Basic Bulletin

20ste jaargang november 2013 Nummer 3

HCC BASIC interessegroep

Inhoud

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

Hoofdstuk 8 van PowerBASIC bereikt zijn laatste deel over modulair programmeren. De laatste paragrafen gaan over recursie, units en twee statements die we met gebruik van units nodig hebben: PUBLIC en EXTERNAL.

BBC BASIC is altijd een belangrijke BASIC versie geweest, vooral in de tijd toen zulke programmeertalen nog werden gebruikt op technische scholen.

Wie herinnert zich nog de tijd van de Basicode? Op tv werden de listings weergegeven die zo algemeen geschreven waren dat ze op elke homecomputer in elk BASIC dialect overgenomen konden worden.

Marco Kurvers

BASIC leren – PowerBASIC hoofdstuk 8 (vervolg).

Modulair programmeren

Recursie

Recursie is het proces waarbij een functie of een procedure zichzelf direct, hetzij indirect, aanroepen kan. Hier is een recursieve uitlevering van FN*faculteit##*.

```
DEF FNFaculteit##(getal%)
     IF getal% > 1 AND getal% <= 1754 THEN
          FNFaculteit## = getal% * FNFaculteit##(getal%-1)
     ELSEIF getal% = 0 OR getal% = 1 THEN
          FNFaculteit## = 1
     ELSE
          FNFaculteit## = -1
     END IF
END DEF
```

Eén ding is onmiddellijk duidelijk: het recursieve algoritme is korter dan een niet-recursief algoritme. FN*Faculteit##* is teruggebracht tot één IF blok zonder lokale variabelen. Beknoptheid is een karakteristiek van recursieve programma's.

Een andere karakteristiek van recursieve code is misleidende complexiteit. De beste manier om recursieve algoritmen te begrijpen is om te werken door middel van sommige testgevallen met potlood en papier. Laten we de faculteit van 3 bepalen, het PowerBASIC programma zou na het volgende statement het volgende geven:

```
PRINT FNFaculteit##(3)
```

Om te beginnen, een argument van 3 wordt doorgegeven bij waarde aan de functie. Wat FN*Faculteit##* als eerste doet is controleren om te zien of het argument groter dan 1 en kleiner of gelijk is aan 1754; 3 doorstaat deze test, dus het statement na het eerste THEN clause is uitgevoerd:

```
FNFaculteit## = 3 * FNFaculteit##(2)
```

Deze regel kent aan de functienaam de waarde 3 (tot dusverre goed) toe, vermenigvuldigd bij de waarde van FN*Faculteit##*(2), wat dat ook is. Voordat de toekenning plaats kan vinden, moet u FN*Faculteit##* opnieuw aanroepen, nu met het argument van 2. Houdt het gezette 'wordt vervolgd' toekenningsstatement op, dan zult u later weer terugkeren.

Een tweede aanroep van FN*Faculteit##* ontvangt het argument van 2 en brengt normaal op. Het ontdekt weer dat het argument groter is dan 1 en kleiner of gelijk is aan 1754, dus vindt er weer een recursief statement plaats

```
FNFaculteit## = 2 * FNFaculteit##(1)
```

Nadat het gezette toekenningsstatement ook weer ophoudt, roept het voor de derde keer FN*Faculteit##* aan, deze keer met een argument van 1.

Het is gemakkelijk om FN*Faculteit##*(1) uit te zoeken – het is gedefinieerd als 1 in de tweede THEN clause. Deze keer tot slot is FN*Faculteit##* toegestaan om terug te keren, en het geeft de waarde te-

rug van 1 naar het meest recent gezette opgehouden aanroep van FN*Faculteit##*. En zijn toekenningsstatement is toegestaan om te beëindigen:

```
FNFaculteit## = 2 * 1
```

Met dit complete aanroepmiddel van FN*Faculteit##* is een waarde van 2 teruggegeven aan de originele aanroep van FN*Faculteit##* en het eerste 'wordt vervolgd' statement:

```
FNFaculteit## = 3 * 2
```

Na deze laatste toekenning, wordt de controle teruggegeven aan het PRINT statement en een 6 verschijnt op het scherm.

Er zijn geen zwart en wit antwoorden naar wanneer u recursie zult gebruiken, maar wel in algemeen gebruik van als het probleem zelf een recursieve smaak heeft. *Faculteiten* zijn bijvoorbeeld soms gedefinieerd in wiskundige recursieve leerboeken: Voor elke positieve integer *n*,

```
if n > 1 then
     n! = n * (n-1)!
else
     n! = 1
```

Met behulp van een recursief algoritme zal waarschijnlijk worden vereist dat u uw programma met het $STACK metastatement het uitvoerstack moet laten verhogen, aangezien elk niveau van recursie maar liefst 125 bytes kan bevatten (in de meeste gevallen wordt er veel minder vereist dan die hoeveelheid). Om de hoeveelheid ruimte in stack links te kunnen bepalen, kunt u gebruik maken van de functie FRE(-2).

Als u wilt om recursieve technieken te gebruiken, zorg er dan voor dat u begrijpt dat er soms bijwerkingen kunnen optreden. Ten eerste, recursie gebruikt extra stackruimte. Ten tweede, en belangrijker nog, kan recursieve programmering verleidelijk elegant zijn, en het is duivels lastig om te zuiveren, want het is moeilijker om de exacte stappen te volgen in de verwerking. In het bijzonder, controleer om zeker te weten dat de functie een weg heeft naar een einde; vaak is een stack overflow fout een symptoom van een weggelopen recursie.

Units

Units zijn aparte gecompileerde modules van PowerBASIC. Elke unit is gelijkend een apart programma, behalve dat het alleen samengesteld is uit functies en procedures. U kunt units gebruiken om library's te creëren van geteste, perfecte programmacode die u telkens in elk ander programma kunt gebruiken. U mag zoveel units hebben als u wilt. Gebruik het $LINK metastatement om uw programma te vertellen waar en wanneer een unit te gebruiken is.

Units beheersen de volgende regels:

1. Een unit kan maximaal 64K aan complete code bevatten ($SEGMENT metastatements zijn niet toegestaan in een unit).
2. Een unit kan alleen uit procedures, functies, DEF FN functies, EXTERNAL statements, SHARED statements en metastatements bestaan. Natuurlijk kunt u elk voorgedefinieerde functie of statement, zo lang u het binnen een functie of procedure invoegt, gebruiken.
3. In een unit kunnen geen COMMON variabelen gedeclareerd worden.
4. Een unit is gelinkt binnen het hoofdprogramma bij het invoegen van een $LINK metastatement in het hoofdprogramma.
5. Een unit mag code bevatten die een routine aanroept in een extern object bestand, zo lang dat bestand gelinkt is binnen het hoofdprogramma en zo lang een DECLARE statement voor de

routine is ingevoegd in de unit code voordat de routine aangeroepen is.

