DE

**BASIX1**

Programmeertaal

Jarno Witjes | EHI2VSe

Saxion Compilers en Operating Systems

Jarno Witjes

449590

EHI2VSe

GitHub Repository

<https://github.com/JarnoWitjes/Basix>

Inhoud

[Inleiding 4](#_Toc10368295)

[Ontwerp van Basix1 5](#_Toc10368296)

[Features 5](#_Toc10368297)

[Beginnen met Basix1 6](#_Toc10368298)

[Voorbeeld 1: Hello World 6](#_Toc10368299)

[Voorbeeld 2: Variabele 6](#_Toc10368300)

[Voorbeeld 3: When-statement 7](#_Toc10368301)

[Voorbeeld 4: Complexe When-statement 8](#_Toc10368302)

[Voorbeeld 5: loop\_when statement 9](#_Toc10368303)

# Inleiding

De Basix1 programmeertaal is ontwikkeld als een opdracht voor Saxion’s module: Compilers en Operating Systems. Basix1 is een makkelijke taal waarmee je snel kleine basisprogramma’s kan schrijven en uit laten voeren. De eerste versie van de taal ondersteunt twee datatypes, conditional branching en loops. Deze eerste versie is niet object oriënterend.

Dit document bespreekt de taal aan de hand van voorbeelden.

# Ontwerp van Basix1

Basix1 is gebouwd voor programmeurs die snel een simpel programma willen bouwen voor verschillende doeleinden. Het programmeren met Basix1 gebeurt door ingebouwde functies aan te roepen en argumenten aan die functies te geven.

## Features

* Printen naar console
  + Tekst en integers
* Ondersteuning voor variabele
* Conditional branching
* Loop
* Basale checker

# Beginnen met Basix1

In de volgende pagina’s wordt uitgelegd hoe de taal werk. Er wordt uitgelegd aan de hand met voorbeelden.

## Voorbeeld 1: Hello World

We beginnen met de klassieke ‘Hello world!’ programma. Dit programma zal de tekst ‘Hello World’ printen naar de console en daarna afsluiten.

BASIX\_BEGIN  
  
print::text << “Hello World!” ;  
  
BASIX\_END

Regel voor regel:

BASIX\_BEGIN : Dit markeert het begin van een Basix1 programma.

print::text << “Hello World!” ; : Dit is de print functie. print geeft aan dat je iets wil printen naar de console. Na print moet je de aangeven wat je gaat printen, je kan tekst (text) of getallen (integer) printen. Nu dat je de print functie aan heb geroepen moet je aangeven wat je wilt printen je geeft dit aan met ‘<<’ met vervolgens je waarde.

Let op!

Je kan geen getal printen als je aan hebt gegeven dat je tekst gaat printen en andersom. Voorbeeld:

BASIX\_BEGIN  
  
print::integer << “Hello World!” ;  
print::text << 2020;  
  
BASIX\_END

BASIX\_END : Markeert het einde van een Basix1 programma.

## Voorbeeld 2: Variabele

Basix1 ondersteund variabele maar alleen voor twee datatypes: text en integer. Er zijn verschillende functies voor het werken met variabele:

make\_known::”type”::”name” << “value” ;

remake\_known::”name” << “value” ;

get\_known::”name” ;

De make\_known functie maakt een variable van het gegeven type, naam en waarde.

De remake\_known functie hangt een nieuwe waarde aan een al bestaande variabele.

De get\_known functie geeft de waarde terug van het gegeven al bestaande variabele. Voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::myint << 10 ;  
print::text << “Variable myint is now: ” ;  
print::integer << get\_known::myint; ;  
  
remake\_known::myint << 20 ;  
print::text << “Variable myint is now: ” ;  
print::integer << get\_known::myint; ;  
  
make\_known::integer::mathint << 20 + 12 ;  
print::text << “20 + 12 =” ;  
print::integer << get\_known::mathint ;  
  
make\_known::text::myname << “John Doe” ;  
print::text << “My name is: ” ;  
print::text << get\_known::myname; ;  
  
BASIX\_END

In het eerste blokje code wordt de variabele ‘myint’ aangemaakt. ‘myint’ Is van het type integer en heeft de waarde van 10. De waarde van ‘myint’ wordt uitgeprint naar de console.

In het tweede blokje wordt de variabele ‘myint’ aangepast, het heeft nu de waarde 20. Vervolgens wordt de nieuwe waarde geprint naar de console.

In het derde blokje code wordt een nieuwe variabele ‘mathint’ gemaakt. Deze variabele heeft de waarde van som 20 en 12.

In het vierde blokje wordt een text variabele ‘myname’ aangemaakt. Deze heeft de waarde ‘John Doe’. Vervolgens wordt deze geprint naar de console.

