Basic Bulletin

23ste jaargang juni 2016 Nummer 2

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	06-30896598	m.a.kurvers@live.nl
Ledenadministratie	Fred Luchsinger	0318-571187	f.luchsinger@kader.hcc.nl
Webmaster	Jan van der Linden	071-3413679	j.vd.linden@kader.hcc.nl

http://www.basic.hcc.nl

Redactioneel

In dit onderwerp van XNA Game Studio laat ik zien hoe we een paddle kunnen laten bewegen door-middel van de pijltjestoetsen of de muis. Dit is de *besturingscontrole*. Als een klap op de vuurpijl zal de bal nu ook een grote rol spelen. Door een *intersectie* uit te voeren, kan de bal stuiteren op de paddle.

BASIC is een geschikte taal om te leren. Waarom? Alleen omdat de letter met *Beginners* begint? Natuurlijk niet, want u kiest welke programmeertaal u wilt gebruiken.

Marco Kurvers

BBC BASIC for Windows – Aan de slag.

Na de meest gebruikte library's van BBC BASIC for Windows te hebben gezien, gaan we eens kijken wat we in de editor kunnen programmeren. In de tijd van BBC BASIC, toen er nog geen Windows was, was het dialect een lelijk eendje, namelijk met statements die in geen ander BASIC dialect te vinden is. We spreken nog steeds van hetzelfde dialect, met uitbreiding voor Windows.

U hebt gezien dat BBC BASIC een gestructureerd BASIC dialect is zonder IDE. De editor kan als een directe modus dienen, maar dat betekent niet dat u regelnummers moet gebruiken als u een programma wilt schrijven.

Start BBC BASIC for Windows en typ eens een paar PRINT statements. Klik op de knop **Run** en uw programma wordt uitgevoerd. Laten we eens nader bekijken wat we in de editor allemaal in kunnen voeren.

```
PRINT 52
```

Als we dit starten, opent Windows een nieuw scherm en wordt het getal op het scherm geprint. Het scherm sluiten we zelf door rechtsboven op het kruisje te klikken.

BBC BASIC for Windows kent alle mogelijkheden wat andere programmeertalen ook kennen, zoals expressies, condities en functies, maar er zijn ook wat extraatjes.

We weten dus dat PRINT alles op het scherm zet wat erachter staat, zo ook tekstuitvoer.

```
PRINT "50+2"
```

Geeft weer:

```
50+2
```

PRINT is een flexibel statement, zoals:

```
PRINT "50*2 = ";50*2

50*2 = 100
```

Een tekst en een expressie moet gescheiden worden door een puntkomma (;) of een komma (,). Een komma geeft meer ruimte.

```
PRINT "50*2 = ",50*2

50*2 =          100
```

Een enkele PRINT statement geeft een lege regel. Normaal geven we drie PRINT statements voor drie lege regels. BBC BASIC kent een kortere mogelijkheid door gebruik te maken van een apostrof (**'**). Om drie lege regels te maken kunnen we typen:

```
PRINT '''
```

Een andere bekende statement dat we samen met PRINT gebruiken is TAB. We kennen TAB door een afstand te kiezen tussen de teksten of een afstand vanaf de linkerkant te kiezen en dan tekst te printen. Onderstaand voorbeeld kennen we wel allemaal:

```
PRINT TAB(10);"Hallo"
```

maar deze niet:

```
PRINT TAB(10,2);"Hallo"
```

Voor sommige BASIC dialecten kunnen we dit vergelijken met onderstaande regel:

```
LOCATE 2,10:PRINT "Hallo"
```

De TAB werkt in BBC BASIC met een *x* en *y* locatiepunt; LOCATE bepaalt de locatiepunt andersom.

Variabelennamen

Variabelennamen kunnen elke gewenste lengte zijn, alleen beperkt door de maximumlengte van een regel, en alle karakters zijn belangrijk. Ze moeten beginnen met een letter (A..Z, a..z), een onderstreping (_) of een ` teken (CHR$96). De resterende karakters moeten zijn A..Z, a..z, 0..9, onderstreping of ` (CHR$96). Variabelennamen zijn hoofdlettergevoelig (‘nummer’, ‘Nummer’ en ‘NUMMER’ zijn verschillende variabelen). Variabelennamen mogen niet beginnen met een BASIC sleutelwoord (met enkele uitzonderingen).

Met behulp van lange, zinvolle, variabelennamen, zal uw programma gemakkelijker te begrijpen en te debuggen zijn, en zal geen invloed op prestaties hebben zo lang als het programma is gecompileerd met de **abbreviate names crunch** optie (voor het instellen van afgekorte namen).

Naamgevingsconventies

Bij het benoemen van variabelen raad ik u aan een van de volgende conventies aan te nemen:

- Lokale variabelen (dat zijn variabelen met alleen een beperkt bereik, formele parameters van functies en procedures en variabelen in een LOCAL of PRIVATE statement) moeten volledig in kleine letters, behalve voor de statische integer variabelen. Bijvoorbeeld `number%`, `text$`.
- Globale variabelen (dat zijn gedeclareerde variabelen over uw hele programma en mogelijk toegang in functies of procedures) moeten een mix zijn van kleine letters en hoofdletters. Bijvoorbeeld `Click%`, `Title$`.
- Constanten (dat zijn variabelen die ingesteld zijn met een vaste waarde tijdens de initialisatie en nergens meer gewijzigd worden) moeten alleen in hoofdletters. Bijvoorbeeld `MAXFILES`, `VERSION$`.

Aanleiding van deze aanbevelingen zal uw programma's gemakkelijker maken om te begrijpen en te voorkomen dat problemen ontstaan, zoals een interrupt routine ten onrechte toegang tot een lokale variabele heeft in plaats van een globale variabele.

Variabelentypes

BBC BASIC for Windows gebruikt de volgende types:

- Statische variabelen
- Integer numerieke variabelen
- Byte numerieke variabelen
- Variant numerieke variabelen
- String variabelen
- Arrays
- Structuren
- Pseudo-variabelen
- Systeemvariabelen

Let op dat in BBC BASIC de verschillende variabelentypes compleet onafhankelijk zijn. Bijvoorbeeld, de integer variabele **list%**, de numerieke variabele **list**, de string variabele **list$** en de array **list%()**, **list()** en **list$()** zijn allemaal volledig gescheiden.

Statische variabelen
De variabelen A%..Z% heten statische variabelen. Het zijn numerieke waarden met gehele waarden die niet gewist worden door de statements RUN, CHAIN of CLEAR. Deze variabelen kunnen als zodanig nuttig zijn voor het overbrengen van waarden tussen programma's of voor het houden van de informatie, die moet worden gehandhaafd wanneer de rest van de variabelen worden gewist. Ze zijn ook sneller toegankelijk dan andere variabelen.

Daarnaast hebben de variabelen A % B % C %, D %, F %, X en Y % speciale toepassingen in de CALL en USR routines, en L, O % tot P % hebben een speciale betekenis in de assembler.

Statische variabelen zijn opgeslagen in 32 bits en kunnen een geheel getal van –2147483648 tot +2147483647 bevatten.

Integer numerieke variabelen
Integer numerieke variabelen hebben namen die eindigen met een procent teken (%). Ze zijn opgeslagen in 32 bits en kunnen een geheel getal van -2147483648 tot +2147483647 bevatten. Het is niet nodig om integer variabelen te declareren voor gebruik van snelle berekeningen, bijvoorbeeld voor de FOR … NEXT lussen voeren een integer snelheid uit al of niet de besturingsvariabele een ‘integer variabele’ (% type) is, zo lang het een integer waarde heeft.

Byte numerieke variabelen
Byte numerieke variabelen hebben namen die eindigen met een ampersand teken (&). Ze zijn opgeslagen in 8 bits en kunnen een geheel cijfer van 0 tot +255 (byte variabelen zijn zonder teken). Byte variabelen zijn nuttig voor structure members, omdat ze gebruikt kunnen worden als bouwstenen voor het maken van gegevensstructuren van elke grootte.

Variant numerieke variabelen
Variant numerieke variabelen kunnen integer of real (floating-point) waarden bevatten. Real waarden zijn in twee vormen, afhankelijk van de *FLOAT mode die effectief is. In *FLOAT 40 mode (de standaard) zijn real getallen opgeslagen in 40 bits (5 bytes) en kunnen ongeveer $\pm 5.9^{E}-39$ tot $\pm 3.4^{E}38$. Het getal bestaat uit een 32-bits mantisse en de exponent van een 8-bit geeft een nauwkeurigheid van ongeveer negen significante cijfers.

In *FLOAT 64 mode zijn real getallen opgeslagen in 64 bits (8 bytes) en kunnen ongeveer $\pm 2.3^{E}-308$ tot $\pm 1.7^{E}308$. Het getal bestaat uit een 53 bits mantisse en met een 11 bit exponent geeft het een precisie van ongeveer 15 significante cijfers.

