2 de waarde van cel A1 komen. Het lijkt er dan op alsof we cel A1 gekopieerd hebben en in cel A2 hebben geplakt. Maar dat is dus niet het geval. Toets nu de waarde 10 in cel B1, maar kopieer nu cel B1 en plak deze in cel B2. Nu staat daar ook de waarde 10.

Verander in cel A1 de waarde 10 in een andere waarde. U ziet dat in cel A2 direct de waarde wordt veranderd, zonder dat de verwijzing wordt veranderd. In principe gebeurt er in cel A2 helemaal niets.

Verander in cel B1 ook de waarde. In cel B2 wordt de waarde niet mee veranderd, en dat komt omdat er geen verwijzing is gemaakt naar cel B1.

Ook al kan een verwijzing maken nuttig zijn, toch zit er niets anders op dan te moeten kopiëren. Zoals onderstaande verwijzing juist is, werkt deze toch niet:

```
=Blad1!A1:C1
```

In de cel, waar u de verwijzing hebt getypt, zal alleen de waarde verschijnen van dezelfde cel in het opgegeven bereik, dat kan bijvoorbeeld cel A1 zijn. U hoeft ook niet te proberen eerst een selectie te maken en dan met de muis de verwijzing in de actieve cel te maken, want dan zal ook alleen de waarde in de cel, waar u de verwijzing hebt gemaakt, weer dezelfde cel uit het opgegeven bereik worden.

Enkele voorbeelden:

Selectie in Blad1	`A1 = 10`	`B1 = 32`
Verwijzing in Blad2:	`A1 = 10`	

Selectie maken in Blad2: we selecteren van A2 naar B2.
Verwijzing in Blad2: `A2 = 10` `B2 =` *geen waarde*

We selecteren nog een keer de cellen A2 en B2, maar nu andersom: van B2 naar A2.
Verwijzing in Blad2: `A2 =` *geen waarde* `B2 = 32`

Het lijkt erop alsof Excel het eindbereik van de selectie negeert. Als het dan toch niet werkt, waarom laat Excel het dan wel toe? Dat heeft te maken met de actieve cel die in de selectie wit wordt aangegeven. De actieve cel zal ook het belangrijkste zijn, want van daaruit zal de waarde altijd worden gehaald.

Kringverwijzingen
Verwijzingen kunnen inderdaad nut hebben, maar in grote werkbladen en door teveel gebruik van verwijzingen, kunnen er complicaties ontstaan. Eén daarvan is een verwijzing die ervoor zorgt dat cellen elkaar kruisen. De cellen ontmoeten elkaar tijdens het verwijzen en dat mag niet.

Als we weer de drie cellen in Blad1 nemen: A1, B1 en C1 en we geven in Blad2 cel A1 op:
`=Blad1!A1:C1`
is er niks aan de hand.

Geef in blad1 in cel C1 op: `=Blad2!A1`
U zou misschien denken dat dit niet kan. We hebben al in Blad2 cel A1 verwezen naar het bereik A1:C1 in blad1 en dan komt er een verwijzing in cel C1 van Blad1 naar Blad2!A1. Dit lijkt op een kringverwijzing, maar net als eerder gezegd zal cel C1 in het bereik worden genegeerd zodat beide cellen elkaar niet ontmoeten.

Nu geven we in Blad1 in cel D1 op: `=C1+Blad2!A1`. Onderstaand tabel geeft de waardenrij aan met de formulerij:

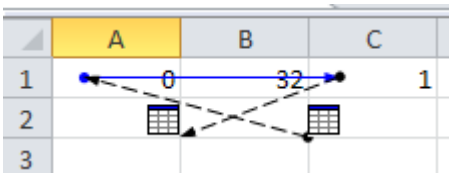
Blad 1

1	32	1	2
=C1		=Blad2!A1	=C1+Blad2!A1

Het enige wat we in Blad 2 hebben staan is de formule: `=Blad1!A1:C1` en zoals net uitgelegd ontstaan er geen problemen.

Veranderen we echter cel A1 door een verwijzing te maken naar cel C1, zie in het rood gegeven, dan komt Excel plotseling met een foutmelding en met een uitleg waarom dit niet kan.

Expres nam ik cel C1 omdat C1 genegeerd zou worden in het bereik. Dat klopt, we mogen cel C1 best gebruiken zoals we in het tabel zien in cel D1, maar gaan we echter naar C1 verwijzen in een cel waar nog eens verwezen wordt naar een bereik waar dezelfde cel C1 in het bereik staat, dan ontstaat er wel een kringverwijzing. Excel geeft dan ook een tekening om te laten zien wat er precies gebeurd is.



De blauwe lijn is de kring, de verwijzing die niet geldig is. De gestippelde lijnen zijn hulplijnen. Excel wil hiermee aangeven waar de cellen naartoe verwijzen.

De cellen, waar de pijlen eindigen, zijn niet de cellen waarmee aangegeven wordt dat de cellen daarheen verplaatst moeten worden om de fout te herstellen. De pijl in cel C1 verteld dat in de formule vanuit Blad 2 terug verwezen wordt naar Blad 1 in cel A1, maar de pijl in A1 verteld weer dat er een

verwijzing staat naar C1 die weer de verwijzing naar Blad1 in Blad2 aanroept, en zo ontstaat de kringverwijzing.

Als we op een pijl dubbelklikken, kunnen we een nieuwe verwijzing kiezen door die in te geven of, als ze er staan, er een te kiezen. Als er niets gebeurt, kan het hele probleem niet eens opgelost worden.

Dit probleem was natuurlijk alleen maar een voorbeeld en we weten best dat de kringverwijzing opgelost wordt als we in cel A1 gewoon weer een getal invoeren, of een verwijzing maken naar een cel die wel geldig is.

Meer informatie over kringverwijzingen: zie de help van Excel of via Google.

Werken met de Sheet en Range objecten

We weten hoe we kunnen werken met de werkbladen en hoe we met meerdere cellen kunnen werken. In VBA gebruiken we een Range object die een parameter bereik verwacht. Dat mag een enkele cel zijn.

In plaats van een Range te gebruiken voor enkele cellen is het niet verkeerd om ook gebruik te maken van de eigenschap Cells.
Het voordeel is dat we direct de rijen en kolommen op kunnen geven en het daardoor ook mogelijk is de eigenschap in lussen te gebruiken.
Het nadeel van Cells is dat de parameters andersom werken. In een Range moeten we kolom en rij opgeven en niet rij en kolom. Cells is een eigenschap en, anders dan bij Range, kunnen we niets opgeven met een punt erachter.

Onderstaande regels werken op dezelfde manier:

```
Blad1.Range("A2").Value = 15
Blad1.Cells(2, 1) = 15
```

Buiten de werkbladen

Na we een blad object hebben ingetoetst met een punt, verschijnt er een lijst met objecten en eigenschappen. Het kan gebeuren dat na het opgegeven blad geen lijst verschijnt. Het probleem is dat dan het blad, of een instantie naar een blad, niet bestaat. Een voorbeeld kan Blad4 zijn. Als die niet (meer) bestaat, zal de lijst ook niet verschijnen. Evenmin is het mogelijk een nieuw blad aan te maken genaamd ‘Blad4’. We zien ook in een Dim … As regel dat in die lijst alleen de bladen in staan die aangemaakt zijn door Excel of door uzelf op de voorgrond. We mogen instanties maken naar die bladen, maar zelf een blad maken kan helaas niet.

Maar het kan nog erger worden. Stel dat u in VBA gebruik maakt van Blad3 en het werkt perfect met verwijzingen vanuit andere bladen. Plotseling wordt het derde blad verwijderd. Het Blad3 object is dan niet meer geldig. Mocht er nog een vierde blad zijn dan zal deze automatisch naar het derde blad worden verplaatst. Helaas zal de naam van het object niet veranderen, en ook al zou u de naam van Blad4 veranderen in Blad3, het object zal Blad4 blijven.

De conclusie is: als we een blad kwijt zijn en we maken een nieuwe aan met de oude naam zullen de verwijzingen naar het blad niet meer werken en dat komt omdat altijd een nieuw blad zelf een unieke objectnaam heeft.

De oplossing is door helemaal geen gebruik meer te maken van de bladobjecten. In VBA is er een collectie Sheets die als een werkblad array werkt. U kunt Sheets direct gebruiken het bevat dezelfde objecten en eigenschappen als de gewone bladobjecten. Het voordeel is dat u kunt achterhalen hoeveel werkbladen er aanwezig zijn (eigenschap Count) en het is zelfs mogelijk nieuwe werkbladen toe te voegen (methode Add).

Let op! Pas op met het gebruik van deze makkelijke mogelijkheid. Als u geen groot project maakt en niet veel bladen gebruikt en u zeker weet dat er tijdens de uitvoer geen bladen toegevoegd hoeven te worden, dan is het niet verkeerd om gewoon gebruik te maken van de bladobjecten. De lijst van de collectie Sheets is groter en als u niet veel nodig hebt kan het voor uw programmaatje veel tijd en geheugen kosten. De collectie werkt helemaal dynamisch en maakt gebruik van het heap geheugen. Excel kan zelf niet weten hoeveel werkbladen u in de collectie maakt omdat Excel alleen de voorgrond is; u bent de achtergrond!

Dit betekent niet dat het een ramp zou betekenen voor het programma. Het is nog altijd VBA en niet Visual Basic 6 of een van de .NET versies. Excel is nog altijd heer en meester en VBA is alleen maar de besturingscode voor de werkbladen.

Dat zien we ook als we in een VBA module twee werkbladen zouden aanmaken:

```
Sheets.Add
Sheets.Add
```

Zouden drie bladobjecten al bestaan, dan zouden we na het uitvoeren van deze twee regels de bladen Blad1, Blad2, Blad3, Blad4 en Blad5 in de projectlijst zien staan.

Onderstaande regels maken beide gebruik van het eerste werkblad:

```
Blad1.Range("A5").Value = 100
Sheets(1).Range("A6").Value = 100
```

In blad 1 staat dus in cel A5 en in cel A6 het getal 100.

Als we nu Blad1 verwijderen, zouden we denken dat beide regels niet meer werken. Dat is niet het geval. Alleen de regel die Blad1 gebruikt zal ongeldig zijn. De tweede regel zal wel correct werken, ook al wordt het eerste item gebruikt. Het eerste item betekent dus niet dat het te maken had met Blad1. Het item zal altijd het eerste werkblad zijn, ook al is het bladobject veranderd.

Werken met ranges voor gevorderden
Met een range kunnen we meer. We hebben een Range object waarmee we een bereik op kunnen geven, maar het bereik van een Range is zelf ook weer een Range waarmee we nog een keer een bereik kunnen opgeven, en die heeft ook weer een Range, en zo kunnen we wel doorgaan.

