Van variabelen naar Files

Tot nu toe heb je altijd gebruik gemaakt van variabelen in het interne (RAM) geheugen. Dit heeft een nadeel, zodra het programma wordt of gesloten of de computer wordt uitgedaan dan is alle opgeslagen informatie in het interne geheugen verdwenen.

Zolang de informatie niet meer nodig is, maak dit niet veel uit. Maar wanneer je iedere keer de informatie wilt raadplegen, zoals bijvoorbeeld een contactenlijst dan is dit niet handig. Om dit probleem op te lossen kunnen computers bestanden (files) opslaan op externe geheugens.

Data opgeslagen in bestanden wordt ook wel ***persistent data*** genoemd omdat het de informatie blijft behouden ook wanneer de computer is uitgemaakt,

Data opvragen van een bestand noemt men ***reading the file***. Deze files noemt men ***input files***. Net zo, wordt het sturen van data naar een bestand ***writing to the file*** genoemd. Het bestand heet dan een ***output file***.

Elk bestand heeft een naam en een extensie. Vroeger mocht de naam uit maximaal 8 letters bestaan met een extensie van drie letter. Deze extensies zijn er nog steeds en geven aan om wat voor soort bestand het gaat. De tabel hieronder geeft een aantal veel voorkomende extensie weer.

Extension	File type	Gebruikt voor
.dat	General data file	Informatie over een specifieke toepassing opslaan
.exe	Executable file	Applicatie
.gif	Graphic Interchange Format	Website images
.html	Hypertext Markup Language – website file	Web pagina’s
.jpg	An image encoded with the JPEG standard	Photo van digitale camera
.mp3	Music encoded in MPEG layer 3 audio format	Audio bestanden
.pdf	Portable Document Format	Ebooks
.txt	General text file	Pure tekst bestanden

Het Small Basic bestand object

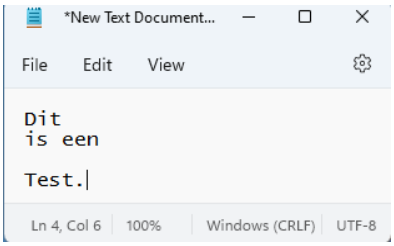
Het Small Basic **File**-object bevat alle methoden voor het lezen en schrijven van bestandsgegevens, het verwijderen en kopiëren van bestanden en het weergeven van directory-inhoud.

FIle I/O methods

De meest gebruikte methoden van het File-object zijn die welke worden gebruikt om gegevens naar een bestand te schrijven en gegevens uit bestanden te lezen.

Opdracht 1:

Start Notepad en type wat woorden in, zoals in de afbeelding. Zorg ervoor dat er **NIET** op de <ENTER>-toets wordt gedrukt na de laatste regel.



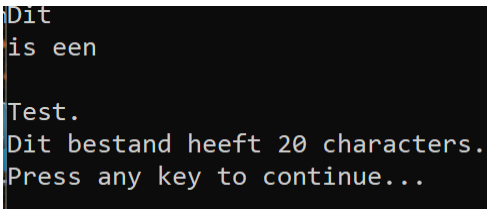
Waarom moet er niet op <ENTER> gedrukt worden na het laatste woord?

Sla het document op als Test1.txt in C:\Temp\Test1.txt zo dat het een absolute pad naam is.

Start Small Basic en type de onderstaande code in.

```
1 'ReadContentsDemo.sb
2 path = "C:\Temp\Test1.txt"
3 str = File.ReadContents(path)
4 len = Text.GetLength(str)
5 TextWindow.WriteLine(str)
6 TextWindow.WriteLine("Dit bestand heeft " + len + " characters.")
7
```

Voer het programma uit. De uitvoer is zoals in de afbeelding.



Uitleg broncode:

Regel 2 stelt het absolute pad van het bestand in.

Regel 3 leest de volledige inhoud van het bestand en slaat de geretourneerde tekenreeks op in een variabele met de naam `str` met behulp

van de `ReadContents()`-methode. `ReadContents()` heeft één argument: de padnaam van het bestand dat u wilt lezen.