String variabelen
String variabelen hebben namen die eindigen met een dollar teken ($). Ze kunnen tekenreeksen hebben tot 65535 karakters in lengte.

Arrays
Bij arrays zijn de numerieke integer, byte en variant en string variabelen toegestaan. Alle arrays moeten gedimensioneerd zijn voordat u ze kunt gebruiken. De numerieke integers, bytes, varianten en strings kunnen niet worden gemixt in een multidimensionale array; u kunt maar één array voor elk type variabele gebruiken. Een integer array heeft een naam die eindigt met een procent teken, een byte array met een naam die eindigt met een ampersand en een string array met een naam die eindigt met een dollar teken.

De waarde die in de array declaratie gegeven wordt is het maximum waarde dat de subscript kan hebben. Omdat het minimum waarde van een subscript nul is, zijn het totaal aantal elementen in de dimensie gelijk aan het gegeven waarde plus één. Bijvoorbeeld, de tweedimensionale array gedeclareerd met:

```
DIM marks%(10,5)
```

heeft een totaal van 66 elementen (10+1 rijen bij 5+1 kolommen).

Pseudo variabelen
Pseudo variabelen zijn sleutelwoorden die op variabelen lijken. Deze zijn LOMEM, HIMEM, PAGE, PTR en TIME/TIME$. Er mag mee geschreven worden, bijvoorbeeld TIME = 0, of gelezen worden, bijvoorbeeld T = TIME, afhankelijk van de context. De pseudo variabelen kunnen niet worden gebruikt als formele parameters van functies of procedures. Ze kunnen ook niet lokaal worden gemaakt. Ze kunnen niet worden gebruikt als de besturingsvariabele van een FOR-instructie. Ze kunnen niet worden doorgegeven als een parameter aan CALL en kunnen niet worden toegewezen in de INPUT-, MOUSE-, READ- of SYS statements.

Systeemvariabelen
Systeemvariabelen hebben namen die beginnen met een '@' teken. Systeemvariabelen zijn voor gedefinieerd: u kunt niet uw eigen systeemvariabelen declareren. De bekendste systeemvariabele- en de enige die in de oorspronkelijke versie van BBC BASIC bestaat - is @%. Deze variabele bestuurt de print formattering.

De andere systeemvariabelen zijn als volgt: Ze zijn meestal voor gebruik bij het benaderen van de Windows API-functies van BASIC.

Naam	Waarde
@hwnd%	Het 'venster handle' voor het uitvoervenster van het BASIC programma
@memhdc%	De 'device context' voor het schermgeheugen van BASIC
@prthdc%	De 'device context' voor de huidige printer (als er een is)
@hcsr%	De handle voor de muiscursor
@hpal%	De handle voor het kleurpalet
@msg%	De MSG waarde (voor gebruik met ON MOUSE, ON MOVE en ON SYS)
@wparam%	De WPARAM waarde (voor gebruik met ON MOUSE, ON MOVE en ON SYS)
@lparam%	De LPARAM waarde (voor gebruik met ON MOUSE, ON MOVE en ON SYS)
@midi%	De MIDI device ID (niet-nul als een MIDI bestand aan het spelen is)
@ispal%	Een Boolean waarde die niet-nul is als het display palet is
@hfile%(n)	Een array van bestand handles geïndexeerd door kanaalnummers
@vdu%	Een pointer naar de BASIC tekst en grafische parameters (zie later hieronder)
@cmd$	De commandoregel van een 'gecompileerd' programma
@dir$	De directory (map) van waar uw programma geladen is
@hmdi%	Het Multiple Document Interface venster handle (als er een is)
@flags%	Een geheel getal waarmee de BASIC controle vlaggen worden gebruikt
@lib$	De directory (map) waarin de BBC BASIC library bestanden aanwezig zijn
@ox%	De horizontale offset (in pixels) tussen de uitvoer bitmap en de inhoud van het venster
@oy%	De verticale offset (in pixels) tussen de uitvoer bitmap en de inhoud van het venster
	De volgende variabelen zijn alleen aanwezig in BBC BASIC for Windows versie 5.90a of later:
@hwo%	De handle van het WAVEOUTPUT device
@hevent%	De handle van het event die gebruikt wordt om het blokkeren in bestands- en seriële I/O te voorkomen

@tmp$	De tijdelijke directory (map)
@usr$	Het gebruikers Documents directory (map)
@vdu{}	Een structuur met als inhoud de belangrijkste VDU variabelen
	De volgende variabelen zijn alleen aanwezig in BBC BASIC for Windows versie 5.92a of later:
@haccel%	De handle van het keyboard accelerator, wanneer in gebruik
@hwacc%	Het venster handle van welk keyboard accelerator de commando's verstuurd moeten worden

De @cmd$ variabele
De variabele **@cmd$** kan de toegang tot de opdrachtregel van een uitvoerbaar bestand verkrijgen dat gemaakt wordt met de Compile utility. De variabele @cmd$ is leeg in het geval wanneer een programma vanaf de interactieve omgeving uitgevoerd wordt.

De @dir$ en @lib$ variabelen
De variabelen **@dir$** en **@lib$** zijn nuttig wanneer library's geïnstalleerd zijn of wanneer elk ander *resource* bestand (images, gegevensbestanden enz.) geladen zijn, nodig voor uw programma. Als u er voor zorgt dat deze bestanden opgeslagen zijn, beide in de library directory of in dezelfde directory als het programma zelf, kunt u simpel de bestandsnamen voorvoegen respectievelijk met @lib$ of @dir$:

```
INSTALL @lib$+"MYLIB"
OSCLI "DISPLAY "+@dir$+"MYIMAGE"
SYS "PlaySound", @dir$+"MYSOUND", 0, &20001
```

Bestanden opgegeven op deze manier worden automatisch opgenomen in een uitvoerbaar bestand gemaakt met de Compile opdracht.

De @tmp$ en @usr$ variabelen
De variabelen **@tmp$** en **@usr$** zijn nuttig voor het opslaan van gegevensbestanden. De eerste kan worden gebruikt voor tijdelijke bestanden die kunnen worden verwijderd zodra het programma is voltooid en de tweede voor bestanden met informatie die relevant is voor de huidige gebruiker, zoals opgeslagen output van het programma. In tegenstelling tot @dir$ zijn deze locaties gegarandeerd beschrijfbaar (in normale omstandigheden).

VDU variabelen
Met de variabele @vdu% hebt u toegang tot getalwaarden van interne tekst en grafische parameters van BASIC.

! Omdat de variabele @vdu% en de structuur @vdu{} samen met VDU veel informatie bevat, valt het onderdeel over VDU buiten het bestek van dit onderwerp. In een later Bulletin kom ik met een onderwerp over VDU terug.

Het gebruik van de variabelen
Variabelen kunnen gemaakt en gebruikt worden op verschillende manieren:

- Als resultaat van een toekenningsstatement:

```
Count% = 0
name$ = ""
Value += 1
```

- Door variabelen LOCAL of PRIVATE te maken in een functie of procedure. In dit geval zijn hun waarden geïnitialiseerd naar nul of een lege tekenreeks (in het geval van PRIVATE, alleen bij het eerste gebruik):

```
DEF PROC1 : LOCAL Count%, name$
DEF PROC2 : PRIVATE Count%, name$
```

- Door variabelen te gebruiken als formele parameters in een functie of procedure:

```
DEF PROC1(Count%, name$)
```

- Door ze toe te voegen als een parameterlijst van een CALL statement:

```
CALL code, Count%, name$
```

- Door waarden toe te kennen met een INPUT, MOUSE, READ of SYS statement:

```
INPUT Count%
MOUSE x,y,b
READ name$
SYS "GetTickCount" TO Count%
```

- Ze door laten geven aan een functie of procedure:

```
PROC1(Count%)
DEF PROC1(RETURN C%)
```

- Met gebruik van de address of operator:

```
SYS "GetFileSize", @hfile%(F%), ^sizehigh%
```

In de volgende Bulletins komen uitgebreid de structuurtypen aan bod. Hoe ze werken, hoe we ze zelf kunnen maken en hoe we ze als parameters in functies en procedures door kunnen geven.

XNA Game Studio 4.0 – De basisklasse en de zoonklassen.

Het bouwen van een solution project met het .NET framework en/of met het XNA framework bepaalt niet de structuur van de programmeur. U bent nog altijd degene die bepaalt hoe de applicatie in elkaar gezet moet worden. Dat geldt niet alleen voor de solution; ook de inhoud speelt een grote rol. De vorige keer hebt u gezien dat we sprites kunnen renderen met of zonder objecten. Het is ook niet nodig om meerdere projecten in een solution te maken, alleen maar omdat we de klassen – waar de inhoud van de sprites in worden bewaart – apart willen houden. Het kan best in één project worden gemaakt, als alles maar object geörienteerd blijft.