6. Alle procedures en functies zijn standaard in een unit ontoegankelijk (private) van de rest van het programma. U moet in het hoofd van elke procedure of functie aangeven of u het open wilt buiten de unit (gebruik het PUBLIC sleutelwoord). De openlijke routines moeten dan in het hoofdprogramma gedeclareerd worden met het DECLARE statement voordat er verwezen kan worden.
   Alle variabelen, constanten en DEF FN functies zijn altijd verborgen.
7. Aangezien alle procedures en functies in units standaard ontoegankelijk (private) zijn of andere externe code (in objectbestanden), moet u niet vergeten in het hoofd van elke procedure of functie PUBLIC op te geven voor het hoofdprogramma dat vanuit een unit wordt benaderd.

Twee PowerBASIC statements, PUBLIC en EXTERNAL, worden gebruikt voor het definiëren van de interface tussen variabelen in het hoofdprogramma en deze in units.

PUBLIC statement

Gebruik het PUBLIC statement om variabelen te declareren in uw hoofdprogramma die u zichtbaar wilt maken en toegankelijk voor uw units. Deze variabelen zullen ook toegankelijk worden van elk object (.OBJ) bestand die u linkt binnen uw programma met het $LINK metastatement. PUBLIC statements zijn alleen aanwezig in uw hoofdprogramma, niet in externe units. De syntaxis is:

```
PUBLIC var [,var]...
```

Als voorbeeld:

```
PUBLIC a!,b$,c$,hond&
```

Het PUBLIC statement heeft niets te maken met het sleutelwoord PUBLIC, die is toegevoegd aan procedure en functie headers om de routines zichtbaar te maken buiten units of het hoofdprogramma.

EXTERNAL statement

Het EXTERNAL statement is de tegenhanger van PUBLIC. Het wordt gebruikt in een unit om variabelen te declareren die de unit gebruikt, maar die zijn gedefinieerd in het hoofdprogramma (met een PUBLIC statement).

Als u vergeet een externe variabele te declareren met het EXTERNAL statement, zal de compiler de variabele behandelen alsof het lokaal is voor de entiteit die het gebruikt. Als u een variabele declareert met EXTERNAL en die variabele is niet gedefinieerd in het hoofdprogramma met PUBLIC, zult u een foutmelding krijgen wanneer u het hoofdprogramma compileert. Gedeelde variabelen tussen het hoofdprogramma en externe units moeten ook op dezelfde naam gedeeld worden in de hoofd- en externe code in tegenstelling tot COMMON blokken (gebruikt met geketende programma's), waar verschillende namen kunnen verwijzen naar dezelfde variabele opslaglocatie.

De syntaxis is:

```
EXTERNAL var [,var]...
```

Als u bijvoorbeeld heeft

```
PUBLIC a!,b$,c$,hond&
```

in uw hoofdprogramma, dan heeft u

```
EXTERNAL a!,c$
```

in uw unit. Dit zorgt ervoor dat in de unit naar de variabelen in het hoofdprogramma *a!* en *c$* gerefereerd kan worden.

Tot slot

U hebt enkel te maken gehad over alles wat je kunt doen om nuttige, krachtige en flexibele programma's te schrijven. U hebt een onderwerp met meer informatie nodig: hoe behandel je bestanden. Dit onderwerp wordt behandeld in het volgende hoofdstuk. Natuurlijk, zodra u uw tenen nat hebt gekregen, kunt u om in te duiken meer leren van geavanceerde technieken, bijvoorbeeld over hoe afbeeldingen weergegeven worden of hoe assembler routines gemaakt worden, enzovoort. Deze onderwerpen en meer komen aan bod in Deel 3.

BBC BASIC beter leren kennen.

Gegevens groeperen – werken met arrays.

Om redenen die zal blijken worden matrices vaak gebruikt in combinatie met lussen. Ter illustratie van de noodzaak voor arrays, laten we doen alsof u werkt op een space invaders spel. Ergens in de buurt van het einde van de code moet u zien of er nog aliens in leven zijn, zodat u met een nieuw level beginnen kunt. Als elke alien een variabele is die zeggen kan of het nog in leven is, dan kunt u dit doen:

```
REM ...
NewLevel=TRUE : REM Assume new level
IF Alien1Alive THEN NewLevel=FALSE
IF Alien2Alive THEN NewLevel=FALSE
IF Alien3Alive THEN NewLevel=FALSE
IF NewLevel THEN
  REM setup new level
ENDIF
REM ...
```

In de eerste regel wordt ervan uitgegaan dat een nieuw level vereist is. Vervolgens inspecteert het op hun beurt elke AlienXAlive vlag en als deze alien nog in leven is, zal de NewLevel vlag opnieuw op FALSE worden gezet. Als de controle is voltooid en de vlag nog steeds waar is, dan moeten de aliens dood zijn en wordt er een nieuwe level gestart. Vraag: wat gebeurt er als er 40 of 100 aliens zijn? Met behulp van deze methode kost dat een heleboel regels code. Er moet dus een betere manier zijn:

```
DIM AlienAlive(50)
REM ...
NewLevel=TRUE : REM Assume new level
FOR I%=1 TO 50
  IF AlienAlive(I%) THEN NewLevel=FALSE
NEXT I%
IF NewLevel THEN
  REM setup new level
ENDIF
REM ...
```

In de eerste regel ziet u het sleutelwoord DIM. DIM staat voor dimensie en is een instructie om het computergeheugen van een arrayvariabele te creëren. De array vertelt hoeveel basisvariabelen tussen de haakjes staan achter de naam. In dit geval vertellen we de computer dat we 50 aliens nodig hebben, dus worden er 50 locaties gereserveerd, één voor elk. Eerder definieerden we onze BASIC

variabelen apart voor gebruik, maar een array is anders. Door het definiëren van de grootte van een matrix, kan BASIC telkens een selectievakje uitvoeren wanneer die gebruikt wordt in code. Als we proberen buiten de matrix te komen, zal BASIC het programma stopzetten en met een bericht komen. Hoewel niet verplicht, is het verstandig om alle arrays op de top van het programma te plaatsen. Dit zet ze allemaal samen in één plaats en worden ze allemaal verwezen naar het geheugen wanneer het programma wordt opgestart.

Zodra het gedeclareerd is, kunt u vrij veel met een matrix dan met een andere variabele doen. Elk afzonderlijk element wordt benaderd door zijn nummer te geven: AlienAlive(1), AlienAlive(39) of zoals in de regel gebruikt: AlienAlive(I%)… de teller in een FOR-lus beslist welk element geïnspecteerd moet worden. Met behulp van deze methode kunnen we zoveel aliens allemaal in hetzelfde aantal regels controleren.

