Dumke

Makroprogrammierung

Einführung – Übungen – Praxis

Mathematik

für

Ingenieure

VEB FACHBUCHVERLAG

LEIPZIG

Herausgegeben von

H. BIRNBAUM,

Dr.-Ing. H. GÖTZKE,

Prof. Dr.-Ing. H. KREUL,

Dr.-Ing. W. LEUPOLD,

Dr. F. MÜLLER,

Prof. Dr. P. H. MÜLLER,

Dr. H. NICKEL,

Prof. Dr. H. SACHS

MAKROPROGRAMMIERUNG

Einführung – Übungen – Praxis

Von Dr. rer. nat. REINER DUMKE

Mit 11 Bildern, 73 Übungsaufgaben und einem Anhang

Dumke, Reiner:

Makroprogrammierung: Einführung, Übungen, Praxis / von Reiner Dumke. – 1. Aufl. – Leipzig : Fachbuchverl., 1988. – 194 S.: mit 11 Bild., 73 Übungsaufg. u. e. Anh.
(Mathematik für Ingenieure)
NE: GT

ISBN 3-343-00228-3

1. Auflage
Lizenznummer 114-210/104/88
LSV 1083
Verlagslektor: Helga Fago
Gestaltung: Christine Graichen
Printed in GDR
Satz und Druck: VEB Druckhaus Köthen, DDR-4370 Köthen
Buchbinderische Verarbeitung: VEB Druckhaus „Maxim Gorki", DDR-7400 Altenburg
Redaktionsschluß: 15. 10. 1987
Bestellnummer 5472271
02200

Vorwort

Die Makroprogrammierung ist gegenwärtig und war auch zuvor kein Modewort, das in aller Munde geführt wird. Dennoch ist der feste Platz der Makroprogrammierung innerhalb der Theorie und Praxis der Programmierung von Groß-, mittleren, Klein- und Mikrorechnern wohl unbestritten. Das zeigt nicht zuletzt auch die Tatsache, daß heutzutage jedes größere Software-Entwicklungssystem bzw. -Erstellungssystem einen Makroprozessor enthält. Aber auch für die gegenwärtig hohen Effektivitätsanforderungen bei der Erstellung von Software für die Mikrorechentechnik ist die Makrotechnik ein wichtiges Hilfsmittel zur Realisierung zweckmäßiger Lösungen.

Für ein Studium der Makroprogrammierung existieren bereits einige Bücher, die sich mehr oder weniger ausführlich mit der Darstellung und zweckmäßigen Anwendung dieser Programmierungstechnik befassen. Vor allem seien dabei die Bücher von BROWN ([25] und [29]), von CAMPBELL-KELLY ([36]) und von COLE ([210]) genannt. Neben der Beschreibung einiger Makrosprachen widmen sie sich insbesondere dem Aufbau und der Wirkungsweise von Makroprozessoren. Diese Bücher sind jedoch als Lehrbuch wenig geeignet (allenfalls als Ergänzungsliteratur), weil

- sie nur in englisch oder russisch vorliegen;
- sich bereits neuere Erkenntnisse auf dem Gebiet der Makroprogrammierung ergeben haben, wie beispielsweise die Anwendung der Makroprogrammierung auf weitere höhere Programmiersprachen oder für die Realisierung moderner Programmierungsmethoden.

Andererseits existieren bereits deutschsprachige Lehrbücher zur Makroprogrammierung wie von HAUPT/KRETZSCHMAR ([158]) und SOBOTTA ([166])], die sich jedoch ausschließlich mit der Makroprogrammierung für eine spezielle ASSEMBLER-Sprache befassen. Ziel des vorliegenden Buches ist es, eine zusammenfassende Darstellung des heute bereits sehr umfangreichen Gebietes der Makroprogrammierung zu geben. Dabei werden Grundkenntnisse in der Programmierung vorausgesetzt. Der der Behandlung der Makroprogrammierung vorangestellte Abschnitt zu einigen Grundlagen der Programmierung dient der Vorbereitung zur Einordnung der Makroprogrammierung und der Festlegung spezieller Darstellungsweisen. Dem Abschnitt zur allgemeinen Darstellung der Makroprogrammierung, der Beispiele spezieller (auch internationaler) Makrosprachen enthält, folgen Abschnitte, die zum einen die ASSEMBLER-Makroprogrammierung für drei Computerarten und zum anderen die PL/1-Makroprogrammierung ausführlicher beschreiben. Die Darstellung der ASSEMBLER-Makrosprachen wurde dabei nur soweit gegeben, wie sie ohne die Kenntnis der jeweiligen ASSEMBLER-Sprache möglich ist.

Neben der Einführung für den Anfänger auf dem Gebiet der Makroprogrammierung informiert dieses Buch auch den spezialisierten Programmierer über die bereits vorhandene Vielfalt der Makroprogrammierung (Anhang 2) und unterstützt ihn durch eine umfangreiche Literaturangabe und eine Zusammenfassung der wichtigsten Begriffe zur Makroprogrammierung in deutsch, russisch, englisch und französisch (Anhang 3) bei der selbständigen Einarbeitung in dieses Fachgebiet.

Mein Dank gilt den Mitarbeitern meines Wissenschaftsbereiches und dem Gutachter Herrn Dr.-Ing. H. GÖTZKE für wertvolle Hinweise bei der Abfassung des Buches sowie dem VEB Fachbuchverlag für die verständnisvolle Zusammenarbeit.

Der Verfasser

Preface

This book gives a concentrated representation of the macro programming, that is today an extensive field. Knowledges in computer programming are required.
Before the representation of macro programming is given a chapter with some fundamentals of programming for the preparation of the reader for the classification of the macro programming and the definition of special notions and forms are introduced.
After the chapter of the general macro programming there is a detailed representation of the assembly macro languages for three kinds of computers and a chapter of the PL/1-macro programming. The representation of the assembly macro languages is given in such a way that special knowledges in the assembly language are not required.
This book is written for beginners in this field. But also specialist will find a lot of special information (appendix 2) and help for an autodidactic work in this field (help extensive bibliography and appendix 3).

the author

Avant-propos

Ce livre comprend une représentation concentrée de la macro-programmation quelqu'est un domaine étendu aujourd'hui. Pour cela est suppose des connaissances en programmation. Avant la représentation de la macro-programmation est présent un chapitre avec quelques notions fondamentaux de la programmation pour la préparation de lecteur pour la classification de la macro-programmation et pour la définition des notions et formes spéciales.
Aprés le chapitre de la macro-programmation générale il présente une description detaillée des macro-langages d'assemblage pour trois sortes d'ordinateurs et un chapitre du PL/1-macro-programmation.
La représentation des macro-langages d'assemblage est présentée sans la nécessité de connaissances du langage d'assemblage. Ce livre est adressé pour les deputants dans le domaine de la macro-programmation. Mais aussi il présente beaucoup d'informations (appendice 2) et des moyens pour le travail autodidactique de programmeur spécialiste dans ce domaine (une bibliographie étendu et appendice 3).

l'auteur

Inhaltsverzeichnis

Contents

Contenu

0. Einleitende historische Bemerkungen

Die Makroprogrammierung – im Zusammenhang mit ihren Realisierungsmitteln auch als Makrotechnik bezeichnet – gliedert sich in die Programmierungstechniken ein, das heißt, sie ist ein spezieller Weg, zu Computerprogrammen (allgemein als Software bezeichnet) zu gelangen. Sie entstand etwa gleichzeitig mit dem Entwurf und der Implementierung der ersten Programmiersprachen Ende der 50er, Anfang der 60er Jahre. Dabei wurden zunächst vor allem die computerspezifischen ASSEMBLER-Sprachen unterstützt. Theoretisch zusammengefaßt wurden die Prinzipien der Makrotechnik erstmals durch McILROY ([94]) und ergänzt durch CHEATHAM ([38]).

Bald erkannte man jedoch die universelle Anwendbarkeit, die diese spezielle Technik zur rationellen Erzeugung von Programmen bietet. Eine Verallgemeinerung der Anfangsgedanken zur Makroprogrammierung stellt der erste sogenannte allgemeine Makroprozessor GPM (**G**eneral **P**urpose **M**acrogenerator) von STRACHEY ([125]) dar. Er war bereits nicht mehr für eine spezielle Programmiersprache zugeschnitten, sondern allgemeiner anwendbar.

Bereits ein Jahr später (1966) implementierte BROWN den allgemeinen Makroprozessor zu ML/I (**M**acro **L**anguage **1**), dessen Anwendungsbreite und Nutzungsbeständigkeit wohl diese Bezeichnung (als Makrosprache Nummer 1) rechtfertigte.

Ein Beitrag von LEAVENWORTH ([89]) über sogenannte Syntaxmakros gab wiederum neue Impulse für die Entwicklung der Makroprogrammierung. So wurden beispielsweise auf dieser Basis auch spezielle Probleme der Systemprogrammierung, wie der Compilerbau, mittels der Makrotechnik effektiven Lösungen zugeführt.

Aber nicht nur die Anwendungsbreite erweiterte sich. Auch die Makroprozessoren selbst wurden als spezielle für weitere Programmiersprachen, wie PL/1, FORTRAN und COBOL, erschlossen beziehungsweise als allgemeine Makroprozessoren wirkungsvoller bei der Textgenerierung eingesetzt.

Bemerkenswert ist, daß sich die mathematische Darstellung und Behandlung der Makroprogrammierung erst viel später vollzog. Auch können die bisherigen Beiträge von FISCHER ([219]), KAUFMAN ([83]) und ENGELFRIET ([52], [53] und [54]) nur als Ansätze betrachtet werden. Im vorliegenden Buch wird eine mathematische Behandlung der Makroprogrammierung nur soweit gegeben, wie sie bei der Darstellung jener als unbedingt notwendig erscheint.

Für die exakte Darstellung der Mittel und Methoden der Makroprogrammierung werden zunächst grundlegende Begriffe der Programmierung zusammengefaßt.

1. Grundbegriffe der Programmierung

1.0. Einleitende Bemerkungen

Die Realisierung von Algorithmen auf einer elektronischen Datenverarbeitungsanlage (computer) geschieht durch die Abarbeitung einer endlichen Menge von Operationen, die eindeutig bestimmt sind. Diese Operationsmenge (auch Befehlsfolge oder Anweisungsfolge genannt) ist in einer maschineninternen Darstellung gegeben und wird als *Programm* (program) bezeichnet. Dabei wurden im Laufe der Zeit Methoden entwickelt, die das Erstellen eines Programms in der maschineninternen Darstellung vereinfachten. Es wurden sogenannte Programmiersprachen definiert, die in ihrem Wortumfang und Inhalt den auf einem Computer zu realisierenden Problemen (als Algorithmen) nahekommen. Diese Programmiersprachen haben gegenüber der natürlichen Sprache einen begrenzteren Wortumfang und sind im allgemeinen durch formale Regeln exakt definiert. Sie gehören daher zu den formalen Sprachen.

1.1. Generative Grammatiken und formale Sprachen

Eine *formale Sprache* (formal language) ist eine Menge von Wörtern über einem endlichen Alphabet. Ein *Wort* (word) ist dabei eine Buchstabenfolge (Zeichenfolge, Symbolfolge) aus einer Menge von Buchstaben (Zeichen, Symbolen), dem Alphabet. So sind beispielsweise **Baum**, **Feld** und **Laub** Wörter aus dem Alphabet A_1 mit $A_1 = \{A, B, C, \ldots, Z, a, b, c, \ldots, z\}$. Ein Alphabet kann sich aber auch aus beliebigen anderen Zeichen zusammensetzen, wie zum Beispiel $A_2 = \{+, -, (,), a, b\}$. Wörter über diesem Alphabet sind beispielsweise $a + b$, $(a + b) - a$, $((a - b) + b)$, aber auch $+a)(, b - -+a$ und $(((a$.
Die Menge aller erzeugbaren Wörter über einem Alphabet A wird mit A^* bezeichnet. Dabei gehöre auch das (zweckmäßigerweise eingeführte) *leere Wort* ε zu dieser Menge. Jedes Wort besteht aus einer bestimmten Anzahl von Zeichen bzw. Buchstaben, die die *Länge* (length) eines Wortes ausdrückt. Für ein Wort w wird diese Länge durch $|w|$ dargestellt. Es gilt also $|\mathbf{Baum}| = 4$, $|a + b| = 3$ usw. Speziell sei $|\varepsilon| = 0$ für das leere Wort.
Wörter können miteinander verbunden werden. Diese Operation heißt *Verkettung* (concatenation). So ergibt sich beispielsweise aus der Verkettung der Wörter **Laub** und **Baum** das Wort **LaubBaum**. Allgemein drückt w_1w_2 die Verkettung der Wörter w_1 und w_2 aus. Bezüglich der Länge des erhaltenen Wortes gilt $|w_1w_2| = |w_1| + |w_2|$.
Die n-malige Verkettung eines Wortes w mit sich selbst lautet $\underbrace{www \ldots w}_{n+1}$ und wird kurz mit w^{n+1} bezeichnet. Speziell wird 0^n für $\underbrace{000 \ldots 0}_{n}$ geschrieben.

Wie die natürlichen Sprachen, so besitzen auch die formalen Sprachen Grammatiken, sogenannte formale Grammatiken. Dabei unterscheidet man *generative formale Grammatiken* (generating formal grammars), die die Erzeugung der zur jeweiligen formalen Sprache ge-

hörenden Wörter beschreiben, und *entscheidende formale Grammatiken* (decisiving formal grammars), die nur feststellen, ob ein Wort zur jeweiligen formalen Sprache gehört oder nicht.
Im folgenden sollen ausschließlich generative formale Grammatiken (kurz generative Grammatiken genannt) betrachtet werden.
Eine generative Grammatik sei formal beschrieben in der Form $G = (A, H, \sigma, P)$ mit

A als Grundalphabet (terminales Alphabet),
H als Hilfsalphabet (nichtterminales Alphabet),
σ als (nichtterminales) Anfangssymbol (Axiom) und
P als endliche Menge von Regeln der Form

$$u_i \to v_i$$

mit $u_i, v_i \in A \cup H$ für $i = 1, 2, 3, \ldots, n$.

Der Pfeil in der Regel kennzeichnet dabei die Überführung einer Zeichenfolge in eine andere.
Ein Beispiel für eine generative Grammatik ist gegeben durch $G_1 = (\{0, 1\}, \{\alpha, \sigma\}, \sigma, P_1)$ mit $P_1 = \{\sigma \to 0\alpha 1, \alpha \to 0\alpha 1, \alpha \to \varepsilon\}$, wobei ε wiederum für das leere Wort steht. Die Anwendung der Regeln aus P_1 wird immer mit dem Anfangssymbol σ begonnen. Die sukzessive Anwendung der Regeln aus P_1 heißt *Ableitung* (derivation). Eine mögliche Ableitung in G_1 ist dann zum Beispiel

$$\sigma \xrightarrow{(1)} 0\alpha 1 \xrightarrow{(2)} 00\alpha 11 \xrightarrow{(2)} 000\alpha 111 \xrightarrow{(3)} 000111.$$

Die Numerierung über den Pfeilen gibt die Anwendung der jeweiligen Regel aus P_1 an.
Enthält das durch eine Ableitung generierte Wort nur Symbole bzw. Zeichen des Grundalphabetes, so liegt eine *vollständige Ableitung* vor. Sonst heißt die Ableitung unvollständig.
Eine *unvollständige Ableitung* in G_1 hat beispielsweise die Form

$$\sigma \to 0\alpha 1 \to 00\alpha 11.$$

Eine Ableitung kann auch verkürzt dargestellt werden. Dabei kennzeichnet ein Stern über dem Ableitungspfeil weggelassene Zwischenschritte. Für die obige vollständige Ableitung kann dann also auch $\sigma \xrightarrow{*} 000111$ geschrieben werden.
Bild 1 zeigt die durch G_1 erzeugbaren Wörter.

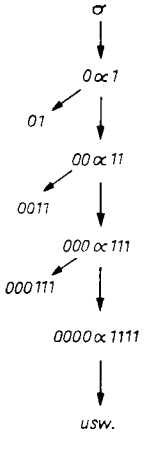


Bild 1. Ableitungsbaum der durch G_1 erzeugbaren Wörter

Die Grammatik G_1 generiert also eine formale Sprache $L(G_1)$, deren Wörter aus einer Anzahl von Nullen, gefolgt von derselben Anzahl Einsen, bestehen, oder kurz $L(G_1) = \{0^n1^n \mid n \geqq 1\}$.
Die durch eine generative Grammatik über einem Alphabet A erzeugte Wortmenge ist stets eine Teilmenge von A^*.
Der grammatikalische Aufbau der Wörter einer Sprache ist dabei die *Syntax* (syntax), während die inhaltliche Bedeutung einer formalen Sprache deren *Semantik* (semantic) darstellt. Generative Grammatiken beschreiben also im wesentlichen die Syntax einer formalen Sprache.
Ein weiteres Beispiel einer generativen Grammatik ist

$$G_2 = (\{a, b\}, \{\sigma\}, \sigma, P_2) \quad \text{mit} \quad P_2 = \{\sigma \to a\sigma a, \sigma \to b\}.$$

Eine vollständige Ableitung in G_2 ist beispielsweise

$$\sigma \to a\sigma a \to aa\sigma aa \to aabaa.$$

Die Grammatik G_2 erzeugt also die Sprache $L(G_2) = \{a^nba^n \mid n \geqq 0\}$, wobei für a^0 das leere Wort einzusetzen ist.
Im folgenden soll die Erarbeitung der Regelmenge einer generativen Grammatik zu einer vorgegebenen formalen Sprache gezeigt werden. Diese Sprache sei die Menge der ganzen positiven Dualzahlen, wobei die Eins durch das Zeichen »L« dargestellt werden soll. Die diese Sprache erzeugende Grammatik hat also das Grundalphabet {0, L}. Durch die Regeln

$$\sigma \to 0 \quad \text{und} \quad \sigma \to \mathrm{L}$$

werden die beiden Dualzahlen 0 und L erzeugt. Die Regelerweiterung auf

$$\sigma \to 0\varkappa, \sigma \to \mathrm{L}\varkappa \quad \text{und} \quad \varkappa \to \varepsilon$$

liefert zunächst dasselbe Ergebnis. Erst die Hinzunahme der weiteren Regeln

$$\varkappa \to 0\varkappa \quad \text{und} \quad \varkappa \to \mathrm{L}\varkappa$$

ergibt die mögliche Generierung dieser Dualzahlen. Diese Regelmenge besitzt aber noch einen »Schönheitsfehler«. So können durch sie auch die Dualzahlen 000 ... 0, 000L0LL u.a.m. erzeugt werden. Um diese mit Vornullen behafteten Dualzahlen bei der Generierung auszuschließen, wird die Regel $\sigma \to 0\varkappa$ wieder in ihre ursprüngliche Form $\sigma \to 0$ gebracht. Damit lautet die vollständige Regelmenge

$$P_3 = \{\sigma \to 0, \sigma \to L\varkappa, \varkappa \to 0\varkappa, \varkappa \to \mathrm{L}\varkappa, \varkappa \to \varepsilon\}$$

und die generative Grammatik $G_3 = (\{0, \mathrm{L}\}, \{\varkappa, \sigma\}, \sigma, \mathrm{P}_3)$. Vollständige Ableitungen in G_3 sind beispielsweise

$$\sigma \to \mathrm{L}\varkappa \to \mathrm{L}0\varkappa \to \mathrm{L}0,$$

$$\sigma \to 0,$$

$$\sigma \to \mathrm{L}\varkappa \to \mathrm{LL}\varkappa \to \mathrm{LL}0\varkappa \to \mathrm{LL}0\mathrm{L}\varkappa \to \mathrm{LL}0\mathrm{L}.$$

Die Erweiterung des Quadrupels, das eine generative Grammatik beschreibt, um eine weitere Menge von Nichtterminalen definiert als Quintupel die sogenannte Indexgrammatik.
Eine *Indexgrammatik* (indexed grammar) ist die generative Grammatik $G = (A, H, T, \sigma, P)$, wobei A, H, σ und P die obengenannten Bedeutungen zukommen und T die sogenannte Indexmenge darstellt. Für die Regeln in P gilt, wenn

$$u \to v_1v_2v_3 \ldots v_n$$

eine Regel in P ist, so ist auch

$$ut \to v_1 t v_2 t v_3 t \ldots v_n t$$

mit $n \geqq 1$ in P enthalten. Dabei gilt $u, v_i \in A \cup H$ für $i = 1, 2, \ldots, n$ und $t \in T^*$. u wird *indiziertes Nichtterminal* (indexed nonterminal) genannt.
Die Grammatik $G_4 = (\{a, b, c\}, \{\varkappa, \beta, \gamma, \delta, \zeta, \sigma\}, \{f, g\}, \sigma, P_4)$ mit

$$P_4 = \{\sigma \to \beta g, \beta \to \beta f, \beta \to \gamma\delta\zeta, \gamma f \to a\gamma, \delta f \to b\delta,$$
$$\zeta f \to c\zeta, \gamma g \to a, \delta g \to b, \zeta g \to c\}$$

ist beispielsweise eine Indexgrammatik. Sie erzeugt die Sprache $L(G_4) = \{a^n b^n c^n \mid n \geqq 1\}$.
Gemäß der oben gegebenen Definition einer Indexgrammatik schließt also die Regel $\beta \to \gamma\delta\zeta$ auch die Regeln $\beta f \to \gamma f \delta f \zeta f$ oder $\beta g \to \gamma g \delta g \zeta g$ ein.
Die Ableitung für das Wort aabbcc in G_4 lautet daher

$$\sigma \xrightarrow{(1)} \beta g \xrightarrow{(2)} \beta f g \xrightarrow{(3)} \gamma f g \delta f g \zeta f g \xrightarrow{(4)} a\gamma g \delta f g \zeta f g \xrightarrow{(7)} aa\delta f g \zeta f g$$
$$\xrightarrow{(5)} aab\delta g \zeta f g \xrightarrow{(8)} aabb\zeta f g \xrightarrow{(6)} aabbc\zeta g \xrightarrow{(9)} aabbcc.$$

Die Zahlen über den Pfeilen geben wiederum die Anwendung derjenigen Regel an, die man durch fortlaufende Numerierung der Regeln in P_4 erhält.

Übungsaufgaben

1.1. Welche Sprache $L(G_5)$ wird durch die generative Grammatik $G_5 = (\{(,)\}, \{\varkappa, \sigma\}, \sigma, P_5)$ mit
$P_5 = \{\sigma \to \varkappa, \varkappa \to (\,), \varkappa \to (\varkappa), \varkappa \to \varkappa\varkappa\}$
erzeugt?

1.2. Geben Sie eine generative Grammatik G_6 an, die die Sprache $L(G_6) = \{a^n b^m \mid n, m \geqq 1, n < m\}$ erzeugt!

1.3. Welches Wort liefert letztlich die Ableitung
$\sigma \xrightarrow{*} \gamma fff g \delta fff g \zeta fff g$
mit der Indexgrammatik G_4?

1.4. Welche Sprache wird durch die Indexgrammatik
$G_7 = (\{a, b\}, \{A, B, \sigma\}, \{f, g\}, \sigma, P_7)$ mit
$P_7 = \{\sigma \to Tg, T \to Tf, T \to AB, Af \to aA, Bf \to bbB, Ag \to a, Bg \to bb\}$
generiert?

1.2. Programmiersprachen

Eine *Programmiersprache* (programming language) faßt alle Anweisungen (als formalisierte Befehle für einen Computer) zusammen, aus denen eine bestimmte Klasse von Programmen gebildet werden kann. Bezüglich der Wirkung dieser Anweisungen unterteilen sie sich in

(a) *beschreibende* (*definierende*) *Anweisungen* (descriptive statements); zum Beispiel für das Festlegen des Variablentyps (ganze Zahl, reelle Zahl, Zeichenkette usw.) oder der Variablenstruktur (einfache Variable, Feld, Struktur usw.),
(b) *Ausführungsanweisungen* (imperative statements); als Zuweisungen (einschließlich der Ein- und Ausgabe), Sprünge, Zyklen, Prozeduren u.a.m.

