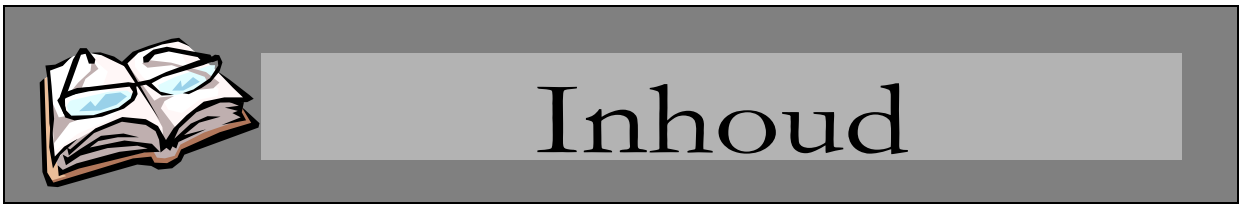
Programmeren Bulletin

25ste jaargang **Nummer 1**
extra dik winter tot zomer 2018

hcc! programmeren

Interessegroep



Inhoud

Redactioneel

Door een groot probleem met mijn laptop en mijn ziekte was ik helaas te laat met de Bulletin winter editie. Daarom zal deze editie een extra dikke Bulletin worden met de lente editie erbij en met veel onderwerpen.

Veel succes allemaal.

In dit deel van de tutorial Arkanoid komt de bal erbij en laat ik zien hoe we de bal laten botsen tegen het racket en tegen de borderranden. Wilt u nog eens kijken op de Engelstalige pagina? Hier nog eens het internetadres: https://noobtuts.com/unity/2d-arkanoid-game

Hobbyisten kunnen genieten van een leuk C onderwerp. Hoe kunnen we gebruik maken van strings en dynamische arrays? Er is een handige header string.h die een aantal functies bevat waarmee we met strings kunnen werken.

Ook voor BASIC gebruikers heb ik een onderwerp over Liberty BASIC. De gewone Liberty BASIC structuur in plaats van de API aanroepingen.

Marco Kurvers

Programmeren in Pascal – Documentatie.

Pascal kent allerlei soorten types. Het is heel uitgebreid. Daarom kunnen we in Pascal meer met de types doen dan in Basic.

Een Pascal type gebruikt u als volgt:

var variabele: <typenaam>;

Hieronder zien we een lijst met de types, sommige gegeven in groepen (Engelstalig).

Base types	Ordinal types	Real types	Character types
Char or AnsiChar	WideChar	Other character types	
Single-byte String types		Multi-byte String types	
Constant strings		PChar - Null terminated strings	
String sizes	Structured Types	Arrays	Record types
Set types	File types	Pointers	Forward type declarations
Procedural types	Variant types	Definition	
Variants in assignments and expressions		Variants and interfaces	Type aliases
Managed types			

Willen we structuren declareren, dan moeten we type aliassen gebruiken. Het is niet toegestaan om bijvoorbeeld een variabele te declareren met een bepaald subbereik, zie later bij Type aliases.

Base types
De base types, of ook wel standaard types genoemd, zijn de meest gebruikte types. Deze types zijn:

Ordinal types, zoals:
- Integers
- Boolean types
- Enumeration types
- Subrange types

Real types

De ordinale types hebben de volgende karakteristieken:
1. Ordinale types zijn telbaar en werken op volgorde, met andere woorden, het is mogelijk ze één voor één op- of af te tellen in een specifieke volgorde. Met deze eigenschap kunt u de werking van functies als Inc, Ord, Dec op ordinale typen definiëren.
2. Ordinale waarden hebben een kleinste mogelijke waarde. Probeert u de kleinste mogelijke waarde toe te passen op de Pred functie, dan wordt er een Range Check Error gemeld als het selectievakje voor Range Checking is ingeschakeld.
3. Ordinale waarden hebben een hoogste mogelijke waarde. Probeert u de hoogste mogelijke waarde toe te passen op de Succ functie, dan wordt er een Range Check Error gemeld als het selectievakje voor Range Checking is ingeschakeld.

! Int64 en QWord worden beschouwd als ordinale types op 64-bits processoren. Op 32-bits processoren hebben zij enkel de ordinale kenmerken, maar ze kunnen dan bijvoorbeeld niet worden gebruikt in for lussen.

Hieronder is een lijst te zien van Integer types:

Integer
Shortint
SmallInt
Longint
Longword
Int64
Byte
Word
Cardinal
QWord
Boolean
ByteBool
WordBool
LongBool
Char

De integer types, en hun reeksen en grootte, die vooraf zijn gedefinieerd in Free Pascal staan hieronder. Houd er rekening mee dat de beschikbare types qword en int64 geen echte ordinale types zijn, dus in sommige Pascal versies zullen deze twee integer types niet werken. Merk op dat niet alle vooraf gedefinieerde types van de lijst links hieronder staan. Zij hebben namelijk geen bepaald bereik. Ook een Char is een ordinaal integer type, hoewel deze wordt gebruikt voor alfanumerieke waarden.

Hieronder een lijst met de bereiken en de groottes van de vooraf gedefinieerde types:

Type	Bereik	Grootte in bytes
Byte	0 .. 255	1
Shortint	-128 .. 127	1
Smallint	-32768 .. 32767	2
Word	0 .. 65535	2
Integer	kan een smallint of een longint zijn	grootte 2 of 4
Cardinal	longword	4
Longint	-2147483648 .. 2147483647	4
Longword	0 .. 4294967295	4
Int64	-9223372036854775808 .. 9223372036854775807	8
QWord	0 .. 18446744073709551615	8

In Free Pascal is een Integer type standaard een Smallint type. In Delphi en ObjFPC mode is het standaard een Longint type. Het Cardinal type is altijd een Longword type.

! Alle decimale-constanten die niet binnen de -2147483648 .. 2147483647 bereik liggen, zullen automatisch worden geparseerd als 64-bits geheel getal constanten vanaf versie 1.9.0. Eerdere versies converteren het naar een real getypeerde constante.

Boolean type
Free Pascal ondersteunt het Boolean type met twee vooraf gedefinieerde waarden True en False. Dit zijn de enige twee waarden die toegekend kunnen worden en elke expressie die als een boolean resulteert wordt als een True of een False beschouwd.

Naam	Grootte	Ord(True)
Boolean	1	1
ByteBool	1	Elke geen-nul waarde
WordBool	2	Elke geen-nul waarde
LongBool	4	Elke geen-nul waarde

Free Pascal ondersteunt ook de ByteBool, WordBool en LongBool types. Dit zijn dezelfde als de Byte, Word en Long, maar worden beschouwd als een Boolean type: de waarde False is gelijk aan 0 (nul) en elke geen-nul waarde wordt als waar beschouwd bij een conversie naar een boolean type. Een boolean waarde True wordt geconverteerd naar een -1, in gelijkenis zal dit ook een LongBool zijn.

Onderstaande toekenningen van het type Boolean zijn geldig.

```
B := True;
B := False;
B := 1<>2;     { resulteert als B := True }
```

Boolean condities kunnen ook in expressies worden gebruikt.

! In Free Pascal zijn Boolean expressies standaard altijd geëvalueerd op een zodanige wijze dat wanneer het resultaat bekend is, de rest van de expressie niet langer zal worden geëvalueerd: dit heet Booleaanse evaluatie van de kortere weg.

In het volgende voorbeeld zal de functie Func niet worden aangeroepen. Doordat de waarde van B een False is en er een And operator wordt gebruikt, zal de functie niet aangeroepen worden. Was de waarde True geweest, dan wel. Zouden we de And operator in een Or veranderen, dan zal de functie ook aangeroepen worden, ook al is de waarde False.

```
...
B := False;
A := B And Func;
```

Func is in dit voorbeeld een functie die een booleaanse waarde resulteert.

Dit gedrag is controleerbaar door de compilerinstructie {$B}.

Enumeratietypes
Free Pascal ondersteunt enumeratietypes. Bovenop de Turbo Pascal-implementatie kent Free Pascal ook een uitbreiding van de C-stijl van het enumeratietype, waar een waarde wordt toegewezen aan een bepaald element uit de enumeratielijst.

```
Type
     Richting = (Noord, Oost, Zuid, West);
```

Een C-stijl enumeratietype ziet er als volgt uit:

```
Type
     EnumType = (Eén, Twee, Drie, Veertig := 40, Eénenveertig);
```

Als resultaat is Veertig gelijk aan 40 en niet 3, maar wel als ‘:= 40’ er niet had gestaan. Zo is Eénenveertig gelijk aan 41 en niet 4, maar wel als de toekenning er niet had gestaan. Wanneer de compiler een toekenning tegenkomt bij een element, zal het volgende element dezelfde waarde krijgen plus 1, en dat ook zo met het volgende element, enzovoort.

Bij het maken van enumeratietypes moet u goed onthouden dat u de elementen in oplopende volgorde plaatst. Onderstaand enumeratietype geeft een compiler fout:

```
Type
     EnumType = (Eén, Twee, Drie, Veertig := 40, Dertig := 30);
```

Het is belangrijk om de elementen Veertig en Dertig in de juiste volgorde te zetten. Wanneer u enumeratietypes gebruikt, moet u het volgende weten:

1. De Pred en Succ functies kunnen niet worden gebruikt bij enumeratietypes. Wanneer u op elke wijze het toch doet, zal er een compiler fout verschijnen.

2. Enumeratietypes zijn opgeslagen met behulp van standaard, onafhankelijke, werkelijke aantal waarden: de compiler probeert niet de ruimte te optimaliseren. Dit gedrag kan veranderd worden met de {$PACKENUM n} compiler directive, die aan de compiler het minimale aantal bytes doorgeeft die gebruikt moeten worden voor het enumeratietype. Hieronder een voorbeeld:

```
type
{$PACKENUM 4}
  GroteEnum = (GroteEén, GroteTwee, GroteDrie);
{$PACKENUM 1}
  KleineEnum = (één, twee, drie);
var K: KleineEnum;
    G: GroteEnum;
begin
  Writeln('Kleine enum: ', SizeOf(K));
  Writeln('Grote enum : ', SizeOf(G));
end.
```

zal na het uitvoeren het volgende weergeven:

```
Kleine enum: 1
Grote enum : 4
```

Subrange types
Sommige voorgedefinieerde integer types zijn subrange types:

- ```
  type
    Longint  = -2 147 483 648..2 147 483 647;
    Integer  = -32768..32767;
    Shortint = -128..127;
    Byte     = 0..255;
    Word     = 0..65535;
  ```

Zoals ik eerder heb uitgelegd kan een subrange niet direct met een variabele gedeclareerd worden. De volgende declaratie is niet toegestaan:

```
var bereikA: 10..50;
```

Om dit op te lossen moet onderstaande alias worden gebruikt door een subrange type te definiëren:

```
type TBereik = 10..50;
...
var BereikA: TBereik;
```

Subrange types van enumeratietypes kunnen ook worden gedefinieerd:

```
type
  Dagen = (maandag, dinsdag, woensdag, donderdag, vrijdag,
           zaterdag, zondag);
  WerkDagen = maandag .. vrijdag;
  WeekEnd = zaterdag .. zondag;
```

Real types
Free Pascal gebruikt de rekenkundige coprocessor (of emulatie) voor alle floating-point berekeningen. Het Real type is processor afhankelijk, maar is Single ofwel Double. De Turbo Pascal compatible types zijn hieronder te vinden.
```

Type	Bereik	Significante cijfers	Grootte
Real	platform afhankelijk	???	4 of 8
Single	1.5E-45 .. 3.4E38	7-8	4
Double	5.0E-324 .. 1.7E308	15-16	8
Extended	1.9E-4932 .. 1.1E4932	19-20	10
Comp	-2E64+1 .. 2E63-1	19-20	8
Currency	-922337203685477.5808 .. 922337203685477.5807	19-20	8

Het Comp type is, in feite, een 64-bits integer en is niet beschikbaar op alle doelplatforms. De Currency is een fixed-point real data type die intern wordt gebruikt als een 64-bits geheel getal (automatisch geschaald met een factor 10000). Dit vermindert afrondingsfouten.

Char of AnsiChar
Free Pascal ondersteunt het type Char. Een Char is precies 1 byte groot en heeft als inhoud één ASCII teken. Een tekenconstante kan tussen enkele aanhalingstekens worden opgegeven, zoals ‘a’ of ‘A’ zijn beide tekenconstanten. Een teken kan ook met een tekenwaarde worden opgegeven (algemeen een ASCII code), een ordinale waarde voorafgaand met het nummer symbool (#). Als voorbeeld: het opgeven van #65 is hetzelfde als ‘A’. Ook kan het invoegteken (^) worden gebruikt in combinatie met een letter om een ASCII-waarde van minder dan 27 op te kunnen geven. Dus ^G resulteert in #7 - G is de zevende letter van het alfabet. De compiler is nogal slordig over de tekens achter de caret, maar algemeen moet men aannemen dat alleen letters zijn toegestaan.

Wanneer het enkele aanhalingsteken weergegeven moet worden, moet het twee keer achtereenvolgens worden getypt, dus `''''`.

Om Char van WideChar te onderscheiden, definieert de systeemeenheid ook het type AnsiChar, dat een alias is van het Char type. In toekomstige versies van FPC komt er mogelijk een alias van het Char type voor WideChar of AnsiChar.

WideChar
Free Pascal ondersteunt het type WideChar. Een WideChar is precies 2 bytes groot en heeft als inhoud één UNICODE teken in UTF-16 codering. Een unicode-teken kan worden aangegeven door zijn tekenwaarde (een UTF-16 code), een ordinale waarde voorafgaand met het nummer symbool (#). Een letterlijke normale ASCII (1-byte) teken kan ook worden gebruikt voor een WideChar, de compiler zal het automatisch converteren naar een 2-byte teken UTF-16.

Het volgende definieert sommige Griekse tekens (pi, omega):

```
const
  C3: WideChar = #$03A8;
  C4: WideChar = #$03A9;
```

Hetzelfde kan worden gedaan door een Word te typecasten naar een WideChar:

```
const
  C3 : WideChar = WideChar($03A8);
  C4 : WideChar = WideChar($03A9);
```

Ander soort character types

Free Pascal definieert sommige andere tekentypen in de systeemeenheid zoals UCS2Char, UCS4Char, UniCodeChar. Echter, geen speciale ondersteuning voor deze tekentypen bestaat. Ze zijn alleen gedefinieerd voor Delphi compatibiliteit.

Single-byte String types
Free Pascal ondersteunt het type String zoals dit is gedefinieerd in Turbo Pascal: een opeenvolging van single-byte tekens met een optionele grootte. Het ondersteunt ook ansistrings (met onbeperkte lengte) en codepage informatie zoals in Delphi.

Als er een aangegeven grootte is (het gebruik van vierkante haakjes), dan is de maximumwaarde 255. Als er een aanduiding is van een codepage (met behulp van ronde haken), dan is dit een ansistring met bijbehorende code pagina-informatie.

De betekenis van de declaratie-instructie van een string zonder grootte en code pagina-aanduiding wordt anders opgevat afhankelijk van de {$H}-schakelaar:

```
var
  A: string;
```

Als er geen grootte en codepagina indicatievermelding aanwezig is, kan de bovenstaande declaratie ook als een ansistring of een shortstring gelden.

Ongeacht de werkelijke aard, single-byte strings kunnen door elkaar worden gebruikt. De compiler zorgt altijd voor de nodige typeconversies. Merk echter op dat het resultaat van een expressie dat ansistrings als korte tekenreeksen bevat altijd een ansistring zal zijn.

Short strings
Een string declaratie declareert een short string in de volgende gevallen:

1. Als de $H schakelaar uitstaat: {$H-}, zal de string declaratie altijd een short string declaratie zijn.
2. Als de $H schakelaar aanstaat: {$H+} met een maximaal opgegeven lengte, dan zal de string declaratie een short string declaratie zijn.

Short strings werken altijd vanuit het gebruik van de system codepage. Het voorgedefinieerde type `ShortString` is gedefinieerd als een string met een 255 grootte:

```
    ShortString = string[255];
```

Als de grootte van de string niet is opgegeven, is 255 de standaard. De werkelijke lengte van de string kan worden verkregen met de standaard runtime `Length()` functie.

Zoals onderstaand voorbeeld:

```
{$H-}

type
  NaamString = string[10];
  StraatString = string;
```

NaamString kan als inhoud 10 tekens hebben, terwijl StraatString een inhoud kan hebben van 255 tekens.

Omdat een short string een maximale lengte kan hebben van 255 tekens, zal de compiler een foutmelding geven als men toch de maximale lengte probeert te overschrijden:

```
Error: string length must be a value from 1 to 255
```

Voor short strings, is de lengte opgeslagen in het teken bij index 0. Oude Turbo Pascal code berust op dit, en het is op dezelfde manier uitgevoerd in Free Pascal.

Ondanks dit, om overdraagbare code te schrijven, is het beste om de lengte van een shortstring met de SetLength() statement in te stellen, en met de Length() functie op te vragen. Dit zal altijd werken, ongeacht de interne representatie van de shortstrings of ander gebruik van de strings: hierdoor is het gemakkelijk schakelen tussen de verschillende string types.

Ansistrings
Ansistrings zijn tekenreeksen die geen lengte-limiet hebben, en hebben een gekoppeld codetabel. Ze zijn nul-beëindigd. Een ansistring wordt intern behandeld als een pointer: de feitelijke inhoud van de tekenreeks wordt opgeslagen op de heap. Er wordt zoveel geheugen als nodig is, voor het opslaan van de inhoud van de tekenreeks, toegewezen.

Als geen codepage in het statement wordt gegeven, wordt uitgegaan van de codetabel van het systeem. Deze codepage wordt bepaald door de constante `DefaultSystemCodePage` in de system unit. Dit is allemaal transparant behandeld, dat wil zeggen ze kunnen worden behandeld als een normale short string. Ansistrings kunnen worden gedefinieerd met behulp van de vooraf gedefinieerde AnsiString type of met behulp van het string sleutelwoord in de modus {$H+}.

Onderstaand voorbeeld laat zien hoe een reference count werkt en hoe de codepage gebruikt wordt:

```
type
  TString1 = type string(1252);
  TString2 = type string(1251);

var
  A: TString1;
  B: TString2;

begin
  A:='123'+'345'+IntToStr(123);
  B:=A;
  Writeln('B : ', StringRefCount(B));
  Writeln('A : ', StringRefCount(A));
end.
```

De compiler zal de inhoud van string B in de codepage van string A omzetten. Merk op dat als een codetabelconversie plaatsvindt, het verwijzingsmechanisme niet wordt gebruikt: een nieuwe string wordt toegewezen.

Deze automatische conversie van codetabellen kan serieuze vertragingen veroorzaken, dus zorg ervoor dat de codepage conversies tot een minimum beperkt zijn.

De codetabel van een string kan expliciet met behulp van de SetCodePage-routine van de system unit worden ingesteld. Het aanroepen van deze routine zal de waarde van een string omzetten naar de gevraagde codepage.

RawByteString
Het voorgedefinieerde RawByteString type is een AnsiString-tekenreekstype zonder codepage informatie (CP_NONE):

```
type
  RawByteString = type AnsiString(CP_NONE);
```

Als de converteer routines in een bron- of doelstring een `CP_NONE` tegenkomen, zal geen codepage conversie plaatsvinden; de codepage van de bronstring zal worden behandeld. Om deze reden gebruiken de meeste single-byte string routines in het systeem en in sysutils units het RawByteString type.

PChar – Null beëindigde strings
Free Pascal ondersteunt de Delphi implementatie van het PChar type. PChar is gedefinieerd als een pointer naar een Char type, maar staat extra bewerkingen toe. Het PChar type kan het beste worden begrepen als een Pascal vergelijking met een C-stijl null beëindigde string, dat wil zeggen een variabele van het type PChar is een pointer die verwijst naar een array van het type Char en beëindigd wordt met een null-teken (#0). Free Pascal ondersteunt geïnitialiseerde getypeerde PChar constanten of een directe toekenning. Als voorbeeld zijn de volgende stukjes code hetzelfde:

```
program één;
var P: PChar;
begin
  P := 'Dit is een null beëindigende string.';
  Writeln(P);
end.
```

Resulteert op dezelfde manier als

```
program twee;
const P: PChar = 'Dit is een null beëindigende string.';
begin
  Writeln(P);
end.
```

Deze voorbeelden laten ook zien dat het mogelijk is om de inhoud van een string weg te kunnen schrijven naar een bestand van het type Text. De strings unit bevat procedures en functies die de PChar type manipuleren als in de standaard C library. Sinds het naar een pointer toe vergelijkbaar is naar een variabele van het type Char, is het ook mogelijk het volgende te doen:

```
program drie;
var S: string[30];
    P: PChar;
begin
  S := 'Dit is een null beëindigende string.'#0;
  P := @S[1];
  Writeln(P);
end.
```

Dit zal hetzelfde resultaat geven als de twee vorige voorbeelden. Null beëindigende strings kunnen niet als normale Pascal strings worden toegevoegd. Als twee PChar strings samengevoegd moeten worden, moeten de functies vanuit de unit strings worden gebruikt.

Toch is het mogelijk om wat met rekenkundige pointer operatoren te doen. De operators + en - kunnen worden gebruikt voor wat PChar pointer operaties. In onderstaande tabel ziet u `P` en `Q`, die van het type `PChar` zijn en `I` is van het type `Longint`.

Operatie	Resultaat
`P + I`	Voegt `I` toe aan het adres verwezen naar `P`.
`I + P`	Voegt `I` toe aan het adres verwezen naar `P`.
`P - I`	Trekt `I af` vanuit het adres verwezen naar `P`.
`P - Q`	Resulteert, als een integer, de afstand tussen 2 adressen (of het aantal tekens tussen `P` en `Q`)

String groottes

Hoeveel geheugen wordt bezet door een tekenreeks is afhankelijk van het stringtype. Bepaalde stringtypes wijzen de stringgegevens toe in het geheugen van de heap, anderen hebben de stringgegevens op de stack. In onderstaand tabel vindt u een overzicht van het geheugengebruik van de verschillende string types. In de tabel worden de volgende symbolische constanten gebruikt:

1. L is de actuele lengte van de string.
2. HS hangt af van de Free Pascal versie, maar is 16 bytes in Free Pascal 2.7.1.
3. UHS grootte is 8 bytes voor alle Free Pascal versies.
4. WHS is in Windows 4 bytes groot voor alle Free Pascal versies. Op alle andere platforms is WHS hetzelfde als UHS, omdat het WideString type hetzelfde is als het UnicodeString type.

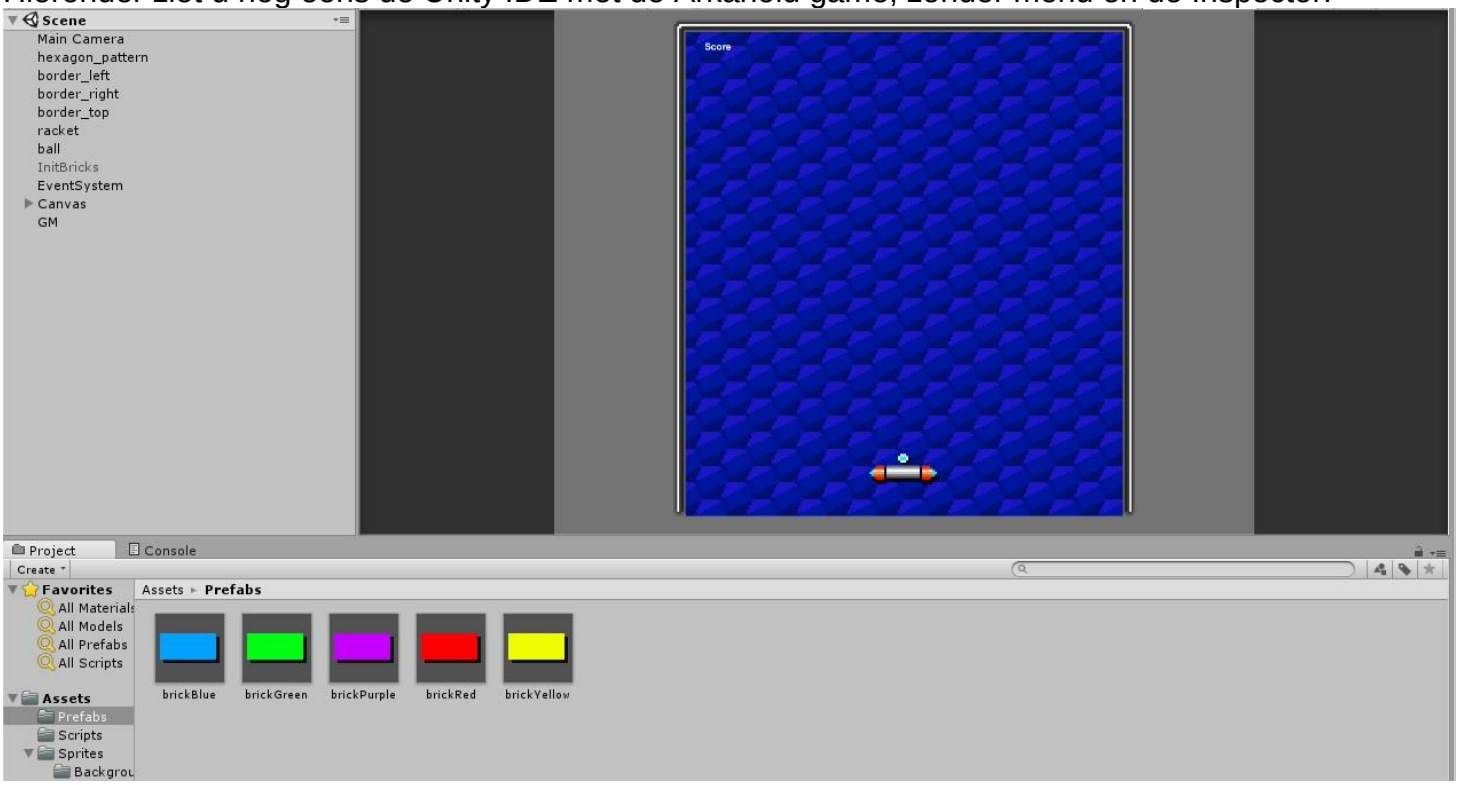
String geheugen groottes

String type	Stack grootte	heap grootte
ShortString	Gedeclareerde lengte + 1	0
AnsiString	Pointer grootte	L + 1 + HS
WideString	Pointer grootte	2*(L + 1) + WHS
UnicodeString	Pointer grootte	2*(L + 1) + UHS
PChar	Pointer grootte	L+1

Unity 2D – Tutorial: een leuke Arkanoid (2).

In deel 1 van deze tutorial heb ik laten zien hoe u het 2D speelscherm opzet en hoe u het racket met de muis kunt laten bewegen. In dit deel gaan we de bal op het scherm laten bewegen en laten botsen tegen de borderranden en het racket.

Hieronder ziet u nog eens de Unity IDE met de Arkanoid game, zonder menu en de Inspector.

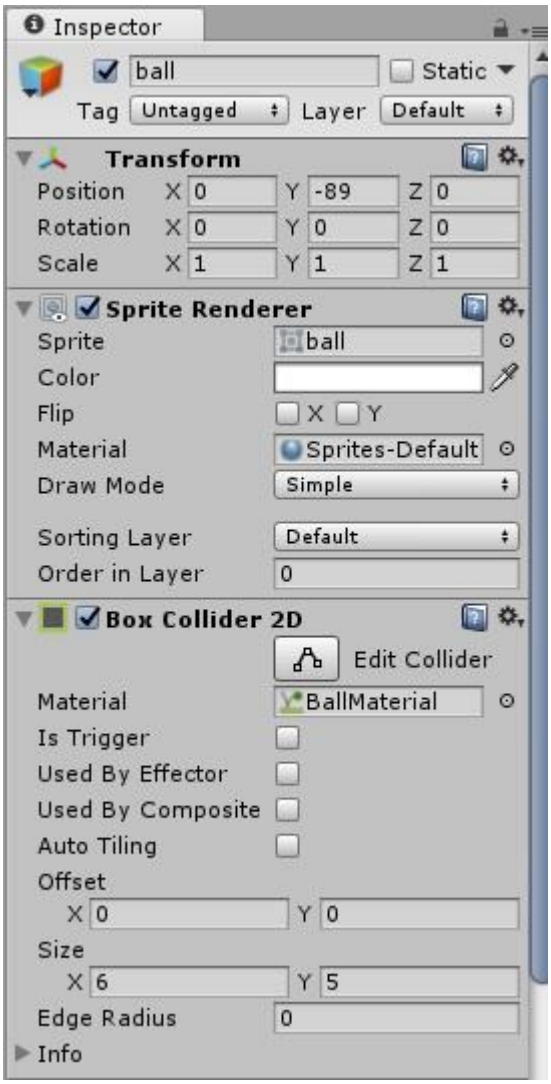


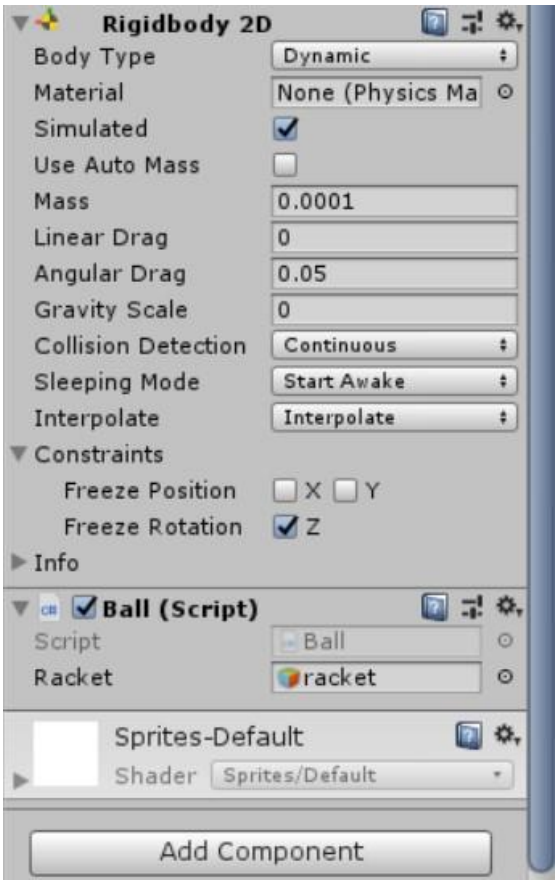
U kunt natuurlijk ook de Engelstalige versie raadplegen, maar houd er rekening mee dat bepaalde delen van deze tutorial niet overeenkomen met de website. Bijvoorbeeld de BallSticky() functie die straks wordt uitgelegd om de bal aan het racket te laten plakken als nog niet op de linker muisknop wordt gedrukt, dat wil zeggen, als nog niet de bal *afgevuurd* wordt. Zie https://noobtuts.com/unity/2d-arkanoid-game als u de Engelstalige tutorial wilt bekijken. U kunt op de pagina ook de sprites downloaden die in de tutorial nodig zijn.

Open het Arkanoid project in Unity. In de Assets gaat u naar Sprites en sleept u de ball sprite naar de Hierarchy. Klik op de ball zodat de Inspector verschijnt.

Zoals vorige keer uitgelegd, bestaat de Inspector uit meerdere componenten. Eén component is altijd aanwezig en kan ook niet worden verwijderd: de Transform component. Andere componenten kunt u toevoegen met de Add Component knop onderaan de Inspector en u kunt ook uw eigen script componenten maken en aan de Inspector toevoegen.

Op de volgende pagina ziet u de Inspector van het ball GameObject. Die heb ik in tweeën gedeeld, omdat de Inspector te lang was om in zijn geheel te kunnen kopiëren. Deze delen worden ook onder de loep genomen.





Er is echter nog een component die automatisch wordt toegevoegd zodra een sprite naar de Hierarchy wordt gesleept: de Sprite Renderer. Een leeg GameObject heeft deze component niet.

Stel de positie van de bal net zo in zoals in deze afbeelding staat. Een negatieve waarde -89 zorgt er voor dat de bal onderaan het scherm wordt gerendeerd, precies boven het racket. Dit moet lijken alsof de bal werkelijk op het racket blijft liggen.

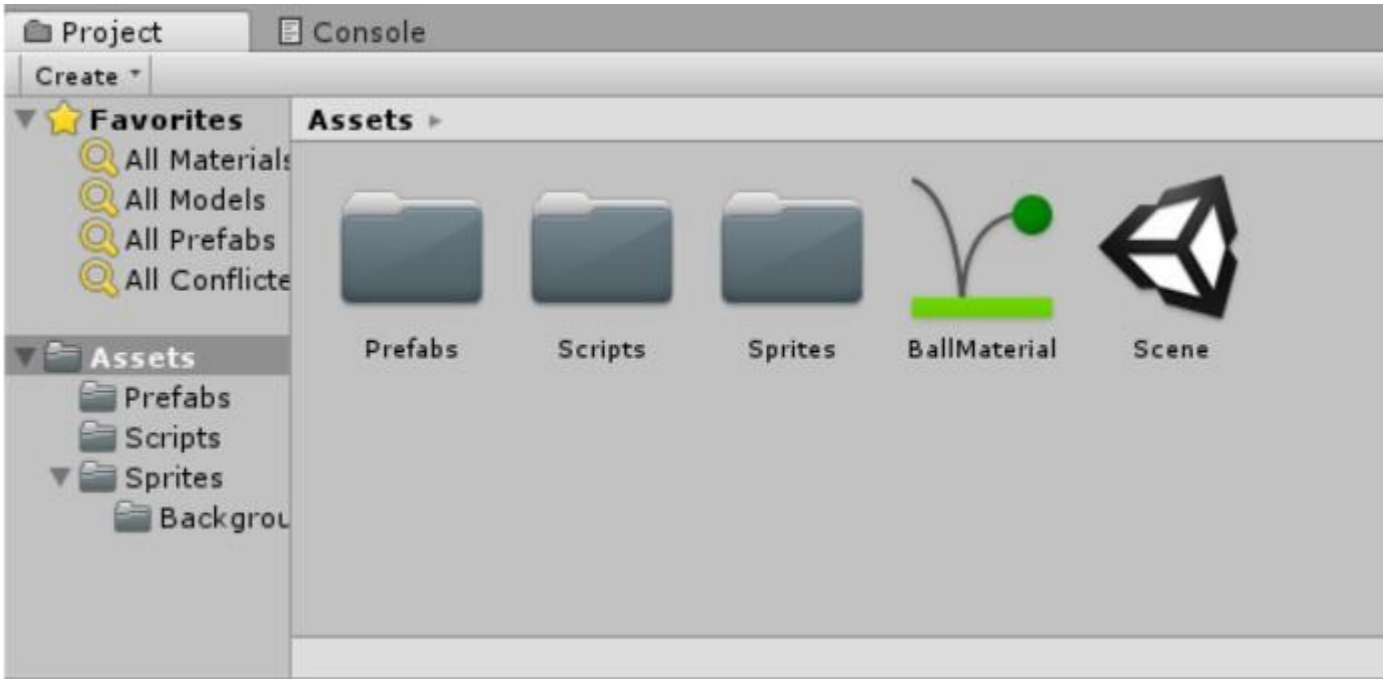
Hoewel de Sprite Renderer niet altijd belangrijk is, kan de component toch eens nut hebben. De sprite kan bijvoorbeeld een extra kleurtje krijgen en kan worden gespiegeld. Met Flip kunt u bepalen of de sprite horizontaal of verticaal gespiegeld moet worden, of beide.

Net zoals bij de andere objecten, moet het ball GameObject ook een Box Collider 2D component hebben. Op het tweede deel van de ball Inspector ziet u onderaan de knop Add Component. Klik daarop. Kies Physics 2D en klik dan op Box Collider 2D. Op deze afbeelding ziet u een material object. Dit material zorgt voor het kaatsen van de bal. Het voordeel is dat we het kaatsen niet zelf hoeven te programmeren, maar we moeten wel een bepaalde velocity meegeven in het script.

In het tweede deel van de Inspector ziet u de Rigidbody 2D component. Voeg die er ook bij door op de Add Component knop te klikken, te kiezen voor Physics 2D en dan klikken op Rigidbody 2D. Neem de gegevens over zoals u het op de afbeelding ziet staan. Vergeet vooral de Constraints niet om er voor te zorgen dat de bal niet gaat tollen. Freeze daarom de Z rotatie.

De instellingen, zoals de Mass, de Linear Drag en de Angular Drag beïnvloeden allemaal het gedrag van de bal. De waarden die u ziet zijn bepaald voor dit spel, maar niet elk spel heeft dezelfde waarden. Als u dus zelf een spel wilt maken, moet u met deze waarden experimenteren voor het GameObject waar de Inspector aan toebehoort. Dit geldt ook voor de Collision Detection en de Sleeping Mode. Deze waarden zijn nodig om de bal te laten kaatsen tegen de andere objecten. Niet alle waarden zijn nodig om te weten. Soms zijn de Sleeping Mode en de Interpolate niet nodig en kunnen standaard zo blijven staan. We kunnen dit alles ook aan Unity zelf overlaten door Use Auto Mass te kiezen. Dit kan echter niet altijd goed gaan. Gebruik het alleen wanneer weinig of geen scripts worden gebruikt voor het GameObject.

Voordat we het Ball script maken, moeten we de ballMaterial toevoegen aan de Box Collider 2D.



Klik met de rechter muisknop op de Assets. Kies Create en klik op Physics Material 2D. Het groene symbool, zie afbeelding boven, verschijnt. Wijzig de naam in BallMaterial.

Klik op BallMaterial en zie de Inspector. Stel onderstaande waarden in:

Friction	0
Bounciness	1

Klik op het ball GameObject en zie de Inspector. Sleep de BallMaterial naar de eigenschap Material in de Box Collider 2D component, zie afbeelding vorige pagina.

Het Ball script

Nu we de Inspector van de bal ingesteld hebben, moeten we alleen nog de besturing van de bal maken. De laatste component ontbreekt nog: het Ball script.

Klik in de Assets lijst op de map Scripts. Klik met de rechter muisknop op het lege gedeelte van de Scripts. Kies Create en klik op C# Script. In de Scripts map verschijnt een nieuwe script. Wijzig de naam in Ball. Open het script door erop te dubbelklikken of naar rechtsboven te gaan en te klikken op Open….

In de editor ziet u onderstaande geraamte:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Ball : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }
```

```
    // Update is called once per frame
    void Update () {

    }
}
```

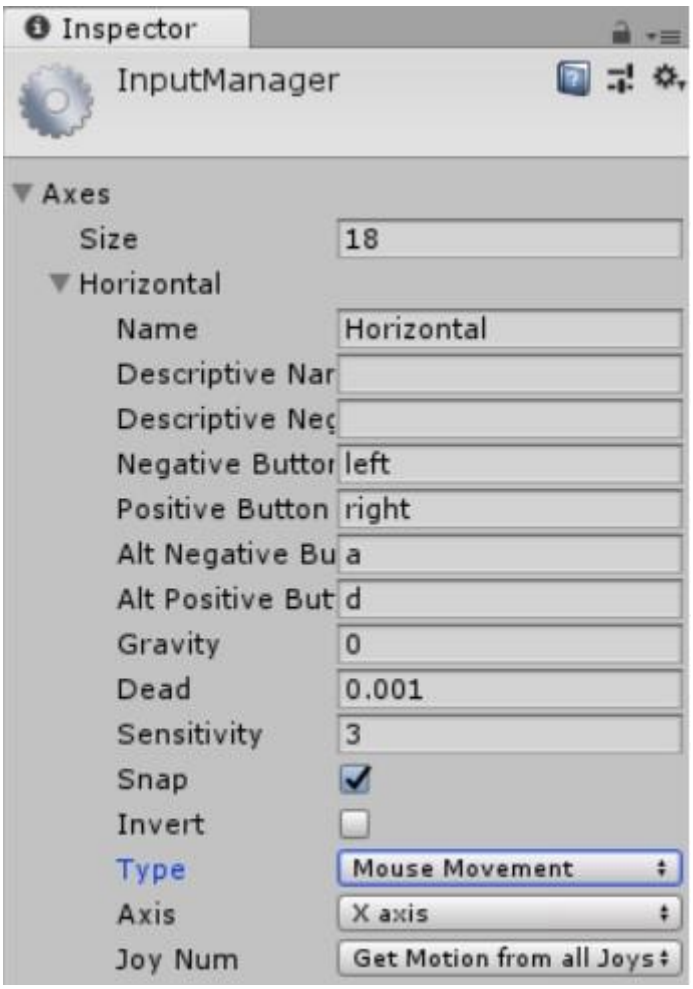
We hebben waarden nodig voor de balsnelheid en het bepalen of de bal in beweging is of niet. Typ onderstaande declaraties boven de Start() functie:

```
    private float speed = 0f;
    private bool allowStuff = false;
```

In de Engelstalige tutorial wordt de Update() functie niet gebruikt. Ik gebruik wel een update functie: de FixedUpdate() functie. Wijzig Update() in FixedUpdate() en typ onderstaande code in de functie:

```
        if (Input.GetMouseButtonDown(0) && !allowStuff)
        {
            allowStuff = true;
            speed = 65f;
            GetComponent<Rigidbody2D>().velocity = Vector2.up * speed;
        }
```

De invoer is zeer belangrijk. Voordat we op de methode GetMouseButtonDown(0) kunnen testen, moeten we de muis instellen in de InputManager. Sla het script op en ga terug naar het Unity project. Klik op het menu Edit, kies Project Settings en klik op Input. De Inspector opent een InputManager, zie de afbeelding. Het Type eigenschap kan ingesteld worden op toetsenbord of muis, of beide. Kies voor Mouse Movement.



Ga terug naar het script. Nu zal de methode wel werken. De waarde 0 is voor de linker muisknop. Er wordt ook gecontroleerd of alles al is ingesteld, dus of de bal al de snelheid heeft en een richting. Deze controle is belangrijk, want de instelling is maar éénmalig. De velocity hoeft niet bij elke FixedUpdate() aanroep ingesteld te worden.

Om de bal omhoog te laten schieten, wordt de Vector2.up maal de speed berekend voor de richting. Vector2.up is hetzelfde als Vector2(0,1).

Ga terug naar het Unity project en start Arkanoid. U zult merken dat de bal nog niet met het racket meebeweegt. Klik op de linker muisknop en blijf met het racket eronder staan. De bal zal telkens op en neer kaatsen. Er moet dus meer worden gedaan om een bal te krijgen die allerlei kanten op gaat.

Het Ball script uitbreiden

In de Engelstalige tutorial schiet de bal direct omhoog zodra de Arkanoid wordt gestart. In deze tutorial laat ik zien dat er een manier is om eerst de bal mee te laten bewegen met het racket, een foefje alsof de bal 'vastgeplakt' zit.

Voordat we de functie kunnen maken, moet het script weten met welk object de bal mee moet bewegen. We moeten dus aan het script het racket GameObject meegeven. Typ boven de speed declaratie onderstaande declaratie:

```
public GameObject racket;
```

De functie die voor het vastplakken moet zorgen heb ik BallSticky() genoemd. Typ onderstaande code onder de FixedUpdate() functie:

```
private void BallSticky()
{
     Vector2 vec = new Vector2();
     vec.x = racket.transform.position.x;
     vec.y = racket.transform.position.y + 8;

     transform.position = vec;
}
```

De x en y eigenschappen zijn alleen-lezen. U kunt niet direct waarden aan de position eigenschappen toekennen. Dit moet via een omweg, vandaar dat ik eerst een nieuwe Vector2 heb gemaakt en daaraan de racket position x en y heb toegekend. Het is wel mogelijk een object aan een ander object (van hetzelfde type) toe te kennen.

