Ein Vergleich von FORTRAN IV und PEARL an Hand eines umfangreichen Programmes

Burkhard Menzel, Bielefeld

1. Zusammenfassung

Es wurden gleiche Programme zur Textverarbeitung in PEARL und in FORTRAN IV erstellt. Diese Anwendung von PEARL in der "klassischen" Datenverarbeitung zeigte im Vergleich zu FORTRAN IV folgende Ergebnisse:

Die Laufzeiten der FORTRAN- und PEARL-Programme waren praktisch gleich.

Der Quellcode der FORTRAN-Programme enthielt ca. 3,5 mal soviele Zeilen wie die PEARL-Programme.

Die Untersuchung zeigt, daß PEARL wegen seiner besseren Eignung zur struktuierten Programmierung auch für Probleme der "klassischen" Datenverarbeitung bei weitem besser geeignet ist als FORTRAN IV.

2. Einleitung

In der Regel werden zwei verschiedene Programmiersprachen auf der Basis verglichen, daß die einzelnen Sprachelemente gegenübergestellt werden. Erheblich seltener wird man ein und dasselbe Programm in zwei Sprachen auf der gleichen Anlage benutzen. Es soll hier gezeigt werden, was beide Sprachen bei der Lösung eines Textverarbeitungsproblems zu leisten vermögen.

3. Die Aufgabe

Die Aufgabe bestand darin, einen sog. PREPROZESSOR und weitere Hilfsprogramme für die TOP-DOWN-Entwicklung von FORTRAN- und PEARL-Programmen in FORTRAN IV zu schreiben. Die FORTRAN IV Programme sollten in Anlehnung an bereits vorhandene PEARL-Programme erstellt werden. Um das Textverarbeitungsproblem, das der PREPROZESSOR löst, genauer verstehen zu können, soll zunächst das Verfahren zur TOP-DOWN-Entwicklung von Programmen näher beschrieben werden.

3.1 Beschreibung des Programmier-Verfahrens

Das Verfahren (1), (2) dient zur TOP-DOWN-Entwicklung von Programmen und verzichtet auf graphische Entwurfs -Hilfsmittel, wie Strukturdiagramme und ähnliches.

Die Programmentwürfe werden direkt in den Rechner eingegeben und bilden gleichzeitig die vollständige Programm-Dokumentation.

Bei der Entwicklung eines Programmes hat man zu Beginn meist nur eine vage Vorstellung von der Lösung des Problems. Man kann aber schon in dieser Phase das Programm in Abschnitte teilen, wie z. B. Deklaration der Datenbasis, Vereinbarung von Prozeduren, Beschreibung des Tasking (Abb. 1). Diese Abschnitte können dann durch schrittweise Verfeinerung in Unterabschnitte aufgeteilt und dadurch genauer erklärt werden.

```
1.      PROGRAM HAUPT
2.      /*
3.
4.
5.      #&S     SPEZIFIKATIONEN
6.      #D      DEKLARATIONEN
7.      #F      FORMAT-ANWEISUNGEN
8.      #P      PROGRAMM
9.      #&M     FEHLERMELDUNGEN
10.     #&T     TECHNISCHE-DATEN
11.     #&L     LOGBUCH
12.     */
13.     END
14.     /*
15.
```

```
221.#P      PROGRAMM
222.========================
223.
224.        #PI     INITIALISIEREN
225.        #P1     MELDE BEREIT, ERFRAGE UND BILDE DATEI-NAMEN
226.        #P2     HOLE BEARBEITUNGSBEFEHLE UND BILDE
227.                ZIELDATEINAMEN
228.        #P3     EROEFFNE DATEIEN
```

```
253.
254.#P1     */
255.        WRITE(TKAN,2011)
256.        CALL DIALG
257.        /*
258.
259.#PI     */
260.        CALL DATIN
261.        ZEINR=0
262.        GRBFEL=.FALSE.
263.        ADRBUL=.FALSE.
264.        /*
265.
266.#P2
267.        #P21    HOLE BEFEHLE
268.        #P22    UNTERSCHEIDE TESTVERSION UND ENT-
269.                GUELTIGES PROGRAMM DURCH VERSCHIEDENE
270.                VERSIONSNUMMERN.
271.
```

Abb. 1 Beispiel für die Aufteilung des Programms in Abschnitte

Die TOP-DOWN-Entwicklung besteht nun darin, daß zunächst die Abschnitt-Überschriften (aus gleich erkennbarem Grund als Pseudocodes bezeichnet) in der Reihenfolge notiert werden, in der die Abschnitte selbst später vom Compiler benötigt werden. Die

einzelnen Abschnitte selbst werden dann in beliebiger, jedoch von der Logik des Entwurfsprozesses diktierter Reihenfolge weiterbehandelt:

Sie werden in Unterabschnitte aufgeteilt, von denen zunächst wieder nur die Pseudocodes (Abschnitt-Überschriften) notiert werden. Die Unterabschnitte werden wiederum in Unterabschnitte aufgeteilt usw.