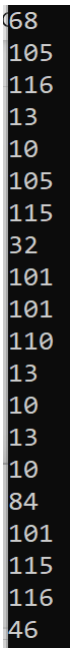
Regel 4 haalt de lengte van de string op en slaat deze op in de variabele met de naam `len`.

Regels 5 - 6 geven de variabelen `str` en `len` weer.

Ga na waarom `GetLength()` het totaal van 20 karakters laat zien.

Voeg de code van de onderstaande afbeelding toe aan de al geschreven code.

```
8  For N = 1 To len
9    ch = Text.GetSubText(str,N,1)       'Selecteerd 1 karater
10   code = Text.GetCharacterCode(ch)    ' Selecteer de code voor deze karakter
11   TextWindow.WriteLine(code)          ' Toon de code
12 EndFor
13
```



In de afbeelding hiernaast zie je de uitvoer die deze code produceert. Kun je het nu verklaren?

Bekijk de onderstaande tabel voor de uitleg.

code	68	105	116	13	10	105	115	32	101	101	110	13	10	13	10	84	101	115	116	46
Karakter	D	i	t	Carriage return	Line feed	i	s	Space	e	e	n	Carriage return	Line feed	Carriage return	Line feed	T	e	s	t	.

Notepad voegt twee speciale tekens in, de zogenaamde carriage return en *line feed*, waarvan de ASCII-codes 13 en 10 zijn, om het einde van elke regel te markeren. Beschouw de markering *newline* (of eind-of-line) als een paar tekens die worden geproduceerd wanneer u op ENTER op het toetsenbord drukt. Zonder deze tekens zouden de regels in het bestand samen in één lange regel lopen. De *newline* karakters zijn controle

karakters; ze controleren alleen de positie van de cursor op het scherm of de printer. De methode `ReadContents()` retourneert de volledige inhoud van het bestand als een enkele tekenreeks, inclusief nieuwe regeltekens tussen regels in het bestand.

Schrijven naar een bestand

Met de methode `WriteContents()` kun je de inhoud van een tekenreeks in een programma opslaan in een bestand naar keuze. Als je meerdere regels tekst wilt maken, moet je de *newline* tekens handmatig invoegen.

Opdracht 2:

Voer de onderstaande programmacode in Small Basic in.

```
1  'WriteContentsDemo.sb
2  CR = Text.GetCharacter(13)                    'Code voor cariage return
3  LF = Text.GetCharacter(10)                    'Code for line feed
4  outFile = "C:\Temp\Out.txt"                   'Absolute pad voor output file
5
6  strOut = ""                                   'Tekst die naar bestand wordt geschreven
7  strIn = ""                                    'Een regel (invoer gebruiker)
8  While(strIn <> "exit")                        'tot dat gebruiker "exit" invoert
9    TextWindow.Write("Data (exit to end): ")    'Vraagt om tekst in te voeren
10   strIn = TextWindow.Read()                   'Leest regel
11   If (strIn <> "exit") then                   'Als gebruiker GEEN "exit" heeft ingevoerd
12     strOut = strOut + strIn + CR + LF         'plakt tekst aan strOut
13   EndIf
14 EndWhile
15
16 File.WriteContents(outFile, strOut)          'schrijft strOut naar bestand
```

Na de code in Small Basic te hebben ingevoerd sla het op als **WriteContensDemo.sb**.

Voer het programma uit en voer de tekst van de onderstaande afbeelding in het programma in. Kijk daarna in C:\Temp\ naar het bestand out.txt in NotePad.

```
Data (exit to end): If Peter Piper picked a peck of pickled peppers,
Data (exit to end): Where's the peck of pickled peppers? I'm hungry.
Data (exit to end): exit
```

Bekijk de code en bepaal hoe het werkt.

