

# Test Driven Development mit TUSTEP

am Beispiel von #SORTIERE

# Gliederung

- 1) Test Driven Development (TDD)
- 2) TDD mit TUSTEP
- 3) #SORTIERE und #SVORBEREITE
- 4) TDD am Beispiel von #SORTIERE und #SVORBEREITE

# Test Driven Development (TDD)

# Test Driven Development (TDD)

- TDD ist ein Paradigma der Softwareentwicklung.
- Es wurde von Kent Beck im Rahmen agiler Softwareentwicklung popularisiert (Literatur: Kent Beck: Test-Driven Development By Example. 2002).
- Die Grundidee besteht darin, dass die zu entwickelnde Software bestimmte **Anforderungen** erfüllen soll. Diese Anforderungen werden in Form von **Tests** beschrieben.
- Die zu entwickelnde Software gilt nur dann als korrekt, wenn sie alle Tests erfolgreich besteht.

# Test Driven Development (TDD)

- Softwareentwicklung mit TDD erfolgt in drei sich ständig in kurzen Abständen wiederholenden Phasen:
  - 1) **Schreiben eines Tests**, der genau eine Anforderung der zu entwickelnden Software beschreibt.
  - 2) **Schreiben** nur genau **des benötigten Codes**, der dazu führt, dass die Software genau diesen Test besteht.
  - 3) **Refactoring**, d. h. Aufräumen des Quelltextes ohne an der Funktionalität etwas zu ändern. [Bei jedem Schritt des Refactorings werden alle Tests erneut ausgeführt, z. B. Umbenennen von Variablen, Entfernen von Duplikation, Abstraktion von Funktionen etc.]

# Test Driven Development (TDD)

- Zentrale Vorteile von TDD gegenüber „herkömmlicher“ Programmierung:
  - Das Verhalten der Software kann jederzeit durch das Ausführen der Tests objektiv **geprüft** werden.
  - Die entwickelte Software ist möglichst **modular** aufgebaut, da sie immer nur in sehr kleinen Bestandteilen entwickelt wird (Test für Test).
  - Sofern die Tests ausreichend formuliert sind, ist eine aufwändige Fehlersuche (**Debugging**) nicht notwendig, neu eingeführte Fehler fallen schnell auf.
  - Fügt man neue Funktionen hinzu, muss man kaum befürchten, an anderer Stelle etwas „kaputt“ zu machen. TDD bietet **Sicherheit**.

# Test Driven Development (TDD)

- Ein Test besteht aus drei Schritten:
  - a) Arrange: **Testdaten** (u. ggfs. das erwartete Ergebnis) **werden vorbereitet.**
  - b) Act: Das zu testende **Programm wird** mit den Testdaten **durchgeführt.**
  - c) Assert: Das **Testresultat wird** auf Übereinstimmung mit dem erwarteten Ergebnis **geprüft.**

Entspricht das Testresultat den Erwartungen, war der Test erfolgreich, ansonsten schlägt der Test fehl.

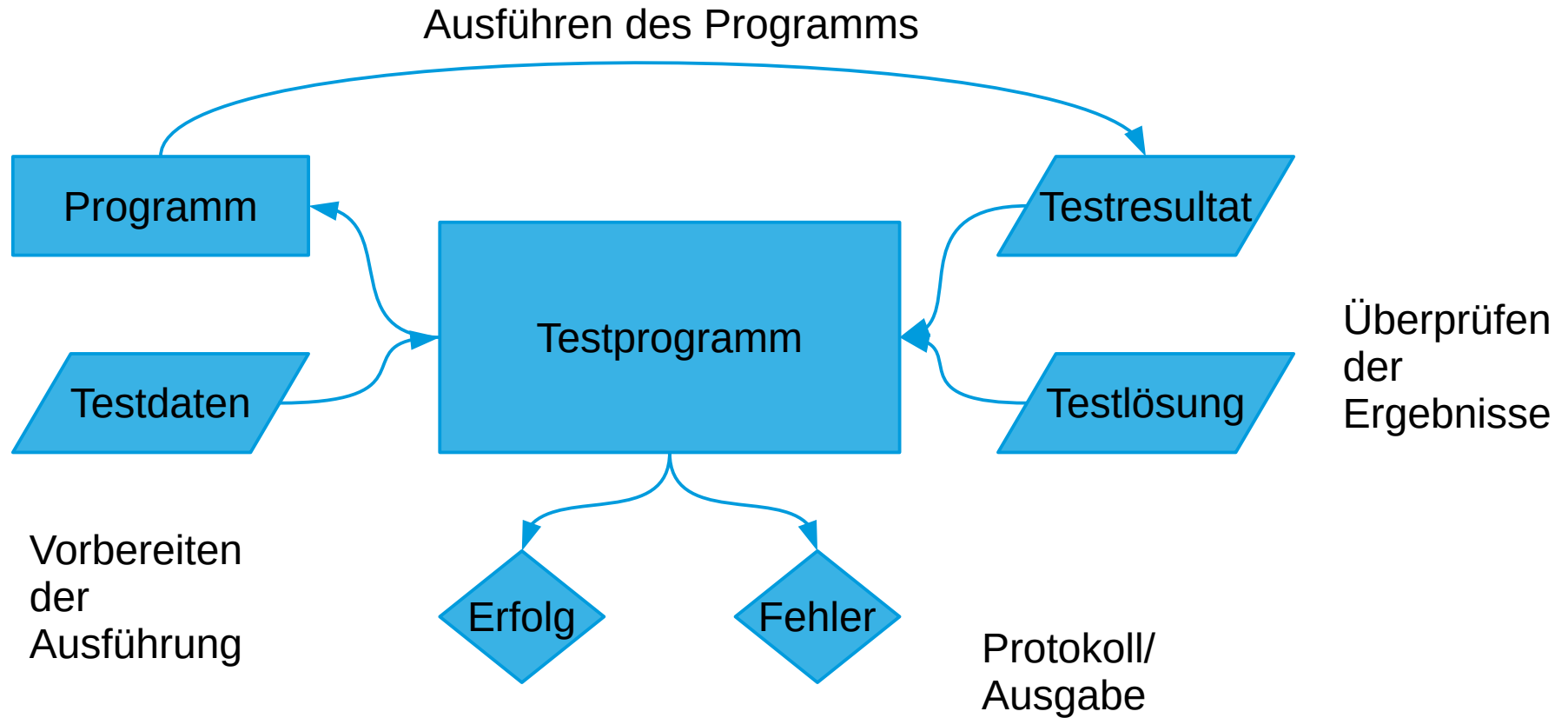
TDD mit TUSTEP?