Een basis sprite klasse maken en gebruiken

Voordat we een klasse gaan toevoegen voor de paddle, wil ik u eerst uitleggen dat het gebruik van meer klassen voor de sprites ook meer ruimte in de solution kost en dus ook meer geheugen kost. Vaak hebben sprite objecten eigenschappen nodig dat in elk object hetzelfde zijn. Elke sprite heeft een texture object nodig om gerendeerd te kunnen worden en de positie die bepaald moet worden. Maar ook vaak de snelheid die met het bewegen op het scherm nodig is, maar die staat echter als laatste prioriteit omdat er ook sprite objecten zijn die op een vaste plaats blijven staan, zoals muren en tegels.

We kunnen natuurlijk de objecten uit de bal klasse kopiëren en in de paddle klasse plakken en zo ook in andere klassen, maar dat is veel werk en kost teveel wielen om uit te vinden. Om zuinig te zijn met de inhoud van de klassen, gaan we alles delen. We maken zoonklassen die de eigenschappen en methoden mogen erven van hun basisklasse.

Hebt u in de solution het tweede project **SpriteLib**, klik dan met de rechter muisknop op dat project. Zo niet, klik dan met de rechter muisknop op het project **VBGame**. Kies **Add -> Class…**. Noem de klasse **SpriteBase** en klik op de knop **Add**.

Onderstaande klasse zal verschijnen:

```
Public Class SpriteBase

End Class
```

Zorg ervoor dat u de klasse in de juiste namespace plaatst, VBGame of SpriteLib. Herinnering: knip eerst de klasse uit, typ het blok Namespace *naam* … End Namespace en plak de klasse in het Namespace blok.

In de klasse moeten we al het inhoud erin zetten waar de zoonklassen gebruik van moeten kunnen maken. Enkele voorbeelden zijn:

- het texture object
- de positie
- breedte en hoogte van het speelvenster
- de snelheid (als de zoonklasse dat nodig heeft)
- het renderen van het texture object

We hebben al een klasse met inhoud: de Bal klasse. Als u nog eens naar bovenstaand lijstje kijkt, dan ziet u dat enkele voorbeelden dus ook aanwezig zijn in de Bal klasse. Maar dat niet alleen. Als we de Paddle klasse straks erbij gaan maken, hebben we ook weer hetzelfde inhoud nodig. We gaan deze inhoud in de SpriteBase klasse maken.

Zorg ervoor dat boven de Namespace de twee Imports regels staan:

```
Imports Microsoft.Xna.Framework
Imports Microsoft.Xna.Framework.Graphics
```

Typ onderstaande Private variabelen in de SpriteBase klasse:

```
Private sTexture As Texture2D
Private pXSpeed As Integer
Private pYSpeed As Integer
Private pMaxWidth As Integer
Private pMaxHeight As Integer
Private pspritePos As Vector2 = New Vector2
```

Om de texture te kunnen gebruiken, maken we een texture eigenschap. Deze eigenschap hebben we nodig om elke sprite te kunnen renderen.

Typ de eigenschap eronder:

```
Public Property spriteTexture() As Texture2D
    Get
        Return sTexture
    End Get
```

```
        Set (value As Texture2D)
            sTexture = value
        End Set
    End Property
```

Voor sommige objecten moet er een snelheid in een bepaalde richting ingesteld kunnen worden. Typ onderstaande twee eigenschappen die voor de X en Y waarden moeten zorgen.

```
    Public Property XSpeed() As Integer
        Get
            Return pXSpeed
        End Get
        Set(value As Integer)
            pXSpeed = value
        End Set
    End Property

    Public Property YSpeed() As Integer
        Get
            Return pYSpeed
        End Get
        Set(value As Integer)
            pYSpeed = value
        End Set
    End Property
```

De volgende twee eigenschappen zijn alleen-lezen. Ze zijn bedoeld om te achterhalen wat de breedte en de hoogte van het ingestelde scherm of van het meegegeven viewport is. Typ onderstaande eigenschappen in.

```
    Public ReadOnly Property MaxWidth() As Integer
        Get
            Return pMaxWidth
        End Get
    End Property

    Public ReadOnly Property MaxHeight() As Integer
        Get
            Return pMaxHeight
        End Get
    End Property
```

Als we de eigenschappen gedaan hebben, hebben we een methode nodig waarmee we de breedte en de hoogte in kunnen stellen. In principe zou u ook alleen de eigenschappen daar voor kunnen gebruiken, door ze zowel lezen als schrijven te maken. Maar dat maakt niet uit.
De laatste eigenschappen hebben wat uitleg nodig, want een Vector2 object in een klasse gebruiken is normaal gesproken geen probleem. Nu het object in de SpriteBase staat, moet er een Vector2 eigenschap gemaakt worden om deze in de zoonklassen te kunnen gebruiken. Typ onderstaand Vector2 eigenschap in.

```
    Public Property spritePos() As Vector2
        Get
            Return pspritePos
        End Get
        Set (value As Vector2)
```

```
            pspritePos = value
        End Set
    End Property
```

In de zoonklassen kunnen we gewoon onderstaande expressie gebruiken die we in de Update() methode in de Bal klasse nodig hebben:

```
spritePos += spriteSpeed * 0.1F
```

Willen we echter de bal positie controleren en andere waarden toekennen, dan hebben we de eigenschappen X en Y van spritePos nodig. Onderstaand If … Then en End If blok zouden we nodig hebben:

```
If (spritePos.X + spriteTexture.Width > MaxWidth) Then
    spritePos.X = MaxWidth - spriteTexture.Height
    spriteSpeed.X *= -1
End If
```

Maar de regel die de expressie MaxWidth – spriteTexture.Height toekent aan spritePos.X zal echter niet werken. Er is gewoonweg geen toegang tot de eigenschappen van de eigenschap spritePos die in de klasse SpriteBase staat. De controle werkt wel, want de eigenschappen van spritePos zijn wel te lezen.

Om het probleem te verhelpen, moeten er twee extra eigenschappen onder de spritePos eigenschap komen die de Vector2 eigenschappen X en Y moeten regelen. Typ onderstaande eigenschappen in.

```
Public Property spritePosX() As Single
    Get
        Return pspritePos.X
    End Get
    Set(value As Single)
        pspritePos.X = value
    End Set
End Property

Public Property spritePosY() As Single
    Get
        Return pspritePos.Y
    End Get
    Set(value As Single)
        pspritePos.Y = value
    End Set
End Property
```

Nu we de eigenschappen in de SpriteBase klasse hebben gemaakt, zijn de methoden aan de beurt. De GameScreen methode moet zorgen dat de juiste breedte en hoogte van het scherm ingesteld kunnen worden. Typ onderstaande methode in.

```
Public Sub GameScreen(ByVal Width As Integer, ByVal Height As Integer)
     pMaxWidth = Width
     pMaxHeight = Height
End Sub
```

In elk object moeten we Draw() aan kunnen roepen. We hebben maar één methode nodig die we vanuit de zoonklassen kunnen gebruiken. Typ onderstaande methode in.

```
Public Sub Draw(ByVal spriteBatch As SpriteBatch)
     spriteBatch.Draw(sTexture, pspritePos, Color.White)
End Sub
```

Het is een gewoonte om in de klassen geen gebruik te maken van de gemaakte eigenschappen, maar alleen van de Private variabelen. Ten eerste zijn de eigenschappen bedoeld voor de zoonklassen en de objectinstanties en ten tweede is het veiliger om er geen gebruik van te maken om fouten te voorkomen. Voorbeeld: u gebruikt per ongeluk een eigenschap om een waarde aan toe te kennen terwijl deze als alleen-lezen is gemaakt. Deze vorm wordt ook wel *polymorfisme* genoemd.

Terug naar de Bal klasse

Veel variabelen en methoden zijn in de Bal klasse niet meer nodig. We moeten de hele klasse doorlopen om aanpassingen te maken. Als eerste zorgen we ervoor dat de Bal klasse van de SpriteBase klasse kan afstammen. Dit wordt gedaan met het **Inherits** sleutelwoord. Typ die precies zo, zoals het hieronder staat.

```
Public Class Bal
     Inherits SpriteBase
     ...
End Class
```

Verwijder alle Private variabelen die onder de Inherits regel staan, maar verwijder niet de **spriteSpeed** variabele, want die moet gewijzigd worden.

Hoewel de XSpeed en YSpeed eigenschappen vanuit de SpriteBase klasse gebruikt worden, hebben we toch een Private spriteSpeed in de Bal klasse nodig. De reden is dat alleen de bal een ingestelde richtingssnelheid moet hebben. Andere klassen, en dus ook de Paddle klasse, hebben dat niet nodig. De paddle zal alleen door middel van de besturing van de gebruiker bewogen worden.

