Nebeneffekt bei mit EOF zu beendenden PASCAL-Einleseschleifen

Walter Riemer, N/NA, TGM

Das folgende Programm hat die Aufgabe, eine nichtnegative, höchstens dreistellige ganze Zahl zu lesen und zu verarbeiten (das heißt, auszugeben); das Ganze soll wiederholt werden, bis es mit EOF (Eingabe von CTRL-Z) beendet wird. Ungültige Zahlen sollen zurückgewiesen werden; dies erledigt die Prozedur Lesen. Damit man überhaupt in die Lese-/Prüfschleife in dieser Prozedur eintritt, wird (ein allgemein üblicher Kunstgriff!) die Zahl zunächst auf einen ungültigen Wert, nämlich 1000, gesetzt.

Das Programm funktioniert einwandfrei, wie das Ausführungsbeispiel zeigt:

```
VAR Zahl: INTEGER;
PROCEDURE Lesen;
BEGIN Zahl:=1000;
      WHILE (Zahl<0) OR (Zahl>999) DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl)
            END;
END;
BEGIN Lesen;
      REPEAT WRITELN ('Die gelesene Zahl ist ',Zahl);
             Lesen
      UNTIL EOF
END.
```

```
Zahl eingeben: -5
Zahl eingeben: 12345
Zahl eingeben: 35
Die gelesene Zahl ist 35
Zahl eingeben: 1
Die gelesene Zahl ist 1
Zahl eingeben: 0
Die gelesene Zahl ist 0
Zahl eingeben: ^Z
```

Man könnte auch auf die Idee kommen, den Bereich der gültigen Zahlen auf positive Zahlen einzuschränken, also 0 auszuschließen. Man ändert also die fünfte Zeile auf

```
      WHILE (Zahl<1) OR (Zahl>999) DO
```

In dieser Version wird EOF nicht mehr erkannt: Warum ?

```
Zahl eingeben: -5
Zahl eingeben: 12345
Zahl eingeben: 35
Die gelesene Zahl ist 35
Zahl eingeben: 1
Die gelesene Zahl ist 1
Zahl eingeben: 0
Zahl eingeben: ^Z
Zahl eingeben: ^Z
Zahl eingeben: ^Z
```

Begreiflicherweise neigt so mancher Programmierer, wenn er mit einem derartigen schwer durchschaubaren Sachverhalt konfrontiert wird, dazu, eine pragmatische Lösung zu suchen. Man könnte zum Beispiel "hoffen", daß die Sache mit einer WHILE-Schleife im Hauptprogramm besser funktioniert:

```
BEGIN Lesen;
      WHILE NOT EOF DO
            BEGIN WRITELN ('Die gelesene Zahl ist ',Zahl);
                  Lesen
            END
END.
```

Der Effekt ist aber genau der gleiche: Wenn 0 erlaubt ist, funktioniert alles, wenn 0 ausgeschlossen ist, wird `EOF` nicht mehr erkannt.

Die Ursache ist offensichtlich ein Nebeneffekt beim Einlesen von der Tastatur: Laut Standard-PASCAL (N. Wirth) sollte der Wert der zu lesenden Variablen undefiniert sein, wenn CTRL-Z eingegeben wurde; TURBO-PASCAL jedoch weicht davon ab und setzt den Wert einer Zahlenvariablen in diesem Fall auf 0.

Die Schleife in der Prozedur `Lesen` in der ersten Version des Programms wird daher nach Eingeben von CTRL-Z nicht aus Disziplin allein verlassen, sondern weil als Nebeneffekt `Zahl` auf 0 gesetzt wird; dies gilt als gültige Zahl.

In der zweiten Variante ist jedoch 0 eine ungültige Zahl, da nur positive Zahlen als gültig gelten; daher wird schon die Schleife in der Prozedur `Lesen` nicht verlassen (man bleibt in einer Endlosschleife) und CTRL-Z, das ja erst im Hauptprogramm bei `UNTIL EOF` geprüft werden würde, bleibt wirkungslos.

Man könnte nun dieses so erkannte Problem so lösen, daß man in der Prozedur `Lesen` auch auf `EOF` abfragt:

```
PROCEDURE Lesen;
BEGIN Zahl:=1000;
      WHILE ((Zahl<1) OR (Zahl>999)) AND NOT EOF DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl)
            END
END;
```

```
-5
Zahl eingeben: Zahl eingeben: 12345
Zahl eingeben: 35
Die gelesene Zahl ist 35
Zahl eingeben: 1
Die gelesene Zahl ist 1
Zahl eingeben: 0
Zahl eingeben: ^Z
```

Nun funktioniert erwartungsgemäß zwar das Erkennen von EOF, aber beim ersten Aufruf der Prozedur Lesen wird die Eingabeaufforderung zunächst übersprungen (also nicht ausgegeben), dafür aber verspätet gemeinsam mit der nächsten Eingabeaufforderung ausgespuckt.

Dieses Verhalten erscheint schon recht ungewöhnlich: Immerhin basiert die Kunst des Programmierens auf dem Axiom, daß Anweisungen, die innerhalb eines Anweisungsblocks hintereinander stehen, auch in dieser Reihenfolge ausgeführt werden. PASCAL kehrt in diesem Fall die Reihenfolge der Ausführung um, und zwar gesteuert durch die Schleife:

> `Zahl:=1000;`
> `WHILE ...` (Prüfen der Laufbedingung)
> einlesen, ohne daß TRACE auf der READ-Anweisung stehen bleibt
> `WRITE ('Zahl eingeben: ');`
> `READ (Zahl);` bewirkt nichts, obwohl TRACE auf der READ-Anweisung steht !!
> `WHILE ...` (Prüfen der Laufbedingung)
> Verlassen der Schleife, falls gültige Zahl eingegeben wurde.

Das Problem ist sofort bereinigt, wenn man vor dem Eingeben der ersten Zahl RETURN drückt; das ist natürlich sehr wenig benützerfreundlich.

Man könnte zu der Annahme kommen, daß die Erscheinung nur auf die Standard-Eingabe beschränkt ist und vielleicht in der in TURBO-PASCAL ebenfalls verfügbaren, wesentlich besseren Konsolhandhabung nicht auftritt, die allerdings explizit vereinbart werden muß (das Unit `CRT` muß eingebunden werden). Neben wesentlich schnelleren Ein-/Ausgabeoperationen, Möglichkeit der Umleitung der Ein-/Ausgabe sowie ausgefeilter Steuerung des Bildschirms im Textformat (Attribute, Fenster usw.) bietet `CRT` für unseren Fall auch noch die Möglichkeit, den Eingabeweg "Tastatur" rückzusetzen. Man könnte meinen, daß damit das Problem gelöst ist; leider ist dies jedoch auch nicht der Fall: am Verhalten des Programms hat sich nichts geändert, obwohl vom Unit `CRT` aus implizit `ASSIGNCRT(INPUT)` ausgeführt, wodurch ein `RESET(INPUT)` erst ermöglicht wird):

```
BEGIN RESET (INPUT);
      CheckEOF:=TRUE;
      Lesen;
      WHILE NOT EOF DO
            BEGIN WRITELN ('Die gelesene Zahl ist ',Zahl);
                  Lesen
            END
END.
```

Selbstverständlich dürfte auch nicht darauf vergessen werden, die in `CRT` definierte Systemvariable `CheckEOF` auf `TRUE` zu setzen, da sonst CTRL-Z überhaupt nicht als gültige Eingabe akzeptiert wird und daher auch nicht erkannt werden kann.

Die nächste Idee könnte sein, die Situation mit einem Kunstgriff zu bereinigen, der nicht ganz den Prinzipien strukturierten Programmierens entspricht: Man setzt einfach die Variable `Zahl`, falls sie durch CTRL-Z-Eingabe 0 wurde, auf einen gültigen Wert, zum Beispiel 1, damit die Schleife in der Prozedur `Lesen` verlassen wird.

```
PROGRAM LesProc1;
VAR Zahl: INTEGER;
PROCEDURE Lesen;
BEGIN Zahl:=1000;
      WHILE (Zahl<1) OR (Zahl>999) DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl);
                  IF Zahl=0
                     THEN Zahl:=1 {Zahl gültig machen}
            END
END;
BEGIN Lesen;
      WHILE NOT EOF DO
            BEGIN WRITELN ('Die gelesene Zahl ist ',Zahl);
                  Lesen
            END
END.
```

Das gleiche Prinzip kann natürlich auch für die Variante mit REPEAT-Einleseschleife angewendet werden, jedoch zeigt sich hier für diesen speziellen Fall, daß die WHILE-Einleseschleife, weil sie kopfgesteuert ist, zu bevorzugen ist: die REPEAT-Schleife wird nämlich jedenfalls betreten, selbst wenn bei der allerersten Eingabe schon CTRL-Z eingegeben wird; dies führt zu unerwünschten Ausgaben:

Zahl eingeben: Die gelesene Zahl ist 1

Zahl eingeben: (nach der ersten Eingabeaufforderung wurde CTRL-Z eingegeben)

```
PROGRAM LesProc0;
VAR Zahl: INTEGER;
PROCEDURE Lesen;
BEGIN Zahl:=1000;
      WHILE (Zahl<1) OR (Zahl>999) DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl);
                  IF Zahl=0
                     THEN Zahl:=1 {Zahl gültig machen}
            END
END;
BEGIN Lesen;
      REPEAT WRITELN ('Die gelesene Zahl ist ',Zahl);
             Lesen
      UNTIL EOF
END.
```

Allen Verwirrspielen zum Trotz gibt es aber doch eine einwandfreie Lösung; der Schlüssel liegt gar nicht in der EOF-Behandlung! Der Eintritt in die Lese-/Prüfschleife beruht, wie schon oben erwähnt, darauf, die Zahl zunächst auf einen ungültigen Wert zu setzen:

```
PROCEDURE Lesen;
BEGIN Zahl:=1000;
      WHILE ((Zahl<1) OR (Zahl>999)) AND NOT EOF DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl)
            END
END;
```

Wenn nun zusätzlich auch noch die Abfrage auf `NOT EOF` Bestandteil der Laufbedingung geworden ist, ist es entscheidend, wie der logische (BOOLEsche) Ausdruck in der WHILE-Anweisung ausgewertet wird. Im vorstehenden Beispiel ist die Zahl zunächst ungültig, der OR-Ausdruck ist daher `TRUE`. Er muß noch mit `NOT EOF` Und-verknüpft werden, also braucht das Programm einen Zugriff auf die Eingabedatei "Tastatur", um den Status "EOF oder nicht?" festzustellen; dies erfordert eine Eingabe, zumindest mittels RETURN-Taste.

Wenn man diesem unerwünschten Verlangen des Programms nach einer mit RETURN abzuschließenden Eingabe Rechnung trägt, ist das Problem mit einem Schlag gelöst:

```
PROCEDURE Lesen;
BEGIN WRITE ('Zahl eingeben: ');
      READ (Zahl);
      WHILE ((Zahl<1) OR (Zahl>999)) AND NOT EOF DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl);
            END
END;
```