Pad bestaat niet

Type het onderstaande programma in Small Basic in. Sla het op als "BadPath.sb" en voer het programma daarna uit.

```
1 'BadPath.sb
2 path = "C:\Temp\Folder1\Out.txt"
3 res = File.WriteContents(path, "Hello")
4 TextWindow.WriteLine(res + ": " + File.LastError)
```

Bij het uitvoeren toont het programma het volgende:

```
FAILED: Could not find a part of the path 'C:\Temp\Folder1\Out.txt'.
Press any key to continue...
```

Het programma probeert de tekenreeks "Hello" naar een output bestand te schrijven (regels 2-3). De directory *Temp* bestaat, maar de subdirectory *Folder1* bestaat niet, dus `WriteContents()` mislukt (*Fails*).

Appending to A File

De methode `AppendContents()` opent het opgegeven bestand en voegt gegevens toe aan het einde van het bestand zonder de oorspronkelijke inhoud te wissen. `AppendContents()` heeft twee argumenten: de pad naam van het uitvoerbestand en de string die aan het einde van het bestand moet worden toevoegt.

Als de bewerking succesvol is, retourneert de methode "`SUCCESS`"; anders wordt "`FAILED`" geretourneerd. Als het bestand dat je doorgeeft aan `AppendContents()` niet bestaat, wordt het voor je gemaakt en wordt de tekenreeks ernaar geschreven. Als het bestand al bestaat, wordt de tekenreeks aan het einde toegevoegd.

Laten we zeggen dat je een logbestand moet bijhouden waarin acties, fouten en andere gebeurtenissen in uw programma worden vastgelegd om de `AppendContents()`-methode in gebruik te zien. Laten we, om het programma eenvoudig te houden, de tijden noteren waarop uw programma wordt uitgevoerd. Elke keer dat jouw programma wordt uitgevoerd, voeg je een record toe aan een logbestand met de datum en tijd.

Type het onderstaande programma in Small Basic in. Sla het op als "AppendContentsDemo.sb" en voer het programma daarna uit.

```
1  'AppendContentsDemo.sb
2  outFile = Program.Directory + "\Log.txt"
3
4  strLog = Clock.WeekDay + ", " + Clock.Date + ", " + Clock.Time
5  result = File.AppendContents(outFile, strLog)
6  If (result = "FAILED") Then
7    TextWindow.WriteLine("Failed to write to: " + outFile)
8    TextWindow.WriteLine(File.LastError)
9  EndIf
10
11 TextWindow.WriteLine("Thank you for using this program. And for using deodorant.")
```

Na uitvoer zoek het log.txt bestand. Deze bevindt zich op dezelfde plaats als het .exe bestand van het gemaakte programma. Dubbelklik op het bestand om het te openen in notepad. De inhoud zou er zoals in de onderstaande afbeelding uit kunnen zien.

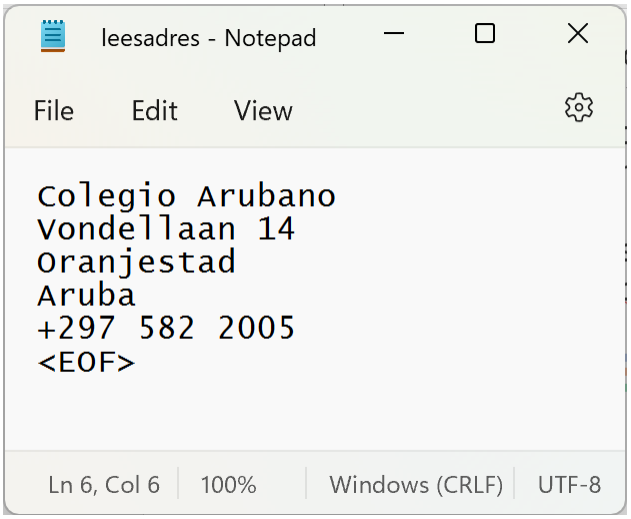
```
Monday, 9/19/2022, 6:41:22 PM
Monday, 9/19/2022, 6:41:28 PM
```

ReadLine(), WriteLine, and InsertLine()

Met de methoden `ReadContents()` en `WriteContents()` kun je de volledige inhoud van een bestand in één keer lezen en schrijven. Soms is dit net wat je nodig hebt. Maar in andere situaties is het misschien beter om regel voor regel te lezen of te schrijven.