Onderstaande spriteSpeed declaratie moet gewijzigd worden:

```
Private spriteSpeed As New Vector2(XSpeed, YSpeed)
```

in

```
Private spriteSpeed As New Vector2
```

Het instellen van de X en Y richtingssnelheid wordt nu gedaan in de constructor van de Bal klasse.

Hieronder ziet u de constructor, die we ook gaan wijzigen.

```
Public Sub New(ByVal texture As Texture2D, _
               ByVal graphics As GraphicsDeviceManager)
     Me.texture = texture
     Me.graphics = graphics
End Sub
```

Verwijder het graphics argument en de **Me** regels in de constructor en typ onderstaande nieuwe regels in.

```
spriteTexture = texture

'X en Y snelheid van de sprite
XSpeed = 20
YSpeed = 40
spriteSpeed = New Vector2(XSpeed, YSpeed)
spritePos = Vector2.Zero
```

We hadden natuurlijk de waarden 20 en 40 net zo goed in de declaratie van spriteSpeed in kunnen stellen, want de XSpeed en YSpeed eigenschappen hebben we verder niet nodig. Toch kan het handig zijn ze in de SpriteBase klasse te hebben en, misschien als u van plan bent VBGame uit te breiden, kunt u nieuwe methoden tegenkomen die plotseling wel de eigenschappen nodig hebben.

In de Update() methode moet er ook wat aangepast worden, want de code zal nu niet meer werken.

Wijzig in de eerste regel eerst de waarde 0.01F in de waarde 0.1F om de bal wat sneller te laten bewegen. Verder hoeft er met deze regel niets te gebeuren.

Verwijder de twee declaratieregels MaxWidth en MaxHeight. Ze hoeven niet meer lokaal gedeclareerd te worden, omdat we nu gebruik gaan maken van de eigenschappen MaxWidth en MaxHeight die in de SpriteBase klasse staan.

Eerder had ik het nog over het gebruik van een eigenschap met als type Vector2. Het gebruik ervan kan een probleem veroorzaken als we de eigenschap van de gemaakte eigenschap willen gebruiken, zoals X en Y. Daarom staan er in de SpriteBase klasse de twee extra eigenschappen die voor de X en Y eigenschappen moeten zorgen.

Wijzig de **If** … **End If** code door het volgende te doen:

- verwijder de punt die tussen alle **spritePos** en **X** en **Y** eigenschappen staan;
  voorbeeld: spritePos.X wordt spritePosX
- wijzig de **texture** namen in de naam **spriteTexture**.

Als laatste kunnen we de Draw() methode in de Bal klasse verwijderen. Vanuit het bal object kunnen we de Draw() methode gewoon aanroepen, omdat de methode geërfd wordt van de SpriteBase klasse.

Wat we ook nog aan moeten roepen is de methode GameScreen(). De methode staat in de SpriteBase klasse, dus zouden we kunnen zeggen: we roepen hem aan in de constructors van de zoonklassen. Helaas kan dat niet, want de GraphicsDeviceManager werkt alleen in de Game1 klasse. We roepen de methode daarom aan nadat we het bal object gedefinieerd hebben.

Ga naar de Game1 klasse en knip onderstaande code uit de LoadContent() methode.

```
Dim textureStream As IO.Stream = New _
     IO.StreamReader(Application.StartupPath & _ "\Textures\Ball-
Blue.png").BaseStream
'Laad de texture in.
Dim texture As _
     Texture2D = Texture2D.FromStream(GraphicsDevice, textureStream)
bal = New Bal(texture, graphics)
```

Dit codeblok moet geplakt worden in een aparte methode die hieronder staat. Typ de methode in de Game1 klasse.

```
Private Sub CreateBall()

End Sub
```

Plak nu de code in de methode.

Verwijder de **graphics** parameter in de definitie van de bal constructor. De regel moet dan worden:

```
bal = New Bal(texture)
```

In de zoonklassen maken we gebruik van de eigenschappen MaxWidth en MaxHeight. Ze moeten echter nog ingesteld worden. Typ onderstaande regel onder de bal definitie.

```
bal.GameScreen(graphics.GraphicsDevice.Viewport.Width, _
               graphics.GraphicsDevice.Viewport.Height)
```

Typ onderstaande regel in de LoadContent() methode om ervoor te zorgen dat de bal geladen en gemaakt wordt.

```
CreateBall()
```

Dankzij de OOP techniek is het mogelijk om één methode door tientallen objectinstanties aan te laten roepen. Dit bespaart zeer veel programmeerwerk en tijd. Als we verder gaan met de Paddle klasse, zult u zien dat ook die van de SpriteBase zal afstammen.

De Paddle klasse

De Paddle klasse hadden we nog niet. Maak deze aan in het juiste project, VBGame of SpriteLib. In ieder geval, zorg ervoor dat de klasse in het project komt te staan waar ook de klassen SpriteBase en Bal in staan. Let weer op dat de klasse in het Namespace blok staat en de twee XNA Framework Imports regels boven de Namespace staan, dezelfde die ook in de Bal klasse staan. Nu echter hebben we een derde Imports regel nodig voor de besturing van de paddle. Typ de Imports regel bij de andere Imports regels.

```
Imports Microsoft.Xna.Framework.Input
```

Typ in de klasse de Inherits regel om af te kunnen stammen van SpriteBase.

Typ onderstaande constructor in de klasse.

```
Public Sub New(ByVal texture As Texture2D)
     spriteTexture = texture
     spritePos = New Vector2(MaxWidth / 2, MaxHeight - 32)
End Sub
```

De Paddle klasse zal een hele andere Update() inhoud krijgen dan de Update() van de Bal klasse heeft. In de Update() zal de inhoud voornamelijk de besturing zijn en de controle die er voor zorgt dat de paddle binnen het scherm blijft. Typ onderstaande methode in de Paddle klasse.

```
Public Sub Update()

End Sub
```

Voor de besturing van de gebruiker hebben we in Windows twee structures: Keyboard en Mouse. De structures hebben een object GetState() met verschillende eigenschappen. Maken we gebruik van het Keyboard en controleren we de toetsen Z en M voor paddle naar links en paddle naar rechts, dan hebben we een extra variabele nodig die de positie met een bepaalde stap moet verlagen (links) of moet verhogen (rechts). Ik kies de muisbesturing, omdat dan de extra variabele niet nodig is. De muis zorgt zelf voor de bewegingssnelheid van de paddle. Dit doen we door de X waarde van Mouse.Get-State() op te vragen en in een lokale variabele op te slaan. De Y waarde hebben we echter niet nodig. We willen niet met de paddle vliegen, dus moeten we de Y waarde van de paddle vast zetten.

Typ hieronder de code in de Update() methode.

```
Dim mx As Integer = Mouse.GetState().X

spritePosY = MaxHeight - 32
If mx > 0 And mx < (MaxWidth - spriteTexture.Width) Then
     spritePosX = mx
ElseIf mx <= 0 Then
     spritePosX = 0
Else
     spritePosX = (MaxWidth - spriteTexture.Width)
End If
```

We hoeven geen Draw() methode te maken, omdat de Draw() methode vanuit SpriteBase wordt aan-geroepen.

De paddle klasse is klaar, maar we moeten de paddle nog inladen en maken.

Ga naar de Game1 klasse en maak onderstaande methode.

```
Private Sub CreatePaddle()

End Sub
```

Declareer in de klasse boven de constructor onderstaand paddle object.

```
Dim paddle As Paddle
```

Kopieer de inhoud die in de CreateBall() methode staat en plak het in de CreatePaddle() methode. Pas de code aan zoals u het hieronder ziet staan.

```
Dim textureStream As IO.Stream = New _
     IO.StreamReader(Application.StartupPath & _ "\Textures\paddle-
bruin.png").BaseStream
'Laad de texture in.
Dim texture As _
     Texture2D = Texture2D.FromStream(GraphicsDevice, textureStream)
paddle = New Paddle(texture)
paddle.GameScreen(graphics.GraphicsDevice.Viewport.Width, _
                  graphics.GraphicsDevice.Viewport.Height)
```