Merk op dat ik de transform component direct gebruik. Eerder heb ik verteld dat het transform component de enige vaste component is. Deze hoeft dan ook niet met GetComponent<>() opgegeven te worden.

Nu moeten we de functie aanroepen. Dat doen we in de Start() functie. Typ onderstaande aanroep in de Start() functie:

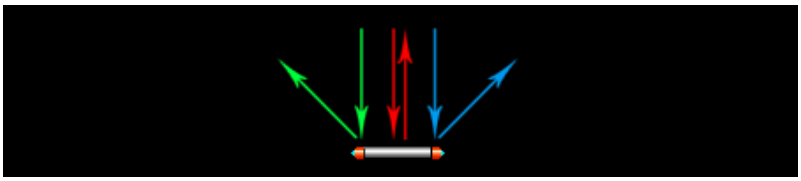
```
     BallSticky();
```

Als we het script opslaan en we het project starten, dan zal alsnog de bal niet met het racket meegaan. De reden is dat de aanroep alleen in de Start() functie niet voldoende is. We moeten de functie dus ook in de FixedUpdate() functie aanroepen, maar niet in de code van het if() statement.

We moeten controleren of de speed gelijk is aan 0, want dan alleen mag BallSticky() worden aangeroepen. Typ onderstaande code in de FixedUpdate() functie boven het andere if() statement:

```
     if (speed == 0)
     {
          BallSticky();
     }
```

Starten we nu Arkanoid dan zien we dat de bal meegaat met het racket, totdat er op de linker muisknop wordt geklikt. Maar wat de bal doet is echter nog niet wat we willen. De bal moet alle kanten op kunnen. Op de Engelstalige pagina laat men zien wat de bal moet doen als het tegen verschillende plaatsen van het racket komt. Hier laat ik de afbeelding ook zien.



De afbeelding is een schets met de gekleurde plaatsen van het racket, maar de bal kan ook ertussen komen. Elke plaats moet een andere hoek bepalen.

De hitFactor() functie
Elke plaats moet een nieuwe factor teruggeven. Het is dus een formule, een expressie, die de factor berekent, zodat de bal met een nieuwe velocity terugkaatst.

Typ in het script onderstaande hitFactor () functie:

```
float hitFactor(Vector2 ballPos, Vector2 racketPos, float racketWidth)
{
    return (ballPos.x - racketPos.x) / racketWidth;
}
```

De argumenten van de functie spreken voor zich. Om de factor te kunnen berekenen, moet eerst ball-Pos met racketPos worden verminderd en het resultaat daarvan worden gedeeld door racketWidth. Het is belangrijk de haakjes niet te vergeten, want Meneer Van Dalen Wacht Op Antwoord!

Waar halen we de parameterwaarden vandaan? Hoe kunnen we weten op welke plaats de bal op het racket kwam, want alleen daarmee kunnen we hitFactor() aanroepen.

Unity kent een event waarmee dat bepaald kan worden, OnCollisionEnter2D(). Het event heeft een parameter dat het object doorgeeft waarmee de bal in contact is gekomen. Hoewel we alleen het racket hebben, gaan we toch controleren of het doorgegeven object wel het racket is. Vergeet niet dat het event ook aangeroepen wordt als de bal tegen de borderranden komt.

We hoeven niet het racket GameObject te gebruiken om de naam te controleren, want de parameter heeft een name eigenschap vanuit het gameObject object van de parameter. Zie hieronder het event. Maak het event aan door te beginnen met *void On*. Een eventlijst zal verschijnen en kies het event OnCollisionEnter2D. Het geraamte van de event functie zal meteen worden aangemaakt. Hieronder ziet u de event functie met de code die erin moet komen:

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.name == "racket")
    {
        float x = hitFactor(transform.position,
                            collision.transform.position,
                            collision.collider.bounds.size.x);
        Vector2 dir = new Vector2(x, 1).normalized;
        GetComponent<Rigidbody2D>().velocity = dir * speed;
    }
}
```

Tip! In plaats van de events uit de eventlijst te kiezen, kunt u ook zelf het event typen met de parameter, zoals u ziet in de Engelstalige pagina waar de private ontbreekt en de parameter col wordt genoemd. Functies die gelijk met void of een typenaam beginnen zijn standaard private.

Het is niet moeilijk om de position op te vragen van de collision parameter, maar voor de lengte van het object is het anders. De lengte (size) is geen transform eigenschap, maar van de omvang van de collider, het object dat in contact kwam met de bal (zie in de code: *collider.bounds*).

Nu er een nieuwe x-factor is berekend, moeten we dit converteren naar een goede direction waarde. Om ervoor te zorgen dat dit alleen geldt voor de x-as, moet de y vector behouden blijven. Dit wordt gedaan door een 1 mee te geven, maar dan moet ook de eigenschap normalized worden gebruikt. Zouden we dat niet doen, maar direct de factor aan de velocity meegeven, dan zal het kaatsen niet goed werken. Onthoud dat de teruggeven x-factor een relatieve waarde is.

Sla het script op en ga terug naar Unity. Start Arkanoid op. Beweeg het racket en de bal gaat mee. Klik op de linker muisknop. De bal gaat nu nog recht omhoog, maar laat de bal nu neerkomen op een andere plaats van het racket, zoals op de hoek. U ziet dat de bal een andere richting gaat.

Mocht er iets niet goed gaan; hieronder laat ik het hele Ball script zien. Bekijk het en controleer het met het script dat u geschreven hebt.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Ball : MonoBehaviour {
    public GameObject racket;
    private float speed = 0f;
    private bool allowStuff = false;
    // Use this for initialization
    void Start () {
        BallSticky();
      }
    void FixedUpdate()
    {
        if (speed == 0)
        {
            BallSticky();
        }

        if (Input.GetMouseButtonDown(0) && !allowStuff)
        {
            allowStuff = true;
            speed = 65f;
            GetComponent<Rigidbody2D>().velocity = Vector2.up * speed;
        }

    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "racket")
        {
            float x = hitFactor(transform.position,
                                collision.transform.position,
                                collision.collider.bounds.size.x);
            Vector2 dir = new Vector2(x, 1).normalized;
            GetComponent<Rigidbody2D>().velocity = dir * speed;
        }
    }
    float hitFactor(Vector2 ballPos, Vector2 racketPos, float racketWidth)
    {
        return (ballPos.x - racketPos.x) / racketWidth;
    }
    private void BallSticky()
    {
        Vector2 vec = new Vector2();
        vec.x = racket.transform.position.x;
        vec.y = racket.transform.position.y + 8;

        transform.position = vec;
    }
}
```

U zult wel zien dat er private staat bij BallSticky(), maar niet bij hitFactor. Zoals ik bij de tip op de vorige pagina heb verteld, zijn ze echter beide private, dus het maakt niet uit.

Volgende keer
De volgende keer, deel 3, gaan we de bricks GameObjecten plaatsen en zorgen dat de bal ze laat verdwijnen als ze met de bal in contact komen.

Maar ik ga verder, want er komt een onderwerp prefabs bij. We gaan dan de bricks in prefabs opslaan en de prefab als GameObject gebruiken.

Ook zullen we verschillende soorten bricks gaan maken die hun eigen hits eigenschap hebben en een brick die indestructable is, dat wil zeggen, die dan niet verwijderd kan worden. Voor deze doeleinden zal het event OnCollisionEnter2D() belangrijk zijn, want alleen hiermee kunnen we controleren met welke brick de bal in contact kwam.

Elke brick moet zoveel punten kunnen geven. We gaan bovenaan een score tekst weergeven met de canvas. Helaas werken we met scripts, er is geen algemene module om deze gegevens in te kunnen bewaren. We kunnen wel een script maken dat static is. Statische scripts kunnen zichzelf aanroepen met eigenschappen en functies. Hoe dat allemaal werkt ziet u ook de volgende keer.

C – Geheugen alloceren en vrijmaken.

De programmeertaal C is een mooie taal om te leren, en te programmeren. C werkt het beste voor console programma's. De taal is cryptisch, maar werkt daardoor zeer snel.

Wie C niet (goed) kent, zal het moeilijk vinden om de code te begrijpen. Zelf vind ik het ook niet makkelijk maar omdat ik er wat van geleerd heb op de MTS, kan ik wel wat laten zien. Hoe kunnen we bijvoorbeeld met tekst werken en hoe maken we dynamische arrays?

De meeste programmeertalen hebben een string type, een manier om meerdere tekens in een variabele op te kunnen slaan. C kent echter geen string type. Er zijn twee mogelijkheden om meer tekens te kunnen bewaren: in een array of doormiddel van een pointer.

Een array is duidelijk, net als in andere programmeertalen:

```
char naam[20];
```

We hebben nu een naam array met 20 elementen. We kunnen dus 20 tekens in opslaan. Wel van element 0 tot en met 19. De array kunnen we nu vullen met tekens:

```
naam[0] = "M";
naam[1] = "a";
naam[2] = "r";
...
naam[19] = "e";
naam[20] = "!";     // kan niet, want element 19 is de hoogste
```

In plaats van het teken voor teken te doen, kunnen we ook direct een naam toekennen, zoals in een declaratie:

```
char naam[] = "Marco Kurvers";
```

Met de string functies uit de header string.h kunnen we deze array kopiëren naar een andere array, met wel al de gegeven aantal elementen:

```
char naam2[20];
strcpy(naam2, naam);
```

Onderstaand programma geeft een volledig voorbeeld:

```
/* strcpy voorbeeld */
#include <stdio.h>
#include <string.h>

int main ()
{
     char str1[] = "Simpel string";
     char str2[40];
     char str3[40];
     strcpy(str2, str1);
     strcpy(str3, "Succes gekopieerd");
     printf("str1: %s\nstr2: %s\nstr3: %s\n", str1, str2, str3);
     return 0;
}
```

Maar wat nu als we het aantal elementen niet weten, of later willen wijzigen? In de header string.h is er nog zo'n kopieerfunctie: strncpy(). Met deze functie moeten we een derde waarde meegeven voor de lengte van het array. Onderstaand voorbeeld laat zien wat u moet doen voor deze mogelijkheid:

```
/* strncpy voorbeeld */
#include <stdio.h>
#include <string.h>

int main ()
{
     char str1[] = "Te zijn of niet te worden";
     char str2[40];
     char str3[40];

     /* kopiëren naar formaat buffer (overloop veilig): */
     strncpy(str2, str1, sizeof(str2) );

     /* gedeeltelijke kopie (alleen 5 tekens): */
     strncpy(str3, str2, 5);
     str3[5] = '\0';   /* null-teken handmatig toegevoegd */

     puts(str1);
     puts(str2);
     puts(str3);

     return 0;
}
```

De eerste strncpy() aanroep werkt hetzelfde als de strcpy() functie, omdat de lengte 40 blijft, maar voor de tweede aanroep is het een ander geval. Er wordt een kopie gemaakt met maar 5 tekens van de inhoud van str2 naar str3. Het null-teken moet dan ook handmatig toegevoegd worden om de rest af te kappen.

Merk op dat u met puts() een array kunt opgeven om de inhoud weer te geven.

! Het gebruik van de functie strncpy() is onveilig. Een foutje kan snel gemaakt zijn als de buffer te klein is of vergeten is het '\0' teken aan het einde te zetten. Zo'n foutje kan grote gevolgen hebben als een hacker uw programma kan misbruiken om ongewenste code uit te voeren en dan allerlei schade kan veroorzaken.

Pointers
Is er een verschil tussen pointers en arrays? Ja, want onderstaande declaraties zijn het volgende:

```
char *array = "Een goede zaak over muziek";
char array2[] = "Een goede zaak over muziek";
```

Vastgesteld in het geheugen, ziet de eerste regel er uit als:

```
+-------+          +------------------------------+
| array |     →    | "Een goede zaak over muziek" |
+-------+          +------------------------------+
```

en gewoon dit voor de array2 regel:

```
+-----------------------------+
| "Een goede zaak over muziek" |
+-----------------------------+
```

Een pointer kan gewijzigd worden, een array niet:

```
array = "Een andere zaak";  /* in orde */
array2 = "Een andere zaak"; /* geeft compileer fout */
```

Hebben we nog geen inhoud bij een pointer maar willen we dat pas later doen, dan moeten we door middel van een aantal opgegeven elementen zelf het geheugen reserveren, of ook wel gezegd *alloceren*. Hieronder een voorbeeld:

```
/* calloc voorbeeld */
#include <stdio.h>      /* printf, scanf, NULL */
#include <stdlib.h>     /* calloc, exit, free */

int main()
{
     int i, n;
     int *pData;
     printf("Aantal nummers dat moet worden ingevoerd: ");
     scanf("%d", &i);
     pData = (int*) calloc(i, sizeof(int));
     if (pData == NULL) exit(1);
     for (n = 0; n < i; n++)
     {
          printf("Nummer invoeren #%d: ", n + 1);
          scanf("%d", &pData[n]);
     }
     printf("U hebt ingevoerd: ");
     for (n = 0; n < i; n++) printf("%d ", pData[n]);
     free(pData);
     return 0;
}
```

Merk op dat bij scanf() altijd de adres operator wordt gebruikt, ook al is het geen pointer. Dat komt omdat er om invoer wordt gevraagd en wat ingevoerd wordt, doorgegeven moet worden. Vergeet niet dat het bij andere functies net zo werkt. Waarden die doorgegeven moeten worden, gaan via het geheugen waarnaar verwezen wordt en niet naar de inhoud. Hieronder een voorbeeld:

```
functieTest (&waarde);   →   void functieTest(int *w);
```

Krijgt argument w een nieuwe waarde, dan zal parameter waarde de nieuwe waarde ontvangen, maar dan wel via het geheugenadres.

Excel – VBA leren.

Men zegt wel eens: "Wie VBA wil leren moet Visual Basic 6.0 kennen". Zelf vind ik dat niet, want VBA en VB6.0 lijken sterk op elkaar, en dat is ook zo. Op één ding na: VBA heeft alleen een objectlibrary waarmee we alleen objecten kunnen gebruiken voor het Office programma waarmee we werken, zoals Excel. Een voorbeeld als Word; VBA in Word heeft een Document object. Dit object kunnen we in Excel niet terugvinden en evenmin het Sheets() collectie object voor de werkbladen die we dus niet in Word terug kunnen vinden. Dat vaak wordt gezegd Office VBA doet men denken dat VBA één objectlibrary is van het hele Office pakket, maar we weten nu dat het niet zo is.