Het File-object biedt de `ReadLine()`-methode voor het lezen van een enkele regel tekst uit een bestand. Een regel tekst bestaat uit een *string* (reeks tekens) die eindigt met een *carriage* return en een *line feed*. `ReadLine()` leest alle tekst op die regel tot (*maar **niet** inclusief*) het *carriage return* teken. Deze methode heeft twee argumenten: het pad van het bestand en het regelnummer van de te lezen tekst. De eerste regel van een bestand is regel nummer 1, de methode retourneert de tekst op die regel. Anders wordt een lege *string* geretourneerd.

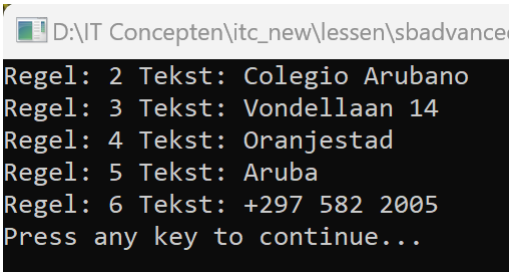
Start notepad en schrijf de volgende tekst percies als in het onderstaande voorbeeld in notepad.



Start Small Basic en voer het onderstaande programma code in. Sla het programma op als adressenlezen.sb

```
1  'adressenlezen.sb
2  inFile = Program.Directory + "\leesadres.txt"     'Lokatie van tekst bestand
3  strIn = ""
4  lineCount = 1
5  strIn = File.ReadLine(inFile, lineCount)      'Eerste regel vooruitlezen
6
7  While (strIn <> "<EOF>")
8    If (strIn <> "<EOF>") Then
9      lineCount = lineCount + 1
10     Textwindow.WriteLine("Regel: " + lineCount + " Tekst: " + strIn)
11     strIn = File.ReadLine(inFile, lineCount)    'Lees volgende regel
12   EndIf
13 EndWhile
14 lineCount = lineCount - 1            'EOF regel weer aftrekken
```

Start het programma bekijk de uitvoer en bestudeer hier na de code goed.
De uitvoer van het programma is als volgt:



Het `File`-object biedt ook de methode `WriteLine()` voor het uitvoeren van een regel tekst naar een bestand. Deze methode heeft drie argumenten nodig: het pad van het bestand, het regelnummer waarnaar tekst moet worden geschreven en de tekst die moet worden geschreven.

Houd rekening met de volgende informatie wanneer deze methode gebruikt wordt:

1. Als het bestand niet bestaat, maakt `WriteLine()` het aan.
2. Als het bestand het opgegeven regelnummer bevat, overschrijft `Writeline()` die regel.
3. Als het opgegeven regelnummer groter is dan het aantal regels in het bestand, wordt de opgegeven tekst aan het einde van het bestand toegevoegd.
4. `Writeline()` schrijft automatisch een regelterugloop en regelinvoer aan het einde van de doorgegeven tekst. Dit betekent dat u deze tekens niet handmatig aan uw string hoeft toe te voegen.
5. Als de bewerking is geslaagd, retourneert `WriteLine() "SUCCESS";` anders wordt `"FAILED"` geretourneerd.

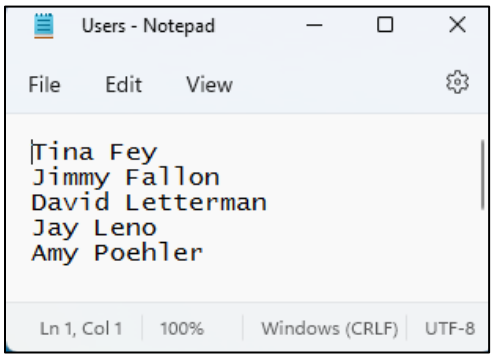
Naast `ReadLine()` en `WriteLine()`, biedt het File-object de **`InsertLine()`**-methode waarmee u een regel tekst in een bestand kunt invoegen met een opgegeven regelnummer. Net als bij de methode `WriteLine()` heeft deze methode drie argumenten nodig: het pad van het bestand, het regelnummer waar u de nieuwe tekst wilt invoegen en de tekst die u wilt invoegen. `InsertLine()` overschrijft geen enkele bestaande inhoud op de opgegeven regel. Als de bewerking succesvol is, retourneert `InsertLine() "SUCCESS";` anders wordt `"FAILED"` geretourneerd.