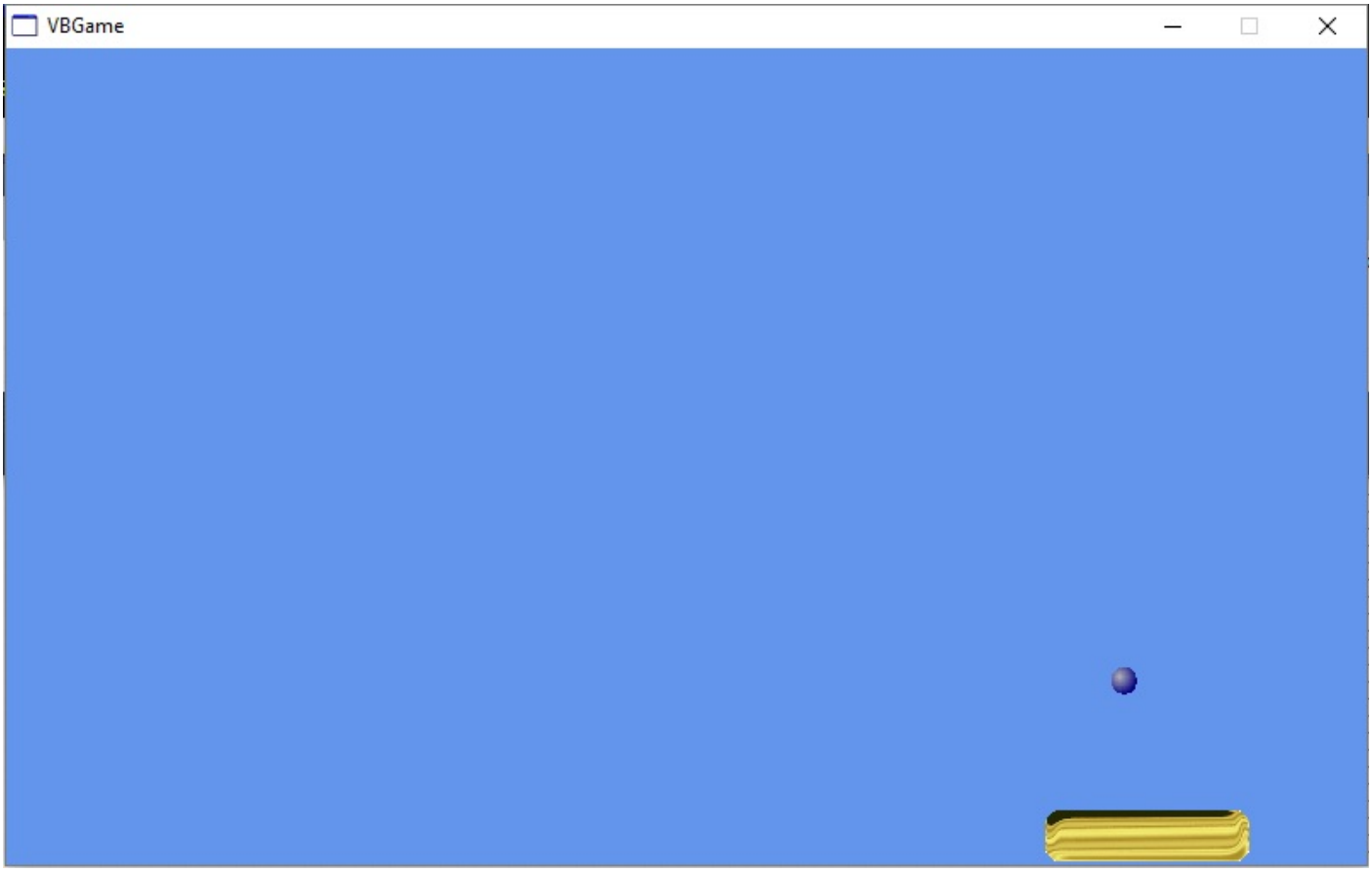
Wijzig de bestandsnaam van de paddle mocht het niet paddlebruin heten.

In de LoadContent roepen we de Update() methode aan. Typ onderstaande regel onder de bal Update() methode.

```
paddle.Update()
```

In de Draw() methode moeten we ook de paddle renderen. Voor de bal is het al in orde. Typ onderstaande regel.

```
paddle.Draw(spriteBatch)
```



Als u de game start, ziet u weer de bal, met de paddle. Beweeg eens de muis. U zult zien dat de paddle op de muis reageert en alleen maar horizontaal kan bewegen. U zult ook zien dat de bal dwars door de paddle kan gaan. We moeten dus actie ondernemen om de bal op de paddle te laten stuiteren. Hoe kunnen we bepalen wanneer de bal de paddle ontmoet, of beter gezegd wanneer ze elkaar overlappen? Gelukkig heeft XNA een oplossing door gebruik te maken van de Intersects() methode in het Rectangle object. We moeten dus wel het een en ander toevoegen in de klassen om de actie echt te laten werken.

De intersection

De intersection controleert op overlapping. Als de rechthoeken van de sprites elkaar overlappen, geeft de Intersects() methode een Waar. De speler heeft niet in de gaten dat het in principe niet om de vorm van de sprites gaat. Elke sprite heeft een rechthoek.

Maar als er gebotst wordt, op welke 'rechthoek' moeten we dan controleren? Hoe kunnen we actie ondernemen zodat we weten dat we de bal weer de andere kant (kaatsend) op kunnen sturen?

De actie moet in de Update() methode in Game1 uitgevoerd worden. De Intersects() is een methode van het Rectangle object. We hebben twee Rectangle objecten nodig, voor de bal en de paddle. Maar dankzij de SpriteBase klasse hoeven we de de rechthoek voor elk object maar één keer te definiëren.

Ga naar de SpriteBase klasse en typ onderstaande functie boven de Draw() methode.

```
Public Function Rect() As Rectangle
     Dim r As Rectangle
     With r
          .X = pspritePos.X
          .Y = pspritePos.Y
          .Width = sTexture.Width
          .Height = sTexture.Height
     End With
     Return r
End Function
```

De functie zal de ingestelde waarden van pspritePos en sTexture eerst toekennen aan de lokale rectangle. Pas dan zal het hele rectangle object **r** geretourneerd worden.

Omdat de spriteSpeed een lokale variabele is van de bal, kunnen we het terugkaatsen van de bal niet in de Update() uitvoeren. We hebben dus een methode nodig.

Ga naar de Bal klasse en typ onderstaande methode.

```
Public Sub Collide()
     spriteSpeed.Y *= -1
End Sub
```

De spriteSpeed kan niets zonder actie. De Intersection moet controleren of de Collide() methode aangeroepen mag worden of niet.

Ga naar de Update() methode in Game1 en typ onderstaande code onder de paddle Update() aanroep.

```
If bal.Rect.Intersects(paddle.Rect()) Then
     bal.Collide()
End If
```

Als u de game nu start en u zorgt ervoor dat de bal op de paddle terecht komt, zult u zien dat de bal in dezelfde horizontale richting, maar verticaal in tegengestelde richting botst. Dit noemen we ook wel een *bounce* effect, een effect dat de gebruiker laat denken alsof de bal echt de paddle raakt!

Het nadeel van de Intersects() methode

Nu we alles zover hebben voor het maken van een goede game, kan er meer actie worden toegevoegd, bijvoorbeeld Bricks. Stenen om de bal ook daar tegen te laten botsen. Dankzij de Intersects() methode kunnen we veel doen en de bal overal op laten reageren. Maar helaas, als we de game nog eens starten en we wat meer met de bal gaan spelen, zult u zien dat er een fout is. Onderstaande afbeelding laat zien wat er gebeurd als u probeert de bal aan de zijkant van de paddle te raken.

De fout wil echter niet zeggen dat het ook echt fout is. De code is goed. Zulke foutjes die tijdens de uitvoering aanwezig zijn, komen het meest voor en zijn het moeilijkst op te sporen.

De oplossing voor het aanraken van elke zijde of hoek is er niet meteen. Met de Intersects() methode kunnen we niet bepalen aan welke zijde of hoek de paddle geraakt is. Kortom, alleen de Collide() methode is niet voldoende.

Er bestaat een techniek dat de *Line Intersection* wordt genoemd. De rechthoeken van de objecten hebben vier zijden en vier hoeken. Aan de hand van welke zijde een zijde raakt, welke zijde een hoek raakt, welke hoek een zijde raakt of welke hoek een hoek raakt, moet de juiste richting van de bal ingesteld worden, voordat de Intersects() methode bepalen kan of er werkelijk een overlapping is.

In de volgende Bulletin zal ik eens wat laten zien over de Line Intersection techniek en kom ik met een algemene samenvatting over XNA Game Studio Framework voor Visual C# en Visual Basic.

Fuseren met andere programmeertalen.

Zoals u weet zijn we samen met de andere programmeergroepen C en Pascal. Nu we gefuseerd zijn tot één programmeergroep, kunnen we het ook over andere programmeertalen hebben.

Tot nu toe heb ik veel laten zien en uitgelegd hoe bepaalde BASIC dialecten eruit zien en hoe de twee grote Microsoft Framework library's werken: .NET en XNA. Ik vond het al eerder belangrijk om ook Visual C# erbij te halen, vanwege de uitleg over het XNA sjabloon en de Content Pipeline.

Samenwerken met leden die andere programmeertalen gebruiken kan nut hebben. Vaak willen we programma's converteren wanneer we een listing in een andere programmeertaal leuk vinden.