Zum Aufstellen von Programmen sind in der jeweiligen Programmiersprache über die oben genannten, elementaren Anweisungen weitere, komplexere Anweisungen konstruierbar.

Dabei werden zunehmend exakte Strukturierungsprinzipien angewandt. Benutzt man beispielsweise die in Bild 2 angegebenen Grundstrukturen, so realisiert man die *Strukturierte Programmierung* (structured programming). Die Zusammenfassung einer Anweisungsmenge nach funktionellem Aspekt ist in den meisten Programmiersprachen durch ein sogenanntes *Unterprogramm* (subprogram, subroutine, function, procedure) möglich. Ein Unterprogramm ist durch einen Namen identifiziert und kann eine Parametermenge für die Übergabe spezieller Werte besitzen. Die im Unterprogramm (kurz UP) definierten Parameter heißen *formale Parameter* (formal parameters), während die Parameter(-werte) bei der Nutzung des UP, dem Aufruf des UP, als *aktuelle Parameter* (actual parameters) bezeichnet werden.

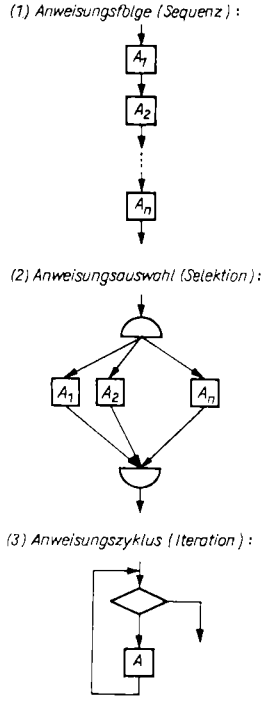


Bild 2. Grundstrukturen der Strukturierten Programmierung

Die Tatsache, daß man zu einem vorgegebenen Algorithmus in einer Programmiersprache (formal) unendlich viele Programme aufstellen kann, zeigt das unbedingte Erfordernis, die Übereinstimmung dieses zumeist verbal beschriebenen Algorithmus mit dem zugehörigen Programm hinsichtlich seines Funktionsverhaltens bei der Abarbeitung zu überprüfen. Dieser Vorgang heißt *Programmtest* (program testing). Die Programmtestung kann in verschiedener Weise erfolgen. Die Methode, die Programmniederschrift mit dem zu realisierenden Algorithmus zu überprüfen, heißt *Programmverifikation* (program verification). Die Programmniederschrift in einer speziellen Programmiersprache wird auch als *Quellprogramm* (source program) bezeichnet. Bei der Programmverifikation wird das nach jeder Anweisung des Programms erforderliche Resultat gemäß dem vorgegebenen Algorithmus als formale Beschreibung eingefügt. Bei der nun folgenden schrittweisen Interpretation des Quellprogramms wird das letztlich zu erzielende Resultat als allgemeiner, logischer Ausdruck hergeleitet und mit den vorgegebenen Zielen verglichen. Eine weitere Testmethode, die sich ebenfalls auf das Quellprogramm bezieht, ist der *symbolische Programmtest* (symbolic execution). Dabei wird das Quellprogramm mittels vorgegebener Symbole für die Eingabe-

daten durchlaufen (interpretiert). Das Ergebnis ist eine formelmäßige Ablaufprotokollierung des Programms, welche dann mit dem geforderten Resultat des zugrunde gelegten Algorithmus, das ebenfalls formelmäßig vorliegen muß, verglichen wird.
Die noch am häufigsten angewandte Testmethode ist das *systematische Testen* (systematic program testing, program execution). Es bezieht sich auf das abarbeitungsfähige Programm. Die Systematik liegt in der Auswahl der jeweiligen Eingabedaten (Testdaten) für das Programm. Diese Testdaten werden zumeist so gewählt, daß alle Programmzweige durchlaufen werden oder daß sie für das Ergebnis besonders bedeutungsvoll sind u.a.m. Letztlich wird durch den systematischen Test jedoch immer nur die Korrektheit des Programms für die vorgegebenen Testdaten gezeigt.
Im folgenden soll auf einige Klassifikationen der Programmiersprachen eingegangen werden.
Die auf einem ESER-Computer abarbeitungsfähige Form eines arithmetischen Ausdrucks zur Multiplikation zweier Zahlen und Addition mit einer dritten hat die maschineninterne Darstellung »5850 C0E8 5C40 C0EC 5A50 C0F0«. Jedes Programm in irgendeiner Programmiersprache ist letztlich für diesen Computer in diese Form zu überführen. Je nach der »Nähe« einer Programmiersprache zu dieser Darstellung unterscheidet man in *maschinenorientierte* (mashine-oriented programming languages) und *höhere Programmiersprachen* (high level programming languages). Die der maschineninternen Darstellung nächstliegende Programmiersprache, die die Möglichkeit der Verwendung symbolischer Bezeichnungen besitzt, wird bei allen Computern *ASSEMBLER-Sprache* (assembly language) genannt.
Höhere Programmiersprachen werden nach der möglichen inhaltlichen Menge der durch sie implementierbaren Algorithmen in *universelle Programmiersprachen* (universal programming languages, generalpurpose programming languages), wie zum Beispiel PL/1 (**P**rogramming **l**anguage **1**) oder ALGOL 68 (**Algo**rithmic **l**anguage 1968), und in *spezielle Programmiersprachen* (special-purpose programming language), wie zum Beispiel FORTRAN (**For**mular **trans**later) oder COBOL (**Co**mmon **bu**siness **o**riented **l**anguage), unterschieden.
Spezielle Programmiersprachen werden auch als *problemorientierte Programmiersprachen* (problem oriented programming languages) bzw. bei der Orientierung auf einen bestimmten Anwenderkreis als *Nutzersprachen* (user languages) bezeichnet. Dient eine Programmiersprache den Nutzern eines speziellen Fachgebiets, so wählt man die Bezeichnung *Fachsprache*. Fachsprachen können dabei die unterschiedlichste Gestalt haben. So zum Beispiel die Notensprache der Musik, die Diagramme in der Pictographie, die Symbole in der Ideographie oder auch die symbolische Sprache der mathematischen Logik.
Programmiersprachen, die die interaktive Arbeit mit einem Computer gestatten, werden als *Dialogsprachen* (dialogue programming languages) bezeichnet. Besteht eine derartige Sprache ausschließlich aus Komplexanweisungen (sogenannten Kommandos), so handelt es sich um eine *Kommandosprache* (command language). Eine andere Charakterisierung einer Programmiersprache ergibt sich aus der syntaktischen Form der in der jeweiligen Sprache möglichen Anweisungen. Haben diese ausschließlich die Form

$$\text{name}(p_1, p_2, \ldots, p_n),$$

wobei für »name« irgendeine Funktionsbezeichnung und für $p_1, p_2, \ldots, p_n$ Parameterwerte anzugeben sind, so liegt eine *Parametersprache* (parameter language) vor.

Übungsaufgaben

1.5. Entspricht der in Bild 3 gegebene Programmablaufplan den Prinzipien der Strukturierten Programmierung?
1.6. Kennzeichnen Sie den Unterschied zwischen der Programmverifikation und dem symbolischen Testen!

1.7. Welche Grundstruktur der Strukturierten Programmierung wird durch Parametersprachen realisiert?
1.8. Von welchem Programmiersprachtyp hinsichtlich der Nähe zur maschineninternen Darstellung sind im allgemeinen Fachsprachen?

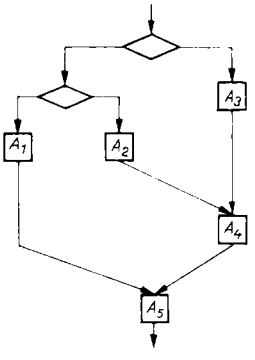


Bild 3. Programmablaufplan

1.3. Compiler, Interpreter und Generatoren

Die Übersetzung eines in einer Programmiersprache geschriebenen Programms in die maschineninterne Form eines Computers wird durch einen *Compiler* – ebenfalls ein Programm – realisiert. Formal sei ein Compiler dargestellt in der Form

$$\overset{c}{L_1,L_2,L},$$

wobei L_1 die zu übersetzende Programmiersprache (Definitionsbereich), L_2 die Zielsprache (Wertebereich) und L die Sprache ist, in der der Compiler geschrieben wurde. Für ein Programm p in der Sprache L_1 stellt dann $c(p) = q$ dessen Übersetzung in ein Programm q der Sprache L_2 dar. Im eingangs beschriebenen Fall ist L_2 die Maschinensprache des jeweiligen Computers. Das Programm p in der Programmiersprache L_1 ist das Quellprogramm.
Eine andere Möglichkeit, zum Programm in der Maschinensprache zu gelangen, ist zum Beispiel die Verwendung mehrerer Compiler in der Art

$$\overset{c_1}{L_1,L_2,L} \ * \ \overset{c_2}{L_2,L_3,L} \ * \ \overset{c_3}{L_3,L_4,L} = \overset{c}{L_1,L_4,L}$$

mit '$*$' als Symbol für die Hintereinanderausführung (*Komposition*) von Compilern. L_2 und L_3 sind dabei sogenannte *Zwischensprachen* (intermediate languages). Andererseits liefert die Anwendung eines Compilers c_1 auf den Compiler c_2 in der Art

$$\overset{c_1}{L_1,L_2,L} \left(\overset{c_2}{L_3,L_4,L_1} \right) = \overset{c_2'}{L_3,L_4,L_2}$$

einen Compiler c_2', der in der Zielsprache L_2 des Compilers c_1 geschrieben ist und die Leistung (Übersetzung von L_3 in L_4) des Compilers c_2 besitzt.
Sind L_1 und L_2 hierbei Sprachen zweier unterschiedlicher Computer, so realisiert c_1 die Übertragung des Compilers c_2 auf einen anderen Computer. Die so mögliche Übertragung von c_2 heißt *Portabilität* (portability).

Werden durch einen Compiler aus unterschiedlichen Eingabeprogrammen einer Sprache Programme erzeugt, die selbst Compiler sind, das heißt, wenn gilt

$$c(p_1)=c_1,\ c(p_2)=c_2,\ \ldots,\ c(p_n)=c_n, \text{ so heißt } c \textit{ Compiler-Compiler.}$$

Der Übersetzungsprozeß selbst ist im allgemeinen in Stufen bzw. Phasen gegliedert. Die einzelnen Arbeitsstufen sind dabei die lexikalische Analyse, die syntaktische Analyse, die Zielprogrammerzeugung und die Zielprogrammoptimierung. Bild 4 zeigt die einzelnen Aktivitäten in diesen Stufen. Während ein Compiler Programme einer Sprache in eine andere Sprache übersetzt, hat sich auf Grund der Tatsache, daß es für einen Computer zumeist mehrere Programmiersprachen und damit mehrere Compiler gibt, die Notwendigkeit ergeben, mehrere, aus verschiedenen Programmiersprachen übersetzte Programme zu einem Programm zu verbinden. Das geschieht wiederum mit Hilfe eines Programms, dem sogenannten *Programmverbinder* (linkage editor). Das setzt voraus, daß die Zielsprache dieser Compiler einheitlich ist. Sie wird im allgemeinen als *Objektsprache* (object language) bezeichnet. Das Übersetzungsergebnis ist dann das *Objektprogramm* (object program), das im allgemeinen noch nicht auf dem jeweiligen Computer abarbeitbar ist. Der Programmverbinder v hat formal die Gestalt

$$\overset{v}{L_O,L_Z,L}$$

und verbindet somit Programme in der Objektsprache L_O zu einem Programm in der Zielsprache L_Z und ist selbst in der Sprache L geschrieben. Speziell liefert die Verbindung der Objektprogramme $q_1, q_2, q_3, \ldots, q_n$ das verbundene Programm r, also es gilt

$$v(q_1,q_2,q_3,\cdots,q_n)=r\ .$$

Die einzelnen Funktionen des Programmverbinders sind dabei beispielsweise das »Einbinden« der Anweisungen für die sogenannten Standardfunktionen der Programmiersprache, die konkrete Umsetzung von definierten Überlagerungsstrukturen beim Programmablauf u.a.m. Auf diese Einzelheiten soll jedoch hier nicht weiter eingegangen werden, genauso wie das Laden eines Programms zur endgültigen Herstellung der Abarbeitungsfähigkeit hier als letzte Stufe der Programmverbindung eingeordnet wird.
Die Abarbeitung eines Programms in einer Sprache (i. allg. der Maschinensprache) auf dem jeweiligen Computer wird als *Interpretation* bezeichnet, und das Programm, das diese Interpretation realisiert, ist der *Interpreter*. Formal dargestellt in der Form

$$\overset{i}{L_1,L_2,L}$$

handelt es sich um einen Interpreter i, der Programme in der Sprache L_1 interpretiert und selbst in der Sprache L geschrieben ist. Während Compiler stets Programme erzeugen, sind das Ergebnis der Interpretation berechnete Werte u.a.m. Formal handelt es sich bei diesen Werten aber auch um Wörter einer Sprache, die hier mit L_2 bezeichnet wurde. Ein Programm p mit dem Interpretationsergebnis w wird formal dargestellt durch $i(p)=w$, und es gilt $w \in L_2$.

Spezielle Interpretationstechniken sind

- die direkte Interpretation eines Quellprogramms,
- die Interpretation einer im Compiler gebildeten Zwischensprache,
- die Interpretation eines Programms in der Maschinensprache (als Abarbeitung desselben).

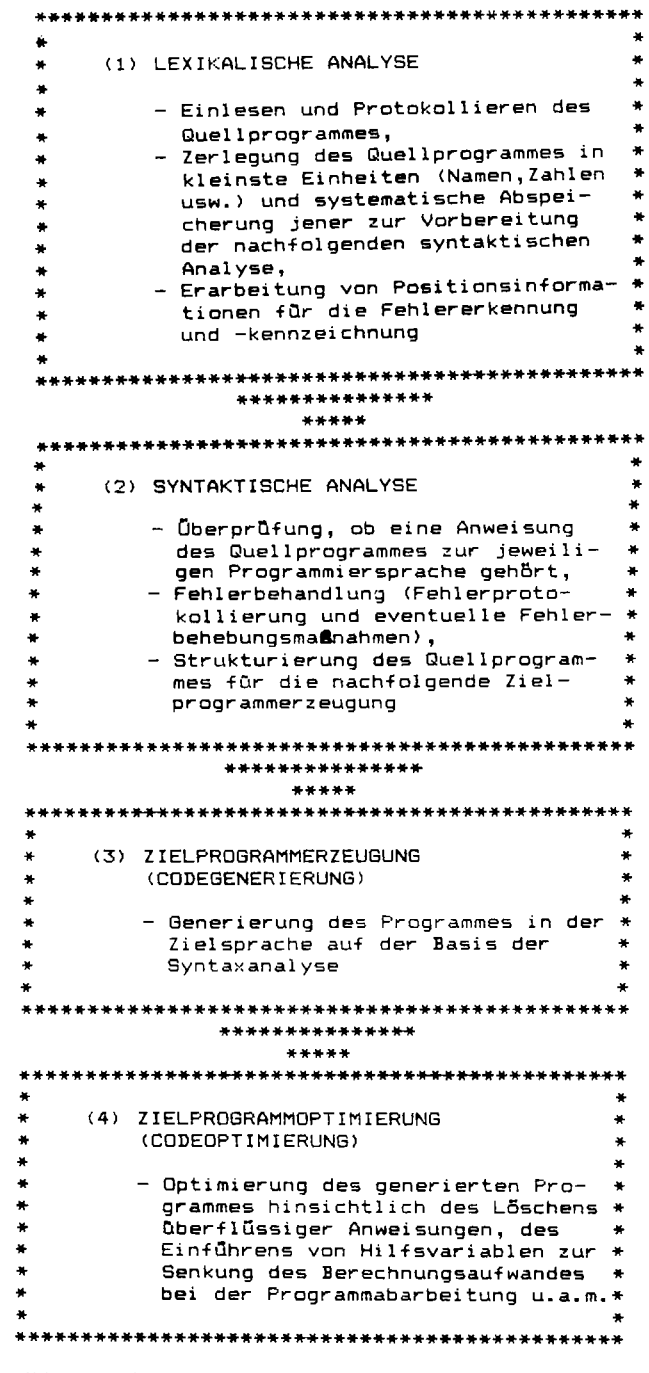


Bild 4. Arbeitsstufen eines Compilers

Bei leistungsstarken Interpretern sind der eigentlichen Interpretation zumeist Kontrollfunktionen des syntaktischen Aufbaus der eingegebenen Wörter vorgelagert. Der Hauptunterschied zwischen Interpreter und Compiler wird aber auch dabei nicht aufgehoben; nämlich die Transformation des *gesamten* Quellprogramms durch den Compiler zum einen und die Interpretation von aufeinanderfolgenden *Teilen* eines Programms durch den Interpreter. Einfache Formen der Sprachübertragung werden mit Hilfe sogenannter Generatoren realisiert. Ein *Generator* sei hierbei ein Programm mit der formalen Beschreibung

$$\overset{g}{L_1,L_2,L}$$

Er generiert eine Zeichen- bzw. Wortfolge aus der Sprache L_1 in eine Zeichen- bzw. Wortfolge der Sprache L_2 und ist selbst in der Sprache L geschrieben. Ist L_2 eine Programmiersprache, so heißt g *Programmgenerator* (program generator). Im Gegensatz zum Compiler ist die Zielsprache L_2 von g nicht ausschließlich eine maschinenorientierte Sprache eines Computers.
Programmgeneratoren realisieren also inhaltlich die Arbeitsstufe »Zielprogrammgenerierung« des Compilers. Leistungsfähige Generatoren realisieren auch Funktionen der syntaktischen Kontrolle und Fehlerbehandlung für die Wörter der Eingabesprache. Für ein eingegebenes Programm p_1 (zumeist eine Menge von Parameterwerten) in der Sprache L_1 wird speziell eine Zeichenfolge bzw. ein Programm p_2 in der Sprache L_2 generiert, also $g(p_1)=p_2$.
Compiler, Interpreter und Generatoren bilden zusammen die Menge der *Prozessoren* (processors).
Die oben genannten Operationen über mehrere Compiler (die Komposition und das »Anwenden auf«) gelten auch für Interpreter und Generatoren. So wird mit

$$\overset{i_1}{L_1,L_2,L} \;*\; \overset{i_2}{L_2,L_3,L} \;=\; \overset{i}{L_1,L_3,L}$$

die Hintereinanderausführung der Interpreter i_1 und i_2, die ein Programm in der Sprache L_1 mit dem Ergebnis in der Sprache L_3 interpretieren, dargestellt. Dabei ist die Wahl der Ein- bzw. Ausgabesprache jedes Interpreters für die sinnvolle Komposition bedeutsam, das heißt, für die Tatsache, daß jeder folgende Interpreter die Sprache des vorangegangenen als Eingabesprache besitzt.
Die Komposition der hier behandelten Prozessoren ist daher nicht kommutativ, das heißt, das oben genannte Beispiel schließt die Komposition $i_2 * i_1$ aus.
Bei der Komposition von Prozessoren wird der vorgelagerte *Preprozessor* und der nachgelagerte *Postprozessor* genannt.
Im obigen Beispiel ist i_1 der Preprozessor zu i_2 und i_2 der Postprozessor bezüglich i_1.
Die Anwendung eines Interpreters i_1 auf einen Interpreter i_2 hat die formale Gestalt

$$\overset{i_1}{L_2,L_3,L}\;(\;\overset{i_2}{L_1,L_3,L_2}\;) = w$$

und liefert somit das Interpretationsergebnis w mit $w \in L_3$. Hierbei wird also die Interpretation des Interpreters i_2 durch i_1 beschrieben. Im Gegensatz zur Anwendung eines Compilers auf einen anderen, dessen Ergebnis wiederum ein Compiler ist, ist das Ergebnis w nur im Ausnahmefall selbst wieder ein Interpreter. Hingegen liefert die Anwendung eines Compilers auf einen Interpreter bzw. Generator wieder einen Interpreter bzw. Generator und kann somit, wie bereits oben erwähnt, die Portabilität auch für diese Prozessoren rea-

lisieren. Speziell gilt

$$\underset{L_1,L_2,L}{c}\ (\ \underset{L_3,L_4,L_1}{i}\ ) = \underset{L_3,L_4,L_2}{i'}$$

Wendet man umgekehrt einen Interpreter auf einen Compiler an, also $i(c)=w$, so wird dadurch die unmittelbare Abarbeitung des Compilers c auf einem Computer zum Ausdruck gebracht. Analoges gilt für $i(g)$, also der Abarbeitung eines Generators g durch i. Praktisch irrelevant hingegen sind solche Anwendungen eines Prozessors auf einen anderen, wie $g(c)$ und $g(i)$.

Auf Grund der fundamentalen Bedeutung der genannten Prozessoren gehören diese zu den *Systemprogrammen* (system programs) eines Computers. Die Erstellung und Modifikation dieser Prozessoren bildet den Hauptgegenstand der *Systemprogrammierung* (system programming).

Übungsaufgaben

1.9. Welche Übersetzung wird durch die Compilerfolge

$$\underset{L,L_1,L}{c_1} \ast \underset{L_1,L_2,L_3}{c_2}$$

realisiert?

1.10. Geben Sie eine Komposition für die drei Generatoren g_1, g_2 und g_3 an!

1.11. Schreiben Sie die Anwendungen der genannten Prozessoren auf einen Generator g auf, also $c(g)$, $i(g)$ und $g(g)$, und charakterisieren Sie die jeweiligen Resultate!

1.12. Geben sie eine formale Beschreibung für die Komposition
Quellprogramminterpretation,
Übersetzung,
Programmverbindung,
Abarbeitung des Programms
an, wobei das Programm in der Programmiersprache L_A geschrieben ist!

2. Grundbegriffe der Makrotechnik

2.1. Einführung

Es sei ein einfacher Programmausschnitt in der Programmiersprache PL/1 vorgegeben, und zwar

DO I=1 TO N; S=S+V(I); END; .

Inhaltlich realisieren diese Anweisungen die Summation $S = V_1 + V_2 + V_3 + \ldots + V_N$. Die entsprechenden Anweisungen für die Summation $S = V_K + V_{K+1} + V_{K+2} + \ldots + V_L$ lauten (ebenfalls in PL/1)

DO I=K TO L; S=S+V(I); END; .

Für die Berücksichtigung dieser und weiterer Indizes bei der Summation lautet dieser Programmausschnitt als allgemeineres Schema

DO I=α TO β; S=S+V(I); END; .

Für α und β sind dann die entsprechenden Indexbezeichnungen anzugeben.
Verallgemeinert man auch den Textteil für den Summanden '$V(I)$', um beispielsweise auch $S = V_1W_1 + V_2W_2 + V_3W_3 + \ldots + V_NW_N$ zu realisieren, so lautet schließlich das allgemeine Schema

DO I=α TO β; S=S+γ; END; .

Für ein konkretes Wertetripel (α,β,γ) erhält man dann

DO I=1 TO N; S=S+V(I); END;

für (α,β,γ) = ('1', 'N', '$V(I)$'),

DO I=1 TO N; S=S+V(I)*W(I); END;

für (α,β,γ) = ('1', 'N', '$V(I)*W(I)$').