Bei diesem Prozeß der schrittweisen Verfeinerung können auf jeder Ebene schon Anweisungen der Programmiersprache eingestreut werden.

Ein Programm-Entwurf ist vollständig ausgearbeitet, wenn jeder Pseudocode seine Entsprechung in einem Code-Abschnitt der Programmiersprache gefunden hat. Wie gesagt, stehen die Verfeinerungen in scheinbar beliebiger Reihenfolge. Sie werden daher mit Hilfe von Schlüsseln (Kennziffern) mit ihren zugehörigen Pseudocodes assoziiert. Unter der oben erwähnten Voraussetzung, daß die Pseudocodes in der vom Compiler benötigten Reihenfolge stehen, kann dann durch Umsortieren der Zeilen ein kompilierbares Programm erzeugt werden. Dieses Umsortieren erledigt der erwähnte PREPROZESSOR.

Die Aufgabe des PREPROZESSORS ist es dabei, hinter jedem Pseudocode die zugehörige Verfeinerung einzuordnen.

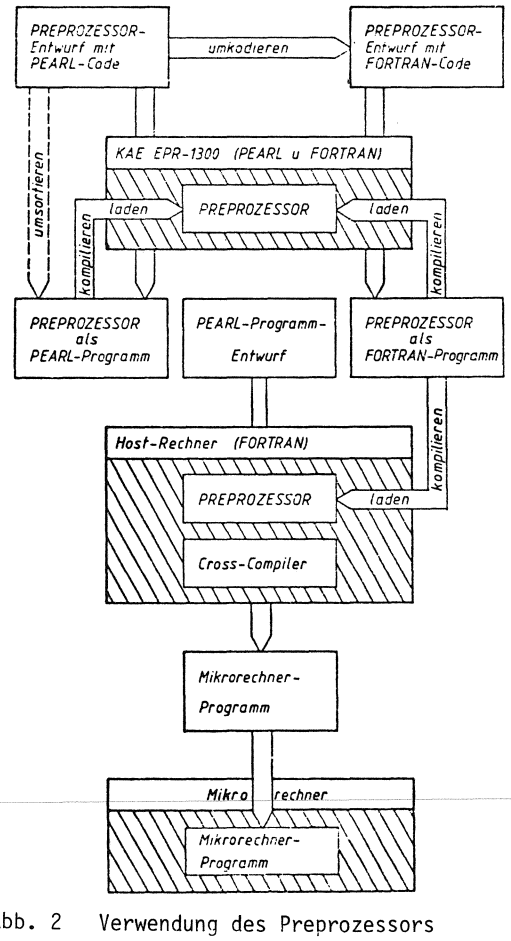

Abb. 2 Verwendung des Preprozessors

3.2 Entstehungsgeschichte der Programme

Ausgangspunkt für die Arbeit war die Aufgabe, daß PEARL-Programme für Mikroprozessoren auf einem Host-Rechner mit Hilfe des oben beschriebenen Entwurfsverfahrens erstellt werden sollten (Abb. 2). Der Cross-Compiler und alle weiteren Hilfsprogramme waren in FORTRAN IV geschrieben. Da zur Programmierung des Host-Rechners PEARL (wegen Fehlens eines Compilers) nicht verwendet werden konnte, mußte auch der PREPROZESSOR in FORTRAN IV vorliegen. Das bedeutete, daß ein bereits in PEARL vorhandener PREPROZESSOR nach FORTRAN IV umkodiert werden mußte.

Der in PEARL erstellte PREPROZESSOR existiert auf dem Rechner Krupp-Atlas Elektronik EPR 1300, auf dem auch der FORTRAN-PREPROZESSOR entworfen wurde. Der Prozessrechner EPR 1300 besitzt einen FORTRAN 77 Compiler, es wurden allderdings aus Kompatibilitätsgründen keine Sprachelemente benutzt, die über FORTRAN IV hinausgehen, wie z. B. die FORTRAN 77 CHARACTER-Verarbeitung.

Zwei PREPROZESSOR-Versionen waren zu erstellen:

1. Ein PREPROZESSOR mit Fehlerprüfung, z. B. richtige Verwendung der Kommentarzeichen, Schlüssel paarweise, ...
2. Ein PREPROZESSOR ohne diese Fehlerprüfung, der dadurch eine verkürzte Laufzeit hat.