Is het een goed idee als wij ons ook even in gaan verdiepen in andere programmeertalen? Hoe zit de structuur in C en in Pascal eruit? Wat is Delphi voor een programmeertaal? Is Lazarus ook een Pascal taal?

Daarom, hieronder een paar programmeertalen met wat uitleg.

Free Pascal
Free Pascal is een console vrije programmeertaal gestructureerd met één listing. Er ontbreekt veel in Free Pascal en er moet gebruik gemaakt worden van **uses** regels om de taal meer functionaliteit te geven.

Het geraamte van de taal ziet er uit zoals hieronder.

```
program Test1;
var
     Naam: string;
begin
     WriteLn('Hallo allemaal!');
     Write('Wat is uw naam: ');
     ReadLn(Naam);
     WriteLn('Uw naam is: ', Naam);
end.
```

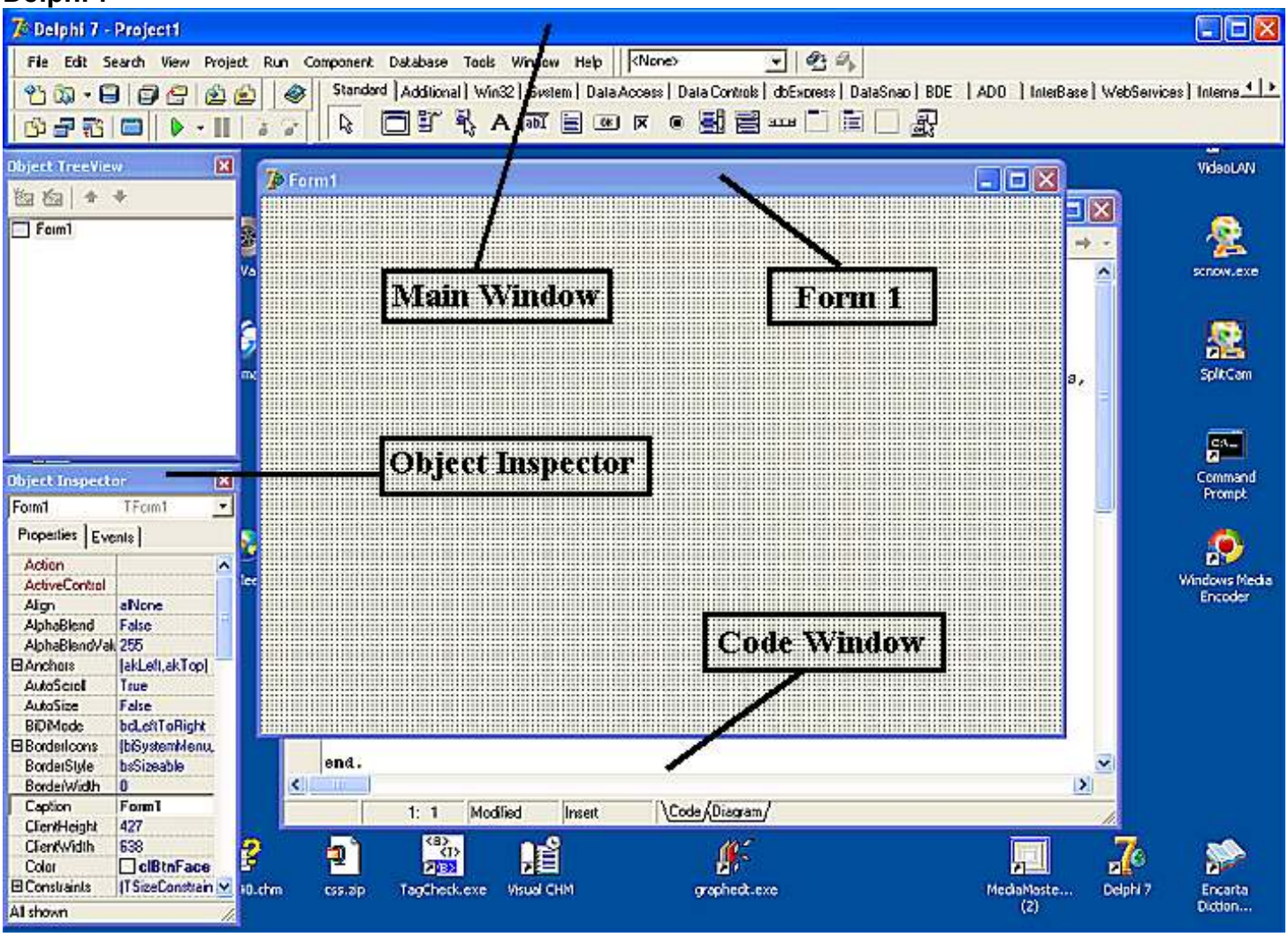
De variabelen kunnen niet in het **begin … end** blok gedeclareerd worden, maar altijd boven **begin**.

Het toekennen van waarden wordt met de **:=** operator gedaan. Voor BASIC gebruikers komt de operator ook bekend voor als een *argumentoperator* in een procedure of in een functie. Meer over deze verschillen, zie het onderwerp 'Structuren' in de volgende Bulletin.

Turbo Pascal
Turbo Pascal werkt op dezelfde manier als Free Pascal, maar ondersteunt het gebruik van klassen. Niet de klassen die wij kennen, want in Turbo Pascal zijn het *Object Occurrences*. Een object occurrence is onbeschermd maar kent wel OOP en polymorfisme.

Lazarus
Deze Pascal taal is een programmeertaal met een menu en losse vensters zonder achtergrond. Het heeft een IDE die niet al te moeilijk is en fijn is te gebruiken. Lazarus is gratis en er kunnen gewoon projecten mee geschreven en gecompileerd worden. Het is ook een Object-achtige Pascal, omdat er met klassen gewerkt kan worden.

Delphi 7



Delphi 7 en Lazarus lijken heel veel op elkaar. Delphi 7 heeft wel veel meer mogelijkheden in de menubalk en in de werkbalken. Net als in Lazarus heeft Delphi 7 geen MDI achtergrondvenster.

C / C++

De programmeertalen C en C++ werden als leertalen veel gebruikt in middelbare scholen. C en C++ zijn net zoals Free Pascal console programmeertalen, zoals onderstaande voorbeelden.

```
// C code
#include <stdio.h>

int main(void)
{
      char naam[100];
      int leeftijd;

      printf("Wat is uw naam? ");
      scanf("%s", naam);
      printf("\nWat is uw leeftijd? ");
      scanf("%d", &leeftijd);
      printf("\nUw naam is: %s", naam);
      printf("\nUw leeftijd is: %d", &leeftijd);
      return 0;
}
```

```
// C++ code
#include <iostream>

int main(void)
{
      char naam[100];
      int leeftijd;

      cout << "Wat is uw naam? ";
      cin >> naam;
      cout << endl << "Wat is uw leeftijd? ";
      cin >> leeftijd;
      cout << endl << "Uw naam is: " << naam << endl;
      cout << "Uw leeftijd is: " << leeftijd;
      return 0;
}
```

Later werd C++ uitgebreid en kwam Borland met nieuwe versies.

Borland C++ versie 3 en versie 5
C++ versie 3 was de eerste16-bit Windows programmeertaal. De Windows Message Loop moest zelf geprogrammeerd worden. Het was veel programmeerwerk om vensters aan te sturen. Versie 5 is een uitbreiding van versie 3 met een IDE.

OWL
Borland bracht met de overgang van 16 bit naar 32 bit een allereerste Object Windows Library uit. We kunnen het een beetje vergelijken met de Framework library. Hiermee wordt alle Windows besturing op de achtergrond gehouden en hoeft de programmeur zich niet meer te bekommeren om de Windows besturing. OWL was vooral geschikt voor 32 bit projecten. Hoewel OWL zeer verouderd is, werkt het nog steeds goed.

Codegear RAD Studio 2009
Codegear RAD Studio zijn pakketten in meerdere versies (2009 is de versie die ik heb) waar we twee programmeertalen en wat ‘spullen’ in kunnen vinden, zoals documentaties en sjablonen. De twee programmeertalen werken visueel, net zoals in Visual Studio, en het zijn Delphi en Borland C++ Builder. De Builder zorgt ervoor om met C++ nog gemakkelijker en sneller projecten te kunnen bouwen. Hoewel Delphi van Codegear RAD Studio heel veel op Delphi 7 lijkt, zal het verschil toch snel te zien zijn. Bijvoorbeeld, het MDI venster die nu wel een achtergrond heeft en de zwevende vensters die nu vastgezet kunnen worden (docking).