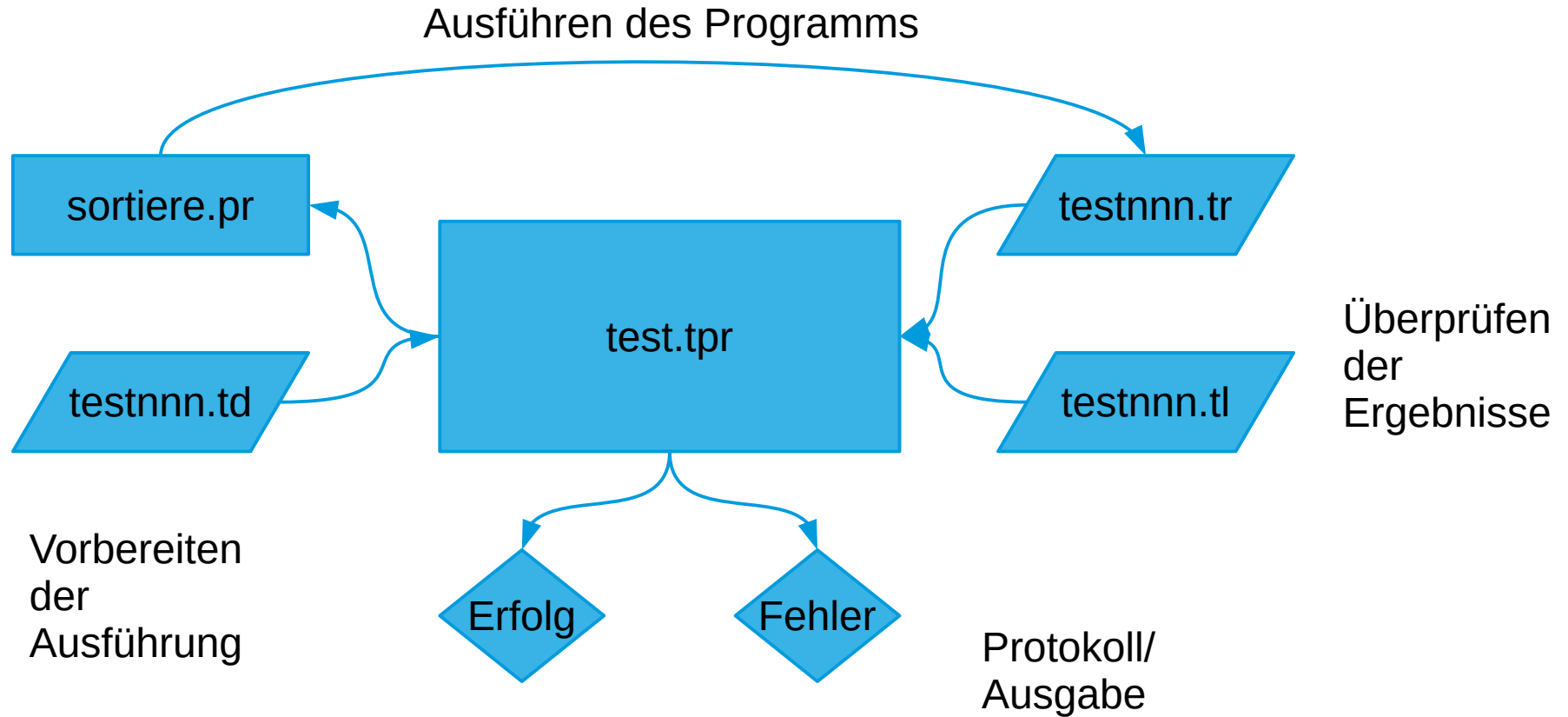
# TDD mit TUSTEP

- TDD ist auch mit TUSTEP möglich.
- Benötigte Dateien:
  - a) Permanente Testdateien (.td) enthalten die Daten.
  - b) Permanente Lösungsdateien (.tl) enthalten die erwarteten Ergebnisse.
  - c) Das Testprogramm (.tpr) enthält den Aufruf des zu testenden Programms.
  - d) Das zu testende Programm (.pr) ist die Software, die entwickelt werden soll.
  - e) Temporäre Resultatdateien (.tr) enthalten die tatsächlichen Ergebnisse.
  - f) [Optional, aber empfohlen:] Eine Konfigurationsdatei (.cfg) enthält zum Beispiel Namen, Nummern und sonstige Informationen über die Tests.

# TDD mit TUSTEP

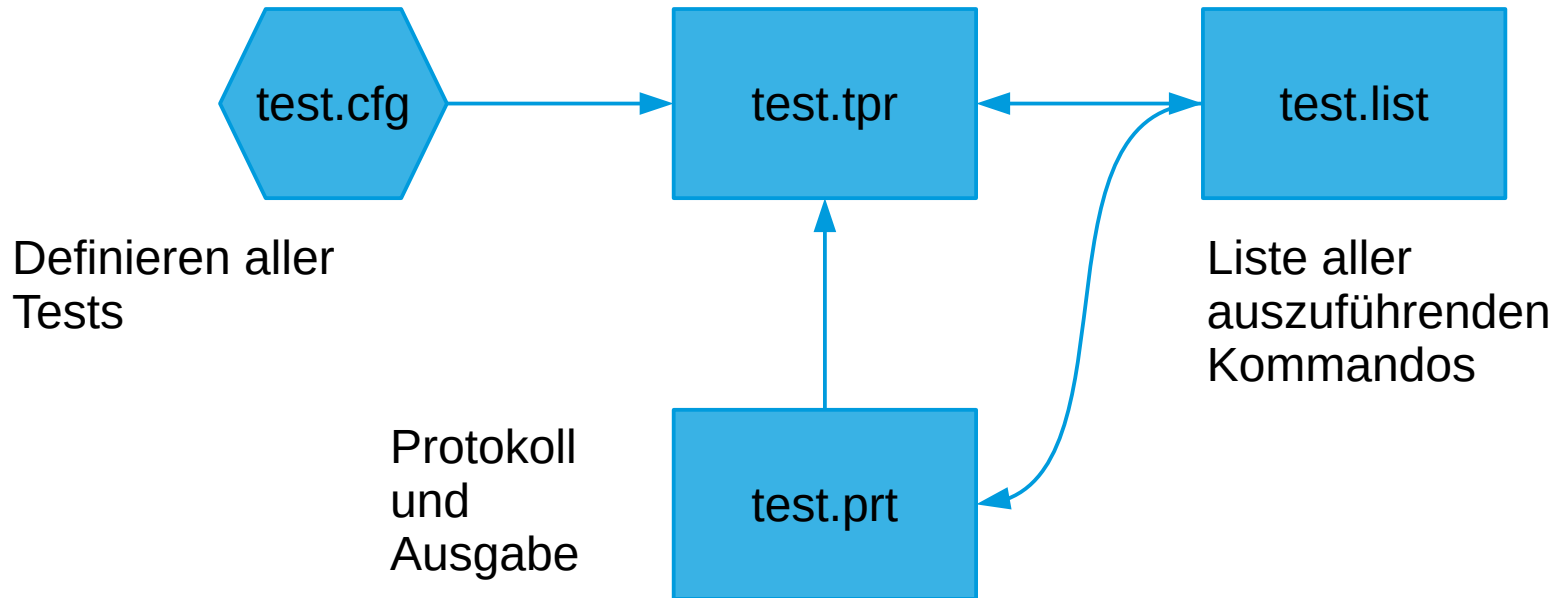


# TDD mit TUSTEP



# TDD mit Konfigurationsdatei

Durch eine Konfigurationsdatei können beliebig viele Programme gleichzeitig getestet werden.



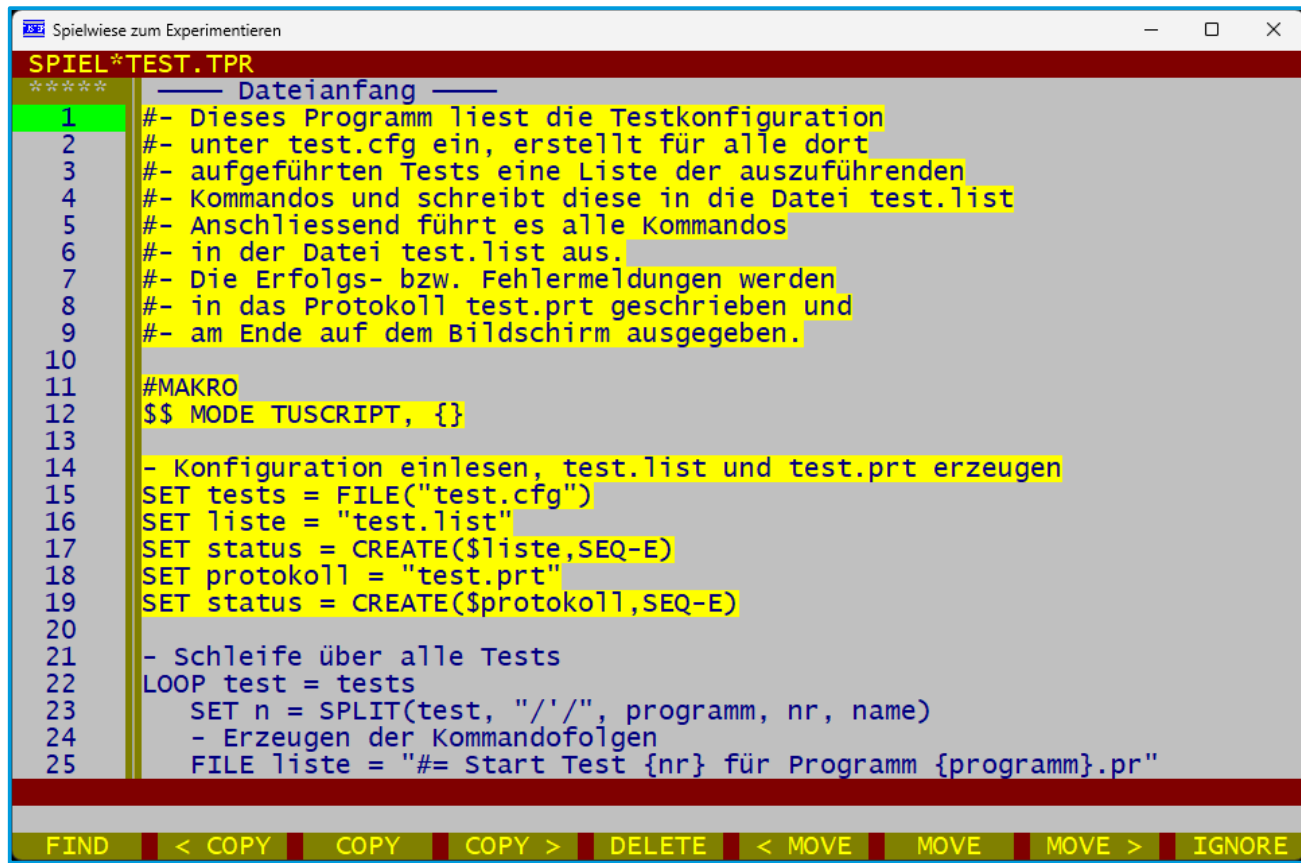
# Aufbau der Konfigurationsdatei



## Die Datei enthält

- 1) die Namen der zu testenden Programme.
- 2) die Nummern der Tests.
- 3) die Namen der Tests.

# Aufbau des Testprogramms 1



```
Spielwiese zum Experimentieren
SPIEL*TEST.TPR
*****
1  #- Dieses Programm liest die Testkonfiguration
2  #- unter test.cfg ein, erstellt für alle dort
3  #- aufgeführten Tests eine Liste der auszuführenden
4  #- Kommandos und schreibt diese in die Datei test.list
5  #- Anschliessend führt es alle Kommandos
6  #- in der Datei test.list aus.
7  #- Die Erfolgs- bzw. Fehlermeldungen werden
8  #- in das Protokoll test.prt geschrieben und
9  #- am Ende auf dem Bildschirm ausgegeben.
10
11 #MAKRO
12 $$ MODE TUSCRIPT, {}
13
14 - Konfiguration einlesen, test.list und test.prt erzeugen
15 SET tests = FILE("test.cfg")
16 SET liste = "test.list"
17 SET status = CREATE($liste,SEQ-E)
18 SET protokoll = "test.prt"
19 SET status = CREATE($protokoll,SEQ-E)
20
21 - Schleife über alle Tests
22 LOOP test = tests
23     SET n = SPLIT(test, "/", " ", programm, nr, name)
24     - Erzeugen der Kommandofolgen
25     FILE liste = "#= Start Test {nr} für Programm {programm}.pr"
```

Zu Beginn wird die Konfigurationsdatei eingelesen und die Datei für die Kommandos (test.list) sowie die Protokolldatei (test.prt) erstellt.

# Aufbau des Testprogramms 2

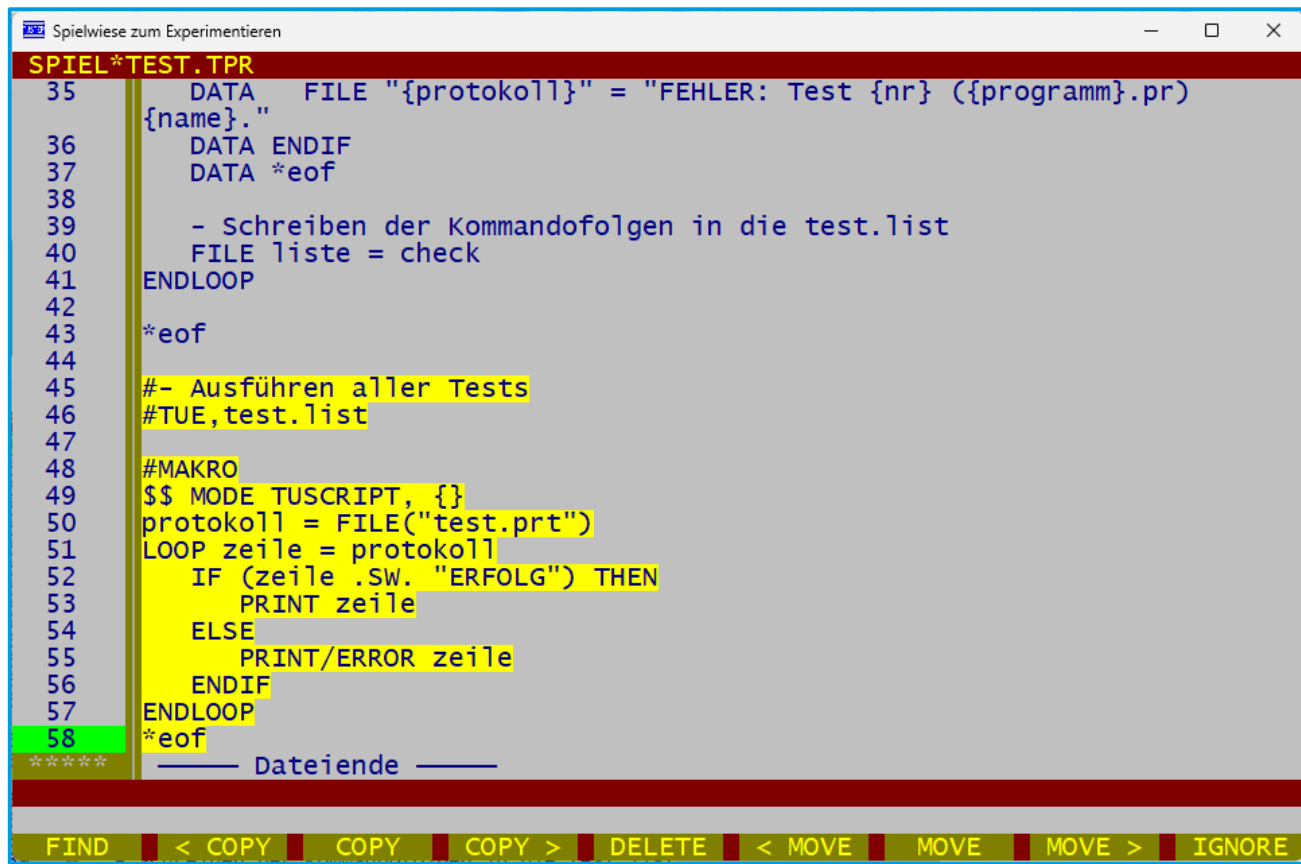
```
Spielwiese zum Experimentieren
SPIEL*TEST.TPR
20
21 - Schleife über alle Tests
22 LOOP test = tests
23   SET n = SPLIT(test, "/" /, programm, nr, name)
24   - Erzeugen der Kommandofolgen
25   FILE liste = "#= Start Test {nr} für Programm {programm}.pr"
26   FILE liste = "#DATEI,test{nr}.tr,SEQ-AT,-"
27   FILE liste = "#TUE,{programm}.pr,,test{nr}"
28   SET check = *
29   DATA #MAKRO
30   DATA $$ MODE TUSCRIPT, {}
31   DATA SET assert = COMPARE("test{nr}.tr", "test{nr}.tl")
32   DATA IF (assert == "YES") THEN
33     DATA FILE "{protokoll}" = "ERFOLG: Test {nr} ({programm}.pr)
{name}."
34   DATA ELSE
35     DATA FILE "{protokoll}" = "FEHLER: Test {nr} ({programm}.pr)
{name}."
36   DATA ENDIF
37   DATA *eof
38
39   - Schreiben der Kommandofolgen in die test.list
40   FILE liste = check
41 ENDLOOP
42
43 *eof

FIND < COPY COPY > DELETE < MOVE MOVE MOVE > IGNORE
```

In einer Schleife über alle Tests werden die Kommandos in die Liste geschrieben.

1. Erstellen der Resultatdateien (#DATEI)
2. Ausführen des Programms (#TUE)
3. Kontrollieren der Ergebnisse (#MAKRO)

# Aufbau des Testprogramms 3



```
Spielwiese zum Experimentieren
SPIEL*TEST.TPR
35 DATA FILE "{protokoll}" = "FEHLER: Test {nr} ({programm}.pr)
   {name}."
36 DATA ENDIF
37 DATA *eof
38
39 - Schreiben der Kommandofolgen in die test.list
40 FILE liste = check
41 ENDLOOP
42
43 *eof
44
45 #- Ausführen aller Tests
46 #TUE,test.list
47
48 #MAKRO
49 $$ MODE TUSCRIPT, {}
50 protokoll = FILE("test.prt")
51 LOOP zeile = protokoll
52 IF (zeile .SW. "ERFOLG") THEN
53 PRINT zeile
54 ELSE
55 PRINT/ERROR zeile
56 ENDIF
57 ENDLOOP
58 *eof
*****
Dateiende
```

Dann wird die Liste ausgeführt, d. h. alle Tests werden durchlaufen und das Protokoll erzeugt.

Zuletzt wird das Protokoll ausgelesen und auf dem Bildschirm ausgegeben.



# Protokollausgabe 1

```
Spielwiese zum Experimentieren
**** Ausgabe: 20 Sätze auf Scratch-Datei SPIEL*TEST004.TMP
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
Ende  SORTIERE  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
=====
#KOPIERE,
  quelle = test004.tmp,
  ziel = test004.tr,
  modus = +,
  loeschen = +,
Start KOPIERE  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
**** Eingabe: 20 Sätze von Scratch-Datei SPIEL*TEST004.TMP
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
**** Ausgabe: 20 Sätze auf Scratch-Datei SPIEL*TEST004.TR
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
Ende  KOPIERE  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
=====
#MAKRO
Start MAKRO  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
Ende  MAKRO  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
=====
#= Ende Test 004 für Programm sortiere.pr
#MAKRO
Start MAKRO  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
ERFOLG: Test 001 (sortiere.pr) Sortierung von A-Z.
ERFOLG: Test 002 (sortiere.pr) Sortierung von Umlauten.
ERFOLG: Test 003 (sortiere.pr) Sortierung von ß.
@@@@@@@@ FEHLER: Test 004 (sortiere.pr) Sortierung von Groß- und K.. @@@@@@@@
Ende  MAKRO  auf: CHRISTIAN-PC  am: 13.08.25  um: 16:15:24
=====
Gib Kommando >
```

Wenn ein Test fehlschlägt, sieht das Protokoll so aus.

Drei der Tests waren erfolgreich, doch einer schlug fehl.

Asche über mein Haupt.

# Protokollausgabe 2

```
Spielwiese zum Experimentieren
**** Ausgabe: 20 Sätze auf Scratch-Datei SPIEL*TEST004.TMP
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
Ende SORTIERE auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:41
=====
#KOPIERE,
  quelle = test004.tmp,
  ziel = test004.tr,
  modus = +,
  loeschen = +,
Start KOPIERE auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
**** Eingabe: 20 Sätze von Scratch-Datei SPIEL*TEST004.TMP
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
**** Ausgabe: 20 Sätze auf Scratch-Datei SPIEL*TEST004.TR
          Satzlänge: 1/4/6 Satznummern: 1.1 - 1.20
Ende KOPIERE auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
=====
#MAKRO
Start MAKRO auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
Ende MAKRO auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
=====
#= Ende Test 004 für Programm sortiere.pr
#MAKRO
Start MAKRO auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
ERFOLG: Test 001 (sortiere.pr) Sortierung von A-Z.
ERFOLG: Test 002 (sortiere.pr) Sortierung von Umlauten.
ERFOLG: Test 003 (sortiere.pr) Sortierung von ß.
ERFOLG: Test 004 (sortiere.pr) Sortierung von Groß- und Kleinschreibung.
Ende MAKRO auf: CHRISTIAN-PC am: 13.08.25 um: 16:13:42
=====
Gib Kommando >
```

Wenn alle Tests erfolgreich waren, sieht die Ausgabe ungefähr so aus.

- Alles richtig! Party!



#SORTIERE und #SVORBEREITE?

# Anwendungsfälle von #SORTIERE

- Typische Anwendungsfälle, in denen #SORTIERE gebraucht wird
  - Gegeben sind Listen von Daten, die in zufälliger Reihe vorliegen. Sie sollen alphabetisch sortiert werden. [Einfaches Sortieren]
  - Die Listen sollen benutzerdefiniert sortiert werden, z. B. nach einer Referenz oder abweichend von der alphabetischen Reihenfolge. [Komplexes Sortieren]
  - Gegeben ist ein Text, der bestimmte Datentypen enthält. Diese sollen aus dem Text extrahiert, gruppiert und nach verschiedenen Kriterien sortiert werden. [Registererstellung, (KWIC-)Indizes]

# Das Kommando #SORTIERE

- #SORTIERE hat folgende wichtige Spezifikationen:
  - 1) quelle = Name der Datei mit den zu sortierenden Daten
  - 2) ziel = Name der Datei für die sortierten Daten
  - 3) sortierfeld = sf-s (steigend), sf-f (fallend), 0 (nach Satznummern), n-m (Sortierfeld von Zeichen n bis m), n+m (Sortierfeld von Zeichen n ausgehend m Zeichen lang)
  - 4) loeschen = Daten in der ZIEL-Datei überschreiben.
  - 5) tilgen = - (unverändert ausgeben), n-m (Zeichen n-m löschen), n+m (von Zeichen n aus m Zeichen tilgen, + (nur Daten ausgeben)

# Sortierfeld, Sortierschlüssel

- #SORTIERE kann steigend oder fallen sortieren.
- #SORTIERE braucht, um die Anforderungen umsetzen zu können, mindestens ein **Sortierfeld**.
- Ein **Sortierfeld** besteht aus einem oder mehreren **Sortierschlüsseln**.
- Ein **Sortierschlüssel** ist eine Zeichenfolge, nach der die Daten sortiert werden.
- Zuerst wird nach dem ersten Sortierschlüssel sortiert, dann nach dem zweiten, dann nach dem dritten usw. Durch Angabe mehrerer Sortierschlüssel erreicht man dadurch eine Binnensortierung.

# Das Kommando #SVORBEREITE

- Sortierfelder und -schlüssel werden nicht mit #SORTIERE selbst erstellt, sondern mit #SVORBEREITE.
- #SVORBEREITE hat folgende wichtigen Spezifikationen:
  - quelle = Name der Datei mit den vorzubereitenden Daten
  - ziel = Name der Datei für die vorbereiteten Daten
  - modus = - (nur Sortierschlüssel), + (Referenzen und Sortierschlüssel)
  - loeschen = Überschreiben der Zieldatei
  - parameter = Die Regeln, nach denen die Sortierung vorbereitet wird.

# Typischer Workflow

- 1) Temporäre Dateien erstellen (#DATEI)
- 2) Daten zum Sortieren vorbereiten (#SVORBEREITE oder #RVORBEREITE, wenn es um Register geht)
- 3) Daten sortieren (#SORTIERE)
- 4) Ergebnis umkopieren (#KOPIERE) oder aufbereiten/weiterverarbeiten (#RAUFBEREITE, wenn es um Register geht).

Achtung! Nach dem Sortieren sind die Satznummern in der Datei nicht mehr aufsteigend, deswegen müssen die Sätze neu nummeriert werden (z. B. mit #KOPIERE).



# TDD am Beispiel von #SORTIERE

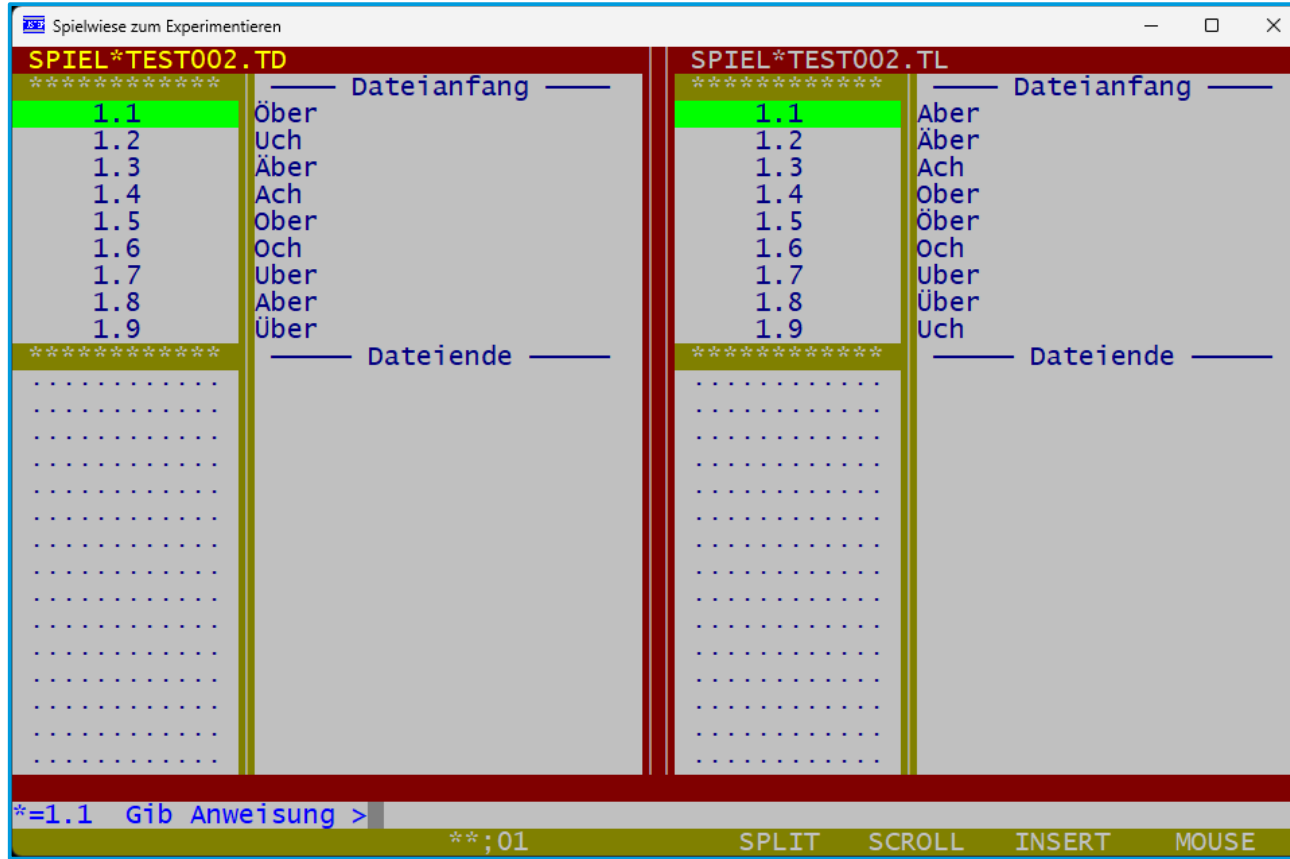
# Test für einfaches Sortieren

SPIEL*TEST001.TD		SPIEL*TEST001.TL	
*****		*****	
Dateianfang		Dateianfang	
1.1	Julius	1.1	Anton
1.2	Emil	1.2	Bertha
1.3	Richard	1.3	Cäsar
1.4	Yvonne	1.4	Dora
1.5	Gustav	1.5	Emil
1.6	Paula	1.6	Friedrich
1.7	Ludwig	1.7	Gustav
1.8	Viktor	1.8	Heinrich
1.9	Anton	1.9	Ida
1.10	Samuel	1.10	Julius
1.11	Otto	1.11	Kaufmann
1.12	Dora	1.12	Ludwig
1.13	Zacharias	1.13	Martha
1.14	Bertha	1.14	Nordpol
1.15	Theodor	1.15	Otto
1.16	Xanthippe	1.16	Paula
1.17	Ida	1.17	Quelle
1.18	Kaufmann	1.18	Richard
1.19	Friedrich	1.19	Samuel
1.20	Cäsar	1.20	Theodor
1.21	Heinrich	1.21	Ulrich
1.22	Ulrich	1.22	Viktor
1.23	Nordpol	1.23	Wilhelm
1.24	Quelle	1.24	Xanthippe
1.25	Martha	1.25	Yvonne

\*=1.1 Gib Anweisung >      \*\*:01      SPLIT      SCROLL      INSERT      MOUSE

- Einfaches Beispiel für eine Testdatei und eine Lösungsdatei
- Das Testdatum ist eine unsortierte Liste von Namen, das erwartete Ergebnis ist eine alphabetisch sortierte Liste.

# Test für das Sortieren von Umlauten



- Grundregel: Umlaute werden wie Grundbuchstaben behandelt.
- Bei ansonsten gleicher Schreibung kommt zuerst der Grundbuchstabe, dann der Umlaut

# Test für das Sortieren von $\beta$

The screenshot displays two windows from a software application titled 'Spielwiese zum Experimentieren'. The left window, 'SPIEL\*TEST003.TD', and the right window, 'SPIEL\*TEST003.TL', both show a list of items under the heading 'Dateianfang'. The right window's list is sorted alphabetically. Both windows have a 'Dateiende' section below the list and a command prompt at the bottom.

Item	Left Window (TD)	Right Window (TL)
1.1	Mast	Maske
1.2	Masse	Maß
1.3	Maße	Mass
1.4	Maßanzug	Massa
1.5	Mass	Maßanzug
1.6	Matador	Maße
1.7	Massa	Masse
1.8	Maß	Mast
1.9	Maske	Matador

Command prompt: `*=1.1 Gib Anweisung >`

Footer: `**;01`    SPLIT    SCROLL    INSERT    MOUSE

- Grundregel:  $\beta$  wird wie ss behandelt.
- Bei ansonsten gleicher Schreibung kommt zuerst  $\beta$  dann ss.

# Test für das Sortieren von Klein- und Großbuchstaben

SPIEL*TEST004.TD		SPIEL*TEST004.TL	
	_____ Dateianfang _____		_____ Dateianfang _____
1.1	^S	1.1	acht
1.2	Acht	1.2	Acht
1.3	oft	1.3	ächt
1.4	ächt	1.4	Ächt
1.5	Zu	1.5	bar
1.6	öft	1.6	Bar
1.7	ünten	1.7	christ
1.8	ß	1.8	Christ
1.9	acht	1.9	oft
1.10	Bar	1.10	Oft
1.11	Christ	1.11	öft
1.12	Ächt	1.12	Öft
1.13	Öft	1.13	ß
1.14	Ünten	1.14	^S
1.15	christ	1.15	unten
1.16	Oft	1.16	Unten
1.17	unten	1.17	ünten
1.18	Unten	1.18	Ünten
1.19	zu	1.19	zu
1.20	bar	1.20	Zu
*****	_____ Dateiende _____	*****	_____ Dateiende _____
.....		.....	
.....		.....	
.....		.....	

\*=1.14 Gib Anweisung >      \*\*:01      SPLIT      SCROLL      INSERT      MOUSE

- Grundregel:  
Kleinbuchstabe  
kommt vor  
Großbuchstabe.

# Exkurs: Sortierreihenfolgen 1

- Sortierung ist nicht gleich Sortierung?!

Testdaten	DIN 5007-1	DIN 5007-2	Österr. Sort.
1    Goldmann	1 Göbel	1 Göbel	1 Goethe
2    Götz	2 Goethe	2 Goethe	2 Goldmann
3    Göthe	3 Goldmann	3 Göthe	3 Göbel
4    Goethe	4 Göthe	4 Götz	4 Göthe
5    Göbel	5 Götz	5 Goldmann	5 Götz

- Je nach Kontext kann eine unterschiedliche Sortierung nötig sein:
- DIN 5007-1 z. B. regelt Lexika, DIN 5007-2 hingegen Telefonbücher.

# Exkurs: Sortierreihenfolgen 2

- Sortierreihenfolgen müssen oft mehrere Aspekte einer Sprache berücksichtigen.
- Beispiel Neuhochdeutsch: Umlaute, ß, Groß- und Kleinschreibung

DIN 5007-1	DIN 5007-2	Österreich
ä wie a	ä wie ae	ä nach az
ö wie o	ö wie oe	ö nach oz
ü wie u	ü wie ue	ü nach uz
ß wie ss	ß wie ss	ß nach ss

- Tests verschaffen Klarheit darüber, wie tatsächlich sortiert werden soll.

# Aufbau des Sortierprogramms 1

```
Spielwiese zum Experimentieren
SPIEL*SORTIERE.PR
1  #- Einrichten einer temporären Datei
2  #DATEI,
3  name = ?1.tmp,
4  typ = seq-at,
5  fragen = -
6
7  #- Vorbereiten der Daten für die Sortierung
8  #SVORBEREITE,
9  quelle = ?1.td,
10 ziel = ?1.tmp,
11 modus = -,
12 loeschen = +,
13 parameter = *
14     - Es werden drei Sortierschlüssel eingerichtet
15     - Jeder Sortierschlüssel ist 20 Zeichen lang
16 SSL      20 20 20
17     - Austauschen für 1. Sortierschlüssel
18     - Umlaute/ß wie Grundbuchstaben
19 XS1      |ä|a|ö|o|ü|u|ß|ss|
20     - Austauschen für 2. Sortierschlüssel
21     - Grundbuchstabe vor Umlaut
22 XS2      |ä|az|ö|oz|ü|uz|
23     - Austauschen für 3. Sortierschlüssel
24     - Kleinbuchstabe vor Großbuchstabe
25 XS3      |{\a}|1|{\A}|2|
26 *eof
27
28
29
FIND  < COPY  COPY  COPY >  DELETE  < MOVE  MOVE  MOVE >  IGNORE
```

- Erst wird eine temporäre Datei für das Vorbereiten angelegt.
- Dann folgt das Vorbereiten selbst mithilfe von Parametern.



# Aufbau des Sortierprogramms 1

- Parameter

SSL      20 20 20

XS1      |ä|a|ö|o|ü|u|ß|ss|

XS2      |ä|az|ö|oz|ü|uz|

XS3      |{\a}|1|{\A}|2|

- Erklärung

SSL: Es werden drei Sortierschlüssel von je 20 Zeichen Länge angelegt.

Xsn: In jedem Sortierschlüssel werden Zeichenfolgen ersetzt, um die Sortierregeln zu erfüllen.

# Aufbau des Sortierprogramms 2

```
Spielwiese zum Experimentieren
SPIEL*SORTIERE.PR
19      - Umlaute/ß wie Grundbuchstaben
20      XS1      |ä|a|ö|o|ü|u|ß|ss|
22      - Austauschen für 2. Sortierschlüssel
23      - Grundbuchstabe vor Umlaut
24      XS2      |ä|az|ö|oz|ü|uz|
26      - Austauschen für 3. Sortierschlüssel
27      - Kleinbuchstabe vor Großbuchstabe
28      XS3      |{\a}|1|{\A}|2|
29      *eof
30
31      #- Vorbereitete Daten werden sortiert.
32      #- Die Satznummern in der Zieldatei sind nicht mehr aufsteigend.
33      #SORTIERE,
34      quelle = ?1.tmp,
35      ziel = ?1.tmp,
36      sortierfeld = 1-60,
37      loeschen = +,
38      tilgen = 1-60
39
40      #- Umkopieren, um die Satznummern neu zuvergeben.
41      #KOPIERE,
42      quelle = ?1.tmp,
43      ziel = ?1.tr,
44      modus = +,
45      loeschen = +,
46      *****
47      _____ Dateiende _____

FIND  < COPY  COPY  COPY >  DELETE  < MOVE  MOVE  MOVE >  IGNORE
```

- Dann wird sortiert anhand der Zeichen 1-60 (3 x 20 SSL) sortiert und die Sortierschlüssel wieder getilgt.
- Zuletzt wird umkopiert, damit die Satznummern wieder stimmen.