KERNFORSCHUNGSANLAGE JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik

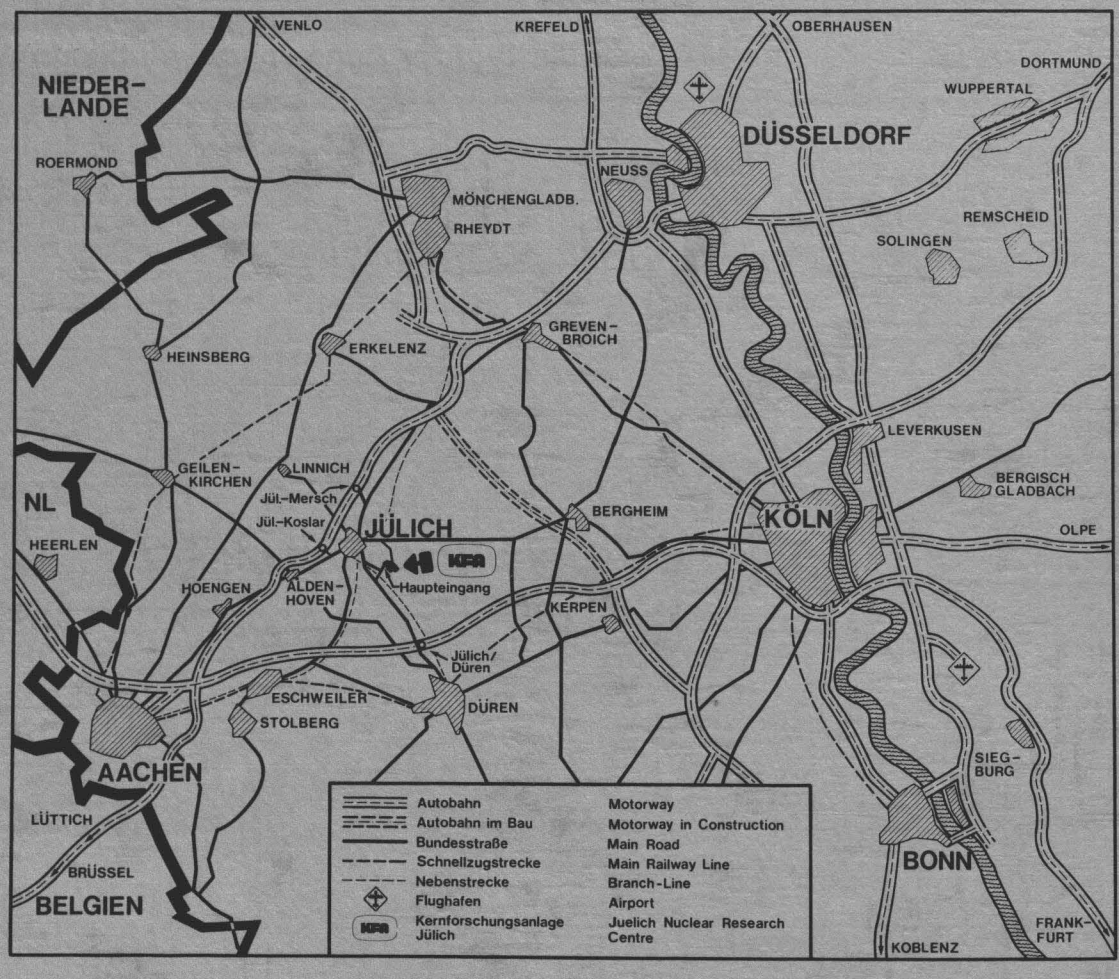
SFORTRAN

Strukturierte Programmierung mit Hilfe eines Preprocessors

von

K. Wolkersdorfer

Jül - Spez - 33
Februar 1979
ISSN 0343-7639



Als Manuskript gedruckt

Spezielle Berichte der Kernforschungsanlage Jülich – Nr. 33

Zentralinstitut für Angewandte Mathematik Jül - Spez - 33

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH,
Jülich, Bundesrepublik Deutschland

SFORTRAN

Strukturierte Programmierung mit Hilfe eines Preprocessors

von

K. Wolkersdorfer

Abstract

Es wird ein Preprocessor für FORTRAN-Programme vorgestellt. Dieser Preprocessor besizt standardmäßig einen Sprachumfang, in dem die Programmiersprache FORTRAN vollständig enthalten ist. Bei der Wahl der zusätzlichen Statements wurden weitgehend Elemente der strukturierten Programmierung berücksichtigt. Ebenso wurden viele Statements der ANSI FORTRAN 77 - Norm aufgenommen, um bereits jetzt Programme weitgehend an diese neue Norm anpassen zu können.

Daneben bietet SFORTRAN weitere Möglichkeiten eines Preprocessors, wie z.B. das Einfügen von Source-Code an bestimmte Stellen im Programm (INCLUDE) sowie die Ausgabe des Quellprogrammes in strukurierter Form.

Da SFORTRAN auch als Macro-Processor konzipiert ist, können Benutzer-eigene Macros definiert werden.

SFORTRAN kann auf jedem Computer-System eingesetzt werden, da der Preprocessor vollständig in ANSI FORTRAN 66 (FORTRAN IV) geschrieben ist.

0. Inhaltsverzeichnis

1. Einführung

In der modernen Software-Entwicklung liegt das Hauptgewicht vor allem auf zwei Aspekten, wenn man fordert, daß geschriebene Software über den Programmierer und das verwendete Computer-System hinaus zugänglich gemacht werden soll:

- Strukturierte Programmierung
- Portabilität

Natürlich trifft die zweite Forderung speziell für den wissenschaftlich- technischen Bereich zu, wo ein Austausch von Verfahren und Programmen zu den Selbstverständlichkeiten gehört.

In den letzten Jahren war aber nun zu beobachten, daß die strukturierte Programmierung kaum Eingang in diesen Bereich gefunden hat, während im kommerziellen Bereich verstärkt davon Gebrauch gemacht worden ist.

In einer Großforschungseinrichtung wie der Kernforschungsanlage Jülich (KFA) lag es also nahe, sich näher mit diesem Thema auseinanderzusetzen. Das Zentralinstitut für Angewandte Mathematik (ZAM), das eine zentrale Stellung in der KFA in Bezug auf Datenverarbeitung besitzt, beschäftigt sich deshalb im Rahmen seines Forschungs- und Entwicklungsprogrammes auch mit der Verbesserung von Programmiertechniken. Ein besonderes Gewicht besitzt dabei die Programmiersprache FORTRAN, da zwei Drittel der in der KFA gerechneten Programme diese Sprache verwenden. In diesem Rahmen entstand vorliegende Arbeit.

2. Portable Software

Wie in [6] soll hier unter "portabler Software" ein Programm-Paket verstanden werden, das (bis auf Kontrollkarten) nicht verändert zu werden braucht, um auf verschiedenen Computer-Systemen eingesetzt werden zu können. Ein solches Programm-Paket besteht typischerweise aus Anwendungsprogrammen mathematischer oder physikalisch-technischer Natur, die in einer höheren Programmiersprache geschrieben sind.

Bereits in den frühen sechziger Jahren zeichnete sich ab, daß im wissenschaftlich-technischen Bereich trotz aller Mängel die Programmiersprache FORTRAN diese Rolle würde übernehmen müssen, zum Teil einfach deswegen, weil sie schon damals die mit Abstand am weitesten verbreitete (höhere) Sprache war.
Einen Hauptbeitrag dazu lieferte auch das "American Standard Committee" für FORTRAN, das sich 1962 konstituierte und das schließlich 1966 die erste Norm für eine Programmiersprache (nach der Backus-Notation für ALGOL 60) herausbrachte: ANSI FORTRAN, 1966. Diese Norm wurde später verfeinert (USASI FORTRAN X3, 1969 und ANSI FORTRAN X3J3, 1971) und fand schließlich 1972 Aufnahme bei der "International Standard Organisation" (ISO). Genaueres findet sich bei Muxworthy [6].

Bereits zu dieser Zeit entwickelte man Übersetzerprogramme, die verschiedene FORTRAN-Dialekte ineinander umsetzten, und es zeigte sich, daß ein solcher Übersetzer (source-to-source preprocessor) auch lebensfähig war, wenn er selbst in (Standard-) FORTRAN geschrieben war.

Als dann in den frühen siebziger Jahren die strukturierte Programmierung Eingang in die Programmentwicklung fand, wurden solche Preprocessoren verstärkt als Mittel zum Programmentwurf vor allem in der kommerziellen Ebene eingesetzt.

3. Strukturierte Programmierung in FORTRAN

Der Name "strukturierte Programmierung" tauchte erstmals 1968 als Titel eines Arbeitspapiers von E.W. Dijkstra für die Nato-Konferenz über "Software Engineering Technics" auf.
Der Grundgedanke, der dahinter steckt, ist der, daß 'aus der statischen Niederschrift des Programmes der dynamische Ablauf erkennbar werden soll' [9].

Eines der Mittel, diesen Effekt zu erreichen, sind die sogenannten "closed control structures", von denen die wichtigsten drei die Sequenz, die Selektion (IF-THEN-ELSE) und die Iteration (DO-WHILE) sind.
Dabei ist anzumerken, daß ein Sprungbefehl (GO TO) hier nicht nur überflüssig ist, sondern auch gefährlich, wie die Diskussion um Dijkstra's Provokation "GO TO statement considered harmful" [4] zeigte.

Obwohl die obigen drei Kontrollstrukturen bereits in ALGOL 60 realisiert waren, zeigten Böhm/Jacopini in [1] erst 1966, daß damit jeder programmierbare ("berechenbare") Algorithmus formuliert werden kann.

Daneben erwies es sich als nützlich, die Mehrfach-Auswahl (SELECT-CASE) als Grundstruktur aufzunehmen, um einen Algorithmus möglichst kurz und effizient zu formulieren.

In ANSI FORTRAN. 1966 fand die strukturierte Programmierung jedoch überhaupt keinen Niederschlag. Wie sollte sie auch. hätten die Designer dieser Programiersprache doch schwerlich angenommen. daß sie sich zur Hauptsprache für portable Software entwickeln werde. Im Gegenteil. manche FORTRAN Programme waren und sind nur schwer zu lesen und zu durchschauen. was vielfach an den vielen Statement-Labels. an EQUIVALENCE und an falscher Parameter-Übergabe liegt.

Natürlich wurde bald nach der Diskussion um die strukturierte Programmierung versucht. FORTRAN zu erweitern und die fehlenden "closed control structures" aufzunehmen. Dazu mußte aber gewartet werden. bis eine neue Norm geschaffen wurde. die überdies weitgehend kompatibel mit der alten Norm sein sollte. und das nahm einige Zeit in Anspruch.

1976 wurde dann ein neues "Draft Proposed Standard" für FORTRAN vorgestellt. das einige wichtige Verbesserungen (z.B. CHARACTER Datentyp) enthielt. aber noch keine Kontrollstrukturen. 1978 wurde dann endlich die neue Norm mit einem verbesserten Block-IF Statement von der ISO angenommen:
ANSI FORTRAN X3J3. 1977 [2]. [5].

In der Zwischenzeit - und d.h. auch in der Gegenwart, da noch kein FORTRAN 77 Compiler in Sicht ist - blieb also jemand, der auf Portabilität seines Programmes und auf strukturierte Programmierung Wert legte, nichts anderes übrig, als einen der vielen Preprocessoren zu benutzen, die das ermöglichen konnten.

Diese Preprocessoren waren in den siebziger Jahren entstanden und im Anhang zu [6] sind 67 Processoren genannt, von denen hier SHELTRAN (G. A. Croes, Shell International Petroleum, London) und STRUFORT (O. Murro, C.S.A.T.A., Bari) angeführt werden sollen, da sie einige Bedeutung für den europäischen Bereich besitzen.

Als nächstes muß nun die Frage beantwortet werden, welche Forderungen an einen Preprocessor den Ausschlag für das ZAM gaben, einen eigenen Preprocessor zu entwickeln.

4. Entwicklungs - Konzept für SFORTRAN

Im folgenden werden die wichtigsten Forderungen genannt, die bei der Konzipierung von SFORTRAN die entscheidende Rolle spielten:

1. Man wollte ein Hilfsmittel haben, das es dem Benutzer erlaubte, bereits jetzt wesentliche Teile der neuen FORTRAN 77 Norm zu übernehmen.

2. Solche Programme sollten dann ohne Änderung auf späteren FORTRAN 77 Compilern übersetzt werden können.

3. Alle wichtigen "closed control structures" sollten aufgenommen werden und ohne Konflikte zu bestehenden Statements implementiert werden.

4. Das Hilfsmittel sollte portabel sein, d.h. Programmiersprache mußte ANSI FORTRAN 66 sein.

5. Ein INCLUDE-Statement sollte vorhanden sein, um eine wiederkehrende Sequenz von Source-Code mehrmals in das Quellprogramm einfügen zu können. (z.B. identische COMMON-Blöcke in allen Unterprogrammen)

6. Benutzer-Macros sollten möglich sein.

7. Der Zeitaufwand für die Erstellung sollte nicht größer als ein Mannjahr sein.

Nicht enthalten im Forderungskatalog ist z.B., <u>alle</u> FORTRAN 77 Statements aufzunehmen (z.B. CHARACTER-Datentyp), da das prinzipielle Schwierigkeiten bereiten würde.

Auf der Suche nach einem geeigneten Preprocessor wurde in MORTRAN [3] ein Macro-Preprocessor gefunden, der immerhin die Punkte 3,4,6 und 7 erfüllte. Da überdies noch das Quellprogramm von MORTRAN zur Verfügung stand, war es naheliegend, den Versuch zu unternehmen, MORTRAN aufzubohren und die restlichen Punkte hinzuzufügen.

Dieser Versuch scheiterte jedoch, da MORTRAN selbst als Bootstrap-Preprocessor aufgebaut ist und sich über mehrere Level aus wenigen Macros generiert, die für die zusätzlichen Forderungen nicht erweiterbar schienen.

Trotzdem gab dem Verfasser die Beschäftigung mit MORTRAN so viele Anregungen, daß er beschloß, einen MORTRAN-ähnlichen Preprocessor in MORTRAN zu schreiben, der alle Forderungen erfüllen konnte. Das führte 1976/77 zu der 1. und 2. Version von SFORTRAN, wobei die 2. Version selbst in SFORTRAN geschrieben war.

Als dann Mitte 1978 bekannt war, wie die neuen Kontrollstrukturen der FORTRAN 77 Norm aussehen würden, wurde die vorliegende 3. Version von SFORTRAN hergestellt, die völlig in sich selbst geschrieben war. Das war möglich, weil die Sprachdefinition von SFORTRAN Version 3 allein als System von Macros vorhanden ist und mit Hilfe von SFORTRAN Version 2 ein lauffähiges ANSI FORTRAN 66 Äquivalent hergestellt werden konnte.

5. SFORTRAN - Statements

Alle SFORTRAN Statements werden vom Preprocessor in Standard FORTRAN Statements übertragen. Damit kann das erzeugte FORTRAN Programm von jedem FORTRAN Compiler übersetzt werden.

Es ist daher zunächst unerheblich, ob man sich auf die neuen ANSI FORTRAN 77 Statements beschränkt oder ob zusätzliche SFORTRAN Statements verwendet werden. Nur wer das ursprüngliche Programm von einem späteren FORTRAN 77 Compiler übersetzen lassen will, muß sich auf die Statements in 5.1 beschränken.

Eine ausführliche Beschreibung, welcher Standard FORTRAN Code aus den SFORTRAN Statements erzeugt wird, findet sich im Benutzer-Handbuch für SFORTRAN [10].

5.1 ANSI FORTRAN 77 - Statements

5.1.1 Parameter - Statement

Mit Hilfe dieses Statements ist es möglich, arithmetischen Konstanten beliebigen Typs einen Namen zu geben.

```
PARAMETER ( n1=k1 [ ,n2=k2 .... ] )
```

wobei n1,n2 beliebige FORTRAN-Namen (1. Zeichen: Buchstabe)
und k1,k2 arithmetische Konstanten beliebigen Typs sind

Im Programm wird dann immer Name n1,n2 durch die aktuellen Konstanten k1,k2 ersetzt. Damit lassen sich bereits im Deklarationsteil Namen als Indexgrenzen angeben.

Beispiel:

```
PARAMETER ( N=10, M=20, PI=3.14159 )
DIMENSION A(N), B(N,M)
```

Zu beachten ist dabei, daß hier der Preprocessor eine Substitution auf der Ebene des Quellprogrammes ausführt. Es handelt sich also hier nicht um eine "dynamische Feld-Deklaration", sondern um eine Ersetzung der Namen durch Konstanten vor der Compilierung des Programmes.

5.1.2 IF - Statements

Es gibt 3 Arten von IF - Blöcken:

```
1. IF ( logical expression ) THEN
      .
      .     true-block
      .
   END IF
```

```
2. IF ( logical expression ) THEN
      .
      .    true-block
      .
   ELSE
      .
      .    false-block
      .
   END IF
```

```
3. IF      ( logical expression ) THEN
      .
      .     true-block
      .
   ELSE IF ( logical expression ) THEN
      .
      .     else-block 1
      .
   ELSE IF ( logical expression ) THEN
      .
      .     else-block 2
      .
   ELSE
      .
      .     else-block n
      .
   END IF
```

wobei logical expression ein logischer Ausdruck ist und die true-, else- bzw. false-Blöcke (S)FORTRAN-Statements enthalten.

Diese Block-IF Statements können natürlich wiederum Block-IF Statements enthalten (Schachtelung). Ebenso können diese Statements mit anderen SFORTRAN-Statements (z.B. Schleifen) gemixt werden, ohne daß eine Konfusion eintritt.

5.1.3 DO - Schleife

Bei der folgenden DO-Schleife ist zu beachten, daß jetzt endlich bei Anfangs-, Endwert und Schrittweite arithmetische Ausdrücke statt INTEGER-Variablen und -Konstanten angegeben werden können.
Ebenso ist es möglich, über die INTEGER-Laufvariable hinaus, andere Variablen-Typen zu verwenden.

```
        DO label [,] var= expr1, expr2 [, expr3 ]
        .
        .
        .
label ...
```

Es bleibt anzumerken, daß sich - bis auf die angegebenen Erweiterungen - die syntaktische Form der DO-Loop nicht verändert hat, wenn man davon absieht, daß neuerdings nach dem label ein Komma gesetzt werden kann. Damit kann nämlich die DO-Loop schneller erkannt werden.

5.2 Weitere Statements zur strukturierten Programmierung

Die folgenden Statements sind nicht Bestandteil der ANSI FORTRAN 77 Norm. Sie sind jedoch vielfach von Nutzen, wenn Wert auf strukturierte Programmierung gelegt werden soll, weil mit ihrer Hilfe leicht eine weitere Reduzierung der Statement Labels möglich ist. Außerdem beabsichtigt das "Standard Committee" bei einer eventuellen Erweiterung der FORTRAN Norm (Arbeitstitel: FORTRAN 82) solche Statements aufzunehmen. Die Syntax der Statements entspricht den Vorschlägen für diese neue Norm.

1. FOR var= expr1, expr2 [, expr3]
   .
   .
   .
   END DO

Diese inkrementelle Schleife entspricht der neuen FORTRAN 77 DO-Loop, ohne daß ein label erforderlich ist.

2. DO WHILE (logical expression)
   .
   . true-block
   .
   END DO

3. DO UNTIL (logical expression)
   .
   . false-block
   .
   END DO

Bei den beiden letzten Schleifen ist ebenfalls kein Statement-Label erforderlich. Zu beachten ist, daß bei diesen Schleifen die Prüfung auf Verlassen der Schleife am Anfang vollzogen wird.

4. DO

.

. loop-block

.

END DO [WHILE (logical expression) | UNTIL (...)]

Im Gegensatz zu 2. und 3. wird hier die Prüfung auf Verlassen der Schleife am Ende gemacht.

5.3 Sonstiges

Neben den oben genannten Statements bietet SFORTRAN noch folgende Möglichkeiten:

- Mehrfach-Zuweisungen

  Beispiel: X= Y= Z= 3.1

- Statement-Begrenzer

  Damit können mehrere Statements pro Zeile eingegeben werden, ohne dabei jedoch von den in FORTRAN üblichen Konventionen abzuweichen.

  Beispiel: I=0; J=1; K=2;

- Alphanumerische Statement-Labels

  Obwohl durch strukturierte Programmierung in SFORTRAN kaum Labels nötig sein werden, können zusätzlich zu den numerischen FORTRAN-Labels auch alphanumerische Labels verwendet werden. Diese müssen dann grundsätzlich in Doppelpunkte (:) eingeschlossen werden, dürfen aus beliebig vielen Buchstaben und Ziffern bestehen und können überall dort vorkommen, wo FORTRAN-Labels erlaubt sind.

- Abkürzung für formatierte Ein/Ausgabe

  Manchmal ist es mühsam, für einen READ/WRITE - Befehl zwei Statements (und einen Label) verwenden zu müssen. Deshalb wurden dafür Abkürzungen eingeführt.

  Beispiel: Statt

```
      WRITE (6,10) X
10    FORMAT (' X= ',I2)
```

  kann jetzt kürzer geschrieben werden:

```
      PUT /X/ (' X= ',I2)
```

Einen genauere Beschreibung dieser Möglichkeiten findet sich im Benutzer-Handbuch [10].

5.4 Programm - Beispiele

Die folgenden Beispiel-Programme sollen einen Oberblick über das Aussehen eines SFORTRAN-Programmes vermitteln. Die Beispiele wurden so gewählt, daß die verwendeten Algorithmen möglichst bekannt sein sollten. Außerdem wurde Wert auf Verwendung von "closed control structures" gelegt.

1. Primzahlen-Berechnung (bis zu einer gegebenen Zahl N):

Die verwendete Methode ist das "Sieb des Eratosthenes":
In ein vorgegebenes Feld F werden diejenigen Feldelemente mit 1 besetzt, deren Indizes identisch mit den Vielfachen von bereits gefundenen Primzahlen sind. Am Ende werden dann die Primzahlen aus den Indizes erhalten, deren zugehöriges Feldelement nicht mit 1 besetzt wurde:

```
PARAMETER (N=10000, NWURZ=100)
INTEGER F(N)
FOR I=3, N, 2
    F(I)= 0
END DO
FOR I=3, NWURZ, 2
   I2= I+I
   K = I*I
   DO UNTIL (K .GT. N)
       F(K)= 1
       K= K + I2
   END DO
END DO
PUT // ('       1'/'       2')
FOR I=3, N, 2
   IF (F(I) .EQ. 0) PUT /I/ (I7)
END DO
STOP
END
```

2. Acht-Damen-Problem:

Auf einem Schach-Brett sollen acht Damen so angeordnet werden, daß keine Dame eine andere bedroht. Gesucht sind alle möglichen Lösungen:

```
      PARAMETER (N=8)
      INTEGER POS(N)
      FOR I= 1, N
          POS(I)= 0
      END DO
      K= 1;   LC= 0
:LAB: DO WHILE (K .GT. 0)
         J=POS(K) + 1
         IF (J .GT. N) THEN
            K=1
         ELSE
            I=1
            DO UNTIL (I .EQ. K)
               IH=IABS (J - POS(I))
               IF       (IH .NE. 0  .AND.  IH .NE. K-I) THEN
                        I= I+1
               ELSE IF (J .LT. N) THEN
                        I= 1;    J= J+1
               ELSE
                        K= K-1
                        GOTO :LAB:
               END IF
            END DO
            POS(K)= J
            IF (K .EQ. N) THEN
               LC= LC+1;    PUT / (POS(L),L=1,N) / (N(13))
               K= K-1
               GOTO :LAB:
            END IF
            K= K+1;     POS(K)= 0
         END IF
      END DO
      PUT / LC / (17,' LOESUNGEN')
      STOP
      END
```

6. Preprocessor - Eigenschaften

Da SFORTRAN als Preprocessor realisiert ist, können einige zusätzliche Vorteile verwendet werden, die ein Compiler im allgemeinen nicht zur Verfügung hat:

6.1 Einfügen von Source - Code

Oft ist es bei einem FORTRAN-Programm lästig, umfangreiche Deklarationen in jedem Unterprogramm zu wiederholen. Das führt regelmäßig dazu, daß nur die notwendigsten Deklarationen pro Unterprogramm vorgenommen werden und dadurch Fehler wie falsche Reihenfolge bei COMMON-Blöcken oder Weglassen von notwendigen Variablen- Spezifikationen entstehen können.

Deshalb wäre es sinnvoll, die Deklarationen einmal hinzuschreiben, auf einem Dataset abzulegen und nach Bedarf zum Programm hinzuzuladen. Diese Möglichkeit bietet SFORTRAN:

Seien folgende Deklarationen auf einem Dataset abgelegt:

```
PARAMETER (N=10,K=200)
COMMON A(N,N),B(K)
INTEGER A,B,M(N),J
DATA M,J / N*0, 1 /
```

Sei für diesen Dataset weiterhin eine FORTRAN reference number (wie z.B. die 5 in READ(5,...)) definiert, sagen wir 20. Dann kann ein Programm z.B. so aussehen:

```
   %INCLUDE 20
         .
         .      (Hauptprogramm)
         .
         END
         SUBROUTINE SUB (...)
   %INCLUDE 20
         .
         .      (Unterprogramm 1)
         .
         END
         FUNCTION FUNC (...)
   %INCLUDE 20
         .
         .      (Unterprogramm 2)
         .
         END
```

Die %INCLUDE-Karte ist dabei eine Preprocessor-Steuerkarte und bewirkt das Dazuladen der Deklarationen von dem Dataset mit der reference number 20. Damit wird sichergestellt, daß in allen Unterprogrammen die gleichen Deklarationen vorhanden sind.

6.2 Strukturiertes Auflisten des Quellprogrammes

Mit Hilfe dieser Möglichkeit können unstrukturiert eingegebene Programme übersichtlich aufgelistet werden. Dabei wird bei Statements mit Blockstruktur je nach der Tiefe der Schachtelung diese auf der Liste entsprechend nach rechts eingerückt ausgegeben.

Beispiel:

Eingegebenes Programm:

```
.
.
.
DO WHILE (...)
IF (...) THEN
.
.
.
ELSE
DO UNTIL (...)
.
.
.
END DO
END IF
END DO
.
.
.
```

Dieses unstrukturiert eingegebene Programm wird durch den Preprocessor auf folgende Weise aufgelistet:

Aufgelistetes Programm:

```
0       .
0       .
0       .
0       DO WHILE (...)
1          IF (...) THEN
2             .
2             .
2             .
2             ELSE
2             DO UNTIL (...)
3                .
3                .
3                .
3                END DO
2             END IF
1          END DO
0       .
0       .
0       .
```

Dabei geben die Zahlen an der linken Seite die Schachtelungstiefe bei den strukturierten Statements an. Zusätzlich wird dann noch eine Statement-Numerierung ausgegeben, die sich auf das eingegebene Quellprogramm bezieht.

6.3 Sonstiges

Im Benutzer-Handbuch für SFORTRAN [10] findet sich eine ausführliche Beschreibung aller möglichen SFORTRAN Steuerkarten. Hervorzuheben wäre hier noch folgendes:

Da ein Preprocessor kein Compiler ist, können einige Programmfehler erst endeckt werden, wenn der FORTRAN-Compiler das erzeugte Programm übersetzten will.
Falls das der Fall ist, muß die vom FORTRAN-Compiler angegebene Zeilennummer nicht mit der Zeilennummer des SFORTRAN-Programmes identisch sein.

Nehmen wir an, jemand vergißt bei einem PUT-Statement die letzte schließende Klammer:

```
   19         .
   20         .
   21         PUT / X / (' X= '.F8.4
   22         .
   23         .
```

Dann erzeugt erst der FORTRAN-Compiler eine Fehlermeldung, z.B.

```
   34         .
   35         .
   36         WRITE (6,10001) X
   37 10001   FORMAT (' X= '.F8.4
   38         .
   39         .

  **** ERROR IN LINE 37: NOT ENOUGH RIGHT PARENTHESIS
```

Der eigentliche Fehler tritt jedoch in Zeile 21 des SFORTRAN-Programmes auf.

Es gibt deshalb standardmäßig die Möglichkeit, die ursprünglichen Zeilennummern per Kommentarkarten in das erzeugte FORTRAN-Programm einzustreuen, wie in dem Beispiel:

```
   33         .
   34         .
   35 C                                                    21
   36         WRITE (6,10001) X
   37 10001   FORMAT (' X= '.F8.4
   38 C                                                    22
   39         .
   40         .
```

Damit ist eine einfache Rückverfolgung der FORTRAN Compiler-Fehlermeldungen auf den SFORTRAN Quelltext möglich.

7. Macro - Konzept

Im Grundsatz wurde das Macro-Konzept ebenfalls von MORTRAN [3] übernommen. Hinzugefügt wurde jedoch eine flexible Macro-Parameter-Gestaltung und eine umfangreichere FORTRAN Statement-Label-Verwaltung. Eine ausführliche Beschreibung dieser Möglichkeiten findet sich in [11]. Hier soll nur kurz die Arbeitsweise eines Macro-Preprocessors dargestellt werden.

7.1 Preprocessor - Initialisierung

Ein Macro-Preprocessor - für welche Sprache auch immer - besteht grundsätzlich aus drei Teilen:

Schritt 1: Lesen des Source-Textes

Schritt 2: Veränderung des Source-Textes durch Macros
(bzw. Nicht-Veränderung)

Schritt 3: Ausgabe des erzeugten Textes als Ergebnis der Veränderung von Schritt 2

Der vorliegende Macro-Preprocessor hält sich grundsätzlich an diese Einteilung, wobei zu beachten ist, daß die Abarbeitung der drei Schritte aus internen Platzgründen auf Statement-Ebene erfolgt.

Im Prinzip ist es dabei völlig gleichgültig, für welche Sprache der Preprocessor verwendet werden soll. Dies hängt allein von den verwendeten Macros ab, die außerhalb des Preprocessors auf dem Initialisierungs-File st n.

Nur diese "Grund-Macros" - die hier die Sprachdefinition von SFORTRAN bilden - bestimmen, wie Ein- und Ausgabe des Macro-Processors aussehen soll.

Wenn beispielsweise diese "Grund-Macros", die jederzeit veränderbar sind, nicht mit dem Source-Code übereinstimmen, findet keine Veränderung des Codes statt.

7.2 Benutzer - Macros

Für den Assembler-Programmierer ist es längst eine Selbstverständlichkeit, Macros zu benutzen. Ohne eine solche Hilfe wären manche Assembler-Programme überhaupt nicht mehr denkbar.

Für den Programmierer in einer höheren Programmiersprache jedoch hat sich der Umgang mit Macros noch nicht eingebürgert, obwohl diese auch hier eine große Hilfe wären, z.B. bei Programmumstellungen, bei Benutzung von Maschinen-abhängigen Befehlen oder bei Veränderung bestimmter Programmteile (wie Generierung von Zahlen mit variabler Genauigkeit).

In Kap. 7.1 wurde erwähnt, daß die Sprachdefinition von SFORTRAN allein durch Macros geschieht, die auf dem Initialisierungs-File stehen. Ebenso können vom Benutzer eigene Macros definiert werden, die als Preprocessor-Steuerkarten in das SFORTRAN-Quellprogramm eingestreut werden können. Eine genaue Beschreibung dieser Möglichkeiten findet sich in [10] und [11].

Als Beispiel für die generellen Einsatzmöglichkeiten der Benutzer-Macros sei hier eine Untersuchung am ZAM über das Verhalten von Benutzerprogrammen mit großen Datenfeldern genannt.
Untersucht werden sollte, wann und auf welche Weise die Arrays in den Programmen indiziert und referiert wurden und damit welchen Einfluß sie auf das Paging-Verhalten des Betriebssystems ausübten.
Dazu wurden mit geringem Aufwand einige Benutzer-Macros geschrieben, die die Quellprogramme der Benutzer so veränderten, daß zur Laufzeit dann eine Statistik über Anzahl und Art der Array-Indizierung erstellt werden konnte.

8. Zusammenfassung

Im ganzen kann SFORTRAN als ein Beitrag zur Verbesserung der Programmiertechnik in FORTRAN-Programmen angesehen werden. Besonderen Wert wurde dabei auf die Portabilität der Programme gelegt, ebenso wie auf eine vernünftige Auswahl von Statements zur strukturierten Programmierung.

Daneben bietet SFORTRAN alle wichtigen Vorteile eines Preprocessors, sowie die Möglichkeit, Macros zu verwenden.

Außerdem ist es mit SFORTRAN leicht möglich, bereits vor Fertigstellung der ersten FORTRAN 77 Compiler, die Programme an diese neue (recht gute) Norm anpassen zu können.

9. Literaturverzeichnis

[1] Böhm, C. and Jacopini, G. : Flow diagrams, Turing machines and languages with only two formation rules
CACM, vol. 9, pp 366-371 (1966)

[2] Brainerd, W. (Editor) : FORTRAN 77
CACM, Vol. 21, No 10 (1978)

[3] Cook, J. : MORTRAN Version 1
Stanford Linear Accelerator Center
P.O. Box 4349 Stanford, California 94305 (1973)

[4] Dijkstra, E.W. : GO TO statement considerd harmful
CACM, vol. 11, pp 147-148 (1968)

[5] Engel, F., Jr. (Chairman) : ANS FORTRAN 77 unofficial
X3J3/90.5, X3.9-1978

[6] Muxworthy, D. : The New Standard FORTRAN
SEAS Newsletter, Vol. 5, No 3 (1976)

[7] Muxworthy, D. : A Review of Program Portability and FORTRAN Conventions
European Computer Program Institut (EUROCOPI),
Ispra, Italy
Technical Papers Series, Report No. 1, 1976

[8] Phillips, C.A. (Chairman) : USA Standard FORTRAN (ANS-FORTRAN), X3.9-1966

[9] Schnupp, P. / Floyd, C. : Software
de Gruyter, 1976

[10] Wolkersdorfer, K. : Benutzer-Handbuch für SFORTRAN Ver. 3
Interner Bericht des ZAM/KFA, 12/78

[11] Wolkersdorfer, K. : Arbeitsweise und Anwendungsmöglichkeiten des Macro-Pre-Processors SFORTRAN Ver. 3
Interner Bericht des ZAM/KFA, 12/78