Toch is het mogelijk om de VBA libraries samen te gebruiken. We kunnen in Word een referentie maken naar de VBA library in Excel. Maar door eerst een referentie te kiezen is het mogelijk, anders niet.

De VBA mogelijkheden in Word zijn slecht te noemen. Zelf heb ik VBA uitgeprobeerd, maar Excel heeft gewoon tal van mogelijkheden. Denk maar aan de functies in de categorielijst, die functies zijn ook in VBA te gebruiken en omdat VBA meer flexibiliteit heeft, kunnen we deze functies in een eigen geschreven functie bundelen en uitvoeren.

De werkbladen
In VBA kunnen we achterhalen hoeveel werkbladen er geopend zijn. Er zijn twee manieren:

```
ActiveWorkbook.Worksheets.Count
ActiveWorkbook.Sheets.Count
```

Het object ActiveWorkbook is optioneel, maar kan handig zijn wanneer u meerdere werkboeken open hebt staan. Denk erom dat elk werkboek een apart project is, maar wel in één solution staat. We gaan daarom straks wat meer ermee oefenen.

Er zijn twee manieren, zoals u ziet, maar u mag zelf kiezen welke manier het beste is. Ik werk altijd met het Sheets object.

Kijken we naar het project in VBA, dan zien we ook Blad1 staan, en zo. We kunnen daarom ook zeggen:

```
Print Blad1.Rows.Count
 1048576
```

Wáááááát? Heb ik er zoveel aangemaakt?!

Nee, VBA toont zoveel rijen omdat het werkblad nou eenmaal er al zoveel heeft. Dat wil niet zeggen dat ook al die rijen worden opgeslagen als u uw werkboek opslaat. Dit geldt net zo voor de kolommen.

Willen we dit opvragen met Sheets(), dan moeten we de index meegeven van het werkblad waarmee gewerkt wordt, bijvoorbeeld werkblad 1:

```
Print Sheets(1).Rows.Count
```

Maar als we nou niet weten welke index bij welk werkblad hoort? We kunnen het werkblad waarmee we werken opvragen en in het Sheets() object gebruiken als index:

```
Print Sheets(ActiveSheet.Index).Rows.Count
```

We kunnen dan achterhalen wat de naam is van het werkblad:

```
Print Sheets(ActiveSheet.Index).Name
Blad1
```

Wat gebeurt er als we de naam van Blad1 wijzigen in TestBlad? Laten we dat eens doen. Wijzig de naam en ga dan terug naar VBA. Loop met de cursor naar dezelfde Print regel en ga aan het einde staan, achter Name. Druk op Enter.

Nu verschijnt de naam TestBlad in plaats van Blad1.

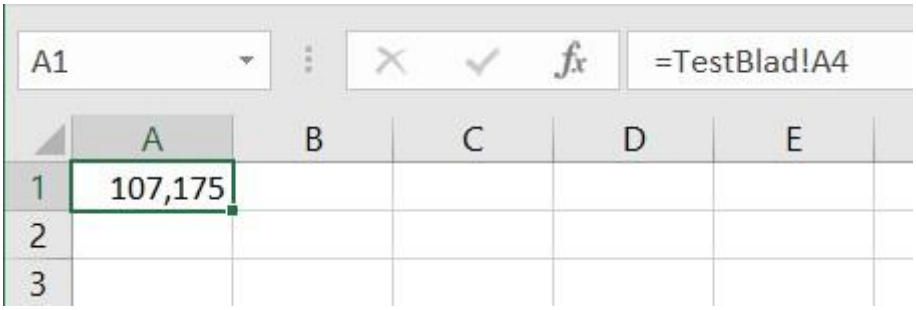
Dan zouden we zeker nu kunnen zeggen: `TestBlad.Rows.Count`. Echter verschijnt er een foutmelding. De naam wordt door VBA niet als een object herkent. We moeten nog steeds Blad1 gebruiken, want die is wel geldig als objectnaam.

Toch is het mogelijk om de objectnaam te wijzigen. Kijk in de projectlijst. Daar ziet u Blad1 (TestBlad) staan. Klik erop. In de eigenschappenlijst ziet u (Name). Dit is de hoofdeigenschap van het object. Wijzig de naam in TestBlad en probeer nog eens het aantal rijen op te vragen met TestBlad. Nu zal er geen foutmelding verschijnen en zal het net zo werken als eerder met Blad1 werd gedaan. Ook nu zal Name van het Sheets() object geen Blad1 meer weergeven, maar TestBlad.

Het is niet mogelijk objectnamen te wijzigen in Excel zelf. U weet nu dat gewijzigde werkbladnamen beschrijvingen zijn en geen objectnamen.

Verwijzingen in detail
Ik heb het vast wel eens over verwijzingen gehad. Er kunnen naar andere cellen verwezen worden om de inhoud te kopiëren, maar dan zo, dat de wijzigingen doorgevoerd worden. Handig bij het maken van jaarverslagen en boekhoudingen.

Terug naar het TestBlad. Laten we eens een nieuw blad erbij gebruiken, Blad2. Onderstaande afbeelding laat een waarde zien die uit TestBlad komt.



De waarde is een resultaat van een formule die in cel A4 staat van TestBlad. Om dit ook te proberen, doet u het volgende:

- Zorg er voor dat u een waarde in een cel heeft, bijvoorbeeld in Blad1;
- Ga naar een ander blad, bijvoorbeeld naar Blad2;
- Typ in een cel het = teken;
- Ga naar Blad1 en klik op de cel die de waarde heeft;
- In de formulebalk verschijnt een verwijzing; ga niet zelf terug naar Blad2;
- Druk op Enter en u ziet dat Blad2 verschijnt met de waarde in de cel.

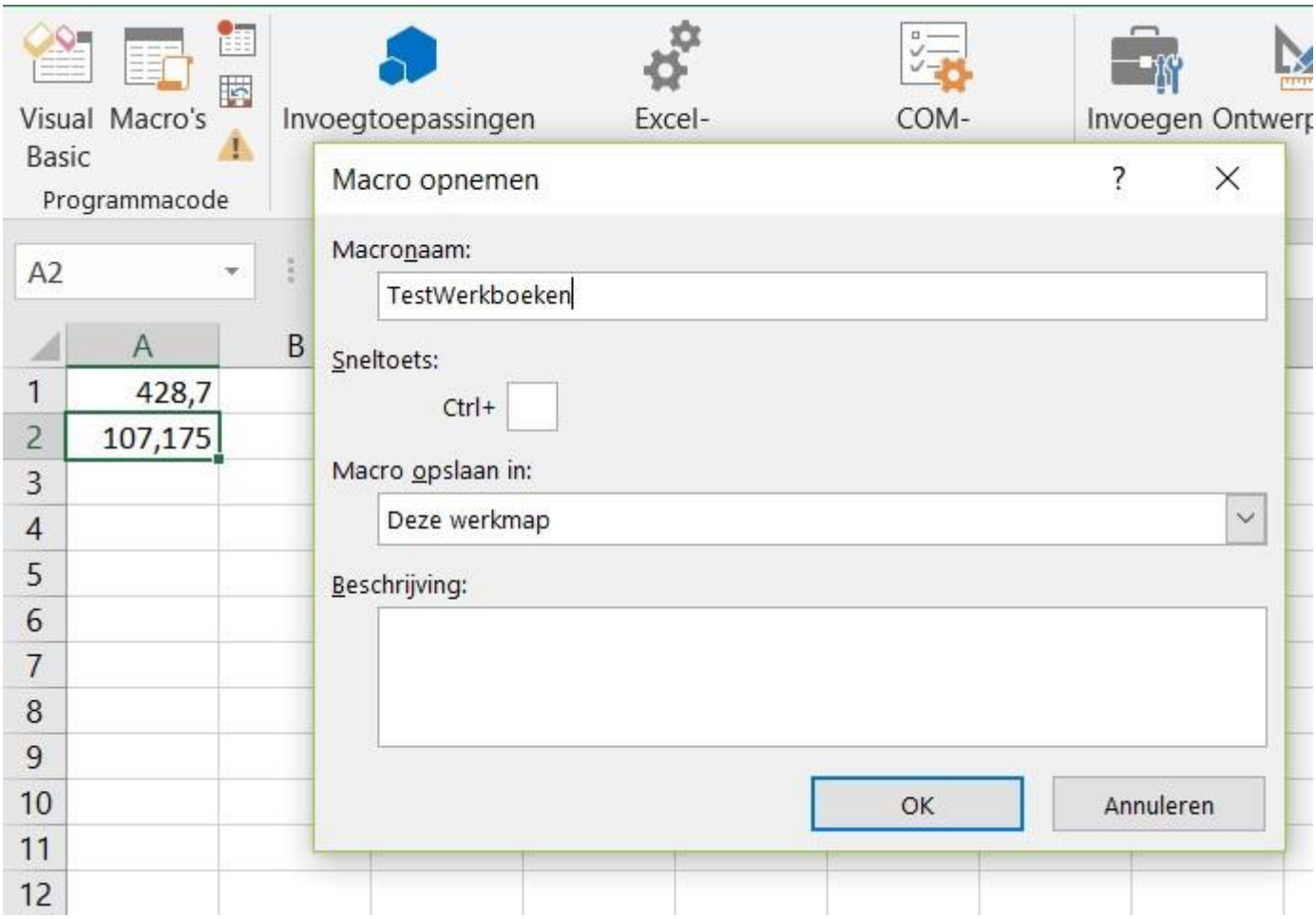
In plaats van met de muis naar een werkblad te gaan, mag u de verwijzing ook handmatig intypen.

Naar cellen verwijzen in één werkboek kan, maar het is ook mogelijk naar andere werkboeken te verwijzen. Hoe doen we dat? Eigenlijk op dezelfde manier, maar de verwijzingsregel in de formulebalk zal er anders uit komen te zien. Ook gaan we de verwijzing opnemen in een macro, want hoe reageert VBA als we met meer werkboeken gaan werken?

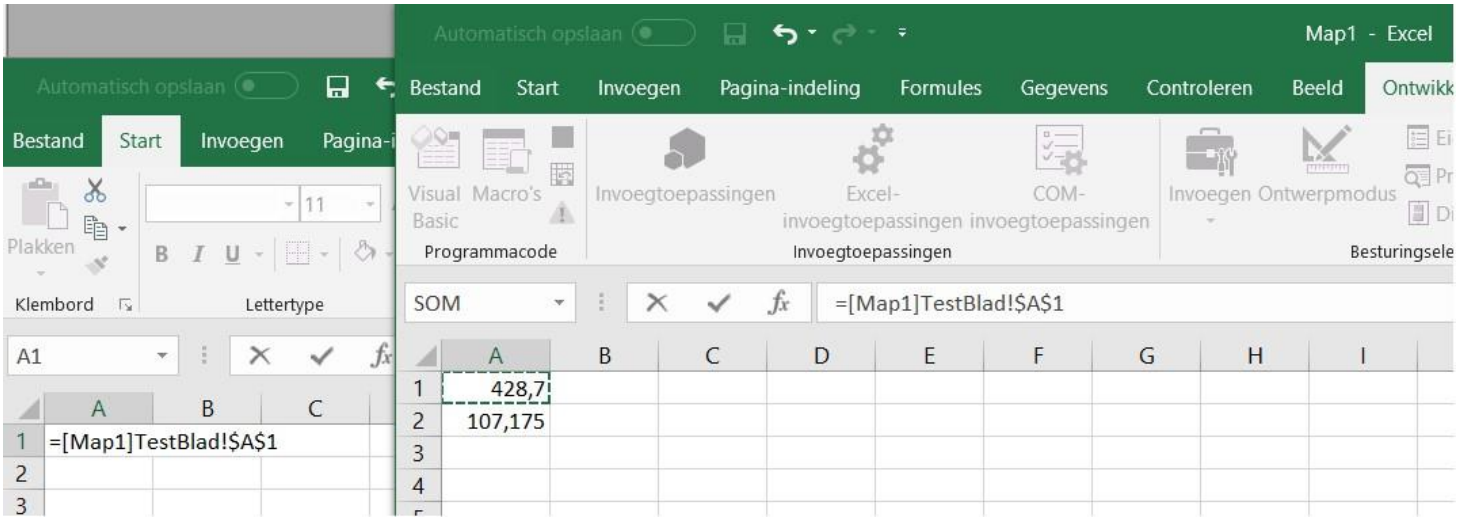
Ga naar Bestand en kies Nieuw. Klik op een leeg werkboek. Map2 zal verschijnen. Ga terug naar Map1. Klik dan op de macro opnemen knop op het lint. Voor Office 2016 en Office365 gebruikers ziet de knop er zo uit:

Macro opnemen knop

Onderstaand afbeelding verschijnt:



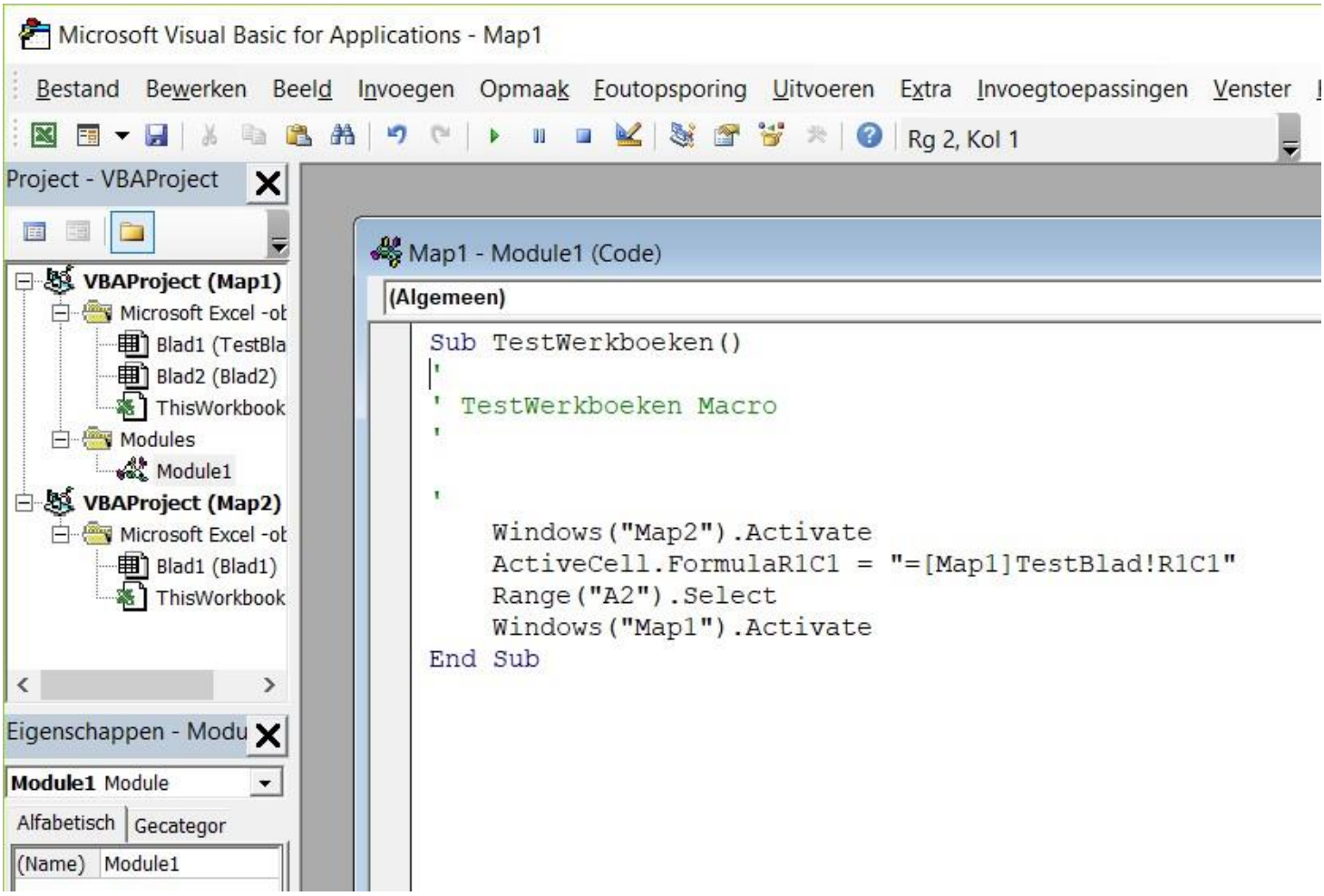
Geef de Macronaam een duidelijke naam, bijvoorbeeld TestWerkboeken en klik op OK.
Ga terug naar Map2. Zie onderstaand voorbeeld en ga verder met de stappen:



Typ de verwijzing in Map2 in cel A1 als volgt in:

- Typ een = teken;
- Ga met de muis terug naar Map1;
- Ga naar het werkblad met de cel met een waarde en klik die aan;
- Nu ziet u ongeveer bovenstaande afbeelding;
- Echter bent u nog niet klaar, want de verwijzing lijkt in zichzelf te staan, namelijk Map1 (zie de stippellijn om de cel);
- Druk op Enter en u keert terug (vanzelf) naar Map2. Nu wordt er met dezelfde verwijzing wel verwezen naar de actieve cel in Map2.

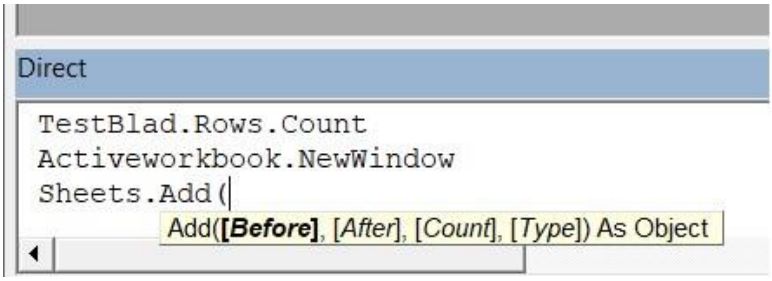
Ga weer terug naar Map1 en stop het opnemen van de macro. We gaan eens kijken wat VBA voor ons gemaakt heeft.



U ziet dat er nu twee projecten zijn en de macro opgenomen is in projectmap 1. Toch kunnen we in VBA verwijzen naar een andere projectmap. We kunnen dat alleen doen met het Windows() object. Net als bij Sheets() is Windows() dus ook een collection.

Toevoegen en invoegen

Wat we met Sheets() kunnen doen, kan met Windows() niet; invoegen. Het Windows object heeft geen Add() methode, wat Sheets wel heeft. De afbeelding laat code zien in de directe modus.



Stel dat Map1 uw actieve werkboek is. Er is een NewWindow methode die een werkboek toevoegt, dat dan doet denken aan bijvoorbeeld Map3.

Helaas, dat gebeurt echter niet. Het blijft Map1, maar dan gedeeld in een ander nummer: Map1 – 2. Roept u hem weer aan, dan wordt het Map1 – 3, enzovoorts. Wijzigt u een waarde in een cel van Map1 – 3, dan zullen alle andere deelmappen ook die waarde in dezelfde cel krijgen. Wat precies het nut hiervan is, weet alleen degene die hier gebruik van maakt.

Werkbladen toevoegen in VBA
U ziet in bovenstaande afbeelding dat het Sheets object een Add() methode heeft. De parameters zijn als volgt:

Before	Voegt één werkblad of meer werkbladen voor het opgegeven werkblad toe.
After	Voegt één werkblad of meer werkbladen na het opgegeven werkblad toe.
Count	Het aantal werkbladen die toegevoegd moeten worden.
Type	Geef hier het werkbladtype.

Meer informatie over de verschillende werkbladtypes, zie de internetlink:
https://msdn.microsoft.com/enus/library/office/microsoft.office.interop.excel.worksheets.add.aspx

Sheets kent nog een Add() methode: Add2(). Met deze methode kan er ook een nieuw layout worden toegevoegd. De layout is een apart object die dan als parameter meegegeven kan worden.

Amiga AMOS: IFF plaatjes laden.

Wie de Commodore Amiga nog kent, weet ook van Amiga BASIC. De BASIC versie die anders in elkaar zit dan de andere Commodore BASIC versies, zoals de VIC 20, de 64 en de 128. De Commodore 128 BASIC 7.0 was de grootste BASIC programmeertaal met commando's en functies. De meeste daarvan bestaan nog steeds.

BASIC? Dat nooit!
Het programmeren lijkt moeilijk en dat is vaak terecht. Computers zijn machines en kennen geen fouten die wij maken. Ze kennen geen pardon als een programma met fouten toch uitgevoerd wordt en kan crashen.
Maar het kan ook anders. Waarom zou de mens de taal van de computer moeten leren? Laat de computer maar een stap dichter naar de mens toe komen. Programmeertalen zoals Logo, Turtle en AMOS doen dat. Een grafisch IFF plaatje op het scherm zetten gebeurt in AMOS met 'LOAD IFF'. Dat lijkt al veel op standaard Engels.
In machinetaal kost hetzelfde karwei honderden regels cryptische programmeertekst. Wie geen masochistische neigingen heeft en bovendien tevreden is met de geboden snelheid, zal een blijvende vriend zijn van AMOS. Helaas geniet BASIC, en daarmee ook AMOS, een slechte reputatie. Immers kan iedereen, ook de grootste kluns, meteen aan de slag in BASIC. Maar slordige programmeurs zullen er altijd zijn, ook onder de machinecode-kenners. Dat heeft niets met de taal te maken.

En wij dan?
Alle begin is moeilijk, zeker wat betreft programmeren. Wie aan de listings, die als laatst pas komen, echt geen touw kan vastknopen hoeft nog niet te wanhopen. De programma's werken ook in Easy AMOS, een variant van deze oorspronkelijke Amiga taal. Easy AMOS biedt minder mogelijkheden, zoals minder commando's, maar voor de echte beginner is het een 'must'.

Amiga AMOS nu: XAMOS
Wie nieuwsgierig is kan AMOS uitproberen. Er is een versie, herschreven in C++, die XAMOS wordt genoemd. XAMOS is een cross-platform re-implementatie van AMOS BASIC en is te downloaden. XAMOS is onder andere te vinden op https://sourceforge.net/projects/xamos/.

Helaas wilde het programma bij mij niet werken. Werkt het bij u wel of heeft u nog de echte Amiga, dan kunt u proberen om de code uit te voeren. Het grote programma dat later komt zorgt voor een snellere manier om IFF plaatjes in te laden en effecten te geven. En denk eraan: gebruik niet Amiga BASIC. Voor de listings heeft u AMOS nodig, want Amiga BASIC en Amiga AMOS zijn niet hetzelfde.

Het bestand libstdc++-6.dll werd niet gevonden. Ga naar:
https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/rubenvb/gcc-dw2-4.6-release/

Download het bestand i686-w64-mingw32-gcc-dw2-4.6.3-2-release-win32_rubenvb.7z.
Het is een 7z zip bestand. Als het met de standaard Windows uitpaksysteem niet lukt, download dan gratis het uitpakprogramma 7zip. Zet het bestand libstdc++-6.dll uit de bin folder bij Xamos.exe, daarna werkt het bij mij toch wel.

IFF THEN ELSE
IFF betekent Interchangeable File Format en betekent niets anders dan: grafisch plaatje. Wie plaatjes tekent of verzamelt wil ze natuurlijk ook op het scherm kunnen zien. Onderstaand programma is een zeer kort programma dat een IFF plaatje op het scherm zet.

```
Load Iff Fsel$("", "", ""), 1
Wait Key
```

In de eerste regel staat LOAD IFF om aan te geven dat AMOS een IFF plaatje moet laden. Normaal staat daarachter meteen de naam van het bestand. Maar door de FSEL$() functie te geven, zal AMOS een zogenaamde File Selector laten zien. Met de muis kunnen we door mappen bladeren en op een bestand klikken. Dat bestand zal AMOS gebruiken voor het LOAD IFF commando. Aan het einde van de regel staat nog een ',1' om aan te geven dat AMOS scherm nummer 1 moet openen om daar het IFF bestand op te laten zien. Als er nog geen Screen 1 geopend is, zal AMOS dat doen. Wanneer er wel al een Screen 1 aanwezig is, zal AMOS dat scherm sluiten om meteen daarna Screen 1 weer te openen en het IFF plaatje erin plaatsen.
De tweede regel zorgt ervoor dat de Amiga wacht totdat we een willekeurige toets indrukken voordat het programma eindigt. Ziezo, een complete IFF file viewer met ingebouwde file selector na 1 minuut programmeren.

De AM_SlideShow
Even een IFF-je laden in AMOS stelt blijkbaar weinig voor. Echt spannend wordt het pas als we effecten toevoegen zodat de plaatjes op speciale manieren op het scherm verschijnen en er weer van verdwijnen. Om bijvoorbeeld het plaatje Mountain.IFF in te laden van de diskette Plaatjes1: met verschijneffect 4 en verdwijneffect 5, typen we:

```
Plaatjes1:Mountain.IFF,4,5
```

Er staat dus steeds een bestandsnaam gevolgd door een komma, een getal, weer een komma en weer een getal. Extra spaties mogen niet voorkomen. Na iedere regel drukken we natuurlijk de ENTER toets in om naar de volgende regel te gaan, maar ook achter de laatste regel moeten we de ENTER toets één keer indrukken, anders raakt AMOS bij het inlezen van de lijst helemaal van slag bij de laatste regel. Overigens hoeven de bestandsnamen van de IFF plaatjes niet per se te eindigen op '.IFF'.
Onderstaande listing is een voorbeeld van een AM_SlideShow.Lijst. Na het intypen van de lijst bewaren we deze als ASCII (de meeste editors doen dat automatisch). Daarbij geven we het bestand de naam 'AM_SlideShow.Lijst'.

```
Plaatjes1:Mountain.IFF,4,5
Plaatjes1:River.IFF,1,3
Plaatjes1:Moon.IFF,6,1
Plaatjes1:Waterfall.IFF,3,9
```

De kern van de zaak
Het programma zal het bestand AM_SlideShow.Lijst openen en regel voor regel inlezen. De verschijneffecten staan geprogrammeerd in de procedures Verschijn1, Verschijn2, enzovoort, en zo ook bij de Verdwijn procedures. Het programma berust op een simpel principe:

1. Laad een IFF plaatje in op Screen 1, maar toon het nog niet.
2. Gebruik een verschijneffect.
3. Wacht een paar seconden.
4. Gebruik een verdwijneffect.
5. Ga terug naar stap 1.

De code van het complete programma is doorspekt met commentaar om zoveel mogelijk uitleg te geven over de gebruikte commando's en functies.

Hide, fade en scroll
Verderop in de code staan de procedures. Bij iedere verschijnprocedure hoort een verdwijnprocedure die hetzelfde effect gebruikt. Zo haalt Verschijn3 de afbeelding in een flits tevoorschijn. Verdwijn3 laat het IFF plaatje ook weer in een flits verdwijnen. Alle procedures gaan ervan uit dat in Screen 1 een afbeelding staat. Hoeveel kleuren en welke afmetingen die afbeelding heeft, speelt geen rol. Door de-

ze logische opbouw zijn de procedures uitermate geschikt om in eigen programma's opnieuw te gebruiken. De in totaal zes effecten zien er als volgt uit:

1. Plaatje verschijnt/verdwijnt ineens
2. Plaatje verschijnt/verdwijnt langzaam (fade)
3. Plaatje verschijnt/verdwijnt in een witte flits
4. Plaatje wordt van boven naar beneden (on)zichtbaar
5. Plaatje schuift van links naar rechts in(uit)beeld
6. Plaatje verschijnt/verdwijnt in etappes (lamellen effect)

Bij iedere procedure staat uitgebreid commentaar. Voor de volledigheid lopen we de commando's en functies in de procedures even langs.

- 'Verschijn1' gebruikt 'Screen Show' om een van tevoren met 'Screen Hide' onzichtbaar gemaakt scherm weer zichtbaar te maken. 'Verdwijn1' gebruikt 'Screen Close' om het scherm niet alleen leeg te maken, maar ook meteen te sluiten.
- 'Verschijn2' opent een piepklein tweede scherm. Om daarin het palet van het IFF bestand te bewaren. Het palet geeft aan welke kleuren het IFF bestand gebruikt. Daarna zetten we alle 32 kleurregisters van het IFF-je op zwart. Met het Fade commando haalt AMOS langzaam de oorspronkelijke kleuren weer terug uit de paletgegevens van het tweede scherm. 'Verdwijn2' heeft het gemakkelijker. Daarin staat alleen een Fade commando dat alle kleuren van het IFF-je langzaam in zwart verandert.
- 'Verschijn3' gebruikt bijna hetzelfde trucje als 'Verdwijn2'. Alleen worden de kleurregisters van het IFF plaatje niet op zwart maar op wit gezet. Van daaruit gaan de kleuren weer terug naar de oorspronkelijke stand. Dat geeft het effect van een witte flits. Ook 'Verdwijn3' heeft het gemakkelijk. Alle kleuren verdwijnen weer met Fade naar één kleur; ditmaal wederom wit in plaats van zwart.
- 'Verschijn4' zit al wat moeilijker in elkaar. We bedekken het IFF plaatje met een groot zwart scherm. Door het zwarte scherm met 'Screen Display' langzaam te laten zakken, wordt het plaatje van boven naar beneden zichtbaar. Op dezelfde manier trekt 'Verdwijn4' het zwarte scherm weer omhoog.
- 'Verschijn5' maakt een kopie van het plaatje. Daartoe wordt een Screen 2 geopend dat voldoende groot is en alle kleurregisters gebruikt. Met 'Screen Copy' kopiëren we de afbeelding naar Screen 2. We maken Screen 1 leeg. In een For ... Next lus halen we de afbeelding lijntje voor lijntje terug van Screen 2 naar Screen 1. Op dezelfde manier tekent 'Verdwijn5' allemaal evenwijdige lijnen om de afbeelding weer te verbergen.
- 'Verschijn6' is een variant op 'Verschijn5'. Alweer wordt een kopie van het IFF-je terug gekopieerd naar Screen 1. Alleen gebeurt dat nu in vier stappen. In iedere stap worden, verdeeld over de gehele breedte van het scherm, verticale lijnen verplaatst. Daardoor ontstaat het effect van lamellen die langzaam open gaan. Tenslotte tekent 'Verdwijn6' in vier stappen voldoende verticale lijnen om de afbeelding weer te verbergen.