Visual Studio en Visual Studio .NET
In de tijd dat 16 bit overging naar 32 bit, was er Visual Studio 4. De programmeertalen bestonden in twee delen om 16 bit projecten over te kunnen zetten in 32 bit. Visual Studio 6 was de laatste versie waarvan alles nog in losse onderdelen werkte en formulieren nog als zwarte dozen werkten.
In het jaar 2003 kwam de eerste Visual Studio .NET met een OOP waarmee alle onderdelen in een bibliotheek werden gestopt. De formulieren werden ook klassen. De bibliotheek wordt nog steeds uitgebreid met de XNA bibliotheek erbij.

Met elkaar samenwerken en elkaar helpen
Nu we met BASIC het ook over andere programmeertalen kunnen hebben, is het een goed idee om samen te werken en elkaar te helpen. Kent u alleen maar BASIC, geen probleem. BASIC blijft ook gewoon doorgaan. In de volgende Bulletin zal ik eens wat structuren laten zien van meerdere programmeertalen en komt Visual C# ook weer terug met XNA.

Liberty BASIC API Reference.

GetSystemDirectory

Deze functie haalt de map van de Windows systeem directory op. De systeem directory heeft als inhoud verschillende soorten bestanden als Windows bibliotheken en font bestanden. Hier is een voorbeeld:

```
WindowsSys$=space$(200)+chr$(0)
calldll #kernel32, "GetSystemDirectoryA", _
     WindowSys$ as ptr, _      'buffer die de teruggegeven string ontvangt
     200 as ulong, _           'buffergrootte
     result as ulong           'groote van teruggegeven string
print "De Windows systeem directory is: "; trim$(WindowsSys$)
```

GetUserName

Deze functie haalt de gebruikersnaam op. Het geeft een niet-nul als het gelukt is. Als de stringbuffer niet groot genoeg is om de hele gebruikersnaam te houden, zal de functie falen. Als de functie slaagt, zal het de aantal karakters kopiëren naar de buffer in de struct, vandaar de reden waarom de grootte van de buffer doorgegeven moet worden als ByRef. De enige manier om een waarde als ByRef door te geven in Liberty BASIC is om gebruik te maken van een struct.

```
open "advapi32" for dll as #advapi
buf$=space$(25)+chr$(0)
struct size,L as long
size.L.struct=25
calldll #advapi,"GetUserNameA", _
     buf$ as ptr, _       'string adresbuffer voor de naam
     size as struct, _    'adres van
     re as boolean        'niet-nul is succes
print trim$(buf$)
close #advapi
end
```

GetVersion

Deze functie haalt de versie van Windows op die op dit moment uitgevoerd wordt. Hier is een voorbeeld:

```
struct OSVERSIONINFO, _
     dwOSVersionInfoSize as ulong, dwMajorVersion as ulong, _
     dwMinorVersion as ulong, dwBuildNumber as ulong, _
     dwPlatformId as ulong, szCSDVersion as char[128]

L=len(OSVERSIONINFO.struct)
OSVERSIONINFO.dwOSVersionInfoSize.struct=L

calldll #kernel32, "GetVersionExA", _
     OSVERSIONINFO as struct, result as boolean

print "Major Versie is ";
print OSVERSIONINFO.dwMajorVersion.struct
print "Minor Versie is ";
print OSVERSIONINFO.dwMinorVersion.struct
```

dwMajorVersion
Identificeert het major versie nummer van het operating system. Bijvoorbeeld, voor Windows NT versie 3.51 is het major versie nummer 3 en voor Windows NT versie 4.0 is het major versie nummer 4.

dwMinorVersion
Identificeert het minor versie nummer van het operating system. Bijvoorbeeld, voor Windows NT versie 3.51 is het minor versie nummer 51 en voor Windows NT versie 4.0 is het minor versie nummer 0.

GetWindowsDirectory

Deze functie haalt de map van de Windows directory op. De Windows directory bevat soorten bestanden als Windows applicaties, initialisatiebestanden en helpbestanden. De teruggegeven string wordt opgeslagen in de WindowsDir$ buffer en de lengte van de string wordt geretourneerd in “result”.

```
WindowsDir$=space$(200)+chr$(0)
calldll #kernel32, "GetWindowsDirectoryA", _
     WindowsDir$ as ptr, _    'buffer om informatie te ontvangen
     200 as long, _           'buffergrootte
     result as long
print left$(WindowsDir$, result)
```

LoadLibrary

Gebruik deze functie om een handle van een DLL of een uitvoerbaar programma te krijgen. Als file$ geen map bevat, kijkt Windows in de applicatie directory, in de huidige directory en in de Windows\System directory.

```
calldll #kernel32, "LoadLibraryA", _
     file$ as ptr, _     'mogelijk notepad.exe
     hLib as ulong       'handle naar library
```

QueryPerformanceCounter en QueryPerformanceFrequency

De QueryPerformanceFrequency functie haalt de frequentie van de hoge-resolutie prestatieteller, als er een bestaat. Als de geïnstalleerde hardware de hoge-resolutie prestatieteller ondersteunt, is de geretourneerde waarde niet-nul. Als het niet zo is, is de geretourneerde waarde nul. Als het systeem deze functie ondersteunt, biedt QueryPerformanceCounter een hoge-resolutie timer. De QueryPerformanceCounter functie haalt de huidige waarde op van de hoge-resolutie prestatieteller, als er een bestaat. Als de geïnstalleerde hardware een hoge-resolutie prestatieteller ondersteunt, is de geretourneerde waarde niet-nul, anders zal de waarde nul zijn. Deze functies vereisen een pointer naar een numerieke variabele, zo dat ze doorgegeven moeten worden als een enkele-member struct in Liberty BASIC. De functie stelt deze waarden in, zodat het teruggegeven kan worden vanaf de struct nadat de functie het geretourneerd heeft.

```
struct res, lowPart as ulong, highPart as long

'zie als systeem heeft een hoge frequentie teller
calldll #kernel32, "QueryPerformanceFrequency", _
     res as struct, result as boolean

if result=0 then
     print "Systeem ondersteunt niet hi-res teller."
     end
else
     freq=largeInt2Dec(res.lowPart.struct, res.highPart.struct)
     calldll #kernel32, "QueryPerformanceCounter", _
          res as struct, result as boolean
```

```
     first=largeInt2Dec(res.lowPart.struct, res.highPart.struct)
     calldll #kernel32, "QueryPerformanceCounter", _
          res as struct, result as boolean
     second=largeInt2Dec(res.lowPart.struct, res.highPart.struct)

     print "Frequentie in aantal per seconde: ";freq
     print "Eerste waarde: ";first
     print "Tweede waarde: ";second
     print "Tijd (tikken) verstreken: ";second-first
end if
end

function largeInt2Dec(low, high)
     lowHex$ = right$("0000" + dechex$(low), 4)
     highHex$ = right$("0000" + dechex$(high), 4)
     largeInt2Dec = hexdec(highHex$ + lowHex$)
end function
```

Wordt vervolgd

PowerBASIC lusstructuren.

De structuren, niet alleen PowerBASIC.

Elke BASIC dialect en programmeertaal kent structuren die de uitvoervolgorde in een programma bepalen. Hier laat ik PowerBASIC zien, maar andere hobbyisten kunnen eenvoudig deze structuren in hun eigen taal gebruiken. Elke taal is weer anders, maar het geraamte zal hetzelfde zijn.

FOR … NEXT

Deze structuur is een herhalingsstructuur die een codeblok blijft uitvoeren tot er een bepaalde waarde is bereikt. De eindwaarde staat altijd vast en kan niet door bijvoorbeeld een conditie worden bepaald.

```
numcats% = 0
FOR count% = 1 TO 10
     numcats% = numcats% + count%
     PRINT numcats%; "te veel? Castreer uw kat!"
NEXT count%
```

U kunt ook de waarde van count% binnen de lus wijzigen. Wees daar wel voorzichtig mee om oneindige uitvoer te voorkomen. Onderstaand voorbeeld zal oneindig uitvoeren:

```
numcats% = 0
FOR count% = 1 TO 10
     numcats% = numcats% + count%
     PRINT numcats%
     ' onderstaande regel voorkomt het tellen wanneer deze hoger is dan 5
     IF count% > 5 THEN count% = count% - 1
NEXT count%
```

Het is ook mogelijk de waarde van de teller in stappen te laten lopen met gebruik van het STEP sleutelwoord. Normaal wordt er verhoogd met één. Bij gebruik van STEP kunt u het verhogen op elke manier die u wilt, zo ook voor een lus die verlaagd:

```
numcats% = 0
FOR count% = 10 TO 1 STEP -2
     numcats% = numcats% + count%
     PRINT numcats%
NEXT count%
```

Om even te weten over het laatste voorbeeld, de lus loopt vijf keer. In de laatste stap zal *count%* gelijk zijn aan 2. Wanneer de verhoging van STEP is afgetrokken (optelling van -2), *count%* zal lager zijn dan de laagste limiet van één, dus de lus stopt. Een beetje oplettendheid is wel nodig bij het gebruik van STEP om zeker te weten dat de lus uitgevoerd zal worden zoals u wilt.