Hier is een volledig voorbeeld dat de rangen opslaat voor een aantal leerlingen en dan iets met de gegevens doet:

```
REM Student grade program
DIM Grade%(5)

REM First, collect the data
FOR I%=1 TO 5
  PRINT "Enter grade for student ";I%;": ";
  INPUT Grade%(I%)
NEXT I%

REM Work out the average
Total%=0
FOR I%=1 TO 5
  Total%=Total%+Grade%(I%)
NEXT I%
REM Print the average
PRINT "The average grade was ";Total%/5

REM Find the minimum grade
Minimum%=999
FOR I%=1 TO 5
  IF Grade%(I%)<Minimum% THEN
    Minimum%=Grade%(I%)
  ENDIF
NEXT I%
REM Print the minimum
PRINT "The minimum grade was ";Minimum%

END
```

Arrays kunnen worden gebruikt om de gegevens te bewaren, maar een van de problemen is het verkrijgen van de gegevens in de array. In het programma hierboven is het invoeren fijn omdat u waarschijnlijk verschillende waarden wilt bewaren. Beschouw dit:

```
REM Days in a month
DIM Month%(12)

REM First, collect the data
FOR I%=1 TO 12
  PRINT "Enter days in month ";I%;": ";
```

```
  INPUT Month%(I%)
NEXT I%

REM Now ask for month
INPUT "Enter month number: " M%
PRINT "Month ";M%;" has ";Month%(M%);" days."

END
```

Met deze routine kan een probleem optreden bijvoorbeeld als het aantal dagen niet juist is voor de ingevoerde maand.

```
REM Days in a month
DIM Month%(12)

REM First, collect the data
FOR I%=1 TO 12
  READ Month%(I%)
NEXT I%

REM Now ask for month
INPUT "Enter month number: " M%
PRINT "Month ";M%;" has ";Month%(M%);" days."

END
DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

Dat is veel beter. Er zijn een paar trefwoorden hier, READ en DATA. DATA bevat een verzameling numerieke gegevens of tekenreeksen die kunnen worden gebruikt in het programma. Wat de gegevens zijn en de volgorde die u aanhoudt is geheel aan u. Wanneer het programma een READ statement tegenkomt, gaat het af op het DATA statement. Vervolgens leest het de waarde terug in de variabele, een beetje zoals INPUT doet. BASIC onthoudt waar het de waarde kreeg, zodat het de volgende keer die waarde ontmoet en vanaf daar de volgende waarde weer terugleest.

DATA regels mogen worden gesplitst. Bovenstaande DATA regel kunt u bijvoorbeeld in meerdere regels plaatsen, of zelfs elke waarde in een aparte DATA regel plaatsen. Het is geheel aan u hoe u het programma wilt schrijven. Dit geldt ook voor de andere BASIC dialecten, dit hoeft dus niet speciaal voor BBC BASIC te gelden.

U kunt de gegevens mixen zolang het juiste type in de juiste variabele gelezen wordt.

```
REM Days in a month
DIM Month%(12), Name$(12)

REM First, collect the data
FOR I%=1 TO 12
  READ Month%(I%)
  READ Name$(I%)
NEXT I%

REM Now ask for month
INPUT "Enter month number: " M%
PRINT Name$(M%);" has ";Month%(M%);" days."

END
```

```
DATA 31,January,28,February,31,March
DATA 30,April,31,May,30,June
DATA 31,July,31,August,30,September
DATA 31,October,30,November,31,December
```

DATA instructies kunnen overal worden geplaatst in het programma, BASIC zal ze negeren tot er verteld wordt om ze te gebruiken. Ze zijn meestal geplaatst op de bodem van het programma na het einde zodat er geen rommelcode ontstaat, met uitzondering die hieronder wordt gegeven.

Er zijn gelegenheden wanneer het nodig is om een set gegevens te herlezen. Stel dat u een standaardset van waarden hebt die opnieuw gelezen moeten worden van een optie in een menu. Als u wilt dat de gegevensaanwijzer terug naar een specifieke plaats gaat, gebruikt u de opdracht RESTORE. RESTORE heeft een paar opties; bij geen gegeven optie wordt de gegevensaanwijzer op de eerste DATA instructie in het programma ingesteld.

Een ander gebruik is: regelnummers. Jawel, ook BBC BASIC kent de oude methode om programma's te schrijven. Wanneer u regelnummers gebruikt, kunt u achter RESTORE een regelnummer plaatsen zodat de juiste DATA regel ingesteld wordt.

```
 10 REM Months in English and French
 20 DIM Month$(12)
 30 REM First, collect the data
 40 INPUT "Do you want French? (y/n) " Ans$
 50 IF Ans$="N" OR Ans$="n" THEN
 60   RESTORE 170
 70 ELSE
 80   RESTORE 210
 90 ENDIF
100 FOR I%=1 TO 12
110   READ Month$(I%)
120 NEXT I%
130 REM Now ask for month
140 INPUT "Enter month number: " M%
150 PRINT "Month ";M%;" is ";Month$(M%)
160 END
170 DATA January,February,March
180 DATA April,May,June
190 DATA July,August,September
200 DATA October,November,December
210 DATA Janvier,Fevrier,Mars
220 DATA Avril,Mai,Juin
230 DATA Juillet,Aout,Septembre
240 DATA Octobre,Novembre,Decembre
```

Als u de regelnummers niet wilt gebruiken, geeft RESTORE ons nog een andere optie. We specificeren een verschuiving van de regel met de RESTORE instructie (niet de regel met het eerste DATA statement). Het gegeven nummer vertelt BASIC het aantal gegevens vanaf de huidige positie vooruit. Om aan te geven dat we een offset en niet een regelnummer gebruiken, moet het nummer worden voorafgegaan met een +. Hier is het bovenstaande programma zonder regelnummers met behulp van deze methode. Bekijk de nummers in de RESTORE regels en tel vooruit zodat u kunt zien vanaf welke DATA regel gelezen wordt. Let op dat de offset *niet* per regel vooruit telt, maar vooruit telt op het aantal gegevens.

```
REM Months in English and French
DIM Month$(12)
```

```
REM First, collect the data
INPUT "Do you want French? (y/n) " Ans$
IF Ans$="N" OR Ans$="n" THEN
  RESTORE +11
ELSE
  RESTORE +13
ENDIF
FOR I%=1 TO 12
   READ Month$(I%)
NEXT I%
REM Now ask for month
INPUT "Enter month number: " M%
PRINT "Month ";M%;" is ";Month$(M%)
END
DATA January,February,March
DATA April,May,June
DATA July,August,September
DATA October,November,December
DATA Janvier,Fevrier,Mars
DATA Avril,Mai,Juin
DATA Juillet,Aout,Septembre
DATA Octobre,Novembre,Decembre
```

Net als in Visual Basic .NET is het ook in BBC BASIC mogelijk om gegevens direct in een array te initialiseren. Hieronder ziet u een voorbeeld:

```
REM Inline initialization
DIM MyArray%(3)

MyArray%() = 1,2,3,4
FOR I%=0 TO 3
  PRINT MyArray%(I%)
NEXT I%

END
```

U kunt ook direct een waarde in de hele array initialiseren door onderstaand voorbeeld te gebruiken:

```
REM Set each element to 50
DIM MyArray%(100)

MyArray%() = 55
FOR I%=0 TO 100
  PRINT MyArray%(I%)
NEXT I%
```

Multidimensionale arrays zijn geen probleem. Wilt u lange initialisaties splitsen, vergeet dan niet om voor de schuine streep een komma te plaatsen (zie hieronder) :

```
REM Initializing a multi-dimensional array
DIM MyArray%(2,3)

MyArray%() = 1,2,3,4,10,20,30, \
\ 40,100,200,300,400
FOR I%=0 TO 2
```

```
  FOR J%=0 TO 3
    PRINT MyArray%(I%,J%)
  NEXT J%
NEXT I%

END
```

De grootte van een array vinden.

De DIM instructie kan worden gebruikt als een functie. Het kan één of twee doorgegeven argumenten hebben. De eerste parameter is altijd de naam van de matrix. Wanneer alleen gebruik gemaakt wordt met alleen de naam, zal DIM reageren met het aantal dimensies van de matrix.