Wat voor nut heeft dat?

Omdat elk Range object lokaal zijn eigen bereik heeft maar ook zijn eigen subrange object heeft, kunnen we vaststellen dat we hiermee ongelooflijk veel uit kunnen halen wat op de voorgrond onmogelijk is.

Stel dat we in Blad1 de waarde 15 in het bereik A1:A20 willen plaatsen. We zouden dan onderstaande code kunnen gebruiken:

```
For R = 1 To 20
     Blad1.Cells(R, 1) = 15
Next R
```

Deze regels kunnen we echter wijzigen in één regel met gebruik van een Range object:

```
Blad1.Range("A1:A20").Value = 15
```

Alle rijen van 1 t/m 20 in kolom A zullen de waarde 15 krijgen.
Kijk eens naar onderstaande regel:

```
Blad1.Range("A1:A20").Range("A3:A8").Value = 30   ¹
```

De rijen A3 t/m A8 zullen overschreven worden met de waarde 30. Als u goed met VBA overweg kan, ziet u misschien ook dat net zo goed alleen de subrange voldoende was geweest. De waarden zouden dan alsnog in dezelfde cellen worden geplaatst. Maar schijn bedriegt. Als u onderstaande regel probeert ziet u waarom:

```
Blad1.Range("A3:A20").Range("A3:A8").Value = 80   ²
```

Het is vreemd als u dit ziet, want het lijkt alsof gewoon weer de cellen A3 t/m A8 overschreven worden, maar omdat het een subrange is en het hoofdbereik niet meer begint bij A1 maar bij A3, zullen de cellen worden bepaald zodat de waarden op een heel ander bereik komen dan opgegeven wordt. Hieronder ziet u de punten 1 en 2 die hierboven staan met de uitkomsten.

	1.	**2.**
A1	15	15
A2	15	15
A3	30	30
A4	30	30
A5	30	80
A6	30	80
A7	30	80
A8	30	80
A9	15	80
A10	15	80
A11	15	15
A12	15	15
A13	15	15
A14	15	15
A15	15	15
A16	15	15
…	…	…

Geen opgenomen macro kan tegen deze mogelijkheden op. Met de VBA objecten kunnen we veel functionaliteit gebruiken en tijd besparen, en zouden we meer bereiken uit hetzelfde bereik willen halen dan is er zelfs een mogelijkheid een eigen Range object te declareren.

Het Range type
Hebben we het hoofdbereik vaker nodig dan kunnen we gebruik maken van het With statement, maar VBA laat ook toe het With statement korter te maken met een instantie naar een bereik.

```
With Blad2.Range("A1:F1")
     .Range("B1:F1").Value = 100
     .Cells(1, 1) = 50
End With
```

We kunnen de code ook zo schrijven en eventueel met een With statement:

```
Dim R As Range

Set R = Blad2.Range("A1:F1")
R.Range("B1:F1").Value = 100
R.Cells(1, 1) = 50
Set R = Nothing
```

We zouden een spelletje kunnen spelen met vragen, zoals:

In welke cel krijgen we de waarde 75 met de regel:
`R.Range("D1:F1").Range("C1").Value = 75`?
Antwoord: In cel F1.

Wat gebeurt er als we cel C1 veranderen in K1?
Antwoord: Ondanks cel K1 buiten het bereik ligt geeft VBA geen foutmelding. De waarde 75 komt terecht in cel N1.

Uit de tweede vraag en antwoord blijkt dat we niet per se binnen het bereik hoeven te blijven.

Een target gebruiken
We kunnen dus eigen Range objecten declareren en initialiseren met bereiken uit verschillende werkbladen. Het is ook mogelijk Range bereiken door te geven als target parameters. Een target is een zwevend bereik. Het is geen vast object dat met een Dim statement gedeclareerd is. Het argument mag een constante zijn of een objectvariabele van het type Range. Als het een constante is, moet het een geldig bereik zijn met een eventueel opgegeven sheet-item of bladobject.

Een target hoeft u niet vrij te geven met een Nothing als u denkt hem niet meer te willen gebruiken. Een argumentwaarde die doorgegeven wordt naar een parametervariabele zal na het uitvoeren van de subroutine of functie automatisch worden opgeruimd.

Contact met de buitenwereld
De meeste event subroutines in Excel maken gebruik van een Target parameter, zoals de *Worksheet_SelectionChange()* event en de *Worksheet_Change()* event. U kunt dus VBA code aanroepen zodra er een selectie wordt gewijzigd of van werkblad wordt gewijzigd. Deze events zorgen voor contact met de voorgrond en de achtergrond. Dankzij de events bent u niet meer verplicht telkens een macro te starten. U kunt een *Main* subroutine maken die de rest van het project beheert. Om niet steeds de Main subroutine te hoeven starten, kunt u het ook aanroepen in het *Workbook_Open()* event die u kunt vinden in *ThisWorkbook* in het VBA project.

In de volgende nieuwsbrief gaan we daar eens verder naar kijken.

Marco Kurvers

Grafisch programmeren in GW-BASIC (12).

Hoofdstuk 7: Educatieve toepassingsprogramma's

De voorgaande 35 grafische programma's zijn geen toepassingsprogramma's. Ze waren bedoeld om een overzicht te geven van de mogelijkheden van graphics met hoog oplossend vermogen. In dit hoofdstuk zullen we vijf praktijkgerichte grafische programma's presenteren. Deze programma's zijn:

1. Tekenen van een landkaart
2. Maken van een histogram (een stavengrafiek)
3. Demonstratieprogramma voor de breking van lichtstralen
4. Demonstratieprogramma voor de 'speldenworp' van Buffon
5. Prooi-roofdierpopulaties

1. Tekenen van een landkaart

Educatieve programma's over topografie gebruiken dikwijls landkaarten van een land of van een werelddeel die door de computer getekend worden. We zullen laten zien hoe de computer zo'n landkaart kan tekenen. We gaan de landkaart (althans de grensomtrek) van Zwitserland tekenen. Waarom Zwitserland? Wel, in de eerste plaats omdat de auteur dit gekozen heeft en in de tweede plaats omdat Zwitserland geen eilanden heeft. Het tekenen van de contouren van Nederland gaat in principe net zo, alleen kosten al die eilanden een hoop extra werk, vandaar!

Hoe tekenen we nu de omtrek van een bepaald land? Heel eenvoudig! We pakken gewoon een atlas, een stukje overtrekpapier, een potlood, een liniaal en millimeterpapier. We kiezen in de atlas een land of werelddeel uit en trekken het op overtrekpapier over. Daarna leggen we het overgetrokken plaatje op millimeterpapier, waar we van te voren een coördinatenstelsel op getekend hebben (een x- en een y-as). We bepalen nu van een (groot) aantal punten op de omtrek van het land (of werelddeel) de coördinaten (x,y) en schrijven deze op. Als we ons op het beeldscherm ook een coördinatenstelsel voorstellen en als we hierin de punten, waarvan we de coördinaten in een programma opnemen, met elkaar laten verbinden, dan ontstaat een 'landkaart' op het scherm.

In het onderstaand programma, dat 'de kaart' van Zwitserland tekent, is een 'echte' kaart gebruikt met een schaal van 1:2.000.000. Om een enigszins natuurgetrouwe weergave te verkrijgen zijn de coördinaten van 90 grenspunten berekend. De coördinaten van deze punten, die opgegeven zijn in millimeters ten opzichte van de oorsprong van het gekozen coördinatenstelsel, zijn in DATA regels opgenomen. Als het programma het coördinatenpaar (0,0) leest, weet het dat de 'kaart' af is.

De x-coördinaten liggen tussen 6 en 177. Om deze naar het hoge-resolutiebereik 0-320 te transformeren vermenigvuldigen wij zowel de x- als de y-coördinaten met de factor K = 2. Hierdoor blijft nog wat ruimte over om een kader rond de kaart te tekenen. De werking van het programma zal hiermee duidelijk zijn.

We hebben op deze manier ook een kaart van Europa en zelfs een wereldkaart getekend. Het probleem met de eilanden hebben we als volgt opgelost. Als het programma in een DATA regel negatieve x- en y-coördinaten leest, weet het programma dat dit punt niet met het vorige verbonden moet worden en dat dit punt dus het begin is van een apart stukje 'land'.


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

```
Public Class frmProg36

    Private Const K = 2
    Private Const H = 0.5
    Private Const V = 300
    Private Kaart() As Byte = _
        { _
            69, 108, 71, 107, 70, 104, 75, 104, 76, 106, _
            80, 107, 81, 104, 86, 105, 91, 108, 94, 107, _
            101, 106, 100, 108, 105, 108, 106, 110, 101, 110, _
            98, 112, 102, 117, 108, 118, 112, 112, 114, 115, _
            118, 110, 128, 110, 139, 102, 145, 103, 146, 95, _
            142, 86, 144, 78, 154, 77, 154, 72, 163, 67, _
            168, 68, 173, 76, 177, 73, 174, 59, 177, 56, _
            177, 52, 171, 51, 167, 56, 161, 50, 165, 43, _
            166, 34, 162, 34, 157, 42, 143, 36, 139, 48, _
            136, 45, 133, 48, 132, 40, 122, 23, 125, 15, _
            122, 11, 119, 12, 114, 20, 116, 25, 100, 35, _
            102, 45, 94, 42, 88, 35, 90, 28, 79, 15, _
            75, 15, 66, 19, 60, 14, 52, 12, 48, 13, _
            37, 29, 39, 36, 37, 40, 39, 43, 29, 45, _
            16, 38, 18, 33, 13, 29, 6, 28, 6, 32, _
            11, 34, 13, 40, 10, 45, 12, 53, 25, 63, _
            26, 73, 30, 73, 48, 94, 42, 94, 46, 102, _
            54, 102, 53, 99, 61, 98, 63, 102, 69, 108, _
            0, 0 _
        }
    Private Sub frmProg36_Paint(…, ByVal e As ….PaintEventArgs) …
        Dim X2, Y2 As Integer

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

        Dim X1 As Integer = Int(K * X + H)
        Dim Y1 As Integer = Int(V - K * Y + H)

        X = Kaart(2) : Y = Kaart(3)
        Dim Data As Integer = 4
        While X <> 0
            X2 = Int(K * X + H) : Y2 = Int(V - K * Y + H)
            e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
            e.Graphics.DrawLine(Pens.Black, X1, Y1 + 1, X2, Y2 + 1)
            X1 = X2 : Y1 = Y2
            X = Kaart(Data) : Y = Kaart(Data + 1)
            Data += 2
        End While
    End Sub
End Class
```

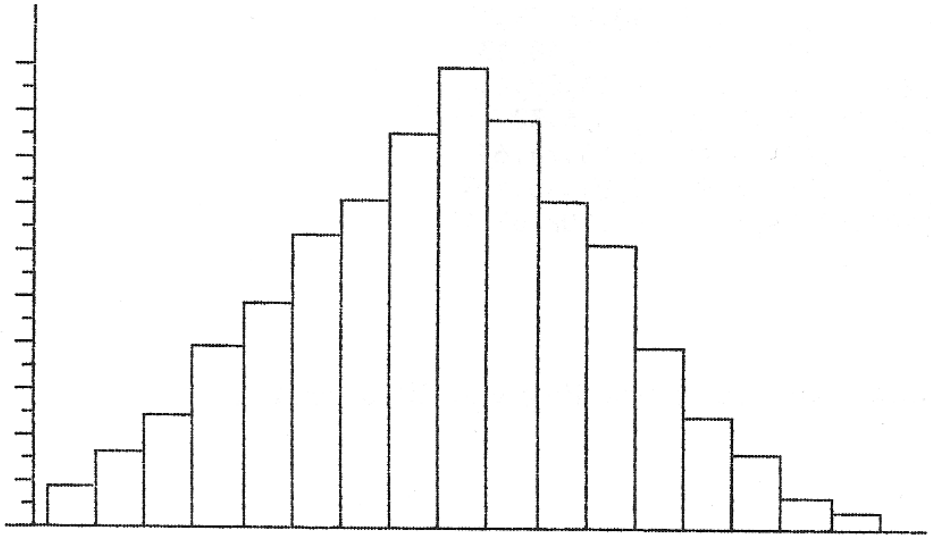
Zie de appendix voor een uitbreiding van dit programma.

