Nieuwsbrief

18de jaargang september 2011 **Nummer 3**

HCC BASIC interessegroep

Inhoud

Deze uitgave kwam tot stand met bijdragen van:

Naam	Blz
Gordon Rahman	14 en 16

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

Deze zomer was helaas nat en koud. Niet deze keer maart, maar deze zomer trok aan ze staart!
Daarom zal de derde nieuwsbrief van dit jaar gezellig uitkomen met weer veel leer- en leesplezier.

Over webpagina's maken met de Crimson Editor sla ik even een tijdje over. Er komt een mooi onderwerp over events, geschreven door Gordon Rahman. Ik wil de nieuwsbrief niet al te groot maken.

De volgende grafische programma's zullen deze keer zeer wiskundig zijn. Het gaat nog steeds over 3D tekenen.
Het is ook erg lastig geweest om de voorbeelden in Visual Basic .NET om te zetten. Tot nu toe is dat steeds gelukt.

Marco Kurvers

BASIC leren – PowerBASIC (1).

We kennen BASIC niet beter als een oude programmeertaal met een interpreter. Het was een scherm dat werkte als een 'directe mode'. Het voordeel was dat formules achter de PRINT statements direct uitgevoerd werden en problemen snel gevonden konden worden. Nu moeten we debuggen tijdens de uitvoer van een programma om een probleem te kunnen vinden.

Interpreters.

Echter zijn interpreters langzaam. Ze gebruiken teveel tijd om uit te vinden wat er gedaan moet worden. Om de programma statements uit te voeren moet de interpreter eerst elk statement inlezen (Wat vraagt men aan mij wat ik hier moet doen?) en dan pas de operatie van kracht te brengen. De statements binnen in de lussen worden herhaaldelijk ingelezen. Bekijk dit drieregelig programma eens:

```
10 FOR n = 1 TO 1000
20   PRINT n, SQR(n)
30 NEXT n
```

De BASIC interpreter zal voor de eerste keer door dit programma uitvinden dat regel 20 bedoeld wordt:

1. Converteer numerieke variabele *n* naar een string.
2. Zend de string naar het scherm.
3. Verplaats de cursor naar de volgende print zone.
4. Bereken de vierkantswortel van *n* (square root).
5. Converteer het resultaat naar een string.
6. Zend de string naar het scherm.

Voor de tweede keer door de lus zal het allemaal nogmaals uitgevonden moeten worden, totaal vergeten wat het een milliseconde daarvoor geleerd had over deze regel. Evenzo in de volgende 998 keren.

Compilers.

Een compiler is een tekst-naar-machinetaal vertaler die de brontekst leest, de ontwerpsyntaxis van de taal vergelijkt en de machinetaal produceert. De machinetaal uitvoer (genoemd *objectcode*) wordt dan uitgevoerd in onafhankelijke stappen. Met andere woorden, een compiler voert niet de programma's uit, maar het bouwt ze.

Om met een compileertaal zoals PowerBASIC te werken, zult u in twee termen over uw programma's moeten denken: compileertijd en uitvoertijd. Met een interpreter is er alleen maar de uitvoertijd.

De snelheidstoename bij PowerBASIC hangt af van het programma. De meeste programma's zullen vier tot tien maal sneller zijn dan hun interpretief equivalent. Als je er op aanstuurt is een 100 voudige snelheidstoename in sommige gevallen mogelijk. Programma's die echter de meeste tijd besteden aan het wisselen tussen allerhande bestanden of op invoer moeten wachten zullen een minder spectaculaire snelheidstoename te zien geven.

BASIC interpreters hebben regelnummers nodig om de weg te vinden wanneer GOTO's en GOSUB's uitvoer zenden naar een statement dat niet sequentieel de volgende zal zijn. Daarnaast vormen de regelnummers de kern van het wijzigingsproces.

Toegegeven, PowerBASIC heeft geen regelnummers nodig. In plaats van GOTO 5000 kunt u in PowerBASIC iets soortgelijks zeggen als GOTO *ErrorExit*, waar *ErrorExit* een label is voor een start van een *ErrorExit* routine.

De omgeving gebruiken.

Om meer te begrijpen, zullen we eens onderstaand voorbeeld gebruiken. Vanuit het hoofdmenu, kies **F**ile en dan **O**pen. Typ als bestandsnaam MYFIRST en druk op *Enter*. U bent nu in het invoervenster.

Typ het programma in precies zoals het hieronder staat, want PowerBASIC doet uitsluitend wat u hem vertelt.

```
WHILE -1
    FOR n = 1 TO 8
```

```
        READ a$
        PRINT a$ " ";
    NEXT n
    PRINT
    RESTORE
WEND
DATA Presenting, PowerBASIC--, The, Fastest, BASIC, on, Earth
```

Druk op F2 om uw programma op te slaan (dit is een sneltoets voor **F**ile/**S**ave).
Als u daarvoor nooit in PowerBASIC geprogrammeerd heeft, zult u merken dat bij dit programma de regelnummers ontbreken. Regelnummers spelen een aparte rol voor het wijzigen en het uitvoeren van interpretive BASIC programma's. Echter, sinds PowerBASIC een inbouwvenster editor met een bekwaam zoekkenmerk heeft, en aangezien het een smart compiler is, zijn regelnummers niet meer nodig. Maar PowerBASIC is een onbevangen vak op zich. U kunt nog steeds elke regel nummeren, of elke derde regel, op volgorde of achterwaarts. Of u kunt regels kiezen om te nummeren dat het doel is van een GOTO of GOSUB. Maar het is beter om labels te gebruiken die duidelijke namen bevatten.

De tekstversie is net zo als het bronprogramma (ook zo als *broncode* en *bronbestand*). Het wordt genoemd als *bron* omdat dit het enige is waar alle andere vormen van het programma van gemaakt zijn, en het is alleen die vorm die voor mensen begrijpelijk is. Voordat uw computer MYFIRST kan maken, moet deze brontekst vertaald of gecompileerd worden in een vorm die de computer begrijpen kan – de machinetaal.

Verlaat nu de editor. Druk op F10 om naar het hoofdmenu te gaan. U kunt nu een optie selecteren. F10 wisselt van het hoofdmenu en het invoerscherm. U kunt ook uw muis gebruiken: Klik op het hoofdmenu of een andere menunaam. Klik op het invoervenster om terug te keren om te kunnen wijzigen.

Zolang het invoerscherm niet geselecteerd is, dus niet toegankelijk is, kunt u er niet mee werken. Maar de tekst van uw programma is in het geheugen aanwezig, klaar om voor een bepaald moment te worden toegevoegd of te worden gewijzigd.

Het programma starten.

Nu het bronprogramma vertaald is in machinetaal, kunt u het gaan testen door te kiezen **R**un/**Run** of de sneltoets *F9* in te drukken.

Hmm, MYFIRST rent door de seconden heen en dan plots krijgt het een oplawaai. Dat komt omdat we een gemaakte fout ingevoegd hebben in MYFIRST als uitgangspunt: wat geanalyseerd werd door de compiler zal tijdens de uitvoer van het programma niet altijd hetzelfde zijn, zodat niet alles wat mogelijk fout zou kunnen zijn onderschept kan worden. De compiler controleert hoofdzakelijk de syntaxis fouten, zoals foutgeschreven commando's, gemiste komma's en ongelijk aantal ronde haken.

Onthoud heel goed dat sommige vergissingen meestal ontdekt worden tijdens compileertijd; anders tijdens de uitvoertijd. PowerBASIC kan u helpen om beide problemen snel te corrigeren.

Over de fout.

Om aan u te vertellen dat een "Out of Data" fout verscheen, zal PowerBASIC laten zien waar de fout was door u te plaatsen in de fout. Omdat MYFIRST.BAS in het geheugen aanwezig is, zal dit snel gebeuren.

De fout wordt gegeven in de READ regel, maar eigenlijk zit de fout daar niet. Het komt door de FOR regel:

```
FOR n = 1 TO 8
```

Waarom? Omdat er maar zeven stukjes in de dataregel werden gegeven. Verander de FOR lus door het getal 8 te wijzigen in getal 7 en U zult nu een werkend programma hebben.

Kies **R**un/**R**un of druk op F9 om de gecorrigeerde versie te starten. Nu zal MYFIRST goed werken en telkens onderstaande zeven stukjes tekst in een regel weergeven.

```
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
```

```
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
Presenting PowerBASIC-- The Fastest BASIC on EARTH
...
```

En zo gaat het maar door. Druk op *Ctrl-Break*. Wanneer u terug bent in de editor dan kunt u altijd de uitvoer terugzien door te drukken op *Alt-F5*.

Programmeren in PowerBASIC.

PowerBASIC is één van de simpelste talen om te programmeren. U hoeft geen ingewikkelde structuren, declaraties en andere syntaxisvormen te leren. Het enige is, u moet de compiler kunnen vertellen wat u wilt. Als u een ruime tijd er voor neemt om het programma te ontwikkelen, zal u later minder tijd hoeven besteden aan het debuggen van of wijzigen aan het programma.

Programma's hebben meestal verschillende soorten informatie (invoer of gegevens), verwerken het, en produceren andere soorten uitvoer. De invoer kan komen uit verschillende bronnen: bestanden, het toetsenbord, een communicatiepoort, een joystick of een lichtpen. De gegevens worden verwerkt door *controlstructuren*. Er zijn drie soorten controlstructuren: sequencers, loops en conditionele takken. Deze risicovolle simpele structuren zijn enorm krachtig. Uitvoer kan verzonden worden, ongeformatteerd of geformatteerd in gevarieerde richtingen, naar een printer, naar het scherm, naar de harde schijf, naar het geheugen of zelfs via een communicatiepoort naar een andere computer.

Onthoud! Als ik het over getallen heb met drijvende komma, moet u weten dat ik de punt beschouw als onze komma, dus niet de notatie die wij normaal gebruiken.

Programma elementen.

PowerBASIC programma's worden gebouwd uit eenregelige statements. Wanneer in uw programma maar één statement in een regel staat en de volgende daarop volgt dan gebruikt u de simpelste controlstructuur, genoemd *sequence*. Programma's mogen ook labels en regelnummers hebben.

Natuurlijk zullen de meer gecomplexeerde programma's gemaakt zijn uit meer gecomplexeerde structuren (zoals twee soorten controlstructuren: loops en conditionele takken, welke in samenwerking zijn met geformuleerde functies, procedures, subroutines en units). Deze meer complexe structuren onderhouden meerdere logische wegen bij het groeperen van uw programmataken.

Statements

De statements zijn de simpelste bouwstenen waarmee we programma's kunnen maken. PowerBASIC heeft meer dan 130 soorten statements (zie de appendix voor een lijst van de statements met allerlei soorten BASIC versies. U kunt hem vinden op de BASIC site. De nieuwe lijsten zijn nog in de maak). Een PowerBASIC regel kan bestaan uit geen, één of meerdere statements, elk gescheiden door een dubbelepunt. Het simpelste programma in PowerBASIC is een eenregelig statement, zoals:

```
PRINT "Hallo wereld"
```

U wilt natuurlijk langere programma's kunnen schrijven. Het combineren van verschillende statements in één regel, gescheiden door dubbele punten (:) , kunt u ook doen. Het is beter om een aparte regel voor elke statement te gebruiken. U kunt commentaar toevoegen aan het einde van een regel en/of regelnummers gebruiken aan het begin van een regel. PowerBASIC programma's beschikken over meerdere brontekstregels, waarvan onderstaande formaten bestaan:

```
[regelnummer] [statement] [:statement]... [' commentaar]
```

of

```
label:
```

of

```
$metastatement
```

Wat niet het geval is bij interpretive BASIC is dat in het algemeen PowerBASIC geen probleem geeft voor spaties en gecommentarieerde regels – spaties, commentaar en lege regels worden genegeerd door de compiler. Daardoor bestaat er ook geen limiet in de lengte van een regel. Wordt toch de regel te lang dan kunt u deze scheiden door een onderstreping (_). Op die manier kunt u de code overzichtelijk houden. Het

maakt voor de compiler niet uit hoe een regel getypt wordt.

Commentaar
Commentaar kan allerlei tekst zijn toegevoegd aan het einde van een regel en gescheiden door een enkel aanhalingsteken ('). Het enkel aanhalingsteken kan gebruikt worden in plaats van :REM om commentaar van de statements te scheiden – hoewel, is het aan het einde van een DATA statement dan zal DATA denken dat het aanhalingsteken een deel is van het laatste item van het statement. Om een DATA statement te kunnen commentariëren hebt u een dubbelepunt en een enkel aanhalingsteken nodig. DATA statements commentariëren is zeer aan te bevelen. Kunt u echt onthouden waar al die onderstaande nummers voor dienen?

```
DATA 130, 140, 150   :' x, y, z coördinaten, ruimteschip 1
DATA 135, 156, 157   :' x, y, z coördinaten, ruimteschip 2
```

Voor ander gebruik is het niet verplicht om een enkel aanhalingsteken te scheiden achter een statement door een dubbelepunt. Onderstaande regels zijn voor de ogen van de compiler hetzelfde:

```
area = radius^2 * 3.14159      ' bereken het gebied
area = radius^2 * 3.14159      : REM bereken het gebied
```

Commentaar kan ook teveel worden. Er is bijvoorbeeld niet altijd commentaar nodig bij duidelijke code:

```
x = x + 1     ' tel 1 op bij x
```

Is er een speciale reden dan kan commentaar best wel een helpende hand zijn:

```
amt = amt * 1.081    ' regelt de omzetbelasting
```

Een nog duidelijke manier is om een extra variabele te definiëren:

```
salestax = .081
amt = amt * (1 + salestax)
```

Een potentieel gevaar van commentaar is, dat als de code verandert, dat de tekst niet meer klopt. Ook heeft U code en commentaar zonder enige relatie. Slecht commentaar is minder goed dan er geen een te gebruiken, omdat die andere programmeurs kunnen misleiden en geen goed vertrouwen geeft.