3.3 Aufbau der FORTRAN-Programme

Der PEARL-PREPROZESSOR wurde etwas modifiziert, so daß er auch FORTRAN-Programme verarbeiten kann, damit die FORTRAN-PREPROZESSOREN mit Hilfe des beschriebenen Entwurfsverfahrens erstellt werden konnten.

Auch die Programmstrukturen wurden weitestgehend aus den PEARL-Programmen übernommen. Abweichungen gibt es nur an den Stellen, wo beide Programmiersprachen soweit verschieden sind, daß keine gleichwertigen FORTRAN-Statements vorhanden waren. Das betrifft in erster Linie die CHARACTER-Verarbeitung und die Ein-/Ausgabe, sowie Positionierungsprobleme innerhalb der Dateien.

Um auch in FORTRAN IV eine gute und übersichtliche Struktur beizubehalten, wurden ungefähr genauso viele Unterprogramme benutzt wie in PEARL. Das erforderte die Deklaration von COMMON-Blöcken, die in jedem Unterprogramm (nicht immer vollständig) wiederholt werden mußten. In der Regel wird man solche Deklarationen mit Hilfe eines Editors in die

Unterprogramme kopieren. Das Mehrfach-Deklarieren führt bei späteren Änderungen jedoch schnell zu Fehlern.

4 Vergleiche

4.1 Längenvergleiche

Der Entwurf des PEARL-PREPROZESSORS mit Fehlerprüfung ist 1504 Zeilen lang und enthält dabei 703 Zeilen mit reinem PEARL-Code. Der Rest besteht aus etwa 250 teilweise mehrzeiligen Pseudocodes (Quasi Kommentare) und Leerzeilen zur optischen Aufgliederung des Programmes (Tabelle 1). Der entsprechende FORTRAN-PREPROZESSOR weist dagegen eine Länge von 5571 Zeilen mit 2517 FORTRAN-Zeilen auf. Das bedeutet, daß der FORTRAN-PREPROZESSOR 3,7 mal so lang ist wie das gleichwertige PEARL-Programm. Setzt man die Anzahl der Zeilen mit reinem FORTRAN-Code ins Verhältnis zu den Zeilen, die PEARL-Code enthalten, so erhält man den gleichen Faktor.

	PREPROZESSOR mit Fehlerprüfung		*PREPROZESSOR ohne Fehlerprüfung*	
Texte + Statements	*PEARL*	*FORTRAN*	*PEARL*	*FORTRAN*
	1504	*5571*	*842*	*3065*
Statements	*703*	*2517*	*326*	*1151*

Tabelle 1 Tabelle zu den Längenvergleichen

Daß das keine Zufall ist, bestätigt der Vergleich der beiden PREPROZESSOR-Versionen ohne Fehlerprüfung. Der FORTRAN-Programmentwurf ist 3,6 mal so lang wie der entsprechende PEARL-Entwurf. Bei der Anzahl der Zeilen, die Statements enthalten,ergibt sich der Faktor 3,5 (FORTRAN:PEARL).

Die Länge der FORTRAN-Programme hat folgende Ursachen:

1. Mehrfach-Deklarationen der COMMON-Blöcke.

   In PEARL wurde das ganze Programm in ein einziges MODUL geschrieben. Variable und Konstanten, die über die Grenzen eines Unterprogramms bzw. einer TASK hinaus bekannt sein sollten, wurden im PROBLEM-Teil deklariert. Diese Deklarationen brauchten hier nur ein einziges Mal niedergeschrieben werden. Damit waren alle dort aufgeführten Größen im gesamten MODUL bekannt. Bekanntlich wird in FORTRAN IV jedes Unterprogramm für sich übersetzt. Daten, die von mehreren Unterprogrammen gemeinsam benutzt werden, müssen in COMMON-Blöcke geschrieben werden, wenn man auf die Übergabe mittels Parameterlisten verzichten will. Die Deklarationen dieser COMMON-Blöcke müssen grundsätzlich in jedem Unterprogramm, in dem die globalen Größen verarbeitet werden sollen, erneut niedergeschrieben oder mit einem Editor kopiert werden.
   Diese Vorschrift erhöht die Länge einiger Dateien beträchtlich. Es sind hier oft um die 100 Programmzeilen. Bei kurzen Unterprogrammen mit z. B. 20 ausführbaren Anweisungen ist die Benutzung solcher COMMON-Block-Deklarationen dafür verantwortlich, daß diese Dateien erheblich länger sind als ihre PEARL-Äquivalente.