2. Maken van een histogram

Vaak willen we een aantal waarnemingen grafisch in een stavendiagram of histogram weergeven. Hiertoe dient het volgende programma. Dit programma tekent een horizontale en een verticale as. De verticale as wordt in stukjes van acht schermpuntjes verdeeld. Dit komt overeen met 5% van de hele verticale as. Elk tweede streepje op deze as geeft de volgende 10% aan en is iets breder getekend. Hiermee kan de lengte van de staven redelijk geschat worden.

Het programma kan maximaal 100 staven verwerken. Uit esthetische overwegingen moet u echter niet meer dan 40 staven invoeren, omdat de staafjes anders meer op lijntjes dan op balkjes gaan lijken.

Het zou mooi zijn als we bij de staven, de assen en de schaalverdeling tekst en getallen zetten, maar dat is een heel werk, dus doen we het nu maar niet. Denkt u erom dat de getallen die u intoetst de lengte van de staven aangeven. Als elke staaf een bepaalde klasse met waarnemingen voorstelt, dan voert u dus steeds het aantal waarnemingen (de frequentie) van een klasse in. Het programma kan uitgebreid worden door er een stuk voor te zetten dat ruwe gegevens inleest, die vervolgens netjes in klassen verdeeld worden. De frequentie van de waarnemingen in de klassen wordt vervolgens door het programma 37 gebruikt om de stavengrafiek te tekenen.



```
100 'programma 37                       HISTOGRAM
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS:KEY OFF
140 PRINT "HISTOGRAM TEKENEN"
150 PRINT "-----------------"
160 PRINT: PRINT
170 INPUT "HOEVEEL GEGEVENS ( < 40 ) "; N
180 DIM A(N) : MX=-1E+10 : PRINT
190 FOR J=1 TO N
200     PRINT "WAARDE"; J; TAB(12);
210     INPUT A(J)
220     IF A(J) > MX THEN MX=A(J)
230 NEXT J
240 CLS
250 '                       HORIZONTALE AS
260 LINE (FNX(4),300)-(FNX(320),300),1
270 '                         VERTICALE AS
280 LINE (FNX(10),0)-(FNX(10),300),1
290 '                      SCHAALVERDELING
```

```
300 FOR J=10 TO 1  STEP -1
310     Y1=300-J*26 : Y2=Y1
320     LINE (FNX(4),Y1) - (FNX(10),Y2),1
330     Y1=Y2+13    : Y2=Y1
340     LINE (FNX(7),Y1) - (FNX(10),Y2),1
350 NEXT J
360 '           STAVEN TEKENEN   B=BREEDTE
370 B=INT(310/N) : H=.5
380 FOR J=1 TO N
390     X1=(J-1)*B+10 : Y1=300
400     X2=X1+B: Y2=INT(300-260*A(J)/MX+H)
410     LINE (FNX(X1),Y1)-(FNX(X2),Y2),1,B
420 NEXT J
430 A$=INKEY$: IF A$="" THEN 430
440 CLS: KEY ON: END
```

Om bovenstaand programma in Visual Basic 2008 uit te kunnen voeren, moeten we de FOR lus (regels 190 – 230) anders aanpakken. We kunnen namelijk niet uit een ingevoerd getal voor de hoeveelheid gegevens het aantal componenten bepalen om de waarden in te kunnen voeren, of we zouden gebruik kunnen maken van een ListView component met daarnaast een invoerbalk en een 'Toevoegen' knop. Het is echter een enorme klus om zoiets eventjes in elkaar te zetten. Als u de tijd ervoor wilt nemen om iets te maken, zodat de rest van de regels van bovenstaand programma uitgevoerd kan worden, voor een prima histogram, plaats dan eerst vijf invoerbalken zodat in ieder geval maximaal vijf staven getekend kan worden. Programma 37 kan dan toch uitgevoerd worden en kunt u altijd later nog het programma uitbreiden met bijvoorbeeld een ListView component.

Voor Visual Basic 6 gebruikers is er ook een mogelijkheid om bovenstaand programma uit te kunnen voeren door gebruik te maken van het InputBox() statement, die door de Visual Basic .NET versies niet meer ondersteund wordt.

Hieronder ziet u de code in Visual Basic 2008. Als u onderstaande code vergelijkt met bovenstaande code dan zult u zien dat er weer wijzigingen zijn aangebracht om een goed histogram te kunnen tekenen. Ik heb gebruik gemaakt van vaste waarden in dezelfde array A().

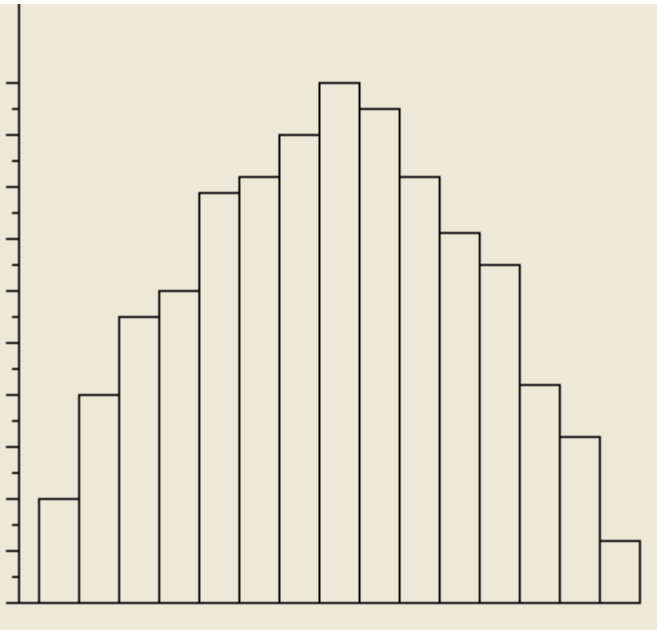
```
Public Class frmProg37

    Private A() As Integer = _
    { _
        20, 40, 55, 60, 79, 82, 90, 100, 95, 82, 71, 65, 42, 32, 12 _
    }
    Private MX As Double = -10000000000.0
    Private Const H = 0.5

    Private Sub Waarde()
        For J As Integer = 0 To A.Length() - 1
            If A(J) > MX Then MX = A(J)
        Next
    End Sub

    Private Sub frmProg37_Paint(…, ByVal e As ….PaintEventArgs) …
        Waarde()
        With e.Graphics
            ' Horizontale as
            .DrawLine(Pens.Black, 4, 300, 320, 300)
            ' Verticale as
```

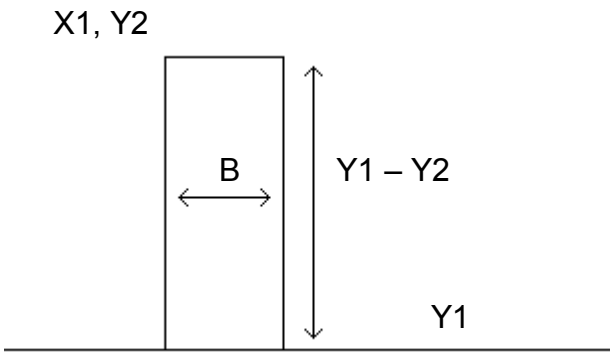
```
            .DrawLine(Pens.Black, 10, 0, 10, 300)
            ' Schaalverdeling
            For J As Integer = 10 To 1 Step -1
                Dim Y1 As Integer = 300 - J * 26, Y2 As Integer = Y1
                .DrawLine(Pens.Black, 4, Y1, 10, Y2)
                Y1 = Y2 + 13 : Y2 = Y1
                .DrawLine(Pens.Black, 7, Y1, 10, Y2)
            Next
            ' Staven tekenen    B=Breedte
            Dim B As Integer = Int(310 / A.Length())
            For J = 0 To A.Length() - 1
                Dim X1 As Integer = J * B + 20
                Dim Y1 As Integer = 300
                'Dim X2 As Integer = X1 + B
                Dim Y2 As Integer = Int(300 - 260 * A(J) / MX + H)
                .DrawRectangle(Pens.Black, X1, Y2, B, Y1 - Y2)
            Next
        End With
    End Sub
End Class
```



Waar de wijziging in de code vooral om gaat is de code die voor de staven zorgt, vanaf de commentaar-regel ‘Staven tekenen’. In de For … Next lus wordt X1 berekend met de lusteller J maal de breedte van de staven plus 20 (marge vanaf de verticale as). Niet met (J – 1), omdat de lusteller al met 0 begint.

Variabele Y1 is gewoon 300 dus geen wijziging, maar zie eens wat ik met de variabele X2 heb gedaan. Deze staat in commentaar, dus uitgeschakeld. U zult zich misschien afvragen waarom, want we moeten toch horizontaal de staven rechtop naast elkaar kunnen tekenen?
Dit heeft te maken op de manier hoe de methode *DrawRectangle()* de staaf tekent; op een hele andere manier dan het LINE commando met de B parameter het toen tekende.

De verklaring is dat de methode geen gebruik maakt van een tweede hoekpunt waar X2 en Y2 voor nodig waren, maar dat alleen de linker bovenhoek, de breedte van de staaf en de hoogte voldoende zijn. Hoe de methode precies de staaf tekent, ziet u hieronder.