Jetzt wird zunächst einmal gelesen und damit der Status "EOF oder nicht?" hergestellt. Nur wenn gleich bei der ersten Eingabemöglichkeit CTRL-Z eingegeben wurde, ist der Ablauf noch nicht perfekt: Der OR-Ausdruck ist `TRUE`, `NOT EOF` aber `FALSE`; `TRUE AND FALSE` ergibt natürlich `FALSE`, also wird die Schleife, wie gewünscht, nicht ausgeführt. Durch den Nebeneffekt des Nullsetzens der Variablen Zahl wird jedoch in der REPEAT-Schleife eine Ausgabe durchgeführt "Die gelesene Zahl ist 0" und eine weitere Eingabe von CTRL-Z verlangt, was wohl unerwünscht ist; wenn man jetzt eine Zahl eingibt, kümmert sich das Programm überhaupt nicht mehr um das vorher eingegebene CTRL-Z! Diese Erscheinung (welche nicht auftritt, wenn beim ersten Lesen eine gültige Zahl eingegeben wurde) kann man unter Beibehaltung der REPEAT-Schleife nur mittels unschöner, unstrukturierter Abfrage beseitigen. In diesem Fall erweist sich die WHILE-Schleife als geeigneter:

```
VAR Zahl: INTEGER;
PROCEDURE Lesen;
BEGIN WRITE ('Zahl eingeben: ');
      READ (Zahl);
      WHILE ((Zahl<1) OR (Zahl>999)) AND NOT EOF DO
            BEGIN WRITE ('Zahl eingeben: ');
                  READ (Zahl);
            END
END;
BEGIN Lesen;
      WHILE NOT EOF DO
            BEGIN WRITELN ('Die gelesene Zahl ist ',Zahl);
                  Lesen
            END
END.
```

Und hiermit ist endlich die "perfekte" Lösung gefunden! Man sollte sich abschließend bewußt machen, daß die Hauptschwierigkeit eigentlich daher kam, daß man die von der Urversion stammende Bedingung für das Eintreten in die Schleife `Zahl:=1000`, also Ungültigmachen der Variablen `Zahl`, gar nicht mehr in Frage gestellt hat. Es lohnte sich jedenfalls, auch die Auswertung der Laufbedingung nach den Regeln der BOOLEschen Algebra zu analysieren, zum Beispiel bewirkte die Fassung

```
PROCEDURE Lesen;
BEGIN WRITE ('Zahl eingeben: ');
      READ (Zahl);
      WHILE (Zahl<1) OR (Zahl>999) AND NOT EOF DO
            BEGIN WRITE ('Zahl eingeben: ');
```

, daß nach Eingabe von CTRL-Z die Lese-/Prüfschleife nicht mehr verlassen wird, während

```
PROCEDURE Lesen;
BEGIN WRITE ('Zahl eingeben: ');
      READ (Zahl);
      WHILE NOT EOF AND (Zahl<1) OR (Zahl>999) DO
            BEGIN WRITE ('Zahl eingeben: ');
```

einwandfrei funktioniert. Im ersten Fall ist der erste logische Operand wegen `Zahl=0` schon `TRUE`; gleichgültig, ob er mit einem `TRUE` oder einem `FALSE` als Ergebnis der Und-Verknüpfung Oder-verknüpft wird, die Laufbedingung bleibt `TRUE`.

Abhilfe ist auch dadurch möglich, daß man die Oder-Verknüpfung nochmals in Klammern einschließt. Dann ergibt sie für sich `TRUE`, welches mit `NOT EOF = FALSE` Und-verknüpft jedenfalls `FALSE` ergibt, wodurch die Schleife nicht mehr betreten wird.

Die Erklärung für den zweiten Fall ist, daß nach Eingabe von CTRL-Z schon der erste logische Operand `FALSE` ist und die darauffolgenden Operanden können wegen der Und-Verknüpfung daran nichts mehr ändern.

Abschließend sollte noch darauf hingewiesen werden, daß TURBO-PASCAL normalerweise die Auswertung eines logischen Ausdrucks abbricht, sobald das Ergebnis feststeht; nur wenn man explizit den Compilerschalter `$B+` setzt, erfolgt immer eine vollständige Auswertung. □