```
REM Finding the size of an array
DIM Array1D(10)
DIM Array2D(10,9)
DIM Array3D(10,9,8)

PRINT "Array","Dimensions"
PRINT "Array1D",DIM(Array1D())
PRINT "Array2D",DIM(Array2D())
PRINT "Array3D",DIM(Array3D())

END
```

In andere BASIC dialecten bestaan ook zulke functies. Echter, in elk dialect kan de functienaam anders zijn.

Zodra u het aantal dimensies weet, is het mogelijk DIM te vragen om de grootte van het gegeven nummer van de dimensie. Geef dit als tweede parameter door.

```
REM Finding the size of an array
DIM Array3D(10,9,8)
PRINT "Dimension 2 has "; \
\       DIM(Array3D(),2);" elements."
END
```

We kunnen voor elke dimensie de grootte vinden door de PRINT regel in een FOR lus te plaatsen.

Probeer eens hiermee te experimenteren als u BBC BASIC heeft. Hieronder ga ik verder met objecten.

Gegevens groeperen in structuren.

Arrays zijn een geweldige manier om gegevens te ordenen. In uw programma's komt u over talrijke toepassingen voor hen. Uiteindelijk kan het volgende gebeuren: stel dat u een killer game wilt schrijven. De speler heeft verschillende variabelen met hem verbonden: de x-coördinaat, de y-coördinaat, het nummer van het leven en de naam. U kunt gemakkelijk verscheidene variabelen definiëren zoals dit:

```
PlayerX%=5
PlayerY%=12
PlayerLives%=3
PlayerName$="GR8"
```

Of u kunt een matrix gebruiken voor de eerste drie variabelen en maak een mentale notitie waarvan index 1 naar de x-positie verwijst, de y-positie als index 2 en index 3 om te leven. PlayerName is een probleem en zou als een afzonderlijke tekenreeks moeten blijven.

BBC BASIC biedt de mogelijkheid om een eigen 'super' variabele te maken die al deze variabelen zal groeperen op één plaats zodat u ze gemakkelijker bij kunt houden. Dit systeem wordt een structuur genoemd. Om een structuur te declareren, gebruikt u DIM, zoals een array, maar nu met accolades in plaats van normale ronde haken. BASIC weet dan dat we een structuur bedoelen. Geef de namen van de variabelen op die u wenst in de groep binnen de accolades. Er zijn gebruikelijke regels voor de naamgeving van toepassing. Laten we eens naar een voorbeeld kijken:

```
DIM Player{X%,Y%,Lives%,Name$}
```

We kunnen toegang tot een van de variabelen krijgen en normale dingen met ze doen. De structuur alleen fungeert als een groeperingsmechanisme. Met behulp van de naam van de structuur krijgt u toegang, met Player in ons geval, gevolgd door een punt en de naam van de variabele die we willen. Als u het aantal levens wilt wijzigen wanneer een speler geraakt wordt, zou u dit kunnen doen:

```
Player.Lives%=Player.Lives%-1
IF Player.Lives%<=0 THEN PRINT "GAME OVER"
```

De structuur is:

```
StructureName.MemberName
```

Nu kunt u kleine eenheden declareren en al uw variabelen in logische groepen houden. Maar er is meer en u wist waarschijnlijk al dat ik dat zou zeggen, niet? De leden van een structuur zijn niet beperkt tot enkele variabelen. U kunt een array als een lid van een structuur hebben. Stel dat uw speler maximaal 10 objecten van apparatuur zou moeten hebben: onderwater vlammenwerper, anti-zwaartekracht laarzen, enzovoort. U moet een manier verzinnen om hem elk van deze items op te laten pikken. Laten we eens een array genaamd Item% nemen. Hieronder ziet u hoe de structuur eruit zal zien:

```
DIM Player{X%,Y%,Lives%,Name$, Item%(10)}
```

De toegang werkt net zo als daarvoor:

```
IF Player.Item%(5)=TRUE THEN
  PRINT "You have the 1000 league boots"
ENDIF
```

Wat er nu komt kan even wat moeilijker zijn, want in een structuur kunnen we nog veel meer doen. U kunt een andere structuur hebben als een lid van de structuur. Het is bijvoorbeeld gebruikelijk (voornamelijk in Windows programma's) om een structuur te houden voor de X- en Y-coördinaten. Dat kunt u doen als zo:

```
DIM Player{Posn{X%,Y%},Lives%, \
\           Name$, Item%(10)}
```

Posn (voor positie) wordt aangeduid als een substructuur en de toegang tot de onderdelen gaat als volgt:

```
Player.Posn.X% = Player.Posn.X% + 5
```

Substructuren kunnen andere substructuren bevatten, de arrays en de variabelen in elke gewenste combinatie. Het idee is natuurlijk hier om de gegevens op te splitsen in logische, gemakkelijk te manipuleren groepen, niet om te laten zien hoe slim je bent door 10 niveaus van gegevens binnen 10 niveaus van gegevens te nesten.

Arrays van structuren

Wanneer onze speler het einde van het spel heeft bereikt, zegevierend en bedekt met gore, dan zouden wij toe willen voegen om een hoge score te laten indienen zodat hij tegen zijn vrienden kan opscheppen. De tabel zal uit de 10 hoogste scores bestaan en tonen. Hieronder zal het duidelijk zijn hoe dit wordt gedaan:

```
DIM Table{Name$,Score%}
```

Mooi, maar dat alleen draagt zorg voor de bovenste positie; we willen een tabel. Hieronder kunt u zien hoe een array van structuren eruit zal zien:

```
DIM Table{(10) Name$,Score%}
```

Het eerste item in de variabelen lijst is het aantal elementen tussen ronde haakjes. Dit geeft ons een array genaamd Table met 11 structuren, 0 tot en met 10. Om de inhoud van de hoge scorelijst te kunnen weergeven, verkrijgen we toegang tot de matrix als volgt:

```
FOR I% = 1 TO 10
  PRINT I%,Table{(I%)}.Name$;
  PRINT Table{(I%)}.Score%
NEXT I%
```

Door de matrixindex tussen accolades te plaatsen, vertelt het BASIC dat u werkt met een structuur. Net als bij normale arrays zijn multidimensionale arrays ook toegestaan.

Prototype structuren

Terug in onze hypothetische spel hebben we een structuur te houden van de aliens: huidige positie, gezondheidspunten, enzovoort. Aan het eind van de level is er een alien-baas die verslagen moet worden voordat men door naar het volgende level kan. Deze baas heeft dezelfde kenmerken als een gewone alien, maar het heeft niet echt zin om hem met de achterban te laten maken. We moeten een andere structuur kunnen maken die voor hem dezelfde kenmerken heeft als zijn minderen, maar gescheiden is, zodat we hem als een uniek geval kunnen behandelen. In BBC BASIC kunnen we een reeds gedefinieerde structuur gebruiken als de sjabloon of prototype voor een ander.