Es kann aber auch

DO I=K TO L; S=S+V(I)*W(I); END;

für (α,β,γ) = ('K', 'L', '$V(I)*W(I)$')

hergeleitet werden u.v.a.m.
Als weiteres Beispiel sei der folgende Teil eines Programms in der Programmiersprache COBOL gegeben.

```
03 COLUMN 10 PIC 9(4) SOURCE ELEMENT ( 1 ).
03 COLUMN 15 PIC 9(4) SOURCE ELEMENT ( 2 ).
03 COLUMN 20 PIC 9(4) SOURCE ELEMENT ( 3 ).
03 COLUMN 25 PIC 9(4) SOURCE ELEMENT ( 4 ).
03 COLUMN 30 PIC 9(4) SOURCE ELEMENT ( 5 ).
03 COLUMN 35 PIC 9(4) SOURCE ELEMENT ( 6 ).
03 COLUMN 40 PIC 9(4) SOURCE ELEMENT ( 7 ).
03 COLUMN 45 PIC 9(4) SOURCE ELEMENT ( 8 ).
03 COLUMN 50 PIC 9(4) SOURCE ELEMENT ( 9 ).
03 COLUMN 55 PIC 9(4) SOURCE ELEMENT ( 10 ).
```

Ein allgemeineres Schema für diese Anweisungsfolge lautet beispielsweise

```
03 COLUMN δ PIC 9(4) SOURCE ELEMENT (ϱ). ,
```

wobei sich mit $\{(\delta, \varrho)\} = \{(10,1), (15,2), (20,3), \ldots, (55,10)\}$ die obere Zeilenfolge ergibt. Andererseits liefert $\{(\delta, \varrho)\} = \{(18,3), (22,5), (70,6)\}$ die COBOL-Zeilen

```
03 COLUMN 18 PIC 9(4) SOURCE ELEMENT ( 3 ).
03 COLUMN 22 PIC 9(4) SOURCE ELEMENT ( 5 ).
03 COLUMN 70 PIC 9(4) SOURCE ELEMENT ( 6 ).
```

Zur Realisierung dieser Quellprogrammanipulationen wurden zunächst Hilfsmittel in Form von speziellen Anweisungen geschaffen, die der Textsubstitution, der bedingten und zyklischen Textgenerierung dienten. Erstmals wurden derartige Möglichkeiten für die Programmiersprache ASSEMBLER geschaffen und sind dort unter dem Begriff »bedingte Assemblierung« (siehe Abschnitt 3.) bekannt.
Die Anwendung der Unterprogrammtechnik für diese Hilfsmittel führte zur Realisierung eines Makros. Angewandt auf die oben genannten Schemata sind dann α, β, γ und δ, ϱ die jeweiligen Parameter.
Das Wort *Makro* (macro) hat seinen Ursprung im Griechischen und bedeutet dort als »makros« (μακρος) soviel wie »weit« und »groß« (im Gegensatz zu »mikros«). Im übertragenen Sinne soll hier ein Makro eine gewisse Weite und Größe innerhalb einer Programmiersprache (zum Beispiel als Zusammenfassung von Anweisungen) zum Ausdruck bringen.
Die Implementation von Makros hat dabei neben den bisher gezeigten formalen Möglichkeiten vor allem einen praktischen Ausgangspunkt. Es geht darum, ähnliche Anweisungsmengen oder -folgen in verschiedenen Programmen rationeller und sicherer niederzuschreiben.
In diesem Sinne realisiert das erste oben genannte Schema eine variable Nutzung ein und derselben Anweisungsstruktur, während das zweite Beispiel insbesondere das wiederholte Schreiben gleichartiger Anweisungen rationalisiert.

2.2. Makrogrammatiken und Basissprachen

Im folgenden sollen die Art und Weise und die Formen derartiger Makros, wie sie sich in der sogenannten Makrogrammatik darstellen, formal angegeben werden. Ausgangspunkt ist die in Abschnitt 1. definierte Indexgrammatik. Dabei werden die indizierten Nichtterminale zu parametrisierten erweitert.
Eine *Makrogrammatik* (macro grammar) G_M ist das Quintupel

$$G_M = (A,H,\{p_1,p_2,p_3,\cdots,p_n\},S,P) \quad \text{mit}$$

- A terminales Alphabet,
- H Menge der *Makronamen* (macro names) mit ihren *Argumenten* (arguments) (auch Menge der *Makroaufrufe* (macro calls) genannt),
- $\{p_1,p_2,p_3,\cdots,p_n\}$ Menge der formalen Parameternamen,
- S *Anfangsmakro* (initial macro) ohne Parameter,
- P Menge der Regeln (*Makrodefinitionen* (macro definitions) oder kurz *Makros*) in der Form

  $$m(p_1,p_2,p_3,\cdots,p_k) \to Q$$

  mit $k \geqq n$, m als Makroname und Q als Zeichenkette, die durch Substitution und Konkatenation von Ausdrücken entsteht; die Ausdrücke können dabei formale Makroaufrufe oder sogenannte Elementarausdrücke sein, das heißt, der Menge $\{p_1,p_2,p_3,\cdots,p_n\} \cup A \cup \{\varepsilon\}$ angehören.

Die durch G_M erzeugte Sprache $L(G_M)$ wird *Makrosprache* (macro language) genannt.
G_M ist eine *einfache Makrogrammatik* (basic macro grammar), wenn für die Parameter $p_1,p_2,p_3,\cdots,p_n$ als aktueller Wert kein Makroaufruf zugelassen ist, sonst bezeichnet man G_M als *erweiterte Makrogrammatik* (extended macro grammar).
Ein Beispiel für eine einfache Makrogrammatik ist

$$G_{M_1} = (\{a,b\},\{F(p_1,p_2),S\},\{p_1,p_2\},S,P_{M_1}) \quad \text{mit}$$

$$P_{M_1} = \{S \to F(a,b),\ F(p_1,p_2) \to aF(p_1,p_2)b,$$

$$F(p_1,p_2) \to p_1p_2\}.$$

Eine Ableitung des Wortes *aaaabbbb* lautet in G_{M_1}

$$S \xrightarrow{(1)} F(a,b) \xrightarrow{(2)} aF(a,b)b \xrightarrow{(2)} aaF(a,b)bb$$

$$\xrightarrow{(2)} aaaF(a,b)bbb \xrightarrow{(3)} aaaa,bbbb,$$

wobei die Nummern wiederum die Anwendung der jeweiligen Regel (fortlaufend gezählt) aus P_{M_1} kennzeichnen.
G_{M_1} erzeugt die Basissprache $L(G_{M_1}) = \{a^nb^n \mid n \geqq 2\}$.

Ebenfalls eine einfache Makrogrammatik, die das erste Beispiel aus 2.1. realisiert, lautet

$$G_{M_2} = (\{\text{DO } I=,;S=S+,;\text{END};von,bis,summand\},$$

$$\{F(p,q,r),S\},\{p,q,r\},S,P_{M_2}) \quad \text{mit}$$

$$P_{M_2} = \{S \to F(von,bis,summand),\ F(p,q,r) \to \text{DO } I=p \text{ TO } q;S=S+r;\text{END};\}.$$

Eine vollständige (hier einzige) Ableitung in G_{M_2} lautet dann beispielsweise

$$S \to F(von,bis,summand) \to \text{DO } I=von \text{ TO } bis;S=S+summand;\text{END};$$

Das folgende Beispiel zeigt eine erweiterte Makrogrammatik. Sie habe die Gestalt

$$G_{M_3} = (\{\text{IF,THEN,ELSE},A,\&,|,A1,A2\},\{F(p_1,p_2,p_3),G(p_1),S\},$$

$$\{p_1,p_2,p_3\},\ S,P_{M_3}) \quad \text{mit}$$

$$P_{M_3} = \{S \to F(G(A),A1,A2),\ F(G(p_1),p_2,p_3) \to$$

$$\text{IF } G(p_1) \text{ THEN } p_2 \text{ ELSE } p_3,\ G(p_1) \to p_1,\ G(p_1) \to p_1\&p_1,$$

$$G(p_1) \to p_1|p_1\}.$$

Eine vollständige Ableitung in G_{M_3} ist dann

$$S \to F(G(A),A1,A2) \to \text{IF } G(A) \text{ THEN } A1 \text{ ELSE } A2$$

$$\to \text{IF } A\&A \text{ THEN } A1 \text{ ELSE } A2.$$

Auf eine Programmiersprache bezogen, erzeugt das Makro $F(p_1,p_2,p_3)$ die bedingte Anweisung 'IF ... THEN ... ELSE ...' und das Makro $G(p_1)$ einfache logische Ausdrücke, wobei '&' für das logische UND und '|' für das logische Oder stehen. Für A ist jeweils ein Variablenname und für $A1$ und $A2$ sind Anweisungen einzusetzen.

Übungsaufgaben

2.1. Bestimmen Sie die durch die Makrogrammatik

$G_{M_4} = (\{a,b,c\},\{T(p,q),R(p),S\},\{p,q\},S,P_{M_4})$ mit

$P_{M_4} = \{S \to T(a,c),T(p,q) \to T(p,R(q)),$

$R(p) \to ap,T(p,q) \to bq\}$

generierte Makrosprache $L(G_{M_4})$!

2.2. Definieren Sie eine einfache Makrogrammatik, die die Sprache $L(G_{M_5}) = \{ab^n \mid n \geq 1\}$ erzeugt!

2.3. Geben Sie für das zweite Beispiel eines Anweisungsschemas in 2.1. eine Makrogrammatik an!

2.3. Makrosprachen

2.3.0. Einleitende Bemerkungen

Die Menge aller Makrodefinitionen (Makros) aus der Regelmenge P_M einer Makrogrammatik G_M nennt man *Makrosprache*[1]) (macro language). Eine Folge von Anweisungen einer Makrosprache bezeichnet man als *Makroprogramm* (macro program). Eine Programmiersprache, deren Anweisungen ausschließlich Makroaufrufe darstellen, wird *makroorientierte Sprache* (macro oriented language) genannt.
Ein Makro hat, wie oben definiert, die allgemeine Form

$$m(p_1,p_2,p_3,\ldots,p_k) \rightarrow Q.$$

Formal gliedert es sich also in die Bestandteile '$m(p_1,p_2,p_3,\ldots,p_k)$' und '$\rightarrow Q$'. Durch den ersten Teil wird der Makroaufruf *definiert.* Dieser Teil eines Makros wird daher *formaler Makroaufruf* genannt. Der Pfeil drückt die jeweilige konkrete Umsetzung des Makroaufrufes in den durch das Makro erzeugten Text Q aus. Die formale Sprache, die den generierten Text Q zusammenfaßt, heißt *Basissprache* (basic language). Dabei kann Q selbst ebenfalls Makros einer anderen Makrosprache als Terminale enthalten.
Nach den Formen und Mitteln, die diesen Makrobestandteilen zugrunde liegen, werden die Makros klassifiziert.
Im folgenden soll auf die Formen und Mittel der bekanntesten bisher existierenden Makrosprachen näher eingegangen werden.

2.3.1. Definition des Makroaufrufes

Hierbei handelt es sich um den ersten Teil der Makrodefinition, also

$$m(p_1,p_2,p_3,\ldots,p_k) \rightarrow Q.$$

Diese Beschreibung des Makroaufrufes ist jedoch eine funktionelle Darstellung, das heißt, es kommt nur zum Ausdruck, daß ein Makro den Namen m und die Menge formaler Parameter $p_1,p_2,p_3,\ldots,p_k$ besitzt. Die konkrete Darstellungsform in implementierten Makrosprachen gliedert sich gemäß ihrer syntaktischen Form in drei Hauptgruppen:

(1) den Makroaufruf mit *einfachem Makronamen* (single macro name), wie beispielsweise

$m(\mathrm{par}_1,\mathrm{par}_2,\mathrm{par}_3,\ldots,\mathrm{par}_k)$, $k \geqq 1$,

$m\ \mathrm{par}_1,\mathrm{par}_2,\mathrm{par}_3,\ldots,\mathrm{par}_k$ oder speziell

m (ohne Parameter);

(2) den Makroaufruf mit *verteiltem Makronamen* (distributed macro name), wie

$$m_1\ \mathrm{par}_1\ m_2\ \mathrm{par}_2\ m_3\ \mathrm{par}_3\ \ldots\ m_k\mathrm{par}_k$$

mit $m = m_1m_2m_3\ldots m_k$;

(3) den Makroaufruf mit in der jeweiligen Makrosprache wählbarer syntaktischer Form.

---

[1]) Eine derartige Definition einer Makrosprache bekennt sich zu den *sprachlichen* Mitteln bei der Makroprogrammierung im Gegensatz zu BROWN [25], der ausschließlich den Begriff Makroprozessor dafür zuläßt.

Für die Makroaufrufdefinition nach (1) lauten spezielle Beispiele von Makroaufrufen

MAKRO1(A,B,C) ,

MAKRO2(X1,Y1) ,

MAKRO3 A,24,B-C ,

MAKRO4 .

Konkrete Makroaufrufbeispiele für die Definition nach (2) sind

for $k=1$ **to** $n+1$ **do** $j=j+m[k]$,

wobei **for to do** der verteilte Makroname und '$k=1$', '$n+1$' und '$j=j+m[k]$' die aktuellen Parameterwerte sind;

$m_1 \ast\ast m_2 \ast m_3 \ast$,

$m_1 < p_1 > m_2 < p_2 > m_3 < p_3 >$.

Konkrete Beispiele für eine wählbare Aufrufform sind insbesondere in 2.3.2.3. bei der Darstellung vollständiger Makroabarbeitungen gegeben.

2.3.2. Definition der Zeichenkettengenerierung

Im folgenden soll auf den Teil der Makrodefinition eingegangen werden, der in der Regel der Makrogrammatik kurz mit '$\rightarrow Q$' bezeichnet wurde und allgemein *Makrokörper* (macro body) genannt wird, also

$$m(p_1,p_2,p_3,\cdots,p_k) \rightarrow Q.$$

Wie sich dabei zeigen wird, ist dies zumeist der umfangreichste Teil einer Makrodefinition. Durch die konkrete Makrosprache sind eine ganze Reihe von Möglichkeiten gegeben, aus dem formalen Makroaufruf die Ergebniszeichenkette Q zu generieren. Dabei soll zunächst die Art und Weise der Nutzung der Parameterwerte näher betrachtet werden.

2.3.2.1. Methoden der Makroparameterinterpretation

Bezüglich der Identifizierung des Parameters im Makroaufruf unterscheidet man *Stellungsparameter* (positional parameter) und *Kennwortparameter* (keyword parameter). Beim Stellungsparameter wird dieser in der Makrodefinition gemäß seiner Position in der Parameterliste identifiziert. Ist beispielsweise der Wert des dritten Parameters eines Makros MAKRO5 'X' und sind die anderen Parameterwerte leer, so lautet der entsprechende Makroaufruf MAKRO5 ,,X . Das Komma ist hierbei das Trennzeichen zwischen den Parameterwerten.
Bei der Realisierung eines Kennwortparameters erhält der formale Parametername eine spezielle Kennzeichnung. Dieser Parametername ist beim Makroaufruf wieder zu verwenden. Dabei spielt jedoch die Reihenfolge der anzugebenden Parameternamen keine Rolle. Lautet eine Definition des Makroaufrufes beispielsweise

MAKRO6 A=,B=,C=

mit '=' als Kennwortparameterkennzeichnung, so lautet ein für dieses Makro gültiger Aufruf

MAKRO6 C=3,A=12,B=7 .

Hinsichtlich der Wertübernahme des (aktuellen) Parameterwertes eines Makroaufrufes unterscheidet man

(a) den *Namensaufruf* (call by name), wobei
(a1) der aktuelle Parameter Name einer Variablen ist, der an allen Stellen des Auftretens des formalen Parameters im Makrokörper eingesetzt wird;
(a2) der aktuelle Parameter Name eines Makros ist und analog (a1) verwendet wird;
(b) den *Wertaufruf* (call by value), wobei
(b1) der durch den aktuellen Parameter unmittelbar gegebene Wert im Makrokörper verwendet wird;
(b2) der Wert, der sich nach Abarbeitung eines als aktuellen Parameter verwendeten Makroaufrufes ergibt, in den Makrokörper eingeht.

Die Parameteraufrufformen (a2) und (b2) kennzeichnen gemäß ihren Definitionen eine erweiterte Makrogrammatik. Die Aufrufform (b2) wird *statische Makroerweiterung* (static macro expansion) und die Form (a2) *dynamische Makroerweiterung* (dynamic macro expansion) genannt.
Ist für einen formalen Parameter im Makroaufruf eine Liste von Unterparametern in der jeweiligen Makrosprache zugelassen, also

$$p_i = (q_1,q_2,q_3,\cdots,q_m) \text{ mit } m \geqq 2 \text{ und } 1 \geqq i \geqq k,$$

so ergibt sich die Notwendigkeit, diese Parameter der Reihe nach dem Makrokörper zur Verfügung zu stellen. Dafür existieren in speziellen Makrosprachen die Funktionen

(p1) Bereitstellung des j-ten Unterparameterwertes aus p_i, $1 \geqq j \geqq m$,
(p2) Bestimmung der Anzahl m der Unterparameterwerte von p_i.

Eine spezielle Lösung besteht bei der Unterparameterwertbereitstellung im indizierten Aufruf, also $W(5)$ für die Bereitstellung des fünften Parameterwertes des Parameters W (siehe 3.4.).
So wie die Parameter als Liste interpretiert (zerlegt) werden können, ist es ebenso auch möglich, sie im Makrokörper zusammenzufassen, zu verketten.
Im Gegensatz zu Unterprogrammkonzepten in einigen Programmiersprachen ist es bei den Makrosprachen im allgemeinen zulässig, daß die aktuelle Parameterliste kürzer als die formale ist. Zu beachten ist dabei jedoch, daß beim Weglassen eines Stellungsparameterwertes innerhalb der Folge durch ein zusätzliches Trennzeichen (zumeist ein Komma) die Parameterposition entsprechend den formalen Parametern gewährleistet ist. Als Beispiel hierfür wurde bereits oben der Makroaufruf MAKRO5 „X angegeben.
Dafür, daß jedoch ein aktueller Parameterwert leer sein kann, muß im Makrokörper eine korrekte Verarbeitung abgesichert sein. In den verschiedenen Makrosprachen sind hierzu folgende Möglichkeiten vorhanden:

(p3) automatische Generierung eines Parameterwertes als Null oder als Bezeichnung (Alphazeichen gefolgt von einer laufenden Nummer),
(p4) Anwendung einer logischen Funktion, die bei vorhandenem aktuellem Parameterwert den Wert »wahr« und sonst den Wert »falsch« hat.

Ist im Makrokörper ebenfalls ein Makroaufruf enthalten, so handelt es sich auch um eine statische Makroerweiterung.

2.3.2.2. Methoden der Zeichenkettenmanipulation

Im folgenden werden die realisierbaren Variablenarten und die möglichen Operationen über diese Variablen in den Makrosprachen zusammengefaßt. Alle diese Möglichkeiten sind kaum in einer einzelnen Makrosprache gegeben, wie auch aus den konkreten Beispielen in 2.3.2.3., die jeweils einige Eigenschaften der Makrosprache zeigen, zu ersehen ist.
Folgende Variablen (bezüglich ihrer Gültigkeit bei der Makroabarbeitung auch als temporäre Variablen bezeichnet) sind definierbar:

(v1) Zeichenkettenvariablen,
(v2) numerische Variablen,
(v3) logische Variablen [mit den Wahrheitswerten »wahr« (als 'true' oder '1') und »falsch« (als 'false' oder '0')],
(v4) indizierte Variablen [als Elemente eines Feldes (Vektors, Matrix usw.)],
(v5) Kellervariablen [als Variablen, in denen der Reihe nach Werte eingespeichert werden, die in umgekehrter Reihenfolge nacheinander (einzeln) gelesen werden können].

Über diese Variablen besteht die Möglichkeit der Anwendung folgender Operationen:

(o1) Verketten zweier Zeichenketten, bei dem ein spezielles Verkettungszeichen anzuwenden ist,
(o2) arithmetische Operationen,
(o3) logische Verknüpfung von logischen Variablen bzw. von Vergleichen mittels dem logischen Und, dem logischen Oder und der Negation zu logischen Ausdrücken,
(o4) Aufruf eines Feldelementes durch Angabe des Elementnamens und des Indexwertes,
(o5) Einspeichern und Lesen aus Kellervariablen.

Weiterhin besteht in einigen Makrosprachen die Möglichkeit, folgende Zeichenkettenverarbeitungsfunktionen anzuwenden:

(f1) Bestimmung der Länge (Anzahl der Zeichen) einer Zeichenkette,
(f2) Prüfung, ob eine Zeichenkette in einer anderen enthalten ist,
(f3) Bildung einer Teilzeichenkette durch Angabe der Länge und der Position, ab der diese Teilzeichenkette aus einer anderen zu bilden ist,
(f4) Generierung einer Zahl als Zeichenkette, die sich bei jedem Aufruf dieser Funktion um eins erhöht.

Die jeweiligen Variablen erhalten ihre Werte durch einfache Zuweisungen. Die Möglichkeit sogenannter Ein- oder Ausgabeanweisungen ist in den Makrosprachen i. allg. nicht gegeben. Durch diese Zuweisungen ist aber zunächst nur eine sequentielle Zeichenkettenerzeugung realisierbar. Für die bedingte Zeichenkettengenerierung besteht die Möglichkeit der Anwendung bedingter Anweisungen auf der Basis von logischen Ausdrücken oder einfach nur von Vergleichen. Die zyklische Zeichenkettenerzeugung wird in den Makrosprachen realisiert durch

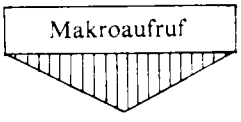
- die Anwendung von Markierungen und Sprunganweisungen zu den jeweiligen Marken und/oder
- die Anwendung geschlossener zyklenorganisierender Anweisungen.

Schließlich sind diese Anweisungen auch zu Blöcken oder Unterprogrammen (den eigentlichen Makros) zusammenfaßbar.
Im folgenden Abschnitt soll die oben gegebene verbale Zusammenfassung der Möglichkeiten der Zeichenkettenmanipulation durch Makrosprachen an konkreten Beispielen gezeigt werden.

2.3.2.3. Makrobeispiele in ausgewählten Makrosprachen

Bei der Darstellung der Beispiele werden vollständige Makrodefinitionen, das heißt, sowohl die Definition des Makroaufrufes als auch des Makrokörpers, angegeben, sowie durch ein Beispiel eines Makroaufrufes das Abarbeitungsergebnis des jeweiligen Makros gezeigt.
Die allgemeine Darstellungsform für die Beispiele lautet

Makroprogramm

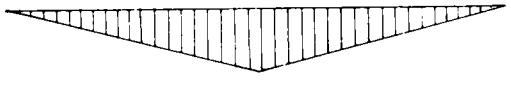


generierte Zeichenkette.

Bei der Anwendung der Anweisungen einer Makrosprache zur Zeichenkettengenerierung ohne Unterprogramm (also ohne das eigentliche Makro) entfällt der Makroaufruf.
Ein einfaches Beispiel einer Zeichenkettengenerierung lautet in der Makrosprache von GPM (General Purpose Macrogenerator) [125]

Beispiel 1

```
§DEF,IF,WENN;
§DEF,THEN,DANN;
§DEF,ELSE,SONST;
§IF; A > B §THEN; X=2; §ELSE; X=-X;
```

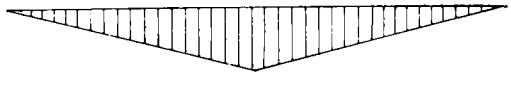


```
WENN A > B  DANN  X=2;  SONST  X=-X;
```

Hierbei werden mittels §DEF die Zeichenkettenvariablen IF, THEN und ELSE mit den jeweiligen Werten 'WENN', 'DANN' und 'SONST' definiert. Bei ihrer Verwendung im Text werden sie mit dem Zeichen '§' gekennzeichnet und einem Semikolon abgeschlossen. Bei der Verarbeitung durch GPM werden für diese Variablen an den jeweiligen Stellen ihrer Anwendung ihre Werte eingesetzt. Eine Zeichenkettengenerierung, bei der der syntaktische Aufbau des Eingabetextes transformiert wird, zeigt das folgende in der Makrosprache KATE (Kent Algorithm for Transformation and Evaluation) [129] gegebene

Beispiel 2

```
.CREATE. = AS EQUATE('#1', '#2'):
.CREATE. + AS SUM ('#1', '#2'):
.CREATE. - AS DIFF ('#1', '#2'):
.CREATE. * AS MULT ('#1', '#2'):
.CREATE. DIV AS DIV('#1'):
NEWRHO = RHO - DT * DIV[RHO + V]:
```

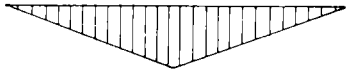


```
EQUATE(NEWRHO,DIFF(RHO,MULT(DT,DIV(SUM(RHO,V))))).
```

Nach '.CREATE.' folgt der zu transformierende Operator ('=', '+' usw.), und nach 'AS'('als') wird das Transformationsergebnis beschrieben. Dabei kennzeichnen '#1' und '#2' die jeweiligen Bezeichnungen vor und nach dem Operator im Eingabetext.
Eine zyklische Textgenerierung in der Makrosprache MACTRAN (**Mac**ro-FOR**TRAN**) [109] hat beispielsweise die Form

Beispiel 3

```
      %INTEGER J;
      %DO 5 J=2,4;
        Z(%CALL PUTI(J,1);)=X(%CALL PUTI(J,1);)+1;
%5    CONTINUE
```

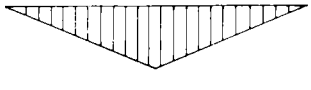


```
      Z(2)=X(2)+1;
      Z(3)=X(3)+1;
      Z(4)=X(4)+1;
```

Durch '%INTEGER' wird die ganzzahlige Variable *J* definiert. Die %DO-Anweisung kennzeichnet den Zyklus, bei dem die Variable *J* die Werte von 2 bis 4 (also 2, 3, 4) annimmt. Die Zahl 5 dient der Markierung der letzten Anweisung des Zyklus (hier die sogenannte Leeranweisung 'CONTINUE'). Die mit CALL aufgerufene Funktion PUTI liefert als Resultat jeweils die Zahl *J* als Zeichenkette der Länge 1.
Das folgende Beispiel in der Makrosprache des SP/1 (FORTRAN integrated string **p**rocessor **1**) [92] transformiert algebraische, geklammerte Ausdrücke in der gewöhnlichen (Infix-) Schreibweise in die sogenannte Polnische Notation, das heißt, in eine Form, bei der der (dyadische) Operator jeweils vor den beiden zu verknüpfenden Operanden steht.

Beispiel 4

```
%OP → "**" & "*" & "/" & "+" & "-"
1 E:  "(" & *(U)* & %OP & *(V)* & ")"
        → %OP & U & V GOTO 1
      ((A*B)+(C/D-E)))
```



```
      + *AB/C-DE
```

Hierbei definiert die erste Zeile den Wertevorrat für die Zeichenkettenvariable OP (also **, *, /, +, -), wobei '&' für 'und' steht. '1E:' markiert die Transformationsanweisung, die durch 'GOTO 1' als Sprunganweisung zyklisch abgearbeitet wird. *U* und *V* kennzeichnen die beiden Operanden. Die Transformation wird durch den Pfeil ausgedrückt.
Die Makrosprache COBRA ([13]) gestattet die Verwendung sogenannter Register (mit *@nummer* bezeichnet) für die Speicherung von maximal 30 Zeichen langen Zeichenketten. Mit einem (angenommenen) Inhalt für den globalen Parameter %1 als Zeichenkette 'ANFANG' ergibt das COBRA-Programm

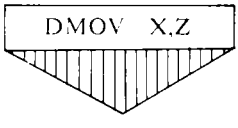
Beispiel 5

```
$MOVE %1    TO @102
$CHAR @102  TO @101
$CHAR @102  TO @103
$TAG  @103  TO @101
```

für @101 den Inhalt 'AN', für @102 die Zeichenkette 'ANFANG' und für @103 den Inhalt 'N'. Die Funktion $CHAR stellt der Reihe nach jeweils ein Zeichen aus %1 bereit, während $TAG der Verkettung zweier Registerwerte dient.
Die Verwendung einer einfachen Makrodefinition zeigt das folgende Beispiel in MACRO-11 ([41]).

Beispiel 6

```
.MACRO DMOV,A,B
MOV    A,B
MOV    A+2,B+2
.ENDM
```

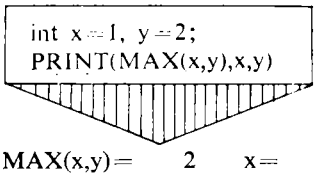



```
MOV    X,Z
MOV    X+2,Z+2
```

Durch .MACRO wird die Makrodefinition mit dem Namen DMOV und den formalen Parametern A und B eingeleitet und durch .ENDM beendet. Die Wirkung besteht in der einfachen Ersetzung der im Makrokörper angegebenen formalen Parameter durch die aktuellen Parameterwerte.
Eine verschachtelte Makrodefinition soll ein Beispiel in der Preprozessorsprache der Programmiersprache C ([84]) zeigen.

Beispiel 7

```
#define PR(a)   printf("a=%d\t",(int) (a))
#define PRT(a)    PR(a);putchar('\n')
#define PRNT(a,b) PR(a);PRT(b)
#define PRINT(a,b,c) PR(a);PRNT(b,c)
#define MAX(a,b)  (a > b ? b:a)
```



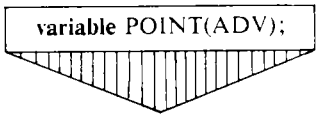

```
MAX(x,y)=    2    x=    1  y=    2
```

Dabei bezieht sich jeweils die nachfolgende Makrodefinition auf die unmittelbar vorhergehende. Die Textgenerierung ist stets noch in derselben Zeile angegeben.

Das folgende Beispiel ist in der Makrosprache POLYP (**P**roblem **O**riented **L**anguage for System Software **P**rogramming) [100] geschrieben.

Beispiel 8

```
cproc ADV  $
      cbegin
      A24,B(32,8)
      cend $
```

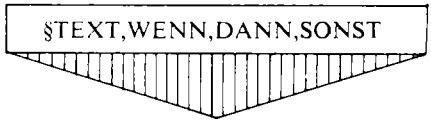


variable POINT(A24,B(32,8));

Der Beginn der Makrodefinition ist durch 'cproc' gekennzeichnet. Der Name des parameterlosen Makros ist ADV. Der Makrokörper ist durch 'cbegin' und 'cend' eingeklammert und liefert als Resultat stets die Zeichenkette 'A24,B(32,8)'. Der Makroaufruf ist in einer Zeichenkette eingebettet und bewirkt eine einfache Zeichenkettensubstitution.
Ein parametrisierter Makroaufruf in der Makrosprache von GPM hat die Form

Beispiel 9

```
§DEF,TEXT, ~1 A > B ~2 X=2; ~3 X=-X;;
```



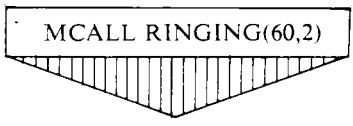
WENN A > B DANN X=2; SONST X=-X;

Das Zeichen '~' kennzeichnet die fortlaufend numerierten formalen Parameter des Makros TEXT. Beim folgenden (Wert-) Aufruf werden diese formalen Parameter der Reihe nach durch die aktuellen Werte 'WENN', 'DANN' und 'SONST' ersetzt.
Eine besondere Art der Makroaufrufdefinition zeigt das folgende Beispiel in der Makrosprache SL1 (**S**pecification and Design **L**anguage **1**) [57]. Hierbei werden die Parameter implizit definiert, das heißt durch ihre Verwendung im Makrokörper, und sind explizit im Makroaufruf mit ihren konkreten Werten anzugeben. Das Zeichen '%' kennzeichnet hier die wiederum fortlaufend numerierten formalen Parameter. Während die Parameteranwendung '%2' in der 6. Zeile der Makrodefinition nur eine Wertersetzung bewirkt, wird durch '%1/(%2+1)' als arithmetischer Ausdruck erst eine Berechnung realisiert und dann der erhaltene Wert an dieser Stelle eingesetzt, also 60/(2+1)=20.

Beispiel 10

```
MACBEGIN RINGING;
    DO %1/(%2+1);
    START-RINGING;
    WAIT 1S,100MS,STATE RINGING;
```

```
    STOP-RINGING;
    WAIT %2S,1000MS,STATE RINGING;
    OD;
MACEND;
```

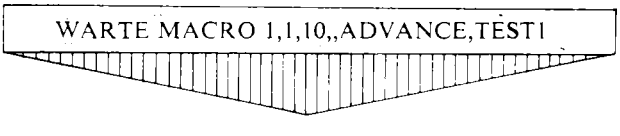



```
DO 20;
START-RINGING;
WAIT 1S,100MS,STATE RINGING;
STOP-RINGING;
WAIT 2S,1000MS,STATE RINGING;
OD;
```

Durch 'MACBEGIN' ist der Anfang und durch 'MACEND' das Ende des Makros RINGING gekennzeichnet.
Analog verhält es sich bei der Makrodefinition in der Programmiersprache SIMDIS-2 ([15]), die zur Simulation von Prozessen dient.
Dabei sind die formalen Parameter mit dem Zeichen '#' gekennzeichnete Buchstaben, deren aktuelle Werte gemäß ihrer alphabetischen Ordnung anzugeben sind. Weiterhin ist im Makroaufruf das Weglassen des aktuellen Wertes für den Parameter #D zu erkennen, das die Wertersetzung und das vor dem Parameter stehende Komma unterdrückt.

Beispiel 11

```
WARTE STARTMACRO
#F      QUEUE       #A
        SEIZE       #B
        DEPART      #A
        #E          #C,#D
        RELEASE     #B
        ENDMACRO
```

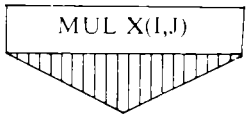



```
TEST1   QUEUE       1
        SEIZE       1
        DEPART      1
        ADVANCE     10
        RELEASE     1
```

Das folgende Beispiel ist in der Makrosprache SIXTRAN (**SIX**-language FOR**TRAN** based) [34] geschrieben und lautet

Beispiel 12

```
#'MUL @;' = ' @1 * @1';
```

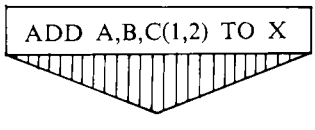



```
X(I,J) = X(I,J) *X(I,J)
```

Das Zeichen '@' kennzeichnet den formalen Parameter, dessen aktueller Wert auch eine Werteliste sein kann. '@1' liefert den ersten (hier einzigen) Wert von @.
Eine andere Form der Auswertung einer aktuellen Parameterwerteliste zeigt das folgende Beispiel in der Makrosprache des MP/1 (FORTRAN Macroprocessor 1) [91].

Beispiel 13

```
%DEF    ADD @(,) = '1' @ TO @ @
%       CALL ARGS( @1 @,#LV)
%       #LW=0
% 10    #LW =#LW + 1
%        @2 @ = @2 @ + @1:#LW @
%       IF (#LW.LT.#LV)  GOTO  10
%END
```




```
X = X + Y
X = X + B
X = X + C(1,2)
```

Das Makro ist durch die Anweisungen '%DEF' und '%END' eingegrenzt. Die verteilten Makronamen lauten ADD und TO. Durch '#' sind die arithmetischen Variablen im Makrokörper gekennzeichnet. Sie dienen der Auswertung der formalen Parameter @1 @ und @2 @. Die Funktion ARGS dient der Ermittlung der Anzahl aktueller Parameterwerte. Sie wird durch 'CALL' aufgerufen, bestimmt hierbei diese Anzahl aus @1 @ und weist sie der Variablen #LV zu. Durch '@1:#LW @' wird der #LW-te Parameterwert von @1 @ bestimmt und an dieser Stelle eingesetzt.
Der Ausdruck '@(,)='1'@' im formalen Makroaufruf drückt aus, daß für einen leeren Parameterwert eine Eins verwendet wird, das heißt, das Generierungsergebnis lautet dann dafür 'X = X + 1'. Der Makrokörper im Beispiel 13 realisiert durch die Sprunganweisung 'GOTO 10' und der Markierung mit '10' einen Zyklus. Das Ende des Zyklus wird erreicht, wenn in der bedingten Anweisung 'IF ...' der logische Ausdruck (verbal #LW < #LV, also 'verwendeter Parameterwert nicht der letzte') nicht erfüllt ist. Ebenfalls eine Parameterlistenauswertung zeigt das folgende Makrobeispiel in ALEC (**A** Language with an **E**xtensible **C**ompiler) [104].

Beipiel 14

```
        CLASS
[EXPRESSION LIST] = LIST OF [EXPRESSION], SEPERATED BY,
        ROUTINE
[OPEN ALEC PIECE] :
SET ARRAY [IDENTIFIER a]=[EXPRESSION LIST b];
BEGIN MACRO BLOCK
CALL ARRAY A =[IDENTIFIER a] BY TRSUBST
index = 0
FOR EACH i ON LIST [EXPRESSION LIST b]:
{index = index + 1
VALUE OF [INTEGER p] = index
COMPILE: A([INTEGER p]) = [EXPRESSION b(i)];}
END MACRO BLOCK
```

```
SET ARRAY STARTING VALUE = 25,B-3.142,0,(X+Y)/SQRT(X*Y);
```

```
BEGIN;
STARTING VALUE(1) = 25;
STARTING VALUE(2) = B-3.142;
STARTING VALUE(3) = 0;
STARTING VALUE(4) = (X+Y)/SQRT(X*Y);
END;
```

Nach **CLASS** wird hierbei der formale Makroaufruf beschrieben. Seine syntaktische Form ist nach den Bezeichnungen '**SET ARRAY**' angegeben, und zwar als 'bezeichnung' = 'liste von ausdrücken' ('IDENTIFIER' und 'EXPRESSION LIST'). Nach der Anfangskennzeichnung 'BEGIN MACRO BLOCK' wird die Zeichenkettengenerierung im Makrokörper beschrieben. Dabei erhält die Zeichenkettenvariable *A* den Wert der 'bezeichnung'. Die eigentliche Textgenerierung wird in der Zeile 'COMPILE: ...' beschrieben. Der Makroname lautet **SET ARRAY**. Die Eingrenzungen '**BEGIN**;' und '**END**;' werden hierbei automatisch erzeugt.

Ein Beispiel für eine nicht explizit begrenzte Makrodefinition in der Makrosprache des LIMP (Language Independent Macro Processor) [137] hat die Form

Beispiel 15

```
* = * + *
END PU = 'FETCH,'20 '
ADD,'30 '
STORE,'10 /
```

```
WERT = WERT1 + WERT2
```

```
FETCH,WERT1
ADD,WERT2
STORE,WERT
```

Die mit '*' bezeichneten formalen Parameter werden im Makrokörper durch '10, '20 und '30 gemäß der Reihenfolge ihres Auftretens ausgedrückt. Die Funktion 'END PU' (**pun**ching) beschreibt die spezielle Ausgabeweise des generierten Textes. Dem formalen Makroaufruf '* = * + *' können dabei noch weitere Funktionen, die eine spezielle Generierungsform beinhalten, folgen.
Ein ähnliches Beispiel ist das folgende in der Makrosprache des SIMCMP (**Simp**le **Com**piler) [108]. Dabei wird eine Funktion zur Generierung einer fortlaufenden Zahl bei jedem Aufruf dieser im Makrokörper in Form des Zero-Parameters ('00) verwendet. Die formalen Parameter sind im formalen Makroaufruf wiederum durch das Symbol '*' dargestellt. Im Makrokörper lauten sie '10, '20, '31 und '41. Die spezielle Darstellung des dritten und vierten formalen Parameters mit einer Eins bewirkt die numerische Umsetzung des aktuellen (hier alphabetischen) Parameterwertes.

Beispiel 16

```
TO * * IF CAR *= CDR *.
        I = CDR('31).
        J = CDR('41).
        IF(CAR(I)-CDR(J))'00,'10'20,'00.
'00     CONTINUE
.X
```

```
TO  1 4 IF CAR B = CDR A.
```

```
        I = CDR(38)
        J = CDR(36)
        IF(CAR(I)-CDR(J))101,14,101
101     CONTINUE
```

Die Aneinanderreihung der ersten beiden formalen Parameter im Makrokörper ('10'20) bewirkt eine Verkettung der aktuellen Werte. Das Beispiel 16 zeigt wiederum die Anwendung von verteilten Makronamen 'TO', 'IF CAR' und '= CDR'.
Ein Beispiel für die syntaktische Definition des Makroaufrufes ist im folgenden in der Makrosprache ML/1 (**M**acro **L**anguage **1**) [30] gegeben. Diese Makroaufrufdefinition wird durch 'MCDEF' eingeleitet und beschreibt hier die beiden möglichen syntaktischen Formen 'IF...=...THEN ...ELSE...END' oder 'IF...=...THEN...END'. 'OPT' stellt bereits einen formalen Parameter dar. Durch die Zeichen '<' und '>' wird der Makrokörper eingegrenzt. Hier werden die weiteren formalen Parameter A1, A2, A3 und A4 verwendet. Der Makroaufruf hat also für die erste syntaktische Form die allgemeine Darstellung 'IF A1=A2 THEN OPT A3 ELSE A4 END'.

Beispiel 17

```
MCDEF  IF = THEN OPT ELSE END OR END ALL AS
<       LAC  :A1
        SAD  :A2
        SKP
        JMP XX:T2.
        :A3.
MCGO.L1 UNLESS :D3.=ELSE;
        JMP  YY:T2.
```

```
XX:T2.,:A4.
YY:T2.,MCGO.L2;
:L1.XX:T2.,:L2.>;
```

```
IF A = B THEN JMS SUB
END
IF PIG = DOG THEN LAC C
                  DAC D
ELSE LAC Y
     DAC Z
END
```

```
        LAC  A
        SAD  B
        SKP
        JMP  XX1
        JMS  SUB
XX1,    LAC  PIG
        SAD  DOG
        SKP
        JMP  XX2
        LAC  C
        DAC  D
        JMP  YY2
XX2,    LAC  Y
        DAC  Z
YY2,
```

Im Makrokörper kennzeichnen 'MCGO.L1' und 'MCGO.L2' Sprunganweisungen zu den Marken 'L1' bzw. 'L2' (der letztere also zum Ende des Makrokörpers). 'UNLESS' dient der bedingten Textgenerierung (also wenn ein 'ELSE' im Makroaufruf vorhanden ist oder nicht). Im Makrokörper wird auch hier eine fortlaufende Zahl generiert, und zwar durch 'T2'. Im Beispiel 17 sind zwei Makroaufrufe angegeben. Die generierte Zeile 'XX1, LAC PIG' zeigt dabei den nahtlosen Übergang der beiden Generierungsergebnisse.
Das folgende Beispiel ist in der Makrosprache SUPERMAC-BCPL [28] geschrieben.

Beispiel 18

```
MACRO ("INPUT -- (**,-- **),",INPUTMAC)
LET INPUTMAC() BE
    $(FOR K=1 TO MNA() DO
      $(WRITES(ARG(K));WRITES(":=READ;")$)
    $)
```

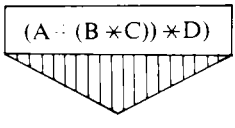
```
INPUT A,C2,BB10,D,WERT;
```

```
A:=READ; C2:=READ; BB10:=READ; WERT:=READ;
```

Durch '— —' wird der formale Parameter des Makros INPUT gekennzeichnet. Der Zusatz '(∗∗,— — ∗∗)' gestattet für diesen formalen Parameter auch eine aktuelle Werteliste. 'INPUTMAC' stellt den Bezug zum Makrokörper her. Hier werden zwei Funktionen zur Parameterauswertung verwendet, und zwar MNA() zur Bestimmung der aktuellen Werteanzahl und ARG(K) zur Bereitstellung des K-ten Parameterwertes. Die zyklische Textgenerierung des Makros INPUT wird durch 'FOR K=1 TO MNA() DO' beschrieben und realisiert die Generierung von 'parameterwert :=READ;' vom ersten (K=1) bis zum letzten (K=MNA()) aktuellen Parameterwert.
Eine Transformation algebraischer Ausdrücke in die sogenannte Postfixform (also genau umgekehrt zur Polnischen Notation) realisiert Makrobeispiel 19 in der Makrosprache von MAX (**Ma**croprocessor for general purpose text manipulation) [107].

Beispiel 19

```
MAC(Φ ∗ Φ) IS 2 4 CONC ∗CONC ENDM
```

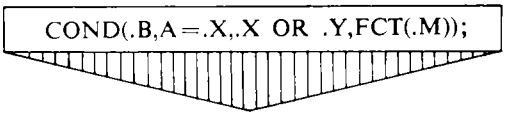



```
ABC ∗ +D ∗
```

CONC kennzeichnet das Verketten, während Φ die Operanden als formale Parameter darstellt. Nach 'IS' folgt die Zeichenkettengenerierung, wobei die '2' das zweite Symbol des Makroaufrufes '(Φ ∗ Φ)' (also das erste Φ) und '4' das vierte Zeichen (also das zweite Φ) bedeuten. Im formalen Makroaufruf steht das Zeichen '∗' für ein beliebiges Operationszeichen.
Eine statische Makroerweiterung zeigt das folgende Beispiel in der BLISS-Makrosprache (**B**asic **L**anguage for **I**mplementation of **S**ystem **S**oftware) [110].

Beispiel 20

```
MACRO COND(BOOL,EXP) []=
    IF BOOL THEN EXP
      EL($REMAINING)COND($REMAINING)$;
MACRO EL [] = ELSE $;
```

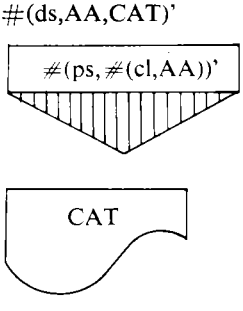



```
IF .B THEN  A=.X
      ELSE IF .X OR .Y THEN FCT(.M)
```

Hierbei sind zwei Makrodefinitionen gegeben. Das Makro EL wird im Makro COND aufgerufen. Die Makros liefern nur Resultate, wenn die Anzahl der aktuellen gleich der Anzahl der formalen Parameter bzw. die aktuelle Parameterliste nicht leer ist. Der Aufruf von COND innerhalb des Makros COND zeigt eine Anwendung des rekursiven Aufrufs. 'REMAINING' bedeutet die Anwendung aller Parameter der aktuellen Parameterliste, die noch nicht bei

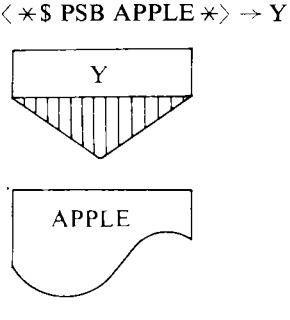
der Makroabarbeitung verwendet wurden, das heißt, ab dem dritten aktuellen Parameterwert; denn die ersten beiden Werte werden den formalen Parametern BOOL und EXP zugeordnet. Die Parameterliste für das Makro EL darf, wie oben bereits genannt, für die Ersetzung durch 'ELSE' nicht leer sein, geht aber in die Zeichenkettengenerierung nicht unmittelbar mit ein.
Weiterhin besteht in der BLISS-Makrosprache die Möglichkeit, eine Folge von Makroaufrufen abkürzend in der Weise anzugeben, daß ein Makroaufruf alle aktuellen Parameterfolgen enthält (also $m \cdot n$ aktuelle Parameterwerte für m Aufrufe mit n Parameterwerten).
Die Abarbeitung von Makroprogrammen in einigen Makrosprachen ermöglicht auch, daß das generierte Ergebnis als Anweisungsfolge in einer Programmiersprache unmittelbar auch ausgeführt wird. So hat das Beispiel in TRAC (Text Reckoning And Compiling) [97] die Form

Beispiel 21



Dabei ist '#' die allgemeine Anweisungskennzeichnung, 'ds' das Definieren einer Zeichenkettenvariablen (hier AA mit dem Wert 'CAT'), 'cl' das »Aufrufen« einer Zeichenkettenvariablen (bzw. auch von Ausdrücken oder anderen Funktionen) und 'ps' das Drucken eines generierten Wertes.
Analoges gilt für das Beispiel in SEL (Self-extensible programming language) [96].

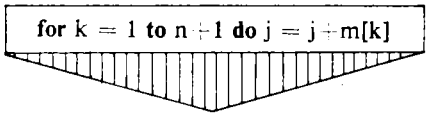
Beispiel 22



Wird die Makrodefinition nicht durch '⟨*' und '*⟩', sondern durch '⟨:' und ':⟩' eingegrenzt, so wird für Y die Zeichenkette ersetzt und keine unmittelbare Ausgabe realisiert.
Das folgende Makrobeispiel wurde im Rahmen einer experimentellen Simulationssprache [89] konzipiert.

Beispiel 23

```
smacro for variable = ausdruck to ausdruck do anweisung
    define begin ◯1 = ◯2;
        M1: if ◯1 = ◯3 then
            begin ◯4; ◯1 = ◯1 + 1;
                goto M1;
            end
        end
    endmacro
```




```
begin k=1;
M1: if k=n+1 then
    begin j=j+m[k]; k=k+1;
        goto M1
    end
end
```

Hierbei sind die formalen Parameter sogenannte syntaktische Elemente der Basissprache. Daher ist dieses Makro auch in der Phase der Programmübersetzung abzuarbeiten. Die Zuordnung der formalen Parameter im Makrokörper lautet

◯1 variable,

◯2 ausdruck,

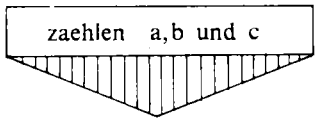
◯3 (zweiter) ausdruck und

◯4 anweisung.

Die Trennung der Definition der Aufrufform (hier mit einem verteilten Makronamen) vom Makrokörper ist durch die Eingrenzung jenes mit **'define'** und **'endmacro'** gegeben.
Ein gleichartiges Beispiel wurde in [35] konzipiert und hat die Form

Beispiel 24

```
define ◯anweisung as 'zaehlen' ◯variable1 %varlist:
        (',' | 'und') ◯variable2 ...
means 'begin ◯variable1:=◯ variable1 + 1';
      while %varlist ¬= nil do
        begin ';◯variable2.%varlist :=
              ◯variable2.%varlist + 1';
              %varlist := next.%varlist
        end
        'end'
endmacro
```

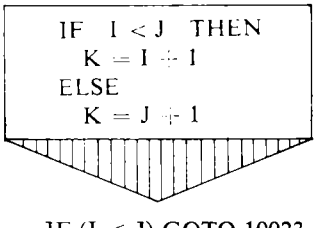



```
begin    a := a+1
   ;     b := b+1
   ;     c := c+1
end
```

Hier ist durch '**define**' der allgemeine syntaktische Typ (nämlich 'anweisung') definiert, und nach '**as**' ('als') folgt die formale Makroaufrufbeschreibung. Dabei wurde eine Variablenliste definiert, deren Elemente mit 'variable1' beginnend entweder durch ein Komma oder durch 'und' getrennt sind. Im Makrokörper werden die Werte der Variablenliste (Parameterliste) der Reihe nach bereitgestellt, bis das Listenende (also 'nil') erreicht ist.
Das folgende Beispiel in der Makrosprache SMOK (**S**intaksičeskij **M**akrogenerator **O**rientirovannyj na **K**onvertirovanie) [82] zeigt wiederum die Definition der Syntaxform des Makroaufrufes.

Beispiel 25

```
MACRO IF TYPE AMBIG
SYNTAX IF ⟨EXPR⟩ THEN ⟨STAT⟩[ELSE ⟨STAT⟩] EOL
BODY IF(%A1) GOTO 10%COUNT
ECOPY
IF PARMSET(%A3) THEN
    BCOPY %A3 ECOPY
ELSE BCOPY GOTO 20%COUNT
    10%COUNT %A2
    20%COUNT CONTINUE
    ECOPY
ENDMACRO
```

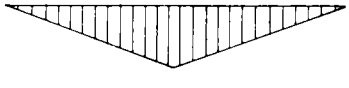



```
    IF (I < J) GOTO 10023
       K = J + 1
       GOTO 20023
10023 K = I + 1
20023 CONTINUE
```

Im Anschluß an die Makrotypkennzeichnung 'MACRO ...' folgt nach 'SYNTAX' die Definition der Aufrufform. Der Makrokörper ist durch 'BODY' und 'ENDMACRO' eingegrenzt. Die Zuordnung der formalen Parameter im Makroaufruf und im Makrokörper lautet hier %A1 – ⟨EXPR⟩, %A2 – ⟨STAT⟩, %A3 – (zweites) ⟨STAT⟩. Die Klammerung ([]) im Makroaufruf zeigt, daß dieser Aufrufteil auch entfallen kann. Zur korrekten Makroabarbeitung wurde daher im Makrokörper die Funktion PARMSET ('parameter gesetzt ?') für den Parameter %A3 verwendet. Die Funktion %COUNT generiert bei jedem Makroaufruf eine fortlaufende Zahl (hier mit '023' beginnend), um im Resultatstext eindeutige Marken ('10023' und '20023') zu gewährleisten. Durch 'BCOPY' wird der Beginn und durch 'ECOPY' das Ende einer (parametrisierten) Zeichenkettengenerierung angezeigt.
Im folgenden ein Beispiel, dessen Generierungsergebnis selbst ein Makroprogramm darstellt, in der Makrosprache des MISEPP (Macro Instruction Structuring Engine Preprocessor) [74].

Beispiel 26

```
macro FACT (&N)
with
%INCLUDE GLOBAL
global Boolean &B;
local int &W,&I;
is
if '-' EQ '&N'(1,1)
then &B ← 1
else
 &B ← 0;
 &I := 1;
 &W := 1;
do &I LE &N then
&W := &W *&I;
&I := &I+1
od;
 [DC F '&W']
fi
mend
```



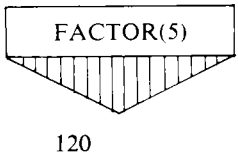
```
MACRO
FACT &N
GBLB &B
LCLA &W
LCLA &I
```

```
    AIF (NOT('-' EQ '&N'(1,1))). @002
        &B SETB (1)
    AGO . @003
. @002 ANOP
&B SETB (0)
&I SETA 1
&W SETA 1
. @004 ANOP
    AIF (NOT(&I LE &N)). @005
&W SETA &W*&I
&I SETA &I+1
  AGO . @004
. @005  ANOP
    DC F'&W'
. @003  ANOP
. @001  ANOP
    MEND
```

MISEPP realisiert die automatische Ergänzung der bedingten Anweisung der Basissprache ('AIF') durch fortlaufend numerierte Marken (' @001', ' @002' usw.) sowie die Umsetzung der zyklischen Anweisung '**do** ... **od**;' in eine Konstruktion mit bedingter Anweisung und Sprunganweisung in der Basissprache. '[...]' bedeutet die unveränderte Übernahme des Textes.
Ein Beispiel für einen rekursiven Makroaufruf zeigt die folgende Makrodefinition in ACSL (Advanced Continuous Simulation Language) [65].

Beispiel 27

```
MACRO FACTOR(N)
MACRO IF (N.EQ.1)
MACRO RETURN N
MACRO ELSE
MACRO SET M=N-1
MACRO RETURN N*FACTOR(M)
MACRO ENDIF
MACRO END
```

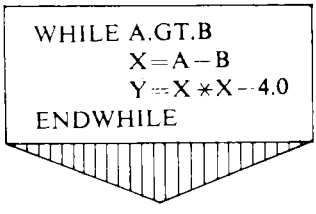


120

Dabei steht 'N.EQ.1' für die Abfrage 'N=1', und 'RETURN' kennzeichnet die Rückgabe des jeweiligen Wertes. 'FACTOR(M)' drückt dabei den rekursiven Aufruf dieses Makros aus.
Abschließend ein Beispiel in der Makrosprache FORMAL ([102]).

Beispiel 28

```
SMACRO WHILE <logical expr>
        <statement list>
        ENDWHILE;
        MEANS "100  IF(.NOT.( @1))GO TO 101;
                @2
                  GOTO 100
           101".
```




```
100 IF (.NOT.(A.GT.B)) GO TO 101
    X=A-B
    Y=X*X-4.0
    GOTO 100
    101 ...
```

Der (geteilte) Makroname ist WHILE ENDWHILE. Die Definition des Makroaufrufs verwendet dabei syntaktische Elemente, die als <logical expr> einen logischen Ausdruck und bei <statement list> eine Anweisungsfolge kennzeichnen. Nach 'MEANS' folgt die Textgenerierung. Dabei bezieht sich ' @1' auf das erste Syntaxelement und ' @2' auf das zweite.

2.3.3. Makroklassifikationen

Für die Klassifikation von Makros werden verschiedene Aspekte zugrunde gelegt.
In Abhängigkeit von der Anzahl der Makrodefinitionen, die eine bestimmte Anzahl von Anweisungen in der Basissprache erzeugen, unterscheidet man in

- *1:n-Makros* (one-to-many macros), wenn *eine* Makrodefinition *n* Anweisungen in der Basissprache generiert (bei *einem* Makroaufruf!);
- *n:n-Makros* (many-to-many macros), wenn *n* Makros *n* Anweisungen in der Basissprache erzeugen;
- *n:1-Makros* (many-to-one macros), wenn *n* Makros *eine* Anweisung in der Basissprache generieren.

Als ein *1:n*-Makro kann das Beispiel 13 in 2.3.2.3. angesehen werden.
Eine weitere Einteilung der Makros ergibt sich aus der Betrachtung des Makrokörpers. Bezieht sich dieser auf Variablen oder Makros außerhalb der Makrodefinition, so handelt es sich um ein *externes Makro* (external macro). Andernfalls liegt ein *internes Makro* (internal macro) vor.
Eine Makrodefinition, die in einem anderen Makro aufgerufen wird, heißt *Hilfsmakro* (service macro) oder *Submakro*. So ist beispielsweise das Makro EL im Beispiel 20 (2.3.2.3.) für das Makro COND ein derartiges Hilfsmakro.

Eine weitere Klassifikation der Makros erhält man bei der Betrachtung des Generierungsergebnisses. Handelt es sich dabei um einen Quelltext einer Programmiersprache, so heißt die jeweilige Makrodefinition *Textmakro* (text macro). Werden bei der Makrogenerierung bereits durch einen Computer abarbeitungsfähige Anweisungen erzeugt, so nennt man dieses Makro ein *Maschinencodemakro* (computational macro). In 2.3.2.3. kann die Makrodefinition im Beispiel 17 als Textmakro betrachtet werden, während Beispiel 21 ein Maschinencodemakro impliziert.
Die Verwendung von Syntaxelementen einer Programmiersprache (Anweisung, Ausdruck usw.), wie sie in den Beispielen 23 bis 25 und 28 in 2.3.2.3. bereits dargestellt wurde, führt zum Begriff des *Syntaxmakros* (syntactic macro). Dieses Makro kann nur in der Phase der Syntaxanalyse eines Compilers realisiert werden, da es das Vorhandensein der entsprechenden Syntaxelemente einer konkreten Programmiersprache als Kontext voraussetzt. Eine letzte Klassifikation soll hier die Einteilung in offene und geschlossene Makros sein. Ein *offenes Makro* (open macro) generiert den gesamten Text an die Stelle, wo es aufgerufen wurde. Ein *geschlossenes Makro* (closed macro) erzeugt hingegen den gesamten Text (zum Beispiel als Unterprogramm in der Basissprache) nur beim ersten Aufruf und liefert bei den folgenden Makroaufrufen nur verschieden parametrisierte Aufrufe dieses Textes (zum Beispiel als Unterprogrammaufrufe).
Bild 5 faßt die bisher genannten Makroklassifikationen noch einmal zusammen.
Die Unterschiedlichkeit der Art und des Inhalts des durch ein Makro generierten Textes führt noch zu einer Menge weiterer Makrobezeichnungen, wie Auswertungsmakro, Bedingungsmakro, Dokumentationsmakro, Druckmakro, Implementationsmakro, Kodierungsmakro, Kontrollmakro, Strukturmakro, Testmakro, Vereinbarungsmakro u.v.a.m. Auf diese Arten soll hier nicht näher eingegangen werden, da sie sich von ihrem Grundaufbau her in die bisher genannten Formen im wesentlichen einordnen lassen.

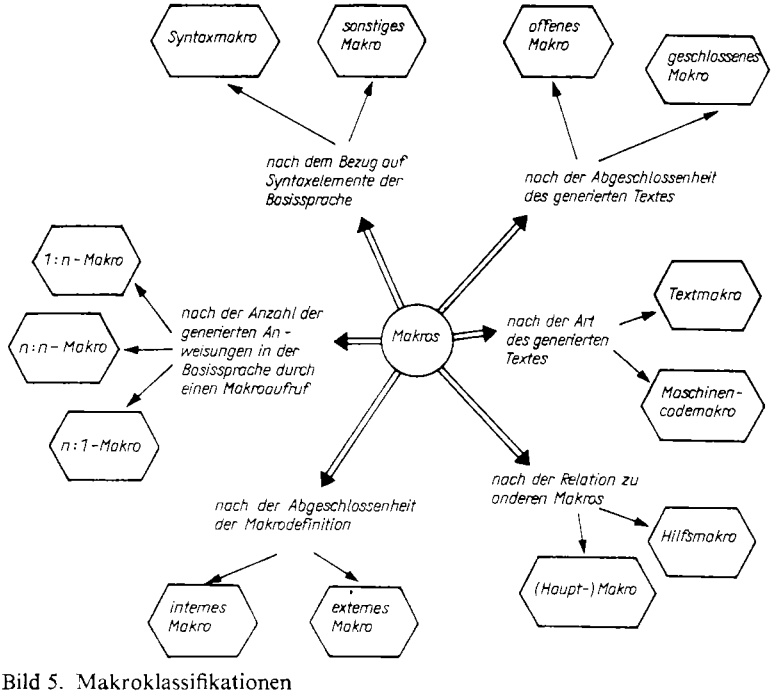


Bild 5. Makroklassifikationen

2.3.4. Makrobibliotheken

Die Makrodefinitionen werden für ihre mehrfache Nutzung im allgemeinen in einer sogenannten *Makrobibliothek* (macro library) gespeichert. Die Speicherungseinheit einer derartigen Bibliothek ist dabei der sogenannte *Bestand* (member). Jeder Bestand besitzt einen Bestandsnamen, unter dem dieser gespeichert und aus der Makrobibliothek gelesen werden kann. Für die Speicherung und Verwaltung von Makrodefinitionen als Bestände einer Makrobibliothek nutzt man folgende Verarbeitungsfunktionen:

- Bestand hinzufügen,
- Bestand ersetzen,
- Bestand löschen,
- Bestand ändern,.
- Bestandsname ändern,
- Bestand ausdrucken,
- Bestand lesen (für die Makroverarbeitung bereitstellen),
- Bestand kopieren (beispielsweise in eine andere Makrobibliothek),
- Bestandsverzeichnis (als Bestandsnamenliste) drucken.

Im allgemeinen enthält *ein* Bestand auch *eine* Makrodefinition. Dennoch kann es für bestimmte Anwendungsfälle sinnvoll sein, von diesem Prinzip abzuweichen. Bild 6 zeigt die grundlegenden Arten der Bestandsinhalte bezogen auf eine Makrodefinition. Konkrete Formen der Bestandsbereitstellung sind beispielsweise in der Makrosprache von MAP (**Ma**cro **P**ascal) [39]

$ INCLUDE (*bestandsname*)

oder in der Sprache JOBOL (**Job O**rganisation **L**anguage) [98]

MACRO *bestandsname*.

Bei dieser Form der Bereitstellung muß stets darauf geachtet werden, daß sie für den jeweiligen Bestand innerhalb einer Makroverarbeitung nur *einmal* vorgenommen wird.
Bei einigen Makrosprachen, wie zum Beispiel bei der ASSEMBLER-Makrosprache ([145]), wird die Bereitstellung der durch einen Makroaufruf geforderten Makrodefinition automatisch realisiert. Ähnlich verhält es sich auch beim Speichern einer Makrodefinition als Bestand in eine Makrobibliothek. So realisiert beispielsweise die Programmiersprache SIMDIS-2 ([15]) bei der Definition eines Makros diese Abspeicherung automatisch.
Derartige Automatismen setzen jedoch die Existenz *einer* Makrobibliothek (nicht mehr, aber auch nicht weniger) voraus.

2.3.5. Makrotestung

Analog zur gewünschten Korrektheit von Programmen, die in einer Programmiersprache geschrieben sind, wird ebenso eine Korrektheit bei der Aufstellung eines Makroprogramms angestrebt. Ziel der Makrotestung ist dabei

(1) eine korrekte Syntax der Makrodefinition zu erreichen,

(2) eine korrekte Semantik der durch den jeweiligen Makroaufruf mit der zugehörigen Makrodefinition generierten Zeichenkette zu gewährleisten.

Zur Erreichung des ersten Zieles sind im allgemeinen bei der Makroabarbeitung Mittel vorhanden, die eine fehlerhafte Syntax, das heißt, eine Anweisungsfolge, die nicht der

Bild 6. Arten der Inhalte des Bestandes einer Makrobibliothek

Notation der jeweiligen Makrosprache entspricht, erkennen und ausweisen (protokollieren); sofern es sich nicht um Fehler in Zyklen handelt, die unter Umständen eine unbegrenzte Makroabarbeitung bewirken. Problematischer ist es bezüglich der korrekten Semantik, das heißt hinsichtlich der Frage, ob das Generierungsergebnis wirklich der festgelegten Basissprache entspricht. Der erhaltene »Wert« bei der Makroabarbeitung ist ja im allgemeinen selbst eine Anweisungsfolge in einer bestimmten Programmiersprache, und es sollen möglichst viele derartige (semantische) Fehler bereits bei der Makrointerpretation erkannt werden, um nicht erst den zumeist nachfolgenden Compiler die dann oft um ein Vielfaches größere Fehlermenge ermitteln zu lassen.
Durch die interpretative Makroabarbeitung (siehe auch 2.4.1.) sind die Möglichkeiten der Testung einer Makrodefinition in dieser Hinsicht sehr begrenzt. Die bei einigen Makrosprachen besonders hohe Variabilität bezüglich der zulässigen Makroaufrufe zu einer Makrodefinition führt im allgemeinen dazu, daß immer »irgendetwas« generiert wird. Zumeist gilt auch hier die Aussage wie beim systematischen Testen in der Programmierung überhaupt; daß die jeweiligen Testbeispiele nur zeigen, daß das (Makro-) Programm genau für diese Beispiele korrekt ist.
Testunterstützungen bei einigen Makrosprachen sind (zusammengefaßt) die folgenden:

(t1) die Initiierung eines Modus, der gewährleistet, daß die Makroabarbeitung in jedem Falle erfolgt (also auch bei einem syntaktischen Fehler);

(t2) die Wahl eines Modus, daß bei der Makroabarbeitung von jedem »durchlaufenen« Makro der Name protokolliert wird;

(t3) die Möglichkeit, sogenannte Testpunkte zu setzen (um beispielsweise die Wertbelegung von Variablen an dieser Stelle oder den bis dahin erzeugten Text zu protokollieren);

(t4) die testweise Abarbeitung eines Makros mit automatisch erzeugten und eingesetzten aktuellen Parameterwerten.

Das zeigt vor allem hierbei die besondere Bedeutung von Testmethoden für Quellprogramme einer (Makro-) Programmiersprache (also beispielsweise die Programmverifikation oder die symbolische Testung in ihrer einfachsten Form, der manuellen Aufstellung einer Ablauftabelle).
Eine derartige Ablauftabelle hat für das Beispiel 3 (2.3.2.3.) die Form

Variablen / Makro-abarbeitung	J	%CALL PUT(J, 1)	generierter Text	J = 4 (Zyklus-ende?)
	2	'2'	Z(2) = X(2) + 1;	nein
	3	'3'	Z(2) = X(2) + 1; Z(3) = X(3) + 1;	nein
	4	'4'	Z(2) = X(2) + 1; Z(3) = X(3) + 1; Z(4) = X(4) + 1;	ja

Der eingerahmte Text zeigt schließlich das Generierungsergebnis.

Übungsaufgaben

2.4. Welche Makroaufrufform beschreibt die Makrodefinition im Beispiel 16 (2.3.2.3.)?
2.5. Welche Parameteraufrufformen definieren eine Makroerweiterung?
2.6. Welche Art der Makroerweiterung beinhalten die Makros

(a)
```
MAKRO1(A,B,C)
FOR I :=A UNTIL B DO
MAKRO2(C)
END;
```

(b)
```
.MACRO MAKRO3 A,B,C
A     B,C
.ENDM
```
mit dem Aufruf MAKRO3 MAKRO4,X,Y?

2.7. Geben Sie für die Beispiele 13, 14 und 17 (2.3.2.3.) die jeweils verwendeten Variablen, Operationen und Funktionen an, wie sie in 2.3.2.2. zusammengefaßt aufgelistet sind.
2.8. Wenden Sie für die Beispiele 12, 15 und 22 (2.3.2.3.) die in 2.3.3. gegebenen Makroklassifikationen an!
2.9. Was wird letztlich beim Einlesen des Bestandes 1 einer Makrobibliothek mit dem in Bild 7 gegebenen Inhalt bereitgestellt?
2.10. Stellen Sie für die Makroabarbeitung im Beispiel 13 (2.3.2.3.) eine Ablauftabelle auf!

Bild 7. Beispiel des Inhaltes dreier Bestände einer Makrobibliothek

2.4. Makroprozessoren

2.4.1. Grundlegende Arbeitsweise eines Makroprozessors

Makroprozessoren werden Prozessoren genannt, die eine Menge von Regeln der Makrogrammatik (als Anweisungen in einer Makrosprache) verarbeiten.
Die allgemeine Arbeitsweise eines Makroprozessors ist in Bild 8 dargestellt.
Ein Makroprozessor erzeugt (generiert) also im allgemeinen Quelltext einer Programmiersprache. Die Arbeitsweise wird daher als *Makrogenerierung* (macro generation) oder einfach *Makroabarbeitung* (macro action) bezeichnet. Auf Grund seiner interpretativen Arbeitsweise werden für Makroprozessoren auch die Synonyme *Makrointerpreter* (macro interpreter) oder eben *Makrogenerator* (macro generator) verwendet.

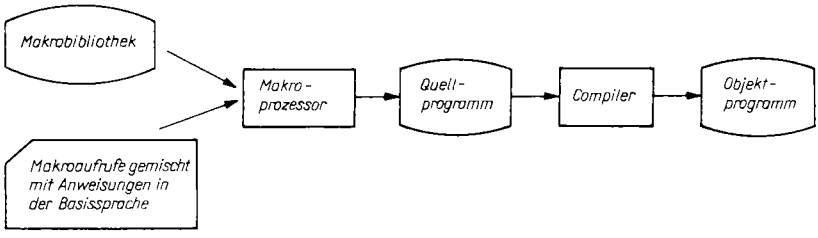


Bild 8. Allgemeine Arbeitsweise eines Makroprozessors

Bild 9 zeigt die speziellere (aber immer noch allgemeingültige) Arbeitsweise eines Makroprozessors mit der Verwendung einer Makrobibliothek.

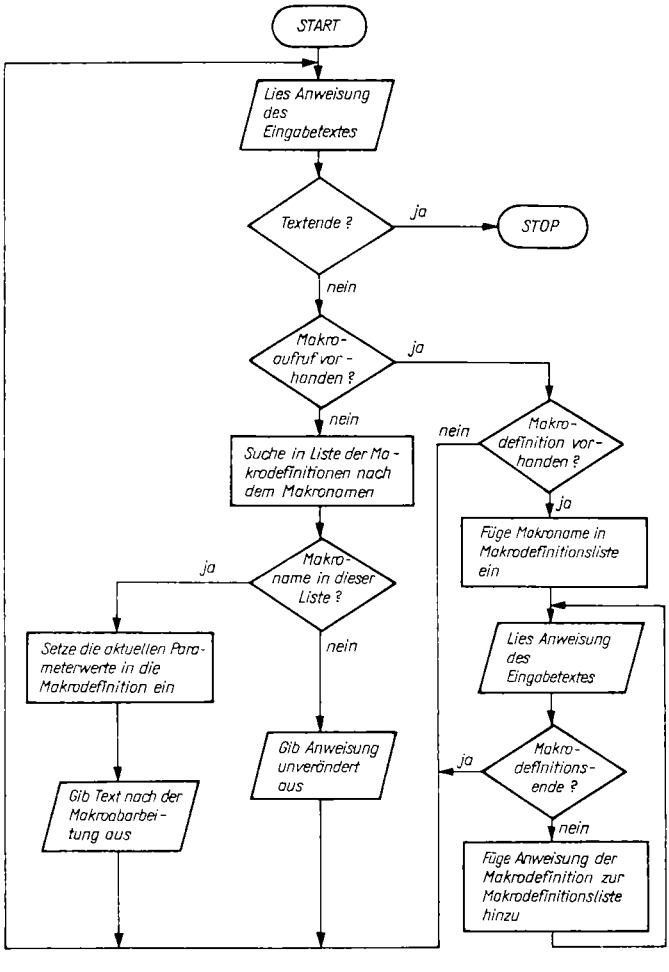


Bild 9. Spezielle Arbeitsweise eines Makroprozessors mit einer Makrobibliothek

Dabei ist ersichtlich, daß das Makroprogramm auch Anweisungen oder Zeichenfolgen der Basissprache enthalten kann. Die Wirkungsweise eines Makroprozessors gliedert sich daher in

(m1) die Zeichenkettengenerierung durch Makroaufrufe und die zugehörigen Makrodefinitionen zu realisieren;

(m2) den sonstigen Text unverändert auszugeben.

Als sonstiger Text wird dabei auch ein Makroaufruf angesehen, für den keine entsprechende Makrodefinition existiert.

2.4.2. Arten von Makroprozessoren

Wie bereits oben genannt realisieren Makroprozessoren im allgemeinen eine interpretierende Arbeitsweise, das heißt, es gilt die formale Darstellung

$$L_M, L_B, L^{i_M} ,$$

wobei L_M für die Makrosprache, L_B für die Basissprache steht und L wiederum die Sprache bedeutet, in der der Makroprozessor geschrieben ist.
In Abhängigkeit von der Anzahl der durch einen Makroprozessor generierbaren Basissprachen unterscheidet man *allgemeine Makroprozessoren* (general-purpose macro processors), wenn mehrere Basissprachen erzeugt werden können (jene also wählbar ist), und in *spezielle Makroprozessoren* (special-purpose macro processors), wenn jene nur für eine Basissprache ausgerichtet sind.
Eine andere Klassifikation ergibt sich aus dem Zusammenhang von Makroprozessor und Compiler gemäß der in Bild 10 beschriebenen Arbeitsweise.

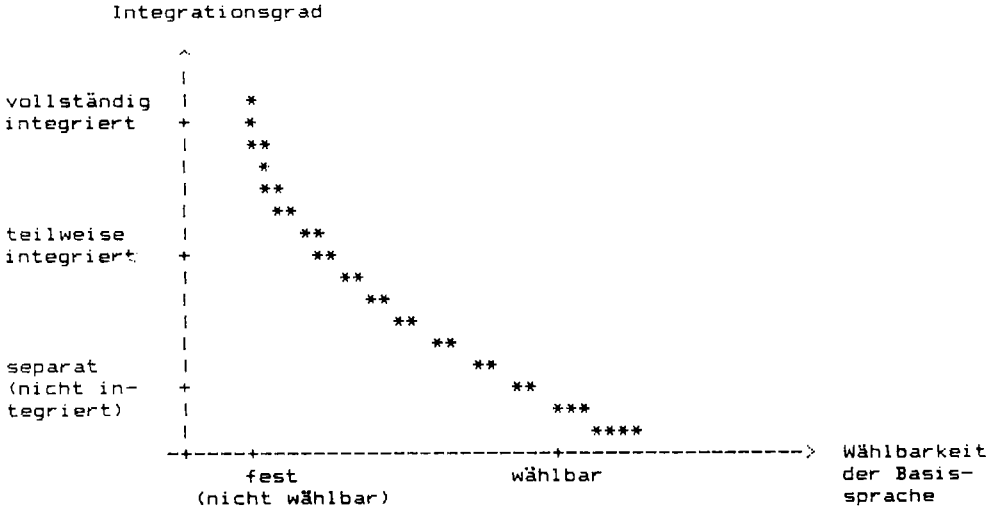


Bild 10. Zusammenhang zweier Charakteristika für Makroprozessoren

Danach wird ein Makroprozessor unterteilt in

(mp1) *Preprozessor*[1]) (preprocessor): der Makroprozessor arbeitet als Vorübersetzer unabhängig vom Compiler, dem er vorgelagert ist;
für L_Z als Zielsprache des nachfolgenden Compilers gilt für diese Arbeitsweise die formale Beschreibung

$$\underset{L_M,L_B,L}{i_M} \quad * \quad \underset{L_B,L_Z,L'}{c} \quad ;$$

(mp2) *teilweise integrierter Preprozessor* (partial-integrated preprozessor): der Makroprozessor und der Compiler sind in der Weise integriert, daß der durch den Makroprozessor generierte Text unmittelbar vom Compiler übernommen wird;
formal sei dieser Zusammenhang dargestellt als

$$\left(\underset{L_M,L_B,L}{i_M} \quad * \quad \underset{L_B,L_Z,L'}{c} \right) ;$$

(mp3) *vollständig integrierter Makroprozessor* (complete-integrated macro processor): Makroprozessor und Compiler bilden eine untrennbare Einheit;
formal gilt hier also

$$\underset{L_M,(L_B,L_Z),L}{c_M} \quad , \text{ das heißt, } i_M \subset c_M.$$

Integrierte Makroprozessoren werden auch als *eingebettete Makroprozessoren* (embedded macro processors) bezeichnet.
Eine spezielle Einbettung wird auch dadurch erreicht, indem man die Makroabarbeitung nicht mit der nachfolgenden Übersetzung integriert, sondern den nachfolgenden Compiler zunächst selbst für die separate Übersetzung der Routinen für die Makroverarbeitung einerseits und der Makrodefinitionen andererseits verwendet. Der dann folgenden Makroabarbeitung sind dann nur noch die jeweiligen Makroaufrufe zuzuführen. Der generierte Text kann dann dem Compiler (wie beim Preprozessor) übergeben werden. Formal gilt hierbei

$$\underset{L_O,L_Z,L}{v} \quad \underset{L_B,L_O,L}{(c(i_M),} \quad \underset{L_B,L_O,L}{c(mdef))} \quad = \quad \underset{L_M,L_B,L_Z}{\dot{i}_M}$$

(mit *mdef* als Makrodefinitionen) als Programmverbindung und dem nachfolgenden

$$\underset{L_M,L_B,L_Z}{\dot{i}_M} \quad * \quad \underset{L_B,L_Z,L}{c} \quad .$$

L_M enthält hierbei nur die Makroaufrufe und (i. allg.) auch anderen (nicht die Makroverarbeitung betreffenden) Text. Hierbei sind also Makroprozessor und Makrodefinitionen in derselben Sprache (der Basissprache L_B) geschrieben und mit $\dot{i}_M$ zusammengefaßt. Diese Makroprozessorart wird SUPERMAC [28] genannt.
Während Preprozessoren und teilweise integrierte Preprozessoren im allgemeinen Textmakros realisieren, können Syntaxmakros nur durch vollständig integrierte Makroprozessoren abgearbeitet werden.
Bild 10 zeigt den Zusammenhang der Klassifikationen von Makroprozessoren bezüglich der Anzahl der generierbaren Basissprachen und der Verbindung zum Compiler.

[1]) Diese Bezeichnung hat sich im Gegensatz zum eigentlich deutschsprachig korrekten Präprozessor durchgesetzt.

2.4.3. Anforderungen an einen Makroprozessor

Zunächst soll natürlich ein Makroprozessor die Anforderungen, die durch die jeweilige Makrosprache gestellt sind, das heißt, den jeweiligen Anweisungsumfang zu verarbeiten, realisieren. Die Art und der Umfang der Makrosprache hängen dabei vor allem von den jeweiligen Anwendungszielen ab. Allgemein gilt der Grundsatz, einen minimalen Sprachumfang für einen maximalen Anwendungsbereich zu erreichen.
Für eine wirkungsvolle Realisierung von Makrosprachen bestehen darüber hinaus die folgenden weiteren Kriterien:

(k1) Makroaufrufe sollten sich nicht so sehr von der Basissprache unterscheiden;

(k2) Makros sollten portabel (also nicht nur auf einen Computer beschränkt) sein;

(k3) das Definieren und Testen von Makros sollte nicht komplizierter sein als in der üblichen Anwendungsprogrammierung;

(k4) dem Programmierer sollten auf jedem Computer Makrosprachen zur Verfügung stehen.

Die Kriterien gelten dabei sowohl für die Auswahl eines bereits existierenden Makroprozessors als auch für eine Neukonzipierung.

2.4.4. Beispiele für Makroprozessoren

Für die in 2.4.2. genannten Arten von Makroprozessoren sollen hier einige Beispiele angegeben werden. Dabei wird die formale Beschreibungsform, wie sie bisher für die Prozessoren verwendet wurde, zugrunde gelegt.

Beispiele für allgemeine Makroprozessoren sind

(1) GPM und
GPM-Sprache,L_B,ASSEMBLER

(2) ML/I
ML/I,L_B,ASSEMBLER

Die Angabe von L_B für die Basissprache bedeutet, daß diese nicht festgelegt (also wählbar) ist. Für diese beiden Makroprozessoren sind bereits im Abschnitt »Makrosprachen« erläuternde Beispiele angegeben worden.
Ein Beispiel für einen speziellen Makroprozessor ist durch MAKFOR (**Mak**ro-**FOR**TRAN) [134] gegeben in der Art

(3) MAKFOR-Prozessor .
MAKFOR,FORTRAN,ASSEMBLER

Während die bisherigen Beispiele gleichzeitig Makroprozessoren vom Preprozessortyp darstellen, lautet ein Beispiel für einen teilweise integrierten Makroprozessor

(4) PL/1-Makrointerpreter .
PL/1-Makrosprache,PL/1,ASSEMBLER

Die besondere Art der teilweisen Integration zum PL/1-Compiler ermöglicht hierbei, die ausschließliche Orientierung auf PL/1 als Basissprache zu umgehen (siehe auch 4.3.).

Beispiele für vollständig integrierte Makroprozessoren sind

(5) MICMAC und
MICMAC-Sprache,ASSEMBLER,ASSEMBLER

(6) MCOBOL ,
MCOBOL-Sprache,COBOL,ASSEMBLER

deren Bezeichnungen bei MICMAC von **Micro** partie d'assembleur et **macros** [162] und bei MCOBOL von **Macro-COBOL** [133] herrühren. Während die bisherigen Beispiele in der Maschinensprache des jeweiligen Computers (ASSEMBLER) geschrieben sind, lautet ein Beispiel eines Makroprozessors, der in PL/1 geschrieben ist,

(7) MISEPP .
MISEPP-Sprache,Macro-ASSEMBLER,PL/1

Ein Anwendungsbeispiel ist hierzu in 2.3.2.3. gegeben. Beispiele für eingebettete Makroprozessoren sind ([28], [33])

(8) SUPERMAC und
SUPERMAC-BCPL,L_B,BCPL

(9) SUPERMAC .
SUPERMAC-PASCAL,L_B,PASCAL

Während BCPL (**B**ootstrap **C**ombined **P**rogramming **L**anguage) eine Systemprogrammiersprache darstellt, ist PASCAL (mit dem Namen des Mathematikers BLAISE PASCAL benannt) eine universelle Programmiersprache.
Eine umfangreichere Auflistung der bekanntesten Makrosprachen (und Makroprozessoren) ist im Anhang gegeben.
Bei den Makroprozessoren vom Preprozessortyp und bei den teilweise integrierten Makroprozessoren ist der Zusammenhang mit einer nachfolgenden Übersetzung formal durch eine Komposition beschreibbar, also beispielsweise

MAKFOR-Preprozessor * FORTRAN-Compiler
MAKFOR,FORTRAN,ASSEMBLER FORTRAN,ASSEMBLER,ASSEMBLER

oder

ML/I * ALGOL-Compiler
ML/I,ALGOL,ASSEMBLER ALGOL,ASSEMBLER,ASSEMBLER

u.a.m.

Übungsaufgaben

2.11. Geben Sie die Arten der Makroprozessoren an, die für die Realisierung der Beispiele 11, 12, 24 und 26 in 2.3.2.3. erforderlich sind!

2.12. Formulieren Sie eine Komposition der Makroprozessoren MAX (Beispiel 19, 2.3.2.3.), MP/1 (Beispiel 13, 2.3.2.3.) und MISEPP (Beispiel 26, 2.3.2.3.) jeweils mit einem Compiler!

2.13. Welche Arten von Makros kann ein Makroprozessor vom Preprozessortyp verarbeiten?

2.5. Anwendungsgebiete der Makroprogrammierung

2.5.0. Einleitende Bemerkungen

Wie zu Beginn des Abschnitts 2. bereits gezeigt, diente die Makroprogrammierung zunächst vor allem dazu, eine gewisse (inhaltlich zusammenhängende) Menge von Anweisungen einer Programmiersprache zusammenzufassen. Bezüglich der Zuordnung zu den umfangreichen inhaltlichen Aufgaben der Programmierung und ihrer Techniken sollen im folgenden die wesentlichsten Anwendungsgebiete an Hand von Beispielen etwas ausführlicher betrachtet werden.

2.5.1. Erweiterung der Basissprache

Dieser Anwendungsaspekt der Makroprogrammierung gilt als ursprünglichstes Ziel, in dem sich – allgemein betrachtet – auch die anderen hier genannten Anwendungsgebiete einordnen lassen. Ein erstes Beispiel sei in der Makrosprache MAKFOR gegeben. Hierbei geht es um die Spracherweiterung der Programmiersprache (Basissprache) FORTRAN. So lautet beispielsweise ein konkreter Zyklus in FORTRAN

```
      DO 1 I=1,N
        S = S + A*B
1     CONTINUE .
```

Das allgemeine durch einen Makro realisierbare Schema sei

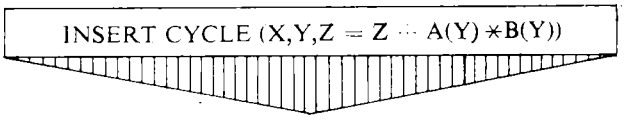
```
      DO  1  α=1,β
      γ
1     CONTINUE
```

Das entsprechende Makro in MAKFOR lautet dafür

```
      SUBROUTINE CYCLE(I,N,ANW)
      INTEGER I,N
      DO 1 I=1,N
      INSERT ANW
1     CONTINUE
      RETURN
      END
```

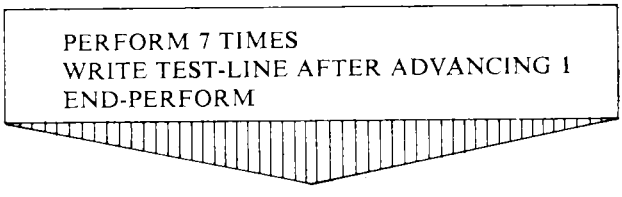
Das Makro wird hier als sogenannte SUBROUTINE (Unterprogramm) formuliert, besitzt den Namen CYCLE und hat die formalen Parameter I und N für die sogenannte Laufvariable I und die Zyklusgrenze N und ANW für die Anweisung, die zyklisch zu realisieren ist. Die INSERT-Funktion kennzeichnet das Einsetzen von ANW in dieser Zeile. Eine konkrete

Anwendung, die die Makrodefinition von CYCLE als gegeben voraussetzt, lautet




```
      DO 1 X=1,Y
      Z = Z + A(Y) *B(Y)
1     CONTINUE
```

Hierbei wurde die Makroprogrammierung zur Definition eines neuen Operators 'CYCLE' für die Basissprache FORTRAN verwendet.
Ein weiteres Beispiel in der Makrosprache MCOBOL zeigt die Strukturierung eines Zyklus in der Programmiersprache COBOL im Sinne der Strukturierten Programmierung. Dabei ist die spezielle COBOL-Kenntnisse erfordernde Makrodefinition nicht mit angegeben.




```
           PERFORM REPEATED-CODE 7 TIMES
           GO TO END-TEXT.
REPEATED-CODE.
           WRITE TEST-LINE AFTER ADVANCING 1.
END-TEXT.
```

Hierbei wurde für den in COBOL üblichen Zyklusbeginn 'PERFORM' die Zyklusendeanweisung 'END-PERFORM' eingeführt. Das Generierungsergebnis enthält zusätzliche Marken ('REPEATED-CODE' und 'END-TEXT') und die Zyklusbeendigung als Sprunganweisung ('GO TO END-TEXT'). Die COBOL-Anweisung, die hierbei siebenmal ('7 TIMES') durchlaufen wird, ist 'WRITE TEST-LINE AFTER ADVANCING 1'.
Weitere Beispiele für Basisspracherweiterungen sind bereits in 2.3.2.3. gegeben. So ist im Beispiel 25 eine Erweiterung für die Basissprache FORTRAN in der Hinsicht gegeben, daß die bei anderen Programmiersprachen (wie z.B. ALGOL oder PL/1) mögliche Form einer bedingten Anweisung mit Hilfe von 'IF' und 'THEN' (bzw. 'ELSE') auch hierbei ermöglicht wurde. Eine analoge Möglichkeit für die Basissprache ASSEMBLER zeigt das Beispiel 17 in 2.3.2.3. mit Hilfe der Makrosprache ML/I.
Beispiele zur Erweiterung der Makrosprache selbst sind in 3.4.3. zur ASSEMBLER-Makroprogrammierung und in 4.3. zur PL/1-Makroprogrammierung gegeben.

2.5.2. Transformation von einer in eine andere formale Sprache

Hierbei handelt es sich nicht nur um die Erweiterung einer Basissprache, sondern um ihre vollständige Generierung durch Makros.
Als Beispiel sei hier zunächst die Makrosprache MAKROBOL (**Makro-COBOL**) [16] genannt. Der Makroaufruf hat dabei die allgemeine Form

$$Fi(p_1,p_2,\cdots,p_k),\ i = 1,2,3,\ \ldots,\ 9.$$

F1 bis F9 sind Funktionen zur Dateiverarbeitung. So definieren F1 und F2 die Dateien (d.h. generieren die jeweiligen COBOL-Anweisungen für diese Definition), während beispielsweise F4 die Bildung von Teildateien durch Satzauswahl und F6 das Umspeichern einer Datei festlegen. p_1 bis p_k sind die jeweiligen Parameterangaben. Generiert werden Quellprogramme in der Basissprache COBOL. Hierbei wird also eine Sprachtransformation von einer Parametersprache in die Programmiersprache COBOL realisiert. Eine weitere Sprachtransformation zeigt auch Beispiel 19 in 2.3.2.3., und zwar die Transformation algebraischer Ausdrücke aus der Infixnotation in die Postfixnotation.
In das Anwendungsgebiet der Sprachtransformation fällt auch die Implementation von Fachsprachen (siehe hierzu 4.4. zur PL/1-Makroprogrammierung).

2.5.3. Realisierung der Portabilität

Die Realisierung der Portabilität ist in gewisser Hinsicht ein Spezialfall der Sprachtransformation, und zwar in bezug auf die Transformation ein- und derselben Sprache für unterschiedliche Computer.
Die Realisierung der Portabilität durch eine Makrosprache geschieht in der Weise, daß die Makrodefinition die Implementationsbesonderheiten für den anderen Computer durch entsprechende Transformation im Makrokörper ausgleicht.
Hierzu wurde beispielsweise für die ASSEMBLER-Sprache verschiedener Computer eine computerunabhängige Beschreibungssprache DLIMP (Descriptive Language Implemented by Macro Processor) [32] definiert und mittels der Makrosprache ML/I implementiert. Die Portabilitätsproblematik ist an einem konkreten Beispiel in 3.3.3. zur ASSEMBLER-Makroprogrammierung gegeben.

2.5.4. Verwendung von Makroprozessoren als Compiler-Compiler

Eine derartige Anwendung ist nur durch allgemeine Makroprozessoren möglich, da die Basissprache gerade bei Compiler-Compilern wählbar sein muß.
Ziel ist es also, einen Makroprozessor als Compiler zu verwenden. Dabei werden die bereits durch den Makroprozessor realisierten Funktionen der Syntaxkontrolle und Codegenerierung ausgenutzt. Zu ergänzen sind Funktionen der Fehleranalyse und Fehlerbehandlung für die zu implementierende Basissprache, die ebenfalls in der jeweiligen Makrosprache des Makroprozessors zu schreiben sind. Ein Beispiel für die Ausnutzung des Makroprozessors zu ML/I als Compiler-Compiler ist in [126] gegeben.

2.5.5. Makrotechnik und andere Programmierungstechniken

In 1.2. wurden bereits die Programmiertechniken »Strukturierte Programmierung« und »Unterprogrammtechnik« prinzipiell erläutert. Neben diesen beiden soll hier noch auf den Zusammenhang zwischen der Makrotechnik und der sogenannten Modulartechnik bzw. dem *Modularen Programmieren* (modular programming) eingegangen werden. Ein **Modul** (module) ist dabei ein (relativ) unabhängiger Programmbaustein (-teil), der über definierte Schnittstellen zu nutzen ist. Er sei als Bestandteil eines Ganzen leicht auswechselbar und soll keine sogenannten Fernwirkungen, das heißt, zum Beispiel Zugriffe zu anderen Moduln (außer den durch die Schnittstellen definierten) u.a.m., besitzen.

Der Zusammenhang dieser Programmiertechnik mit den anderen ist einfach zu erkennen. Definiert man beispielsweise die Schnittstelle eines Programmoduls als Parameter, so entspricht dieser Modul einem Unterprogramm. Der Zusammenhang zur Makrotechnik ist also beispielsweise durch die Generierung geschlossener Makros gegeben.
Ein Beispiel für die Verknüpfung der Strukturierten Programmierung mit der Makroprogrammierung ist durch die Makrosprache MALCROL 68 (Macro-**ALGOL** 68) [72] gegeben. Die Makros besitzen für die jeweils durch sie definierte Programmstruktur eine einleitende und eine abschließende Bezeichnung, so wie es die Strukturierte Programmierung bezüglich der Abgeschlossenheit der Strukturelemente fordert. So lauten beispielsweise die Anweisungen für eine Sequenz

DO ..... **OD**,

für eine Alternative

IF .... THEN .... ELSE .... **FI**

oder eine Verzweigung

CASE OF ..... **ESAC**.

Der Zusammenhang von Makrotechnik und Unterprogrammtechnik ist zunächst unmittelbar dadurch gegeben, daß im allgemeinen ein Makro selbst ein Unterprogramm einer speziellen Makrosprache darstellt. Hier soll jedoch das Generierungsergebnis betrachtet werden, das heißt, der Zusammenhang zum Unterprogramm in der Basissprache. Durch ein Makro können ein oder mehrere Unterprogramme erzeugt werden. Inhaltlich bedeutet dies eine weitere Verallgemeinerung der durch die Anweisungen des Unterprogramms zusammengefaßten Funktionen. Andererseits kann ein Makro die Anweisungsfolge, die bei einer bestimmten Parameterbelegung für das Unterprogramm in diesem durchlaufen wird, generieren und hebt so die Zusammenfassung verschiedener Algorithmen durch ein Unterprogramm auf. Bezüglich der effektiven Anwendung von Makros oder/und Unterprogrammen gilt (gemäß [85])

Einheitliche Funktion in einem oder verschiedenen Programmen		Variable Funktion in einem oder verschiedenen Programmen	
einmalig je Programm:	mehrfach je Programm:	einmalig je Programm:	mehrfach je Programm:
Makro oder Unterprogramm	Unterprogramm	Makro	Kombination Makro/Unterprogramm

Ein Beispiel für die Realisierung der Modulartechnik ist bereits in 2.5.2. mit der Makrosprache MAKROBOL gegeben worden. Die hier nutzbaren Makros generieren funktionell abgeschlossene Programme (bzw. -teile).
Neben dieser Anwendung der Makroprogrammierung sei abschließend auf den speziellen Einfluß jener auf die Programmierweise verwiesen.
Dazu sei das erste Programmschema in 2.1. nochmals angegeben. Es lautet

DO I=α TO β;S=S+γ;END;

für das spezielle Wertetripel (α,β,γ)=('1','N','V(I)') ergibt sich 'DO I=1 TO N;S=S+V(I); END;'. Inhaltlich bedeutet dies die Summation

$$S = V(1) + V(2) + V(3) + \ldots + V(N).$$

Ist das N, das heißt, die Anzahl der Summanden, explizit vorgegeben, so bietet sich bei der Makroprogrammierung die Möglichkeit an, die obige Summation in ihrer inhaltlichen Darstellung direkt zu generieren. Also beispielsweise in der Art

$$S = \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_N$$

mit α_i = 'V(i)', wobei das N einen expliziten Wert

besitzen muß.
In dieser Weise können die Anweisungen einer Programmiersprache, die zur Einsparung des Aufwandes bei der Programmniederschrift dienen und in compilierter Form eine unter Umständen aufwendigere Abarbeitung erfordern, bei der Makroprogrammierung ihre Bedeutung verlieren.

Übungsaufgaben

2.14. Ordnen Sie die Beispiele 1, 10, 13, 18 und 25 in 2.3.2.3. den genannten Anwendungsgebieten der Makroprogrammierung zu!

2.15. Nennen Sie aus den Makrobeispielen in 2.3.2.3. ein weiteres Beispiel für die Sprachtransformation!

3. ASSEMBLER-Makroprogrammierung

3.1. Charakteristika der ASSEMBLER-Makrotechnik

Wie eingangs bereits erwähnt, waren die ersten Anwendungen der Makrotechnik für die symbolische Maschinensprache ASSEMBLER konzipiert. Zur Verarbeitung von Anweisungen der ASSEMBLER-Makrosprache wurde ein Makroprozessor geschaffen, der seinem Prinzip nach zu den vollständig integrierten Makroprozessoren gehört. Die generierten Anweisungen in der Basissprache werden also unmittelbar vom Compiler übernommen und verarbeitet, beziehungsweise, wenn es sich bereits um Maschinenanweisungen handelt, in die Codegenerierung mit einbezogen. Die Basissprache ist daher ausschließlich die ASSEMBLER-Sprache des jeweiligen Computers. Also es gilt formal

ASSEMBLER-Makroprozessor .

ASSEMBLER-Makrosprache,ASSEMBLER,ASSEMBLER

Die ASSEMBLER-Programmiersprache ist maschinenabhängig, so daß jeder Computer seine ASSEMBLER-Sprache besitzt.
Auf die Makroerweiterung der bekanntesten ASSEMBLER-Sprachen, und zwar eines Mikro-, eines Mini- und eines mittleren Computers, soll im folgenden näher eingegangen werden. Dabei werden jeweils zuerst die Möglichkeiten zur Quelltextmanipulation im allgemeinen (als bedingte Assemblierung) und dann unter Verwendung von Makrodefinitionen im besonderen angegeben.

3.2. ASSEMBLER-Makrosprache MACRO-80

3.2.0. Einleitende Bemerkungen

MACRO-80 ist ein Compiler für die ASSEMBLER-Sprache der Mikrorechnersysteme auf der Basis des Zentralprozessors Z80 und besitzt Möglichkeiten zur Makroprogrammierung. Er ist also beispielsweise auch für die sogenannten Bürocomputer A 5120 bzw. A 5130 und Personalcomputer PC 1715 nutzbar.

3.2.1. Bedingte Assemblierung

Variablen sind bei der bedingten Assemblierung von MACRO-80 unmittelbar durch ihre Verwendung bei Wertzuweisungen definiert. Die Wertzuweisung realisiert die Anweisung 'SET' in der Form

name SET *ausdruck*.

name bezeichnet eine Variable und *ausdruck* steht für einen arithmetischen Ausdruck oder eine Zeichenkette. So ergibt beispielweise

```
X   SET 2
Y   SET 'A'
```

die Zuweisung der Zahl 2 für X und der Zeichenkette 'A', die stets durch Apostrophe zu begrenzen ist, für Y.
Für die bedingte Generierung von ASSEMBLER-Anweisungen ist die Anweisung

```
IFxx        [argument]
  ⋮
[ELSE]
  ⋮
ENDIF
```

anzuwenden. Die Klammerung des 'ELSE' bedeutet die wahlweise Anwendung dieses Teiles, der ausgeführt wird, wenn die Bedingung der ersten Zeile nicht gilt, also falsch ist.
Für *xx* bestehen im Zusammenhang mit *argument* folgende Möglichkeiten, Bedingungen zu formulieren.

xx	argument	Bedeutung
T bzw. leer	*ausdruck*	wahr, wenn *ausdruck* nicht Null ist
E bzw. F	*ausdruck*	wahr, wenn *ausdruck* Null ist
NDEF	*symbol*	wahr, wenn *symbol* nicht definiert ist
DEF	*symbol*	wahr, wenn *symbol* definiert ist
B	*arg*	wahr, wenn *arg* leer ist
NB	*arg*	wahr, wenn *arg* nicht leer ist
IDN	*arg1*, *arg2*	wahr, wenn Zeichenkette *arg1* gleich Zeichenkette *arg2* ist
DIF	*arg1*, *arg2*	wahr, wenn Zeichenkette *arg1* ungleich Zeichenkette *arg2* ist

⟨*symbol*⟩ steht für die Bezeichnung einer Variablen und *arg*, *arg*1 und *arg*2 bedeuten Zeichenketten oder Variablenbezeichnungen.
So ergibt beispielsweise die Anweisung

```
IFIDN   <'WERT1'>,<'WERT2'>
PUSH    BX
MVI     BX,'A'
JMP     M1
ELSE
PUSH    BX
MVI     BX,'B'
JMP     M2
ENDIF
```

die ASSEMBLER-Anweisungsfolge

```
PUSH    BX
MVI     BX,'B'
JMP     M2.
```

Für die wiederholte Generierung von ASSEMBLER-Anweisungen gibt es in MACRO-80 drei verschiedene Möglichkeiten. Die zyklische Generierung in Abhängigkeit einer vorgegebenen Anzahl (explizit oder implizit als arithmetischen Ausdruck) wird durch die Anweisung

```
REPT   ausdruck
  ⋮
ENDM
```

realisiert. So ergibt

```
X   SET     4
    REPT    X
    DB      3
    ENDM
```

die Anweisungsfolge

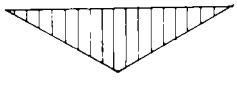
```
DB      3
DB      3
DB      3
DB      3 .
```

Eine wiederholte Generierung realisiert ebenfalls die Anweisung

```
IRP symbol,arglist
  ⋮
ENDM.
```

Ihre Wirkungsweise verdeutlicht das folgende Beispiel

```
IRP     X,<1,2,3,4,5,6,7,8,9>
DB      X
ENDM
```



```
DB      1
DB      2
  ⋮
DB      9
```

Ein Spezialfall dieser Anweisung, schließlich die dritte Möglichkeit der wiederholten Generierung in MACRO-80, lautet

```
IRPC  symbol,kette
  ⋮
ENDM  .
```

Sie generiert die enthaltene Anweisungsfolge so oft, wie Zeichen in *kette* vorhanden sind. Dabei stehen zur Manipulation bei der wiederholten Generierung die Zeichen aus *kette*

über die Variable *symbol* der Reihe nach zur Verfügung. Durch diese Anweisung kann das oben gegebene Generierungsergebnis in der Form

```
IRPC X,123456789
DB   X
ENDM
```

ebenfalls erzeugt werden.

3.2.2. Makrodefinitionen in MACRO-80

Eine Makrodefinition hat in MACRO-80 die allgemeine Form

```
name  MACRO parmlist
  :
ENDM  .
```

Im Makrokörper können die Anweisungen der bedingten Assemblierung verwendet werden. Außerdem kann der Makrokörper durch die Anweisung EXITM vor der ENDM-Anweisung verlassen werden.
Durch *name* wird der Makroname festgelegt, während *parmlist* die Liste der formalen Parameter ausdrückt. Ein einfaches Beispiel einer Makrodefinition lautet

```
BEISP1  MACRO  X
        PUSH   BX
        MVI    BX,'T'
        JMP    X
        ENDM   .
```

Mit dem Makroaufruf 'BEISP1 M1' werden die Zeilen

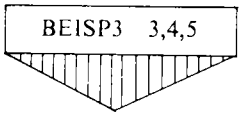
```
PUSH   BX
MVI    BX,'T'
JMP    M1
```

erzeugt. Die Möglichkeit der Verwendung einer durch spitze Klammern eingegrenzten aktuellen Parameterliste zeigt die folgende Makrodefinition

```
BEISP2  MACRO  X
        IRP    Y,<X>
        DB     Y
        ENDM
        ENDM
```

mit dem Makroaufruf 'BEISP2 <1,2,3,4,5,6,7,8,9>'. Sie liefert wiederum die oben angegebene Zeilenfolge 'DB 1, DB 2 ...'. Die Art und Weise der Verwendung mehrerer formaler Parameter zeigt das folgende Beispiel.

```
BEISP3  MACRO  X,Y,Z
        DB     X
        DB     Y
        DB     Z
        ENDM
```



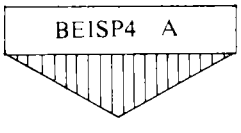

```
        DB     3
        DB     4
        DB     5
```

Für die Verkettung eines Parameterwertes mit einer Zeichenkette wird das Zeichen '&' verwendet.
Der eindeutigen Generierung von Bezeichnungen bei der Verwendung mehrerer Makrodefinitionen dient die Anweisung

LOCAL *symbolliste.*

Die Variablen der *symbolliste* sind nur im Makrokörper gültig und werden bei der Makroabarbeitung zu einer Bezeichnung der Art '..0000', '..0001', usw. bis '..FFFF' (also hexadezimal).
Das Makro BEISP4 zeigt die Anwendung der LOCAL-Anweisung und die Parameterwertverkettung.

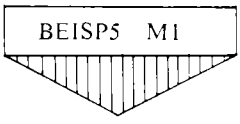
```
BEISP4  MACRO  X
        LOCAL  Y
        PUSH   BX
        MOV    BX,'WERT&X'
        JMP    Y
        ENDM
```




```
        PUSH   BX
        MOV    BX,'WERTA'
        JMP    ..0000
```

MACRO-80 gestattet auch eine statische Makroerweiterung in Form eines Makroaufrufes im Makrokörper. So ergibt sich beispielsweise die folgende Makroabarbeitung, bei der die Makrodefinition BEISP1 als gegeben in der oben definierten Weise vorausgesetzt wurde.

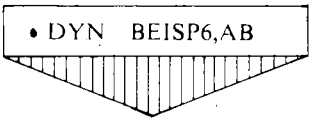
```
BEISP5  MACRO  Z
        LD     BX,A
        BEISP1 Z
        NOP
        ENDM
```




```
        LD      BX,A
        PUSH    BX
        MVI     BX,'T'
        JMP     M1
        NOP
```

Die Realisierung rekursiver Makros ist in MACRO-80 prinzipiell möglich.
MACRO-80 ermöglicht aber auch die dynamische Makroerweiterung, das heißt, die Verwendung eines Makronamens als aktuellen Parameterwert. Hierzu das folgende Beispiel

```
BEISP6  MACRO   X
        DB      X
        ENDM
DYN     MACRO   Y,Z
        PUSH    Z
        Y       5
        ENDM
```




```
        PUSH    AB
        DB      5
```

3.2.3. Anwendung von MACRO-80 zur Basisspracherweiterung

Die hier angegebenen Makros gehen auf eine Idee von APTE [143] zurück, wo es um die Erweiterung der Basissprache ASSEMBLER in der Hinsicht geht, daß weitere bedingte Sprunganweisungen zur übersichtlicheren ASSEMBLER-Programmierung als Makros implementiert wurden. Derartige Makros lauten in MACRO-80:

(1) Sprung bei 'größer als' (JGT):

```
JGT     MACRO   P1
        LOCAL   L1
        JRZ     L1
        JP      P1
L1:
        ENDM
```

(2) Sprung bei 'größer als oder gleich' (JGE):

```
JGE     MACRO   P2
        JRZ     P2
        JP      P2
        ENDM
```

(3) Sprung bei 'kleiner als' (JLT):

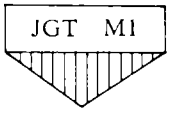
```
JLT     MACRO   P3
        LOCAL   L2
        JRZ     L2
        JMP     P3
L2:
        ENDM
```

(4) Sprung bei 'kleiner als oder gleich' (JLE):

```
JLE     MACRO   P4
        JRZ     P4
        JMP     P4
        ENDM    .
```

Ein Beispiel einer Makroabarbeitung ist dann

```
JGT     MACRO   P1
        LOCAL   P1
        JRZ     L1
        JP      P1
L1:
        ENDM
```




```
        JRZ     ..0000
        JP      M1
..0000:
```

Übungsaufgaben

3.1. Welche Anweisungsfolge generiert MACRO-80 bei der Abarbeitung der folgenden Anweisungen zur bedingten Assemblierung?

```
X   SET     0
    IFT     X+1
    IRP     Y,<B,C,D,E,F,G,H>
    LD      Y,A
    ENDM
```

```
        ELSE
        IRPC    Z,<ABCDEFGH>
        LD      Z,B
        ENDM
        ENDIF
```

3.2. Geben Sie das Generierungsergebnis der folgenden Zeilen, die Aufrufe der in 3.2.3. gegebenen Makrodefinitionen enthalten, an!

```
        CPI     3
        JGE     M0
        MVI     A,4
        CPI     B
        JLT     M1
        NOP
        CPI     2
        JLE     M2
```

3.3. Schreiben Sie eine Makrodefinition in MACRO-80, die die Anweisungsfolge

```
        INC     A
        INC     B
        INC     C
        INC     D
```

durch den Aufruf 'AUFG3 ABCD' erzeugt.

3.3. ASSEMBLER-Makrosprache MACRO-SM

3.3.1. Bedingte Assemblierung

Die Makrosprache MACRO-SM wurde für den Minicomputer SM4 bzw. SM3 geschaffen. Die Anweisung zur bedingten Generierung von Befehlen in der ASSEMBLER-Sprache der oben genannten Minicomputer lautet in MACRO-SM

```
.IF     bedg , arg1[,arg2]
  :
.ENDC
```

Dabei werden die Anweisungen zwischen '.IF' und '.ENDC' bearbeitet beziehungsweise in das generierte Quellprogramm übernommen, wenn die Bedingung *bedg* über die Argumente *arg1* und eventuell *arg2* erfüllt ist. Die dazwischen liegenden Zeilen können auch selbst Anweisungen der bedingten Assemblierung darstellen. Für *bedg* kann in MACRO-SM verwendet werden

DF – wahr, wenn *arg1* (als Variable) definiert ist;

NDF – wahr, wenn *arg1* nicht definiert ist;

B – wahr, wenn *arg1* leer ist;

NB – wahr, wenn *arg1* nicht leer ist;

EQ – wahr, wenn *arg1* (als numerische Variable) gleich Null ist;

NE – wahr, wenn *arg1* ungleich Null ist;

GT – wahr, wenn *arg1* größer als Null ist;

LE – wahr, wenn *arg1* kleiner oder gleich Null ist;

GE – wahr, wenn *arg1* größer oder gleich Null ist;

LT – wahr, wenn *arg1* kleiner als Null ist;

IDN – wahr, wenn *arg1* gleich *arg2* ist;

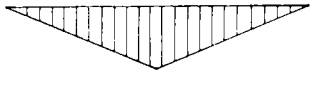
DIF – wahr, wenn *arg1* ungleich *arg2* ist.

Ist für die Bedingungsformen 'DF' bis 'LT' ebenfalls *arg2* angegeben, so lautet jeweils die Bedingung '...*arg1* **oder** *arg2*'. Als Trennzeichen zwischen *bedg* und *arg1* dürfen neben dem Komma auch Leerzeichen verwendet werden.
Soll durch die bedingte Anweisung nur eine Anweisung der Basissprache erzeugt werden, so kann die Kurzform

.IFF *bedg*, *arg1*[,*arg2*], *anweisung*

geschrieben werden.
Innerhalb der .IF-Anweisung können weitere bedingte Anweisungen stehen, die sich auf *bedg* beziehen. Die .IFT-Anweisung bewirkt, daß die ihr folgenden Anweisungen übernommen bzw. generiert werden, wenn *bedg* wahr ist. Die .IFF-Anweisung setzt die Nichterfüllung von *bedg* voraus, während nach '.IFTF' die Zeilen übernommen bzw. generiert werden unabhängig von *bedg*. Das folgende Beispiel zeigt die Anwendung dieser Anweisungen zur bedingten Assemblierung.

```
B:      .BLKW   2
A=0
        .IF     EQ,A
        MOV     #B,R2
        CLR     (R2)+
        .IFF
        MOV     #A,R3
        .IFT
        MOV     #B,R3
        .IFTF
        CLR     (R3)+
        .ENDC
        .END
```



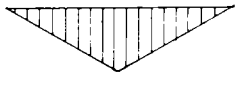
```
        MOV     #B,R2
        CLR     (R2)+
        MOV     #B,R3
        CLR     (R3)+
```

Durch '.BLKW 2' wird B als Variable der Basissprache definiert, während '.END' die ASSEMBLER-Endeanweisung darstellt. Durch die Zuweisung 'A=0' erhält die numerische Variable der bedingten Assemblierung den Wert Null.
Für die wiederholte Generierung von Anweisungen der Basissprache gibt es auch bei MACRO-SM drei Anweisungen.
Die Anweisung

```
.IRP    arg,<par1,par2,...,par n>
 :
.ENDM
```

erzeugt die im Inneren liegenden Anweisungen so oft, wie *par i* vorhanden sind (also *n*-mal). Dabei wird die Variable *arg* der Reihe nach mit den *par i* belegt und kann zur Zeichenkettenmanipulation verwendet werden.
Ein Beispiel für die Anwendung der .IRP-Anweisung lautet

```
.IRP    Q,<A,B,C>
MOV     #Q,(R3)+
.ENDM
```



```
MOV     #A,(R3)+
MOV     #B,(R3)+
MOV     #C,(R3)+
```

Die zweite Anweisung für die zyklische Generierung hat die allgemeine Form

```
.IRPC   arg,<kette>
 :
.ENDM
```

Die Anzahl der wiederholten Generierung ist hier durch die Länge der Zeichenkette *kette* bestimmt. Über *arg* werden hier die einzelnen Zeichen von *kette* für eine Manipulation der zu erzeugenden Basissprachanweisungen zur Verfügung gestellt. So ergibt beispielsweise

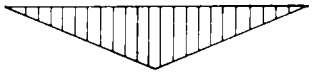
```
.IRPC   Q,ABC
MOV     #Q,(R3)+
.ENDM
```

das bereits im vorherigen Beispiel genannte Generierungsergebnis. Die dritte Anweisung zur zyklischen Generierung ist schließlich

```
.REPT   ausdruck
 :
.ENDM .
```

Dabei wird die Anzahl der Wiederholung durch den arithmetischen Ausdruck *ausdruck* bestimmt. Eine einfache Anwendung der .REPT-Anweisung lautet

```
A=5
        .REPT   A-3
        MOV     R3,#'1
        INC     R3
        .ENDM
```



```
MOV     R3,#'1
INC     R3
MOV     R3,#'1
INC     R3
```

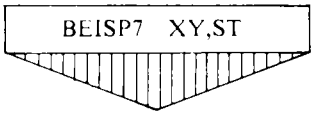
3.3.2. Makrodefinitionen in MACRO-SM

Eine Makrodefinition in MACRO-SM hat die allgemeine Form

```
.MACRO   name par1,par2,...,par n
  ⋮
.ENDM     .
```

Dabei definiert *name* den Makronamen, und *par i* ($1 \leq i \leq n$) stellen die formalen Parameter dar. Der Makrokörper kann vor der .ENDM-Anweisung durch die Verwendung von '.MEXIT' verlassen werden. Wird ein formaler Parameter durch das Zeichen '?' gekennzeichnet, so erhält dieser, falls für ihn im Makroaufruf kein Wert angegeben wurde, den Zeichenkettenwert '64□'. Für jeden weiteren derartig gekennzeichneten formalen Parameter wird dann '65□', '66□' usw. (bis '127□') erzeugt. Diese Zeichenkettenwerte sind beispielsweise für Marken der Basissprache anwendbar. Eine Anwendung zeigt das folgende Beispiel einer Makrodefinition und deren Abarbeitung in MACRO-SM.

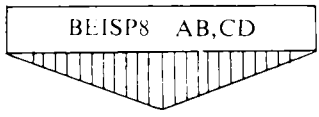
```
        .MACRO   BEISP7   P1,P2,?P3
P3:     MOV      #P1,R0
        MOV      #P2,R2
        BNE      P3
        .ENDM
```




```
64□:    MOV      #XY,R0
        MOV      #ST,R2
        BNE      64□
```

Im Makrokörper können auch die Anweisungen der bedingten Assemblierung verwendet werden. Das Makro BEISP8 ist ein Beispiel dafür.

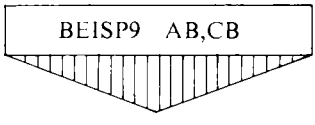
```
.MACRO   BEISP8   P1,P2
.IRPC    Q2,P1'P2
MOVB     #"Q2,(R4)+
.ENDM
.ENDM
```




```
MOVB    #'A,(R4)+
MOVB    #'B,(R4)+
MOVB    #'C,(R4)+
MOVB    #'D,(R4)+
```

Es zeigt auch die Anwendung des Verkettens von Parameterwerten durch einen Apostroph bzw. das Verketten der Variablen Q2 der .IRPC-Anweisung mit der Zeichenfolge '#" in derselben Weise. In MACRO-SM ist ebenfalls die statische Makroerweiterung, wie das folgende Makrobeispiel zeigt, möglich.

```
.MACRO   BEISP9   P1,P2
.IRP     Q1,<P1,P2>
MOV      #Q1,R0
.ENDM
BEISP8   P1,P2
.ENDM
```




```
MOV     #AB,R0
MOV     #CD,R0
MOVB    #'A,(R4)+
MOVB    #'B,(R4)+
MOVB    #'C,(R4)+
MOVB    #'D,(R4)+
```

Dabei wurde die Makrodefinition für BEISP8 als gegeben vorausgesetzt.
In MACRO-SM dürfen Makrodefinitionen auch ineinander verschachtelt auftreten. So liefert die Makrodefinition

```
.MACRO   BEISP10   P1,P2
.IRP     Q1,<P1,P2>
MOV      #Q1,R0
.ENDM
.MACRO   BEISP8    P1,P2
.IRPC    Q2,P1'P2
MOVB     #"Q2,(R4)+
.ENDM
```

```
.ENDM
BEISP8      P1,P2
.ENDM
```

mit dem Makroaufruf 'BEISP10 AB,CD' dasselbe Generierungsergebnis wie im vorherigen Beispiel.
Innerhalb des Makrokörpers können in MACRO-SM Hilfsfunktionen folgenden Inhalts angewendet werden

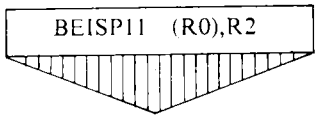
.NARG *symbol* zur Bestimmung der Anzahl der aktuellen Parameterwerte und Zuweisung dieser Anzahl zur Variablen *symbol*;

.NCHR *l*,⟨*kette*⟩ zur Bestimmung der Länge der Zeichenkette *kette* und Zuweisung dieses Wertes zur Variablen *l*;

.PRINT [*ausdruck*]; *text*
zur Protokollierung des Wertes von *ausdruck* und dem *text*; wenn *text* eine Fehlernachricht darstellt, kann für '.PRINT' auch '.ERROR' verwendet werden.

Eine Anwendung dieser Hilfsfunktionen ist im folgenden gegeben.

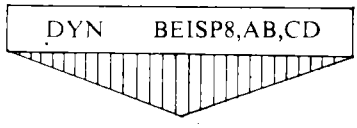
```
.MACRO    BEISP11   P1,P2
.NARG     Q1
.IF       LT,Q1-2
.ERROR    ;FEHLENDE PARAMETERANGABEN
.MEXIT
.ENDC
.NCHR     Q2,⟨P1⟩
.NCHR     Q3,⟨P3⟩
.IF       GT,Q3-Q2
ADD       P1,P2
.IFF
ADD       P2,P1
.ENDC
.ENDM
```




```
ADD       R2,(R0)
```

In MACRO-SM können auch rekursive Makrodefinitionen realisiert werden. Weiterhin kann auch die dynamische Makroerweiterung angewendet werden. Zum Beispiel in der Weise

```
.MACRO    DYN       R1,R2,R3
R1        R2,R3
.ENDM
```




```
MOVB    #'A,(R4)+
MOVB    #'B,(R4)+
MOVB    #'C,(R4)+
MOVB    #'D,(R4)+
```

Dabei wurde die Makrodefinition BEISP8 als gegeben vorausgesetzt. In MACRO-SM besteht auch die Möglichkeit, Makrodefinitionen aus Makrobibliotheken in das Makroprogramm einzufügen. Das geschieht mittels der Anweisung

.MCALL *makroname* .