Regelnummers
Zoals eerder verteld gaat PowerBASIC zeer makkelijk om met regelnummers. Ze mogen worden gebruikt, of zelfs wanneer het alleen maar nodig is. Ze hoeven ook niet op volgorde in het programma te staan. Regelnummers worden geaccepteerd tussen de 1 en 65535 en mogen niet meermalen met hetzelfde nummer voorkomen.

Labels
Het gebruik van labels in plaats van regelnummers maakt de control flow van uw programma meer leesbaarder, bijvoorbeeld:

```
GOSUB BuildQuarks
```

vertelt veel meer dan:

```
GOSUB 1723
```

Elke label moet apart in een regel aanwezig zijn (commentaar mag erop volgen) en identificeert het statement direct die erop volgt. Labels moeten beginnen met een letter en kunnen dan elk nummer of letter en een cijfer bevatten. Labels zijn niet hoofdlettergevoelig – *THISLABEL*, *thislabel* en *ThisLabel* zijn allemaal hetzelfde.
Achter een label moet een dubbelepunt staan, maar een statement die een label heeft, zoals een GOSUB, mag geen dubbelepunt hebben.

Onderstaand voorbeeld is goed:

```
PRINT "Sorteer nu de facturen"
```

```
GOSUB SortFacturen   'Dit roept een label
PRINT "Alles gedaan!"
END

SortFacturen:    'Dit is een legaal label
   ...
RETURN
```

Illegaal gebruik:

```
Sluitpunt: a = a + 1    'Illegaal; labels moeten apart staan
```

De volgende regel wordt wel geaccepteerd:

```
1010 a = a + 1   'regelnummers kunnen op dezelfde regel
```

Metastatements
Metastatements werken op een andere manier dan de standaardstatements. Het zijn commando's voor de compiler, anders dan de rol van het programma. Deze statements beginnen altijd met een dollarteken ($). Standaardstatements besturen de computer in draaitijd (run time); metastatements, zoals de selecties in de **O**ptions/**C**ompiler menu, besturen de compiler in compileertijd.

Bijvoorbeeld, het $INCLUDE metastatement laat de compiler de inhoud van een ander bestand op dat punt introduceren. Er kan maar één metastatement per regel staan en er kunnen meerder opties in een metastatement voorkomen.

De volgende regels zijn in PowerBASIC toegestaan:

```
Start: ' alleen een label
10     ' alleen regelnummers
$INCLUDE "CONST.INC"    ' metastatement
20 a = a + 1     ' regelnummer, statement
a = a + 1 : b = b + 1   ' twee statements
30 a = a + 1 : b = b + 1 : c = a + b  ' regelnummer, 3 statements
```

We weten dat de laatste regel goed is. Deze kunnen we leesbaarder maken door de drie statements te veranderen in:

```
voeginhoudtoe:
a = a + 1
b = b + 1
c = a + b
```

Tekens en symbolen
De letters *A* t/m *Z* of *a* t/m *z* en de cijfers 0 t/m 9 kunnen worden gebruikt in identifiers, of benoemde labels, variabelen, procedures en functies.

De cijfers 0 t/m 9; de symbolen ., + en -; en de letters *E*, *e*, *D* en *d* kunnen allemaal gebruikt worden in numerieke constanten. De letters *A*, *B*, *C*, *D*, *E*, *F*, en de kleine letters ervan kunnen gebruikt worden als hexadecimale integer constanten (wanneer erbij gevoegd beginnend met een &H).

Benoemde variabelen
Strings, numerieke variabelen en arrays kunnen dezelfde naam delen, bijvoorbeeld *fog!*, *fog$$* en *fog%(5)* zijn aparte variabelen. De één is een singel-precisie drijvende punt variabele, één is een flex string en de ander is een element van een integer array. Functies, procedures, subroutines en andere labels kunnen geen namen delen met anderen, ook kunnen ze de namen niet delen met variabelen.

Anders dan proberen te herinneren welke namen ook alweer gedeeld konden worden en welke niet, kunt u beter de optie voor het gebruik van unieke namen uitzetten. Dit maakt uw code meer begrijpelijker wanneer u later in uw programma terugkomt.

Integers (%)
De simpelste en snelste getallen in PowerBASIC programma's zijn integers. Een integer is een getal zonder

decimaal punt (wiskundigen noemen het gehele getallen) van –32768 t/m +32767. Deze waarden vallen onder 16-bit integers: 32768 is 2^{15}.

Integers zijn geïdentificeerd als een variabele naam met een procentteken, bijvoorbeeld var%, of bij gebruik van het DEFINT statement. Bijvoorbeeld, als u onderstaande declaratie gaat gebruiken

```
DEFINT I, J, K
```

zullen alle variabelen die met de letters I, J of K beginnen standaard integers zijn.
U kunt ook integer variabelen declareren met gebruik van het INTEGER sleutelwoord met het DIM statement, als voorbeeld:

```
DIM I AS INTEGER
```

Dit verkleint wel de limiet bij gebruik van integers, echter voor sommige applicaties is het toch voldoende. In alle programma's zullen maar enkele variabelen (zoals de tellers in FOR/NEXT lussen) functioneren met deze beperkingen. Het gebruik van integers werkt zeer snel en verkleint de code. Dit komt vooral omdat elke integer maar 2 bytes geheugen in beslag neemt.

Long integers (&)
Net als de reguliere integers kunnen de long integers ook geen decimale punt hebben. Maar het bereik is een stuk groter, van –2147483648 t/m +2147483647 (-2^{31} t/m $2^{31} – 1$). Dit bereik kost meer ruimte, want long integers nemen 4 bytes geheugen in beslag maar in de modernere versies van PowerBasic werken zij sneller dan integers.

Een long integer wordt genoemd door een variabele naam met een ampersand (var&) of door gebruik van het DEFLNG statement. U kunt ook long integers declareren door het LONG sleutelwoord te gebruiken met het DIM statement, bijvoorbeeld:

```
DIM I AS LONG
```

Quad integers (&&)
Quad integers zijn 64-bit (8-byte) integers met teken (dubbel zoveel bits als long integers) met een bereik van $–2^{63}$ t/m $2^{63} – 1$.

Quad integer variabelen worden genoemd door een variabele naam met twee ampersands (var&&) of door gebruik van het DEFQUD statement. U kunt ook een quad integer variabele declareren door gebruik van het QUAD sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS QUAD
```

Ook al heeft een quad integer een precisie van 19 cijfers, een juistheid van 18 cijfers kan met een PRINT of een STR$ weergegeven worden. Een 19 cijferwaarde zal afgerond worden tot 18 cijfers in wetenschappelijke notatie. Normaal gesproken werken PRINT en STR$ tot standaard 16 cijfers, dus de geavanceerde vorm van STR$, STR$(var, 18), moet gebruikt worden om de 17^{de} en 18^{de} cijfer van een quad integer weer te kunnen geven.

Byte (?)
Bytes zijn 8-bit (één byte) integers zonder teken met een bereik van 0 t/m 255.

Byte variabelen worden genoemd door een variabele naam met een vraagteken (var?) of door gebruik van het DEFBYT statement. U kunt ook byte variabelen declareren door gebruik van het BYTE sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS BYTE
```

Byte variabelen zijn zeer goed te gebruiken voor het opslaan van kleine integers zonder teken zoals een aantal dagen in een maand. Ze zijn ook zeer nuttig voor het leveren van waarden voor PEEK en POKE.

Word (??)
Words zijn 16-bit (twee byte) integers zonder teken met een bereik van 0 t/m 65535.

Word variabelen worden genoemd door een variabele naam met twee vraagtekens (var??) of door gebruik van het DEFWRD statement. U kunt ook word variabelen declareren door gebruik van het WORD sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS WORD
```

Word waarden zijn zeer nuttig voor het leveren van de offset voor PEEK en POKE.

Double Words (???)
Double words zijn 32-bit (vier byte) integers zonder teken met een bereik van 0 t/m 4294967295.

Double word variabelen worden genoemd door een variabele naam met drie vraagtekens (var???) of door gebruik van het DEFDWD statement. U kunt ook double word variabelen declareren door gebruik van het DWORD sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS DWORD
```

Net als word waarden en integers; een double word heeft een hoger positief bereik dan een long integer en heeft nochtans alleen vier bytes nodig. Double word waarden zijn zeer nuttig voor het bepalen van absolute geheugenadressen voor PEEK en POKE.

Single precision floating point (!)
Single precision floating point getallen (of simpelweg single precisie) is het meest veelzijdig numeriek type binnen in PowerBASIC. Single precisie waarden kunnen decimale punten bevatten en hebben een bereik van $\pm 8.43 \times 10^{-37}$ t/m $\pm 3.37 \times 10^{38}$.

Single precisie variabelen worden genoemd door een variabele naam met een uitroepteken (var!) of met niets (als u geen type indicator gebruikt, zal PowerBASIC het als een single precisiewaarde beschouwen), of door gebruik van het DEFSNG statement. U kunt ook single precisie variabelen declareren door gebruik van het SINGLE sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS SINGLE
```

Wat gebeurt er wanneer uw vingers uitglijden en u heeft de variabele `count!` op de ene plaats in uw programma en `count` zonder type indicator op een andere plaats? Tenzij U in een DEF*type* statement aangeeft dat alle variabelen die met de letter "C" beginnen als standaard van een ander type zijn zal de compiler "Count!" en "Count" als een en dezelfde variabele beschouwen.

Sommige getallen zijn enorm goed te gebruiken. Hoewel, zolang single precisie zowel enorme als microscopische getallen kan presenteren, kan het echter geen getallen onthouden buiten het bereik van het zes cijferige juistheid. Met andere woorden, single precisie werkt goed met gegevens als $451.21 en $6411.92, maar $671421.22 kan niet precies gepresenteerd worden omdat het teveel nummers bevat. Ook nooit kan 23456789 of 0.00123456789. Een single precisie representatie gaat zo dicht mogelijk als het kan in zes cijfers: $671421, 234.568 of 0.001234567.

PowerBASIC gebruikt de standaard IEEE voor drijvende punt berekeningen, niet het rechterlijk Microsoft formaat in gebruik bij interpretive BASIC. Een voorbeeld over dit; als u random files gecreëerd hebt in interpretive BASIC dan moet u gebruik maken van de Microsoft/IEEE converteerfuncties (CVMS, CVMD, MKMS$ en MKMD$) voor lees en schrijf single precisie en dubbel precisie drijvende punt opgeslagen in deze gegevensbestanden.

Double precision floating point (#)
Double precision floating point getallen zijn de single precisie getallen van wat de long integers zijn. Ze nemen twee keer zoveel geheugen in beslag (8 bytes versus 4 bytes), en hebben een consequentie langer te rekenen; ze hebben een groter bereik ($\pm 4.19 \times 10^{-307}$ t/m $\pm 1.67 \times 10^{308}$) en een juistheid van 16 cijfers versus 6 cijfers van de single precisie. De opslag benodigdheid van dubbele precisie getallen worden zeer belangrijk wanneer u het deelt met arrays. Een dubbele precisie 5000 element array gebruikt 40000 bytes. Een integer array met dezelfde aantal elementen gebruikt alleen maar 10000 bytes.

Double precisie variabelen worden genoemd door een variabelenaam met een pondteken (var#) of door gebruik van het DEFDBL statement. U kunt ook double precisie variabelen declareren door gebruik van het DOUBLE statement met het DIM statement, bijvoorbeeld:

```
DIM I AS DOUBLE
```

Extended precision floating point (##)
Extended precision floating point getallen zijn de basis van de meeste gerekende drijvende punten in PowerBASIC, met alle gerekende drijvende punten erbij die de 80x87 rekenprocessors gebruiken.

Turbo Basic 1.x gebruikt extended precisie rekenkundig voor alle gemixte mode expressies, met single- en double precisie toewijzingen. In PowerBASIC is een toewijzing geoptimaliseerd dat afhangt van het wiskundig geselecteerd pakket (Emulatie, NPX of Procedure); PowerBASIC gebruikt niet de extended precisie tenzij het niet anders kan. Extended precisie voorziet van een gedeclareerd gegevenstype zo dat u het voordeel heeft van een extra exponent bereik en precisie, als u die wenst.

Extended precisie waarden hebben elk 10 bytes opslag nodig. Zij hebben een bereik van ongeveer $\pm 3.4 \times 10^{-4932}$ t/m $\pm 1.2 \times 10^{4932}$ en kunnen weergegeven worden tot 18 zinvolle cijfers.

Extended precisie variabelen worden genoemd door een variabelenaam met twee pondtekens (##) of door gebruik van het DEFEXT statement. U kunt ook extended precisie variabelen declareren door gebruik van het EXT sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS EXT
```

PRINT en STR$ werken als standaard tot 16 zinvolle cijfers (genoeg voor een double precisie waarde). Om tot 18 zinvolle cijfers van een extended precisie getal te printen of om het op te slaan in een string, moet u gebruik maken van de geïntensiveerde STR$ functie om 18 plaatsen te kunnen specificeren: `PRINT STR$(a##,18)`.

Binary code decimal (BCD) fixed point (@)
BCD fixed point getallen hebben 8 byte binary gecodeerde decimale weergaven van floating point getallen, hetgeen waarmee U rekening moet houden is, dat altijd de vaste nummers van cijfers aan de rechterkant van de decimale punt staan. Deze getallen hebben een bereik van ongeveer $\pm 9.99 \times 10^{-63}$ t/m $\pm 9.99 \times 10^{63}$, inbegrepen tot 18 zinvolle cijfers. Het aantal "vaste" decimale plaatsen in het getal wordt bepaald door de FIXDIGITS systeemvariabele.

Dit gegevenstype is speciaal te gebruiken voor financiële berekeningen omdat het de afrondfouten vermijdt geassocieerd met single, double en extended precisie getallen. Verschillende getallen moeten een exact vermogen hebben van twee zodat het exact gepresenteerd kan worden. Zolang soorten getallen maar in die categorie vallen, getallen als 1.1 zullen dat niet. Het getal 1.100000023841858 heeft het dichtste vermogen tot twee (1.1). Omdat BCD fixed point getallen de actuele decimale cijfers opslaan als een deel van een getal, zal elk decimaal getal tot 18 cijfers exact gepresenteerd worden. Onthoud dat er enkele getallen kunnen zijn die binnen het BCD fixed point bereik vallen en daardoor meer dan 18 zinvolle cijfers (1.0×10^{19} of 1.0×10^{18}, $+1.0 \times 10^{-18}$ als voorbeeld) kunnen hebben. Verschillende getallen zijn niet geschikt voor gebruik in het PowerBASIC binary gecodeerd decimaal systeem, maar zijn overal aanwezig voor gebruik voor programmeurs die het rustig aan doen en die de soorten getallen verwerken. Het BCD fixed point gegevenstype van PowerBASIC is primair ontworpen voor gebruik in financiële berekeningen waarvan 18 cijfers met zich meebrengt.

BCD fixed point getallen zijn altijd afgerond zodat ze de aantal plaatsen kunnen innemen die gespecificeerd worden met de FIXDIGITS systeemvariabele (standaard is het 2 cijfers). Dit maakt het gemakkelijk voor afronding naar centen (gebruik 2 cijfers) of breuken van centen (gebruik 3 of 4 cijfers) in financiële berekeningen, als voorbeeld. De prijs dat u voor dit betaalt is snelheid. BCD fixed point is langzamer dan de andere floating point gegevenstypes.

BCD fixed point variabelen worden genoemd door een variabelenaam met een "@" (var@) of door gebruik van het DEFFIX statement. U kunt ook floating point BCD variabelen declareren door gebruik van het FIX sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS FIX
```

PRINT en STR$ werken als standaard tot 16 zinvolle cijfers (genoeg voor een double precisie waarde) – om alle 18 zinvolle cijfers van een BCD fixed point getal te printen of hem op te slaan in een string, is het wel nodig om gebruik te maken van de geavanceerde STR$ functie om 18 plaatsen te kunnen specificeren:

```
FIXDIGITS=18
...
PRINT STR$(var@,18)
```

Natuurlijk is de weergegeven 18 cijferige waarde van var@ prima, var@ moet toegekend worden zolang de systeemvariabele FIXDIGITS een waarde van 18 bevat. Bekijk eens de help of een boek van PowerBASIC voor meer informatie over het gebruik van FIXDIGITS.

Binary coded decimal (BCD) floating point (@@)
BCD floating point getallen hebben 10 byte binary gecodeerde decimale weergaven van floating point getallen, met een bereik van $\pm 9.99 \times 10^{-63}$ t/m $\pm 9.99 \times 10^{63}$ en kunnen weergegeven worden tot 18 zinvolle cijfers.

Dit gegevenstype is speciaal te gebruiken voor financiële berekeningen omdat het de afrondfouten vermijdt geassocieerd met single, double en extended precisie getallen. Verschillende getallen moeten een exact vermogen hebben van twee zodat het exact gepresenteerd kan worden. Zolang soorten getallen maar in die categorie vallen, getallen als 1.1 zullen dat niet. Het getal 1.100000023841858 heeft het dichtste vermogen tot twee (1.1). Omdat BCD floating point getallen de actuele decimale cijfers opslaan als een deel van een getal, zal elk decimaal getal tot 18 cijfers exact gepresenteerd worden. De prijs die U hiervoor betaalt is snelheid. BCD floating point is langzamer dan de andere floating point gegevenstypes.

Onthoud dat er enkele getallen kunnen zijn die binnen het BCD floating point bereik vallen en daardoor meer dan 18 zinvolle cijfers (1.0×10^{19} of 1.0×10^{18}, $+1.0 \times 10^{-18}$ als voorbeeld) kunnen hebben. Verschillende getallen zijn niet geschikt voor gebruik in het PowerBASIC binary gecodeerd decimaal systeem, maar zijn overal aanwezig voor gebruik voor programmeurs die het rustig aan doen en die de soorten getallen verwerken. Het BCD floating point gegevenstype van PowerBASIC is primair ontworpen voor gebruik in financiële berekeningen waarvan 18 cijfers met zich meebrengt.

BCD float variabelen worden genoemd door een variabelenaam met een dubbele "@" (var@@) of door gebruik van het DEFBCD statement. U kunt ook floating point BCD variabelen declareren door gebruik van het BCD sleutelwoord met het DIM statement, bijvoorbeeld:

```
DIM I AS BCD
```

PRINT en STR$ werken als standaard tot 16 zinvolle cijfers (genoeg voor een double precisie waarde) om alle 18 zinvolle cijfers van een BCD floating point getal te printen of hem op te slaan in een string, is het wel nodig om gebruik te maken van de geavanceerde STR$ functie om 18 plaatsen te kunnen specificeren:

```
PRINT STR$(var@@,18)
```

Een teken kan uit elke ASCII waarde bestaan van 0 t/m 255, inclusief de bekende letters van het alfabet, beide hoofd en klein. Wanneer de tekens samen zijn gevoegd, worden ze aangezien als zogeheten strings. Tekens en strings worden op verschillende manieren gebruikt tijdens programmeren. PowerBASIC biedt een zeer goede mogelijkheid genaamd flex strings.

In de volgende nieuwsbrief zal ik hierover verder gaan, maar ook over nog meer soorten gegevenstypes, zoals de constanten en de equates. Een equate is ook zeer bijzonder voor BASIC versies, want in de andere BASIC versies kunnen we geen equate (genoemd als constante) namen gebruiken en nieuwe expressies eraan toekennen. Als we in andere BASIC versies een constante gebruiken, mogen we de waarden niet meer wijzigen. Maar dat betekent niet dat we in PowerBASIC variabelen aan een equate naam mogen toekennen. Maar om wat u allemaal in PowerBASIC kunt doen maakt dat heus niet uit, dus de kracht van PowerBASIC gaat verder!

Marco Kurvers

De Basic gids.

Leuke Basic snufjes, zoals tips en trucs, af en toe kwamen er wel wat onderwerpen in de nieuwsbrieven. Het is weer een tijdje geleden en daarom vond ik dat het weer tijd werd voor een onderwerp over een Basic gids: een rondleiding over onderdelen die we in de Basic versies kunnen tegenkomen.

Gereedschap, materialen en onderdelen.

Maar voordat ik het over de onderdelen ga hebben, wil ik u eerst vertellen dat we ook gereedschap nodig hebben voor de materialen en onderdelen. Als we een product willen maken moeten we met de wortel beginnen.
Als we gaan programmeren, moeten we denken aan het volgende:

1. We gebruiken gereedschap om materialen, onderdelen en ook de producten te maken;
2. We gebruiken het materiaal om de onderdelen te maken;
3. Met de onderdelen bouwen we uiteindelijk de producten.

Maar het allerbelangrijkste voor het maken van een product is de gereedschap. De gereedschap is de wortel, maar we gebruiken het in alle drie punten. Ook al zouden we bijvoorbeeld punt 2 al klaargemaakt hebben. Een voorbeeld:

1. We hebben drie codeblokken gemaakt waar in elk codeblok meerdere malen PRINT statements in staan. We kunnen een Basic statement beschouwen als gereedschap.
2. We gaan elk codeblok bundelen als een onderdeel zodat het in één keer uitgevoerd kan worden. In Basic noemen we dat een subroutine.
3. We gebruiken gereedschap om elk onderdeel een naam te geven, bijvoorbeeld **SUB Testx()**.
4. We bouwen de onderdelen in een project dat dus in principe het product kan worden.

Een product of applicatie kan echter uit meerdere projecten bestaan. Zulke projecten kunnen onderdelen zijn die voor opslag zorgen zodat het hoofdproject sneller kan werken. Deze projecten worden ook wel secties genoemd (sections, zoals het in de Visual Basic .NET versies heet). Projecten worden altijd in puzzelstukjes geschreven die meestal uit besturingsonderdelen bestaan. Dit zijn de DLL bestanden. Een project op die manier schrijven hoeft eigenlijk niet. U mag ook in één keer een project schrijven, zonder DLL onderdelen, die alleen maar uit een EXE bestand zal bestaan. Toch doet men dat niet. Waarom eigenlijk niet? Waarom tijd verspillen door een project in stukken te schrijven en dan ook nog eens een setup wizard project te schrijven die het hoofdproject (de applicatie) dan weer voor de gebruiker in elkaar moet gaan zetten (installeren)?

Het gaat alleen niet om wat ik daarnet zei over snelheid tijdens de run time van de applicatie. Er komt veel meer bij kijken waarom programmeurs op die manier de projecten schrijven. Het is een zin die zeer bekend is en voor beginnende programmeurs heel belangrijk is: het wiel niet opnieuw uitvinden!

De setup wizard en het setup programma.

Wat is het verschil tussen de setup wizard en het setup programma? De wizard is een ingebouwd programma dat automatisch een setup programma in elkaar zet voor de programmeur. De meeste programmeertalen hebben al zo'n wizard. Er zijn nog veel meer soorten wizards, en ik heb U al een keer verteld dat niet alle wizards 100% te vertrouwen zijn. Zeker de converteer wizards niet.

Dankzij de setup wizard hoeft de programmeur geen code te schrijven voor het setupformulier en de controls. De wizard vraagt enkel om wat gegevens die tijdens de installatie van de applicatie nodig zijn. Dit zijn meestal de mapverwijzingen waar het hoofdproject en de onderdelen te vinden zijn en de bestanden die automatisch door de wizard gevonden zijn te kunnen kiezen. Ook al vindt de wizard deze bestanden, ze hoeven niet speciaal allemaal in de setup opgenomen te worden voor de installatie. De programmeur kiest zelf welke bestanden voor de installatie nodig zijn. De wizard weet dat zelf niet. Zou alles automatisch wel meegenomen worden dan zou er kans zijn dat tijdens de installatie sommige bestanden overschreven worden, of nog erger, dubbel in de computer aanwezig zullen zijn. Hierbij gaat het vooral om de DLL's die door de programmeurs worden geschreven of DLL's die al in de Windows systeem map bestaan, maar niet direct in het project aanwezig waren. Deze DLL's moeten ook in de lijst van de setup wizard geselecteerd worden voor de installatie, anders zal de applicatie niet correct werken.

We kunnen ook zelf een setup programma schrijven. U hebt dan geen wizard nodig. Alleen een formulier dat U als een setupformulier moet maken. Oh, en dan gewoon er voor zorgen dat alle onderdelen en gegevensbestanden in de juiste mappen geïnstalleerd worden? We zouden dus alleen maar in de code al deze bestanden via Dir.Path op te hoeven geven zodat de setup weet welke bestanden in welke mappen geplaatst moeten worden!?

Helaas, dat is niet het geval. De makkelijke theorie gaat aan onze neus voorbij. Degenen die al weten hoe ze een setup programma moeten schrijven hebben waarschijnlijk al door wat ik bedoel. Juist, de onderdelen die voor het wiel zorgen. De DLL bestanden, maar ook de OCR bestanden, en nog vele andere soorten

onderdelen. Voordat deze bestanden op andere computers door de setup geïnstalleerd kunnen worden, moeten ze eerst *geregistreerd* worden.
Bekijk de code van de setup wizard eens goed, daar kunt u alle registratiecode in vinden die nodig is om zulke bestanden te laten functioneren. Gelukkig hoeven we de registratiecode niet zelf te schrijven, we moeten het alleen gebruiken en op de juiste manier aanroepen. Dit zijn de Windows API functies voor de meeste Basic versies, maar in de nieuwe Basic versies worden de API functies niet meer gebruikt. Alles zit nu in het .NET Framework objectbibliotheek. We hoeven nu alleen maar de functies van de objecten te gebruiken en ze niet meer van te voren met DECLARE uit de API DLL's te halen. Gordon Rahman heeft daar meer over verteld in de vorige nieuwsbrief met informatie over hoe API in elkaar zit. Ik kan u verzekeren dat de nieuwe moderne Basic met het Framework objectlibrary heel anders in elkaar zit dan bijvoorbeeld Liberty Basic. Het werkt op dezelfde manier, maar het gereedschap van Liberty Basic is gemakkelijker te gebruiken. Evenzo PowerBASIC. Het zijn niet voor niets Basic versies die geschikt zijn voor gebruikers die moeite hebben met programmeren maar toch hetzelfde doel willen bereiken.

Marco Kurvers

API functies en handles (2).

Over dit onderwerp wou ik even vertellen dat dit het vervolg is van het onderwerp in de vorige nieuwsbrief. Ik was vergeten (1) achter de titel te zetten, vandaar dat er nu (2) staat.

Liberty Basic is een perfecte tool om de API's (de functies in de DLL's) van Windows toe te passen en te bestuderen.
Oeps! Bestuderen is teveel gezegd. De meeste DLL's zijn in C of in C++ geschreven en ook de toelichting en voorbeelden op de MSDN website staan in C en soms in VB aangegeven. Maar we kunnen ze allemaal toepassen als we de namen van de functies maar kennen en weten welke parameter en parametertypes we moeten ingeven.

Er zijn werkelijk tientallen sites waar alle (Microsoft Windows) DLL functies breed en uitvoerig worden toegelicht. Er bestaan zoals eerder gezegd ook programma's in Liberty Basic waarmee je alle functie-, parameter- en variabelennamen kunt uitlezen uit de header van elke DLL. Eentje die ik zelf gebruik is het programma uit de Libby demo schijf, maar zo'n programma schrijf je in drie minuten. De viewers lezen gewoon welke functies in de DLL zitten. De namen van de functies staan aan het eind van de DLL na de naam van de DLL. De kunst is om te weten welke parameters je bij welke functie moet ingeven en hoeveel bytes (nauwkeurigheid) je per parameter dient te reserveren.

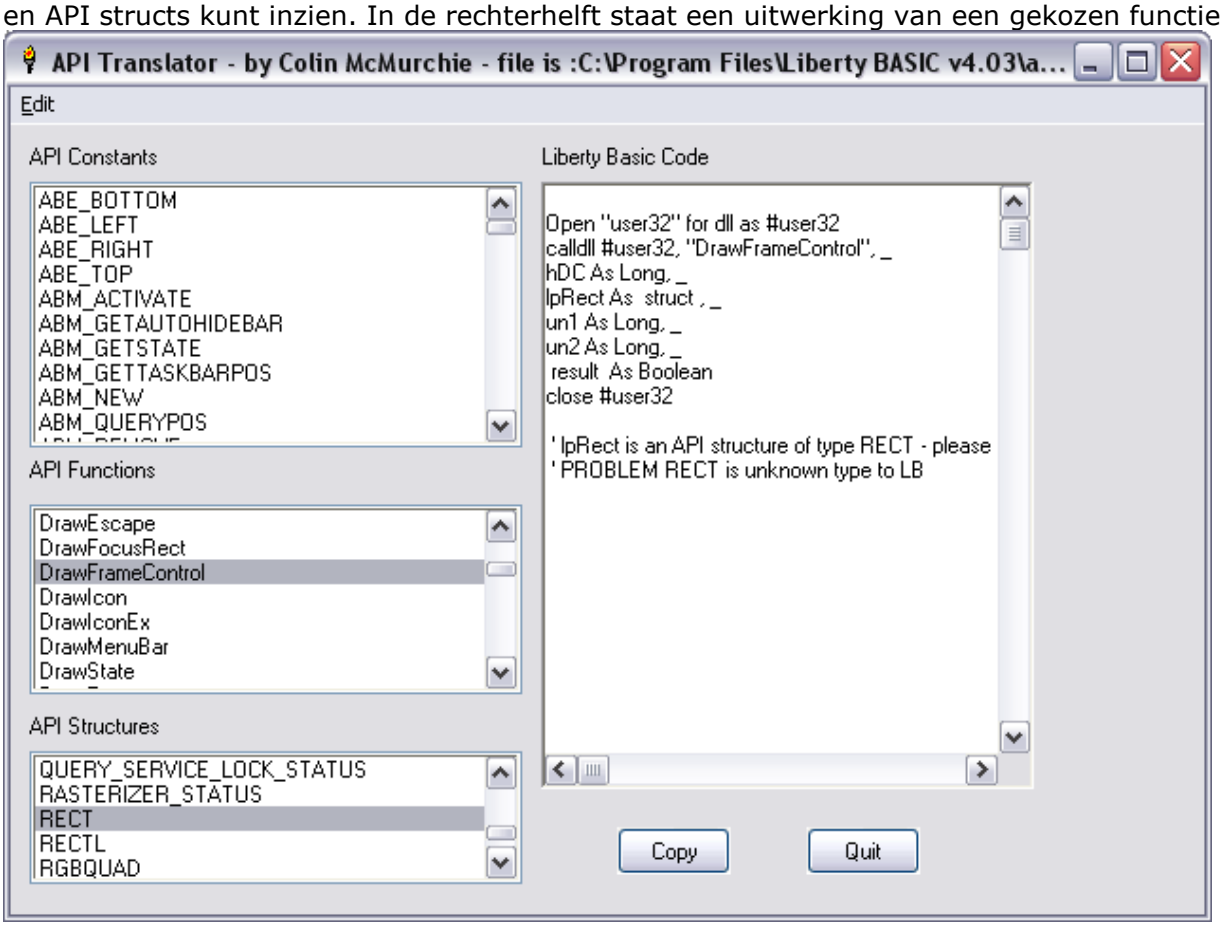
Microsoft op zijn best!
Microsoft heeft al zijn DLL's heel goed gedocumenteerd. Naast voorbeelden voor het gebruik heeft Microsoft ook een groot bestand gemaakt waarin alle benodigde parameters en daarbij horende types (nauwkeurigheid) staan aangegeven. Voorwaar hulde aan Microsoft. Alles staat in een document **win32api.txt**.

Zelf een DLL schrijven?
Ga C of C++ bestuderen. Microsoft geeft een gratis cursus! Hier volgt een inleiding tot het maken van een DLL.
http://web.archive.org/web/20050305154547/syberden.net/libertybelle/dll.htm
Een DLL schrijven is volgens mij niet eenvoudig. Toch mag ik met enige trots erop wijzen dat er op het forum van Liberty Basic Nederland (www.libertybasic.nl) een door een forumlid zelfgemaakte DLL te downloaden is. Met deze DLL en de bijgeleverde instructies schijn je VOIP (Voice over IP) met Liberty Basic te kunnen doen.
http://www.libertybasic.nl/viewtopic.php?f=7&t=364

Uiteraard hebben de Liberty Basic gebruikers de hele win32api door Liberty Basic laten uitlezen. Nu kun je alle namen van de Windows constanten op een rijtje zien. Je kunt alle functienamen van Windows zien, maar vooral... je kunt voorbeelden van het gebruik (dll, functie, parameters) zien. Er was in 2004 een uitdaging/wedstrijd waarbij het de bedoeling was om een programma te maken waarmee win32api.txt ingelezen en uitgerafeld werd.

Het is mogelijk dat je nog geen goed beeld hebt van hoe een DLL opgebouwd is. Bij het gebruik van C/C++ worden de variabelen of constanten apart in een header file geplaatst. Nadat het hele C/C++ project gecompileerd is, staan volgens mij de variabelen in geheimschrift in de DLL file. Zonder een goede eerlijke beschrijving van de parameters kun je een DLL dus niet gebruiken.

Hier volgt een site die alle Windows constanten in een tabel geplaatst heeft.
http://doc.pcsoft.fr/en-US/?6510001
In Figuur 1 zie je een screenshot van het Liberty programma waarmee je alle Windows constanten, functies en API structs kunt inzien. In de rechterhelft staat een uitwerking van een gekozen functie in Liberty Basic.



Figuur 1. **Screenshot van API translator**

In Figuur 1 heb ik willekeurig gekozen voor de API functie DrawFrameControl uit de DLL user32.dll. Uit de gegevens van het scherm blijkt dat lpRect als een struct opgegeven moet worden. Het argument lpRect is van het structuretype RECT. De parameter zal dus van het type RECT meegegeven moeten worden tijdens de aanroep van de functie.

Daarom heb ik ook de RECT struct opgezocht, zie Figuur 2.

Microsoft constanten
Blijft nog één punt over: de Microsoft constanten. Je kunt elke constante direct in de struct of in de API invullen, bijvoorbeeld:

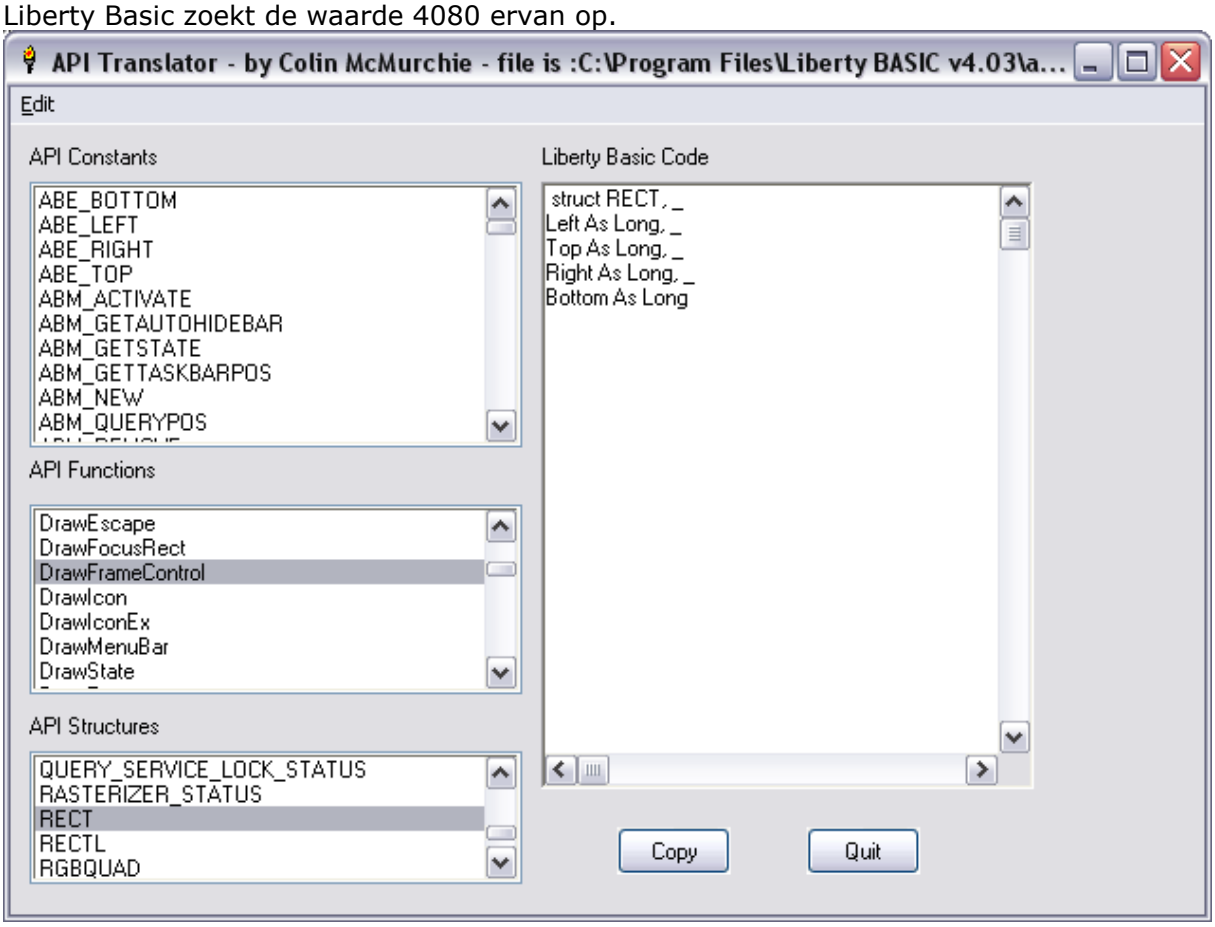
```
STRUCT RECT,_
    123 as long,_
    78 as long,_
    289 as long,_
    785 as long
```

Maar natuurlijk doe je dat liever niet, tenzij je precies weet waar ze voor gebruikt zullen worden. Dus je doet liever:

```
    Left = 123
    Top = 79
    Right = 405
    Bottom = 300
```

en daarna plaats je de struct. Sommige STRUCTS en API's gebruiken Microsoft constanten.
In Figuur 1 en 2 zijn enkele constanten zichtbaar. Deze constanten worden in Liberty Basic herkend als je er een liggend streepje (underscore teken) voor plaatst. Ik heb uitgezocht dat de Windows constante APPCMD_MASK = 4080.

Dat zocht ik eerder uit via de link naar de Franse site waar ze zeggen dat alle constanten in een tabel zijn opgenomen?! Ik mis er een heleboel. Dus in Liberty Basic mag ik gewoon schrijven _APPCMD_MASK en Liberty Basic zoekt de waarde 4080 ervan op.



Figuur 2. **De API constanten**

Tenslotte nog het teruglezen van waarden uit een struct of API.
Stel dat de API een string als antwoord geeft, dan hebben we een probleem. De API geeft alleen een getal weer (de pointer naar de string). Gelukkig kent Liberty Basic de functie WINSTRING() waarmee uit de pointer de bijbehorende string opgehaald wordt.

Ik schrijf geen lange gecompliceerde artikelen. Dus er volgt nog een deel.

Tot de volgende keer.

Gordon Rahman

Event driven programmeren.

Ik kan geen goede omschrijving voor dit begrip vinden, daarom geeft de letterlijke vertaling misschien de beste indruk. *Gebeurtenis (op signalen) gestuurd programmeren*, maar ook: *Op muis of toetsenbordactie gebaseerd programmeren* valt misschien wat te zeggen.

Windows is een besturingssysteem waarbij de meeste programma's en onderdelen een EVENT DRIVEN karakter hebben. Het programma staat het merendeels van de tijd (IDLE) te wachten op een muisbeweging, -klik of toetsenbordaanslag. Dit proces van muis(poort) en toetsenbord(poort) scannen gaat continue door, ook terwijl op de achtergrond vele processen bezig zijn.

Liberty Basic is een Windows programma waarmee Windows applicaties (vensters met alle toeters en bellen) geschreven kunnen worden in een op Basic gelijkende stijl. Ook de meeste Liberty Basic programma's hebben een EVENT DRIVEN karakter. Liberty Basic heeft een eigen VIRTUAL MACHINE die gebruik maakt van de meeste DRIVERS, onderdelen (DLL en API) en de EVENTS keten van Windows.

In dit stuk zal ik het hebben over EVENT DRIVEN programmeren in Liberty Basic, waarbij de volgende begrippen zeker zullen vallen:

TIMER	soort software INTERRUPTS
WAIT	programma staat in IDLE stand te wachten op een bruikbare EVENT
SCAN	test het geactiveerde venster op muis- en of toetsenbordgebruik

Al ten tijde van MSDOS was er sprake van programmeren op software *interruptbasis*, hetgeen EVENT DRIVEN sterk benadert. Soms werd daarbij geen duidelijke scheiding gemaakt met de hardware interrupt. In feite was KEY OFF/ON bij MSX-BASIC en GW-BASIC reeds een begin van software interrupt gestuurd programmeren. Bij de TRS-80 werd BREAK OFF/ON een soort software interrupt naast de hardware interrupt waaraan de BREAK knop achteraan het toestel verbonden was.

We zullen zien dat Liberty Basic voor interrupts gebruik maakt van een timer van Windows. De gebruikers van Liberty Basic hebben intussen uitgevonden hoe meerdere Windows Timers te gebruiken. De documentatie van timers en dergelijke staat allemaal op de MSDN (Microsoft Development Network) website. We zullen zien dat WAIT en SCAN commando's van Liberty Basic zijn.

Wat is een EVENT?

Een EVENT is niet alleen een actie van de muisknop of het toetsenbord, maar ook signalen over en weer, van gaande processen tijdens Windows. Programma's (ook DLL's) kunnen een signaal geven als ze een bepaalde bewerking hebben uitgevoerd. Alle EVENTS worden door Windows opgevangen. Het is mij niet precies bekend hoe. Misschien veroorzaken EVENTS een interrupt of misschien is Windows op gezette tijden de muisknoppen, het toetsenbord en overige processen aan het scannen (of pollen) op bewegingen of veranderingen. Hoe het ook zij, Windows maakt van elk EVENT een bericht dat aan alle gaande processen wordt doorgegeven. Nee, laat me dit beter zeggen, de processen kunnen open staan voor het ontvangen van EVENT berichten.

Wat gebeurt er met een EVENT?

De meeste EVENTS worden intern door de verschillende processen, die voortdurend door Windows actief zijn, verwerkt. De processen geven daar weer signalen (nieuwe EVENTS) van af. Als een EVENT bericht door de VIRTUAL MACHINE van Liberty gebruikt kan worden dan zoekt de VM van Liberty Basic zelf uit of het EVENT bericht betrekking heeft op een actief venster uit het in Liberty Basic gaande programma. Daarna kijkt Liberty Basic of de programmeur een HANDLER voor dit EVENT bericht heeft geschreven. Zo ja, dan wordt de HANDLER uitgevoerd. Liberty Basic stelt niet in alle EVENT berichten belang. Als een venster geminimaliseerd of gemaximaliseerd (vorig formaat) wordt dan laat Liberty Basic de actie aan de HANDLERS van Windows over. Als bij Liberty Basic in een actief venster op AFSLUITEN (kruisje rechtsboven) of ALT+F4 wordt geklikt dan reageert de HANDLER trapclose van het Liberty Basic programma eerst. Indien de programmeur daar geen code voor geschreven heeft dan sluit Windows het venster zelf, met het gevolg dat het Liberty Basic programma verkeerd afgesloten kan worden.

De HANDLER.

Bij Liberty Basic heeft elk venster een eigen unieke HANDLER. Dat is de naam die begint met het # teken. Uit de HANDLER namen kun je concluderen welke CONTROLS bij welk venster horen, omdat het eerste deel van de HANDLER naam overeenkomt. Als Liberty Basic (de VIRTUAL MACHINE) een bruikbare EVENT bericht heeft ontdekt, wordt dit direct aan de betrokken HANDLER doorgegeven. De HANDLER voert de voorgeschreven actie uit. Dit noemen we EVENT DRIVEN PROGRAMMING.

De werking van CONTROLS werd al sinds jaar en dag geïmiteerd. Zelfs ten tijde van DOS zonder muis werden reeds programma's met een menustructuur ingericht, waardoor die programma's als EVENT DRIVEN PROGRAMMING aanvoelden. Het EVENT viel daarbij samen met het invoeren van data of kiezen uit een keuzemenu: MAAK UW KEUZE EN DRUK OP ENTER. Bij het echte EVENT DRIVEN PROGRAMMING is het EVENT willekeurig en kan altijd optreden zolang het venster of Windows openstaat. We zullen zien dat de afhandeling van INPUT of EVENTS toch enige overeenkomst vertoont.

Ik wil EVENT DRIVEN PROGRAMMING in dit stuk niet alleen vergelijken met het zogenaamde lineair of sequentieel programmeren, maar ik wil het ook betrekken in de vergelijking met **o**bject ge**o**riënteerd **p**rogrammeren (OOP).

- Bij lineaire programmering gaat de programmeur ervan uit dat hij een menu moet leveren voor de communicatie over en weer met de gebruiker. De programmeur stopt het programma bij het interventiepunt totdat de benodigde invoer verkregen is en vervolgt dan.
- Bij EVENT DRIVEN PROGRAMMERING staat het programma toe dat de gebruiker op elk eigen gekozen moment aan het toetsenbord zit, de muis beweegt, een muisknop indrukt enzovoort. Het toetsenbord wordt voortdurend door

Windows gescand. Als de muis bewogen of ingedrukt wordt dan is dat een EVENT. Dit EVENT wordt aan het Liberty Basic programma gemeld. Het programma wordt gewekt waarna er actie volgt.

- Bij OOP worden constant procedures doorlopen, waarbij het zelf zo ver gaat dat ook interventie van de gebruiker als een procedure geldt. Het maken van een GUI is daarom moeilijk.

Bij EVENT DRIVEN PROGRAMMING gaat het ook om de afhandeling en niet zozeer om het willekeurig moment en het willekeurig soort EVENT. De programmeur staat toe dat de gebruiker het moment voor een EVENT kiest en garandeert dat daar de juiste reactie op zal volgen. Elke muisbeweging of toetsenbordaanslag is voor Windows een EVENT. Liberty Basic maakt daarvan gebruik door tijdens het verwerken, van sommige EVENTS door Windows, de afhandeling over te nemen. Een EVENT mag en zal op een willekeurig moment komen en wordt altijd afgehandeld door minstens een Windows HANDLER en of doorgegeven aan de Liberty Basic HANDLERS.

Iedereen weet wat een INTERRUPT in de computerwereld is. Een voorbeeld van een alledaagse INTERRUPT zou zijn als je slaapt en 's morgens door het afgaan van de wekker wakker schrikt. Het afgaan van de wekker is een soort interrumperen waarop je reageert door te stoppen met slapen en de wekker af te zetten en direct verder te gaan slapen. De afhandeling van een INTERRUPT gebeurt met een HANDLER. Eigenlijk gaat aan een INTERRUPT een EVENT vooraf, maar in het meeste spraakgebruik worden deze begrippen door elkaar gehaald. In dit geval bent u blijkbaar geprogrammeerd om de wekker af te zetten als die tijdens uw slaap afgaat. Bij Liberty Basic kan men gebruik maken van een Windows TIMER om met regelmaat een INTERRUPT te genereren.

De TIMER kan in de volgende vormen worden geprogrammeerd:

TIMER aantal milliseconden, [Label]
TIMER aantal milliseconden, SUBROUTINE_NAAM

De eerste vorm lijkt op een geregelde GOTO en dient met een WAIT of iets dergelijks afgesloten te worden. De tweede vorm lijkt op een geregelde GOSUB en dient met een END SUB afgesloten te worden.

De lineaire vorm.

Het concept van EVENT DRIVEN PROGRAMMING is vooral vreemd voor hen die bekend zijn met lineaire talen als QuickBASIC of Fortran. Bij dergelijke programmeertalen begint het programma op regel één en het programma vervolgd dan van boven naar beneden door elke regel af te werken. Het is mogelijk dat de gebruiker met één of andere interventie een aftakking naar een ander deel van de listing veroorzaakt, maar daar aangekomen vervolgt het programma zijn weg weer van boven naar beneden en regel voor regel. Elke volgende interventieplek is voorgeprogrammeerd.

EVENTS (gebeurtenissen) verschillen van Basic gebruikerinterventies in die zin dat ze qua tijd niet te voorspellen zijn. Dit is een belangrijk verschil dat ik duidelijk hoop te maken aan de hand van twee programmavoorbeelden. Het volgende gedeelte is een bewerking, met toestemming van auteur Brad Moore, uit de Liberty Basic Newsletter nummers 102 en 104.

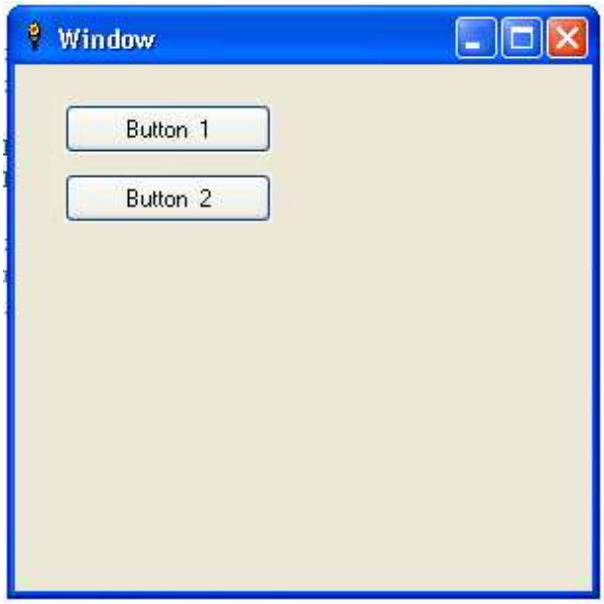
```
'Voorbeeld van sequentiële (lineaire) programmering
[menu]
 cls
 print
 print " *******  Omzetten van Graden en Fahrenheit  *******"
 print
 print "   1) Fahrenheit naar Celsius"
 print "   2) Celsius naar Fahrenheit"
 print "   3) Help"
 print "   4) Stoppen"
 print
 print "   Maak een keuze -> ";
 input a$
 a = val(a$)
 if a = 1 goto [FC2]
 if a = 2 goto [C2F]
 if a = 3 goto [help]
 if a = 4 print
   print "Bedankt voor gebruik van de tool"
   end
```

```
 end if
 goto [menu]

[F2C]
 print
 print "Fahrenheit naar Celsius "
 print
 print "Enter graden Fahrenheit -> ";
 input a$
 a = val(a$)
 if a$ <> "0" and a = 0 then goto [menu]
 print
 print a$;" graden Fahrenheit is gelijk aan ";
 print using("###.#",str$((a - 32) * 5/9));
 print " graden Celsius."
 print
 print "<ENTER> om door te gaan...";
 input a$
 goto [menu]

[C2F]
 print
 print "Celsius naar Fahrenheit"
 print
 print "Enter graden Celsius -> ";
 input a$
 a = val(a$)
 if a$ <> "0" and a = 0 then goto [menu]
 print
 print a$;" graden Celsius is gelijk aan ";
 print using("###.#",str$(a * 9/5 + 32));
 print " graden Fahrenheit."
 print
 print "<ENTER>" om door te gaan...";
 input a$
 goto [menu]

[help]
 cls
 print " U kunt met dit programma de "
 print " temperatuur van graden Celsius "
 print " omzetten naar graden Fahrenheit"
 print " en omgekeerd"
 goto [menu]
```

In het bovenstaand programma wordt een menu getoond waarin de gebruiker een functie kan selecteren. De loop van het programma splitst zich op basis van de keuze die de gebruiker maakt. Eén van de kenmerken van dergelijke programma's is dus dat het systeem, in dit geval bijvoorbeeld Liberty Basic, geïnstrueerd is om te wachten op één enkele bepaalde soort input op elk punt waar interventie door de gebruiker wordt verwacht. Het bovenstaand programma heeft vijf input statements. Op elk van deze plekken wordt een bepaalde soort input verwacht. Data die niet aan de eisen van de verwachte input voldoet geeft een foutconditie. Hierdoor wordt dus een grote inflexibiliteit in het programma geïntroduceerd. De gebruiker kan alleen op de voorgeschreven paden het programma doorlopen. Zelfs om het programma te beëindigen moet ik het voorgeschreven cijfer invoeren. Als het programma eenmaal vraagt om een waarde van temperatuur dan heb ik geen andere keus dan een getal in te geven. Het moment van invoeren (ENTER drukken na invoer) kun je als een soort EVENT beschouwen en het programma daarop inrichten.

Een detail kijkje in een EVENT DRIVEN programma.

EVENT DRIVEN programma's verschillen aanmerkelijk op dit vlak. Elke mogelijke uitkomst (EVENT) kan op elk willekeurig moment gebeuren. Laten we deze mogelijke startseinen eens nader bekijken. Een basisvoorbeeld is een venster met enkele gebruikersknoppen en de drie knoppen rechtsboven waarmee het venster afgesloten, verkleint (minimaliseren naar de werkbalk) en vergroot (vorig formaat) kan worden. Dit venster staat op uw Windows bureaublad of iets dergelijks.



Figuur 1.

Dit is een standaard voorbeeld van een venster met twee knoppen.

Hier volgt daarvoor de code in Liberty Basic:

```
    'Setup
    NoMainWin
    WindowWidth = 300
    WindowHeight = 300
    UpperLeftX = Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight-WindowHeight)/2)

    'Voeg buttons toe
    Button #main.bt1, "Button  1",[bt1],UL, 25, 20, 105, 25
    Button #main.bt2, "Button  2",[bt2],UL, 25, 55, 105, 25

    'open het venster
    Open "Window" For Window As #main

    'vang het venster afsluitkruisje af (hé! Dit is ook een EVENT!)
    Print #main, "trapclose [quit]"

[loop]
    Wait

[quit]
    Close #main
    End

[bt1]
    'Hier wordt eigenlijk niets bijzonders gedaan
    GoTo [loop]

[bt2]
    'Hier wordt eigenlijk niets bijzonders gedaan
    GoTo [loop]
```

U ziet dat het hier gaat om een redelijk eenvoudig programma. Open een venster met twee knoppen (buttons). We willen er voor zorgen dat beide knoppen apart naar een eigen gedeelte van het programma verwijzen. Als u op één van de knoppen klikt springt de uitvoering naar het aangewezen deel van het programma waar de afhandeling voor deze knop staat. Dus als op Button1 gedrukt wordt springt de uitvoering naar label [bt1] en als op Button2 gedrukt wordt naar [bt2]. In ons voorbeeld gebeurt er op die plekken niets, omdat het programma verder gaat naar [loop] en daar staat WAIT. De EVENTS voor minimaliseren en vorig formaat voor het venster worden door Windows afgehandeld. Windows handelt ook ALT+F4 af. Maar het kruisje in de rechterbovenhoek van het venster moet of mag door het programma afgehandeld worden. Deze aftakking heeft in ons voorbeeld de label [quit] meegekregen.

Het programma is dus op alle mogelijke gebeurtenissen (alle eventualiteiten) voorbereid. In geval van de knoppen waren dat de *EVENT HANDLERS*, stukjes code die op de labels volgen. Deze HANDLERS komen dus pas in actie als ze door een EVENT aangeroepen worden. De code (het programma) wordt niet lineair

uitgevoerd, maar er wordt gewoon telkens gewacht op een EVENT alvorens er iets uitgevoerd wordt. Een EVENT DRIVEN programma zorgt ervoor dat de CPU vaak in inactieve (IDLE) staat is.

Omdat dit misschien de eerste keer is dat u een Liberty Basic listing ziet, zal ik even langer bij de listing stilstaan.
De set-up spreekt voor zichzelf. De variabelen WindowWidth en WindowHeight bepalen de breedte en de hoogte van het eerstvolgende venster dat u wilt openen. We hadden ook defaultwaarden kunnen gebruiken door geen vensterbreedte of –hoogte op te geven. Zo geven UpperLeftX en UpperLeftY de X en Y coördinaten aan waar de linkerbovenhoek van het venster komt te staan.
Met de formules Int((DisplayWidth-WindowWidth)/2) en Int((DisplayHeight-WindowHeight)/2) zorgen we ervoor dat het venster precies in het midden van het beeldscherm komt te staan.

Daarna maken we twee knoppen voor het venster dat we zullen openen.
```
Button #main.bt1, "Button  1",[bt1],UL, 25, 20, 105, 25
Button #main.bt2, "Button  2",[bt2],UL, 25, 55, 105, 25
```

De HANDLER van de eerste knop is **#main.bt1**.
Het eerste deel van de HANDLER, het gedeelte **main**, is gelijk aan de HANDLER van ons te openen venster. Hierdoor weet Liberty Basic dat deze CONTROL, zo heet een knop, radioknop, tekstbox enzovoort, in het venster bij de HANDLER **main** hoort. De HANDLER van de knop heeft nog een unieke extensie **bt1**, waardoor Liberty Basic hem kan scannen.

De tekst op de knop wordt: "Button 1".
Als de knop bediend wordt moet de uitvoering van het programma ogenblikkelijk naar de label [bt1] verder gaan.

De relatieve plaats van de knop in het venster wordt bepaald door UL (Upper Left = Links Boven) 25 pixels naar rechts (X-as richting) en 20 pixels naar beneden (Y-as richting).

De laatste twee getallen geven de breedte en de hoogte van de knop aan. Als we die niet hadden opgegeven dan had Liberty Basic ze net zo groot gemaakt als de tekst op de button.

Zodra we een EVENT hebben veroorzaakt, zorgt Liberty Basic ervoor dat het programma zijn uitvoering vervolgt, te beginnen op de plek (label) waar de betrokken HANDLER naar verwijst. In ons geval zal kort daarop het statement WAIT worden bereikt. Dit commando bevriest verdere uitvoering van het programma – het programma wordt voor de CPU inactief – en gaat wachten op het volgende startsein van een EVENT. Overigens zal Liberty Basic vastlopen als je een CONTROL hebt met een HANDLER die naar een label verwijst die niet bestaat. Windows heeft op de achtergrond een heleboel processen gaande om EVENTS te detecteren en te behandelen. Op dit aanknopingspunt met OOP kom ik nog terug.

Waarom is een GUI meestal EVENT DRIVEN?

Een GUI (Graphical User Interface, spreek uit goewie) is een venster met enkele objecten als knoppen (CONTROLS geheten), waarmee de communicatie tussen het programma en de gebruiker over en weer geregeld wordt. Met Liberty Basic kunt u EVENTS genereren als u bovenop de vele verschillende soorten CONTROLS met de muis klikt. De volgende CONTROLS kunt u in Liberty Basic gebruiken om EVENTS te maken die u weer met de bijbehorende HANDLES, met een specifieke label per CONTROL), kunt laten uitvoeren.

- Button (knop met labelverwijzing)
- Checkbox (met set en reset verwijzing)
- Combobox (met input uit !CONTENS?)
- Graphicbox (met WHEN LEFTMOUSEBUTTON enzovoort)
- Listbox (met input uit !CONTENS?)
- Menu (pulldown menu's)
- Popupmenu (popup)
- Radiobutton (met set en reset)

De overige CONTROLS (afsluitkruisje, minimaliseren enzovoort) worden door Windows in de gaten gehouden en voortdurend gescand. EVENTS worden indien bruikbaar opgevangen en ontgrendelen stukken code totdat een van de volgende statements gedetecteerd worden:

1. Scan
2. Input
3. Wait

Bij een sequentieel (lineair) gestructureerd programma wordt van de gebruiker op een bepaalde plek in het programma informatie verwacht. De verkregen informatie moet echter eerst ontleedt worden voordat het programma verder gaat. Als de gebruiker een INPUT A$ moet voeden, moet eerst bekeken worden of het antwoord voldoet aan de verwachtingen van de programmeur. De gebruiker kan van alles invoeren. Stel dat de programmeur verwacht dat de gebruiker het woordje STOP intikt, dan kan die gebruiker echter best EINDE of QUIT invoeren. Omdat al deze mogelijkheden opgevangen moeten worden zal de programmeur de input mogelijkheden van de gebruiker aan de andere kant moeten beperken.

In de EVENT DRIVEN wereld is de programmeur niet genoodzaakt de gebruikersinput eerst te ontleden omdat het programma een invoer omgeving heeft gecreëerd die ervoor zorgt dat de invoer zal zijn wat de programmeur wil. Knoppen voor actie, invoervelden voor data, filters voor menu's en vensterfuncties. Allemaal hebben ze opgegeven parameters en allemaal worden ze achter de schermen afgehandeld met een enkele MACRO EVENT waarvan de HANDLER ingebed is in de WAIT, INPUT en SCAN commando's. EVENTS starten gewoon vooraf geprogrammeerde reacties van de HANDLERS. Dat hebben we reeds eerder gezien. Het gebeurt allemaal als gevolg van de inrichting door de programmeur. De GUI specificaties spelen daarbij een grote rol. Zo zagen we in het Button statement dat er bepaald was waar het programma verder vervolgen moest en hoe.

Eerst vullen we ons voorbeeldprogramma wat aan. We laten de knoppen wat doen. Als u op knop "knop1" drukt (de muisklik is het EVENT) gaat de HANDLER (#main.button1) optellen tot het oneindige. Variabele a wordt geïncrimineerd, dat wil zeggen: er wordt steeds één erbij opgeteld.
Als u op knop "knop2" drukt gaat de andere HANDLER (#main.button2) optellen tot het oneindige. Variabele b wordt geïncrimineerd, dat wil zeggen: er wordt steeds één erbij opgeteld.
Tussendoor wordt elke seconde de tijd geprint.

```
'Event driven program to test TIMER as interrupt
'Form created with the help of FreeForm 3 v06-03-03
'Generated on Feb 13, 2005 at 20:20:40

[setup.main.Window]

    '-----Begin code for #main

    ' nomainwin
    WindowWidth = 400
    WindowHeight = 400
    UpperLeftX=Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY=Int((DisplayHeight-WindowHeight)/2)

    '-----Begin GUI objects code

    button #main.button1,"knop1",[button1Click], UL,  88, 121,  47,  25
    button #main.button2,"knop2",[button2Click], UL, 257, 122,  47,  25

    '-----End GUI objects code
    open "Event driven test 1" for window as #main
    print #main, "font ms_sans_serif 10"
    print #main, "trapclose [quit.main]"

timer 1000, printIets

[main.inputLoop]     'wait here for input event
    wait

[button1Click]       'Perform action for the button named 'button1'
    scan
    a = a + 1
    print "a ";a
    goto [button1Click]

'wait

[button2Click]       'Perform action for the button named 'button2'
```

```
    scan
    b = b + 1
    print "b ";b
    goto [button2Click]

'wait

[quit.main]       'End the program
    close #main
    END

sub printIets
    print "hallo "; time$()
end sub
```

Met het bovenstaand programma wil ik laten zien dat een EVENT iets anders als een INTERRUPT is. Het TIMER statement in ons programma luidt:

Timer 1000, printIets

Dit dirigeert het programma elke seconde naar de SUBROUTINE printIets.
De TIMER is een INTERRUPT die op voorop gezette tijden plaatsvindt. In ons voorbeeld zal deze om de 1000 milliseconden (1 seconde) gebeuren. Het indrukken van één van de knoppen zorgt ervoor dat het programma in een oneindige kringloop geraakt. Elke seconde wordt deze loop stilgezet en wordt het woordje "hallo " met de tijd erachter geschreven. De kringloop wordt hervat of vervolgt, zo u wilt. Een EVENT lijkt op een INTERRUPT maar is dat niet.

Het voornaamste van een activiteit in een EVENT DRIVEN programma is toch het wachten op een EVENT. Als het EVENT zich aan Windows voordoet lijkt het op een INTERRUPT, vooral omdat het zich niet aankondigt. Voor Liberty Basic kan dit een oud EVENT zijn. EVENTS worden namelijk door Windows verzameld. De VIRTUAL MACHINE van Liberty Basic kijkt geregeld in de verzameling EVENTS van Windows. Er zijn voortdurend vele verschillende (low-level en high-level) processen gaande tijdens Windows. Deze processen creëren en consumeren voortdurend boodschappen en dus ook EVENT berichten. Vele berichten geraken niet ver en blijven op de laagste niveaus steken. Andere berichten worden weer steeds herhaald enzovoort. Dieper liggende processen gebruiken deze berichten en sturen sommige verder naar hogerop gelegen processen. De meeste mensen realiseren zich niet dat Windows zo in elkaar zit. Als u in het takenbeheer scherm naar de gaande processen kijkt moet u zich realiseren dat u alleen naar de top-level processen kijkt. De meeste van deze top-level processen consumeren meer berichten dan ze produceren. Windows verschaft een manier om daadwerkelijk bij deze berichten te komen en ze te onderscheppen voor eigen gebruik. Uiteraard gebeurt dit onderscheppen met behulp van verschillende API calls. Dit proces heet SUBCLASSING, maar het werkt niet voor Liberty Basic. Liberty Basic heeft namelijk een eigen VIRTUAL MACHINE die reeds zelf bezig is allerlei berichten te consumeren waardoor elke poging om te SUBCLASSEN strandt in een chaos voor beide partijen.

Wat maakt het programmeren in een EVENTS omgeving dan zo anders?

Wel, stel dat u een programma heeft met 5 vensters met 4 knoppen per venster. Bij sequentieel programmeren staat steeds één menu op het scherm. Bij Windows en Liberty Basic kan de gebruiker echter alle vensters op het scherm houden. Uiteraard is er maar één venster tegelijk actief. De programmeur moet er wel rekening mee houden dat de gebruiker willekeurig een venster kan activeren en een knop kan bedienen. Waar de beperking van de invoermogelijkheden in het voordeel van Windows programma's was, geeft dat nu het nadeel dat meerdere vensters tegelijk op het scherm kunnen staan en eenvoudig geactiveerd kunnen worden. Bij de eenvoudige GUI ontwerpen krioelt het daarom van vensters met knoppen als <Vorige><Terug><Volgende><Annuleren> enzovoort, waarbij u een volgend of vorig venster kunt oproepen en maar één venster tegelijk op het scherm kunt bedienen. U kunt zich voorstellen dat een game programmeur helemaal in de problemen kan raken als een speler de ESCAPE toets gebruikt of de F1 toets (voor Help) en daarna besluit de game te continueren. De programmeur moet zelf de stand onthouden en het beeld bevriezen. Stel je voor dat een schaker even op F1 drukt en het Liberty programma vergeet de laatste stand of de gespeelde tijd. Het is daarom goed een vlag bij de HANDLER labels te plaatsen zodat u tenminste weet met welk proces Liberty Basic bezig was voordat het volgende EVENT startte. De programmateller wordt niet op de stack gezet. Daarmee bedoel ik dat Liberty Basic niet onthoudt waar het gebleven was. Vooropgesteld dat u dat wel wilt weten natuurlijk.

Gelukkig zijn er vele mogelijke oplossingen met Liberty Basic. Er zijn voor de verschillende mogelijkheden andere soorten vensters. Zo zijn er vensters waar het focus altijd op blijft staan. Dergelijke vensters moe-

ten dus eerst afgesloten worden voordat het programma vervolgd kan worden. Zo zijn er vensters die als een POP-UP verschijnen (zonder titelregel). Er zijn vensters zonder de mogelijkheid ze te verkleinen. Er zijn vensters met of zonder schuifbalk, enzovoort.

SCAN is een Liberty Basic commando dat ervoor zorgt dat Liberty Basic even stopt waarmee het bezig was en de Muis en Toetsenbord (EVENTS) berichten van Windows uitleest. Dit is vooral nuttig in die situaties waarin Liberty Basic constant door een proces aan het uitvoeren is, maar waarbij we toch zo nu en dan ook naar het toetsenbord of bepaalde knoppen moeten kijken. Scan lijkt daarom op een INPUT commando waarbij echter niet gestopt en gewacht wordt.

WAIT is ook een Liberty Basic commando dat er simpel voor zorgt dat het programma stopt en wacht op input van de gebruiker. Als de gebruiker een EVENT veroorzaakt met een venster of een control van een Liberty Basic programma dan vervolgt het programma vanaf de betrokken HANDLER(CONTROL). Wait is dus een gewone INPUT, maar ik gebruik het liever omdat deze omschrijving het wezen van dit commando beter benadert.

EVENT DRIVEN PROGRAMMING lijkt vreemder dan het is. Een eerste meeste eenvoudige manier om uw traditionele oude programma's te vernieuwen of over te zetten in Liberty Basic is door gebruik te maken van het PROMPT commando om gegevens van de gebruiker te ontvangen (input) en NOTICE om het resultaat van uw programma (print) te tonen. Zodra uw programma meer gaat vergen van de gebruiker, als u de communicatie niet meer af kunt handelen met CONFIRM, zult u over moeten stappen op vensters en op de daarbij horende knoppen. Het EVENT DRIVEN programmeren komt dan vanzelf.

Gordon Rahman

Grafisch programmeren in GW-BASIC (8).

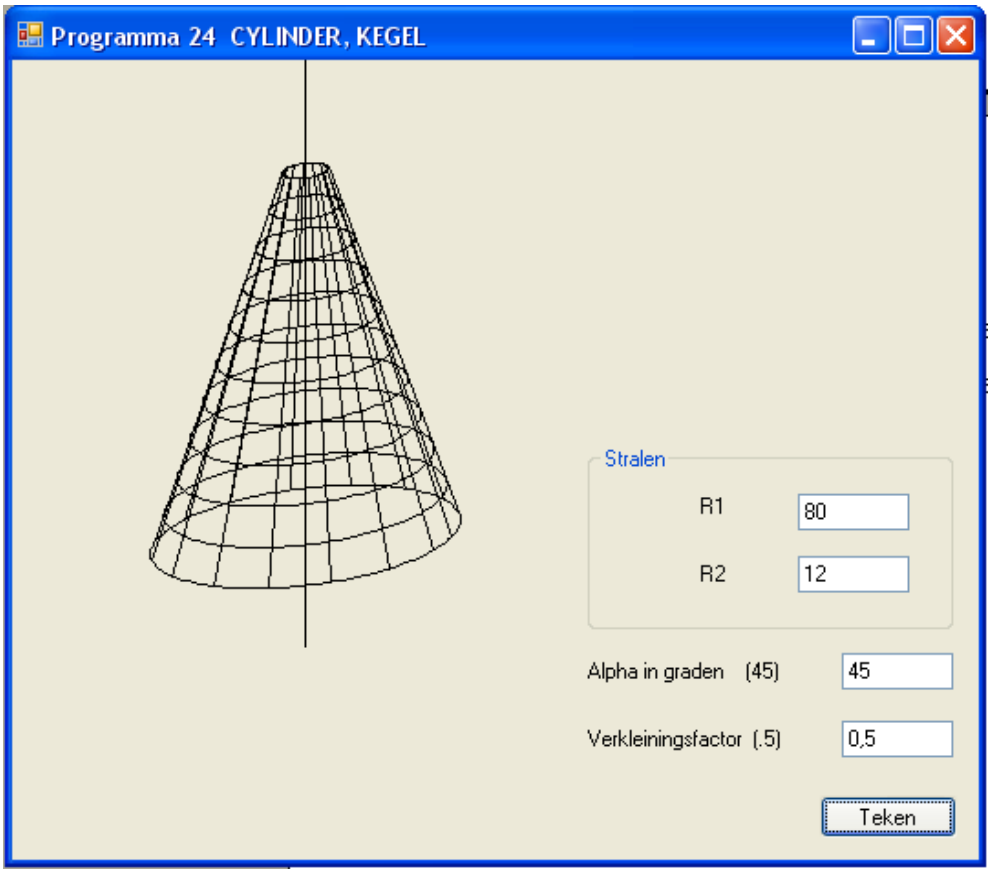
Met de volgende programma's komen er meer soorten functies waarmee we meer kunnen doen dan alleen maar lijnen tekenen. Toch blijven de programma's bij het LINE statement. U kunt ook eens het CIRCLE statement proberen, maar dan moeten wel de functies worden aangepast.

Met **programma 24** kunnen we, naar keuze, cilinders, kegels of afgeknotte kegels tekenen. Als we voor R1 en R2 dezelfde waarde invoeren, krijgen we een cilinder. Als R2 kleiner is dan R1 krijgen we een afgeknotte kegel en voor R2 = 0 krijgen we een hele kegel.

We plaatsen het lichaam zo dat het grondvlak bij z = -100 ligt en het bovenvlak (de top) bij z = +100. Het programma tekent zeven breedtecirkels met een onderlinge afstand van z = 20 en 16 lijnen op de kegel- of cilindermantel loodrecht op de breedtecirkels.
Tot slot wordt verticaal de z-as getekend.

```
100 '       programma 24          CYLINDER,  KEGEL
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "STRALEN   R1 EN R2       "; R1,R2
150 INPUT "ALPHA IN GRADEN     (45) "; A
160 INPUT "VERKLEININGSFACTOR  (.5) "; K
170 U=160:V=160:H=.5:RD=4*ATN(1)/180:DR=(R1-R2)/10
180 W=A*RD : C=K*COS(W) : S=K*SIN(W): N=0
190 CLS
200 FOR Z=-100 TO 100 STEP 20
210    R=R1-N*DR
220    FOR W=0 TO 360 STEP 3
230       W1=W*RD: X=R*COS(W1): Y=R*SIN(W1)
240       XX=INT(U+X+C*Y+H)
250       YY=INT(V-S*Y-Z+H)
260       IF W1=0 THEN X1=XX:Y1=YY:GOTO 300
270       X2=XX: Y2=YY
280       LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
290       X1=X2 : Y1=Y2
300    NEXT W
310    N=N+1
```

```
320 NEXT Z
330 FOR W=0 TO 360 STEP 23
340    W1=W*RD
350    X=R1*COS(W1): Y=R1*SIN(W1)
360    X1=INT(U+X+C*Y+H): Y1=INT(V-S*Y+100+H)
370    X=R2*COS(W1) : Y=R2*SIN(W1)
380    X2=INT(U+X+C*Y+H): Y2=INT(V-S*Y-100+H)
390    LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
400 NEXT W
410 LINE (FNX(U),0) - (FNX(U),320), 1
420 A$=INKEY$: IF A$="" THEN 420
430 CLS: KEY ON: END
```

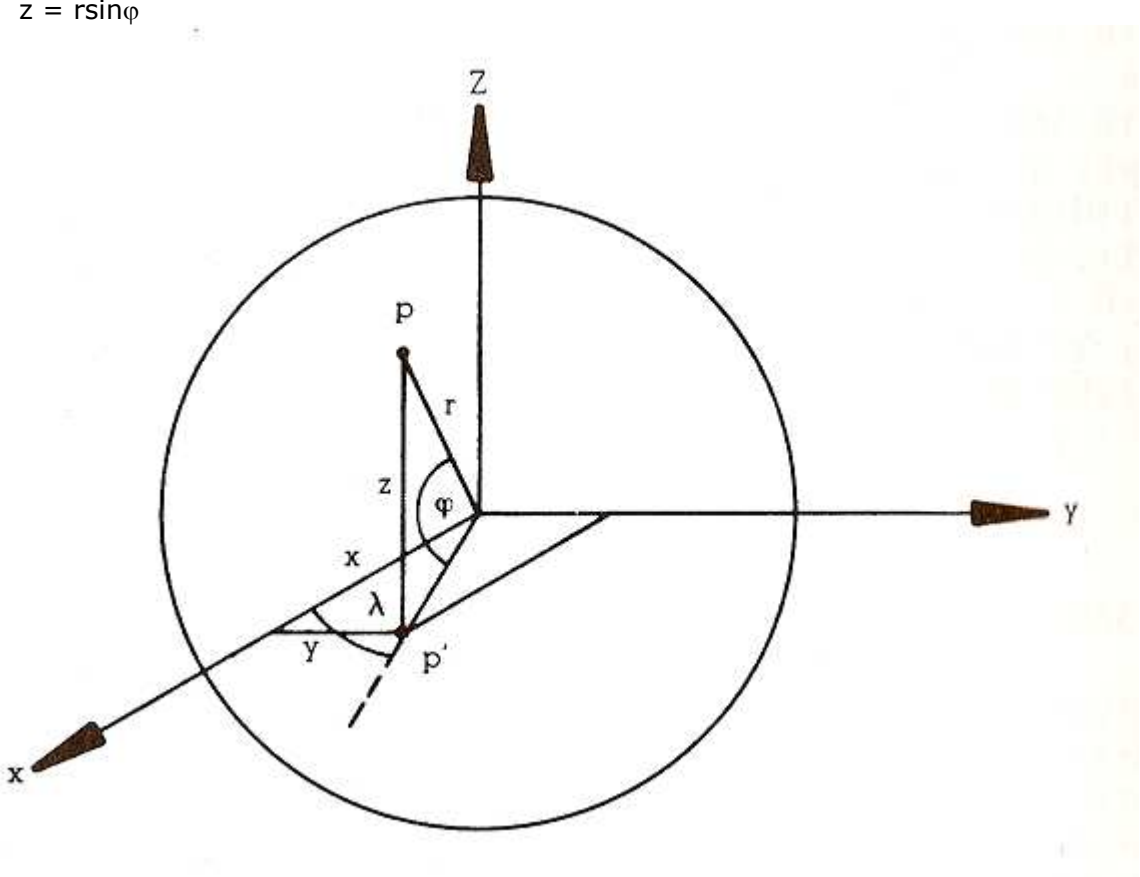



```
Public Class frmProg24
    Private Const U As Integer = 160
    Private Const V As Integer = 160
    Private Const H As Double = 0.5

    Private Sub frmProg24_Paint(ByVal sender As Object, ByVal e As ...PaintEventArgs) ...
        If txtR1.Text() <> "" And txtR2.Text() <> "" And txtAlpha.Text() <> "" And _
           txtFactor.Text() <> "" Then
            Dim R1 As Double = CDbl(txtR1.Text()), R2 As Double = CDbl(txtR2.Text())
            Dim A As Double = CDbl(txtAlpha.Text()), K As Double = CDbl(txtFactor.Text())
            Dim RD As Double = 4 * Math.Atan(1) / 180, DR As Double = (R1 - R2) / 10
            Dim W As Double = A * RD, C As Double = K * Math.Cos(W)
            Dim S As Double = K * Math.Sin(W), N As Integer = 0
            For Z As Integer = -100 To 100 Step 20
                Dim R As Double = R1 - N * DR
                Dim X1, Y1, X2, Y2 As Integer
                For W = 0 To 360 Step 3
                    Dim W1 As Double = W * RD
                    Dim X As Double = R * Math.Cos(W1), Y As Double = R * Math.Sin(W1)
                    Dim XX As Integer = Int(U + X + C * Y + H)
                    Dim YY As Integer = Int(V - S * Y - Z + H)
                    If W1 = 0 Then
                        X1 = XX
                        Y1 = YY
```

```
                    Else
                        X2 = XX
                        Y2 = YY
                        e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                        X1 = X2 : Y1 = Y2
                    End If
                Next
                N += 1
            Next
            For W = 0 To 360 Step 23
                Dim W1 As Double = W * RD
                Dim X As Double = R1 * Math.Cos(W1), Y As Double = R1 * Math.Sin(W1)
                Dim X1 As Integer = Int(U + X + C * Y + H)
                Dim Y1 As Integer = Int(V - S * Y + 100 + H)
                X = R2 * Math.Cos(W1) : Y = R2 * Math.Sin(W1)
                Dim X2 As Integer = Int(U + X + C * Y + H)
                Dim Y2 As Integer = Int(V - S * Y - 100 + H)
                e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
            Next
            e.Graphics.DrawLine(Pens.Black, U, 0, U, 320)
        End If
    End Sub

    Private Sub btnTeken_Click(ByVal sender As Object, ByVal e As System.EventArgs) ...
        Refresh()
    End Sub
End Class
```

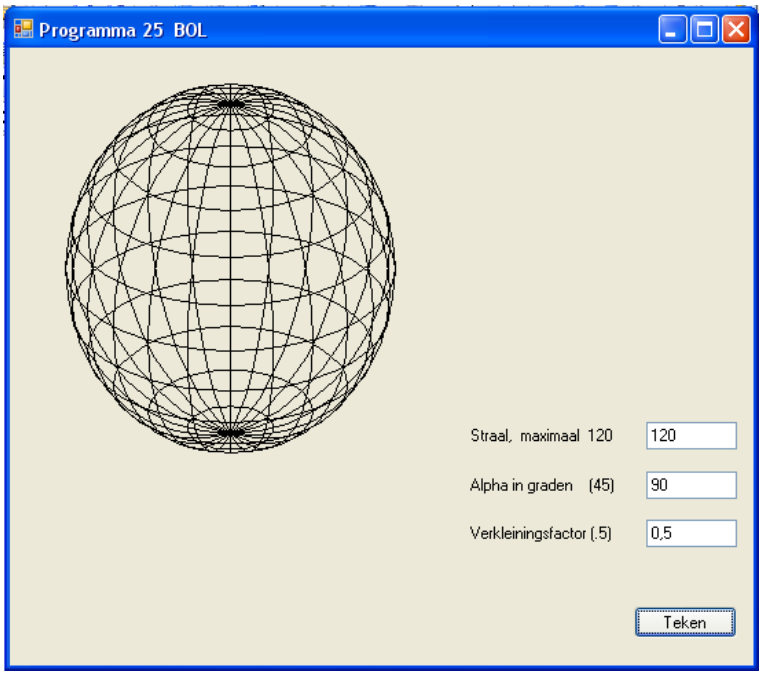
Erg leuk is het tekenen van een bol met breedtelijnen en meridianen. Zoals bekend is kunnen we elk punt op het boloppervlak eenduidig vastleggen met de straal van de bol (r), de geografische breedte (φ) en de geografische lengte (λ), zie figuur. Het omzetten van deze bolcoördinaten r, φ, λ in cartesische coördinaten (x, y, z) geschiedt met de volgende drie vergelijkingen:

$x = r\cos\varphi\cos\lambda$
$y = r\cos\varphi\sin\lambda$
$z = r\sin\varphi$



De bolvergelijking in cartesische vorm is overigens $x^2+y^2+z^2=r^2$.

In **programma 25** wordt de hoek φ door de W en de hoek λ door P vertegenwoordigd. Kies voor alpha 90^0 en voor k 0,5. Andere waarden vervormen de bol.




```
100 '       programma 25                BOL
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "STRAAL,  MAXIMAAL  120  " ; R
150 INPUT "ALPHA IN GRADEN    (45) " ; A
160 INPUT "VERKLEININGSFACTOR (.5) " ; K
170 U=160: V=160: H=.5: RD=4*ATN(1)/180
180 W=A*RD : C=K*COS(W) : S=K*SIN(W)
190 CLS
200 FOR W=-90 TO 90 STEP 15
210    W1=W*RD : R1=R*COS(W1)
220    FOR P=0 TO 360 STEP 10
230       P1=P*RD
240       X=R1*COS(P1):Y=R1*SIN(P1):Z=R*SIN(W1)
250       IF P=0 THEN X1=INT(U+X+H)
260       IF P=0 THEN Y1=INT(V-Z+H): GOTO 300
270       X2=INT(U+X+C*Y+H): Y2=INT(V-S*Y-Z+H)
280       LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
290       X1=X2 : Y1=Y2
300    NEXT P
310 NEXT W
320 FOR P=0 TO 180 STEP 15
330    P1=P*RD
340    FOR W=0 TO 360 STEP 10
350       W1=W*RD: R1=R*COS(W1)
360       X=R1*COS(P1): Y=R1*SIN(P1): Z=R*SIN(W1)
370       IF W=0 THEN X1=INT(U+X+C*Y+H)
380       IF W=0 THEN Y1=INT(V-S*Y-Z+H):GOTO 420
390       X2=INT(U+X+C*Y+H): Y2=INT(V-S*Y-Z+H)
400       LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
410       X1=X2 : Y1=Y2
420    NEXT W
430 NEXT P
440 A$=INKEY$: IF A$="" THEN 440
450 CLS: KEY ON: END
```

```
Public Class frmProg25
    Private Const U As Integer = 160
    Private Const V As Integer = 160
    Private Const H As Double = 0.5

    Private Sub frmProg25_Paint(ByVal sender As Object, ByVal e As ...PaintEventArgs) ...
        If txtR.Text() <> "" And txtA.Text() <> "" And txtK.Text() <> "" Then
            Dim R As Double = CDbl(txtR.Text())
            Dim A As Double = CDbl(txtA.Text())
            Dim K As Double = CDbl(txtK.Text())
            Dim RD As Double = 4 * Math.Atan(1) / 180
            Dim W As Double = A * RD
            Dim C As Double = K * Math.Cos(W), S As Double = K * Math.Sin(W)
            Dim X1, Y1, X2, Y2 As Integer
            For W = -90 To 90 Step 15
                Dim W1 As Double = W * RD, R1 As Double = R * Math.Cos(W1)
                For P As Integer = 0 To 360 Step 10
                    Dim P1 As Double = P * RD
                    Dim X As Double = R1 * Math.Cos(P1)
```

```
                    Dim Y As Double = R1 * Math.Sin(P1), Z As Double = R * Math.Sin(W1)
                    If P = 0 Then
                        X1 = Int(U + X + H)
                        Y1 = Int(V - Z + H)
                    Else
                        X2 = Int(U + X + C * Y + H)
                        Y2 = Int(V - S * Y - Z + H)
                        e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                        X1 = X2 : Y1 = Y2
                    End If
                Next
            Next
            For P As Integer = 0 To 180 Step 15
                Dim P1 As Double = P * RD
                For W = 0 To 360 Step 10
                    Dim W1 As Double = W * RD
                    Dim R1 As Double = R * Math.Cos(W1)
                    Dim X As Double = R1 * Math.Cos(P1)
                    Dim Y As Double = R1 * Math.Sin(P1), Z As Double = R * Math.Sin(W1)
                    If W = 0 Then
                        X1 = Int(U + X + C * Y + H)
                        Y1 = Int(V - S * Y - Z + H)
                    Else
                        X2 = Int(U + X + C * Y + H)
                        Y2 = Int(V - S * Y - Z + H)
                        e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
                        X1 = X2 : Y1 = Y2
                    End If
                Next
            Next
        End If
    End Sub

    Private Sub btnTeken_Click(ByVal sender As Object, ByVal e As System.EventArgs) ...
        Refresh()
    End Sub
End Class
```

Een geliefd demonstratieprogramma van computerleveranciers is een om zijn as draaiend driedimensionaal lichaam. Meestal wordt als 'draai-as' de z-as gekozen, die dan samenvalt met de lengte-as van het lichaam.

Microcomputers die alleen een BASIC vertolker bezitten zijn te langzaam om zo'n draaiing real-time uit te voeren. Er zijn namelijk nogal ingewikkelde trigonometrische berekeningen voor nodig, die tamelijk veel tijd in beslag nemen. Daarnaast duurt het tekenen van het lichaam in een bepaalde stand in BASIC zo'n 1 à 2 seconden. Zouden we in een BASIC programma steeds het lichaam tekenen, uitwissen, draaien, tekenen, uitwissen, draaien, ... enzovoort, dan zien we draaiende lichaam in plaats van een vloeiende beweging een schokkerige beweging maken. De beste oplossing voor dit probleem is het volgende:

Deel de totale middelpuntshoek van 360^0 in n even grote hoeken.
Voor elk van deze hoeken

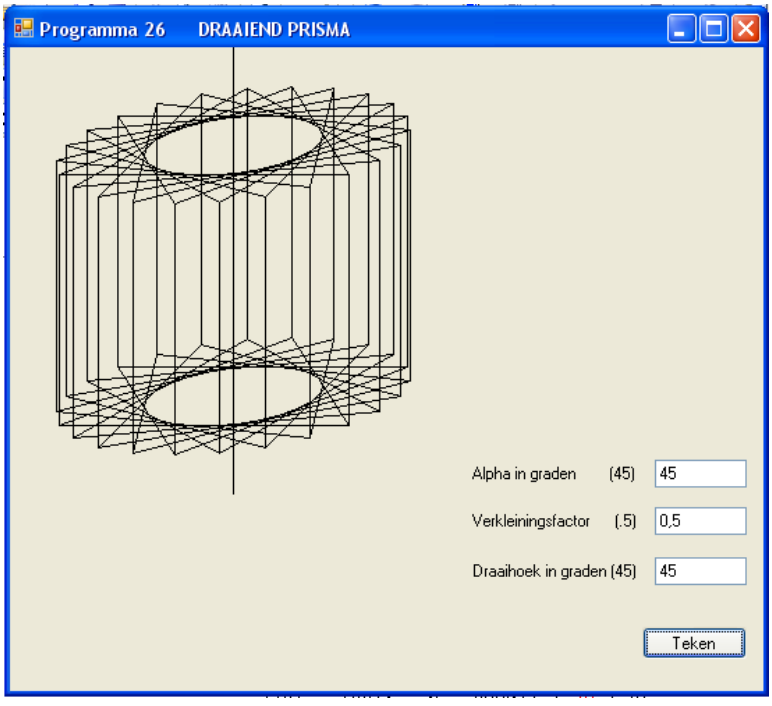
$$\omega_1 = \frac{360^0}{n},\ \omega_2 = 2.\omega_1,\ .....,\ \omega_n = n.\omega_1 = 360^0$$

berekenen we van te voren de coördinaten van de hoekpunten van het lichaam. Voor elke stand slaan we deze hoekpuntscoördinaten in een array op. Al deze berekeningen voeren we in BASIC uit bij een leeg beeldscherm. Als de berekeningen klaar zijn wordt een machinetaalprogramma uitgevoerd dat de eerste groep coördinaten uit de array haalt, het lichaam tekent, na een bepaalde tijd het beeldscherm wist, de volgende groep coördinaten ophaalt, ... enzovoort. Deze oplossing heeft een redelijk vloeiende draaiing tot gevolg. Dit programma is geschreven op een CBM 3016 die een 6502-microprocessor bezit. Hier doen we echter anders.

Programma 26 laat een driezijdig prisma om de z-as draaien. De z-as is zowel de lengte-as als de symmetrie-as van het prisma. Het programma tekent een prisma. Elk volgend prisma wordt over zijn voorgangers heen getekend.

Tegenwoordig kunnen we het draaien niet meer volgen. Ook in BASIC is het tekenen erg snel geworden en hoeven we ons geen zorgen meer te maken over vertraagd werkende programma's. De listing die de tekening laat zien (Visual Basic .NET 2008) tekent de draaiende prisma zo snel, alsof de hele tekening maar één afbeelding is.

```
100 '      programma 26      DRAAIEND PRISMA
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 INPUT "ALPHA  IN GRADEN    (45)"; A
150 INPUT "VERKLEININGSFACTOR  (.5)"; K
160 INPUT "DRAAIHOEK IN GRADEN (45)"; OM
170 U=160: V=160: H=.5: RD=4*ATN(1)/180: R=120
180 W=A*RD : C=K*COS(W) : S=K*SIN(W)
190 CLS
200 LINE (FNX(U),0) - (FNX(U),320),1
210 DIM X(7), Y(7)
220 FOR W=0 TO 360 STEP OM
230    FOR J=0 TO 3
240       W1=(W+J*120)*RD
250       X=R*COS(W1) : Y=R*SIN(W1)
260       X(J)=INT(U+X+C*Y+H):Y(J)=INT(V-S*Y+90+H)
270       X(J+4)=X(J) : Y(J+4)=INT(V-S*Y-90+H)
280    NEXT J
290    FOR J=0 TO 2
300       X1=X(J):Y1=Y(J):X2=X(J+1):Y2=Y(J+1)
310       LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
320    NEXT J
330    FOR J=0 TO 2
340       X1=X(J):Y1=Y(J):X2=X(J+4):Y2=Y(J+4)
350       LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
360    NEXT J
370    FOR J=4 TO 6
380       X1=X(J):Y1=Y(J):X2=X(J+1):Y2=Y(J+1)
390       LINE (FNX(X1),Y1)-(FNX(X2),Y2),1
400    NEXT J
410 NEXT W
420 A$=INKEY$: IF A$="" THEN 420
430 CLS :KEY ON: END
```



Onderstaande code van VB 2008 laat een hele andere opbouw zien. In de J lussen wordt er direct getekend en worden de elementen van de array X en Y niet apart bewaard in X1 en Y1 en X2 en Y2. Hoeft ook niet. Ik zou anders in elke lus telkens die variabelen moeten declareren, want variabelen zijn in de lussen lokaal.
Wat u ook kunt zien zijn de twee variabelen LocalX en LocalY. Waarom noem ik deze zo? In de oude BASIC versies kunnen we nogmaals een variabele gebruiken, ook al hebben we al een array van die variabele. Oud BASIC kan ze uit elkaar houden, maar Visual Basic helaas niet.

```
Public Class frmProg26
    Private Const U As Integer = 160
    Private Const V As Integer = 160
    Private Const H As Double = 0.5
    Private Const R As Integer = 120

    Private Sub frmProg26_Paint(ByVal sender As Object, ByVal e As ...PaintEventArgs) ...
        If txtA.Text() <> "" And txtK.Text() <> "" And txtOM.Text() <> "" Then
            Dim A As Double = CDbl(txtA.Text())
            Dim K As Double = CDbl(txtK.Text())
            Dim OM As Double = CDbl(txtOM.Text())
            Dim RD As Double = 4 * Math.Atan(1) / 180
            Dim W As Double = A * RD
            Dim C As Double = K * Math.Cos(W), S As Double = K * Math.Sin(W)
            With e.Graphics
                .DrawLine(Pens.Black, U, 0, U, 320)
                Dim X(7), Y(7) As Integer
                For W = 0 To 360 Step OM
                    For J As Integer = 0 To 3
                        Dim W1 As Double = (W + J * 120) * RD
                        Dim LocalX As Double = R * Math.Cos(W1)
                        Dim LocalY As Double = R * Math.Sin(W1)
                        X(J) = Int(U + LocalX + C * LocalY + H)
                        Y(J) = Int(V - S * LocalY + 90 + H)
                        X(J + 4) = X(J)
                        Y(J + 4) = Int(V - S * LocalY - 90 + H)
                    Next
                    For J As Integer = 0 To 2
                        .DrawLine(Pens.Black, X(J), Y(J), X(J + 1), Y(J + 1))
                    Next
                    For J As Integer = 0 To 2
                        .DrawLine(Pens.Black, X(J), Y(J), X(J + 4), Y(J + 4))
                    Next
                    For J As Integer = 4 To 6
                        .DrawLine(Pens.Black, X(J), Y(J), X(J + 1), Y(J + 1))
                    Next
                Next
            End With
        End If
    End Sub

    Private Sub btnTeken_Click(ByVal sender As Object, ByVal e As System.EventArgs) ...
        Refresh()
    End Sub
End Class
```

De volgende keer komt de laatste tekening over 3D figuren. Daarna komt er wat theorie, en uiteraard ook weer wat programma's, over het tekenen van vlakken in de ruimte.

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.

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	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) 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