Op de volgende pagina ziet u de listing. Het is een hele grote, dus mooi om de code eens door te kijken en als het kan om ook eens uit te proberen.
Wie goed met programmeren om kan gaan, zou kunnen proberen om de AMOS code te converteren naar een andere programmeertaal. Liberty BASIC en Pascal lijken mij het meest geschikt, maar omdat Liberty BASIC goede grafische mogelijkheden heeft, is die programmeertaal de beste keus.

Heeft iemand de Commodore 128? Mooi, want de AMOS code lijkt ook daar veel op. Gebruik dan wel in plaats van het Screen commando het Graphic commando en vergeet ook niet dat Ink vervangen moet worden met Color. Het kleurregister moet opgegeven worden als eerste parameter van het Draw commando.

```
' AM_SlideShow Versie 1.0
' Sluit het standaard aanwezige scherm 0
Screen Close 0
' Test of het AM_SlideShow.Lijst bestand aanwezig is, zo niet stop programma
TEST=Exist('AM_SlideShow.Lijst')
If TEST=0
      Screen Open 0,320,200,4,Lowres
      Print "Fout: AM_SlideShow.Lijst is niet gevonden!"
      Print
      Print "Druk willekeurige toets"
      Wait Key
      Screen Close 0
      End
End If
'
' Open de lijst waar de beschrijving in staat
Open In 1,'AM_SlideShow.Lijst'
Set Input 10,-1
'
' Hier begint de herhalingslus die doorgaat totdat het einde van de lijst is bereikt
Repeat
      '
      ' Lees een regel uit het AM_SlideShow.Lijst bestand in
      Input #1,NAAM$,VERSCHIJN_NR,VERDWIJN_NR
      '
      ' Laad een IFF bestand in Screen 1 maar laat het nog niet zien
      Load Iff NAAM$,1
      Screen Hide 1
      ' Verberg de muispointer
      Hide On
      ' Verberg de knipperende cursor
      Flash Off
      '
      ' Gebruik een verschijneffect
      On VERSCHIJN_NR Proc VERSCHIJN1,VERSCHIJN2,VERSCHIJN3,VERSCHIJN4,VERSCHIJN5, →
      VERSCHIJN6
      '
      ' Laat het IFF plaatje 5 seconden (250x 1/50 seconde) op scherm staan
      Wait 250
      '
      ' Gebruik een verdwijneffect
      On VERDWIJN_NR Proc VERDWIJN1,VERDWIJN2,VERDWIJN3,VERDWIJN4,VERDWIJN5,VERDWIJN6
      '
Until Eof(1)
'
' De procedures
Procedure VERSCHIJN1
      ' Plaatje verschijnt ineens
      Screen Show 1
End Proc
'
Procedure VERSCHIJN2
      ' Plaatje verschijnt langzaam (fade)
      ' Open een heel klein schermpje screen 2 om daarin het palet van screen 1 te →
      bewaren
      Screen Open 2,32,32,4096,Lowres
      ' Laat screen 2 niet zien
      Screen Hide 2
      ' Flash moet op uit staan, anders knippert kleurregister 3 van screen 2
      Flash Off
      Get Palette 1
      ' Maak screen 1 zwart door alle 32 kleurenregisters op zwart te zetten
      Screen 1
      Palette 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      Wait 15
      ' Laat het inmiddels zwarte screen 1 zien
```

```
      Screen Show 1
      ' Verander het zwarte palet langzaam in het oorspronkelijke palet
      Fade 7 To 2
      ' Wacht totdat de Fade opdracht klaar is
      ' Bij Fade x hoort een Wait x*15
      Wait 105
      ' Screen 2 is nu niet meer nodig
      Screen Close 2
End Proc
'
Procedure VERSCHIJN3
      ' Het plaatje verschijnt in een flits en open een hele kleine screen 2
      ' om daar het palet in te bewaren
      Screen Open 2,32,32,4096,Lowres
      Screen Hide 2
      Flash Off
      Get Palette 1
      ' Verander alle kleuren van het palet van Screen 1 in wit ($FFF)
      Screen 1
      Palette $FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF, →
      $FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF, →
      $FFF,$FFF,$FFF
      ' Indien Screen 1 een HAM plaatje bevat, zal het scherm nog niet helemaal wit
      ' zijn. Laat daarom voor alle zekerheid even een wit vlak zien, door een groot
      ' wit Screen 3 te openen en even later weer te sluiten
      Screen Open 3,400,300,2,Lowres
      Palette $FFF,$FFF
      ' Laat het witte scherm even staan
      Wait 20
      ' Haal het witte scherm weer weg
      Screen Close 3
      Screen 1
      Screen Show 1
      ' Verander de kleuren van het palet snel weer naar de oorspronkelijke waarden
      Fade 3 To 2
      ' Wacht totdat de Fade opdracht klaar is
      Wait 150
End Proc
'
Procedure VERSCHIJN4
      ' Plaatje wordt van boven naar beneden zichtbaar
      ' Bedek het ingeladen IFF plaatje met een groot (overscan) zwart scherm
      Screen Open 2,400,300,2,Lowres
      Palette 0,0
      ' Maak IFF plaatje zichtbaar achter het zwarte Screen 2
      Screen Show 1
      ' Laat het zwarte scherm langzaam zakken
      For Y=0 To 300
            Screen Display 2,128,Y,400,300
            ' Zorg dat het scherm niet te snel zakt
            Wait 1
      Next Y
      ' Sluit het zwarte scherm weer
      Screen Close 2
End Proc
'
Procedure VERSCHIJN5
      ' Het ingeladen plaatje verschijnt van links naar rechts
      ' Maak een kopie van het IFF plaatje
      ' Open daartoe een maximale grote Screen 2 dat alle kleurregisters gebruikt
      Screen Open 2,742,568,4096,Lowres
      ' De kopie mag niet zichtbaar zijn
      Screen Hide 2
      Get Palette 1
      ' De eigenlijke kopieeropdracht
      Screen Copy 1 To 2
```

```
      ' Maak Screen 1 schoon
      Screen 1
      Cls 0
      ' Laat inmiddels lege Screen 1 zien
      Screen Show 1
      ' Hevel Screen 2 over naar Screen 1, blok voor blok
      ' Ga ervan uit dat het scherm maximaal groot is
      ' dus 742x568 (hires interlace overscan)
      For X=0 To 742
            Screen Copy 2,X,0,X+1,568 To 1,X,0
      Next X
End Proc
'
Procedure VERSCHIJN6
      ' Het ingeladen plaatje verschijnt bloksgewijs
      ' Maak een kopie van het IFF plaatje
      ' Open daartoe een maximale grote Screen 2 dat alle kleurregisters gebruikt
      Screen Open 2,742,568,4096,Lowres
      ' Laat de kopie niet zien
      Screen Hide 2
      ' Flash moet uit staan, anders knippert kleurregister 3
      Flash Off
      Get Palette 1
      ' De eigenlijke kopieeropdracht
      Screen Copy 1 To 2
      ' Maak Screen 1 schoon
      Screen 1
      Cls 0
      ' Laat het inmiddels lege Screen 1 zien
      Screen Show 1
      ' Hevel Screen 2 over naar Screen 1, blok voor blok
      ' Doe dat in 4 stappen:
      ' In iedere stap worden verticale lijnen over de gehele breedte zichtbaar gemaakt
      ' Zo ontstaat het effect van lamellen die opengaan
      For HERHALING=0 To 3
            For X=0 To 742 Step 4
                  Screen Copy 2,X+HERHALING,0,X+HERHALING+1,568 To 1,X+HERHALING,0
            Next X
      Next HERHALING
End Proc
'
Procedure VERDWIJN1
      ' Plaatje verdwijnt ineens
      Screen Close 1
End Proc
'
Procedure VERDWIJN2
      ' Plaatje verdwijnt langzaam (fade)
      Fade 7
      ' Wacht totdat de Fade opdracht klaar is
      Wait 105
End Proc
'
Procedure VERDWIJN3
      ' Het plaatje verdwijnt in een flits
      ' Verander alle kleuren in het palet in wit ($FFF)
      Fade 3,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF, →
      $FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF,$FFF, →
      $FFF,$FFF,$FFF
      ' Wacht totdat de Fade opdracht klaar is
      Wait 45
      ' Indien Screen 1 een HAM plaatje bevat is het scherm nog niet helemaal wit
      ' Laat daarom even een wit scherm zien door een grote Screen 2 te openen
      Screen Open 2,400,300,2,Lowres
      Palette $FFF,$FFF
      ' Laat het witte scherm even staan
```

```
      Wait 20
      ' Maak het hele scherm zwart door het palet van het witte vlak ineens
      ' in zwart te veranderen
      Palette 0,0
      Screen Close 1
      Screen Close 2
End Proc
'
Procedure VERDWIJN4
      ' Plaatje wordt van onder naar boven bedekt met een zwart vlak
      ' Open een groot (overscan) zwart scherm
      Screen Open 2,400,300,2,Lowres
      Palette 0,0
      ' Plaats het zwarte scherm onder het IFF plaatje
      Screen Display 2,128,300,400,300
      ' Laat het zwarte scherm langzaam omhoog komen
      For Y=300 To 0 Step -1
            Screen Display 2,128,Y,400,300
            ' Zorg ervoor dat het zwarte scherm niet te snel omhoog komt
            Wait 1
      Next Y
End Proc
'
Procedure VERDWIJN5
      ' Het ingeladen plaatje verdwijnt van rechts naar links
      ' Trek lijnen van boven naar beneden in kleur 0
      Ink 0
      For X=742 To 1 Step -1
            Draw X,0 To X,568
      Next X
End Proc
'
Procedure VERDWIJN6
      ' Het ingeladen plaatje verdwijnt bloksgewijs
      ' Trek verticale lijnen in kleur 0 die de tekening uitvegen
      For HERHALING=0 To 3
            For X=0 To 742 Step 4
                  Draw X+HERHALING,0 To X+HERHALING,568
            Next X
      Next HERHALING
End Proc
```

Liberty BASIC – De programmeertaal i.p.v. API.

Liberty BASIC is een goede BASIC versie om te leren. U kunt, wanneer u de programmeertaal kent, makkelijker verder gaan met andere programmeertalen als C of Pascal of een ander Basic dialect. Misschien wilt u wel met Liberty BASIC blijven programmeren. In ieder geval kunt u er heel veel mee.

Logische regel extensie
Liberty BASIC ondersteunt een logische regel extensie, waarmee u een lange coderegel meerdere keren kunt afbreken, bijvoorbeeld:

```
open "user32" for dll as #u
calldll #u, "GetWindowRect", hMain as long, Rect as struct, result as long
close #u
```

Een regel kan lang en moeilijk leesbaar zijn, zie eens de onderstaande vergelijking.

```
open "user32" for dll as #u
calldll #u, "GetWindowRect",_
     hMain as long,_
     Rect as struct,_
     result as long
close #u
```

Als de regel afgebroken is met het _ teken, is de code beter leesbaar.

Het NOMAINWIN statement
Wanneer een Liberty BASIC programma uitgevoerd wordt, is er een simpele tekstvenster aanwezig als hoofdvenster. Het kan worden gebruikt om tekst weer te geven en de gebruiker te vragen om invoer. Het hoofdscherm is niet speciaal nodig en als u het hoofdvenster ook niet wilt laten zien, gebruikt u het nomainwin statement:

```
nomainwin ' open geen hoofdvenster
menu #draw, "Teken", "Teken nu", [drawNow]
open "Niemandsland, eh... geen hoofdvenster" for graphics as #draw
print #draw, "trapclose [quit]"
wait

[drawNow]
     print #draw, "cls ; home; down ; north"
     for x = 1 to 100
          print #draw, "turn 122 ; go "; str$(x * 2)
     next x
     print #draw, "flush"
     wait

[quit]
     confirm "Wilt u afsluiten?"; quit$
     if quit$ = "Nee" or quit$ = "nee" then wait
     close #draw
     end
```

Het hoofdvenster kan worden gebruikt tijdens het ontwikkelen van een programma. Als een programma blokkeert of crasht, dan kan het worden afgesloten bij het sluiten van het hoofdvenster. Het is be-

langrijk dat uw programma trapclose handelingen heeft voor alle vensters wanneer een nomainwin statement wordt gebruikt, anders zal het programma doorgaan met uitvoeren zonder een mogelijkheid om af te sluiten. Alle programma's moeten beëindigd worden met een end statement (zoals het voorbeeld hierboven) om ervoor te zorgen dat programma's daadwerkelijk zichzelf schoonmaken.

Als een programma doorgaat met uitvoeren zonder een mogelijkheid om af te sluiten, dan kan het worden afgesloten door op het Run menu op de Liberty BASIC editor te klikken en Kill BASIC Programs te kiezen.

Als het nomainwin statement meerdere keren op elke plaats in de code aanwezig is, zal het hoofdvenster niet verschijnen. Het is eigenlijk geen statement, maar een compiler directive. Als de compiler dit statement op elke plaats in de code ziet, wordt het weergeven van het hoofdvenster onderdrukt.