Opdracht 3: Login namen
Laten we als voorbeeld een eenvoudig programma schrijven dat inlognamen maakt van de voor- en achternaam van gebruikers. Het programma zal een invoerbestand ***users.txt*** (*input file*) lezen dat de voor- en achternaam van gebruikers bevat, en het zal een uitvoerbestand ***loginnames.txt*** (*output file*) maken dat de inlognamen voor deze gebruikers bevat.

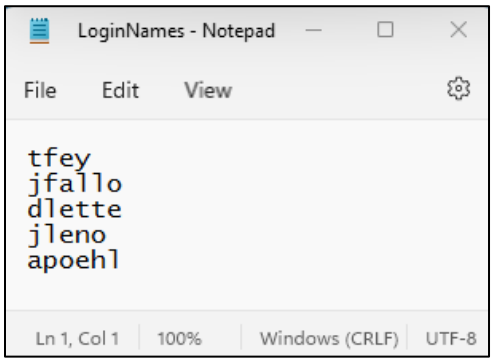
De inlognaam voor een gebruiker bestaat uit de eerste letter van de voornaam van de gebruiker en **maximaal** vijf tekens van de achternaam. Als de gebruikersnaam bijvoorbeeld Jack Skellington is, is zijn loginnaam *jskell*. Als de gebruikersnaam Stan Lee is (achternaam van drie letters), is zijn loginnaam *slee*.

Sla het programma op als **LoginName.sb** en maak een *NotePad* bestand aan op dezelfde plaats als je het **LoginName.sb** hebt opgeslagen.

Vul het invoerbestand met gegevens. Zorg ervoor dat de gegevens zo zijn opgemaakt dat elke regel de voor- en achternaam van een gebruiker bevat, gescheiden door een enkele spatie.



Het uitvoerbestand ziet er als volgt uit:



File Management

Naast de methoden waarmee bestands-I/O kan worden uitgevoerd, biedt het `File`-object ook een aantal methoden met betrekking tot bestands- en directorybeheer. Met deze methoden kunnen bestanden gekopieerd en verwijderd worden, mappen aangemaakt en verwijderd, en bestanden en mappen vanuit een programma worden weergeven.

Kopieren en verwijderen van bestanden

U kunt de methode `CopyFile()` gebruiken om een kopie van een bestaand bestand te maken. Deze methode neemt de pad namen van het bronbestand en het doelbestand als argumenten. Het bronbestand wordt niet beïnvloed door deze bewerking. Als de bewerking succesvol is, retourneert de methode "SUCCESS". Anders wordt "FAILED" geretourneerd. Als het doel pad verwijst

naar een locatie die niet bestaat, probeert de methode deze automatisch te maken. Kijk bijvoorbeeld naar de volgende code:

```
1 srcPath = "C:\Temp\Test1.txt"                    'Pad van het bron bestand
2 dstPath = "C:\Temp\Temp1\Temp2\Test1.txt"        'Pad van het doel bestand
3 File.CopyFile(srcPath, dstPath(
```

Als de submappen *Temp*, *Temp1* en *Temp2* niet bestaan, probeert `CopyFile()` alle mappen in het doel pad te maken. Beginnend met de wortel. Als je deze code uitvoert, heb je twee kopieën van het *Test1.txt*-bestand; het originele bronbestand en het dubbele bestand onder *C:\Temp\Temp1\Temp2*.

LET OP:

- Als het doel pad naar een bestaand bestand verwijst, wordt dat bestand overschreven.
- Het verwijderde bestand gaat niet naar de prullenbak; in plaats daarvan is het volledig uit het systeem verwijderd.

