o++o auf 17 Seiten

(Stand 26.02.2020)

Klaus Benecke

Facebook und Co. arbeiten daran Nachrichten so aufzubereiten, dass sie emotional noch mehr ansprechen, als ob die gesellschaftliche Situation nicht schon aufgeheizt genug ist. Wir arbeiten daran dem Endnutzer Werkzeuge bereitzustellen um seine rationale Urteilskraft mit Hilfe des Computers zu stärken. Dafür benötigt man möglichst einfache aber dennoch leistungsstarke Programmiersprachen und umfangreiche, vertrauenswürdige, öffentlich zugängliche Informationen in Form von vielgestaltigen großen Tabellen und Dokumenten ähnlich der Wikipedia.
Auch wenn die entwickelte Sprache so einfach wie möglich ist, wird sie im Gegensatz zum „Facebookansatz“ einen gewissen Lernaufwand erfordern.
Eine solche Programmiersprache in Kombination mit vertrauensvollen Daten könnte ein großer Schritt in Richtung einer weiteren Demokratisierung der Gesellschaft werden. Viele Falschnachrichten könnten leicht von jedermann durch entsprechende Fakten oder statistischen Auswertungen paralysiert werden.
Vielleicht kann man die Schaffung einer solchen Programmiersprache mit der Schaffung des ersten Alphabets durch die Phönizier oder der Schaffung des ersten Alphabets mit Vokalen durch die Griechen vergleichen. Hätten diese Völker solche Leistungen vollbringen können ohne diese Voraussetzungen. Ich vermute ohne dieses Alphabet hätte es keine griechische Wissenschaft und Kultur gegeben; vielleicht auch keine griechische Demokratie.

Entwurfskriterien für eine solche Sprache:

1. Eine mathematische Fundierung ist erforderlich.
2. Methodisch-didaktische und pragmatische Fragen stehen zunächst vor Effizienzproblemen.
3. Kurze, lesbare Programme; die wichtigsten Schlüsselworte sollten kurz sein
4. Einfache, unstrukturierte Programme; Schleifen führen häufig zu schwer lesbaren und schwer änderbaren Programmen; Rekursion erfordert ein relativ hohes Abstraktionsniveau
5. Universelle Anwendbarkeit; sie muss nicht nur für Relationen (flache einfache Tabellen) sondern auch für strukturierte Tabellen und Dokumente nutzbar sein; sie muss nicht nur für Anfragen an die wichtigsten Systeme sondern auch für vielfältige Berechnungen geeignet sein
6. Um im Schulunterricht einsetzbar zu sein, muss sie die verschiedenen mathematische Teilgebiete unterstützen, sowie Nutzen für die anderen Fächer bieten
7. Sie sollte so mächtig sein, dass sie andere Systeme und Sprachen wie Tabellenkalkulation und SQL ersetzen kann.
8. Aus Endnutzersicht darf es nur ein einheitliches System mit einheitlicher Syntax (Schreibweise) für die Verarbeitung von Massendaten geben, genau wie die Operationen der Einzeldatenverarbeitung (+ - * : sqrt sin …) standardisiert sind.

o++o, genauer ottoPS (ottoProgrammierSprache), ist ein verbesserter Taschenrechner, der auch Anfragen an strukturierte TABellen und DokuMENTe (Tabmente) erlaubt. o++o ist damit auch geeignet Datenbankanfragen zu ermöglichen und später auch Basis einer Suchmaschine zu sein.
o++o wurde so entworfen, dass die Programme möglichst kurz, mit einfacher Syntax und übersichtlich sind. Die folgenden Programme sollte man unter ottoPS.de ausprobieren, da man Programmieren kaum ohne eigenes Testen erlernen kann.

Programm 1: Berechne die vierte Wurzel aus 2.

```
2 sqrt sqrt
```

Man erkennt, dass einstellige Operationen nach dem Inputwert geschrieben werden (postfix). Dadurch sparen wir gegenüber der bekannten Schreibweise sqrt(sqrt(2)) 4 Klammern.

Programm 2: Berechne den Sinus von 1 (Bogenmaß).

```
1 sin
```

Programm 3: Wie viele 10 stellige Dualzahlen gibt es?

```
2 hoch 10
```

Programm 4: Berechne die Kantenlänge eines Würfels des Volumens 2 (die dritte Wurzel aus 2).

```
2 hoch (1:3)
```

oder

```
2 hoch 1/3
```

1/3 ist eine rationale Zahl. D.h., „/“ ist keine Operation im Gegensatz zu „:“. Da wir stets von links nach rechts rechnen, ergibt 1 hoch 1 : 3 = (1 hoch 1):3 = 0.333333333.

Programm 5: Bilde einen Durchschnitt.

```
1 3 2 1 3 4 ++:
```

Um Schreibaufwand zu sparen, kann man bei einer Liste von Werten (hier die Zahlen der Zeile) auf jegliches sichtbare Trennzeichen und jegliche Klammerung verzichten. Auf diese Liste wird jetzt die Operation der Durchschnittsbildung (++:) angewandt. Man kann anstelle des Durchschnittes auch eine Summe (`++`), das Produkt (`**`), die Anzahl (`++1`), das Maximum (`max`), den Sinus (`sin`), ln, … bilden. Da `sin` nur einen Inputwert benötigt, ergibt sich bei Anwendung der Sinusoperation anstelle der ++: -Operation eine Liste von output-Werten. Auf diese Liste könnte man dann wieder ++: anwenden und erhält dann lediglich eine Zahl. Obige Schreibweise ist zunächst vielleicht etwas gewöhnungsbedürftig, aber sie ist kompakter als die alte Schreibweise

```
++:([1 3 2 1 3 4])
```

Insbesondere, wenn mehrere Operationen hintereinander angewandt werden, spart man sich hierbei viele Klammern und damit Fehlerursachen. Wir betrachten `++:` und die anderen hier genannten Operationen als einstellig (unär), genau wie `sin` oder `sqrt` und schreiben sie nach dem Inputwert, da wir die gegebenen Zahlen als **eine** Liste (**ein** Tabment) ansehen.

Programm 6: Berechne den Wert des Terms „`sqrt(abs(sin(7.1))+abs(cos(8.1)))`“.

```
7.1 sin abs
+ 8.1 cos abs
sqrt
```

oder einzeilig mit einem Klammerpaar

```
7.1 sin abs + (8.1 cos abs) sqrt
```

In der dreizeiligen Lösung wird, wie in Programmiersprachen üblich, von oben nach unten gerechnet. Den Wert der zweiten Zeile kann man nicht berechnen, wenn man nicht das „+“-Zeichen weglässt. Daher werden einfach die Werte der ersten und zweiten Zeile addiert. Aus dem Gesamtergebnis der zweiten Zeile wird dann durch die dritte Zeile die Wurzel gezogen.

Programm 7: Ziehe von der ersten Zahl 5 weitere Zahlen ab.

```
500 77.9 45.6 33.5 23.74 25.88 --
```

In o++o werden mehrere Operationen nach dem sehr alten „WaldPrinzip“ geschrieben. Für den Baum gibt es ein Wort und BaumBaum heißt Wald. Obiges `- -` ersetzt daher 5 Minuszeichen

Programm 8: Multipliziere jede Dezimalzahl einer Liste mit einer weiteren Zahl.

```
2.40 2.70 7.90 * 1.19
```

Hierbei wird jede Zahl der Inputliste mit 1.19 multipliziert. Sind die gegebenen Zahlen Nettopreise, so stellt das Ergebnis die zugehörigen Bruttopreise dar; sind die gegebenen Zahlen dagegen Beträge einer Währung und ist 1.19 der Umrechnungskurs, so stellt das Ergebnis die Werte in der anderen Währung dar, sind die Zahlen der Liste Längen von Rechtecken, so ergeben sich 3 Flächen von rechtecken, … . Wir sehen, dass Dezimalzahlen nicht mit dem Komma sondern mit dem Punkt dargestellt werden.

Programm 9: Bilde die Summe von vielen positiven und vielen negativen Zahlen ohne viele Minuszeichen und Klammern zu benutzen.

```
4 5 3 2 1  8 9 ++
- 7 6 5 4 3 2 1 ++
```

Programm 10: Berechne die Umfänge mehrerer Kreise, wobei die Radien gegeben sind.

```
4 5 6 2 3.7 9.77
*pi*2
```

Programm 11: Berechne die Umfänge und Flächen mehrerer Kreise, wobei die Radien gegeben sind.

```
[R! 4 5 6 2 3.7 9.77]
UMFANG:=R*pi*2
FLAECHE:=R*R*pi
```

Durch das R bekommt jedes Element der gegebenen Liste den Namen (tag) R. Durch eine Zuweisung (:=) wird die gegebene Tabelle um eine neue Spalte erweitert. Oben kommen nacheinander die Spalten UMFANG und FLAECHE hinzu, so dass sich eine Tabelle vom Typ R,UMFANG,FLAECHE l ergibt. l steht für Liste.

Programm 12: Berechne die Flächen und Umfänge mehrerer Rechtecke.

```
<TAB!
A,   B l
1.23 5.67
7.65 4.32
9.87 6.54
!TAB>
UMFANG:=A+B*2
FLAECHE:=A*B
```

Die TAB-Klammern (<TAB!, !TAB>) sind nur im Programmteil des Systems erforderlich. In der TAB-Darstellung müssen die Werte genau unter den zugehörigen Spaltennamen beginnen. Bei kleinen flachen Inputtabellen kann auch eine einzeilige Schreibweise benutzt werden.

```
[A,B! 1.23 5.67 7.65 4.32 9.87 6.54]
UMFANG:=A+B*2
FLAECHE:=A*B
```

Es entsteht eine Tabelle mit 4 Spalten.

Programm 13: Bestimme den Gesamtpreis einer einfachen Rechnung.

```
<TAB!
ARTIKEL,  PREIS l
Bier      0.61
Brause    0.23
Schnitzel 2.40
!TAB>
```

```
++
```
Hier wird einfach die Summe über alle Zahlen der gegebenen Tabelle (Liste (l) von Paaren) gebildet. Die Artikelwerte sind Wörter und haben damit keinen Einfluss auf das Ergebnis. Ersetzt man ++ durch
```
*1.10
```
, so entsteht wieder eine Tabelle mit 2 Spalten und drei Zeilen (Sätzen,Tupeln), wobei jede Zahl das Trinkgeld einschließt. Anschließend kann man wieder
```
++
```
anfügen, um auf die Gesamtsumme zu kommen.

Programm 14: Bestimme den Gesamtpreis einer etwas komplizierteren Rechnung, die durch eine einfache Tabelle gegeben ist.
```
<TAB!
ARTIKEL,  PREIS, ANZAHL l
Bier      0.61   7
Brause    0.23   3
Schnitzel 2.40   4
!TAB>
POSPREIS:=PREIS*ANZAHL
++ POSPREIS
```
Durch die Zuweisung wird die gegebene Tabelle um eine neue Spalte mit dem Spaltennamen POSPREIS erweitert, wobei der Preis dreimal mit der zugehörigen Anzahl multipliziert wird. Um die Gesamtsumme des Kneipenbesuchs zu ermitteln, muss man der ++-Operation noch einen zweiten Inputwert geben. Ansonsten würde die Gesamtsumme aller 9 Zahlen der Zwischentabelle gebildet werden. Beide Programmzeilen können auch einfach durch
```
++ PREIS*ANZAHL
```
ersetzt werden. Der erste Inputwert einer Operation, die am Anfang einer Programmzeile steht, ist immer das Ergebnis der vorangehenden Programmzeile.
Das Ergibtzeichen `:=` der Zuweisung ist vom Gleichheitszeichen `=` zu unterscheiden. Das Gleichheitszeichen wie auch `<, >, <=`, in, … wird zur Formulierung von Bedingungen benötigt. Bedingungen dienen der Selektion. Fügt man beispielsweise eine Bedingung
```
avec ARTIKEL=Bier
```
oder einfach
```
avec Bier
```
ein, so ergibt sich im Endergebnis nur der Gesamtpreis für die 7 Bier. Will man den Preis für den Rest berechnen, so kann man stattdessen
```
sans ARTIKEL=Bier
```
oder einfach
```
sans Bier
```
einfügen. Spaltennamen (Metadaten) müssen stets groß geschrieben werden. Die Schlüsselworte (gib, sans, avec, …) müssen stets klein geschrieben werden. Schreibt man ein Wort der Primärdaten immer mit Groß- **und** Kleinbuchstaben, so wird das Programm leichter lesbar.

Programm 15: Bestimme den Gesamtpreis eines längeren Kneipenbesuchs.
```
<TAB!
ARTIKEL,  PREIS,ANZAHLl l
Bier      0.61  7 6 5
                3
Brause    0.23  3 4
Schnitzel 2.40  4 3 2
```

```
!TAB>
POSPREIS:=PREIS*ANZAHL
++ POSPREIS
```
Hierbei haben wir es mit einer strukturierten Tabelle zu tun, die für jede Position mehrere Bestellungen enthält. Wir müssen die ANZAHL-Einträge nicht untereinander schreiben, so dass man den Kneipenbesuch kompakt darlegen kann. Betrachtet man dagegen nicht das Endergebnis sondern nur das Ergebnis der Anwendung der Zuweisung, so müssen die ANZAHL- und POSPREIS-Werte untereinander stehen, da die entstehende strukturierte Tabelle ansonsten zu unübersichtlich werden würde:
```
ARTIKEL,  PREIS, (ANZAHL, POSPREIS l)l
Bier      0.61    7       4.27
                  6       3.66
                  5       3.05
                  3       1.83
Brause    0.23    3       0.69
                  4       0.92
Schnitzel 2.4     4       9.6
                  3       7.2
                  2       4.8
```
Will man erst die ANZAHL-Werte addieren und dann multiplizieren,
```
POSPREIS:= ANZAHLl ++ *PREIS
```
, so ergibt sich wieder eine kompaktere Zwischentabelle:
```
ARTIKEL,  PREIS, POSPREIS,ANZAHLl m
Bier      0.61   12.81    7 6 5 3
Brause    0.23    1.61    3 4
Schnitzel 2.4    21.6     4 3 2
```
Beide Programmzeilen können auch wieder durch ++ PREIS*ANZAHL ersetzt werden.
Man kann die betrachteten Tabellen jeweils unter einem Namen beispielsweise mit der Endung .tab abspeichern und dann diese wieder aufrufen. Dadurch können Tabellen oder Dokumente in mehreren Programmen genutzt werden.

Programm 15 könnte dann so aussehen:
```
kneipenbesuch.tab
++ PREIS*ANZAHL
```

Dieses Beispiel lässt sich auf viele Anwendungen übertragen. Beim Kegeln könnte man ebenfalls eine Tabelle kegeln.tab mit Wiederholgruppe aufbauen: NAME,WURFl m. m kürzt Menge und l Liste ab. Für jeden Kegler kann man die Gesamtsumme dann durch eine Zuweisung ermitteln und anschließend die Daten abfallend sortieren. Will man im Ergebnis nur noch bestimmte Spalten und will man nach diesen noch sortieren, so benötigt man eine gib-Klausel.

Programm 16: Bestimme für jede Person das Gesamtergebnis des Kegelns und sortiere die Daten danach.
```
kegeln.tab
GESAMT:=WURFl ++
gib GESAMT,NAME m-
```
Es geht auch noch etwas kürzer:

Programm 17: Siehe Programm 16.
```
kegeln.tab
```

```
gib GESAMT,NAME m-
    GESAMT:= WURF ! ++
```
Hierbei ergibt sich der Bezug der Aggregation (hier ++) aus dem Kopf der gewünschten Tabelle. GESAMT ist eine Aggregation pro NAME. Bei Mengen (m, m-) wird stets nach den zuerst angegebenen Spaltennamen sortiert.
Wir nehmen jetzt an, dass jeder Fluss in `fluesse.tab` eine `LAENGE`-Spalte hat. Dann findet man alle Flüsse, die länger als 500 km sind durch folgende Anfrage:

Programm 18: Selektiere in einer vorgegebenen Tabelle.
```
fluesse.tab
avec LAENGE > 500
```
„avec“ ist französisch und heißt „mit“. Analog benutzen wir für ohne „sans“.
Jetzt sollen die Flüsse noch nach der Länge sortiert werden und mit Nebenflüssen ausgegeben werden:

Programm 19: Selektion mit anschließender Sortierung.
```
aus  fluesse.tab
avec LAENGE>500
gib  LAENGE,FLUSS,NEBENFLUSSm m
```
Will man die längsten Flüsse zuerst, muss man lediglich, das äußere `m` durch `m-` ersetzen.
Unsere Datei fluesse.tab enthält für jeden Fluss auch eine Wiederholgruppe
`LAND,BUNDESLANDm m`, die angibt durch welche Länder und Bundesländer der Fluss fließt. In fluesse.tab ist also der FLUSS dem Bundesland übergeordnet. Will man das umkehren, kann das wieder einfach durch Angabe des Kopfes der gewünschten Tabelle realisiert werden.

Programm 20: Umstrukturierung der Fluesse-Datei.
```
aus fluesse.tab
gib BUNDESLAND,FLUSSm m
```
Durch diese freie Umstrukturierung werden zu jedem Bundesland alle Flüsse, die durch dieses fließen, gesammelt. Die Bundesländer und die Flüsse innerhalb der Bundesländer werden sortiert. Ein Fluss kann hier in mehreren Bundesländern vorkommen. Jedes Bundesland erscheint nur einmal, da wir als äußere Kollektion eine Menge gewählt haben. Den Beginn eines Programms kann man auch mit `aus` kennzeichnen.
Neben den einfachen Berechnungen (:=), durch die eine Tabelle um eine neue Spalte erweitert wird, gibt es noch die sogenannten rekursiven Zuweisungen. Hierbei sind zwei Formeln gegeben. Die erste Formel ist für das erste Element der Spalte bestimmt und die zweite für die restlichen Elemente. Die Formel für die restlichen Elemente kann sich auf den Vorgänger der Spalte beziehen. Das ist im folgenden Programm `BETRAG pred`.

Programm 21: Was wird aus 100 Euro nach 20 Jahren bei 9 % Wachstum (Zinsen).
```
[BETRAG! 0 .. 20]
BETRAG:= first 100. next BETRAG pred*1.09 at JAHR
```

Programm 22: Wandle eine Dualzahl (hier 10111=23) in eine Dezimalzahl um.
```
[BIT! 1 0 1 1 1]
DEZI:= first BIT next DEZI pred*2+BIT at BIT
```
Ist nicht die ganze Entwicklung sondern nur das letzte Element der Liste gewünscht, so kann man noch die Bedingung BIT `pos- =1` anhängen. Ist das letzte Bit auch nicht gewünscht, so sollte `gib DEZI` folgen.

Programm 23. Wandle eine Dezimalzahl (hier 23) in eine Dualzahl um.

```
DURCH,REST l := first 23 divrest 2
                next  DURCH pred divrest 2
                while DURCH,REST != 0,0
gib RESTl-
```

Hierbei werden mit einer rekursiven Erweiterung gleich 2 neue Spalten eingeführt, DURCH und REST. `divrest` ist hier die ganzzahlige Division mit dem ganzzahligen Rest (ein Paar von Zahlen).

Programm 24: Berechne den gewichteten Durchschnitt für 3 Schüler und den Gesamtdurchschnitt.

```
<TAB!
NAME,  KLAl, NOTEl l
Ernst  1 2   1 2 3 1 3 1 1
Clara  1 1   3
Sophia 1 3   1
!TAB>
DUR:=KLAl ++: *0.6 +(NOTEl ++: *0.4)
GESAMT:=DURl ++:
rnd 2
```

Durch `rnd` wird das Ergebnis gerundet.

Programm 25: Bilde eine Vereinigung (alle StudentenIDs, die in einer der Tabellen enthalten sind).

```
aus examen.tab, projekte.tab
gib STIDm
```

Hierbei seien examen.tab und projekte.tab vom Typ
`STID,KURS,NOTE m` bzw. `STID,PROJ,STUNDEN m`.

Programm 26: Bilde einen Durchschnitt (alle StudentenIDs, die in beiden Tabellen enthalten sind).

```
aus examen.tab, projekte.tab
igib STIDm
```

Mit igib können auch „joins“ ausgedrückt werden und damit auch Anfragen an ganze Datenbanken formuliert werden. Die folgenden Anfragen beziehen sich auf eine Datenbank `uni.tab`, deren zweite Tabelle unstrukturiert ist. Die Anfragen müssen nicht oder nur schwach modifiziert werden, wenn alle Tabellen von uni.tab unstrukturiert (relational) sind.

```
<STUDENTEN!
STID, NAME,  ORT?,    STIP, FAK,  (KURS,       NOTE m),(PROJ, STUNDEN m) m
1234  Ernst  Oehna    500   Mathe  Algebra     1        Fritz  4
                                  Geschichte  1        Otto   2
                                  Logik       2
1245  Sophia Berlin   400   Inf    Algebra     3        Mia    5
                                  Datenbanken 1        Ming   4
                                  Otto        1        Otto   6
3456  Clara  Oehna    450   Inf    Datenbanken 1
                                  OCaml       2
4567  Ulrike          400   Kunst                       Monet 10
5678  Kaethe Gerwisch   0   Kunst  Apel        1        Monet 20
                                  Repin       1
!STUDENTEN>
<FAKS!
FAK,  DEKAN,  BUDGET, STUDKAP m
Inf   Reichel 10000   500
Kunst Sitte    2000   600
```

```
Mathe Dassow   1000   200
Philo Hegel    1000    10
!FAKS>
```
Die Tabelle `STUDENTEN` ist strukturiert, da zu jedem Satz (strukturiertem Tupel) (Studenten) 7 Komponenten gespeichert werden und die letzten beiden Komponenten selbst wieder kleine Tabellen sind. Die vorletzte Komponente von Clara enthält beispielsweise zwei Prüfungsergebnisse und die letzte die leere Menge von Projekten.

Programm 27: Berechne das Gesamtbudget der Uni.
```
aus uni.tab
++  BUDGET
```

Programm 28: Berechne das Gesamtbudget und das Durchschnittsbudget der Uni.
```
aus uni.tab
gib SUMBUD,DURBUD  SUMBUD:= BUDGET! ++
                   DURBUD:= BUDGET! ++:
```
Da die letzte Zeile mit mehr als 3 Leerzeichen beginnt, gehört sie vom logischen Standpunkt noch zur vorangehenden Zeile.

Programm 29: Wie viele Fakultäten hat die Uni.
```
uni.tab
gib FAKS
++1
```
Lässt man `gib FAKS` weg, so erhält man die Anzahl der Studenten und die Anzahl der Fakultäten.

Programm 30: Gib zu jedem Kurs die zugehörigen Studenten mit Note.
```
aus uni.tab
gib KURS,(NAME,NOTE b)m
```

Programm 31: Gib alle Sätze (Tupel), die 1234 enthalten.
```
aus  uni.tab
avec 1234
```

Programm 32: Selektiere in allen Tabellen, die den Studentenidentifikator enthalten, nach dem Studenten mit der Nummer 1234.
```
uni.tab
avec STID=1234
```
Programm 32 unterscheidet sich nur bzgl. der FAKS-Tabelle von Programm 31. Im Ergebnis von Programm 31 ist diese leer und in Programm 32 bleibt sie vollständig erhalten. In anderen Anwendungen könnte man statt des Studentenidentifikators (STID) auch Artikelnummern oder Teilenummern, … wählen.

Programm 33: Gib zum Studenten mit der Nummer 1234 den Dekan und seine Examen.
```
aus  uni.tab
avec STID=1234
igib NAME,DEKAN,(KURS,NOTE m)m
```
Hier wird ein „Join“ ohne Joinbedingung realisiert. Würde man anstelle von i`gib gib` verwenden, so würde die Ergebnismenge leer bleiben, da `gib` keine Verbindung zwischen NAME und DEKAN über FAK herstellt.

Programm 34: Berechne die Anzahl der Einsen für die einzelnen Fakultäten.

```
aus  uni.tab
avec NOTE=1
gib  FAK,ANZ m   ANZ:= NOTE! ++1
```

Programm 35: Teste das folgende Programm.

```
aus  uni.tab
avec DEKAN in [Reichel Dassow]
gib  FAK,DEKAN,(NAME,STIP m)m
```

Programm 36: Gib zu den Fakultäten von Dassow und Reichel alle zugehörigen Studenten.

```
aus  uni.tab
avec DEKAN in [Reichel Dassow]
igib FAK,DEKAN,(NAME,STIP m)m
```

Programm 37: Gesucht sind alle Fakultäten, die Studenten mit mindestens 2 Einsen und einer Drei haben.

```
aus  uni.tab
avec {{1 1 3}} inm NOTEb # inm: in mathematisch
igib FAK,DEKAN m
```

„b“ steht für bag (Multimenge).

Programm 38: Gesucht sind alle Studenten, die genau 2 Einsen haben.

```
aus  uni.tab
avec NOTEl = [1 1]
gib  STUDENTEN
```

Programm 39: Gesucht sind alle Daten von Ernst (einschließlich seiner Fakultätsdaten).

```
aus  uni.tab
avec NAME=Ernst
igib
```

Das Programm muss nicht verändert werden, wenn alle Tabellen von uni.tab in erster Normalform (unstrukturiert) sind.

Programm 40: Eine Flasche mit Kork kostet eine Mark und zehn. Die Flasche ist eine Mark teurer als der Korken. Wie viel kostet der Korken?

```
[FLASCHE! 0 .. 110]
KORK := FLASCHE - 100
avec KORK+FLASCHE = 110
```

Programm 41: Schreibe 1000 mal ich liebe dich:

```
1000 mal “Ich liebe Dich!“
```

„mal“ ist eine binäre Operation, die wie alle anderen binären Operationen infix (zwischen die Inputwerte) geschrieben werden.

Programm 42: Gesucht sind alle Baugruppen und Einzelteile des i40 mit Alufelgen ohne Klimaanlage.

```
<TAB!
OT,      EIGENSCHAFT,  (UT,        ANZ l) m
```

```
Buchse   zylindrisch
Felge                  Alufelge   1
                       Stahlfelge 1
i30      modern        Rad        4
                       Motor      1
                       Karosse    1
i40      schnell       Rad        4
                       Reserverad 1
                       Karosse    1
                       Klimaanl   1
                       Motor      1
Karosse  blau
Klimaanl robust
KolRring rund
Kolben   leicht        KolRring   2
                       Buchse     1
Motor    schwer        Kolben     6
                       Schraube   8
Rad      rund          Schraube   5
                       Reifen     1
                       Felge      1
Reifen   schwarz
Schraube stabil
!TAB>
avec UT! OT=Felge -> UT=Alufelge # MUSS Bedingung
sans UT! OT=i40 & UT = Klimaanl # KANN Bedingung
wege i40
HILFSANZ:=first ANZ next HILFSANZ pred*ANZ at ANZ
avec UT! UT pos- =1
gib  (UT,TOTAL m) TOTAL:= HILFSANZ ! ++
```

Ergebnis:

```
UT,        TOTAL  m
Alufelge    4
Buchse      6
Felge       4
Karosse     1
Kolben      6
KolRring   12
Motor       1
Rad         4
Reifen      4
Reserverad  1
Schraube   28
```

Programm 43: Bestimme den Gesamtpreis eines längeren Kneipenbesuchs, wobei eine separate Preistabelle gegeben ist.

```
<TAB!
ARTIKEL,  NRl m
Bier      2 4 5
```

```
Brause    3 2
Schnitzel 4 3
!TAB>
, artikels.tab
igib TOT,(ARTIKEL,TOT m)  TOT:= NR*PREIS ! ++
```
Ergebnis:
```
TOT0, (ARTIKEL,  TOT1  m)
79.3   Bier      29.7
       Brause    16.
       Schnitzel 33.6
```

artikels.tab:
```
ARTIKEL,  PREIS  m
Bier      2.7
Brause    3.2
Schnitzel 4.8
Steak     4.8
```

Programm 44: Gesucht ist der Dekan von Ernst. (Anfrage an eine relationale Datenbank.)
```
unirelational.tab
avec NAME=Ernst
igib DEKAN
```
unirelational besitzt hierbei das folgende Schema:
```
TABMENT! UNIRELATIONAL
UNIRELATIONAL! STUDENTEN,EXAMEN,PROJEKTE,FAKS
PROJEKTE! (STID,PROJ,STUNDEN m)
STUDENTEN! (STID,NAME,ORT?,STIP,FAK m)
EXAMEN! (STID,KURS,NOTE m)
FAKS! (FAK,DEKAN,BUDGET,STUDKAP m)
```

Programm 45: Gib mir alle Daten von Ernst.
```
unirelational.tab
avec NAME=Ernst
igib
```
Ergebnis als hsq-Ausgabe:
```
STID NAME ORT STIP FAK
 STID KURS NOTE
  STID PROJ STUNDEN
   FAK DEKAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 1234 Algebra 1
 1234 Geschichte 1
 1234 Logik 2
  1234 Fritz 4
  1234 Otto 2
   Mathe Dassow 1000 200
```
Es sei angemerkt, dass alle Daten von Ernst im Ergebnis erscheinen, ohne dass ein Kreuzprodukt benutzt wird. Ansonsten würden die Projekte „Fritz“ und „Otto“ jeweils 3 mal erscheinen.

Programm 46: Gesucht sind die Prüfungsergebnisse vom Studenten mit der Nummer 1234.

```
unirelational.tab
avec STID=1234
igib NAME,DEKAN,(KURS,NOTE m)m
```
Ergebnis:
```
NAME, DEKAN, (KURS,      NOTE  m) m
Ernst Dassow  Algebra    1
              Geschichte 1
              Logik      2
```
Es sei angemerkt, dass das Ergebnis Informationen aus 3 Tabellen enthält, ohne dass eine Joinbedingung genutzt wurde.

o++o für die Schule

A. Merkel „Jeder Schüler soll neben Lesen, Rechnen und Schreiben auch Programmieren lernen."
o++o (ausführlich ottoPS) ist eine tabellenorientierte Programmiersprache, die auf Schleifen verzichtet. Dennoch ist o++o sehr ausdrucksstark und man kann mit ihr nicht nur kompakte Anfragen sondern auch vielfältige Berechnungen für strukturierte Tabellen und strukturierte Dokumente bewerkstelligen.
o++o benutzt und vermittelt viele mathematische Konzepte, daher sehen wir die Hauptvorteile der Vermittlung im Mathematikunterricht, genau wie die wesentlichen Fähigkeiten für die Nutzung des Taschenrechners in Mathematik vermittelt werden. o++o verwendet insbesondere folgende Konzepte: Menge, Multimenge, Liste, Gleichheit und Inklusionsbeziehungen dieser; Tupel; leistungsfähige Operationen zum Selektieren; Berechnen; Restrukturieren, Sortieren und Aggregieren (Summe; Durchschnitt; …),… .
Tabellenkalkulationsprogramme wie EXCEL und die Datenbankstandardabfragesprache SQL kennen keine strukturierten Schemen und Tabellen. Erste Tests mit Vorschulkindern lassen vermuten, dass man mit strukturierten Tabellen leichter rechnen kann als mit Dezimalzahlen. Wir wollen weitere o++o-Beispielprogramme anfügen:

47. Berechne den Wert eines einfachen Terms.
```
2*3+4
```
* und + haben jeweils 2 Inputwerte. Zunächst wird 2*3 (6) berechnet. Die 6 ist erster Inputwert von +, so dass sich insgesamt 24 ergibt. Hier wird also einfach von links nach rechts gerechnet.

48. Schreibe den Term $\cos^3(\sin^2(3.14159))$ in o++o.
```
pi sin hoch 2 cos hoch 3
```
Unserer Meinung nach ist der Ausgangsterm für Otto Normalverbraucher schwer zu lesen. Man beginnt mit pi geht nach links bis zum `sin` dann nach rechts zum `hoch 2` jetzt bewegt man sich wieder nach links zum `cos` und abschließend nach rechts zum `hoch 3`. Diese Schreibweise wurde sicher eingeführt um Klammern zu sparen. Eigentlich müsste der Ausgangsterm um unmissverständlich zu sein, folgendes Aussehen haben:
$(\cos((\sin(3.14159))^2))^3$
Das ist sicher noch schwerer zu lesen und man bewegt sich noch mehr von links nach rechts und umgekehrt.

49. Schreibe den Term $\sin^2(x)+\cos^3(y)$ in o++o.
```
X sin hoch 2 + (Y cos hoch 3)
```
oder
```
X sin hoch 2
```

```
+ Y cos hoch 3
```
Man könnte alle Terme in o++o ohne Klammern schreiben, allerdings müssten dann bestimmte Terme mehrzeilig geschrieben werden.

50. Wie berechnet man den Term 2+3:4*5 ?
```
2+(3:(4*5))=2 3/20
2+((3:4)*5)=5 ¾
o++o: ((2+3):4)*5=6 1/4
```
Man erkennt, dass man mit der Schulweisheit Punktrechnung geht vor Strichrechnung noch nicht auskommt. Man benötigt die Regel „von links nach rechts“ zusätzlich.

51. Berechne den Durchschnitt mehrerer Noten.
```
1 2 3 1 2 ++:
```
Vom methodischen Standpunkt kann man dieses Programm noch verbessern, indem man die Klammern für Listen hinzufügt: `[1 2 3 1 2] ++:`
Man erkennt jetzt, dass die Durchschnittsoperation `++:` einen Inputwert, nämlich eine Liste besitzt und dass `++:` diesem einen Inputwert nachgestellt wird. Da die Nutzer in der Regel nicht viel tippen wollen, gehen wir davon aus, dass die erste Notation in Praxis häufiger benutzt werden wird.

52. Berechne die Durchschnitte der strukturierten Tabelle noten.tab für jedes Fach.
```
noten.tab
DUR:= NOTEl ++:
```
noten.tab könnte so aussehen:
```
FACH,NOTEl l
Ma   1 2 1 3 1 2
Phy  4 3 2 2 1
```
Hierbei kürzt l Liste ab. D.h., noten.tab ist eine einfache strukturierte Tabelle (Liste), die zu jedem Fach eine Liste von Noten enthält. Um Platz zu sparen, wählen wir auch hier die methodisch nicht optimale Darstellung. Wie FACH ist auch NOTE ein Spaltenname, so dass noten.tab eigentlich so dargestellt werden müsste:
```
FACH,NOTEl l
Ma   1
     2
     1
     3
     1
     2
Phy  4
     3
     2
     2
     1
```
Das Ergebnis der Anfrage wieder im „tab-Format“:
```
FACH, DUR,           NOTEl  l
Ma    1.66666666667 1 2 1 3 1 2
Phy   2.4           4 3 2 2 1
```

53. Bilde die Summe der Zahlen von 1 bis 100 (Aufgabe von Gauß Klasse 5).
```
1 .. 100 ++
```

Wie die Addition und die Multiplikation besitzt „..“ zwei Inputwerte (1 und 100). Als Zwischenergebnis entsteht die Liste

```
ZAHLl
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

,deren Zahlen dann aufsummiert werden, sodass sich `5050` ergibt.

54. Berechne näherungsweise das Maximum der Sinus-Funktion im Intervall [1 2].

```
1 ... 2!0.001 sin max
```

„…“ benötigt 3 Inputwerte: 1. den Anfangswert 1, den Endwert 2 und die Schrittweite 0.001. Es entstehen hierbei die Zahlen 1 1.001 1.002 1.003 … 1.999 2.

Auf jede der Zahlen wird die Sinusfunktion angewandt, sodass wieder 1001 Zahlen entstehen. Auf diese Liste wird dann die Funktion max (Maximum) angewandt. Obwohl es sich hierbei um ein Näherungsverfahren handelt, kommt der exakte Wert 1 heraus, wenn die Schrittweite weiter verfeinert wird. `sin` und `max` haben jeweils einen Inputwert (hier eine Liste) aber der Outputwert von `sin` ist wieder eine Liste und `max` erzeugt lediglich eine Zahl, da es sich hier um eine Aggregationsfunktion handelt. Der zweite und der dritte Inputwert einer dreistelligen Operation (oben …) wird jeweils durch ein „!“ getrennt. Das ist in o++o nötig, da das Komma für die Paarbildung bereits vergeben ist und das Leerzeichen bereits Listenelemente trennt.

55. Berechne näherungsweise das Minimum des Polynoms $X^3 + 4 X^2 -3 X+2$ im Intervall [0 2] mit zugehörigem X-Wert.

```
[X! 0 ... 2!0.001]
Y:= X polynom [1 4 -3 2]
MINI:= Yl min
avec Y = MINI
```

avec ist französisch und bezeichnet eine Selektion. Ein konkretes Polynom von einer Variablen X hat stets nur einen Inputwert, der für X eingesetzt wird. `polynom` in Zeile 2 ist dagegen allgemeiner und hat 2 Inputwerte:

1. Den Inputwert für X, der hier alle Zahlen, die in der ersten Zeile generiert wurden, annimmt
2. Eine Liste von Zahlen, die den Koeffizienten des konkreten Polynoms entspricht.

Durch die ersten Zeile entsteht eine Liste von Zahlen, die alle den Namen X bekommen haben. Das erkennt man am besten in der xml bzw. ment-Repräsentation:

<X>0.</X>
<X>0.001</X>
<X>0.002</X>
…

Gesamtergebnis:

```
MINI,       (X,     Y         l)
1.481482037 0.333 1.481482037
```

56. Berechne eine Nullstelle der Cosinus Funktion im Intervall [1 2] näherungsweise.

```
[X! 1 ... 2!0.0001]
avec X cos < 0
avec X pos = 1
```

Hier verbleiben nach der ersten Selektion nur die X-Werte mit einem Funktionswert kleiner 0. Von diesen wird im zweiten Schritt der erste Wert ausgewählt. Da wir wissen, dass *cos* nur eine Nullstelle im betrachteten Intervall besitzt, wird diese durch das Ergebnis angenähert.

57. Berechne das Gesamtwachstum, wenn 5 Jahreswachstumszahlen gegeben sind. Runde das Ergebnis auf eine Stelle nach dem Komma.

```
[W! 0 1.5 2.1 1.3 0.4 1.2]
ACCU:= first 100. next ACCU pred *(W:100+1) at W
rnd 1
```

Die Ergebnistabelle:

```
W,  ACCU  l
0.  100.
1.5 101.5
2.1 103.6
1.3 105.
0.4 105.4
1.2 106.7
```

Der erste ACCU-Wert ergibt sich durch den Ausdruck hinter `first (100.).` Für den zweiten Wert wird für `ACCU pred` der Wert `100.` eingesetzt und der Term nach `next` bewertet. Es ergibt sich 101.5. Diese Zahl wird wieder in `ACCU pred` eingesetzt und der `next`-Term erneut berechnet (rund 103.6),… bis der letzte W-Wert erreicht ist. `pred` ist der predecessor (Vorgänger).

58. Berechne die Fläche unter der Sinuskurve im Intervall [0, pi] näherungsweise.

```
0 ... pi!0.0001 sin *0.0001 ++
```

Hierbei werden nacheinander alle Zahlen zwischen 0 und pi generiert, dann von jeder Zahl der Sinus berechnet und anschließend jede Zahl mit 0.0001 multipliziert. Es entstehen 10000 Rechteckflächen, die anschließend addiert werden.

59. Berechne den DurchschnittsBMI pro Alter und den BMI pro Person und Alter für alle Personen über 20.

```
<TAB!
NAME,       LAENGE,   (ALTER, GEWICHT l) l
Klaus       1.68       18     61
                       30     65
                       56     80
Rolf        1.78       40     72
Kathi       1.70       18     55
                       40     70
Walleri     1.00        3     16
Viktoria    1.61       13     51
Bert        1.72       18     66
                       30     70
!TAB>
avec NAME! 20<ALTER
BMI:= GEWICHT : LAENGE : LAENGE
gib  ALTER,BMIAVG,(NAME,BMI m) m  BMIAVG:= BMI ! ++:
rnd 2
```

Die TAB-Klammern deuten an, dass die eingeschlossenen Daten der TAB-Darstellung entsprechen. Die obige Bedingung selektiert Personen-Sätze, d.h. `NAME,LAENGE,(ALTER,GEWICHT l)` Tupel (strukturierte Tupel bzw. Strupel). Da eine Personen mehrere ALTERs-Angaben besitzt, muss quantifiziert werden. `NAME! 20 <ALTER` selektiert demnach alle Personen, die einen entsprechenden Alterseintrag besitzen. D.h., der Existenzquantor wird nicht geschrieben, gehört

aber zu jeder Bedingung. In diesem kleinen Beispiel könnte man die Selektion natürlich auch per Hand realisieren.
Resultat:

```
ALTER, BMIAVG, (NAME, BMI  m) m
18     20.98    Bert  22.31
                Kathi 19.03
                Klaus 21.61
30     23.35    Bert  23.66
                Klaus 23.03
40     23.47    Kathi 24.22
                Rolf  22.72
56     28.34    Klaus 28.34
```

Das Ergebnis kann beispielsweise durch einfaches Klicken als Säulendiagramm dargestellt werden. Das Beispiel zeigt, das man eine Hierarchie einfach durch Angabe des gewünschten Schemas umkehren kann. Im Ergebnis ist der Name dem Alter untergeordnet.

Es wird insbesondere deutlich, dass die Aufgaben ohne Kenntnisse der Differential- und Integralrechnung gelöst werden können. Mit o++o kann der Mathematikunterricht in vielfältiger Weise unterstützt werden. Das reicht von Klasse 7 oder tiefer bis zur Klassenstufe 12. Es betrifft: Rechnen mit natürlichen Zahlen, Dezimalzahlen, näherungsweise Berechnung von Nullstellen beliebiger Funktionen, Ableitung, Flächen unter Kurven, Extremwerte (kann bereits in der Sekundarschule gelehrt werden), Wahrscheinlichkeitsrechnung, … . Mit o++o können Dinge in einfacher Weise berechnet werden, die sonst nur theoretisch abgehandelt werden. Dadurch kann das Verständnis der Konzepte wesentlich verbessert, erweitert und vertieft werden. Weitere Informationen zu o++o finden Sie unter **ottops.de**.
Wir glauben, dass o++o besondere Vorteile für den Mathematik- und Informatikunterricht bietet aber auch in den anderen Fächern (zukünftig Anfragen an die Wikipedia) sinnvoll genutzt werden kann.

Was ist ein Schema einer strukturierten Tabelle?

Zunächst ist eine Tabelle in n-Spalten A1,A2,…,An gegliedert. Die Spaltennamen im o++o-Programm dürfen keine Kleinbuchstaben enthalten. Haben A1,A2, .. An einen elementaren Typ, so enthält eine A1,A2, …, An-Tabelle genau eine einfache Zeile, z.B.:

```
NAME,ORT,  GEHALT
Paul Oehna 1000
```

Wenn wir ein Kollektionssymbol z.B.. l für Liste anhängen, kann die Tabelle 0,1, oder mehr Zeilen enthalten. Beispiel:

```
NAME,  ORT,    GEHALT l
Paul   Oehna   1000
Sophia Dallgow  900
Emily  Dallgow 2000
Clara  Oehna    900
```

Ersetzen wir das l durch das Mengensymbol m, so wird die Tabelle sortiert dargestellt.

```
NAME,  ORT,    GEHALT m
Clara  Oehna    900
Emily  Dallgow 2000
Paul   Oehna   1000
Sophia Dallgow  900
```

Wenn diese Tabelle nach Ort sortiert werden soll, vertauscht man lediglich die Spalten in einer entsprechenden Anweisung `gib ORT,NAME,GEHALT m`

```
ORT,    NAME,  GEHALT  m
Dallgow Emily  2000
Dallgow Sophia  900
```

```
Oehna   Clara   900
Oehna   Paul   1000
```
Jetzt erkennt man, dass die Tabelle etwas Redundanz enthält. In einer strukturierten Tabelle kann man diese beseitigen:
```
ORT,    (NAME,  GEHALT  m) m
Dallgow Emily  2000
        Sophia  900
Oehna   Clara   900
        Paul   1000
```
Diese Tabelle enthält 2 Strupel (strukturierte Tupel), d.h., z.B., dass die Anzahl (++1) der Elemente der Tabelle nicht mehr 4 sondern 2 ist. Trotzdem können wir diese Tabelle mit o++o genauso verarbeiten wie die vorangehenden. Aus der Ausgangstabelle kann man mit einer gib-Anweisung auch 2 oder 3 Tabellen erzeugen:
```
ORTm,   NAMEm, GEHALTm
Dallgow Clara   900
Oehna   Emily  1000
        Paul   2000
        Sophia
```
Diese Gesamttabelle enthält zwar alle elementaren Daten der Ausgangstabelle, die weiteren Informationen sind jedoch verloren gegangen. Dallgow und Clara befinden sich in dieser TAB-Darstellung zwar in der gleichen Zeile, dennoch wohnt Clara nicht in Dallgow. Man muss sich das Schema daher genau ansehen.

Ein mensch dem zahl (o++o) verborgen ist
Leichtlich der verführt wird mit list
Das nimm zu hertzen bitt ich sehr
Und jeder sein kind rechnen (programmieren) lehr ...

Adam Ries