## Voorbeeld 3: When-statement

Met Basix1 kun je testen uitvoeren met bepaalde conditie en daarop bepaalde acties uitvoeren. Hier onder een simpel voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::myint << 20;  
  
when (get\_known::myint; eq 20) then (  
 print::text << "myint equals 20";  
); otherwise (  
 print::text << "myint doesn't equals 20";  
);  
  
BASIX\_END

In dit voorbeeld maken we eerst een integer variabele met de waarde 20. In onze if statement vergelijken wij de waarde van onze variabele met 20. Als de waarden overeenkomen wordt de tekst “myint equals 20” geprint naar de console. Als de waarden niet overeenkwamen zou de tekst “myint doesn’t equal 20” uitgeprint worden.

Voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::myint << 10;  
  
when (get\_known::myint; eq 20) then (  
 print::text << "myint equals 20";  
); otherwise (  
 print::text << "myint doesn't equals 20";  
);  
  
BASIX\_END

## Voorbeeld 4: Complexe When-statement

Met Basix1 kan je ook meerdere when-statements aan elkaar knopen, hierdoor kan je meerdere tests uit laten voeren.

Voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::a << 20;  
  
when (get\_known::a; eq 20) then (  
 print::text << "a equals 20";  
); otherwise test when (get\_known::a; ls 100) then (  
 print::text << "a is lesser then 100";  
); otherwise (  
 print::text << "a does not equal 20 and is greater than 100";  
);  
  
BASIX\_END

In dit voorbeeld wordt eerst getest of de waarde van a gelijk is aan 20. Als dit zo is wordt de tekst “a equals 20” geprint naar de console. Als dit niet zo is wordt de volgende test uitgevoerd. Deze test kijkt of de waarde van a kleiner is dan 100. Als dit zo is wordt de tekst “a is lesser then 100” geprint. Als alle tests niet gelukt zijn wordt de tekst “a does not equal 20 and is greater than 100” geprint naar de console.

Ook kan er getest worden of een van beide tests kloppen of beide test kloppen.

Voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::a << 20;  
make\_known::integer::b << 90;  
  
when (get\_known::a; ls get\_known::b; AND get\_known::a; gr 0) then (  
 print::text << "a is lesser than b and a is greater than 0";  
);  
  
when (get\_known::b; gr 0 OR get\_known::a; gr 0) then (  
 print::text << "Either b is greater than 0 or a is greater than 0";  
);  
  
BASIX\_END

## Voorbeeld 5: loop\_when statement

Met de loop\_when statement kunnen statements uitgevoerd worden meerdere keren. Deze loop blijft ‘loopen’ totdat een conditie false is.

Voorbeeld:

BASIX\_BEGIN  
  
make\_known::integer::a << 0;  
  
loop\_when (get\_known::a; ls 10) then (  
 print::integer << get\_known::a;;  
 remake\_known::a << get\_known::a; + 1;  
);  
  
BASIX\_END

In dit voorbeeld wordt de variabele a aangemaakt met de waarde 0. Vervolgens wordt de waarde van a geprint net zolang a kleiner is dan 10. De waarde van a wordt verhoogd met 1 na elke loop.

# Foutieve voorbeelden

In dit hoofdstuk komen vijf foutieve voorbeelden aan bod.

## Voorbeeld 1:

BASIX\_BEGIN  
  
print::integer << "420";  
  
BASIX\_END

Dit programma zal niet compileren omdat de print functie een integer waarde verwacht en geen tekst.

## Voorbeeld 2:

BASIX\_BEGIN  
  
make\_known::integer::a << "hello";  
  
BASIX\_END

Je kan geen text waarde geven aan een integer variabele.

## Voorbeeld 3:

BASIX\_BEGIN  
  
make\_known::integer::a << 4;  
  
when (get\_known::a; eq 10) then (  
 make\_known::text::name << "John";  
);  
  
print::text << get\_known::name;;  
  
BASIX\_END

De variabele ‘name’ bestaat niet in de globale scope maar alleen in de when scope.

## Voorbeeld 4:

BASIX\_BEGIN  
  
make\_known::integer::a << 10;  
  
remake\_known::a << "Some text";  
  
BASIX\_END

De variabele ‘a’ is van type integer. Als er een nieuwe waarde aan ‘a’ wordt gebonden moet deze ook van type integer zijn.

## Voorbeeld 5:

BASIX\_BEGIN  
  
print::text << get\_known::sometext;;  
  
BASIX\_END

Dit programma probeert de variable ‘sometext’ uit te printen, maar deze bestaat niet.

# Implementatie problemen

* Grammatica
  + When statement ‘bodies’ had ik als groep met statements staan; Hierdoor werd alleen het laatste statement toegewezen aan die groep; Heb toen een aparte token gemaakt.
* Code generatie
  + ‘and’ & ‘or’ clauses; Ik dacht dat er een speciale functie was hiervoor; Heb het toen uitgeschreven en kwam erachter dat je labels kon gebruiken om de flow te beïnvloeden.