   Würde man in PEARL ein Programm aus mehreren MODULEN aufbauen, so müßten in den Spezifikationen solche globalen Daten ebenfalls aufgeführt werden. PEARL hat jedoch den Vorteil, daß man in den Spezifikationen nur solche Größen beschreiben muß, die auch wirklich benutzt werden. In FORTRAN würde man das nur dadurch erreichen können, daß für jedes Datum ein eigener COMMON-Block geschaffen werden müßte.

   Ein Beispiel soll das veranschaulichen:

   Der PREPROZESSOR mit Fehlerprüfung arbeitet mit 19 Unterprogrammen. Nehmen wir an, daß in jedem Unterprogramm 70 Zeilen benötigt werden, um die nötigen COMMON-Blöcke zu deklarieren, (dieser Wert ist für das gegebene Beispiel sehr realistisch.) Wie sich leicht errechnen läßt, entfallen auf die Deklarationen innerhalb der Unterprogramme 1330 Zeilen.

2. Aufwendigere Zeichenverarbeitung

   In FORTRAN IV wurde eine Zeile in ein INTEGER-Feld eingelesen. Dadurch war es nicht möglich, Zeichenketten so einfach wie in PEARL zu bearbeiten. Um Zeichenketten herauszugreifen, zu vergleichen oder zu manipulieren mußten DO-Schleifen benutzt werden.

3. Fehlen moderner Sprachelemente

   Konstruktinen wie 'REPEAT-WHILE'-Schleifen oder Case-Anweisungen sind in FORTRAN IV nur durch erhöhten Programmaufwand zu realisieren.

4.2 Laufzeitvergleiche und Speicherplatzbedarf

Die Messungen wurden auf dem KAE-Prozeßrechner EPR 1300 durchgeführt. Dieser Rechner arbeitet normalerweise im Time-Sharing-Betrieb, jedoch nahm beim Aufzeichnen der Meßwerte kein anderer Benutzer am Rechnerbetrieb teil. Die PEARL- und FORTRAN-Programme waren praktisch gleichschnell (FORTRAN ca. 4 % schneller).

Aus diesem Sachverhalt kann jedoch keine pauschale Schlußfolgerung gezogen werden. Die Meßwerte be-

treffen nur die EPR 1300 und die hier beschriebene Programmieraufgabe und werden von Anlage zu Anlage anders sein. Dazu noch eine Anmerkung:

Die FORTRAN-PREPROZESSOREN benutzten zum Positionieren innerhalb der Input-Datei eine Anweisung an das Betriebssystem, die dem Find-Statement ähneln dürfte, das auf einigen Anlagen implementiert ist. (In PEARL wurde mittels der SKIP-Anweisung positioniert.)

In den ursprünglichen Versionen der FORTRAN-Programme, die diesen maschinenspezifischen Trick nicht benutzten, wurde durch entsprechende DO-Schleifen-Konstruktionen positioniert. Das hatte zur Folge, daß die Programme um bis zu 30 % langsamer waren als die jetzigen Versionen und deren PEARL-Äquivalente.

Das Beispiel zeigt, daß die Laufzeiten eines Programmes extrem stark von der Verfügbarkeit eines einzigen Statements abhängen können.

Zum Speicherplatzbedarf ist noch folgendes zu bemerken:

Die FORTRAN- und PEARL-Programme liefen hier auf einem Mehrbenutzer-Betriebssystem mit PEARL-Tasking, das sicher umfangreicher ist als ein Mehrbenutzer-Betriebssystem nur für FORTRAN. Andererseits lädt dieses für PEARL entwickelte Betriebssystem für mehrere PEARL-Benutzer das PEARL-Laufzeitsystem nur einmal, während die FORTRAN-Laufzeitroutinen normalerweise zu jedem Benutzerprogramm dazugebunden werden. Deshalb ist auf diesem speziell für PEARL entwickeltem System der Speicherverbrauch für mehrere PEARL-Benutzer kleiner als für mehrere FORTRAN-Benutzer, die ähnliche Aufgaben lösen.

Literatur:

Weiterführende Literatur zum Thema PREPROZESSOR:

{1}

Top-Down-Spezifikation, -Entwicklung und Dokumentation von PEARL-Modulen.

L. Frevert
PEARL-Rundschau Band 2, Nr. 6, S. 53 (Dez. '81)

{2}

Eine Methode zur schnelleren Entwicklung und übersichtlichen Dokumentation von PEARL-Programmen.

L. Frevert

PEARL-Rundschau Band 2, Nr. 3, S 55 (Sept. '81)

Alle übrigen Daten stammen aus einer Diplom-Arbeit an der Fachhochschule Bielefeld.

Anschrift des Autors

Burkhard Menzel
Haspelstr. 25
4800 Bielefeld 1