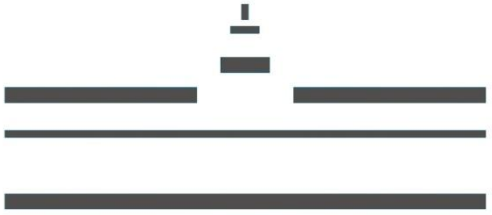


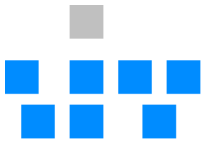
Westfälische
Wilhelms-Universität
Münster

› Lösung eines Sudokus mit PL/I

München 2009

Eberhard Sturm
ZIV (Uni-RZ)
Münster
sturm@uni-muenster.de

wissen.leben
WWU Münster



ZENTRUM FÜR
INFORMATIONS
VERARBEITUNG

› Worum geht es eigentlich?

20:57

Zahlenrätsel

- Ein leichtes Sudoku:

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

5 8 9   6 2 1   3 7 4
3 4 6   9 7 8   2 5 1
1 2 7   5 4 3   8 6 9

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

46 Zahlen fehlen!

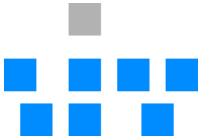
- Ein schweres Sudoku:

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

5 1 6   4 8 2   3 7 9
4 7 9   5 1 3   6 8 2
2 3 8   6 9 7   4 1 5

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

53 Zahlen fehlen!



› Alles ist möglich …

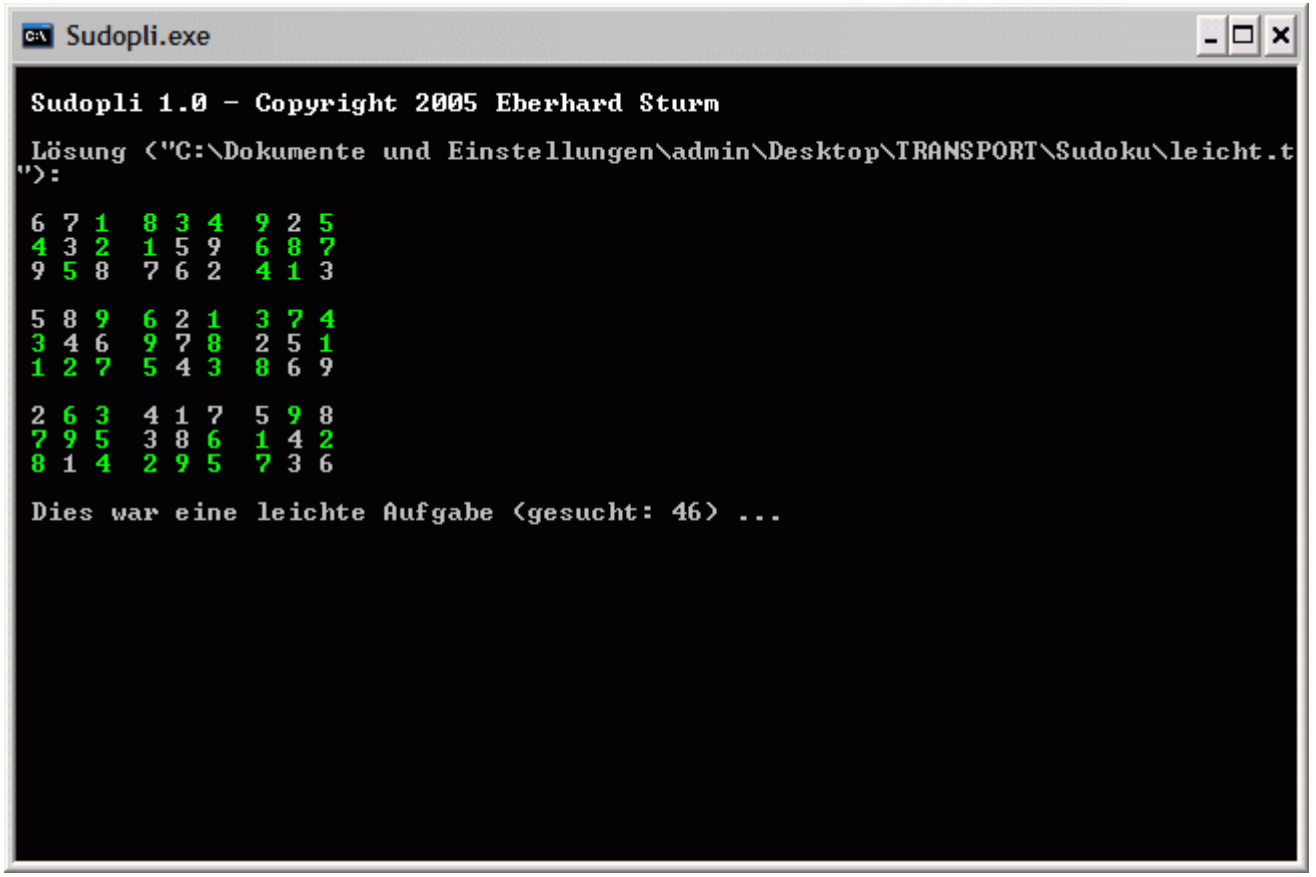
20:57

Wie lösbar?

- leicht : 
- mittelschwer : 
- schwer : 
- unlösbar: 
- mehrdeutig: 
- leer: 

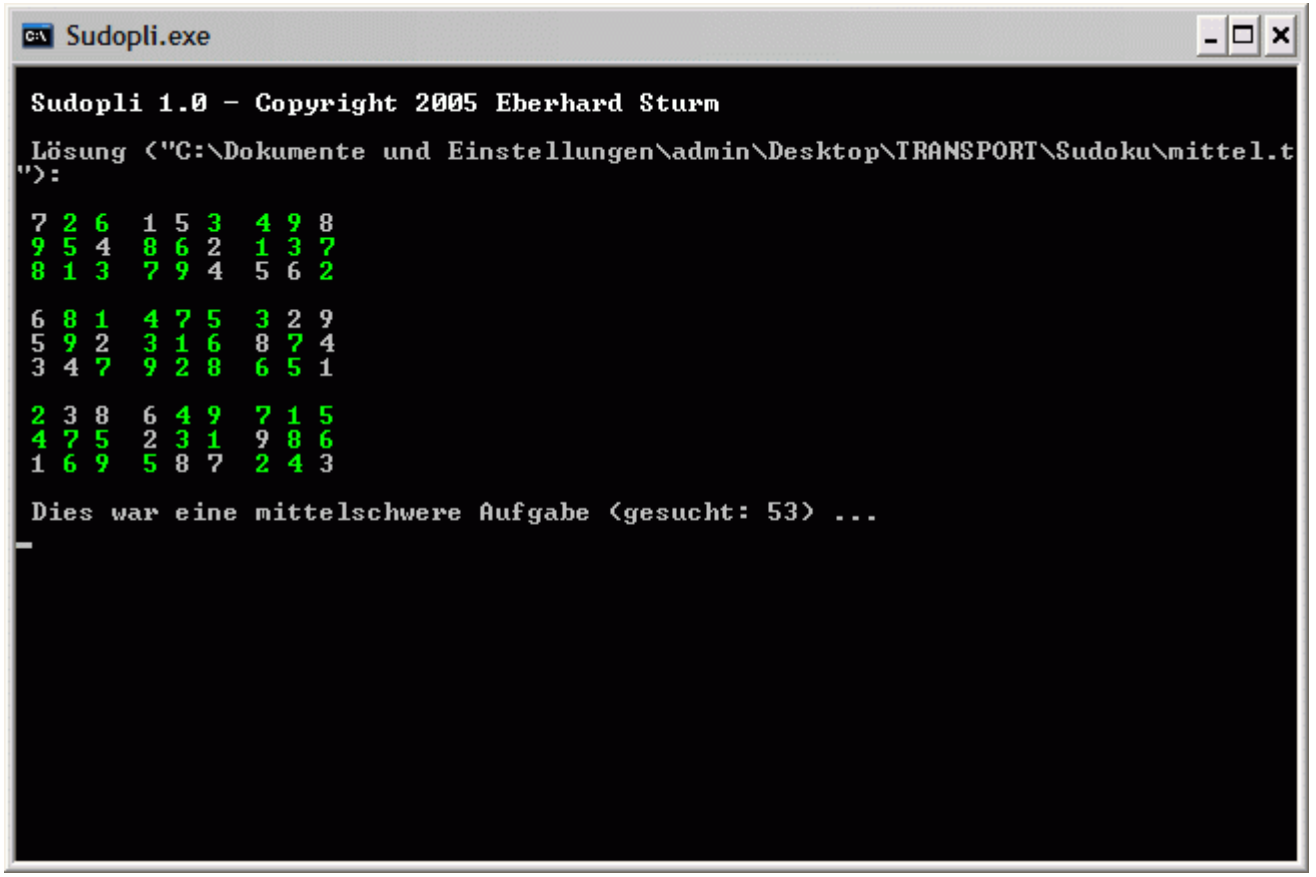
› Ein leichtes Sudoku

20:57



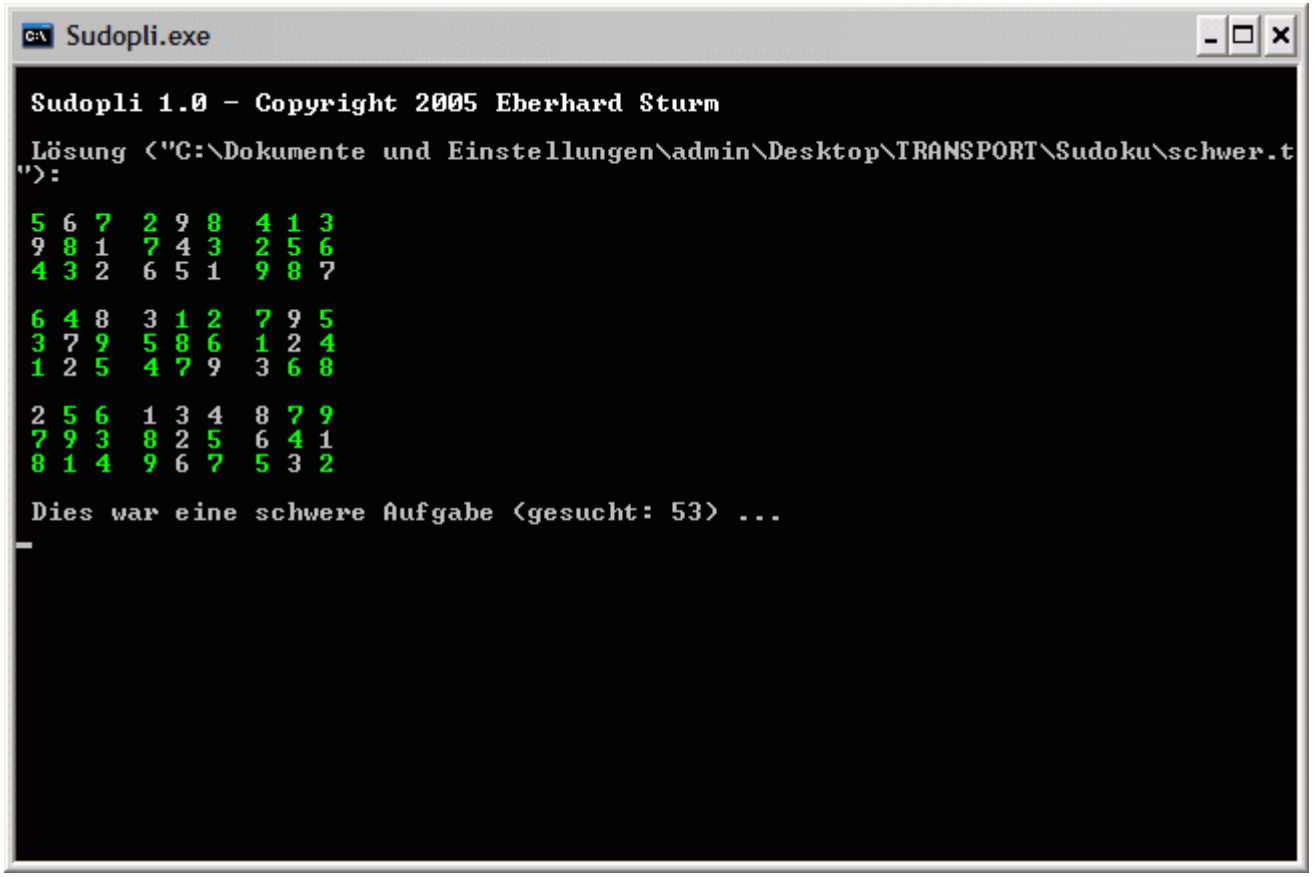
› Ein mittelschweres Sudoku

20:59



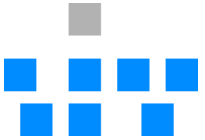
› Ein schweres Sudoku

21:00



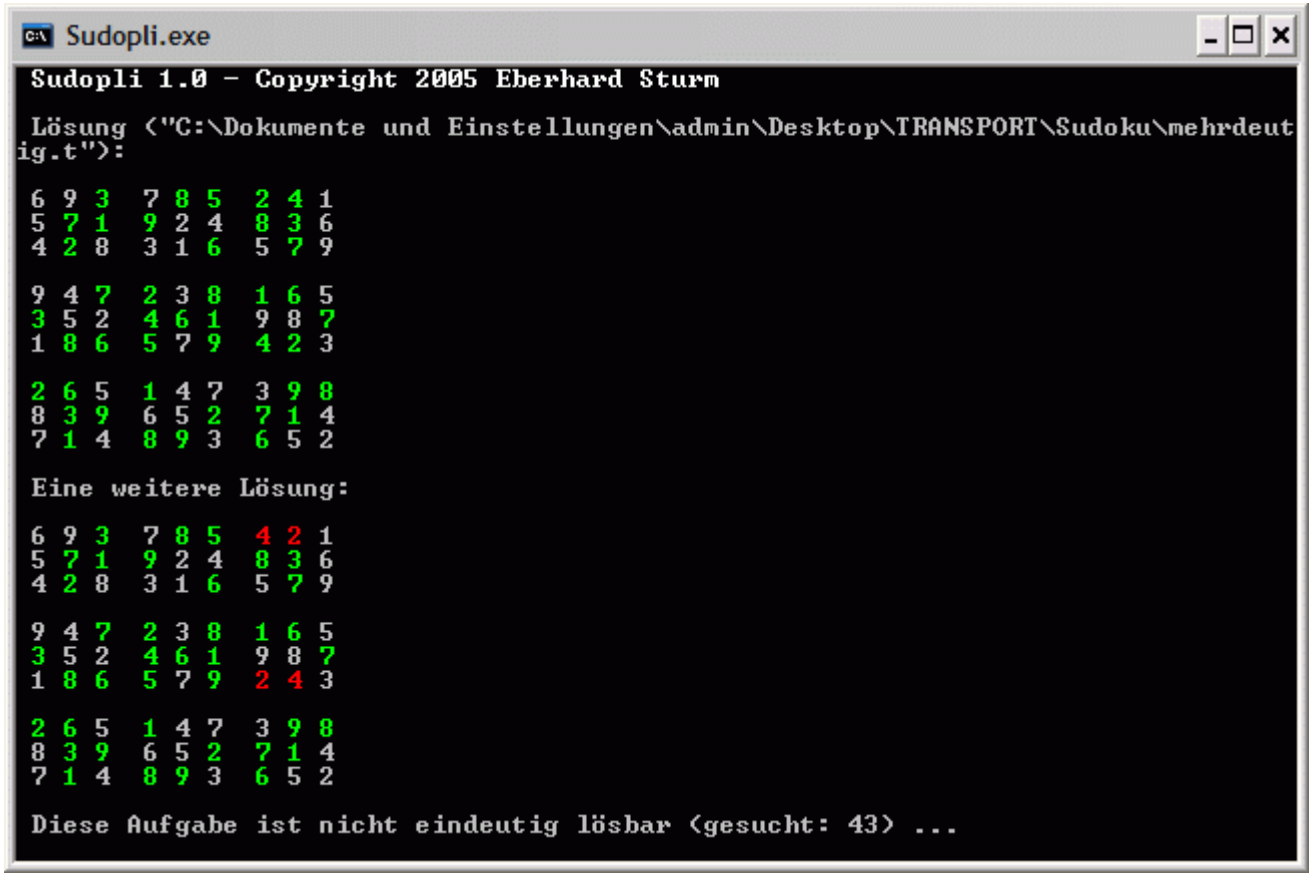
› Ein unlösbares Sudoku

21:00



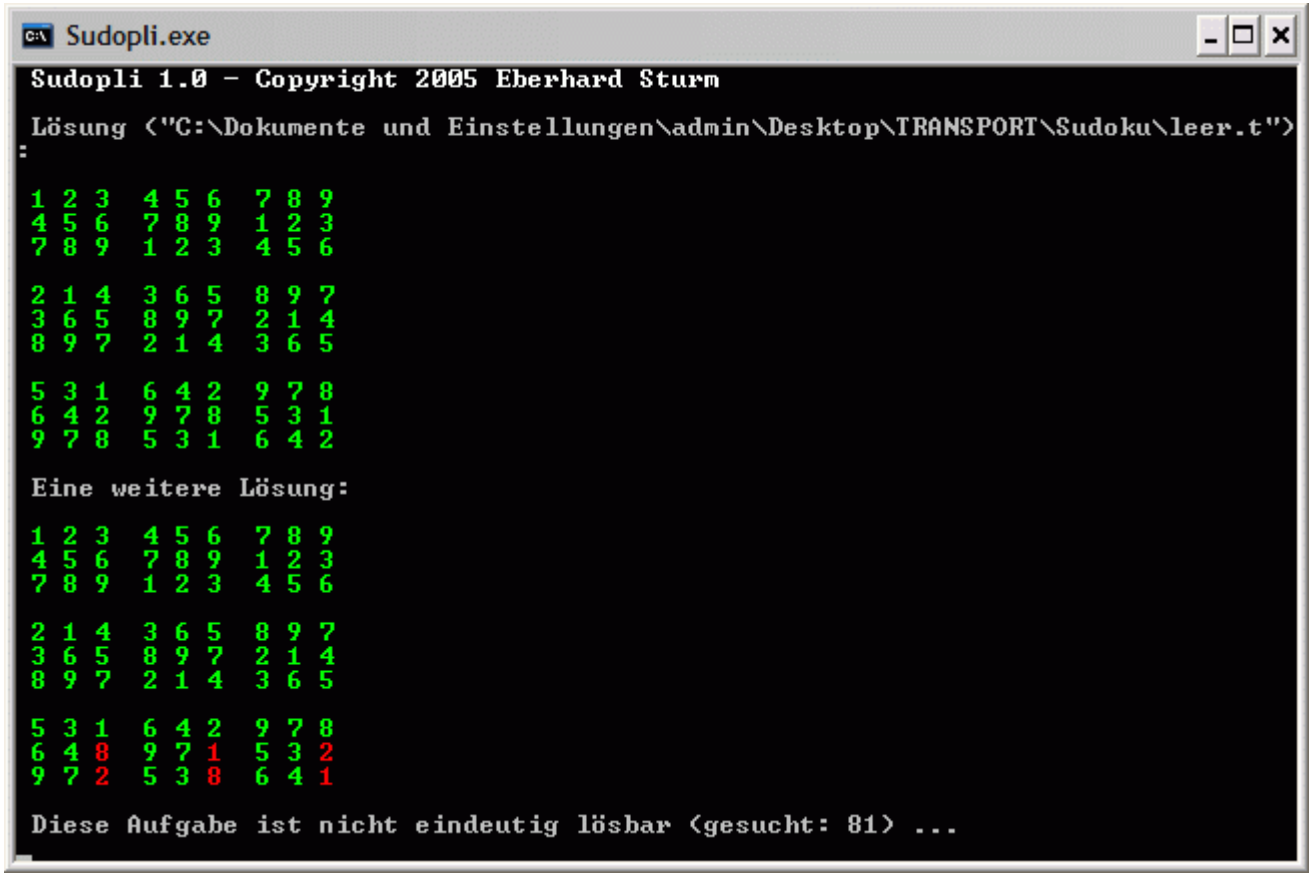
› Ein mehrdeutiges Sudoku

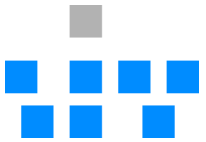
21:02



› Ein leeres Sudoku

21:02





› Warum gerade in PL/I?

Idee

- Wesentlich an Sudoku: Matrizen!
- Zeilen, Spalten, Kästen (Untermatrizen)
- Zeilen und Spalten sind problemlos, aber wie kann man Kästen ansprechen?
- Wiederholung **`defined`**-Attribut:

1. Korrespondenz-Definition:

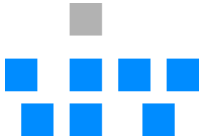
```
dcl A dim (10,10) float, B dim (3,3) float def A;
```

2. Überlagerungsdefinition (siehe auch **`cell`**/**`union`** und **`based`**):

```
dcl A dim (10,10) char, B dim (100) char
                        def A pos (K);
```

3. iSUB-Definition:

```
dcl A dim (10,10) bit (9),
    B dim (10) bit (9) def A(1sub,1sub);
```



› Warum gerade in PL/I?

Realisierung

- Wenn man festhalten will, welche der Zahlen von 1 bis 9 für ein Feld noch möglich sind, schreibt man am besten:

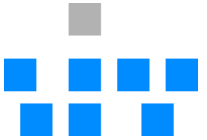
```
define alias bits bit (9) aligned;
```

- Dann kann man Zeilen, Spalten und Kästchen folgendermaßen deklarieren:

```
dcl V_xxx dim (81)    type bits;
dcl Z_xxx dim (9, 9)  type bits based (addr(V_xxx));
dcl K_xxx dim (9, 3, 3) type bits
          defined (V_xxx(trunc((1sub-1)/3)*27
                         + mod(1sub-1, 3)*3
                         + (2sub-1)*9
                         + 3sub));
```

- Umwandlung von Zahl in **type bits**:

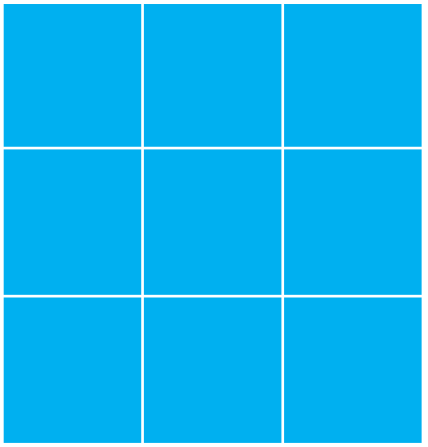
```
dcl Posbit dim (0:9) type bits init ('000000000'b,
  '100000000'b, '010000000'b, '001000000'b,
  '000100000'b, '000010000'b, '000001000'b,
  '000000100'b, '000000010'b, '000000001'b);
```



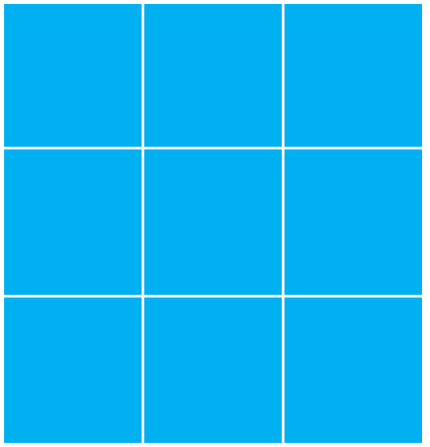
› Arbeitsmaterial

Matrizen

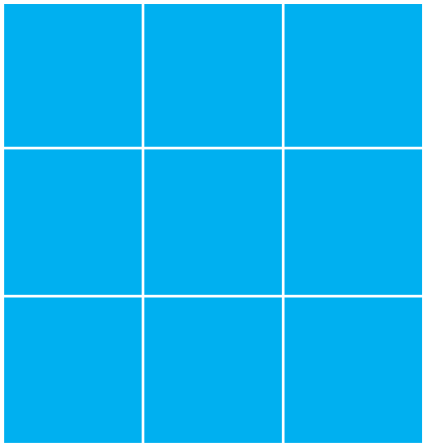
- **`V_ein`**
  (wie eingelesen):



- **`V_tst`**
  (mehrdeutig):



- **`V_aus`**
  (eindeutig):



› Nach dem Abtippen ...

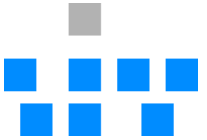
Einlesen

- Bei Null oder Leerzeichen ist die Zahl zu raten:

```
open file (Sysin)
    title ('/' || Parm || ',recsize(1000)');
do I = 1 to 9;
   get file (Sysin) edit (S) (l);
   S = left(S, 9, '0');
   get string (S) edit (Z_ein(I, *)) (f(1));
   end;
close file (Sysin);
```

- Umwandeln in **type bits**:

```
Anzahl_gesucht = 0;
do I = 1 to 81;
   V_aus(I) = Posbit(V_ein(I));
   if V_ein(I) = 0 then Anzahl_gesucht += 1;
   end;
```



› Je nach Schwierigkeitsgrad …

20:57

3 Versuche

- Falls leicht:

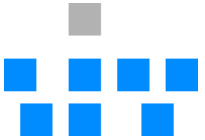
```
do loop;
   V_tst = Posbit(0); /* alles auf Null */
   call Berechnen (Fehlzahl, Alles_durchlaufen);
   if Alles_durchlaufen then leave;
   end;
```

- Falls mittelschwer:

```
do loop;
   call Auswählen (Etwas_gefunden);
   if ¬Etwas_gefunden then leave;
   do loop;
      V_tst = Posbit(0);
      call Berechnen (Fehlzahl, Alles_durchlaufen);
      if Alles_durchlaufen then leave;
      end;
   if Fehlzahl = 0 then leave;
   end;
```

- Falls schwer:

```
on condition (Abbruch) goto Ende;
call Setzen (1, Fertig, Parm);
```



› Wenn es nur noch eine Möglichkeit in einem Feld gibt, setzen wir diese Zahl und fangen ganz von vorn an.

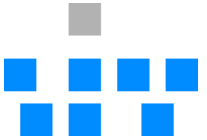
20:57

Versuch 1

```
do I = 1 to 9;
   do J = 1 to 9;
      if Z_aus(I, J) = Posbit(0)
         then do; /* Hier fehlt eine Zahl. */
            Fehlzahl += 1;
            K = Kastennummer(I, J);
            Z_tst(I, J)  = ¬(any(Z_aus(I, *))
                           | any(Z_aus(*, J))
                           | any(K_aus(K, *, *)));
               /* Es sind die Bits an, die noch möglich sind */
            if tally(Z_tst(I, J), '1'b) = 1 then do;
                  /* Genau eine Zahl ist möglich! */
               Z_aus(I, J) = Z_tst(I, J);
               Alles_durchlaufen = '0'b;
               Fehlzahl -= 1; /* fehlt doch nicht */
               return;
               end;
            end;
         else /* Zahl ist vorhanden. */
            Z_tst(I, J) = Z_aus(I, J);
               /* Es ist nur ein Bit an. */
      end;
   end;
```

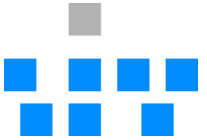
sum → +

any → |



Versuch 2

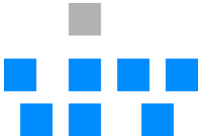
```
Etwas_gefunden = '1'b;
do L = 2 to 8; /* Anzahl Zahlen pro Feld */
   do I = 1 to 9;
      do J = 1 to 9;
         if tally(Z_tst(I, J), '1'b) = L then do;
               /* Es gibt hier L Möglichkeiten (L > 1). */
            do M = 1 to L;
               Array(M) = Zahlpos(Z_tst(I, J), M);
          /* Hier haben wir alle L Zahlen bei (I,J) versammelt. */
               end;
            Hilf = Z_tst(I, J);
            Z_tst(I, J) = Posbit(0); /* hier keine Zahl */
            if In_zeile_gefunden(I, J, Array, L)  then return;
            if In_spalte_gefunden(I, J, Array, L) then return;
            K = Kastennummer(I, J);
            if In_kasten_gefunden(I, J, K, Array, L) then return;
            Z_tst(I, J) = Hilf;
            end;
         end;
      end;
   end;
Etwas_gefunden = '0'b;
```



Für Restfelder alle Möglichkeiten durchprobieren!

Versuch 3

```
Fertig = '0'b;
do Pos = Anfangspos to 81;
   T = tally(V_tst(Pos), '1'b);
   if T > 1 then do;
      do Num = 1 to T;
         Bits = Posbit(Zahlpos(V_tst(Pos), Num));
            /* Die Zahl der NUM-ten Möglichkeit als BITS */
         V_aus(Pos) = Bits; X_tst = V_tst; V_tst(Pos) = Bits;
         if Durchstreichen_möglich(Bits, Pos) then do;
            call Setzen (Pos+1, Fertig, Parm);
            if Fertig then if Lösung_gefunden
               then do; /* nicht eindeutig */
                  call Anzeige2 (Z_tst, Z_mem);
                  signal condition (Abbruch); end;
               else do; /* schwere Aufgabe */
                  call Anzeige (Z_tst);
                  Z_mem = Z_tst;
                  Lösung_gefunden = '1'b; end;
            Fertig = '0'b; end;
         V_tst = X_tst;
         V_aus(Pos) = Posbit(0); end;
      return; end; end;
Fertig = '1'b;
```



Ausschnitte

- Aus **Durchstreichen_möglich**:

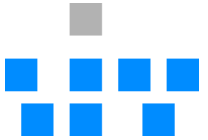
```
Z_tst(I, JJ) &= XX;
Z_tst(II, J) &= XX;
K_tst(K, II, JJ) &= XX;
XX = ¬XX;
```

- Aus **In_zeile_gefunden**, **In_spalte_gefunden**, **In_kasten_gefunden**:

```
Bits = any(Z_tst(I, *));
Bits = any(Z_tst(*, J));
Bits = any(K_tst(K, *, *));
if substr(Bits, M, 1) = '0'b then ...
```

- Aus **Anzeige**:

```
T = trim(char(index(Z(I, J), '1'b)));
```



› Eine Anweisung erweitern …

20:57

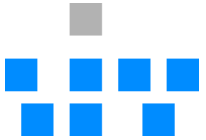
Ausgabe

```
Anzeige: procedure (Z);

dcl Z dim (9, 9) type bits parm nonasgn;
dcl (I, J)       fixed bin (31);
dcl T            char (9) var;

do I = 1 to 9;
   display (' ');
   display (' ') sameline; /* kein NL am Ende */
   do J = 1 to 9;
      T = trim(char(index(Z(I, J), '1'b)));
      if Z_ein(I, J) = 0
         then
            display (T || ' ') sameline color (green);
         else
            display (T || ' ') sameline;
      if mod(J, 3) = 0 & J ¬= 9 then display (' ') sameline;
      end;
   if mod(I, 3) = 0 & I ¬= 9 then display (' ');
   end;

end Anzeige;
```



› Zusammenfassung

20:57

Ausschnitte

- Ohne PL/I kein Sudoku-Programm
- Definition dreier Schwierigkeitsgrade:
  - eindeutige festlegen
  - mehrdeutige festlegen
  - verbleibende Felder durchprobieren
- Darstellung der Möglichkeiten in **`bit (9)`**
- Darstellung des Bretts mit Matrizen
- Kästchen sind normale Untermatrizen (iSUB)
- Bit-Operationen (**`&`**, **`¬`**)
- Matrix-Builtin-Funktionen (**`any`**)
- String-Builtin-Funktionen (**`tally`**, **`index`**)
- **`display`**-Makro mit Farben