Der Stammbaum

Ein Projekt zur Erstellung eines Binärbaumes in Basic von

Andreas Gronemeyer

1. Vorwort

Als ich mich mit dem Thema Stammbaum/ Binärbaum beschäftigte und ich einfach mal zur Definition des Wortes Stammbaum in die Wikipedia schaute, stellte ich fest, dass ein Stammbaum nicht gleich ein Stammbaum ist. Aus Sicht eines Individuums welches die Wurzel darstellt und wir einen Baum seiner Abstammungslinie erstellen wollen, handelt es sich keines Falls um einen Stammbaum. Genealogen sprechen bei dieser Art von Bäumen von einer Ahnentafel.

In dieser Dokumentation geht es aber um die Erstellung eines Binärbaumes und seine Anwendung, und nicht um Genealogie. Es ist es mir aber wichtig diese Begrifflichkeit vorab einmal klarzustellen. Dennoch werde ich in dieser Dokumentation von dem Wort Stammbaum Gebrauch machen, da es letztendlich für die Programmierung eines Binärbaumes irrelevant ist. Und alle anderen tun es ja schließlich auch so.

Literatur zu den Bäumen oder Graphen gibt es reichlich im Internet. Auf Internetseiten von Universitäten, Softwarefirmen oder auch Privatpersonen, wird man schnell fündig. Hier möchte ich auch noch mal auf einschlägige Fachliteratur hinweisen wie Robert Sedgewick's Algorithmen. Das Buch ist mittlerweile seit den ersten Ausgaben, Ende der 80er glaube ich war es, wesentlich dicker und ausführlicher geworden. Damals hieß der Titel noch „Algorithmen in C".

Der Stammbaum ist ein beliebtes Beispiel, um die Funktionsweise der Algorithmen zum Erstellen und Untersuchen von Binärbäumen zu veranschaulichen. Auch werden diese Datenstrukturen gerne als Arbeitsproben von Bewerbern abverlangt.

Ein Binärbaum kann aber weitaus mehr als nur Stammbäume erzeugen.

Diese Dokumentation ist auf meiner Internetseite zu finden. Sie ist dort als PDF Datei hinterlegt: https://getobject.de/Objekte/dokumentationen. Eine .ODT Version für LibreOffice Writer mit einem lauffähigen Makro ist auf den Seiten der Dokumentfoundation als Extension abrufbar.

Ennepetal, den 02.11.2021

1.1. Begriffserklärungen

Traversierung

Unter Traversierung versteht man die Reihenfolge wie Knoten in einem Binärbaum untersucht oder bearbeitet werden. Man unterscheidet zischen Pre-Order, In-Order oder Post-Order Traversierung für die Tiefensuche, und die Level-Order Traversierung für die Breitensuche.

Binärbaum

Man spricht von einem Binärbaum, wenn ein Knoten genau zwei ausgehende Knoten aufnehmen kann. Ein Binärbaum hat aber die Wurzel oben und wird nach unten immer breiter.

Bezugsknoten

Ist der aktive zu betrachtende oder traversierte Knoten.

Ebene

Die Zählung der Ebenen beginnt bei 1 mit der Wurzel. Jeder rekursive Aufruf bewirkt eine Inkrementierung der Ebene. Ein Abstieg dekrementiert die Ebene. Man spricht aber von der Tiefe eines Baumes.

Suchbaum

Der Suchbaum ist der fertig erzeugte Binärbaum.

Parent

Der Parent, übersetzt Elter oder Elternteil, darf nicht mit den Vater- oder Mutterknoten verwechselt werden. Der Knoten ist aus der Sicht des Binärbaumes zu sehen. Er ist der Vorgänger eines Bezugsknotens und liegt ein Level tiefer. Aus der Sicht einer Person ist er aber ein Kindsknoten.

Elterknoten

Die Vater- und Mutterknoten sind aus der Sicht des Suchbaumes Folgeknoten, also Child‘s.

Kindsknoten

Das Kind in der Ahnentafel bildet im Binärbaum den Parentknoten.

Child

Der Child ist ein Folgeknoten im Binärbaum. Er liegt eine Ebene tiefer als der Bezugsknoten.

Verbinder

Der Verbinder ist eine Zeichenkette der die Knoten untereinander visuell zusammenfügt.

Feld

1. Ein Feld bildet die maximale Länge einer Zeichenkette ab. Ist die Zeichenkette länger als das Feld, wird diese in der Regel abgeschnitten. Ist die Zeichenkette kürzer als die Länge des Feldes, wird die Zeichenkette mit Leerzeichen aufgefüllt.
2. Mit Feld wird auch ein Array bezeichnet.

Pre Order

Die Pre Order Traversierung betrachtet zunächst einen Knoten (Wurzel) durchläuft den linken Knoten und anschließend den rechten Knoten.

Post Order

Zuerst wird der linke Knoten durchlaufen, dann der rechte Knoten. Anschließend wird die Wurzel betrachtet.

In Order

Die In Order Traversierung wird auch als symmetrische Reihenfolge genannt. Als erstes wird der linke Knoten durchlaufen, dann die Wurzel betrachtet. Anschließend wird der rechten Knoten durchlaufen.

2. Der Stammbaum

Einen Binärbaum zu erstellen ist eigentlich keine große Sache. Mit ein paar Zeilen Code ist dieser recht schnell implementiert. Um allerdings mit dieser Struktur ordentlich arbeiten zu können ist es wichtig sich zu überlegen, mit welchen Daten er gefüttert wird. Dazu eignet sich der Stammbaum bzw. unsere Ahnentafel hervorragend.

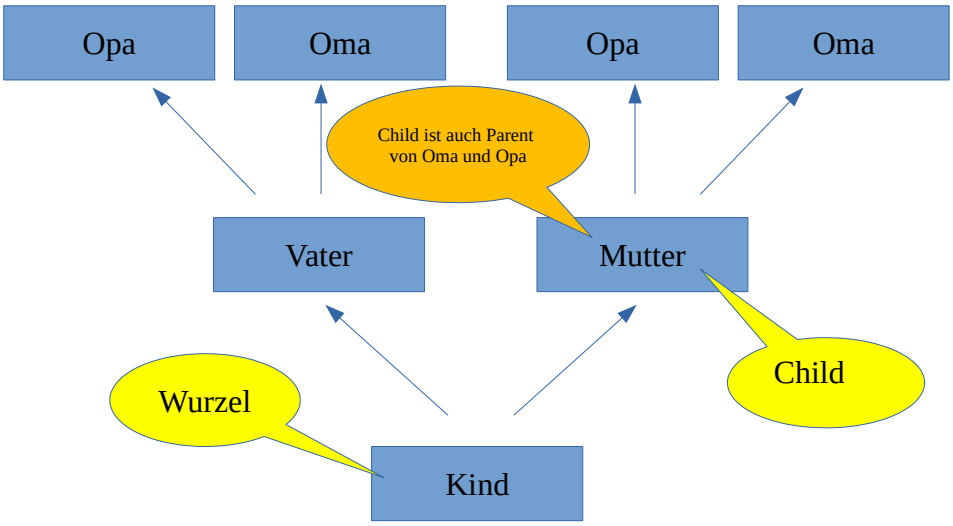

Abbildung 1: Aufbau der Ahnentafel

Ein Binärbaum hat ebenfalls die Baumstruktur wie unsere Ahnentafel. Allerdings steht der Binärbaum auf dem Kopf. Der Aufbau beginnt mit der Wurzel oben an der Spitze und die Ebenen verbreitern sich nach unten.

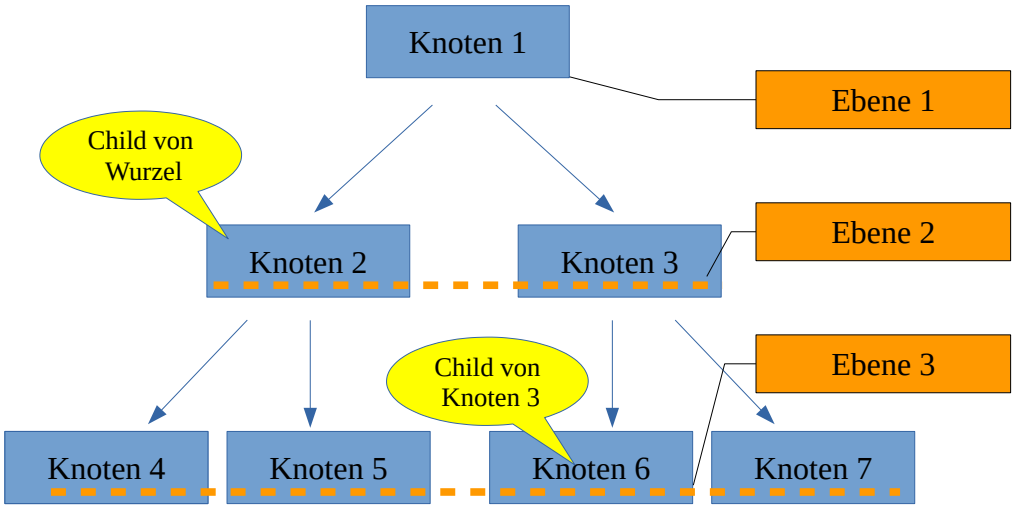

Abbildung 2: Aufbau eines Binärbaumes

2.1. Implementierung

Es gibt zwei Möglichkeiten unseren Stammbaum/Binärbaum in einem Programm zu implementieren. Zum einen besteht die Möglichkeit den gesamten Binärbaum in einem Array bzw. Feld zu sammeln. Der Vorteil dieser Methode ist das einfache berechnen der Knotenindizes. In der Array Variante ist es wichtig das fehlende Blätter durch leere Pseudo-Blätter besetzt werden müssen.

Die zweite Möglichkeit ist die rekursive Implementierung. Bei der rekursiven Variante wird eine Prozedur aus sich selbst heraus mehrfach aufgerufen. Man kann diese Art der Abarbeitung mit einer Treppe vergleichen, wobei die Prozedur die Stufen bildet. Diese Implementierung ist aus meiner Sicht die kompliziertere, aber auch die effektivere Möglichkeit einen Binärbaum zu implementieren. Insbesondere wenn große Datenmengen verarbeitet werden müssen führt eigentlich kein Weg an dieser Variante vorbei.

In der ersten Version dieser Dokumentation benutzen ich die rekursive Implementation. Aber auch die Array Lösung werde ich in der nachfolgenden Version als Beispiel mit aufnehmen.

Als erstes betrachten wir uns ein Beispiel wie unser Ergebnis letztendlich aussehen soll. Die Ahnentafel wird den Aufbau wie unten in Abbildung 3 dargestellt haben. Ein erfahrener Programmierer erkennt schon auf dem ersten Blick das hier einige kleine Nickeligkeiten eingebaut sind.

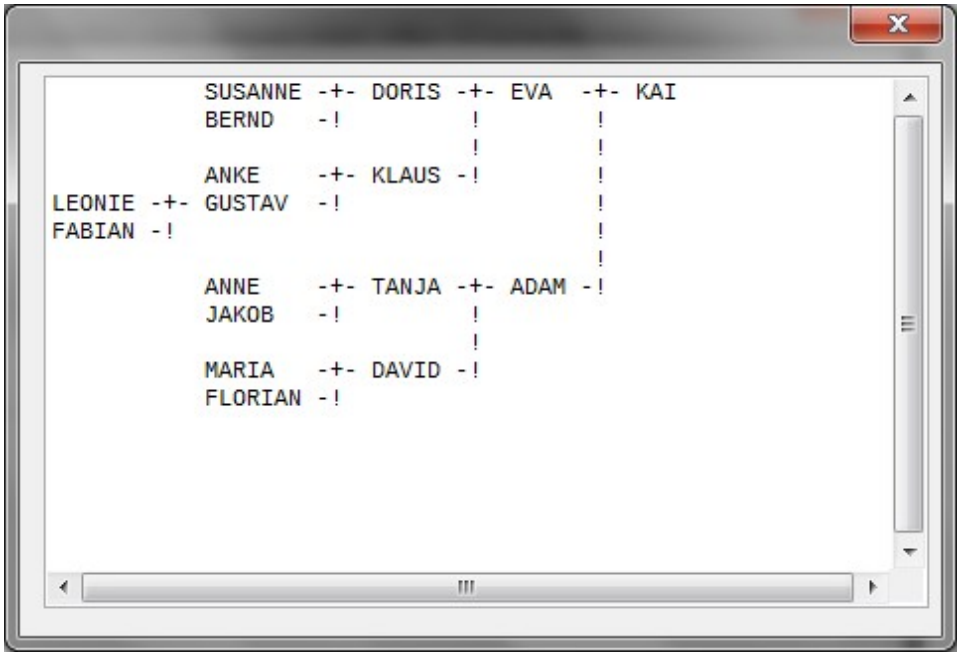

Abbildung 3: Ausgabe des Stammbaumes.

Auffallend an unserem Ergebnisbaum ist, das dieser nicht in gewohnter Form verläuft, sondern von rechts nach links. Diese Tatsache läuft natürlich unbedingt in unseren Vorüberlegungen mit ein.

Unsere Namen die wir zum Erzeugen eines Stammbaumes benötigen sammeln wir in einer Tabelle. Jede Zeile bildet einen Datensatz bestehend aus einem Kind, die dazugehörige Mutter und den Vater. Die Liste ist ungeordnet und kann jederzeit erweitert werden.

Die Verknüpfung der Knoten erfolgt durch die Kindsnamen. Jede Mutter oder Vater ist nun mal auch gleichzeitig Kind.

Kind	Mutter	Vater
David	Maria	Florian
Tanja	Anne	Jakob
Adam	Tanja	David
Klaus	Anke	Gustav
Doris	Susanne	Bernd
Eva	Doris	Klaus

Kind	Mutter	Vater
Kai	Eva	Adam
Birgit	Maria	Florian
Ralf	Maria	Florian
Nina	Anne	Jacob
Tim	Anne	Jakob
Sophie	Nina	Ralf
Mia	Anke	Bernd
Pia	Anke	Bernd
Moritz	Susanne	Gustav
Finn	Susanne	Gustav
Max	Pia	Finn
Sarah	Sophie	Max
Gustav	Leonie	Fabian

Tabelle 1: Namensliste mit Mutter und Vater

2.2. Vorüberlegungen

Betrachten wir also die Struktur unseres Ergebnisbaumes. Wie oben bereits erwähnt ist der Baum von rechts nach links aufgebaut. Wir müssen uns also Gedanken darüber machen, wie die Zeichenkette (String) geschrieben bzw. ausgegeben wird. Für den Anfang ist diese Betrachtung allerdings vernachlässigbar. Diese Aufgabe stellen wir erst mal zurück.

Ausgehend vom Wurzelknoten „Kai“ sieht man das alle *davor* geschriebenen Knoten Mütter sind. Die Mutternamen bilden eine komplette Zeile, wobei die einzelnen Personen mit einer Zeichenkette „-+-“ verbunden sind. Zwischen Namen und der Zeichenkette befinden sich noch Leerzeichen. Die Anordnung der Mütter lässt auf eine **Preorder Traversierung** schließen. Am Ende der Zeile, ganz rechts, gibt es einen Wagenrücklauf.

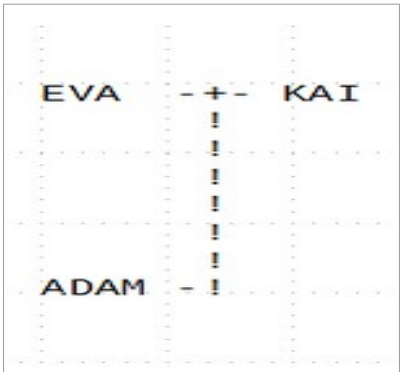

Abbildung 4: Spalte

Jeder Vater wird in eine eigene Zeile geschrieben, dem Namen kann aber eine Mutter oder auch mehrere folgen. Wir erinnern uns das der Baum von rechts nach links aufgebaut ist.

Ist der Baum unausgewogen kann auch ein leerer Bereich vor dem Namen stehen, wie im Fall von „Bernd“. Das gilt für Mutter und Vater gemeinsam.

Die Tatsache das es sich um eine eigene Zeile handelt, impliziert auch hier wieder einen Wagenrücklauf am Ende der Zeile.

Alle Namen die eine Spalte bilden haben die gleiche Breite obwohl die Namen unterschiedliche Längen aufweisen. Dies ist ein sehr wichtiger Umstand der zur Erstellung des Strings von besonderer Bedeutung ist. Die leeren Bereiche scheinen ebenfalls eine Abhängigkeit der Feldbreite aufzuweisen.

Rechts vom Vater befindet sich eine Zeichenkette bestehend aus einem Bindestrich und einem Ausrufungszeichen („-!“). Weiter sehen wir das dem Vater ein Ausrufungszeichen „!“ zeilenmäßig nach unten hin folgt. Das Ausrufungszeichen stellt visuell den Kontakt zum Partner dar.

Weiterhin fällt auf, wenn der letzte Knoten einer Zeile ganz links ein Vater ist, dann folgt ein Wagenrücklauf mit einer Leerzeile. Befindet sich der Name aber innerhalb der Struktur, enthält die Zeile die Ausrufungszeichen eines übergeordneten Paares. Im Falle von Bernd sind das zwei Ausrufungszeichen. Zwischen diesen Ausrufungszeichen befinden sich auch wieder leere Bereiche. Hier scheint es wohl einige Berechnungen zu geben, damit die Ausrufungszeichen auch tatsächlich untereinander stehen.

Zusammenfassend lässt sich also festhalten:

1. Es gibt zwei **Knotentypen**, nämlich Mutter und Vater. Eine Ausnahme gilt für die Wurzel.
2. Die Mutterknoten bilden eine Zeile.
3. Alle Namen einer Spalte bzw. Ebene haben die gleiche Breite. Wir sprechen von einer **Feldbreite.**
4. Es existieren zwei unterschiedliche **Knotenverbinder,** „-+-“ für die Mutter und „-!“ für den Vater. Die Verbinder stehen hinter dem jeweiligen Knoten mit jeweils einem Leerzeichen Abstand.
5. Der Baum kann unausgeglichen sein und wird mit leeren Feldern aufgefüllt. Diese leeren Felder nennen wir **Platzhalter**.
6. Ist der letzte Knoten einer Zeile ein Vaterknoten, dann folgt eine Leerzeile.
7. Eine Leerzeile enthält Platzhalter und Ausrufungszeichen.

2.3. Planung eines Objektes

Wie man sieht sammeln sich für einen Knoten mehr Informationen als nur der Name.

Jeder Knoten findet sich auf seiner Ebene wieder. Man könnte die Ebenen bei jeder Suche zählen um zu einem Knoten zu gelangen. Es gibt allerdings keinen Grund diese Information nicht schon bei der Konstruktion des Baumes im Knoten zu speichern. Ganz im Gegenteil, es werden so zusätzliche Vergleichs- und Zähloperationen vermieden.

Auch für die Feldlänge lassen sich wiederholende Berechnungen vermeiden, wenn wir die Daten direkt dem Knoten mitgeben.
Die Feldlänge setzt sich aus der Anzahl der Zeichen des Namens, und aus dem Unterschied des längsten Namens der Knoten der gleichen Ebene zusammen. Die

Differenz bestimmt die Anzahl Leerzeichen die zum Auffüllen benötigt werden um auf die gleiche Länge zu kommen.

Feldlänge(Knoten) = NamenLänge + (MaxFeldLänge(Ebene) – NamenLänge(Knoten)

Auch der Knotentyp sollte ebenfalls in das Objekt mit aufgenommen werden. Also eine Information ob es sich selbst um eine Mutter= 1 , Vater= 2 oder die Wurzel= 0 handelt.

Zur Verkettung des Baumes benötigen wir selbstverständlich die Mutter- und/oder Vaterknoten als Personobjekt selbst. Diese werden zur rekursiven Betrachtung herangezogen.

Und zu guter Letzt speichern wir auch den **Parentknoten**. Der Parent liefert uns die Informationen zur Erstellung der Platzhalter hinter unserer Bezugsperson. Der Knoten ist der übergeordnete Knoten im Suchbaum.

Somit haben wir alle Informationen für ein Knoten zusammen. Die Daten werden wir der Person Huckepack mitgeben. Die Variablen haben nun die nachfolgenden Eigenschaften.

```
PersonName  As String
NameLaenge  As Integer
FeldLaenge  As Integer
FeldString  As String
Verbinder   As String
Mutter      As Object
Vater       As Object
Ebene       As Integer
isType      As Integer
isParent    As Object
```

Wenn eine Indizierung der einzelnen Knoten gewünscht ist, kann eine weitere Variable in das Objekt mit aufgenommen werden. Ich habe allerdings darauf verzichtet.

Bestückt werden die Variablen durch die Routinen im Modul Person.

Modul Person: Details

Eine Person wird als Personobjekt angelegt. Das Objekt selbst bildet also unseren einzelnen Knoten. Das bedeutet das wir zunächst eine Type Variable definieren müssen, in dem die Eigenschaften als Struktur hinterlegt sind. In Java wird von einer Klasse gesprochen in der auch die Geter – und Seter Funktionen abgelegt sind.

Unser Modul mit diesen Code nennen wir Person.

Globale Einstellungen des Moduls

Im Kopfbereich des Moduls wird die Type Variable definiert. Anschließend folgen die Konstanten des Moduls die wir als Public deklarieren. So können wir die Konstanten auch in den anderen Modulen nutzen. Die Konstanten sind enorm hilfreich um die Übersicht im Code zu behalten.

```
01  Option Explicit
```

```
02
03  Type Person
04          PersonName      As String
05          NameLaenge      As Integer
06          FeldLaenge      As Integer
07          FeldString      As String
08          Verbinder       As String
09          Mutter          As Object
10          Vater           As Object
11          Ebene           As Integer
12          isType          As Integer
13          isParent        As Object              'Person Objekt
14
15  End Type
16
17  public Const _Vater  = 2
18  public Const _Mutter = 1
19  Public Const _Wurzel = 0
20
```

Sub CreatePerson

Die Prozedur ist der Konstruktor für die Person bzw. dem Knoten. Der Name wird als Argument übergeben. Die Länge des Strings Name wird berechnet und in der Variablen NameLaenge abgelegt. FeldLaenge und NameLaenge sind zur Erstellung der Person erst mal gleich.

Übergabeparameter:

sName as String	Name der Person
iNr as String	Knotennummer

Rückgabewert:

As Object	Das Personobjekt, also der Knoten

```
01  Sub CreatePerson (sName As String ) As Object
02  Dim oPerson As Person
03
04          If sName = "" Then Exit Sub
05
06          With oPerson
07                  .PersonName = sName
08                  .NameLaenge    = Len(sName)
09                  .FeldLaenge = .NameLaenge
10          End With
11
12          CreatePerson = oPerson
13
14  End Sub
```

Sub setNode

SetNode speichert den Folgeknoten als Mutter- oder Vaterknoten im Personobjekt ab. Ein `If Else` Abfrage legt fest wie der Knoten bearbeitet wird. Der String für die Verbindung wird erzeugt und der Knoten Type des Child als Vater oder Mutter initialisiert. Den Knoten Type übergeben wir als Argument, den wir durch die Position des Namens in der Namensliste kennen.
Zuletzt wird das **Personobjekt als Kindsknoten im Parent** abgelegt.

Übergabeparameter:

oPerson as Object	Der zu bearbeitende Knoten
oChild as Object	Der Vater oder Mutterknoten
iParentType as Integer	Der Knotentyp 1= Mutter, 2= Vater

```
01  Sub setNode (oPerson as Person, oChild As Person, iParentType As Integer )
02
03          If iParentType = _Mutter Then
04                  rem
05                  oChild.Verbinder = " -+- "
06                  oChild.isType = _Mutter
07                  oPerson.Mutter = oChild
08
09          ElseIf iParentType = _Vater Then
10                  rem
11                  oChild.Verbinder = " -!"
12                  oChild.isType = _Vater
13                  oPerson.Vater  = oChild
14
15          End If
16
17          oChild.isParent = oPerson
18
19  End Sub
```

Sub setFeldLaenge

Diese Prozedur speichert die Feldlänge des Personenobjektes und erzeugt den **Feldstring**. Der Feldstring setzt sich aus den Personennamen, und je nach Feldlänge, mit der Anzahl aufzufüllender Leerzeichen sowie dem Verbinder zusammen. Für die Leerzeichen nutzen wir die interne Funktion spc (Space).

Übergabeparameter:

oPerson as Person	Das Personobjekt

iLaenge as Integer	Die Feldlänge

```
01  Sub setFeldLaenge (oPerson As Person, iLaenge As Integer )
02  Dim oObj As Person
03          oObj = oPerson
04
05          With oObj
06                  .FeldLaenge = iLaenge
07                  .FeldString = .PersonName & spc(iLaenge - .NameLaenge) & .Verbinder
08          End With
09
10  End Sub
```

Sub isNodeType

Liefert den Knotentyp.

Rückgabewert:

isNodeType as Integer	0= Wurzel, 1= Mutter, 2= Vater, kann mit den Konstanten verglichen werden.

```
01  Sub isNodeType(oPerson As Person ) As Integer
02          isNodeType = oPerson.isType
03  End Sub
```

Somit haben wir alle erforderlichen Prozeduren für das Personobjekt zusammengestellt. Nun kann der Aufbau des Baumes erfolgen. Aus Gründen der Übersichtlichkeit habe ich für die Prozeduren die für diese Aufgabe zuständig sind, ein neues Modul Tree angelegt.

2.4. Der Suchbaum

Modul Tree: Details

Ziel dieses Moduls ist wie zuvor geschrieben das Erzeugen des Suchbaumes. Das Prinzip ist recht einfach. Zum Start wird der erste Knoten übergeben. Dieses Objekt ist die Wurzel unseres Baumes. Anhand des Namens holen wir uns zunächst den Namen der Mutter aus der Namensliste. Der Knoten wird konstruiert und im Personobjekt unseres Wurzelknotens in die Objektvariable Mutter abgelegt. Vorraussetzung ist natürlich das ein Name existiert.

Der erzeugte Mutterknoten wird nun als Argument als neuer Wurzelknoten an die Prozedur übergeben, und der Vorgang startet neu mit dem Namen der Mutter des übergeben Objektes. Auf dieser Weise werden zunächst alle linken Knoten abgearbeitet.

Wenn keine Mutter mehr existiert erfolgt der Abstieg und das gleiche Vorgehen erfolgt mit dem Vater.

Während dieser Initialisierungsphase zählen wir die Ebenen unseres Baumes. Das Ergebnis speichern wir in einer globalen Variablen.

Neben dem Konstruktor werden in diesm Modul weitere Prozeduren/Funktionen eingebettet die den Ergebnisbaum betreffen. Eine davon ist z.B. die Berechnung der Feldlänge.

Globale Einstellungen des Moduls

```
01  Option Explicit
02
03  Dim aListe() As Object
04  Dim iTiefe As Integer
05
```

Sub CreateTree

Erzeugt den Suchbaum und zählt die einzelnen Ebenen (Level). Der Aufruf der Prozedur erfolgt rekursiv. Die Integer Variable `iLevel` wird bei jedem Aufstieg inkrementiert. Beim Abstieg wird zur Korrektur das Level dekrementiert. Zwischen den beiden Operationen sichern wir den Wert in der globalen Variable `iTiefe`. Solange iTiefe kleiner als die aktive Ebene ist, wird der Wert überschrieben. Als Ergebnis erhalten wir die Gesamttiefe des Suchbaumes.

Wenn der Suchbaum komplett erzeugt und der Wurzelknoten wieder erreicht wurde erfolgt der Aufruf zur Feldlängenberechnung.

Übergabeparameter:

oRoot as Object	Der Wurzelknoten
iLevel as Integer	Die Ebene des Wurzelknotens

```
01  Sub CreateTree (oRoot As Object ,iLevel As Integer )
02  Dim sName As String
03  Dim oMutter As Object
04  Dim oVater As Object
05
06          rem Abbruch wenn Wurzelknoten Null ist
07          If isNull(oRoot) Then Exit Sub
08
09          rem Aufstieg zaehlen
10          iLevel = iLevel + 1
11
12          rem Mutterknoten erzeugen
13          sName = oRoot.PersonName
14          oMutter = CreatePerson (getElterVonListe(sName ,_Mutter ) )
15
16          If Not isNull(oMutter) Then
```

```
17                setNode(oRoot ,oMutter ,_Mutter)            'Knoten einfuegen
18                CreateTree (oMutter ,iLevel )
19        End If
20
21        rem Ebenen sichern
22        If iTiefe < iLevel  then
23                iTiefe = iLevel                                          'iTiefe
    globale Variable
24        End If
25
26        rem Aktuelle Ebene im Knoten speichern
27        oRoot.Ebene = iLevel
28
29        rem Vaterknoten erzeugen
30        oVater = CreatePerson (getElterVonListe (sName ,_Vater ) )
31
32        If Not isNull (oVater ) Then
33                setNode(oRoot ,oVater ,_Vater)                    'Knoten einfuegen
34                CreateTree (oVater ,iLevel )
35        End If
36
37        rem Abstieg zaehlen
38        iLevel = iLevel - 1
39
40        rem Wenn Wurzel wieder erreicht
41        If iLevel = 0 Then  setAllFieldLength(oRoot )
42
43 End Sub
```

Sub getAllObjFromLevel

Bevor wir die Feldlänge berechnen können, benötigen wir alle Knoten die auf einer Ebene zu finden sind. Dafür ist die `getAllObjFromLevel` Prozedur zuständig. Der gesamte Suchbaum wird durchlaufen, und immer wenn ein Knoten mit der Ebene iLevel gefunden wird, wird der Knoten einem Array angefügt, wobei die vorhandenen Werte nicht gelöscht werden dürfen (ReDim PreServe). Das Array ist global in diesem Modul deklariert als Variable `aListe()`.

```
01 Sub getAllObjFromLevel (oTree As Object , iLevel As Integer )
02 Dim Idx As Integer
03 Dim oKnoten As Object
04
05        rem Abbruch wenn Suchbaum fehlt
06        If isNull(oTree ) Then Exit sub
07
08        rem Ebene nicht richtig?
09        If oTree.Ebene < iLevel Then
10                oKnoten = oTree.Mutter
11                getAllObjFromLevel (oKnoten ,iLevel)
12        End If
13
```

```
14          rem Ueberpruefe auf richtige Ebene
15          If iLevel = oTree.Ebene Then
16                  rem wenn ja
17                  Idx = Ubound(aListe )
18                  rem Array um eine Feld vergroessern
19                  Idx = Idx + 1
20                  ReDim Preserve aListe(Idx )
21                  rem Knoten anhängen
22                  aListe(Idx ) = oTree
23          End If
24
25          rem Weiter mit rechter Seite ?
26          If oTree.Ebene < iLevel Then
27                  oKnoten = oTree.Vater
28                  getAllObjFromLevel (oKnoten ,iLevel)
29          End If
30
31  End Sub
```

Wenn alle Knoten einer Ebene im Array gesammelt wurden, können die Datensätze durchlaufen und untereinander verglichen werden. Wir suchen den Größten Wert aller Knoten der Ebene.

Sub MaxFieldLen

Diese Aufgabe übernimmt die Prozedur MaxFieldLen. Das Array aListe wird bis zum Ende durchlaufen. Die Feldlänge wird in einer Variablen zwischengespeichert. Solange der Wert kleiner als der des nächsten Datensatzes ist, wird Variable aktualisiert. Das Ergebnis wird als Integer an die aufrufenden Prozedur zurückgegeben.

```
01  Sub MaxFieldLen () As Integer
02  Dim ii As Integer
03  Dim iLaenge As Integer
04  Dim iStrLaenge As Integer
05
06          For ii = 0 To Ubound(aListe )
07
08                  If Not isNull(aListe(ii) ) Then
09                          rem Laenge aus der Person im Array
10                          iStrLaenge = aListe(ii).FeldLaenge
11
12                          If iLaenge <  iStrLaenge Then
13                                  iLaenge = iStrLaenge
14                          End If
15                  End If
16          Next
17          MaxFieldLen = iLaenge
18  End Sub
```

sub setFieldLenToLevel

Um Zugriff auf die Feldlänge zu bekommen müssen die Werte natürlich zuvor im Objekt gespeichert werden. Das ist eine recht simple Aufgabe die durch diese Prozedur

abgearbeitet wird. Wir nutzen den Vorteil aus, dass die Knoten der Ebene noch immer im Array gesammelt sind und durchlaufen es erneut.

Das ist eine reine Fleißarbeit. Übergabeparameter benötigt die Prozedur nicht und einen Rückgabewert gibt es auch nicht.

```
01 sub setFieldLenToLevel()
02 Dim ii As Integer
03 Dim iLength As Integer
04 Dim oObject As Object
05
06         iLength = MaxFieldLen(aListe)
07
08         rem Felder durchlaufen
09         For ii = 0 To Ubound(aListe )
10                 oObject = aListe(ii)
11                 setFeldLaenge(oObject , iLength )
12         Next
13
14 End Sub
```

Nun existieren in der Regel mehr als eine Ebene im Suchbaum. Also schreiben wir noch eine Prozedur, die nacheinander die einzelnen Ebenen im Array sammelt, und die Feldlängenberechnung anstößt.

Hier an dieser Stelle noch einmal zur Erläuterung. Die Feldlängenberechnung ist keine Berechnung in dem Sinne, sondern eine Vergleichsoperation, wodurch der größte Wert ermittelt wird.

Sub setAllFieldLength

Also durchläuft diese Prozedur Ebene für Ebene und setzt die Feldlänge in die Variable der Knoten. Vor jedem Durchlauf wird das Array neu Dimensioniert, d.h. eigentlich gelöscht. Dann werden die Knoten der Ebene `ii` im Array gesammelt und die Berechnung angestoßen.

Übergabeparameter:

oTree as Object	Der Suchbaum

```
01 Sub setAllFieldLength(oTree As Object )
02 Dim ii%
03
04         rem Hole alle Ebenen
05         For ii = 1 To iTiefe
06                 rem Array leeren
07                 ReDim aListe()
08                 rem Knoten holen
```

```
09                  getAllObjFromLevel(oTree ,ii)
10
11                  setFieldLenToLevel
12
13          Next
14
15  End Sub
```

Bei der Erstellung des String's stoßen wir auf ein Problem. Beim Erreichen des letzten Knoten eines Zweiges, ist im Falle der Unausgeglichenheit, kein Childknoten verfügbar auf dem wir direkt die Feldlänge entnehmen können. Aus diesem Grund müssen wir eine Suche starten um die Länge für den Platzhalter zu ermitteln. Das erledigt die Prozedur getFieldLenFromLevel.

Sub getFieldLenFromLevel

Der Baum wird auch hier wieder rekursiv durchlaufen. Sobald ein Knoten des Levels gefunden wurde, wird das Ergebnis als Rückgabewert gesichert und die Suche abgebrochen. Ein Abbruch der Routine ist durchaus vorteilhaft, da die Feldlänge in jedem der Knoten zu finden ist. Ein erneuter Aufstieg eines anderen Zweiges macht keinen Sinn und verschwendet nur unnötig viele Takte.

Erzwungen wird der Abbruch in dem im Traversierungsbereich IN- und POST Order eine Überprüfung des Rückgabewertes erfolgt. Wenn dieser ungleich 0 ist wurde die Ebene gefunden.

Übergabeparameter:

oTree as Object	Der Suchbaum
iLevel as Integer	Das zu suchende Level

Rückgabewert:

getFieldLenFromLevel as Integer	Feldlänge

```
01  Sub getFieldLenFromLevel (oTree As Object ,iLevel As Integer  ) As Integer
02  Dim ii As Integer            'sichert die Feldlaenge
03  Dim oKnoten As Object
04
05          rem Abbruch wenn Baum Null ist
06          If isNull(oTree ) Then Exit sub
07
08          If iLevel = oTree.Ebene Then
09          rem Wenn Level gefunden wurde
10                  ii = oTree.FeldLaenge
11                  getFieldLenFromLevel = ii
12                  Exit Sub
13          End If
14
15          rem Mutterknoten traversieren wenn Level noch nicht erreicht
```

```
16          If oTree.Ebene < iLevel Then
17                  oKnoten = oTree.Mutter
18                  ii = getFieldLenFromLevel (oKnoten ,iLevel)
19          End If
20
21          rem Ueberpruefung des Rueckgabewertes
22          If ii <> 0 Then
23                  getFieldLenFromLevel = ii
24                  Exit Sub
25          End If
26
27          rem Vaterknoten traversieren wenn Level noch nicht erreicht
28          If oTree.Ebene < iLevel Then
29                  oKnoten = oTree.Vater
30                  ii = getFieldLenFromLevel (oKnoten ,iLevel)
31          End If
32
33          rem Ueberpruefe den Rueckgabewert
34          If ii <> 0 Then
35                  getFieldLenFromLevel = ii
36                  Exit Sub
37          End If
38
39  End Sub
```

Somit ist die Erstellung des Suchbaumes abgeschlossen, und wir können beginnen den Ausgabestring zu konstruieren.[1]

2.5. Die Konstruktion des Ausgabestrings

Wie ich bereits in unseren Vorüberlegungen geschrieben habe, müssen wir uns Gedanken machen wie der String aufgebaut wird. Es gibt Aufstiege im Baum, aber auch Abstiege. Dazu kommt der Wagenrücklauf der an der richtigen Stelle erfolgen muss. Man kann also schon erahnen das sich dort ein paar Fallstricke offenbaren.

Gehen wir also Stück für Stück, oder besser gesagt, Zeile für Zeile vor. Dazu legen wir eine Variable `sText` an, in der wir zunächst eine komplette Zeile aufbauen. Wenn diese Zeile fertiggestellt ist, sichern wir diese in unserem Ergebnisstring. Der Ergebnisstring sammelt den gesamten Stammbaum als Text und wird in einem Schwung ausgegeben.

Die Variable für das Ergebnis nennen wir `sErgTxt`.

Da unser Stammbaum von rechts nach links aufgebaut ist aber der Rechner oder der Drucker die Zeichen von links aufbaut, müssen wir jeden Knoten den wir anfügen, tatsächlich ***vor*** unseren String schreiben.

```
sText = Knoten & sText
```

Das gilt so weit für den Aufstieg im Suchbaum. Beim Abstieg ist es genau umgekehrt, was nun den der Schreibweise von links nach rechts entspricht.

1

Jeden Wagenrücklauf habe ich zunächst mit einem Tag **<NL>** simuliert. Dieser Schritt ist eigentlich nicht erforderlich und benötigt eine zusätzliche Operation bzw. Prozedur um den tatsächlichen *Carriage Return* einzufügen. Aber zum Debuggen ist diese Vorgehensweise enorm hilfreich.

2.6. Modul String: Details

Globale Einstellungen des Moduls

```
01  Option Explicit
```

Im Modul String sind die Prozeduren zur Stringkonstruktion abgelegt. Der Konstruktor ist *CreateText*. Der *Placeholder* spielt eine zentrale Rolle zur Erzeugung des Strings. Es existieren Platzhalter die vor einem Bezugsknoten stehen, und es existieren Platzhalter die hinter ihm stehen. Die folgenden Konstanten steuern die Art, welcher der Placeholder geschrieben werden soll.

- _front schreibt den Placeholder als reine Leerzeichen.
- _back schreibt den Placeholder mit Leerzeichen und evtl. dem Spaltentrenner "!".

```
01  rem Die Konstanten
02  rem --------------
03  Public Const _front   = False
04  Public Const _back    = True
05
```

Sub PlaceHolder

Diese Prozedur konstruiert den Platzhalter hinter einem Bezugsknoten. Die Feldlänge entnehmen wir dem Parent. Aber nur dann wenn ein Parent auch vorhanden ist. Treffen wir auf die Wurzel wird kein Platzhalter benötigt.

Der Abstand von Feld zu Feld beträgt 5 Zeichen. Das entspricht dem Verbinder der Mutterknoten einschließlich der Leerzeichen vor und hinter dem Verbinder „ -+- „.

Ist der Parent nun ein Mutterknoten, bedeutet das, dass wir uns innerhalb der Baumstruktur befinden. In diesem Fall müssen wir die Länge des Platzhalters um eins weniger korrigieren und an der fünften Stelle das Ausrufungszeichen setzen. Um das zu erreichen nutzen wir die interne Basic Funktion Left().

Der Platzhalter ist nun konstruiert und kann an die aufrufende Prozedur geliefert werden.

Übergabeparameter:

oTree as Object	Der Suchbaum

oPerson as Object	Der Bezugsknoten

Rückgabewert:

Placeholder as String	Der Platzhalter als Zeichenkette

```
01  Sub PlaceHolder (oTree as Object ,oPerson As Object ) As String
02  Dim sPlaceHolder As String
03  Dim iWidth              As Integer    'Feldlaenge
04  Dim iLevel              As Integer
05
06          iLevel  = oPerson.Ebene
07          rem wenn Platzhalter vor den Knoten
08
09                  rem nur wenn Parent vorhanden, sonst nix machen
10                  If Not isNull(oPerson.isParent ) Then
11                                  iWidth = oPerson.isParent.FeldLaenge
12                                  sPlaceHolder = Space(iWidth + 5)
13                                  rem Ueberpruefe ob Parent eine Mutter ist
14                                  If isNodeType(oPerson.isParent) = _Mutter Then
15                                          rem Laengenkorrektur fuer Ausrufungszeichen
16                                          sPlaceHolder = Left(sPlaceHolder, iWidth + 4 ) & "!"
17                                  End If
18                  End If
19
20          rem fertig
21          PlaceHolder = sPlaceHolder
```

Sub AllPlaceHolder

Je nach Ebene des Bezugsknotens werden mehrere Platzhalter in einer Zeile benötigt. Im Falle von `_back` wird ab der Ebene oberhalb des Bezugsknotens bis auf die Ebene 2 runtergezählt und der Platzhalter für jede Ebene konstruiert. Ebene 1 ist die Wurzel und hat nur ein Feld. Den Knoten brauchen wir nicht zu beachten. Die Vorgehensweise muss allerdings zum besseren Verständnis besonders hervorgehoben werden.

> **Wichtig**: Solange der Parent ein Vaterknoten ist, werden die Platzhalter in der Variablen **`tmp`** Ebene für Ebene gesammelt. Trifft man beim runterzählen auf einen Mutterknoten, dann werden die gesammelten Platzhalter geschrieben. Sind aber alle Parents Väter und man trifft auf die Wurzel, dann **verfallen** die Platzhalter. Der Grund ist, wir befinden uns nun außerhalb der Baumstruktur und es wird statt eines Platzhalters nur ein Wagenrücklauf benötigt.

Wenn die Position `_front` ist, wird von der tiefsten Ebene bis zur Ebene unterhalb des Bezugsknotens runtergezählt. Da es keine direkten Knoten gibt wird der Platzhalter künstlich konstruiert. Alle Platzhalter bestehen in diesem Fall nur aus Leerzeichen. Hier muss auch keine Länge korrigiert werden.

Die Platzhalter haben immer die Länge *FeldLänge + 5* und brauchen nur Ebene für Ebene aneinandergereiht zu werden.

An dieser Stelle kommt nun unsere Routine `getFieldLenFromLevel` zum Einsatz.

Übergabeparameter:

oTree as Object	Der Suchbaum
oPerson as Object	Der Bezugsknoten
bPlace as Boolean	Die Position des Platzhalters

Rückgabewert:

AllPlaceHolder as String	Alle Platzhalter vor/ oder nach Bezugsknoten in der Zeile.

```
01  Sub AllPlaceHolder (oTree as Object ,oPerson As Object , bPlace As boolean ) As String
02  Dim sPlaceHolder As String
03  Dim iWidth As Integer                       'Feldlaenge
04  Dim iLevel As Integer
05  Dim oObject As Object                       'Nimmt den Bezugsknoten auf
06  Dim ii%
07  Dim tmp$
08
09          oObject = oPerson
10
11          iLevel = oObject.Ebene
12
13          rem Wenn der Platzhalter hinter dem Knoten kommt
14          If bPlace = _back Then
15
16                      rem Durchlaufe innere Knoten bis Level 2
17                      For ii = iLevel - 1 To 2 Step -1
18                              If isNodeType(oObject.isParent ) = _Mutter Then
19                              rem Setze String zusammen
20                              sPlaceHolder = sPlaceHolder & tmp & PlaceHolder(oTree
    ,oObject)
21                              Else
22                                      rem sammel Platzhalter wenn Vaterknoten
23                                      tmp = tmp & PlaceHolder (oTree , oObject )
24                              End If
25
26                              rem Jetzt Parent holen
27                              oObject = oObject.isParent
28                      Next
29
30          rem wenn der Platzhalter davor kommt
31          Else
```

```
32                  For ii = iTiefe To iLevel + 1 Step - 1
33                          iWidth =  getFieldLenFromLevel(oTree , ii)
34                          rem 5 Leerzeichen fuer Verbinder
35                          iwidth = iWidth + 5
36                          rem Setze String zusammen
37                          sPlaceHolder = sPlaceHolder & Space(iWidth )
38                  Next
39
40          End If
41
42          AllPlaceHolder = sPlaceHolder
43
44  End Sub
```

Sub BlankLine

Diese Prozedur schreibt eine komplette Leerzeile. Leerzeile ist als Aussage so nicht richtig, weil es wie oben beschrieben, durchaus Zeichen zur Ausgabe gibt. Nämlich das Ausrufungszeichen als Spaltentrenner. Ich bezeichne es als Leerzeile weil kein Knotenname ausgegeben wird.

Zuerst werden alle Platzhalter vor dem Bezugsknoten erzeugt. Dann folgen die Leerzeichen die aus der Länge des *Feldstrings* generiert werden, also bestehend aus der Feldlänge und dem Verbinder des Knotens. Das ist jetzt sehr wichtig zu verstehen.

Abschließend werden alle Platzhalter nach dem Namen erzeugt.

Übergabeparameter:

oTree as Object	Der Suchbaum
oPerson as Object	Der Bezugsknoten

Rückgabewert:

BlankLine as String	Die Leerzeile im Stammbaum

```
01  Sub BlankLine(oTree As Object ,oPerson As Object ) As String
02  Dim sText As String
03  Dim tmp As String
04  Dim iLevel As Integer
05
06          rem Platzhalter Zeilenanfang
07          tmp = AllPlaceHolder(oTree ,oPerson  ,_front )
08          sText = tmp
09          rem Jetzt Platzhalter fuer Knoten
10          tmp = Space( Len (oPerson.Feldstring ) )
11          sText = sText & tmp
```

```
12          rem Platzhalter fuer Rest der Zeile
13          tmp = AllPlaceHolder (oTree ,oPerson ,_back )
14         sText =  sText & tmp
15
16          rem Fertig
17          BlankLine = sText
18  End Sub
```

Alle Prozeduren die wir zum Bearbeiten oder Erzeugen des Strings benötigen haben wir jetzt soweit zusammengestellt. Nun können wir uns daran machen den kompletten Stammbaum zu erzeugen.

Sub CreateText

Zuerst wird überprüft ob der Bezugsknoten ein Vaterknoten ist. Wenn ja, dann wird der Platzhalter im `_back` Modus erzeugt. Ein Mutterknoten bekommt keinen Platzhalter.

Jetzt wird der Knoten erzeugt und dem String vorangestellt. Das kann ein Vater- oder Mutterknoten sein, egal, er wird auf jeden Fall geschrieben.

Als nächstes werden zwei Bedingungen überprüft. Wenn es keinen Mutterknoten mehr gibt ist die Zeile eigentlich fertiggestellt. Ist der Baum aber unausgeglichen, was dann der Fall ist wenn die Ebene des Knotens nicht der vollen Tiefe des Baumes entspricht, dann wird ein Platzhalter im `_front` Modus benötigt.

Im Anschluss erfolgt der rekursive Aufruf. Natürlich nur wenn weitere Knoten vorhanden sind.

Vor dem Abstieg wird noch einmal geprüft ob der Bezugsknoten ein Vater oder eine Mutter ist. Ist es ein Mutterknoten, wird ein Wagenrücklauf an die Zeile angehängt und die Zeile ist fertig. Wenn es aber ein Vaterknoten ist **und** dessen *Parent* eine Mutter (Wir befinden uns in diesem Fall innerhalb der Baumstruktur), dann wird zuerst ein Wagenrücklauf mit einer Leerzeile geschrieben.

Übergabeparameter:

oPerson as Object	Der Bezugsknoten rekursiv
oTree as Object	Der Suchbaum

```
01  Sub CreateText (oPerson As Object ,oTree As Object )
02  Dim oMutter As Object
03  Dim oVater As Object
04  Dim sFront As String         'Sichert den Platzhalter vor dem Knoten
05  Dim sBack  As String         'Sichert den Platzhalter hinter dem Knoten
06  Dim iNodeType As Integer
07
08  rem PREORDER
09  rem --------
10
```

```
11          If isNodeType(oPerson ) = _Vater Then
12          rem Platzhalter holen wenn Vater
13                  sBack = AllPlaceHolder(oTree ,oPerson , _back)
14                  stext = sText & sBack
15
16          End If
17
18          rem Schreibe Knoten
19          sText = oPerson.FeldString & sText
20
21          rem Wenn keine Mutter vorhanden ist
22          If isNull(oPerson.Mutter) Then
23
24                  rem Schreibe Platzhalter wenn volle Tiefe noch nicht erreicht
25                  If oPerson.Ebene < iTiefe Then
26                          sFront = AllPlaceHolder(oTree ,oPerson , _front)
27                          rem Platzhalter vor die Person
28                          sText = sFront & sText
29                  End If
30          End If
31
32          rem Mutterknoten traversieren wenn vorhanden
33          If Not isNull(oPerson.Mutter ) Then
34                  oMutter = oPerson.Mutter
35                  CreateText (oMutter ,oTree )
36          End If
37
38          rem Vaterknoten traversieren wenn vorhanden
39          If Not isNull(oPerson.Vater ) Then
40                  oVater = oPerson.Vater
41                  CreateText (oVater ,oTree )
42          End If
43
44  rem POSTORDER
45  rem ---------
46          If isNodeType (oPerson ) = _Vater Then
47                  rem Ueberpruefe Parent Type
48                  If isNodeType (oPerson.isParent ) = _Mutter Then
49                          sText = sText & "<NL>"                          'Wagenruecklauf
50
51                          rem Leerzeile holen wenn der Parent eine Mutter ist
52                          sBack = BlankLine (oTree ,oPerson)
53                          stext = stext & sBack
54                  End If
55          Else
56                  rem Zeile fertig
57                  sText  = sText & "<NL>"                                 'Wagenruecklauf
58                  ErgTxt = ErgTxt & sText
59                  sText  = ""
60          End If
61  End Sub
```

Nach der Erstellung der Module mit seinen Subroutinen benötigen wir jetzt noch die Startprozedur, die uns die Daten mit den Namen der Personen initialisiert und zugreifbar macht.

Wie wir oben in der Tabelle sehen besteht jeder Datensatz aus drei Namen. Kind – Mutter – Vater. Die Namen sind tabellarisch abgelegt, so das wir hierfür ein Array nutzen können. Das Feld bekommt nur eine Dimension. Der Kindsname bildet immer die Wurzel. Zur Verkettung bildet ein Elternteil den Wurzelknoten im Unterbaum.

```
Array("DAVID", "MARIA", "FLORIAN", _
      "TANJA", "ANNE", "JAKOB", _
      "ADAM", "TANJA", "DAVID", _
      "KLAUS", "ANKE", "GUSTAV", _
      "DORIS", "SUSANNE", "BERND", _
      "EVA", "DORIS", "KLAUS", _
      "KAI", "EVA", "ADAM", _
      "BIRGIT", "MARIA", "FLORIAN", _
      "RALF", "MARIA", "FLORIAN", _
      "NINA", "ANNE", "JAKOB"   , _
      "TIM", "ANNE", "JAKOB"  , _
      "SOPHIE", "NINA", "RALF" , _
      "MIA", "ANKE", "BERND", _
     "PIA", "ANKE", "BERND"   , _
      "MORITZ", "SUSANNE", "GUSTAV", _
      "FINN", "SUSANNE", "GUSTAV", _
      "MAX", "PIA", "FINN", _
      "SARAH", "SOPHIE", "MAX", _
      "GUSTAV", "LEONIE", "FABIAN")
```

Der Zugriff auf den Kindsknoten erfolgt in dem wir dem Index immer 3 (Drei Namen pro Datensatz) hinzu addieren. Da der Zugriff öfters erfolgt, benötigen wir auch dafür eine Subroutine.

Modul Module1: Details

Globale Einstellungen des Moduls

```
01  Option Explicit
02
03  Dim sAbListe() As String
04  Dim iBreite As Integer
05
06
```

Sub getElterVonListe

```
01  Sub getElterVonListe (sElter As String ,iElterType As Integer ) As String
02  Dim ii As Integer
03          For ii = 0 To Ubound(sAbListe ) Step 3
04                  If sAbListe(ii) = sElter Then
```

```
05                      If iElterType = _Mutter Then
06                              getElterVonListe = sAbListe (ii + 1) 'Mutter
07                      ElseIf iElterType = _Vater Then
08                              getElterVonListe = sAbListe (ii + 2) 'Vater
09                      End If
10                      Exit For
        'Schleife verlassen
11              End If
12
13      Next
14
15  End Sub
```

Sub Main

Startet das Hauptprogramm. Die Namensliste wird initialisiert und der gewünschte Wurzelknoten erzeugt. Anschließend wird der Suchbaum aufgebaut, dazu muss die Wurzel übergeben werden. Der Suchbaum befindet sich nun als Verkettung in der Person die als Wurzel erzeugt wurde. In unserem Beispiel ist das KAI.

Das Objekt wird anschließend zum erzeugen des Strings zweimal übergeben. Einmal als Suchbaum und einmal als Person. Der Ergebnisstring wird aufgebaut und muss nun noch mit dem Wagenrücklauf aufbereitet werden. Das übernimmt die Funktion REPLACE.

Zum Ende der Prozedur wird die Dialogbox erzeugt und der String zur Ausgabe an das entsprechende Textfeld übergeben. Abschließend wird die Dialog Box gestartet in dem nun der String enthalten ist.

```
01  Sub Main
02  Dim oPerson As Object
03  Dim tmp As String
04  Dim ii%
05
06  REM Werte zuweisen
07  REM -------------        Mutter,  Vater
08  sAbListe = Array("DAVID", "MARIA", "FLORIAN", _
09                               "TANJA", "ANNE", "JAKOB", _
10                               "ADAM", "TANJA", "DAVID", _
11                               "KLAUS", "ANKE", "GUSTAV", _
12                               "DORIS", "SUSANNE", "BERND", _
13                               "EVA", "DORIS", "KLAUS", _
14                               "KAI", "EVA", "ADAM", _
15                               "BIRGIT", "MARIA", "FLORIAN", _
16                               "RALF", "MARIA", "FLORIAN", _
17                               "NINA", "ANNE", "JAKOB"   , _
18                               "TIM", "ANNE", "JAKOB"  , _
19                               "SOPHIE", "NINA", "RALF" , _
20                               "MIA", "ANKE", "BERND", _
21                              "PIA", "ANKE", "BERND"   , _
```

```
                                "MORITZ", "SUSANNE", "GUSTAV", _
                                "FINN", "SUSANNE", "GUSTAV", _
                                "MAX", "PIA", "FINN", _
                                "SARAH", "SOPHIE", "MAX", _
                                "GUSTAV", "LEONIE", "FABIAN")

rem Start Code
rem --------------------------
      oPerson= CreatePerson("KAI")
      CreateTree (oPerson ,_Wurzel )
      CreateText(oPerson ,oPerson )

      ErgTxt = replace(ErgTxt ,"<NL>", chr(10) )

      Dim oDialog As Object
      Dim oTextField As Object

      oDialog = CreateUnoDialog(DialogLibraries.Standard.Dialog1)
      oTextField = odialog.getControl("TextField1")

      oTextField.Text = ErgTxt


       oDialog.execute()


End Sub
```

Die Person KAI kann natürlich durch jede andere Person aus der Liste ersetzt werden. Zum Beispiel hat auch SARAH eine interessante Ahnentafel. Der Code muss natürlich entsprechend abgeändert werden in dem man den Namen beim Aufruf der Prozedur ersetzt.

```
oPerson= CreatePerson("SARAH")
```

Damit ist die Erstellung unseres Stammbaumes abgeschlossen.