Aanmaken en verwijderen van Mappen *(Directories)*

U kunt eenvoudig een map maken of verwijderen. De methode `CreateDirectory()` gebruikt een enkel argument de pad naam van de map die u wilt maken. Als de mappen niet bestaan, probeert de methode alle mappen in het pad aan te maken, te beginnen met de root. Als de bewerking succesvol is, retourneert de methode "`SUCCESS`". Anders wordt "`FAILED`" geretourneerd.

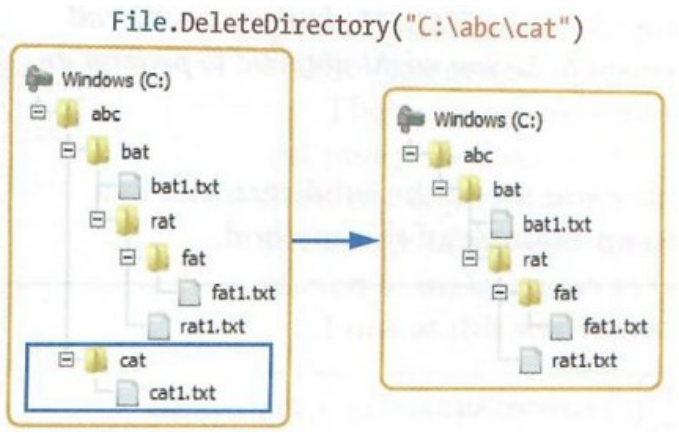
Bijvoorbeeld:

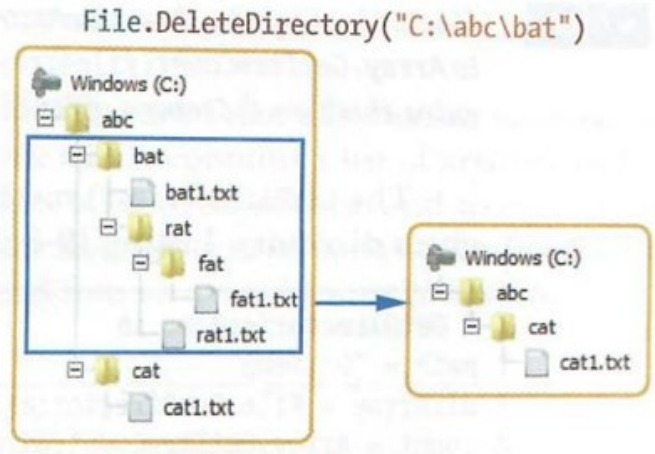
```
File.CreateDirectory("C:\Temp\Temp1\Temp2")
```

Als de mappen *C:\Temp\Temp1* en *C:\Temp\Temp1\Temp2* niet bestaan, `maakt CreateDirectory()` ze aan. Als het directory pad al bestaat, doet de functie niets en retourneert "`SUCCESS`".

De methode `DeleteDirectory()` heeft ook een enkel argument: de padnaam van de map die u wilt verwijderen. Alle bestanden en mappen onder het pad worden verwijderd. Als de bewerking succesvol is, retourneert de methode

"SUCCESS". Anders wordt "FAILED" geretourneerd. De onderstaande afbeeldingen toont een voorbeeld van `DeleteDirectory()`.





LET OP:

Wanneer men `DeleteDirectory()` aanroept, worden alle bestanden en mappen onder de pad naam verwijderd.

List Files and Directories

Het File-object bevat de `GetFiles()`-methode, waarmee u alle bestanden in een map kunt weergeven. Deze methode neemt het pad van de doel map als argument. Schrijf het onderstaande programma in Small Basic en sla het op als **GetFilesDemo.sb**. Het programma laat zien hoe je deze methode kunt gebruiken.

```
'GetFilesDemo.sb
path = "C:\Temp"
fileArray = File.GetFiles(path)
count = Array.GetItemCount(fileArray)
TextWindow.WriteLine(path + " contains " + count + " files:")
For N = 1 To count
  TextWindow.WriteLine(" " + fileArray[N])
EndFor
```

De uitvoer van het programma zou er als volgt uit kunnen zien:

```
C:\Temp contains 2 files:
 C:\Temp\Out.txt
 C:\Temp\Test1.txt
Press any key to continue...
```

LET OP:

Als `GetFiles()` mislukt, slaat `fileArray` de tekenreeks "`FAILED`" op. In dit geval retourneert de aanroep van `Array.GetItemCount(fileArray)` 0. Het is dus mogelijk dat u geen extra controle hoeft uit te voeren op de terugkeer van `GetFiles()`.

Met de methode GetDirectories() kunt u alle submappen in een bepaalde map weergeven. Schrijf het onderstaande programma in Small Basic en sla het op als **GetFirectoriesDemo.sb**. Het programma laat zien hoe je deze methode kunt gebruiken.

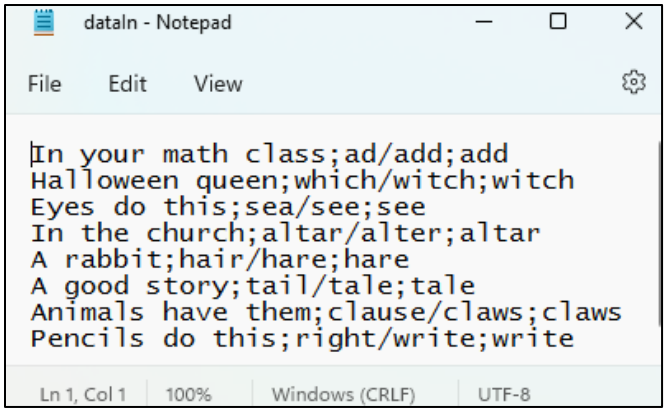
```
'GetDirectoriesDemo.sb
path = "C:\Temp"
dirArray = File.GetDirectories(path)
count = Array.GetItemCount(dirArray)
TextWindow.WriteLine(path + " contains " + count + " directories:")
For N = 1 To count                                    'Toont de array's elementen
  TextWindow.WriteLine(" " + dirArray[N])
EndFor
```

De uitvoer van het programma zou er als volgt uit kunnen zien:

```
C:\Temp contains 1 directories:
 C:\Temp\Temp1
Press any key to continue...
```

Uitdaging

Laten we een spellingquiz schrijven met homoniemen. Homoniemen zijn woorden die hetzelfde klinken maar een verschillende betekenis hebben. Gebruik Kladblok om het volgende tekstbestand te maken:



Elke regel in dit bestand bevat drie velden gescheiden door puntkomma's.

1. Het eerste veld is de hint die je de speler laat zien, zoals "`In your math class`".
2. Het tweede veld zijn de twee mogelijke antwoorden waaruit de speler zal kiezen, zoals "`ad/add`".
3. Het derde veld is het juiste antwoord, zoals "`add`".

Laat het programma in elke ronde de hint en de twee mogelijke antwoorden aan de speler tonen en wacht tot deze een antwoord invoeren.
Laat het programma het antwoord van de gebruiker vergelijken met het juiste antwoord en laat hem vervolgens weten of zijn antwoord juist is.

De uitvoer moet er dan zo uit zien:

```
In your math class (ad, add): add
Correct. You're smarter than the average Yogi the Bear.

Halloween queen (which, witch): _
```

Maak een analyse hoe het programma moet gaan werken. Dit wil zeggen bepaal welke functies van Small Basic nodig zullen zijn, bepaal welke variabelen nodig zullen zijn.

Het programma bestaat uit een hoofd lus die twee functies aanroept. Een functie die de vraag stelt een de speler/gebruiker, bijvoorbeeld `VraagSpeler()` en een functie die regel voor regel het gemaakte tekst bestand doorloopt en verwerkt, bijvoorbeeld `VerwerkRegel()`

Gebruik het programma Flowgorithm of Structorizer om het in schema te brengen.

Succes.