

OPL Workshop 3 - Spielend lernen: Würfel, Bandit & Lotto

Ulrich Krinzner, 11/1998

Im letzten Workshop haben wir uns den Programmiergrundlagen gewidmet. Sie haben etwas über Prozeduren, Variablen, Schleifen und Verzweigungen in Erfahrung bringen können. Anhand eines kommentierten Beispielprogrammes wurden die frisch vermittelten Kenntnisse vertieft.

In dieser Folge werden wir das Beispielprogramm, das sich den Zufallsgenerator zunutze macht, mit drei weiteren Unterprogrammen komplettieren. Das gibt Ihnen die Möglichkeit, weitere OPL-Befehle und -Funktionen kennenzulernen.

Rücken Sie also Ihren Psion zurecht, und öffnen Sie den OPL-Editor. Sie können sich etwas Schreibarbeit sparen, wenn Sie die Datei vom letzten Mal hernehmen und sie unter einem neuen Namen abspeichern, denn ein Teil der Module (synonym: Prozeduren, Unterprogramme, Programmteil, Programm-Modul) werden wir wiederverwenden

Unseren Werte-Generator (PROC zufall:) hatten wir ja gleich auf Zuwachs angelegt, er wird unverändert vom letzten Mal übernommen. Wir wollen ihn nun für weitere Programmteile nutzbar machen. So werden wir das Programm um die Komponenten "Würfel", "Slotmaschine" (oder auch als "Einarmiger Bandit" bekannt) und "Lottoziehung" erweitern. Diese werden (wie bereits der Mathe-Trainer PROC mathe:) in jeweils einer eigenen Prozedur untergebracht.

Die neuen Module sollen (wie zuvor der Mathe-Trainer auch) vom Hauptprogramm PROC main: aufgerufen werden. Wir werden es daher modifizieren müssen. Bevor wir uns jedoch an diese Aufgabe machen, spendieren wird statt fertiger Unterprogramme zuerst einmal nur die Rahmen der neuen Module. Schließlich soll der Aufruf dieser Module nicht ins Leere laufen und eine Fehlermeldung produzieren.

```
PROC wuerfel:
ENDP
PROC bandit:
ENDP
PROC lotto:
ENDP
```

Aufgerufen wird ...

Jetzt sorgen wir dafür, daß die einzelnen Programmteile gezielt aufgerufen werden können. Dazu erhält unsere gute alte PROC main: eine Schleife, in der ein Menü gezeigt wird, mit dem man einerseits die Auswahl treffen, andererseits das Programm sauber beenden kann.

```
PROC main:
  GLOBAL zahl%(7)
  LOCAL taste%
  FONT 268436068,1  REM nur Serie 5! Courier 15pt, fett
  RANDOMIZE MINUTE + SECOND
  DO
    CLS
    PRINT : PRINT
    PRINT "Wähle aus:"
    PRINT "1 - Mathe"
    PRINT "2 - Würfel"
    PRINT "3 - Bandit"
    PRINT "4 - Lotto"
    PRINT "0 - Ende"
    taste% = GET
    IF taste% = %1
      mathe:
    ELSEIF taste% = %2
      wuerfel:
    ELSEIF taste% = %3
      bandit:
    ELSEIF taste% = %4
      lotto:
    ENDIF
  UNTIL taste% = %0
ENDP
```

Wenn sie den Text abtippen und an die Spitze vor allen anderen Modulen anordnen, haben Sie bereits das lauffähige Kern-Programm vor sich. Nach dem Übersetzen und Starten wird das Menü dargestellt, Jedem Punkt ist eine Ziffer zugeordnet. Drückt man auf eine der Ziffern auf der Tastatur, wird der Tastendruck in der Variablen taste% abgelegt und in der folgenden IF/ELSEIF/ENDIF - Konstruktion ausgewertet. Als Folge der Auswertung wird zu dem jeweils angegebenen Modul verzweigt. Dieses wird abgearbeitet und gibt die Kontrolle an PROC main: zurück, wo der DO/UNTIL Kreislauf solange fortgesetzt wird, bis jemand auf die "Null"-Taste drückt. Dadurch wird die Bedingung "UNTIL taste% = %0" erfüllt und das Programm in den verdienten Ruhezustand geschickt.

Sie finden im Programmtext einige neue Dinge vor, über die es sich lohnt, ausführlicher zu sprechen - Sie werden Ihnen immer wieder begegnen.

Eingaben mit GET

Auch im Menü müssen wir wieder einmal dem Computer über die Tastatur mitteilen, was wir eigentlich von ihm wollen. Eine Methode dazu ist uns beim Mathe-Trainer über den Weg gelaufen: INPUT. Die hat jedoch neben ihren zugestanden Vorteilen einen gravierenden Nachteil: Nach Eingabeende muß jedesmal noch die <ENTER> betätigt werden. Bei PROC main: greifen wir daher lieber auf eine Lösung mit GET zurück. GET - bisher nur als "Programmstopper" kennengelernt - ist recht flexibel gerade für Eingaben von nur einem Zeichen zu gebrauchen. Dahinter verbirgt sich nämlich eine Funktion, die den Tastaturcode der gedrückten Taste zurückliefert, man muß nur danach fragen ... In der Praxis kann man den zurückgelieferten Wert einer Variablen zuordnen und so der Weiterverwendung zuführen (taste% = GET).

Um mit dem Tastaturcode etwas anfangen zu können, sollte man ihn schon kennen. Aber woher nehmen, ohne zu stehlen und das Handbuch nicht in der Nähe ist? Ganz einfach: Man ermittelt ihn selber.

Heran"tasten"

In Ihrer Programmierpraxis sollen Sie zu dieser Arbeitsweise ausdrücklich aufgefordert sein. Benutzen Sie einfache und übersichtliche Programmstückchen, um sich einen Überblick über die Arbeitsweise von Befehlen und Funktionen zu verschaffen. Sie müssen nicht immer "aus dem Ei gepellt" sein, sondern lediglich funktionieren. Wir führen Ihnen das hier am Beispiel von GET vor. So ganz nebenher erfahren Sie etwas über das Thema „Werteumwandlung“.

```
PROC get1:
  LOCAL taste%
  DO
    taste% = GET
    PRINT taste%, "=", CHR$(taste%)
  UNTIL taste% = %q REM "q"-Taste
ENDP
```

In einer DO/UNTIL - Schleife wird auf die Tastatureingabe gewartet. Nach jedem Tastendruck wird auf dem Bildschirm der Tastaturcode ausgegeben. "CHR\$(taste%)" bewirkt, daß auch das gedruckte Zeichen ("character") dargestellt wird, da diese Funktion eben die Umwandlung des Tastaturcodes (Integerzahlen) in einen einstelligen String ("Buchstabe") vornimmt. Der ließe sich auch einer Variablen zuweisen (z.B.: zeichen\$ = CHR\$(65), das ist ein "A"), was aber in PROC get1: nicht erforderlich ist, da nur dargestellt und nicht verarbeitet werden soll.

Wenn Sie etwas mit der Tastatureingabe experimentieren, stellen Sie fest, daß Werte unter 33 gar nicht auf dem Bildschirm ausgegeben werden. Ursache ist der ASCII-Zeichensatz, der neben Buchstaben, Zahlen usw. in dem reservierten Wertebereich 1 .. 32 sogenannte Steuerzeichen enthält. Probieren Sie einmal die Eingabe von <Ctrl><G> - ein Pieps ertönt, weil der Wert "7" das Steuerzeichen "BELL" darstellt. In der Praxis ließe sich das nutzen, indem man für einen kurzen Piepser schreibt:

```
PRINT CHR$(7)
```

(OPL stellt hier jedoch eine bessere Variante mit BEEP(dauer%, frequenz%) zur Verfügung ...)

Der %-Effekt

Nun muß man aber nicht in jedem Fall den Tastaturcode kennen, denn OPL hat uns ein prima Hilfsmittel in die Hand gegeben, das wir in PROC get1: auch gleich benutzt haben: eine vordefinierte Festwertzahl (Konstante), der Bezeichner festgeschrieben ist. Die Kombination von Prozentzeichen ("%"), gefolgt von einem Zeichen, das durch die Tastatur erzeugt werden kann ist in OPL eine solche Konstante, die auch gleich den Tastaturcode enthält. Beispiel: %A repräsentiert den Wert 65. Will man also Vergleiche anstellen, kann man diese Konstanten benutzen. In PROC get1 haben wir so die Abbruchbedingung definiert: "Beende das Programm, wenn die Taste 'q' gedrückt wurde" oder eben "UNTIL taste% = %q "

Will man aber auf die "nichtdruckenden" Tasten <ESC> und <ENTER> reagieren, sollte man sich ihre Werte einfach merken: 27 bzw. 13.

Strings gehen auch

Das Programm PROC get2: soll zeigen, daß auch Buchstaben direkt zurückgegeben werden können. wenn GET in der For GET\$ eingesetzt wird. Der Wert muß dann natürlich einem einstelligen String zugeordnet werden!

```
PROC get2:
  LOCAL taste$(1)
  DO
    taste$ = GET$
    PRINT taste$, "=", ASC(taste$)
  UNTIL LOWER$(taste$) = "q" REM "quit"
ENDP
```

Den zugehörigen Integerwert errechnet die Funktion ASC(taste\$). Diese Funktion arbeitet auch mit längeren Strings im Argument, berücksichtigt aber grundsätzlich nur das allererste Zeichen.

GET\$ ist hervorragend für Abfragen geeignet, die geradezu nach Buchstabennutzung schreien: J/N für ja/nein etc. Doch was wird der Benutzer dann wirklich eingeben? Einen Groß- oder einen Kleinbuchstaben? Am besten, wir sorgen dafür, daß das am Ende keine Rolle spielt.

Zu diesem Zweck lassen wir den String einfach in ein bestimmtes Format umwandeln sagen wir in "Buchstabenzwerge". OPL bietet uns dazu LOWER\$(taste\$) an. Alles was in taste\$ steht, wird als kleingeschrieben interpretiert, der String selber aber nicht verändert. Man ist bin so in der Lage, die Abbruchbedingung in PROC get2 mit "q" zu vergleichen, egal ob "q" oder "Q" getippt wurde. Das Gegenstück zu LOWER\$() ist UPPER\$(). Probieren Sie es selber!

Hier geht's 'raus

Sollte Sie sich bei solchen Gelegenheiten einmal vertun und der Psion kommt aus einer unendlichen Schleife nicht mehr heraus, d.h. er reagiert auf keine Eingabe mehr - keine Panik. Drücken Sie auf die Systemtaste, um in den Systembildschirm zu kommen und beenden Sie das laufende OPO-Programm in der gerätespezifischen Art. Alle Serie 3: Cursor auf das Programm stellen und <Psion>< X> drücken, Serie 5: <Ctrl><F> drücken, damit alle laufenden Programme gelistet werden und das Programm schließen.

Nach diesen Erläuterungen finden Sie sich auf Anhieb in PROC main: zurecht. So wenden wir uns nun dem nächsten Schritt zu.

Der Würfel rollt

PROC wuerfel: bietet nichts, was Sie mit dem bisher erworbenen Wissen nicht abdecken könnten.

```
PROC wuerfel:
  LOCAL taste%
  DO
    zufall: (1,6)
    CLS
    PRINT : PRINT
    PRINT " Würfel" : PRINT
    PRINT " -----> ", zahl%(1)
    PRINT
    PRINT "Enter = noch einmal, ESC = Ende"
    taste% = GET
  UNTIL taste% = 27 REM ESC-Taste
ENDP
```

Lediglich eine einzige Zufallszahl wird benötigt, deren Maximalwert 6 sein soll.

Bandit zieht drei

Auch PROC bandit: - die Simulation eines „Einarmigen Banditen“, der Gewinn verkündet, wenn alle drei Zufallszahlen den gleichen Wert besitzen - bietet keine programmtechnischen Höhepunkte mehr. Variieren Sie ein wenig den Maximalwert. Je größer der Wert ist, desto bessere "Karten" hat die Bank und Sie verlieren! Erinnert Sie das nicht irgendwie an das "richtige" Leben?

```
PROC bandit:
  LOCAL taste%
  DO
    zufall:(3,4)  REM 3 Zahlen, Maximalwert = 4
    CLS
    PRINT : PRINT
    PRINT " Gewinnspiel Bandit"
    PRINT " Drei gleiche Ziffern gewinnen!"
    PRINT
    PRINT " -----> ", zahl%(1), "-", zahl%(2), "-", zahl%(3), "-"
    PRINT
    IF zahl%(1) = zahl%(2) AND zahl%(1) = zahl%(3)
      PRINT "Gewonnen! Drei gleiche Zahlen!"
    ELSE
      PRINT "Verloren! Pechvogel!"
    ENDIF
    PRINT
    PRINT "Enter = noch einmal, ESC = Ende"
    taste% = GET
  UNTIL taste% = 27  REM ESC-Taste
ENDP
```

Die Meldung "Gewonnen" bekommen Sie immer dann, wenn die zweite der ersten Zahl gleicht UND das ebenso für den Vergleich der dritten mit der ersten gilt. Programmtechnisch verknüpft man folgerichtig diese beiden Bedingungen bei IF-Abfrage mit dem logischen Operator AND. D.h., nur wenn beide Teilbedingungen richtig sind, wird in diesen Programmzweig eingebogen.

Wenn mehrere Bedingungen zutreffen dürfen, damit ein bestimmter Programmteil abgearbeitet wird, benutzt man den logischen Operator OR (ODER). Etwa so: WENN es Äpfel sind ODER Birnen ODER Pflaumen, DANN handelt es sich um Obst, ANSONSTEN ist es irgendetwas anderes.

So nicht!

Widmen wir uns nun dem, was mittwochs und samstags die halbe Nation um den Verstand bringt Lotto. Wir kreieren hier einmal unser eigenes Lotto "6 von 66". Dazu simulieren wir im folgenden die Ziehungsmaschine, indem wir von unserem Zufallsgenerator 7 Zahlen (6 + Zusatzzahl) abfordern, dessen Maximalwert 66 betragen soll. Das folgende Programm erledigt das so:

```
PROC lottol:
  LOCAL n%
  CLS
  zufall:(7,66)
  n% = 1
  DO
    PRINT zahl%(n%),
      n% = n% + 1
  UNTIL n% > 7
  GET
ENDP
```

Wir haben es ein bißchen schlicht gehalten, denn nach wenigen Ziehungsversuchen merken Sie bereits: es treten Zahlen doppelt auf. Was uns beim "Banditen" noch willkommen war, führt hier zum Chaos! Es sieht ganz danach aus, als müßten wir unsere Anstrengungen vergrößern. Und wenn wir uns einmal dazu durchgerungen haben, sortieren wir die Zahlen am Ende auch gleich noch in einer aufsteigenden Reihenfolge ...

Einer wird gewinnen

Sehen Sie sich das Modul zuerst als Ganzes an, wir untergliedern es dann Stück für Stück. Sie können am Ende dieses "Sonderlotto" nach Ihrem Geschmack abändern oder dem "richtigen" Lotto anpassen, indem Sie die Variablen entsprechend ersetzen.

```

PROC lotto:
    LOCAL lottoz%(7), taste%, aktpos%, suchpos%, max%
    LOCAL links%, rechts%, temp%, endwert%

REM --- Zahlen holen und auf Doppel kontrollieren ---
DO
    CLS
    aktpos% = 1
    max% = 66
    WHILE aktpos% <= 7
        lottoz%(aktpos%) = 1 + INT (RND * max%)
        suchpos% = 1
        WHILE suchpos% < aktpos%
            IF lottoz%(suchpos%) = lottoz%(aktpos%)
                BREAK
            ELSE
                suchpos% = suchpos% + 1
            ENDIF
        ENDWH
        IF suchpos% = aktpos%
            aktpos% = aktpos% + 1
        ENDIF
    ENDWH
REM --- sortieren -----
links% = 1
endwert% = 6
DO
    rechts% = rechts% + 1
    DO
        IF lottoz%(rechts%) < lottoz%(links%)
            temp% = lottoz%(links%)
            lottoz%(links%) = lottoz%(rechts%)
            lottoz%(rechts%) = temp%
        ENDIF
        rechts% = rechts% + 1
    UNTIL rechts% > endwert%
    links% = links% + 1
UNTIL links% = endwert%
PRINT
    PRINT "Lotto - diese Zahlen werden gezogen:"
    PRINT
REM --- Ergebnis ausgeben -----
aktpos% = 1
DO
    IF lottoz%(aktpos%) < 10
        PRINT " ";
    ENDIF
    PRINT lottoz%(aktpos%),
    aktpos% = aktpos% + 1
UNTIL aktpos% > 6
PRINT "Zusatzzahl ", lottoz%(aktpos%)
PRINT
PRINT "Enter = noch einmal, ESC = Ende"
taste% = GET
UNTIL taste% = 27 REM ESC-Taste
ENDP

```

Das Modul unterteilt sich in drei wesentliche Abschnitte: Zahlen holen und miteinander vergleichen, sortieren und schließlich ausgeben.

Beginnen wir (wie schon einst Heinz Erhard mit seinen Gedichten) von vorn.

An der Deklaration der Variablen erkennen Sie, daß wir uns (der Übersicht halber) ein neues Variablenfeld für die Lotto-Zahlenreihe genehmigt haben. Den Variablen haben wir etwas (hoffentlich) verständlichere Namen gegeben. Weiterhin benutzen könnten wir den Zufallswert-Generator PROC zufall:, in diesem Falle wäre es aber eher umständlich und unübersichtlich, denn wir benötigen jeweils nur einen einzelnen Wert. Deshalb beschaffen wir ihn uns unmittelbar durch die unspektakuläre Programmzeile

```
lottoz%(aktpos%) = 1 + INT (RND * max%).
```

Auf keinen Fall doppelt!

Was sich nun im Programm abspielt, funktioniert genauso, als würden Sie es selber tun:

Stellen Sie sich vor, Sie haben 7 von links nach rechts durchnummerierte Fächer wie sie auf alten Postämtern noch heute zum Sortieren verwendet werden. Daneben steht ein Hut mit einer großen Anzahl von auf Zetteln notierten zufälligen Nummern von 1 bis 49 darin - es kommen also mit Sicherheit alle Werte mehrfach vor!. Aus diesem ziehen Sie nun die erste Zahl. Sie muß nicht verglichen werden, sie ist im Moment ja die einzige. Legen Sie sie einfach in das Fach 1 ab. Bereits die zweite gezogene Zahl muß einem Vergleich unterzogen werden. Sie greifen Sie sich also und sehen nach, ob sie mit der Zahl in Fach eins identisch ist.

Ist das der Fall, wird einfach eine neue Zahl gezogen und der Vergleich beginnt von vorn. Sowie beide Zahlen unterschiedlich sind, wird die zuletzt gezogene in Fach zwei abgelegt. Wir stellen uns vor Fach drei und greifen erneut in den Hut.

Der Vergleich beginnt wieder mit Fach eins. Bereits hier besteht die Möglichkeit, daß die Zahlen übereinstimmen. In diesem Falle müßte Fach zwei garnicht erst geprüft werden, wir holen uns sofort eine neue Zahl. Sind aber die neue und die Zahl in Fach eins nicht identisch, wird mit Fach zwei verglichen. Bei Identität wird eine neue Zahl gebraucht, ansonsten kommt die zuletzt gezogene Zahl in Töpfchen - sprich Fach drei.

Verschachtelt

Zur Realisierung dieser Strategie haben wir in unserem Programm zwei ineinander verschachtelte Schleifen angewandt. Die äußere sorgt dafür, daß sich am Ende sieben Zahlen im Feld lottoz%() stehen. Daß sie untereinander verschieden sind, organisiert die innere Schleife. Die Zählvariable für die Außenschleife ist aktpos% (aktuelle Position, Feldnummer). Es ist die Fachnummer, vor der wir stehen bleiben, wenn wir eine neue Nummer ziehen und begutachten.

In der Innenschleife dient uns suchpos% als Zähler. Er steuert, in welches Fach wir sehen sollen, um den Vergleich durchzuführen. Da wir immer wieder mit Fach eins beginnen müssen, wird der suchpos% - Wert eben auch vor dem Schleifeneintritt auf eins gesetzt und im Fortschreiten immer weiter erhöht.

Bei positivem Verlauf (keine doppelte Zahl) stoppt das Hochzählen erst, wenn der Zählerwert suchpos% den Wert von aktpos% erreicht. In dem Falle wird auch die Schleife abgebrochen, denn die neu gezogene Zahl mit sich selber zu vergleichen, wäre einfach sinnlos.

Wird eine identische Zahl bemerkt, kommt der Vergleichsprozeß sofort zum Stillstand - die Schleife wird durch die Anweisung BREAK sofort verlassen. Das Programm setzt seine Arbeit dann erst hinter ENDWH fort. Da die Bedingung "IF suchpos% = aktpos%" nicht stimmt, wird lediglich einer neuer Wert "gezogen", während der äußere Zähler an seiner Position - quasi vor dem Fach - verharret.

Ist der äußere Zähler durchgelaufen, haben wir erst einmal einen Zahlensalat, aber immerhin alles Unikate und das Zwischenziel erreicht.

Sortierte Blasen

Sortieren. Ein nicht ganz einfaches Thema, denn es gibt diverse Verfahren, die der Fachmann je nach Einsatzzweck auswählt. Da wir es hier nicht einem Riesenberg von Zahlen zu tun haben, sind wir in der Auswahl nicht begrenzt und greifen aus dem Angebot die Bubble Sort Methode heraus. Hier werden systematisch Werte verglichen und die jeweils kleineren und größeren in entgegengesetzte Positionen verschoben. Die großen steigen dabei quasi wie Blasen nach oben auf, was der Methode den anschaulichen Namen gegeben hat.

Greifen wir wieder unser Beispiel mit den Fächern auf. Wir wollen erreichen, daß im Fach mit der kleinsten Nummer auch die kleinste Zahl liegt. Fach sieben wird ausgeklammert - das ist die Zusatzzahl. Das Prinzip geht so:

Nimm aus dem ersten Fach den Zettel mit der gezogenen Zahl. Vergleiche sie mit der in Fach zwei. Ist die Zahl aus Fach zwei kleiner, lege sie ins Fach eins, die andere in Fach zwei. Nimm wieder die Zahl aus Fach eins und vergleiche sie diesmal mit der in Fach drei. Ein Austausch ist nur erforderlich, wenn die Zahl in Fach drei kleiner ist. So vergleicht man nach und nach die aktuelle Zahl in Fach eins mit den Werten in den anderen Fächern. Nach Vergleich mit Fach sechs endet die erste Runde. Im Ergebnis hat man nun mit Sicherheit im Fach eins die kleinste Zahl! Daher kann man dieses Fach nun auch getrost vergessen. Gedanklich stellt man sich nun vor Fach zwei, vergleicht Zug um Zug mit den weiter rechts liegenden Zahlen und tauscht nach obigem Prinzip die Plätze, wenn erforderlich. Nach diesem Durchgang liegt nun die zweitkleinste Zahl in Fach zwei. Wir liegen also richtig mit unserer Methode. Setzen Sie das Gedankenspiel selber weiter fort - Sie wissen alles, was man dazu braucht.

Verschleift noch 'mal

Im Programmtext finden sich alle beschriebenen Tätigkeiten wieder. Wie bereits beim Ziehen der Zahlen läßt sich das Problem sehr schön mit einer verschachtelten Schleife bearbeiten. Für die Fachnummer, aus der wir die Zahl holen, die mit allen anderen verglichen wird, steht die Variable `links%`. Sie wird immer dann um eins weitergezählt, wenn die Vergleichsreihe mit Fach sechs fertig ist ("äußere Schleife").

Das kleinste Fach, mit dem verglichen werden kann, befindet sich eine Position nach rechts. Wenn also in die Vergleichsrunde eingetreten wird, wird der Zähler für die innere Schleife (`rechts%`) auf einen um eins höheren Wert als `links%` gesetzt:

```
rechts% = links% + 1
```

Trifft die Bedingung zu, daß der rechte Wert größer ist, wird ein Tausch der Werte vorgenommen. Der ist nur mit Hilfe einer zusätzlichen Variablen zu bewerkstelligen - ein direkter Austausch funktioniert nicht! Sie können das gern in einem separaten "Programmchen" erproben ...

Ist die innere Schleife beim Fach sechs angelangt, wird der Zähler `rechts%` noch einmal erhöht (Wert = 7) und erfüllt so die Abbruchbedingung (`rechts%` ist größer als `endwert%`, der ja anfangs den Wert 6 bekommen hat).

Weiter geht's mit der äußeren Schleife im Sinne der prinzipiellen Erläuterungen ...

Die Zahlen lauten ...

Daß nun die Zahlen wirklich sortiert vorliegen, beweisen wir jetzt mit dem Teil 3 dieses Moduls. Die Ausgabe der Zahlen dürfte Sie als frisch gebackener Schleifenexperte nicht mehr in Verlegenheit bringen. Zwei Kleinigkeiten sind noch zu erwähnen. Wir haben den Zahlen, die kleiner als zehn sind noch eine Leerstelle vor der "richtigen" Zahl spendiert, damit die Abstände gleichmäßiger erscheinen. Außerdem lassen wir die Zahlen nacheinander (und nicht Zeile um Zeile) ausgeben. Das Geheimnis ist ein Komma oder ein Semikolon, das am Ende der PRINT - Zeile stehen muß, damit der sonst übliche Zeilenumbruch verhindert wird. Verwenden Sie ein Semikolon, wenn der folgende PRINT-Befehl keine Lücke lassen soll und ein Komma, wenn eine Lücke (ein Leerzeichen breit) eingeschoben werden soll. Das funktioniert auch innerhalb einer PRINT - Zeile:

Wenn `anzahl%` den Wert "3" hätte, ergäbe

```
PRINT "Der Baum trägt", anzahl%, „Birnen."
```

den Bildschirmausdruck:

```
Der Baum trägt 3 Birnen.
```

Unser Programm ist nun vollständig. Insgesamt gehören diese Module dazu: `main:`, `zufall:`, `mathe:`, `wuerfel:`, `bandit:` und `lotto:`.

Stellen Sie PROC main: an die Spitze aller Module, lassen Sie es übersetzen und erfreuen sich am Ergebnis. Und dann schlägt einmal mehr Ihre große Stunde: Experimentierzeit! Wir wünschen Ihnen viel Spaß damit!

Der Vollständigkeit noch einmal der Programmtext des Moduls PROC mathe:, den wir ja bereits in der letzten Folge ausführlich kommentiert haben.

```
PROC mathe:
  LOCAL eingabe%
  DO
    CLS
    zufall: (2,10)
    PRINT : PRINT
    PRINT "Berechne:",zahl%(1),"*",zahl%(2)
    PRINT "Gib das Ergebnis ein!"
    PRINT "(Null beendet das Programm)"
    INPUT eingabe%
    IF eingabe% = zahl%(1) * zahl%(2)
      PRINT "Ja!. Das ist richtig!"
    ELSE
      PRINT "Nee! Das stimmt leider nicht!"
    ENDIF
    IF eingabe% <>
      PRINT "Beliebige Taste drücken"
      GET
    ENDIF
  UNTIL eingabe%=0
ENDP
```