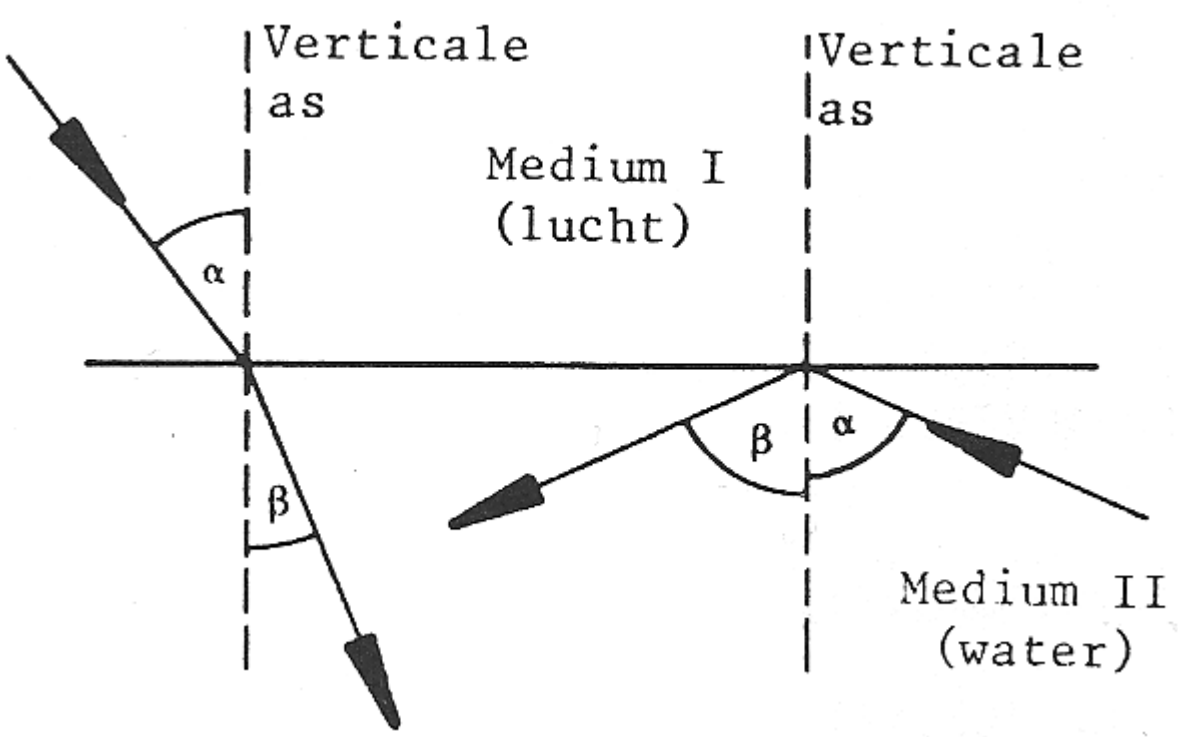


Omdat Y2 de hoogte is van elke staaf, moet die variabele gebruikt worden voor de linker bovenhoek. Normaal gesproken zouden we inderdaad voor de volgende staaf X1 + B moeten toekennen aan X2, maar omdat de derde parameter om de breedte van de staaf vraagt en niet om de X coördinaat, moeten we dus variabele B gebruiken. De vierde parameter vraagt om de hoogte van de staaf. Ik trek echter Y2 af van Y1 om ervoor te zorgen dat alle staven op de horizontale as staan en er niet doorheen getekend worden.

3. Demonstratieprogramma voor de breking van lichtstralen

Met dit programma willen we laten zien hoe we de computer bij natuurkundelessen zouden kunnen gebruiken. Voordat we het programma geven leggen we nog iets uit van de natuurkundige beginselen van de breking van licht.

Als een lichtstraal vanuit de ene stof (bijvoorbeeld lucht) een andere, optisch dichtere, stof (bijvoorbeeld water) binnenkomt, wordt de straal op het scheidingsvlak van beide stoffen naar de verticale as toe gebogen. Zie onderstaande figuur.



Hierbij geldt de brekingswet van Snellius:

$$\frac{\sin\alpha}{\sin\beta} = \frac{c1}{c2} = n$$

$c1$ en $c2$ zijn de lichtsnelheden in respectievelijk de eerste en de tweede stof. De constante n heet de brekingsindex. Bij de overgang van lucht naar water geldt een brekingsindex van 1,33. Voor de overgang van lucht naar glas gelden andere waarden, enzovoorts.

Als het licht vanuit een ‘dichter’ medium overgaat in een ‘dunner’ medium geldt het omgekeerde: de lichtstralen worden nu van de verticale as, die loodrecht op het scheidingsvlak van beide stoffen staat, afgebogen. De brekingshoek α kan nooit groter dan 90^0 zijn.

Voor dit grensgeval ($\alpha = 90^0$) geldt:

$$\frac{\sin 90^0}{\sin\beta^*} = n \Rightarrow \sin\beta^* = \frac{1}{n} \quad \text{(want } \sin 90^0 = 1\text{)}$$

Bij de overgang van lucht naar water geldt voor β^*:

$$\sin\beta^* = \frac{1}{1{,}33} \Rightarrow \sin\beta^* = 0{,}7519 \Rightarrow \beta^* = 48{,}75^0$$

Groter dan $48{,}75^0$ kan β dus niet worden. Dit betekent dat als het licht van water overgaat in lucht met een invalshoek α die groter is dan β^* (zie rechter figuur), het licht niet meer het water 'uitkomt'.
Op het scheidingsvlak van water en lucht wordt het licht geheel teruggekaatst. Op dit principe berusten de moderne glasvezelkabels, waarin informatie in de vorm van licht wordt getransporteerd.

In het onderstaand demonstratieprogramma kan de brekingsindex n ingetoetst worden. De lichtbron bevindt zich in het punt met schermcoördinaten (0, 300). Het scheidingsvlak ligt horizontaal en is de lijn V = 160. De invalshoek van een lichtstraal die van medium 2 (de dichtere stof, onderste helft) in medium 1 (de lichtere stof, bovenste helft) overgaat wordt van 0^0 steeds met stapjes van 3^0 opgehoogd. Op het moment dat deze invalshoek groter wordt dan β^* (deze is afhankelijk van de ingetoetste brekingsindex) treedt totale reflectie (terugkaatsing) op. Rond de figuur wordt een kader getekend. De eindpunten van de diverse lichtstralen worden met trigonometrische functies berekend. De commentaaropdrachten in het programma leggen nog eens uit 'wat waar' gebeurt.

De inverse functie van de sinusfunctie (de boogsinus of arcsinus) bestaat niet in Microsoft BASIC. We lossen dit op door de inverse tangensfunctie (ATN in BASIC) te gebruiken.

Tip! In de Visual Basic .NET versies bestaat de arcsinus functie wel. Deze is te vinden in het Math object. Hoewel in Visual Basic 2008 ook met die functie gewerkt kan worden, heb ik ervoor gekozen om toch de oude manier te gebruiken.
Kunt u wel de arcsinus functie laten werken zoals de ATN functie in het programma werkt?

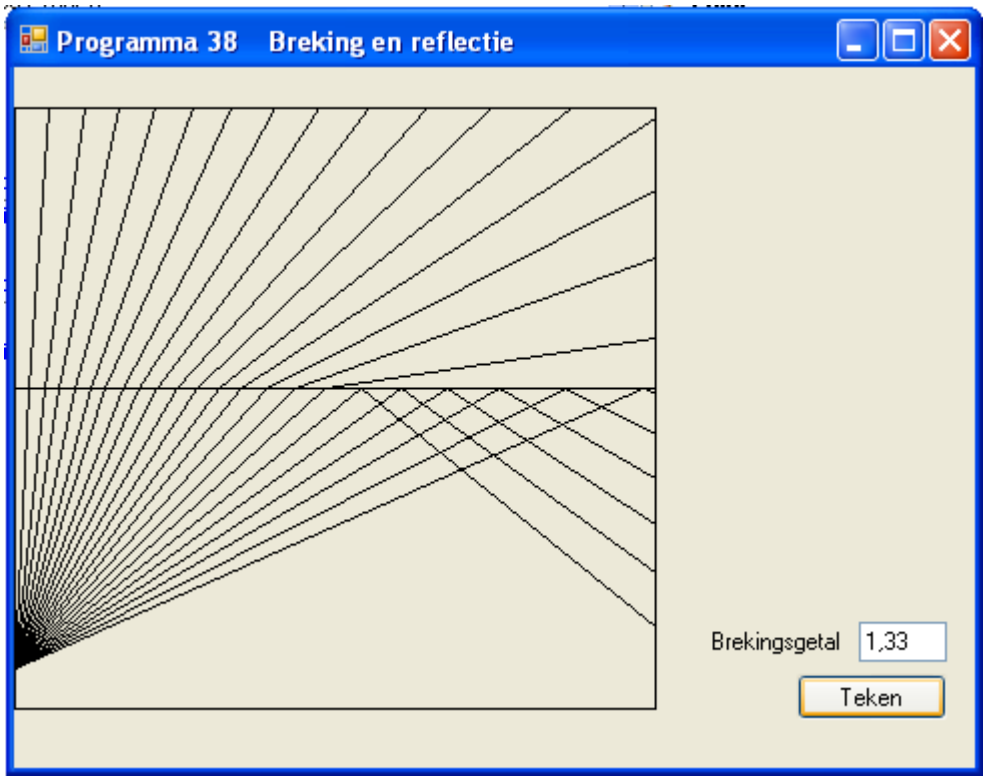
Het voordeel van een dergelijke computersimulatie, boven het uitvoeren van het natuurkundige experiment, is dat we heel gemakkelijk verschillende brekingsindexen kunnen invoeren zonder steeds andere apparatuur op te hoeven stellen en dat de lichtstralen als dunne lijnen zichtbaar gemaakt kunnen worden.

```
100 'programma 38      BREKING EN REFLECTIE
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "BREKINGSGETAL "; N
150 V=160 : H=.5 : RD=4*ATN(1)/180
160 CLS
170 '               KADER EN SCHEIDINGSLIJN
180 LINE (FNX(0),20) -(FNX(320),300),1,B
190 LINE (FNX(0),160)-(FNX(320),160),1
200 '
210 B=0
220 B=B+3: B1=B*RD : X1=0 : Y1=300
230 X2=INT(140*TAN(B1)+H) : Y2=V
240 IF X2 > 320 THEN GOTO 410
250 LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
```

```
260 X1=X2: Y1=Y2: S=N*SIN(B1)
270 IF S > 1 THEN GOTO 340
280 '                                BREKING
290 A1=ATN(S/SQR(1-S*S))
300 X2=INT(X1+140*TAN(A1)+H) : Y2=20
310 IF X2 < 320 THEN GOTO 380
320 X2=320 : Y2=INT(V-(320-X1)/TAN(A1)+H)
330 GOTO 380
340 '                             REFLECTIE
350 X2=X1+X1 : Y2=300
360 IF X2 < 320 THEN GOTO 380
370 X2=320 : Y2=INT(V+(320-X1)/TAN(B1)+H)
380 '
390 LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
400 GOTO 220
410 '
420 A$=INKEY$: IF A$="" THEN 420
430 CLS: KEY ON: END
```

Bovenstaand programma voldoet aan de normen die nodig zijn om Programma 38 uit te kunnen voeren. Helaas ziet de structuur er niet netjes uit. Er worden veel sprongen gemaakt met GOTO en zelfs in die BASIC tijd kon dat al vermeden worden met WHILE en ELSE.

Daardoor zal het programma in Visual Basic 2008 er heel anders uitzien, en toch dezelfde tekening (zie rechts) zal weergeven als in GW-BASIC getekend wordt.




```
Public Class frmProg38
    Private Const V = 160
    Private Const H = 0.5

    Private Sub Breking(ByVal S As Double, ByVal X1 As Integer, _
                        ByRef X2 As Integer, ByRef Y2 As Integer)
        Dim A1 As Double = Math.Atan(S / Math.Sqrt(1 - S * S))
        X2 = Int(X1 + 140 * Math.Tan(A1) + H) : Y2 = 20
        If X2 >= 320 Then
            X2 = 320
            Y2 = Int(V - (320 - X1) / Math.Tan(A1) + H)
        End If
    End Sub
```

```
    Private Sub Reflectie(ByVal B1 As Double, ByVal X1 As Integer, _
                          ByRef X2 As Integer, ByRef Y2 As Integer)
        X2 = X1 + X1 : Y2 = 300
        If X2 >= 320 Then
            X2 = 320
            Y2 = Int(V + (320 - X1) / Math.Tan(B1) + H)
        End If
    End Sub

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

    Private Sub frmProg38_Paint(…, ByVal e As ….PaintEventArgs) …
        If txtN.Text() <> "" Then
            Dim N As Double = CDbl(txtN.Text())
            Dim RD As Double = Math.PI / 180
            With e.Graphics
                ' Kader en scheidingslijn
                .DrawRectangle(Pens.Black, 0, 20, 320, 300)
                .DrawLine(Pens.Black, 0, 160, 320, 160)
                '
                Dim X2 As Integer = 0, B As Integer = 0
                While X2 <= 320
                    B += 3
                    Dim B1 As Double = B * RD
                    Dim X1 As Integer = 0, Y1 As Integer = 300
                    X2 = Int(140 * Math.Tan(B1) + H)
                    Dim Y2 As Integer = V
                    If X2 <= 320 Then
                        .DrawLine(Pens.Black, X1, Y1, X2, Y2)
                        X1 = X2 : Y1 = Y2
                        Dim S As Double = N * Math.Sin(B1)
                        If S > 1 Then
                            Reflectie(B1, X1, X2, Y2)
                        Else
                            Breking(S, X1, X2, Y2)
                        End If
                        .DrawLine(Pens.Black, X1, Y1, X2, Y2)
                    End If
                End While
            End With
        End If
    End Sub
End Class
```

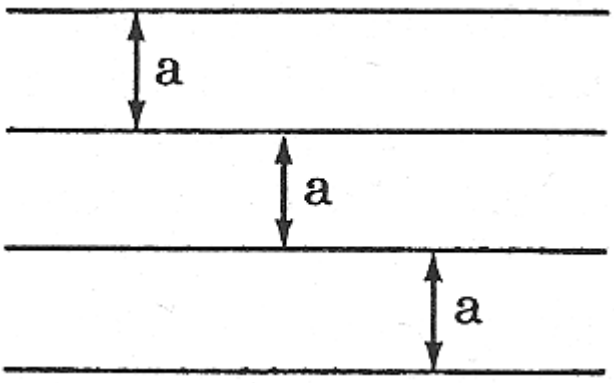
Waarom een While statement die variabele X2 <= 320 moet controleren? Nou, kijk eens naar regel 240. U ziet dat als X2 > 320 er naar regel 410 gesprongen wordt. Dat betekent: zolang X2 <= 320 gaat het programma gewoon door. Dat zien we ook in regel 400 met een GOTO 220. Vandaar dat er een While … End While moest komen en geen If … Then en een GoTo. Bovendien bestaat het GoTo statement niet meer in de VB .NET versies.

De codeblokken die na regel 270 worden uitgevoerd heb ik in subroutines gezet, Reflectie en Breking. Wel moesten de variabelen doorgegeven worden in de parameters. Ik had de codeblokken net zo goed na Then en Else kunnen plaatsen en waren de subroutines niet nodig geweest. Als u het pro-

gramma van versie 2008 wilt overnemen, kunt u zelf bepalen of u de subroutines gaat gebruiken of niet.

4. De speldenworp van Buffon (1773)

Stelt u zich voor dat we in een plat vlak een aantal evenwijdige lijnen met een onderlinge afstand a tekenen.



We werpen nu, zonder echt te 'mikken', een speld met een lengte c die kleiner dan of gelijk aan a is ($c \leq a$) op het vlak met de evenwijdige lijnen. Hoe groot is nu de kans dat een speld een van de lijnen treft, dat wil zeggen snijdt of raakt?

Dit probleem kwam in 1773 op bij de Franse wiskundige Buffon na het zien van de Amerikaanse vlag met de 'stars en stripes'. Buffon heeft berekend dat deze kans gelijk is aan

$$\frac{2c}{a\pi}$$

Omdat deze formule de constante π bevat, heeft men deze 'speldenworp' vaak gebruikt om de waarde van π door simulatie te bepalen. We werpen hiertoe vaak, bijvoorbeeld een lucifer, op een papier waarop een aantal evenwijdige lijnen (met een onderlinge afstand die groter dan of gelijk aan de lengte van de lucifer is). Als de lucifer in n worpen k keer een lijn treft, dan geldt dat

$$\frac{2c}{a\pi} \text{ ongeveer gelijk is aan } \frac{k}{n}.$$

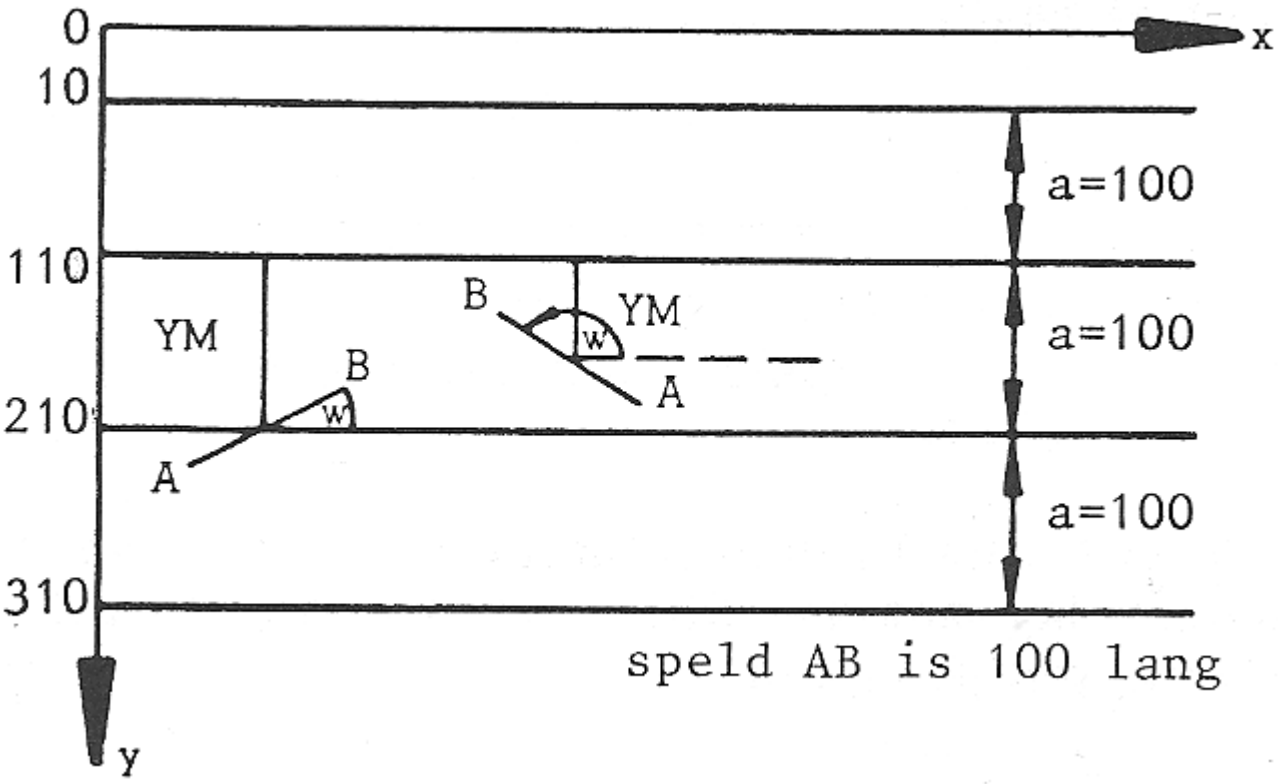
Beide uitdrukkingen geven namelijk een indruk van de kans dat de lucifer een lijn treft.
We kunnen als volgt de waarde van π benaderen:

$$\frac{2c}{a\pi} = \frac{k}{n} \quad \Rightarrow \quad \pi \simeq \frac{2cn}{ak}.$$

Op deze manier heeft de astronoom Rudolf Wolf in 1850 met 5000 luciferworpen voor π de waarde 3,1596 gevonden (π = 3,141592654….).

Met een computer kunnen we gemakkelijk duizenden worpen simuleren. Deze programma's vinden we in bijna elk informaticaleerboek. Dit zijn echter programma's die vrij lang draaien, maar waaraan niets te 'zien' is. Als we bijvoorbeeld in zo'n programma voor n de waarde 10.000 zouden intikken, zien we eerst een tijd niets en vervolgens verschijnt de mededeling dat van de 10.000 keer 6.349 keer een lijn is geraakt, hetgeen voor π de waarde 3,1501024 oplevert.

We gaan een programma maken dat ook dergelijke berekeningen uitvoert, maar dat bovendien elke speld die geworpen wordt laat zien. We zien het experiment als een film aan ons voorbijgaan. Bekijk hiertoe de onderstaande tekening.