De start, het einde en de stap waarden hoeven geen integers te zijn. Bijvoorbeeld, de volgende FOR … NEXT lus is toegestaan:

```
FOR count% = 0.4 TO 1.8 STEP 0.2
     PRINT count%;
NEXT count%
```

Wees voorzichtig met afrondingsfouten wanneer u gebruik maakt van waarden met drijvende komma. Bijvoorbeeld,

```
FOR count! = 0 TO 2 STEP 0.1
     PRINT count!;
NEXT count!
```

loopt maar 20 keer (de laatste waarde die geprint wordt is 1.9), en dat komt omdat de waarde 0.1 niet juist kan worden weergegeven met behulp van getallen met enkele precisie. De single precisie weergave van 0.1 en de vastgestelde waarde van *count!* is groter dan 2.0 in de laatste iteratie, hoewel het er goed op papier uitziet. Dezelfde lus die gebruik maakt van binair gecodeerde decimalen met drijvende komma, werkt zoals verwacht. Onderstaande lus loopt 21 keer (de laatste geprinte waarde is 2):

```
FOR count@@ = 0 TO 2 STEP 0.1@@
     PRINT count@@;
NEXT count@@
```

FOR … NEXT lussen kunnen genest worden met andere FOR … NEXT lussen of andere structuren. Als voorbeeld, om alle elementen van een 10x5x4 array te printen:

```
FOR x% = 0 TO 9
     FOR y% = 0 TO 5
          FOR z% = 0 TO 4
               PRINT A(x%, y%, z%)
          NEXT z%
     NEXT y%
NEXT x%
```

De drie NEXT statements kunnen gecombineerd worden in één:

```
NEXT z%, y%, x%
```

Als u een luie typist bent, kunt u PowerBASIC ook zelf de NEXT variabelen laten bepalen:

```
FOR count = 1 TO 10
     FOR index = 1 TO 20
          ...
     NEXT
NEXT
```

implementeert `NEXT index`, gevolgd door `NEXT count`. Als u echt vaag wilt zijn, kunt u ook die twee combineren

```
NEXT,
```

dat implementeert

```
NEXT index, count
```

Natuurlijk zal de leesbaarheid verminderen en de kans op fouten zal deels door de programmeur toenemen.

WHILE … WEND

Een probleem met FOR … NEXT is dat u moet weten wat de eindwaarde is van *count* voordat u de lus laat uitvoeren. Maar als u niet precies weet wanneer uw variabele het maximum waarde zal bereiken die u wilt toestaan, zal u moeten eindigen met gebruik van een GOTO om de lus te verlaten:

```
FOR count = 1 TO 1000    ' doe dit als MEESTE, 1000 keer
     ' voeg een willekeurig bedrag toe aan het totaal
     total = total + RND * count
     ' laat niet het totaal boven de 1000 komen
     IF total > 1000 GOTO 100
NEXT count
100  PRINT count    ' count zal kleiner zijn dan 1000
```

Hoewel dit niet een kapitaal overtreding is, kan het met behulp van de WHILE … WEND lus worden vermeden, die een voorwaarde controleert, en voor zolang de voorwaarde houdt, wordt de actie uitgevoerd.

```
count = 0
WHILE total < 1000
     count = count + 1
     total = total + RND * count
WEND
```

De FOR … NEXT en WHILE … WEND constructies kunnen eenvoudig elk soort lus vereisen die u hebt, maar soms wilt u wel eens wat meer variaties. PowerBASIC heeft daar de DO … LOOP voor.

DO … LOOP

De DO … LOOP lus kan voorwaarden controleren aan het begin van de lus of aan het einde van de lus. De simpelste DO … LOOP lus ziet eruit als dit (normaal laat u een voorwaarde uitvoeren om te stoppen; deze lus zal oneindig uitvoeren):

```
DO
     cats! = cats! + 1
LOOP
```

U kunt in een DO … LOOP lus een WHILE gebruiken, maar dat converteert de DO … LOOP lus naar een WHILE … WEND structuur:

```
DO WHILE a < b
     {statements}
LOOP
```

die hetzelfde doet als deze:

```
WHILE a < b
     {statements}
WEND
```

zodat u geneigd bent gebruik te maken van een WHILE … WEND lus. De compiler zal in dit geval het sleutelwoord WEND vervangen voor LOOP.

U kunt ook de WHILE statement aan het einde van de lus plaatsen, maar dan wel dat de statements in de lus één keer uitgevoerd zal worden voordat de voorwaarden controleert worden:

```
DO
     {statements}
LOOP WHILE a < b
```

De DO … LOOP ondersteunt ook UNTIL, als voorbeeld:

```
DO
     {statements}
LOOP UNTIL a < b
```

Hier eet u uw dessert eerst, voordat u controleert of de klusjes zijn gedaan. Dit is nuttig als u niet van klusjes houdt of u zorgt ervoor dat er tenminste één scheur in het proces wordt verwerkt, ongeacht wat de status van het programma op het moment heeft. Hier is een plek waar een beetje kennis gevaarlijk is en niettemin zou kunnen. De plaatsing van de UNTIL statement is zeer belangrijk. Als u bekend bent met zulke constructies in andere programmeertalen, wees dan wel voorzichtig. Als u UNTIL vooraan bij DO gaat gebruiken, zal het hetzelfde uitvoeren, maar dan met een tegenovergestelde conditie:

```
DO UNTIL a < b
     {statements}
LOOP
```

is helemaal hetzelfde als

```
DO WHILE a >= b
     {statements}
LOOP
```

dat weer hetzelfde zal zijn als een WHILE … WEND structuur. U kunt deze structuren op die manier gebruiken als u deze vorm makkelijker te lezen vindt. De volgende twee code voorbeelden zijn gelijk; sommigen zullen het ene makkelijker vinden om te lezen en sommigen zullen het andere makkelijk vinden om te lezen.

```
DO WHILE (a > b) OR (c < d)
     {statements}
```

```
LOOP
```
zal gelijk zijn als

```
DO UNTIL (a <= b) AND (c >= d)
     {statements}
LOOP
```

Eerder uitstappen

U kunt ook uit een lus stappen zonder gebruik van GOTO, zie hieronder.

```
FOR CurrentRec& = 1 TO NumRecords&
     ' Lees een record
     ' Vergelijk met de zoek informatie
     IF Found <> 0 THEN
          EXIT FOR
     END IF
NEXT CurrentRec&
' andere uitvoering
```

PowerBASIC kent een EXIT sleutelwoord die voor elke lus werkt: EXIT DO, EXIT WHILE, EXIT FOR. U mag het tweede sleutelwoord overslaan. Het vertelt PowerBASIC vanzelf welke lus afgebroken moet worden. Het vorige voorbeeld zal dan herschreven kunnen worden als dit:

```
FOR CurrentRec& = 1 TO NumRecords&
     ' Lees een record
     ' Vergelijk met de zoek informatie
     IF Found <> 0 THEN
          EXIT
     END IF
NEXT CurrentRec&
' andere uitvoering
```

U kunt ook de lus laten beginnen met de volgende iteratie zonder alle statements uit te voeren binnen de lus, door gebruik te maken van de ITERATE statement.

```
FOR CurrentRec& = 1 TO NumRecords&
     ' Lees een record
     IF Deleted THEN
          ITERATE FOR
     END IF
     ' Record uitvoeren
NEXT CurrentRec&
```

U kunt ook het tweede sleutelwoord overslaan, zoals u ook kunt doen bij EXIT.

```
FOR CurrentRec& = 1 TO NumRecords&
     ' Lees een record
     IF Deleted THEN
          ITERATE
     END IF
     ' Record uitvoeren
NEXT CurrentRec&
```

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 € 5,00.
Meer informatie? Kijk op 'http://www.buurthuisbronveld.nl' of neem contact op met mij. Voor overige informatie: http://www.tronicasoftware.nl

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.

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

Waarover	Wie	Wanneer	Tijd	Telefoon	Email
Liberty BASIC	Gordon Rahman	ma. t/m zo.	19-23	(023) 5334881	grahman@planet.nl
MSX-Basic	Erwin Nicolai	vr. t/m zo.	18-22	(0516) 541680	basic@lordthanatos.com
PowerBasic CC	Fred Luchsinger	ma. t/m vr.	19-21		f.luchsinger@kader.hcc.nl
QBasic, QuickBasic	Jan v.d. Linden				j.vd.linden@kader.hcc.nl
Visual Basic voor Windows	Jeroen v. Hezik	ma. t/m zo.	19-21	(0346) 214131	j.a.van.hezik@kader.hcc.nl
Visual Basic .NET	Marco Kurvers	vr. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl
Basic algemeen, zoals VBA Office en XNA Game Studio	Marco Kurvers	vr. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl