

OPL Workshop 2 - Basics, Zufall & Mathetrainer

Ulrich Krinzner, 10/1998

Nachdem wir uns zuletzt mit den Hilfsmitteln der OPL-Programmierung, ersten Empfehlungen zur Programmgestaltung und einem kurzen Programm beschäftigt haben, wollen wir diesmal auf Programmiergrundlagen eingehen und zeigen, wie sie in OPL umgesetzt werden.

Bei OPL handelt es sich um eine Interpretersprache (wie z.B. Visual Basic), dessen Programme in 3 Stufen erstellt wird:

- Eingabe des Programmes
- Übersetzen des Programmes
- Ausführen des Programmes (falls das Programm nicht funktioniert, können diese Schritte beliebig oft wiederholt werden).

Einführung in OPL

Prozeduren

Wie bei jeder Programmiersprache wollen wir auch in diesem Beispiel das klassische Einsteigerprogramm abdrucken:

```
PROC Main:
  PRINT "Hallo Welt"
  PRINT "Taste drücken = Programmende"
GET
ENDP
```

Anhand dieses einfachen Programmes, welches am Schirm des Psion zwei Textzeilen ausgibt und anschließend mit `GET` auf einen Tastendruck wartet, sieht man den Aufbau einer Prozedur. Prozeduren sind abgegrenzte Programmblöcke, die einfach eine Organisationsform für's Programmieren darstellen.

Jede Prozedur hat einen Namen und wird folglich mit `PROC Name:` begonnen. Beendet wird eine Prozedur mit `ENDP`. Wenn Sie sich von der Funktion des Programmes überzeugen wollen, geben Sie es am besten gleich ein. Lassen Sie es danach übersetzen und beantworten Sie die Frage, ob es gestartet werden soll mit JA. Diese einfachen Programme laufen auf allen Psions!

Ein OPL Programm wird meist in mehrere Prozeduren (auch "Module" genannt) aufgeteilt. Bei komplexeren Programmen ist es nämlich sinnvoll, die Aufgaben in Teile zu zerlegen und auf mehrere kleinere Prozeduren aufzuteilen, weil dadurch die Übersichtlichkeit steigt. Um dies wieder an einem Beispiel zu zeigen:

```
PROC Main:
  Hallo:
  PRINT "Taste drücken = Programmende"
GET
ENDP

PROC Hallo:
  PRINT "Hallo Welt"
ENDP
```

In diesem Programm haben wir die Aufgabe, "Hallo Welt" auf dem Bildschirm auszugeben, in die Prozedur `Hallo:` verlegt. Das sichtbare Ergebnis des zweiten Programms ist das gleiche wie beim ersten. Man kann sehr deutlich den Aufruf der Prozedur `Hallo:` (mit Doppelpunkt!) erkennen. Das Programm startet bei Zeile 1 und enthält in Zeile 2 die Sprungmarke zu einer Prozedur (in unserem Fall `Hallo:`). In der 2. Prozedur gibt das Programm "Hallo Welt" auf dem Bildschirm aus, kehrt wieder in die erste Prozedur zurück und setzt dort das Programm in Zeile 3 fort.

Nun kann jede Prozedur Werte an eine aufgerufene Prozedur mitgeben, die in der aufgerufenen Prozedur verarbeitet werden z.B.:

```
PROC Main:
  LOCAL str$(10)
  str$ = "Hallo Welt"
  Hallo:(str$)
  PRINT "Taste drücken = Programmende"
  GET
ENDP

PROC Hallo:(ins$)
  PRINT ins$
ENDP
```

Das Ergebnis ist wider Erwarten dasselbe, wie bei den ersten beiden Programmen. Um dieses Programm laufen lassen zu können, wurde zusätzlich eine Variable deklariert.

Variablen

Bei Variablen handelt es sich um Speicherplätze (vergleichbar mit Schubladen), in denen Werte (Zahlen, Buchstaben,...) "aufbewahrt" werden können. Will man diese Werte erneut verwenden, kann man an ihrer Stelle dessen eben ersatzweise auch die Variablen verwenden. Diese Variablen müssen, bevor man sie im Programm verwendet, deklariert werden. Das bedeutet, daß den Variablen vom Programm eben jener erforderliche Speicher reserviert wird. Die Variablen bekommen Namen. Die dürfen maximal 7 Buchstaben lang sein und werden mit bestimmten Nachzeichen (Suffix) deklariert. Nun gibt es folgende Variablen:

Bezeichnung	Suffix	Deklaration	Werte
Zahlen			
Integer	%	Name%	-32.768 bis +32.768
Long Integer	&	Name&	-2.147.483.648 bis +2.147.483.648
Fließkomma	kein Suffix	Name\$	Kommazahlen +- 9.9999999999 exp99
Buchstaben			
Text	\$	Name\$(Zahl)	Zeichen beliebiger Reihenfolge

Bei "Zahl" wird die Anzahl der Zeichen angegeben, welche in der Variablen maximal gespeichert wird.

Felder

Ein Feld ist eine Variable, die mehrere Werte des gleichen Typs speichern kann. Sie können z.B. in einem Feld, das wie folgt aussieht: **feld%(5)**, insgesamt 5 Integerwerte speichern. Bei Strings sieht das z.B. so aus: **feld\$(5, 10)**. Die zweite Zahl kommt daher, daß für einen String neben der Anzahl der Felder auch die maximal speicherbaren Zeichen pro Speicherplatz mit angegeben werden müssen. In diesem Falle können 5 Strings mit maximal 10 Zeichen gespeichert werden.

Deklaration und Wertezuweisung

Die Deklaration einer Variable erfolgt zu Anfang jedes Programmes. `REM` leitet einen Kommentar ein, der bei der Übersetzung des Programmes nicht verwendet wird, aber der eigenen Übersicht (insbesondere bei komplizierten Programmabschnitten) hilft.

```
PROC test:
  LOCAL a%, b%, c%, str$(15)
  a% = 1      REM Zuweisung von 1 zur Variablen a%
  b% = 2
  c% = a% + b% REM Die Summe der Werte der Variablen a% und b%
               REM werden der Variablen c% zugewiesen
  str$ = "1 + 2 = ";c%
  PRINT str$
  GET
ENDP
```

Das Ergebnis auf dem Bildschirm sieht so aus:

```
1 + 2 = 3
```

Anhand dieses Programms sieht man die Deklaration von Integer- und Stringvariablen und danach eine Wertezuweisung bei `a%` und `b%`. Anschließend wird `c%` die Summe von `a%` und `b%` zugewiesen. In `str$` wird eine Zeichenfolge mit der Rechenoperation und dem Ergebnis gespeichert und dann mit `PRINT` auf dem Bildschirm ausgegeben. Der Begriff "LOCAL" bedeutet, daß die Variablen nur für diese Prozedur gültig sind. Sollen sie einen weiteren Wirkungskreis haben, deklariert man ihren Gültigsbereich „GLOBAL“.

Soweit die erste Basis, auf die die weiteren Dinge aufsetzen können. Damit ein Programm überhaupt sinnvolle Dinge tun kann, muß es einerseits in der Lage sein, stur Befehl um Befehl abzuarbeiten (zur Not auch mehrfach und im Kreise), als auch andererseits aufgrund von Entscheidungen verzweigte Wege zu gehen. Die zur Verfügung stehenden Methoden beschreiben wir nun.

Schleifen und Verzweigungen

Beim Programmieren ist es notwendig und sinnvoll, gleichartige immer wiederkehrende Anweisungen nicht einzeln zu schreiben, sondern diese Operationen in Schleifen (Wiederholung von Anweisungen), Verzweigungen (Ausführen von alternativen Anweisungen) oder wie schon gezeigt mit Sprungmarken in Prozeduren zu legen.

Für Schleifen gibt es in OPL zwei Befehle:

DO..UNTIL

Beispiel:

```
PROC Main:
  LOCAL a%
  a% = 1
  DO
    PRINT "a% = ";a%
    a% = a% + 1
  UNTIL a% = 10
  PRINT "Ende der DO..UNTIL Schleife"
  GET
ENDP
```

In diesem Beispiel erkennt man den Sinn von Schleifen. Würde es keine Schleifen geben, könnte man das Problem auch lösen, allerdings wäre viel mehr Programmierarbeit notwendig und der Sourcecode wäre erheblich größer.

Bei der `DO..UNTIL` Schleife wird die Bedingung am Ende der Schleife geprüft. Diese Schleife läuft solange bis eine Bedingung erfüllt (`a% = 10`) wird. VORSICHT: Ihr Programm kann abstürzen (hängen bleiben) falls in einer Schleife die Bedingung nie erfüllt wird (Endlosschleife). Gelegentlich sind solche Endlosschleifen aber auch von Nutzen sein. Doch darüber reden wir erst viel später.

WHILE..ENDWH**Beispiel:**

```

PROC Main:
  LOCAL a%
  a% = 1
  WHILE a% < 11
    PRINT "a% = ";a%
    a% = a% + 1
  ENDWH
  PRINT "Ende der WHILE..ENDWH Schleife"
  GET
ENDP

```

Das markante der WHILE..ENDWH Schleife ist: Die Bedingung, die die Schleife steuert, wird bereits an ihrem Anfang geprüft. In unserem Beispiel wird die Schleife solange ausgeführt wie die Bedingung erfüllt wird: "Gebe solange den Wert von a% auf dem Bildschirm aus wie der Wert von a% kleiner als 11 ist".

Verzweigung per IF..ELSEIF..ELSE..ENDIF

In einem Programm sind sehr oft wichtige Entscheidungen für den weiteren Programmverlauf zu tätigen, die zum Abarbeiten alternativer Anweisungen führen. Hier eignet sich die IF..ELSEIF..ELSE..ENDIF Verzweigung. Eine IF..ENDIF Verzweigung beginnt immer mit IF-Bedingung und endet immer mit ENDIF. Die Befehle ELSEIF und ELSE können optional eingefügt werden.

Beispiel:

```

PROC Main:
  LOCAL a%
  a% = 1
  WHILE a% < 11
    IF a% = 1
      PRINT a%
    ELSEIF a% = 5
      PRINT a%
    ELSEIF a% = 3 OR a% = 7
      PRINT a%
    ELSE
      PRINT "a% = ";a%
    ENDIF
    a% = a% + 1
  ENDWH
  PRINT "Ende Programm"
  GET
ENDP

```

Die IF..ENDIF Verzweigung kann auch in Schleifen eingebaut werden. Man sieht deutlich, daß in der Schleife die Werte von a% nochmals geprüft werden. Der ELSEIF Befehl kann beliebig oft in eine IF..ENDIF Konstruktion eingebaut werden. Es allerdings darauf zu achten, daß bei einer IF..ENDIF Verzweigung immer nur eine Bedingung ausgeführt wird. Trifft also eine Bedingung zu, dann springt das Programm nach der folgenden Anweisung (z.B. PRINT a%) ans Ende hinter ENDIF. Wird keine der Bedingungen erfüllt, so wird auf ELSE verzweigt und die Anweisungen zwischen ELSE und ENDIF ausgeführt. Sollte kein ELSE vorhanden und in der IF.. ENDIF Konstruktion wird keine Bedingung erfüllt, dann passiert nichts und das Programm setzt nach der Verzweigung fort.

Mit diesen wenigen Informationen verfügen Sie nun über die wesentlichen "Geheimnisse" eines Programmierers. Nicht, daß es da nicht noch mehr zu wissen gäbe, aber Programme leben einfach von den eben beschriebenen Verfahren. Lassen Sie sich auf keinen Fall entmutigen. Mit steigender Programmierübung wächst auch Ihre Erfahrung mit OPL massiv. Wir werden Sie begleiten.

Programmbeispiel: Mathe-Trainer

Beginnen wir mit der Vertiefung des eben Gelernten anhand eines kleinen Programmes, das ausbaufähig ist. Anhand eines Zufallsgenerators werden 2 Zahlen von 1 bis 10 erzeugt, die auf dem Bildschirm dargestellt werden. Sie sollen die Zahlen im Kopf multiplizieren und das Ergebnis eingeben. Das Programm überprüft dann, ob Sie das kleine Einmaleins noch beherrschen <grins> ... Sie lernen dabei neue OPL-Funktionen und -Befehle kennen, die wir reichlich kommentieren werden, damit Sie die Schritte auch nachvollziehen können. Weil die Kommentare etwas länger ausfallen können, werden

sie nach dem OPL-Code angeordnet. Sie müssen auch nicht mit abgetippt werden. Für den Anfang verzichten wir auf "Schönheit" in der Bildschirmausgabe. Es geht uns erst mal um die Prinzipien.

Wir haben das Programm auf drei Prozeduren verteilt. Beginnen wir mit dem Hauptprogramm. Sofern man den Programmtext direkt auf dem Psion eingibt und mit mehreren Prozeduren arbeitet, ist immer die "oben" stehende erste Prozedur der Programmteil, der vom Psion auch zuerst abgearbeitet wird.

Die von uns verwendete Variable `zahl%` ist eine Feldvariable. Damit sie von allen weiteren Programmteilen benutzt werden kann (sie wird von `mathe:` und `zufall:` benötigt), haben wir sie GLOBAL deklariert. Die Anzahl der Felder ist hier zu "7" festgelegt und absichtlich etwas überdimensioniert, damit wir die Variable auch in Folgeprogrammen gleich so weiterverwenden können.

Da wir mit dem Zufallsgenerator arbeiten wollen, versetzen wir diesen gleich zu Programmbeginn in einen Grundzustand. Dazu wird der Befehl `RANDOMIZE ...` in Verbindung mit der Systemzeit verwendet. Das sichert, daß bei erneutem Programmstart tatsächlich jedesmal andere Zufallszahlen generiert werden.

```
PROC main:
  GLOBAL zahl%(7)
  RANDOMIZE MINUTE + SECOND
  mathe:
  CLS                      REM Löscht den gesamten Bildschirm
  PRINT : PRINT           REM erzeugt zwei Leerzeilen, damit die erste
                          REM Zeile nicht am oberen Bildschirmrand klebt
  PRINT "Programmende. Taste drücken"
  GET                     REM gib den Text aus und warte auf einen
                          REM beliebigen Tastendruck
ENDP
```

Die Prozedur `main:` übernimmt lediglich eine dirigierende Aufgabe und verteilt die gestellte Programmieraufgabe an das Modul `mathe:`.

Sowie das Modul `mathe:` seine Aufgabe erledigt hat, übernimmt `main:` wieder die Kontrolle und fährt mit `CLS` usw. fort. Die Schreibweise: `"PRINT : PRINT"` zeigt, daß man durchaus zwei (oder mehr) Anweisungen in eine Zeile schreiben kann, wenn man sie durch einen Doppelpunkt voneinander trennt. Üblicherweise tippt man aber jede Programmanweisung in eine eigene Zeile - das macht es am Ende einfach übersichtlicher.

```

PROC mathe:
  LOCAL eingabe%
  DO
    CLS
    zufall: (2,10)  REM Aufruf der Prozedur "zufall:", um die
                   REM Erzeugung der Zufallszahlen zu bewirken.
                   REM Die Zahlen in Klammern sind Werte (Parameter),
                   REM die an die aufgerufene Prozedur mitgegeben werden.
                   REM Anstelle der Zahlen lassen sich auch Variablen
                   REM verwenden, die vom gleichen Typ sein müssen
                   REM wie sie das aufgerufene Programm verarbeiten kann.
                   REM Die Zufallszahlen stehen nach Ablauf der Prozedur
                   REM zufall: in den zwei Feldvariablen zahl%(1) und
                   REM zahl%(2) zur Verfügung.

    PRINT : PRINT
    PRINT "Berechne:", zahl%(1), "*", zahl%(2)
                   REM Auf dem Bildschirm wird dargestellt, was
                   REM im Kopf berechnet werden soll; alle Texte
                   REM müssen in Anführungsstriche gesetzt werden,
                   REM die einzelnen Elemente werden durch ein Semikolon
                   REM getrennt, benutzt man wie hier ein Komma, wird
                   REM zwischen den Elementen auch noch ein Leerzeichen
                   REM eingefügt
    PRINT "Gib das Ergebnis ein!"
    PRINT "(Null beendet das Programm)"
    INPUT eingabe%  REM fordert zur Eingabe einer Integerzahl (der Lösung)
                   REM auf, so wird dem Psion das selbst berechnete
                   REM Ergebnis mitgeteilt
    IF eingabe% = zahl%(1) * zahl%(2)
                   REM Entscheidung: präsentiert die Eingabe ein
                   REM richtiges Ergebnis? Wenn JA, dann wird sofort
                   REM eine Erfolgsmeldung ausgegeben
      PRINT "Ja!. Das ist richtig!"
    ELSE
                   REM Wenn die IF-Bedingung nicht erfüllt ist, erfolgt
                   REM eine Fehlermeldung
      PRINT "Nee! Das stimmt leider nicht!"
    ENDIF

    IF eingabe% <> 0  REM Nur, wenn die Eingabe eine Zahl (außer Null) war,
                   REM wird die folgende Meldung am Bildschirm gezeigt
                   REM und auf den nächsten Tastendruck gewartet
      PRINT "Beliebige Taste drücken"
      GET
    ENDIF
    UNTIL eingabe%=0  REM war der Wert der Eingabe Null, beendet die
                     REM DO .. UNTIL-Schleife ihre Arbeit
  ENDP

```

In diesem Falle wird in der DO .. UNTIL-Schleife nicht gezählt und der Abbruch nach einer bestimmten Anzahl von Durchläufen gestoppt, sondern auf eine bestimmte, von außen eingefügte Bedingung gewartet. Nur wenn diese erfüllt wird, verläßt das Programm die Schleife. So können Sie die Mathe-Übung nach eigener Lust und Laune fortführen oder abbrechen.

Die folgende Prozedur `zufall:` wurde auf Erweiterbarkeit angelegt. Daher muß man an sie beim Aufruf Werte ("Parameter") übergeben. Anhand der Parameter verhält sie sich dann unterschiedlich.

Im Fall unseres Matheprogramms sollen zwei Zufallszahlen erzeugt werden. Die größte dabei vorkommende Zahl soll "zehn" sein. Deshalb übergibt die Prozedur `mathe:` eben diese Zahlen.

Die Prozedur `zufall:` nimmt die Parameter dann genau in der Reihenfolge wieder auf und weist sie eigenen Variablen zu, deren Gültigkeit sich auf diese Prozedur beschränkt. Die Variablen werden durch die vorgeschriebene Schreibweise praktisch automatisch deklariert.

```

PROC zufall:(anzahl%,max%)
  LOCAL n%          REM eine Zählvariable;
                    REM Die von der Prozedur mathe: übergebenen Parameter:
                    REM anzahl% = 2, die Anzahl der abgeforderten Werte
                    REM max% = 10, die größte zulässige Wert

  n%=1
  WHILE n%<= anzahl%
    REM Solange der Zähler noch kleiner oder gleich
    REM der Anzahl der abgeforderten Werte ist
    zahl%(n%)=1+INT(RND * max%)
    REM Erzeuge eine Zufallszahl von 1 bis 10
    REM und ordne sie der Position n% des
    REM Variablenfeldes zahl%() zu.
    n%=n%+1
    REM Erhöhe den Schleifenzähler um eins
  ENDWH
ENDP

```

In unserem Mathe-Beispiel wäre es durchaus einfacher gewesen, die Zufallszahl durch die Programmzeilen

```

zahl%(1)=1+INT(RND*10)
zahl%(2)=1+INT(RND*10)

```

zu erzeugen. Aber bereits ab sechs zu erzeugenden Werten lohnt sich der Aufbau der hier verwendeten Schleife.

Die Funktion `RND` erzeugt eine Zufallszahl vom Typ Floating Point zwischen 0 (einschließlich) und 1(ausschließlich). Multipliziert man daher den erhaltenen Wert mit 10, bekommt man ein Spektrum von Null bis 9,9999... Für unseren Zweck benötigen wir eine Ganzzahl (Integer), die wir uns durch die Anwendung von `INT` berechnen lassen. `INT` schneidet alles hinter dem Komma einer Fließkommazahl ab, so daß in unserem Beispiel Werte von Null bis 9 anfallen. Die Addition von eins ändert diesen Bereich auf 1 bis 10 - genau so wollten wir es haben!

Sie sollten für dieses Mal genügend Stoff zum Ausprobieren zur Verfügung haben. Variieren sie einfach ein wenig und schauen Sie sich jeweils das Ergebnis an. Sie wissen schon: Übung macht den Meister ... Viel Vergnügen!