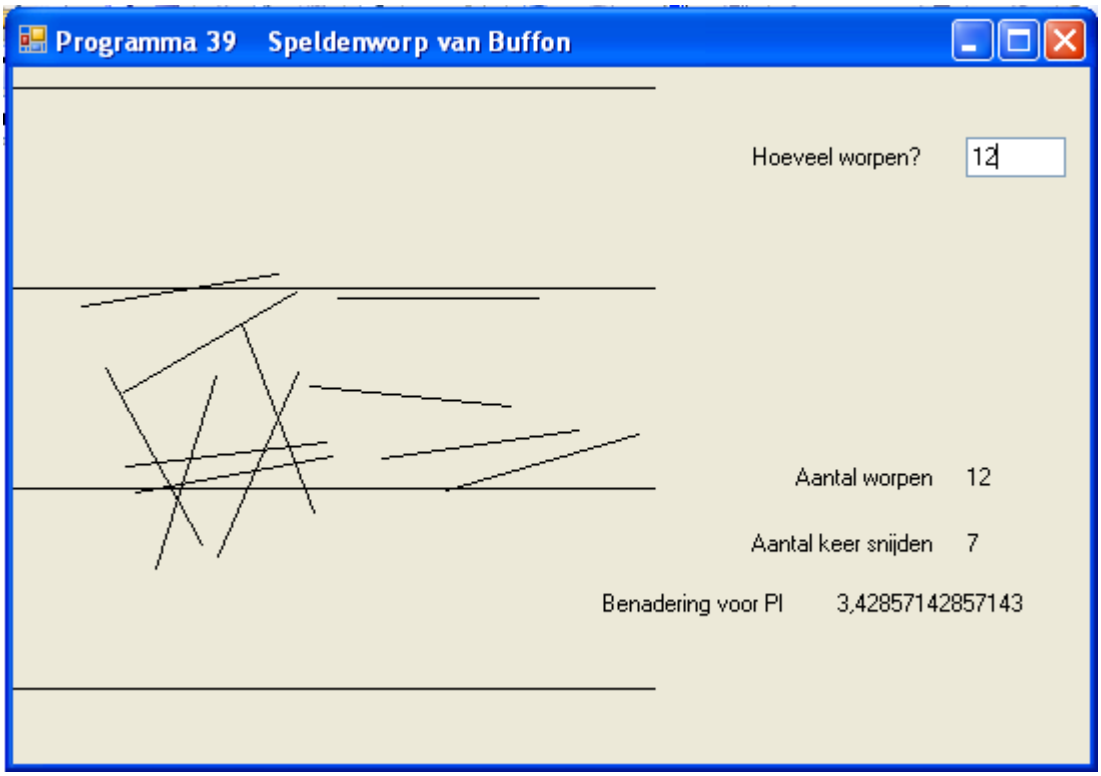
De manier waarop een speld valt kan met twee toevalsgetallen bepaald worden. De twee toevalsgetallen bepalen respectievelijk de afstand YM van de speld tot de daarboven liggende lijn en de hoek W die de speld met de lijnen maakt:

$0 \leq YM \leq a$ YM is de afstand van het midden van de speld tot de daarboven liggende horizontale lijn

$0 \leq W \leq 180^0$ W is de hoek (in positieve zin, dat wil zeggen bij draaiing tegen de klok in) die de speld met de positieve x-richting maakt

Hierna volgt een illustratie van hoe het eruit kan zien en het programma.

```
100 'programma 39     SPELDENWORP VAN BUFFON
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 PRINT "SPELDENWORP VAN BUFFON"
150 PRINT "----------------------"
160 RANDOMIZE
170 INPUT "HOEVEEL WORPEN "; N
180 M=0: H=.5: PI=4*ATN(1)
190 CLS
200 FOR Y1=10 TO 310 STEP 100
210    LINE (FNX(0),Y1)-(FNX(320),Y1),1
220 NEXT Y1
230 '            N MAAL GOOIEN EN TEKENEN
240 FOR J=1 TO N
250    XM=INT(220*RND+ 50+H)
260    YM=INT(100*RND+110+H)
270    W=PI*RND:DX=50*COS(W):DY=50*SIN(W)
280    X1=INT(XM-DX+H) : Y1=INT(YM+DY+H)
290    X2=INT(XM+DX+H) : Y2=INT(YM-DY+H)
300    LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
310    IF Y1>=210 OR Y2<=110 THEN M=M+1
320 NEXT J
330 LOCATE 24,1
340 PRINT "AANTAL WORPEN       "; N
350 PRINT "AANTAL KEER SNIJDEN "; M
360 PRINT "BENADERING VOOR PI  "; 2*N/M
370 A$=INKEY$: IF A$="" THEN 370
380 CLS: KEY ON: END
```



In de regels 330-360 ziet u hoe u in SCREEN 105 gewoon tekst op het grafische scherm kunt afdrukken. Dit kan alleen bij SCREEN 105!
Het ‘oneerlijke’ van dit programma is, dat het een benadering van π berekent, maar zelf (regel 180) de ‘juiste’ waarde van π hierbij gebruikt!

```
Public Class frmProg39
    Private Const H = 0.5

    Private Sub frmProg39_Paint(…, ByVal e As ….Forms.PaintEventArgs) …
        If txtN.Text() <> "" Then
            Dim N As Integer = CInt(txtN.Text())
            Dim M As Integer = 0
            Randomize()
            For Y1 As Integer = 10 To 310 Step 100
                e.Graphics.DrawLine(Pens.Black, 0, Y1, 320, Y1)
            Next
            '           N maal gooien en tekenen
            For J As Integer = 1 To N
                Dim XM As Integer = Int(220 * Rnd() + 50 + H)
                Dim YM As Integer = Int(100 * Rnd() + 110 + H)
                Dim W As Double = Math.PI * Rnd()
                Dim DX As Double = 50 * Math.Cos(W)
                Dim DY As Double = 50 * Math.Sin(W)
                Dim X1 As Integer = Int(XM - DX + H)
                Dim Y1 As Integer = Int(YM + DY + H)
                Dim X2 As Integer = Int(XM + DX + H)
                Dim Y2 As Integer = Int(YM - DY + H)
                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                If Y1 >= 210 Or Y2 <= 110 Then M += 1
            Next
            lblWN.Text() = N.ToString()
            lblSM.Text() = M.ToString()
            lblPI.Text() = (2 * N / M).ToString()
        End If
    End Sub

    Private Sub txtN_TextChanged(…, …) Handles txtN.TextChanged
        Refresh()
    End Sub
End Class
```

Uit de waarden voor YM en W berekenen we de coördinaten van de uiteinden A en B van de speld. Het programma tekent hiermee de speld op het beeldscherm.

Een speld treft een lijn als de y-coördinaat van A groter dan of gelijk aan 210 is of als de y-coördinaat van B kleiner dan of gelijk aan 110 is. De x-coördinaten van A en B doen er dan in het geheel niet toe.

Het programma tekent als ‘speelveld’ vier evenwijdige lijnen met een onderlinge afstand van 100. De spelden zijn trouwens ook 100 lang. Als alle spelden geworpen zijn kunnen door een toets in te drukken de waarden voor n, k en π worden afgelezen.

De waarden worden in het programma van VB 2008 direct afgelezen, zodra er een cijfer is ingetoetst (zie de TextChanged event). Deze keer kon een Teken knop met een Refresh() niet worden gebruikt, omdat door elke muisbeweging anders de Paint event aangeroepen zal worden.

5. Prooi-roofdierpopulaties

Het laatste educatieve programma is een (deterministische) simulatie van een ecologisch systeem. Het is een demonstratieprogramma voor een biologieles.

Het ecologische systeem bevat gras, hazen en vossen. Tussen deze drie ecologische componenten gelden de volgende betrekkingen:

1. De hazen eten gras en de vossen eten hazen.
2. Als er meer gras groeit, neemt ook het aantal hazen toe.
   Deze hazen eten echter van het gras en verminderen zo hun eigen groei.
3. Als er meer hazen komen, neemt ook het aantal vossen toe.
   Omdat vossen hazen eten, verminderen zij zelf hun groei, net zoals bij de hazen en het gras.

Als we het aantal hazen op een bepaald tijdstip t aangeven met h(t) en het aantal vossen met v(t), kunnen we, volgens Lotka en Volterra (1920), voor het aantal hazen en vossen op tijdstip t+1 de volgende vergelijkingen opstellen:

$h(t+1) = h(t) + a.h(t) - b.h(t).v(t)$ vergelijking (1)
$v(t+1) = v(t) + c.v(t).h(t) - d*v(t)$ vergelijking (2)

De toename van het aantal hazen tussen de tijdstippen t en t+1 is evenredig met het aantal hazen op tijdstip t, dus:

$$\underbrace{h(t+1) - h(t)}_{\text{toename hazen}} = \underbrace{a}_{\text{groeifactor}}.\underbrace{h(t)}_{\text{aantal hazen op tijdstip t}}$$

De afname van het aantal hazen is evenredig met het aantal aanwezige hazen (natuurlijk verloop) en met het aantal vossen, want die eten hazen, dus:

$$\underbrace{h(t+1) - h(t)}_{\text{groei van het aantal hazen}} = \underbrace{a.h(t)}_{\text{toename aantal hazen}} - \underbrace{b.h(t).v(t)}_{\text{afname aantal hazen}}$$

Dit is vergelijking (1). We nemen in het programma aan dat er altijd genoeg gras voor de hazen voorhanden is.

De toename van het aantal vossen is evenredig met het aantal aanwezige vossen en met het aantal hazen (prooi). De afname van het aantal vossen is alleen evenredig met het aantal, omdat in dit systeem de vossen zelf geen prooidieren zijn. Voor de vossen krijgen we dus:

$$\underbrace{v(t+1) - v(t)}_{\text{groei van het aantal vossen}} = \underbrace{c.v(t).h(t)}_{\text{toename aantal vossen}} - \underbrace{d.v(t)}_{\text{afname aantal vossen}}$$

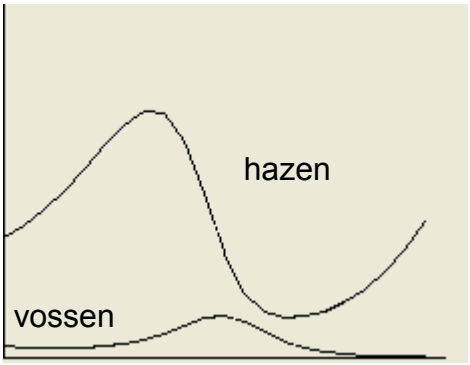
De vergelijkingen zijn als zogeheten differentievergelijkingen opgesteld. Het gras speelt, zoals gezegd, eigenlijk geen rol; er is altijd genoeg om alle hazen te voeden.

Bij het draaien van het simulatieprogramma hebben we de volgende waarden gekozen:
X (= h(0)) = 200; Y (= v(0)) = 20; a=0,3; b=0,01; c=0,002 en d=0,5.

De grafiek die het programma tekent laat heel mooi de groei en de afname van de hazen- en vossenpopulaties zien. Het aantal hazen groeit eerst, bereikt een maximum en neemt vervolgens af. De vossen groeien ook, maar later, bereiken later een maximum en nemen dan ook af, waarna de cyclus zich herhaalt. In de biologie noemen we dit een dynamisch evenwicht.

U hoeft de waarden voor de coëfficiënten a, b, c en d maar iets te veranderen of het systeem kan ontregeld worden. Hierbij neemt òf het aantal hazen enorm toe òf alle hazen en vossen sterven snel uit. Deze eenvoudige simulatie toont aan hoe desastreus een kleine ingreep in een bestaand ecologisch systeem dat in een dynamisch evenwicht is kan zijn.

```
100 'programma 40    ROOFDIER-PROOIDIER-SYSTEEM
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "BEGINPOPULATIE PROOIDIEREN (200) "; X
150 INPUT "BEGINPOPULATIE  ROOFDIEREN (20)  "; Y
160 INPUT "GROEIFACTOR  PROOIDIEREN   (.3)  "; A
170 INPUT "AFNAMEFACTOR PROOIDIEREN   (.01) "; B
180 INPUT "GROEIFACTOR   ROOFDIEREN   (.002)"; C
190 INPUT "AFNAMEFACTOR  ROOFDIEREN   (.5)  "; D
200 CLS
210 LINE (FNX(0),320)-(FNX(320),320),1
220 LINE (FNX(0),320)-(FNX(0),0),1
230 K=.3 : H=.5 : V=320
240 FOR X1=0 TO 320 STEP 10
250    XB=X+(A*X-B*X*Y) : YR=Y+(C*X*Y-D*Y)
260    '                POP. PROOIDIEREN TEKENEN
270    Y1=INT(V-K*X+H):X2=X1+10:Y2=INT(V-K*XB+H)
280    IF Y2 < 0 OR Y2 > 320 THEN GOTO 360
290    LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
300    '                POP.  ROOFDIEREN TEKENEN
310    Y1=INT(V-K*Y+H):X2=X1+10:Y2=INT(V-K*YR+H)
320    IF Y2 < 0 OR Y2 > 320 THEN GOTO 360
330    LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
340    X=XB : Y=YR
350 NEXT X1
360 A$=INKEY$: IF A$="" THEN 360
370 CLS: KEY ON: END
```



Voor Visual Basic 2008 heb ik deze keer alleen een deel van het programma hieronder staan met dezelfde variabelen. Deze variabelen moeten weer de ingevoerde waarden krijgen net zoals het in de vorige VB programma's staat, dat wil zeggen: de tekstboxen met de Text eigenschappen moeten aan de variabelen worden toekend.

Merk op dat de For lus niet tot 320 loopt, maar tot 200. Toen ik hem wel liet herhalen tot 320 werd wel de juiste grafieken getekend, maar na X1 = 200 begonnen de grafieken op een hele vreemde manier te reageren, en waarom de VB code dezelfde grafieken tekent met een For lus tot 200 als bovenstaande GW-BASIC code die een FOR lus heeft tot 320 heb ik geen verklaring voor. Alles heb ik vergeleken en beide Basic dialecten doen toch hetzelfde tekenwerk.

```
With e.Graphics
     .DrawLine(Pens.Black, 0, 320, 220, 320)
     .DrawLine(Pens.Black, 0, 320, 0, 0)
     For X1 As Integer = 0 To 200 Step 10
          Dim XB As Double = X + (A * X - B * X * Y)
          Dim YR As Double = Y + (C * X * Y - D * Y)
          ' Populatie prooidieren tekenen
          Dim Y1 As Integer = Int(V - K * X + H)
          Dim X2 As Integer = X1 + 10
          Dim Y2 As Integer = Int(V - K * XB + H)
          If Y2 < 0 Or Y2 > 320 Then Exit For
          .DrawLine(Pens.Black, X1, Y1, X2, Y2)
          ' Populatie roofdieren tekenen
          Y1 = Int(V - K * Y + H) : X2 = X1 + 10
          Y2 = Int(V - K * YR + H)
          If Y2 < 0 Or Y2 > 320 Then Exit For
          .DrawLine(Pens.Black, X1, Y1, X2, Y2)
          X = XB : Y = YR
     Next
End With
```

Samenvatting

Dit waren de 40 programma's uit het boek '40 grafische programma's in IBM- en GW-BASIC'. Deze uitgave van Academic Service was zeer interessant en leerzaam. Van Academic Service kreeg ik de auteursrechten en kon ik de uitgave in de nieuwsbrieven overschrijven samen met nieuwe oplossingen en tips, en met een BASIC versie erbij, Visual Basic 2008, om deze programma's in deze tijd alsnog uit te kunnen proberen.

In de volgende nieuwsbrief komt de appendix. In dat appendix zullen er uitbreidingen komen van verschillende programma's.

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

Delphi en Basic.NET – Het Canvas tekengebied (1).

Bijna elke applicatie gebruikt het beeldscherm om de gegevens die worden gemanipuleerd weer te geven. Hier leg ik de beginselen van zelf tekenen in Delphi en Basic.NET uit. Als basis voor de materie wordt eerst algemeen het hoe en waarom uitgelegd van schermopbouw en tekenen. Daarna volgt een beschrijving en uitleg van een eenvoudig programma waarin enkele veel voorkomende tekenopdrachten centraal staan.

Dit onderwerp is bedoeld voor iedereen die wil beginnen met tekenen in Delphi en Basic.NET, maar ook voor iedereen dat algemeen (meer) achtergrondinformatie over schermopbouw en tekenen wil opdoen.

Vereist gereedschap
In dit onderwerp wordt er vanuit gegaan dat u op de hoogte bent van handelingen zoals het plaatsen en gebruiken van standaard *controls*, het instellen van properties en het aanmaken van een standaard *eventhandler*. Ook de begrippen zoals *methode* en *private* moeten bekend en vertrouwd zijn. Verder is het handig om te weten dat een programma of venster wordt aangestuurd door *messages* en dat deze messages door *messagehandlers* worden verwerkt. Aannemen dat dit zo werkt volstaat ook..

Voor het kunnen namaken van het voorbeeldprogramma is elke standaard Delphi-installatie vanaf Delphi 5 en elke Basic.NET-installatie vanaf Visual Basic 2003 voldoende.

Tekenen: definitie

De term tekenen omvat in dit verband alle schermoperaties: tekst schrijven, inkleuren, lijnen trekken, cirkels tekenen, figuren weergeven, enzovoort…

Het beeldscherm waarop we tekenen is, zoals ongetwijfeld bekend, opgebouwd uit een aantal pixels welke gelijk is aan de schermresolutie, bijvoorbeeld 1024 pixels breed en 768 pixels hoog. Alles wat op uw beeldscherm wordt weergegeven, wordt getekend door deze pixels in te kleuren. Dit lijkt misschien logisch of overbodig te vermelden, maar het is voor de begrijpelijkheid van de rest van dit onderwerp belangrijk om de betekenis hiervan helder te hebben. Uw monitor heeft slechts één scherm waarop getekend kan worden. Dus als er getekend wordt, dan wordt er over het vorige heen getekend en is dat wat er eerst stond gewoon verdwenen.

De Windows- en programmeerterm voor tekenen is *painting* of *drawing*. Alles wat niet automatisch door uw programma wordt getekend, maar waar u als programmeur zelf zorg voor draagt wordt *custom painting* of *custom drawing* genoemd. Vanwege gemak en het Nederlandstalig karakter vallen al deze begrippen in dit onderwerp onder de term tekenen.

Het beeldscherm wordt aangestuurd door de videokaart. De videokaart wordt met commando's aangestuurd door het besturingssysteem[1]. Een citaat uit de Win32 Developer's Reference[2].

Het tekenen via Windows vraagt kennis van WinAPI[3]. Gelukkig nemen Delphi en Basic.NET een groot deel van het programmeren via WinAPI voor onze rekening, zodat we op een makkelijke manier via het besturingssysteem iets op het scherm weer kunnen geven.

1) De videokaart kan ook rechtstreeks door uw applicatie aangestuurd worden, maar die mogelijkheid valt ver buiten de doelstelling van dit onderwerp.
2) In Delphi bereikbaar via het Help-menu "Windows SDK" (Delphi 5). In Basic.NET zit dit ingekapseld in de Help-lijst, waar alles over het Framework systeem te vinden is.
3) Windows Application Programming Interface. Veel van de WinAPI-functies zijn in Delphi toegankelijk gemaakt via de Windows-unit. In Basic.NET is dit toegankelijk gemaakt via de klasse System.Windows.

Tekenen: de aansturing
Hoe weet het besturingssysteem wat er getekend moet worden?

Stel dat u een lijn op uw Form wilt tekenen. Stel vervolgens dat u het Form minimaliseert, of dat u een ander Form over uw Form plaatst. We weten dat de lijn nu weg is, dus hoe weet het besturingssysteem dat de lijn opnieuw getekend moet worden als uw Form weer de focus krijgt? We zouden de volgende twee mogelijkheden kunnen bedenken:

1. het besturingssysteem onthoudt dat de lijn is getekend en tekent hem opnieuw vanuit het geheugen,
2. het besturingssysteem onthoudt dit niet, maar berekent wat er getekend moet worden.

De eerste mogelijkheid valt al snel af als we bedenken dat dat veel geheugen zou vragen. Ook zou deze mogelijkheid intensieve communicatie over en weer met het geheugen vragen om de wijzigingen van het venster bij te houden (denk bijvoorbeeld aan bewegende beelden).[4]

Dus er wordt elke keer opnieuw berekend wat er getekend moet worden. Nu besteed Windows deze berekening op een handige manier uit. Windows laat aan het Form dat (opnieuw of deels) in beeld komt weten dat het zichzelf moet tekenen. Dit doet Windows met een bericht in de vorm van een WM_PAINT message:

- Een toepassing wordt getekend in een venster op verschillende tijden: tijdens het maken van een venster, bij het wijzigen van de grootte van het venster, wanneer het venster achter een ander venster verplaatst, minimaliseren of maximaliseren van het venster, bij het weergeven van gegevens uit een geopend bestand als tijdens het schuiven, wijzigen, of een gedeelte van de weergegeven gegevens te selecteren. Windows beheert acties zoals het verplaatsen en vergroten/verkleinen van een venster. Als een bewerking invloed heeft op de inhoud van het venster, markeert Windows het getroffen gedeelte van het venster als klaar voor het bijwerken en, bij de volgende gelegenheid, een WM_PAINT bericht verzendt naar het venster.

Het Form “vangt” dit bericht af met een WM_PAINT messagehandler en reageert navenant door zichzelf te tekenen. Hoe dit tekenen in zijn werk gaat komen we in het voorbeeldprogramma op terug. Voorlopig is het voldoende om te weten dat het Form zichzelf opnieuw tekent als het een WM_PAINT message van Windows krijgt.

Wanneer moet het Form opnieuw getekend worden? De redenen op opnieuw te moeten tekenen, zoals hierboven genoemd in het voorbeeld met de lijn, zijn vanzelfsprekend: bij het verkrijgen van de focus als het daarvoor niet zichtbaar was (je activeert het Form bijvoorbeeld vanaf de taakbalk). Maar er moet vaker opnieuw getekend worden dan u zult denken! Zelfs een actief Form wordt veelvuldig opnieuw getekend. Denk maar eens aan het scrollen van het Form: waar komt de nieuwe pagina vandaan? Of als u uw Form iets vergroot: waar komt het nieuwe gedeelte van het Form vandaan? Of als u uw Form iets verkleint: waar komt het bureaublad vandaan? Bij al deze acties wordt er steeds weer door Windows aan het Form of het achterliggende venster gevraagd zichzelf opnieuw te tekenen. *Dit zijn dus nogal wat messages!* Uiteraard is het van groot belang om uw tekenopdrachten zo efficiënt mogelijk te houden, juist omdat die als gevolg van al die messages zo ongelofelijk vaak worden uitgevoerd.

Eigen tekenopdrachten zullen we dus op één of andere manier moeten toevoegen aan de WM_PAINT messagehandler. Deze messagehandler zit diep verborgen in een private methode van een voorouder van TForm:

```
private
     procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
```

In het Framework zit deze messagehandler diep verborgen in een private method van een voorouder van Form:

```
Private _
    Sub WMPaint(ByRef Message As TypeWMPaint) Handles WM_PAINT
```

De voorouder van het Form type is het System.Windows.Forms, ook al verschilt dat wat tussen Delphi en Basic. De aanroep van de messagehandler zal echter hetzelfde blijven.

Omdat een dergelijke private methode voor u als gebruiker van het Form niet bereikbaar is, “verlengen” Delphi en Basic deze handler met behulp van een *OnPaint event* (zie het tabblad events in de Object Inspector). Indien u wilt dat er nog meer getekend wordt dan alleen de standaard ingekleurde achtergrond, bijvoorbeeld de hierboven veronderstelde lijn, dan kunt u deze lijn tekenen in uw eigen OnPaint *eventhandler*. Elke keer als het Form nu een WM_PAINT message van Windows krijgt wordt eerst het Form zelf getekend in de messagehandler, wordt vervolgens het OnPaint event van het Form aangeroepen en worden alle door uw toegevoegde tekenopdrachten uitgevoerd.

Beide handlers kunnen meerdere tekenopdrachten hebben, en een tekenopdracht, bijvoorbeeld: “teken lijn”, of: “schrijf tekst”, wordt direct uitgevoerd. U zou kunnen bedenken dat Windows eerst alle opdrachten in het geheugen verwerkt en het resultaat pas laat zien bij het verlaten van de WM_PAINT messagehandler, of bij het verlaten van de OnPaint eventhandler (mits toegewezen), maar dat is standaard niet het geval[5]. We kunnen dus stellen dat het scherm (heel snel) beetje bij beetje wordt opgebouwd.

4) Windows gebruikt wel optimalisaties om vanuit het geheugen te tekenen (bijvoorbeeld bij het verplaatsen van de muis) of om te kunnen beslissen óf er wel opnieuw getekend moet worden. Deze optimalisaties zijn ook aan te sturen d.m.v. het gebruik van o.a. clipping.
5) De techniek om tekenopdrachten eerst naar het geheugen te schrijven wordt Double Buffering genoemd. Vaak wordt dit gebruikt om bijvoorbeeld flikkering tegen te gaan.

Tekenen: schermopbouw
Omdat we eerder al hebben vastgesteld dat een pixel die opnieuw wordt ingekleurd, de oude ingekleurde pixel vervangt, bestaat dus de mogelijkheid dat een tekenopdracht een eerdere tekenopdracht deels of volledig teniet doet. De laatste tekenopdracht voor één bepaalde pixel is dus definitief. Deze werking kan soms lastig zijn, omdat je niet wilt dat vorig tekenwerk wordt overgetekend. Houd daarom altijd rekening met de juiste tekenvolgorde: alleen het laatste is zichtbaar.

Tot dusver hebben we het beeldscherm beschreven, als ware het een krijtbord: je kunt met meerdere kleurtjes krijten tekenen, maar altijd slechts op één plek tegelijk en elke nieuwe tekening vervangt de oude. Zou het niet handig zijn om een ingewikkelde tekening op bijvoorbeeld een stickit-plakkertje te tekenen en deze op het krijtbord te plakken? Voordeel daarvan zou zijn dat je hem niet steeds opnieuw hoeft te tekenen bij eventueel verplaatsen; je pakt het plakkertje gewoon op en plakt hem ergens anders neer. Wel, dergelijke stickit-plakkertjes bestaan in Windows en heten *windowed controls*[6], waarmee het tijd is geworden voor de introductie en uitleg van 3 begrippen binnen Delphi en Basic:

- *WinControl*
- *Canvas*
- *Control*

Een WinControl (van het type TWinControl in Delphi) is, om de vergelijking even te handhaven, het stickit-plakkertje, een windowed control. Voorbeelden van WinControls zijn: TForm, TEdit, TMemo, TButton. Deze controls staan altijd op de voorgrond. Er kunnen uiteraard meerdere WinControls tegelijk zijn, en zelfs ook op dezelfde plaats. In dat geval gaat weer hetzelfde op zoals hiervoor uitgelegd:

de laatste WinControl die getekend wordt staat op de voorgrond. De volgorde waarin de diverse WinControls staan, wordt ook wel *Z-order*[7] genoemd. Hoe hoger de Z-order, hoe later er wordt getekend. De eigenaar van de controls (bijvoorbeeld het Form waar de controls op staan) houdt deze Z-order bij, zodat alles in de juiste volgorde wordt getekend. Deze volgorde kun je als gebruiker wijzigen met de TWinControl methoden BringToFront ("teken later") en SendToBack ("teken eerder").

Belangrijk! Denk erom dat wat hier uitgelegd wordt in Basic.NET precies zo het geval is. Echter bestaat er in het Framework library geen WinControl type.
Alle WinControls zijn ondergebracht in Framework klassen, zoals ik eerder heb laten zien dat het Form van het type System.Windows.Forms.Form is.

Voor de volledigheid is het goed te vermelden dat alleen WinControls de hiervoor besproken WM_PAINT messages kunnen ontvangen. Zoals we weten gaat de betreffende messagehandler de tekenopdrachten verzorgen, echter op een WinControl kan niet getekend worden. Hiermee stopt dan ook voor wat betreft het WinControl de vergelijking met het stickit-plakkertje. We hebben dus een ander hulpmiddel nodig....

Het Canvas (van het type TCanvas) is het krijtbord waarop wel getekend kan worden. De WinAPI term voor Canvas is *Device Context (DC)*. Een Canvas is onderdeel van een WinControl en vormt daarvan de achtergrond. Een Canvas is een in formaat onbeperkt oppervlak welke door het formaat van het WinControl wordt begrensd. Dit wetende is alles op het scherm uiteindelijk dus weer één groot Canvas, opgedeeld in meerdere kleinere canvassen, behorend bij de vele verschillende WinControls.

Net zoals bij de WinControl zal de vergelijking met Basic.NET hetzelfde zijn. In het Framework is er echter geen Canvas type, maar er bestaat wel een ander soort achtergrond dat als een Canvas zal werken: het Graphics object. Ook het Graphics object is onderdeel van een WinControl, vandaar dat alleen in de Paint event toegang is tot het Graphics object met het parametertype *PaintEventArgs*.

Om te kunnen tekenen op een Canvas, biedt deze de volgende gereedschappen:

- een *Pen*, waarmee lijnen, bogen en punten getekend kunnen worden,
- een *Brush*, waarmee kan worden ingekleurd,
- een *Font*, waarmee het lettertype voor het tekenen van tekst wordt bepaald.

Elk gereedschap heeft zo weer zijn eigen eigenschappen en instellingen, welke we verderop gaan behandelen in het voorbeeldprogramma. Voorlopig is het voldoende om te weten dat een Canvas deze gereedschappen biedt en dat het altijd op de achtergrond ligt.

Voor zover de vergelijking met het krijtbord tot nu toe stand hield, houdt deze vergelijking bij het Control (van het type TControl) zeker op. Het Control is geen WinControl en heeft ook geen Canvas. Een control tekent zichzelf op het Canvas van zijn eigenaar die wel een WinControl is. Voorbeelden van Controls: TLabel, TBevel en TImage. Een control heeft net als een WinControl ook een Z-order, maar verliest het daarbij altijd van een WinControl omdat het zichzelf op een Canvas tekent. Een Control kan wel boven een ander Control liggen. [8]
Ook dit is in Basic.NET precies zo het geval. Het Control type is alleen geen TControl, maar van het type System.Control.

6)
   a. Het kunnen oppakken en verplaatsen van een tekening is niet de hoofdreden van het bestaan van windowed controls.
   b. In WinAPI wordt elk windowed control kortweg "window" genoemd.
7) De Z in Z-order slaat op de virtuele Z-as van het coördinatenstelsel waarin getekend wordt, welke loodrecht door je beeldscherm wordt gedacht te lopen.

8) Gemakshalve wordt onder algemeen gebruik de term "controls" zowel die van het type TControl als die van het type TWinControl verstaan. Binnen Delphi zijn beide ook hiërarchisch met elkaar verbonden: een TWinControl is een TControl, TControl is de voorouder van TWinControl.

Tip! Raadpleeg het .NET Framework objectlibrary in Visual Basic MSDN Help voor meer informatie over de hiërarchie van de controls. De hiërarchie van Delphi kan anders in elkaar zitten dan die van het Framework, ook al, zoals ik eerder al zei, is er geen verschil met de theorie.

Tot zover de theoretische achtergrondinformatie bij het tekenen.

In de volgende nieuwsbrief gaan we het voorbeeldprogramma maken. Het voorbeeldprogramma zal niet in één keer worden getoond, maar deze zal stap voor stap uitgelegd worden wat we doen en wat er gebeurd. Er zullen codeblokken eerst in Delphi weergegeven worden die ik daarna in Basic.NET laat zien.

Klik op de gegeven link van *nldelphi* als u interesse heeft in Delphi en graag wat meer wilt weten. Net zoals Basic is Delphi geen moeilijke programmeertaal en is het zelfs een goede overstap wanneer u verder wilt dan alleen Basic.

Marco Kurvers

Cursussen

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

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

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

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

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

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

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

Software

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

Hoe te bestellen

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

ABN-AMRO nummer 49.57.40.314
HCC BASIC ig
Haarlem

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

Alle prijzen zijn inclusief verzendkosten voor Nederland en België.

Vraagbaken

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

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