Functies en subroutines
Liberty BASIC ondersteunt functies en subroutines. Ze lijken erg veel op QBASIC varianten:

```
'definieer een functie voor het retourneren van een vierkantswortel
function squareRoot(value)
     squareRoot = value ^ 0.5
end function
```

en...

```
'maak een subroutine voor het inloggen naar een event log
sub logToFile logString$
     open "c:\logdir\event.log" for append as #event
     print #event, time$()
     print #event, logString$
     close #event
end sub
```

Een zelf-gedefinieerde functie zoals hierboven, kan als inbouwfunctie worden gebruikt:

```
print "De vierkantswortel van 5 is "; squareRoot(5)
```

Subroutines in Liberty BASIC zijn toegankelijk bij gebruik van het call statement, bijvoorbeeld:

```
'Log nu wat info naar disk
call logToFile "De vierkantswortel van 5 is " + squareRoot(5)
```

De variabele scoop in subroutines en functies is lokaal. Dit betekent dat variabelennamen binnen de definitie van een subroutine of functie alleen daarbinnen zinvol zijn. Als voorbeeld, een variabele genaamd "counter" kan bestaan in de hoofdprogrammacode. Het programma kan een functie gebruiken die ook een variabelennaam "counter" bevat in die code. Wordt de functie gebruikt, dan zal de "counter" variabele in de aanroepende code niet zijn waarde verliezen als de functie de waarde van "counter" wijzigt. Het zijn eigenlijk verschillende variabelen, ook al delen ze dezelfde naam. Variabelen doorgegeven in subroutines mogen doorgegeven worden als verwijzing (passed by reference). Worden de waarden van die variabelen gewijzigd, dan zullen de wijzigingen ook worden weergegeven in het hoofdprogramma.

Hier is een voorbeeld dat een functie gebruikt:

```
'stel de variabele counter in
for counter = 1 to 10
     print loop(counter)
```

```
next counter
end

function loop(limit)
     for counter = 1 to limit
     next counter
     loop = counter
end function
```

Uitzonderingen met het variabele scoping mechanisme geven de volgende punten die overal globaal zichtbaar zijn in een Liberty BASIC programma:

- Arrays
- Handelingen (bestanden, vensters, DLL's, communicatiepoorten)
- Structs

GLOBAL
In het algemeen zijn variabelen die in het hoofdprogramma worden gebruikt niet zichtbaar binnen de functies en subroutines. Variabelen binnen de functies en subroutines zijn niet zichtbaar in het hoofdprogramma. Liberty BASIC introduceert het global statement om variabelen te creëren. Variabelen die gedeclareerd zijn met global kunnen overal in het programma gezien worden. De speciale systeemvariabelen WindowWidth, WindowHeight, UpperLeftX, UpperLeftY, ForegroundColor$, BackgroundColor$, ListboxColor$, TextboxColor$, ComboboxColor$, TexteditorColor$, DefaultDir$, Joy1x, Joy1y, Joy1z, Joy1button1, Joy1button2, Joy2x, Joy2y, Joy2z, Joy2button1, Joy2button2 en Com hebben een globale status. Globalen worden opgegeven en gebruikt zoals dit:

```
'definieer een globale stringvariabele
global title$
title$ = "Goed Programma!"
'Speciale systeemvariabelen hoeven niet met global gedeclareerd te worden,
'want hun status is automatisch
BackgroundColor$ = "darkgray"
ForegroundColor$ = "darkblue"
'roep de subroutine om een venster te openen
call openHet
wait

sub openHet
     statictext #het.stext, "Kijk mam!", 10, 10, 70, 24
     textbox #het.tbox, 90, 10, 200, 24
     open title$ for window as #het
     print #het.tbox, "Geen handen!"
end sub
```

Branch labels binnen functies en subroutines zijn niet zichtbaar voor codering buiten de functies en subroutines. Als code in het hoofdprogramma probeert toegang te krijgen naar een branch label binnen een functie of subroutine, dan veroorzaakt dit een fout. Net zo kunnen functies en subroutines niet de branch labels gebruiken die gedefinieerd zijn buiten de scope.

Argumenten doorgeven in subroutines en functies – bij waarde en bij verwijzing
Doorgaans in Liberty BASIC 3 kunnen variabelen in een gebruikers subroutine of functie alleen doorgegeven worden bij waarde. "Passing by value" is een algemene term waarmee bedoeld wordt dat er een kopie gemaakt wordt van de doorgegeven waarde en er geen relatie is met de originele waarde. Is de waarde gewijzigd in de aangeroepen subroutine of functie, dan verandert dat niet in het hoofdprogramma.

Nu in Liberty BASIC 4 kunnen variabelen doorgegeven worden bij verwijzing (by reference). Dit betekent dat de waarde van de variabele kan wijzigen en zodra de subroutine of functie eindigt en de uitvoer verder gaat na de aanroep, zal de wijziging in de variabele, die als parameter werd gebruikt, behouden blijven.

Bedenk goed dat parameters in gebruiker gedefinieerde functies en subroutines in QBASIC en Visual Basic altijd doorgegeven worden by reference. Dat is niet het geval in Liberty BASIC, waar de waarden standaard by value doorgegeven worden. Doorgeven by reference kan alleen worden gedaan als het byref sleutelwoord wordt gebruikt.

Voorbeeld bij doorgeven by value
Dit voorbeeld laat zien hoe het doorgeven by value werkt. De enige waarde, die terugkomt vanuit de functie, is degene die toegekend wordt aan result$ na de aanroep van de functie.

```
'dit is de manier dat altijd werkt
x = 5.3
y = 7.2
result$ = formatAndTruncateXandY$(x, y)
print "x = "; x
print "y = "; y
print result$
end

function formatAndTruncateXandY$(a, b)
     a = int(a)
     b = int(b)
     formatAndTruncateXandY$ = str$(a)+", "+str$(b)
end function
```

Voorbeeld bij doorgeven by reference
In tegenstelling tot het "doorgeven by value" voorbeeld, zal nu elke parameter in de functie in dit voorbeeld verwezen worden (doorgeven by reference). Dit betekent dat wanneer de waarde van a en b gewijzigd zijn in de functie weer terugkomen bij de aanroep van x en y. De parameters x en y zullen dus ook gewijzigd worden van de waarden a en b zodra de functie retourneert. Probeer het voorbeeld stap voor stap uit te voeren in de debugger.

```
'nu kunt u doorgeven by reference
x = 5.3
y = 7.2
result$ = formatAndTruncateXandY$(x, y)
print "x = "; x
print "y = "; y
print result$
'en het werkt ook met subroutines
wizard$ = "gandalf"
call capitalize wizard$
print wizard$
end

function formatAndTruncateXandY$(byref a, byref b)
     a = int(a)
     b = int(b)
     formatAndTruncateXandY$ = str$(a)+", "+str$(b)
end function
```

```
sub capitalize byref word$
     word$ = upper$(left$(word$, 1))+mid$(word$, 2)
end sub
```

Meer over doorgeven by reference

Doorgeven by reference is alleen toegestaan met gebruik van string en numerieke variabelen als parameters. U kunt een numerieke of letterlijke tekenreeks of een uitgerekend getal of string of zelfs een waarde vanuit een array doorgeven, maar de waarden zullen niet terugkomen vanuit de aanroep in elk van deze mogelijkheden. Voer het voorbeeld eens stap voor stap uit in de debugger om te zien hoe het werkt!

```
'u kunt ook aanroepen zonder variabelen, maar de wijzigingen
'komen niet terug
result$ = formatAndTruncateXandY$(7.2, 5.3)
print result$
'en het werkt ook met subroutines
call capitalize "gandalf"
a$(0) = "snoopy"
call capitalize a$(0)
end

function formatAndTruncateXandY$(byref a, byref b)
     a = int(a)
     b = int(b)
     formatAndTruncateXandY$ = str$(a)+", "+str$(b)
end function

sub capitalize byref word$
     word$ = upper$(left$(word$, 1))+mid$(word$, 2)
end sub
```

Structuren in Python.

Python is een unieke aparte programmeertaal. Althans, dat vind ik. Dat komt omdat de structuren heel anders liggen dan bijvoorbeeld in Pascal, in C en in Basic. Laten we eens gelijk met onderstaand voorbeeldje met de deur in huis vallen:

```
fruitmand = "Appels", "Bananen", "Peren", "Druiven"

for fruit in fruitmand:
     print(fruit)
```

We kunnen de variabele fruitmand gebruiken als een array. Willen we Bananen printen, dan moeten we typen:

```
print(fruitmand[1])
```

Maar we kunnen ook de hele lijst in een keer printen, zonder een lus te gebruiken. De lijst wordt dan wel tussen haakjes gezet.

```
print(fruitmand)
('Appels', 'Bananen', 'Peren', 'Druiven')
```

Voor het printen van tekst kunnen we de enkele of de dubbele aanhalingstekens gebruiken:

```
print("Hallo allemaal.")
print('Hallo allemaal.')
```

Maar deze print() code geeft een fout:

```
print('Dit is zo'n fout!')
```

De oplossing is om de backslash te gebruiken:

```
print('Dit is niet zo\'n fout!')
```

Maar deze werkt nog beter en kan alleen in Python werken.

```
print("Dit is niet zo'n fout en het werkt nog beter!")
```

Variabelennamen in Python kan van alles zijn tot grote ergernis, want we kunnen per ongeluk het volgende noemen:

```
print = 20
```

Python geeft geen foutmelding als u dit doet en het ergste is dat u de gewone print() functie kwijt bent. Wanneer u bovenstaande print() functies probeert uit te voeren, wil dat niet meer werken en krijgt u foutmeldingen. Sluit Python even af en start hem weer op. Alles zal dan weer normaal functioneren.

Elke programmeertaal heeft zijn eigen regels, zo ook Python. Dit kan dus niet:

```
tekst = Hee!
```

Symbolen zijn niet toegestaan. De tekst Hee! moet tussen aanhalingstekens staan:

```
tekst = "Hee!"
```

Wilt u meer tekst en/of getallen samen printen, gebruik dan het , teken. U kunt alleen tekst samenvoegen met het + teken:

```
print(tekst, " Ja, jij daar!")     # dit is goed
print(tekst + "Ja, jij daar!")     # dit is ook goed
print("Hij wordt" + 12 + "jaar oud.")   # dit is niet goed
print("Hij wordt", 12, "jaar oud.")     # dit is wel goed
```

Zoals het eerste voorbeeld liet zien, kent Python ook lussen: de for lus en de while lus. Elk heeft zijn eigen codeblok, maar niet gegeven door een begin...end of een {...} zoals we in andere programmeertalen gewend zijn.
In Python moeten we, na dat we de lus hebben opgegeven, de regel eindigen met een : (dubbelepunt). Na op Enter te hebben gedrukt, wordt de codeblok gestart. Dat gebeurt met een inspringfunctie. U hoeft niet op de tab toets te drukken om in te springen. Zolang de code wordt ingesprongen, wordt die code alleen in de lus uitgevoerd. Er is geen afsluitstatement, zoals we een End If of een Next statement kennen in Basic. Maar hoe sluiten we een codeblok dan wel af? Antwoord: door na de Enter toets nogmaals op Enter te drukken.

```
>>> i=0
>>> while i < 10:
     print(i * 2)
     print(i ** 2)
     i += 1


0
0
2
1
4
4
6
9
8
16
10
25
12
36
14
49
16
64
18
81
>>>
```

U ziet dat ** is de macht van de waarde van variabele i; i ** 2 was dan hetzelfde geweest als i * i, want 9 ** 2 = 81 en 9 * 9 = 81.

Een andere lus is de for lus. Bekijk eens onderstaande for lus:

```
>>> lijst = [1,5,6,6,2,1,5,2,1,4]
>>> for x in lijst:
     print(x)
```

Normaalgesproken ziet u de while-lus die wordt gebruikt voor eindige taken met vooraf bepaalde lengte, en de for-lus die wordt gebruikt voor taken met onzekere en variabele tijdskaders.

```
for x in range(1,11):
    print(x)
```

Deze code is eigenlijk wat bekend staat als een generator functie, en is zeer efficiënt. Het bovenstaande werkt als de "Counter"-functie die we ook met een while-lus kunnen maken. Het enige verschil is dat dit veel sneller en efficiënter werkt.

Een If statement is de meest gebruikte statement in de code. Met de opgegeven conditie wordt er bepaald of het resultaat waar is of niet.

```
if x < y:
     print("x is kleiner dan y.")
```

Is x niet kleiner dan y, dan wordt de print() functie niet uitgevoerd. Wat dan wel? Nou, eigenlijk dan niets. Maar Python kent ook een if ... else code, zodat we ook code kunnen uitvoeren als de conditie niet waar is.

```
x = 5
y = 8
if x > 55:
    print('x is groter dan 55')
else:
    print('x is niet groter dan 55')
```

Ook achter een else moet een dubbelepunt staan, want daarna begint er weer een codeblok. Onthoud dat een eenregelige code ook een codeblok is na een dubbelepunt.

In Python bestaat er een elif statement om meerkeuze code te kunnen maken. Alleen de else statement hoort bij de if statement. Die staat er altijd pas als laatste.

```
x = 5
y = 10
z = 22

if x > y:
    print('x is groter dan y')
elif x < z:
    print('x is kleiner dan z')
else:
    print('if en elif nooit uitgevoerd')
```

Eerst wordt gecontroleerd of x groter is dan y. Dat is niet het geval, dus wordt er bij elif gecontroleerd of x kleiner is dan z. Dat is wel het geval en de print() functie wordt dan ook uitgevoerd. Als geen een conditie waar is, wordt alleen de print() functie bij else uitgevoerd.

In Python kunnen we ook functies maken. De manier hoe ze geschreven moeten worden lijkt veel op BBC BASIC, behalve dat in Python alles een functie is, ook al is er geen return statement bij. Net als in BBC BASIC gebruiken we in Python het def statement, dan de functienaam en eventueel erachter de argumenten. De haakjes zijn altijd nodig.

Een voorbeeld is zoals deze:

```
def voorbeeld():
    print('deze code zal worden uitgevoerd')
    z = 3 + 9
    print(z)
```

Ook al is het een functie, we moeten deze aanroepen als een statement: `voorbeeld()`

Het voorbeeld hieronder is een functie met parameters:

```
def simple_addition(num1,num2):
    answer = num1 + num2
    print('num1 is', num1)
    print(answer)

simple_addition(5,3)
```

De parametervolgorde mag worden veranderd, maar dan moeten de argumentnamen ook meegegeven worden.

```
simple_addition(num2=3,num1=5)
```

Een argument kan ook een standaardwaarde hebben. Deze hoeft dan niet als parameter opgegeven te worden, zoals onderstaand voorbeeld:

```
def basic_window(width,height,font='TNR'):
    # let us just print out everything
    print(width,height,font)

basic_window(350,500)
```

De standaardwaarde mag worden gewijzigd door hem op te geven met `font='Courier New'`.

```
basic_window(350,500, font='Courier New')
```

Dus niet als argumentwaarde wijzigen, wel als parameter. Dat betekent dat onderstaande ook mag:

```
basic_window(350,500, '')
```

De volgende keer komt deel 2 van dit onderwerp. We gaan dan hier mee verder en maken programma's met goede Python structuren.