```
REM Prototype structures
DIM Alien{XPosn%, YPosn%, Health%}

REM Setup our alien
Alien.XPosn%=50
Alien.YPosn%=627
Alien.Health%=100

DIM BigBadBoss{}=Alien{}
REM Let's play ...
```

BigBadBoss heeft nu de members:

```
BigBadBoss.XPosn%
BigBadBoss.YPosn%
BigBadBoss.Health%
```

Verwar dit echter niet. Op het punt waar de structuur wordt gedeclareerd, staan alle leden van BigBadBoss ingesteld op nul. Alleen de namen van de structuurleden zijn overgenomen, niet hun werkelijke waarden.

Met behulp van deze techniek is het ook mogelijk om een array van de opgegeven structuur te declareren.

```
REM Arrays of prototype structures
DIM Alien{XPosn%, YPosn%, Health%}

REM Declare a full level of aliens
DIM LotsOfAliens{(20)}=Alien{}
LotsOfAliens{(0)}.XPosn%=50
LotsOfAliens{(0)}.YPosn%=627
LotsOfAliens{(0)}.Health%=100

REM ...
```

We hadden eerder een structuur die we Player hadden genoemd met een geneste structuur Posn genoemd. Deze techniek kan ook worden toegepast op prototypen. Als voorbeeld, we kunnen de positie van onze alien met behulp van XPosn% en YPosn% definiëren, maar we konden dit ook maken in een structuur. Op die manier kunnen we telkens deze velden gebruiken als we elke keer een positie wilde vragen die we nodig hadden, en een prototype geeft ons een kant en klare. Dit is handig, want dat betekent dat we altijd toegang kunnen krijgen tot de positie, op dezelfde wijze waar het is gevonden. Hieronder ziet u hoe we dat kunnen doen:

```
REM Nested prototype structures
DIM Position{X%,Y%}
DIM Alien{Posn{}=Position{}, Health%}
DIM Player{Posn{}=Position{}, \
\           Lives%, Name$, Item%(10)}

REM Initialize our alien
Alien.Posn.X%=50
Alien.Posn.Y%=627
Alien.Health%=100

REM Initialize our player
Player.Posn.X%=50
Player.Posn.Y%=627
Player.Lives%=100

REM ...
```

In de volgende BASIC Bulletin ga ik verder met BBC BASIC. Dan ga ik verder met het gebruik van arrays en structuren, maar ook over het gebruik van routines. BBC BASIC kent PROC en FN statements zoals andere BASIC dialecten SUB en FUNCTION kennen.

Grafisch programmeren in Basic.

In de tijd dat we de programma's behandelden vanaf nummer 12, konden we ontdekken dat het tekenen van bloemen niet zo moeilijk is. Door een kleine wijziging aan een formule aan te brengen, heb je gelijk weer een andere bloem.

Laten we het hoofdstuk beknopt nog eens bekijken. Hieronder ziet u Programma 12 die een blad na elke 45º opnieuw tekent.

```
100 'progr.12  GRAFIEK VAN DE FUNCTIE R*COS(4*PI)
110 CLEAR ,19202 : SCREEN 105,,3,3
120 DEF FNX(X)=INT(1.55*(50+X)+.5)
130 CLS: KEY OFF
140 U=160 : V=160 : H=.5 : RD=4*ATN(1)/180
150 K=160 : P=0 : GOSUB 1000
160 X1=INT(U+K*R*COS(P)+H):Y1=INT(V-K*R*SIN(P)+H)
170 FOR W=1 TO 360 STEP 2
180       P=W*RD : GOSUB 1000
190       X2=INT(U+K*R*COS(P)+H)
200       Y2=INT(V-K*R*SIN(P)+H)
210       LINE (FNX(X1),Y1) - (FNX(X2),Y2),1
220       X1=X2 : Y1=Y2
230 NEXT W
240 A$=INKEY$: IF A$="" THEN 240
250 CLS: KEY ON: END
1000 R=COS(4*P)
1010 RETURN
```

Het programma tekent een achtdelig draai symmetrisch figuur; een bloem met acht blaadjes. Als u met dit programma gaat experimenteren zult u snel ontdekken dat voor even waarden van n een 2n-bladerige en voor oneven waarden van n een n-bladerige bloem ontstaat.

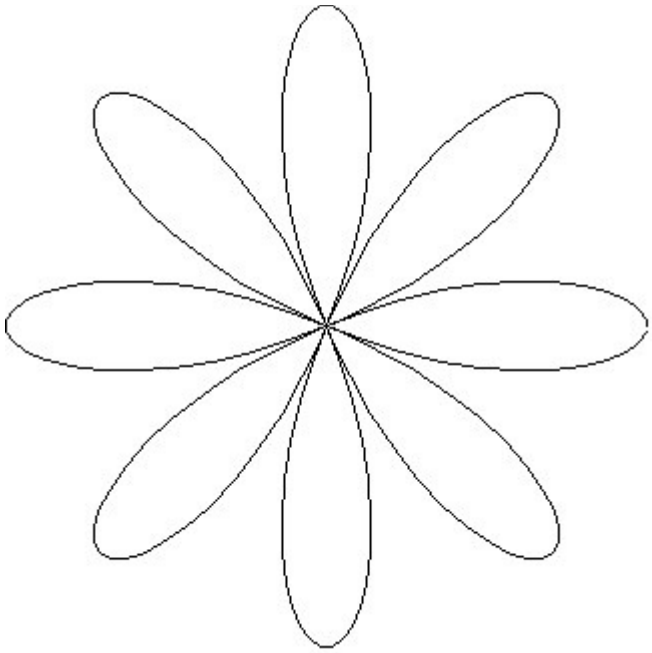
Wilt u dat uw computer sneller tekent? Neem dan een stapgrootte van 3º of van 5º in plaats van 2º. Bedenk dan wel dat de blaadjes wat hoekiger worden.

! **Onthoud dat bij elke andere resolutie de kwaliteit van de blaadjes anders zijn. Hoe hoger de resolutie hoe minder hoekiger de blaadjes worden.**

Hieronder ziet u de code in Visual Basic .NET met een voorbeeld.

```
Public Class frmProg12
    Private Function RCos4PI(ByVal P As Single) As Single
        Return Math.Cos(4 * P)
    End Function
    Private Sub frmProg12_Paint(…, ByVal e As ….PaintEventArgs) Handles Me.Paint
        Const U As Integer = 160
        Const V As Integer = 160
        Const K As Integer = 160
        Const H As Single = 0.5
        Dim RD As Single = Math.PI / 180
        Dim P As Single = 0
        Dim R As Single = RCos4PI(P)
        Dim X1 As Integer = Int(U + K * R * Math.Cos(P) + H)
```

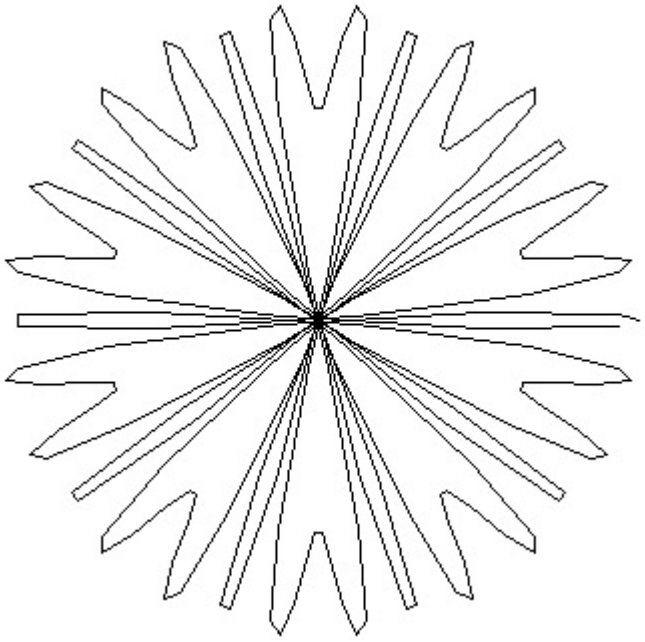
```
        Dim Y1 As Integer = Int(V - K * R * Math.Sin(P) + H)
        For W As Integer = 1 To 360 Step 2
            P = W * RD : R = RCos4PI(P)
            Dim X2 As Integer = Int(U + K * R * Math.Cos(P) + H)
            Dim Y2 As Integer = Int(V - K * R * Math.Sin(P) + H)
            e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
            X1 = X2 : Y1 = Y2
        Next
    End Sub
End Class
```



Verander eens de formule in de functie in onderstaande formule:

```
Math.Cos(4 * Math.Sin(5 * P))
```

Het effect begint dan steeds meer op Mandala te lijken.



Merk op dat aan de rechterkant het nog open is en niet is dichtgetekend. Bij de rest van de blaadjes wel.

Dit geeft de conclusie dat we ontzettend veel kunnen doen met de Math functies Cos en Sin, vooral als we ze samen gebruiken.

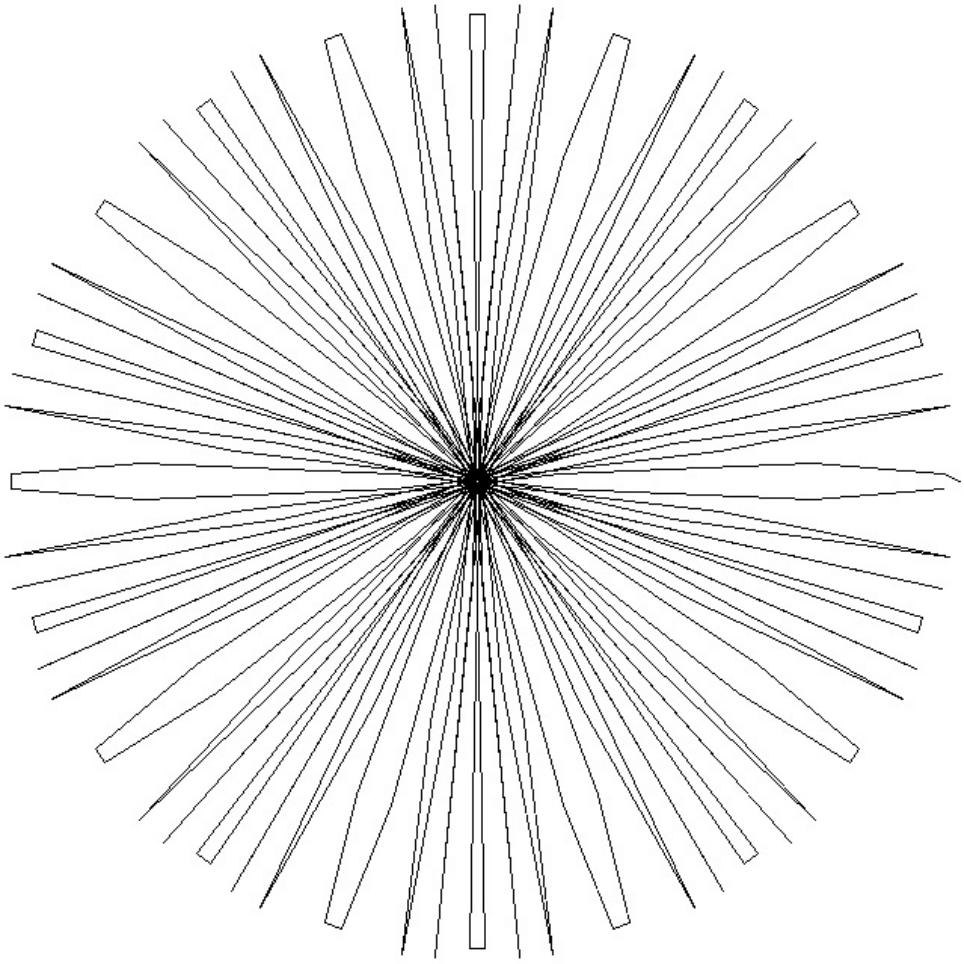
Maar wat doen de andere Math functies eigenlijk? Er zijn nog meer dan alleen maar Cos en Sin.

Hieronder ziet u een afbeelding getekend met onderstaande formule:

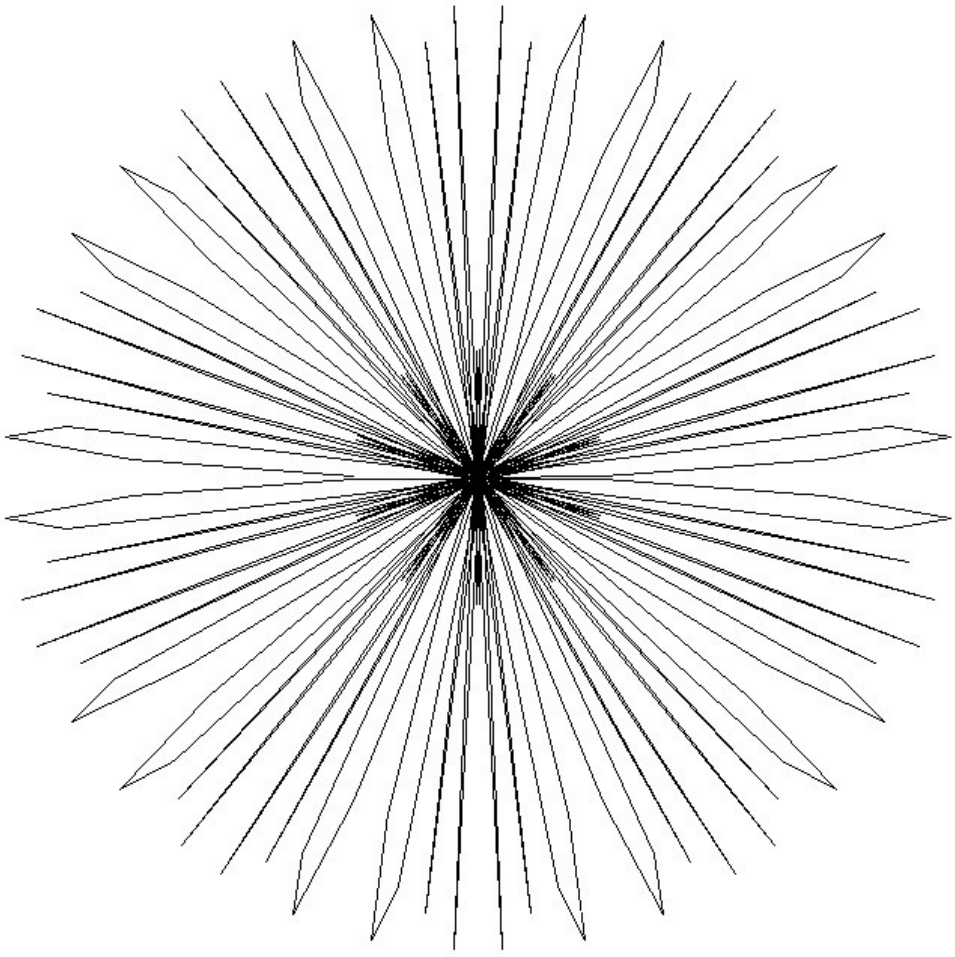
```
Math.Cos(3 * Math.Tan(5 * P))
```

Verander ook alle constanten van 160 waarden in 320 waarden, anders zal de tekening niet goed op het formulier terecht komen. Bovendien zal dan de tekening groter worden en duidelijker te zien zijn.

Kijk eens goed naar het midden, daar kunt u een extra cirkelachtige lijn vinden. Zoiets als we wel eens om de zon kunnen zien, een soort ‘halo’. De halo zal beter te zien zijn wanneer we de 320 waarden veranderen in wat lagere waarden. Hoe hoger de waarden, hoe meer de halo uit elkaar getrokken zal worden.



Veranderen we de Cos functie in de Sin functie, dan zal de tekening veranderen in:



Dit zal ervoor zorgen dat we de ‘halo’ in het midden niet meer te zien krijgen.

Zouden we alleen de waarde van K veranderen in 160 dan zal de tekening kleiner zijn, maar dan wel op de goede plaats op het formulier.

Wat gebeurt er als we K als een binnenlus zouden gebruiken? We zullen dan een bouweffect krijgen, een animatie die het tekenen laat zien. Plaats onderstaande code eens in de Paint event en voer hem uit. Deze keer zonder afbeelding, want nu zal het tekenen een beweging geven.

```
Const U As Integer = 320
Const V As Integer = 320
'Const K As Integer = 320
Const H As Single = 0.5
Dim RD As Single = Math.PI / 180
Dim P As Single = 0
Dim R As Single = RCos4PI(P)
Dim X1 As Integer = Int(U + 160 * R * Math.Cos(P) + H)
Dim Y1 As Integer = Int(V - 160 * R * Math.Sin(P) + H)
For W As Integer = 1 To 360 Step 2
    For K As Integer = 162 To 320 Step 2
        P = W * RD : R = RCos4PI(P)
        Dim X2 As Integer = Int(U + K * R * Math.Cos(P) + H)
        Dim Y2 As Integer = Int(V - K * R * Math.Sin(P) + H)
        e.Graphics.DrawLine(Pens.Black, X1, Y1, X2, Y2)
        X1 = X2 : Y1 = Y2
    Next
Next
```

U kunt gewoon weer de formules gebruiken die behandeld zijn, ook die zullen correct werken met de binnenlus.

Haal meer eruit dan erin zit! Dat is eigenlijk de conclusie met wat we hiermee kunnen doen. Door op een slimme manier de formules te gebruiken en de lusstructuren zo te laten lopen dat het precies zo uitkomt als de bedoeling is, kunnen er tientallen soorten tekeningen worden ontworpen. Van leuke afbeeldingskaarten tot grote mooie mandala tekeningen. Dat laatste vergt wat meer wiskundige technieken, maar door veel te experimenteren kunt u ver komen.

De laatste bovenstaande voorbeelden kunnen ook in GW-BASIC worden uitgevoerd. Pas echter op als u het laatste voorbeeld met de binnenlus wilt uitvoeren. Het kan zijn dat het lopen naar 320 teveel is en het niet goed getekend zal worden.

Excel leren – Celverwijzingen.

Wat zijn celverwijzingen? Dit is een vraag die vaak gesteld wordt als ik het daarover heb. Is het een programmeertechniek? Nee, dat niet. Het is een manier dat gemaakt is door Microsoft in Excel.

Vaak kopiëren we cellen om ze in andere cellen te plakken. Het nadeel is dat ze onafhankelijk van elkaar zijn. Dat betekent dat bij het wijzigen van een waarde in de cel die gekopieerd is, de waarden in de geplakte cellen ongewijzigd blijven.

Stel dat u een blad wilt maken met een maandoverzicht van alle andere bladen die uit maandbladen bestaan. Wordt er in de maand mei een waarde veranderd, dan moet die waarde ook in het maandoverzicht te zien zijn. Om dan telkens de gewijzigde waarden te kopiëren en te plakken, zal het wer-

ken met een jaar-map een ramp worden, vooral als er een totaal in het overzicht berekend moet worden.

Dit is op te lossen door de cellen niet te kopiëren, maar te laten *verwijzen*, waardoor alle wijzigingen op de juiste manier naar het maandoverzicht doorgevoerd worden.

Normaal zijn we gewend om waarden in de cellen te typen, maar we kunnen ook de cellen opgeven. Dit doen we voorafgaand door een **=** operator.

- Open een nieuwe map in Excel en blijf in Blad1.
- Typ in cel **A1** de waarde **20**.
- Typ in cel **B1** de **=** operator met daarachter de cel **A1**. Merk op dat Excel gelijk op de cel A1 reageert door deze een dikke lijn te geven met een opmaakkleur. Die kleur zal bij elke celverwijzing anders zijn mochten er meerdere cellen in een expressie voorkomen.
- Druk op de Enter toets en u zult de waarde 20 zien.

In plaats van de cellen in te toetsen, kunnen we ook op de cel klikken. We moeten dan wel daarna op de Enter toets drukken om de verwijzing te accepteren. Vergeet u om op de Enter toets te drukken en klikt u per ongeluk op een andere cel, zelfs misschien op een cel van een andere map in een ander blad, dan zal de gekozen cel meteen gewijzigd worden. Begrijp goed dat u alleen met de Enter toets de verwijzing kunt beëindigen.

Zoals ik net verteld heb, kunt u inderdaad in een andere map zijn terwijl u met uw eigen map bezig was. Dat kan grote gevolgen hebben met uw eigen map. Verwijzingen kunnen verkeerde waarden geven of zelfs niet meer werken. Het is daarom een goed idee om eerst te bepalen welke cellen u wilt verwijzen en wanneer u op de Enter toets wilt drukken om de verwijzing te accepteren.

Cellen verwijzen vanuit een ander blad

Wilt u ook de waarde in cel A1 in Blad2 hebben? Dat kan, doe onderstaande stappen:

- Ga naar **Blad2** en typ in cel **A1** de **=** in.
- Ga naar **Blad1** en klik met de linker muisknop op cel **A1**. Een kartelrand verschijnt.
- In de invoerbalk ziet u staan: **=Blad1!A1**. Wat het uitroepteken betekent vertel ik hierna.
- Misschien hebt u de neiging terug te gaan naar Blad2, omdat de verwijzing klaar is. De kartelrand om de cel vertelt u echter dat het nog doorgevoerd (geaccepteerd) moet worden.
- Druk op de Enter toets. U keert nu meteen terug naar Blad2 en de cel waar u de verwijzing ingevoerd hebt.

Het uitroepteken !

Dankzij het uitroepteken kunnen we naar elk blad verwijzen die we maar willen. Denk er wel aan deze te gebruiken als u direct wilt verwijzen door het in te voeren en niet de muis te gebruiken.

Stel, we willen onderstaande formule invoeren:

```
=Blad1!A1+40
```

Als u dit vanuit een ander blad wilt doen en u verwijst het met de muis, dan hebt u een probleem. Namelijk dat u nimmer meer de waarde 40 achter de verwijzing in kunt geven. U kunt het wel in de invoerbalk intypen, maar dan zal de verwijzing gestopt worden en zal het resultaat van de formule in Blad1 terecht komen en niet in Blad2.

De oplossing is om bovenstaande regel gewoon in de cel in Blad2 in te voeren. Het resultaat zal dan goed gaan en 60 wordt weergegeven.

Een andere oplossing is toch met de muis, maar u moet dan eerst de verwijzing accepteren voordat u **+40** erachter typt. Na u op de Enter toets hebt gedrukt en terugkeert naar de cel in Blad2, moet u in dezelfde cel weer klikken. In de invoerbalk kunt u dan **+40** erachter typen.

Let op! **Als u dubbelklikt, dan zal de waarde veranderen in de formule. U hoeft dan niet per se deze te wijzigen in de invoerbalk. Hoewel die manier beter en overzichtelijker is.**

Cellen verwijzen vanuit een andere map

Zoals ik eerder verteld heb, kunnen we ook cellen verwijzen vanuit een andere map.

Open een nieuwe map en zorg ervoor dat u beide mappen kunt zien.

In Map2 gaan we een verwijzing maken van cel A1 die in Blad2 staat in Map1.
Activeer cel **A1** in **Map2** door erop te klikken.

Typ weer de **=** in en ga met de muis naar **Map1** in **Blad2** en klik op cel **A1.** U ziet onderstaande verwijzing verschijnen in de invoerbalk:

```
=[Map1]Blad2!$A$1
```

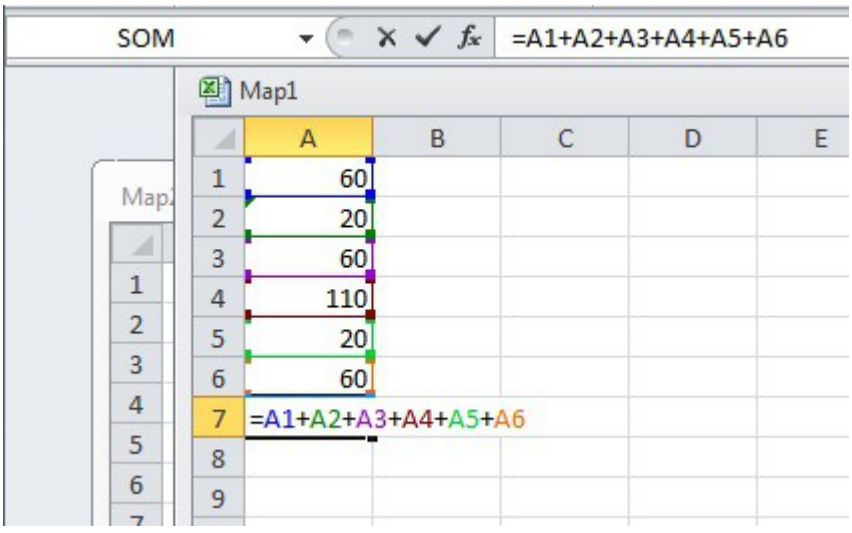
De rechte haken [ ] en het dollarteken $
Wanneer u op Enter drukt keert u weer terug naar **Map2**. De rechte haken zijn nodig om een andere map aan te geven. Maakt u een verwijzing in dezelfde map, dan verschijnt de map niet.

Merk op dat tussen kolom A en rij 1, dollartekens staan. Deze zijn optioneel, maar bij mapverwijzingen worden ze toch tussengeplaatst. Zou u ze weghalen, dan geeft dat geen problemen. Maakt u zelf de verwijzingen zonder de muis, dan hoeft u ze dus niet te gebruiken.

Meer verwijzingen maken

Zoals u hebt kunnen zien, kunt u een waarde achter een verwijzing invoeren. In plaats van een waarde kunt u nog meer verwijzingen in één formule maken. De resultaten kunnen berekend worden en tekstresultaten worden samengevoegd.

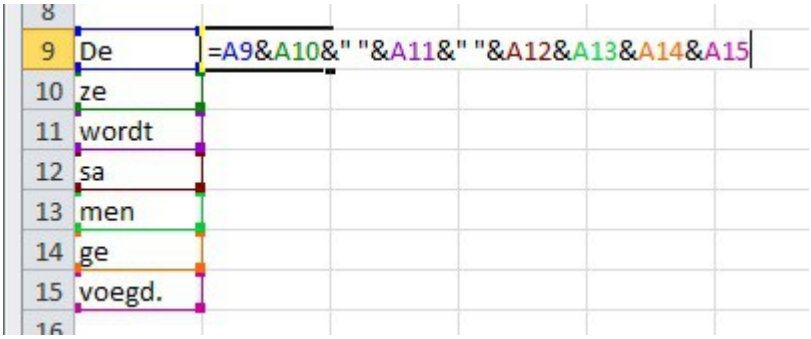
Probeer eens onderstaande voorbeelden na te maken.



Toets de waarden in de cellen vanaf rij 1 tot en met rij 6. In rij 7 typt u de **=** en klik met de muis op cel **A1**. Omdat u in hetzelfde blad blijft, kunt u nu wel daarna op de **+** drukken zonder dat de verwijzing ermee stopt. Klik nu op cel **A2** en u herhaalt dit totdat u cel **A6** gedaan hebt.

U ziet dat elke cel zijn eigen kleur heeft. Dit zal wel verdwijnen zodra u op Enter drukt.

We kunnen niet rekenen met tekst, zoals we in Basic ook niet kunnen. Excel kent dezelfde operator als Basic kent, de ampersand '&'.



Doe deze formule in cel **B9** precies zo als bij de bovenstaande, maar gebruik nu de ampersand. U kunt de ampersand tussen twee spaties plaatsen dat ik niet gedaan heb. Dat maakt namelijk niet uit. Maar wel tussen de aanhalingstekens, want anders zullen de woorden aan elkaar geplaatst worden.

Onderstaand voorbeeld geeft het resultaat van het samenvoegen.

Na op Enter te hebben gedrukt, zal de zin verschijnen alsof het zo in de cel staat. Dat is dus niet zo, want in de invoerbalk zullen we nog steeds de samenvoegformule zien.

U kunt hiermee veel experimenteren. Pas wel op voor onderstaande melding:

`#WAARDE!`

Krijgt u zo'n melding in een cel, dan kan dat verschillende oorzaken hebben:

U hebt geprobeerd een tekst op te tellen met een getal.
U probeerde te delen door 0.
Een map of blad bestaat niet meer.
….

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, woendag en vrijdag in buurthuis Bronveld in Barneveld van 19:00 uur tot 21:00 uur op de dinsdag en van 9:00 uur tot 11:00 uur op de woensdag en vrijdag. 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 19:15 uur tot 21:15 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					
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