Für *makroname* kann auch eine Liste von Makronamen angegeben werden.

3.3.3. Zum Problem der Realisierung der Portabilität

Als Beispiel für die Übertragung einer Programmiersprache soll hier die Übertragung der Makrosprache MACRO-80 auf den Computer SM4 an Hand einer Anweisung der bedingten Assemblierung von MACRO-80 behandelt werden. Für diese Übertragung soll die Sprache MACRO-SM genutzt werden. Die zu übertragende Anweisung sei die IF-Anweisung, deren allgemeine Form und ihre Entsprechung in MACRO-SM lautet

MACRO-80	MACRO-SM
IF*bedg,arg1*[,*arg2*]	.IF *bedg,arg1*[,*arg2*]
⋮	⋮
ELSE	.IFF
⋮	⋮
ENDIF	.ENDC

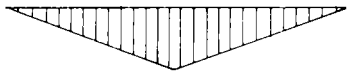
Makrodefinitionen in MACRO-SM zur Umsetzung der IF-Anweisungen von MACRO-80 mit *bedg*='F' und *bedg*='NDEF' (also der Anweisungen IFF und IFNDEF) haben beispielsweise die Form

```
.MACRO   IFF      P1
.IF      EQ,P1
.ENDM
.MACRO   IFNDEF   P2
.IF      NDF,P2
.ENDM
.MACRO   ENDIF
.IFTF
.ENDC
.ENDM
```

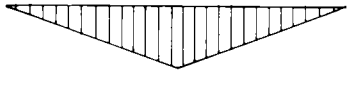
Dabei ist die Einfügung der .IFTF-Anweisung für die ordnungsgemäße .ENDC-Erzeugung notwendig.

Eine mögliche Anwendung, die diese Makros als gegeben voraussetzt, lautet dann

```
A=0
        IFF         A
        CLR         (R2)+
        ENDIF
        IFNDEF      B
        MOVB        #'X,(R1)+
        ENDIF
```



```
A=0
        .IF         EQ,A
        CLR         (R2)+
        .IFTF
        .ENDC
        .IF         NDF,B
        MOVB        #'X,(R1)+
        .IFTF
        .ENDC
```



```
        CLR         (R2)+
        MOVB        #'X,(R1)+
```

Damit ist jedoch die Übertragung der IF-Anweisung von MACRO-80 nicht allgemeingültig gelöst. Ist nämlich im obigen Beispiel bereits die erste Bedingung nicht erfüllt, das heißt, $A \neq 0$, so werden alle anderen Zeilen, da sie keine MACRO-SM-Anweisungen enthalten, ignoriert. Dieses Beispiel verdeutlicht die Problematik der Verwendung vollständig integrierter Makroprozessoren für die Realisierung der Portabilität, ohne die Basissprache einzubeziehen.

Übungsaufgaben

3.4. Welche Zeilen werden nach der Abarbeitung der folgenden MACRO-SM-Anweisungen erzeugt?

```
A=5
B=3
        .IF         GT,A-B
        .REPT       A-B
        CLR         (R2)+
        .ENDM
        .IFF
        .IRPC       Q1,WERT
        MOVB        #'Q1,(R1)+
        .ENDM
        .ENDC
```

3.5. Welche ASSEMBLER-Anweisungen liefert das Makro

```
        .MACRO ZYKL    P1,P2,?P3,?P4
        MOV    C,#P1
        MOV    HL,#P2'P1
P4:     CMP    #A,HL
        BEQ    P3
        INC    HL
        DEC    C
        BNE    P4
P3:     .ENDM
```

mit dem Makroaufruf ZYKL FELD,B ?

3.6. Schreiben Sie eine Makrodefinition, die die ASSEMBLER-Anweisungen

```
ADD     R1,(R5)+
ADD     R2,(R5)+
ADD     R3,(R5)+
ADD     R4,(R5)+
```

durch den Aufruf 'AUFG3 1234' generiert!

3.4. ASSEMBLER-Makrosprache MACRO-OS

3.4.0. Allgemeines

Die Makrosprache MACRO-OS ist für mittlere Computer, und zwar ESER-Computer, implementiert. Sie wird im allgemeinen als ASSEMBLER-Makrosprache des Betriebssystems OS/ES ([145]) bezeichnet und soll hier abkürzend MACRO-OS genannt werden. Im folgenden wird diese Makrosprache näher beschrieben. Dabei wird (wie bereits auch bei den beiden vorangegangenen ASSEMBLER-Makrosprachen) der Sprachumfang nur so weit dargestellt, wie er nicht speziellere Kenntnisse der jeweiligen ASSEMBLER-Sprache (als Basissprache) erfordert.

3.4.1. Bedingte Assemblierung

Innerhalb der bedingten Assemblierung können Variablen als sogenannte SET-Symbole definiert werden. Dabei ist es möglich, sogenannte lokale Symbole, die beispielsweise nur innerhalb einer Makrodefinition gelten, und globale Symbole zu benennen. Arithmetische Variablen werden mit LA, logische bzw. binäre Variablen mit LB und Zeichenkettenvariablen mit LC gekennzeichnet. So definiert

```
LCLA    &A
```

die lokale arithmetische Variable &A, während die globale Variable &B als Zeichenkette durch die Anweisung

```
GBLC    &B
```

benannt ist. Das Zeichen '&' dient der Kennzeichnung der Variablen in MACRO-OS überhaupt. Variablen können auch als einfach indizierte Felder definiert werden. So werden beispielsweise durch

```
LCLB    &X(5)
```

die lokalen logischen Variablen &X(1), &X(2), ..., &X(5) definiert. Werte werden den Variablen durch die SET-Anweisung zugewiesen. Die Wertzuweisung der Zeichenkette 'ABC' zur Zeichenkettenvariablen &B wird durch

```
&B      SETC    'ABC'
```

realisiert. Für arithmetische Variablen ist dabei 'SETA' und für logische 'SETB' zu verwenden.
Das Verketten von Zeichenkettenvariablen wird durch einen Punkt gekennzeichnet. Hat beispielsweise die Zeichenkettenvariable &A den Wert 'WORT' und &B den Wert 'HALB', so wird durch '&B.&A' die Zeichenkette 'HALBWORT' gebildet. Bei der Verkettung einer Variablen mit einer Zeichenkette ist dieser Punkt nur dann anzugeben, wenn die explizit angegebene Zeichenkette nach der Variablen folgt. Dieselbe oben genannte Zeichenkette wird also auch durch 'HALB&A' bzw. '&B.WORT' erzeugt.
Durch Angabe der Position und Länge in geklammerter Form im Anschluß an eine Zeichenkette kann eine Teilzeichenkette aus dieser gebildet werden. So ergibt

```
'HALBWORT'(5,4)
```

die Zeichenkette 'WORT'. Wie bei der Wertzuweisung für Zeichenkettenvariablen bereits zu erkennen ist, werden explizit angegebene Zeichenketten in MACRO-OS stets durch Apostrophe eingegrenzt. Bei der Teilzeichenkettenbildung ist zu beachten, daß die Längenangabe stets kleiner oder gleich 8 sein muß.
Zur unmittelbaren bedingten Generierung von Anweisungen in der Basissprache dienen die MACRO-OS-Anweisungen

AGO *marke*

als Sprunganweisung zu *marke*; *marke* ist dabei eine stets mit einem Punkt beginnende Bezeichnung;

AIF (*log.ausdruck*) *marke*

als Sprunganweisung zu *marke* bei erfülltem *log.ausdruck*; für *log.ausdruck* sind mit dem logischen Und ('AND') und dem logischen Oder ('OR') verbundene Vergleichsausdrücke einzusetzen; als Vergleichssymbole können dabei verwendet werden: EQ('='), NE ('≠'), LT('<'), GT ('>'), LE ('≦') und GE ('≧');

ANOP

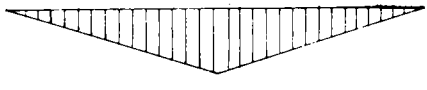
als sogenannte Leeranweisung, die insbesondere durch ihre Markierung (diese Möglichkeit besteht für alle hier genannten Anweisungen) als Sprungziel einer AGO- oder AIF-Anweisung dienen kann;

ACTR *arithm.ausdruck*

zur Beschränkung der möglichen AGO- oder AIF-Sprünge innerhalb einer bedingten Assemblierung (also zum Beispiel zur Vermeidung nichtabbrechender zyklischer Generierungsabläufe) auf eine sich durch den arithmetischen Ausdruck *arithm.ausdruck* ergebende Anzahl.

Eine einfache Anwendung aus den bisher genannten Anweisungen zur bedingten Assemblierung zeigt das folgende Generierungsbeispiel.

```
        LCLC    &Y(4)
        LCLA    &X
&X      SETA    0
&Y(1)   SETC    'BER1'
&Y(2)   SETC    'SATZ'
&Y(3)   SETC    'FELD'
&Y(4)   SETC    'BER2'
.M1     AIF     (&X EQ 4).M2
&X      SETA    &X+1
        L       7,0(&X)
        ST      7,&Y(&X)
        AGO     .M1
.M2     ANOP
```



```
        L       7,0(1)
        ST      7,BER1
        L       7,0(2)
        ST      7,SATZ
        L       7,0(3)
        ST      7,FELD
        L       7,0(4)
        ST      7,BER2
```

3.4.2. Makrodefinitionen in MACRO-OS

Makrodefinitionen in MACRO-OS haben die allgemeine Form

```
        MACRO
[&marke] name       par1,par2,...,par n
         :
        MEND  .
```

Die zweite Zeile, die wahlweise mit einer Markierung (*&marke*) versehen werden kann und den Makroaufruf definiert, wird auch Musteranweisung genannt. Durch *name* wird die Makrobezeichnung und durch *par i* ($1 \leq i \leq n$) werden die formalen Parameter, die stets mit dem Zeichen '&' zu beginnen haben und vom Stellungs- oder Kennworttyp sein können, bezeichnet. Beim Stellungstyp erfolgt eine einfache Auflistung der Parameter, also zum Beispiel

```
BEISP12  &A,&X,&C  ,
```

die die Reihenfolge der Parameterwerte für den Makroaufruf festlegt. So bewirkt beispielsweise der Makroaufruf

```
BEISP12   5,,ABC
```

die Parameterwertzuweisungen &A=5, &X leer und &C='ABC'. Beim Kennworttyp wird der Parameterbezeichnung ein Gleichheitszeichen angehängt, also zum Beispiel

```
BEISP13   &A=,&X=,&C=  .
```

Der Makroaufruf, der die obige Parameterwertzuweisungen analog BEISP12 realisiert, lautet dann hierfür beispielsweise

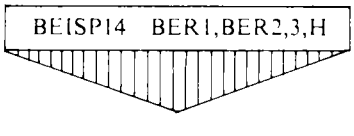
```
BEISP13   C='ABC', A=5  .
```

Die Reihenfolge der Parameterwertzuweisungen ist dabei beliebig. Analog der Zeichenkettenvariablen in MACRO-OS können auch Parameter durch einen Punkt mit anderen Parametern bzw. Zeichenketten verkettet werden.
Die Parametertypfestlegungen können innerhalb einer Makrodefinition auch gemischt auftreten.
Als Parameterwert im Makroaufruf können auch sogenannte Wertetupel verwendet werden. Der Zugriff zu den einzelnen Werten im Makrokörper erfolgt dann durch Indizierung des Parameternamens. So können beispielsweise beim Makroaufruf

```
BEISP12   (1,5,3),,ABC
```

im Makrokörper durch &A das gesamte Wertetupel und durch &A(1), &A(2) und &A(3) die einzelnen Werte 1, 5 und 3 verarbeitet werden. Durch die MEXIT-Anweisung kann der Makrokörper vor der MEND-Anweisung verlassen werden.
Eine Protokollierung von Fehlernachrichten während der Makroverarbeitung realisiert die MNOTE-Anweisung.
Ein einfaches Beispiel einer Makroverarbeitung in MACRO-OS lautet

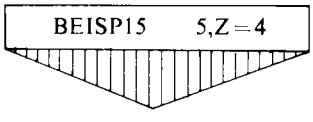
```
MACRO
BEISP14   &A,&B,&X,&Y
ST        &X,&Y
L         &X,&A
ST        &X,&B
L         &X,&Y
MEND
```




```
ST        3,H
L         3,BER1
ST        3,BER2
L         3,H
```

Auch in MACRO-OS sind die Anweisungen zur bedingten Assemblierung im Makrokörper anwendbar, wie das folgende Makrobeispiel zeigt.

```
         MACRO
         BEISP15   &X,&Y=SATZ,&Z=8
         LCLA      &I
&I       SETA      0
.M1      AIF       (&I EQ &X).ME
&Y.&I    DS        CL&Z
&I       SETA      &I+1
         AGO       .M1
.ME      ANOP
         MEND
```




```
SATZ0    DS        CL4
SATZ1    DS        CL4
SATZ2    DS        CL4
SATZ3    DS        CL4
SATZ4    DS        CL4
```

In der Musteranweisung der Makrodefinition ist &X ein Stellungsparameter, während &Y und &Z Kennwortparameter mit den zugewiesenen Anfangswerten 'SATZ' und 8 darstellen. Beim Makroaufruf wird für die Kennwortparameter nur &Z neu festgelegt. Bei der Angabe des Parameternamens entfällt das Zeichen '&'.
Im Makrokörper können außerdem noch folgende Hilfsfunktionen verwendet werden:

&SYSNDX

zur Generierung einer Zahl als Zeichenkette, die bei jeder Anwendung in einem Makrokörper konstant bleibt, sich aber bei der Nutzung dieser Funktion in einem anderen Makrokörper innerhalb einer Makroverarbeitung um eins erhöht; ihr Wertebereich ist '0001', '0002', ..., '9999';

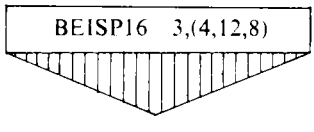
&SYSLIST(*n*)

zur Bereitstellung des *n*-ten Parameterwertes vom Makroaufruf; ist dieser Parameterwert ein Wertetupel, so können die einzelnen Werte dieses Tupels durch eine weitere Indizierung der &SYSLIST-Funktion erreicht werden, also &SYSLIST(*n*,*m*) für den *m*-ten Tupelwert des (als Wertetupel vorliegenden) *n*-ten aktuellen Parameters.

Eine Anwendung dieser Funktionen, sowie der MNOTE- und der MEXIT-Anweisung, zeigt das folgende Beispiel. Die beiden darin angegebenen Makrodefinitionen (BEISP16 und

BEISP17) mit einem Aufruf der einen innerhalb der anderen kennzeichnet die auch in MACRO-OS mögliche statische Makroerweiterung.

```
         MACRO
         BEISP16   &I,&J
         LCLA      &X
         AIF       (T'&I NE 'O').MA
         MNOTE     'FEHLENDER PARAMETERWERT'
         MEXIT
.MA      ANOP
&X       SETA      0
.MO      AIF       (&X EQ &I).ME
&X       SETA      &X+1
         BEISP17   &SYSLIST(2,&X)
         AGO       .MO
.ME      ANOP
         MEND
         MACRO
         BEISP17   &K
A&SYSNDX DS        CL&K
         MEND
```




```
A0002    DS        CL4
A0003    DS        CL12
A0004    DS        CL8
```

Der logische Ausdruck '(T'&I NE 'O')' in BEISP16 besitzt den Wert 'wahr', wenn für &I im Makroaufruf ein Parameterwert vorhanden ist.
Mit BEISP18 ist eine rekursive Makrodefinition realisiert.

```
         MACRO
         BEISP18   &A
         LCLA      &X
&X       SETA      &A
.MO      AIF       (&X EQ 0).ME
         ST        4,(4*&X)(5,6)
&X       SETA      &X-1
         BEISP18   &X
.ME      ANOP
         MEND
```


```
         ST      4,(4*4)(5,6)
         ST      4,(4*3)(5,6)
         ST      4,(4*2)(5,6)
         ST      4,(4*1)(5,6)
```

Eine dynamische Makroerweiterung ist in MACRO-OS nicht möglich.

3.4.3. Ein Beispiel zur Erweiterung der Makrosprache MACRO-OS

Im folgenden soll ein Beispiel zur Erweiterung der Makrosprache selbst angegeben werden. Es handelt sich um die Implementierung der Funktion INDEX, die bei der Makroprogrammierung in MACRO-OS genutzt werden kann. Der Aufruf dieser Funktion habe die Form

INDEX *ketvar1,ketvar2* .

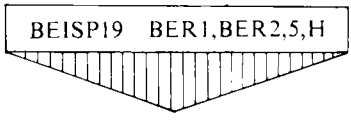
Durch INDEX wird die Position, ab der die Zeichenfolge der Zeichenkettenvariablen *ketvar2* in der Zeichenfolge der Zeichenkettenvariablen *ketvar1* vorkommt, bestimmt und einer global zu definierenden numerischen Variablen zugewiesen.
Eine mögliche Realisierung der Funktion INDEX als Makrodefinition in MACRO-OS lautet

```
         MACRO
         INDEX   &KET1,&KET2
         GBLA    &P
         LCLA    &I
         LCLA    &J
         LCLA    &K
         LCLC    &Z
&I       SETA    1
&J       SETA    K'&KET1
&K       SETA    K'&KET2
&P       SETA    0
.MA      AIF     (&I GT &J).ME
&Z       SETC    '&KET1'(&I,&K)
         AIF     (&Z EQ &KET2).MI
&I       SETA    &I+1
         AGO     .MA
.MI      ANOP
&P       SETA    &I
.ME      ANOP
         MEND
```

Die hierbei verwendete Teilzeichenkettenbildung schränkt die Anwendung der INDEX-Funktion in der Hinsicht ein, daß die Zeichenfolge in *ketvar1* nicht länger als 8 Zeichen sein darf.

Der Ausdruck 'K'&KET1' bestimmt die Länge der Zeichenfolge in &KET1. Die globale Größe, die die Position nach der INDEX-Abarbeitung enthält, ist hierbei &P. Dabei wird auch die Ausnahmesituation, daß &KET2 nicht in &KET1 enthalten ist, durch die Wertzuweisung für &P mit einer Null berücksichtigt.
Eine Anwendung dieser Funktion zeigt die Makrodefinition BEISP19. Dabei ist die Bereitstellung der INDEX-Makrodefinition und des oben definierten Makros BEISP14 vorausgesetzt.

```
         MACRO
         BEISP19   &A,&B,&C,&D
         GBLA      &P
         LCLC      &REG
&REG     SETC      '1347'
         INDEX     &REG,&C
         AIF       (&P, GT 0).M1
         BEISP14   &A,&B,1,&D
         AGO       .M2
.M1      BEISP14   &A,&B,&C,&D
.M2      ANOP
         MEND
```




```
         ST        1,H
         L         1,BER1
         ST        1,BER2
         L         1,H
```

Die INDEX-Funktion wird hierbei zur Prüfung eines Parameterwertes (für &C) bezüglich eines vorgegebenen Wertebereiches '1347' (also 1, 3, 4 oder 7) verwendet.

Für die neueren ESER-Rechner, wie ES 1055 und ES 1056, besitzt die ASSEMBLER-Makrosprache zum MACRO-OS folgende Erweiterungen

(1) die Befehle SET, AIF, AGO, LCL und GBL haben ein erweitertes Format:

[name] AGO (*h*) .*s1*, .*s2*, ..., *sn*
mit *h* als arithmetischem Ausdruck,

dessen (jeweiliger) ganzzahliger Werte das entsprechende Sprungziel .*si* auswählt

[name] AIF (*l1*) .*d1*, (*l2*) .*d2*, ..., (*ln*) .*dn*

mit *li* als logische Ausdrücke, deren erste Erfüllung den Sprung zu . *di* bewirkt

i-name SETA *a1*, *a2*, ..., *an*

mit *i-name* als indiziertes Symbol und Zuweisung einer Anfangswertemenge

(2) die Attributangabe D für den Test, ob eine (Makro-)Variable definiert ist oder nicht

... D' ⟨name⟩

(3) die Möglichkeit der weiteren Parameterlistenverschachtelung und den Zugriff zu den Parameterwerten mittels

&SYSLIST(*i*, *j*, *k*, ...)

(4) die Möglichkeit der zusätzlichen Dateneingabe mittels

```
name  AREAD
```

welches die dieser Zeile folgenden Zeile in die SET-Variable *name* liest

(5) die Möglichkeit der Makroabarbeitungsverfolgung mittels

```
[name]  MHELP  pf
```

wobei *pf* eine bestimmte Art kennzeichnet (z.B. nur die AGO-Anweisungen zu protokollieren usw.)

Übungsaufgaben

3.7. Welche Zeichenketten ergeben

a) die Zeichenkettenausdrücke

'MAKROTECHNIK'(6,7)
'BEISPIEL'(4,5)
'ASSEMBLER'(1,2)

b) die folgenden Verkettungen

'MAKRO'.'TAUSEND'(2,2).'FRUEH'(1,3).'F'
und
'QUELLTEXT'(6,4).'TEIL' ?

3.8. Welche ASSEMBLER-Anweisungen erzeugen die folgenden MACRO-OS-Anweisungen der bedingten Assemblierung

```
         LCLA   &A
         LCLC   &B(2)
&A       SETA   5
&B(1)    SETC   8
&B(2)    SETC   00
.ZYKL    AIF    (&A EQ 1).ZENDE
&A       SETA   &A-1
         L      &B(1),0(&A)
         ST     &B(1),&B(2).&A
         AGO    .ZYKL
.ZENDE   ANOP
```

3.9. Von welchem Typ sind die nachfolgenden Musteranweisungen?

a) SUMME &A,&B,&C
b) WERT &A=,&CD
c) BER &P1=,&P2=,&P3=

3.10. Welche Zeilenfolge wird durch die Makrodefinition

```
         MACRO
         DEF    &A,&B,&C
         LCLA   &K
&A       DC     X'00'
&K       SETA   &B
.ANF     AIF    (&K EQ 0).END
&C.&K    DS     CL&K
&K       SETA   &K-1
         AGO    .ANF
.END     ANOP
         MEND
```

durch den Makroaufruf DEF A,5,B erzeugt?

3.11. Schreiben Sie eine Makrodefinition, die mit dem Makroaufruf AUFG5 4,(A,B,C,D,E,F,G,H) die ASSEMBLER-Anweisungen

```
L    0,A
ST   0,B
L    1,C
ST   1,D
L    2,E
ST   2,F
L    3,G
ST   3,H
```

generiert!

4. PL/1-Makroprogrammierung

4.0. Einleitende Bemerkungen

Die PL/1-Makroprogrammierung gehört – im Gegensatz zu der im vorherigen Abschnitt behandelten – zur Makrotechnik für höhere Programmiersprachen. Sie ist prinzipiell darauf ausgerichtet, als Basissprache PL/1 zu generieren. Im Rahmen des PL/1-Compilers für mittlere Computer (wie beispielsweise ESER-Computer) mit dem Betriebssystem OS/ES ist der PL/1-Makrointerpreter nutzbar. Seiner Arbeitsweise nach stellt er einen teilweise integrierten Preprozessor dar. Formal gilt

PL/1-Makrointerpreter
MACRO-PL1,PL/1,ASSEMBLER.

Die Makrosprache des PL/1-Makrointerpreters sei hier abkürzend mit MACRO-PL1 bezeichnet. Für die drei Grundvarianten des PL/1-Compilers (dem Normalcompiler, dem Optimierungscompiler und dem Testcompiler) existieren zwei PL/1-Makrosprachvarianten [eine für den Normalcompiler (MACRO-PL1) und eine für die beiden anderen Compiler (hier MACRO-PL1E, – als erweiterte MACRO-PL1 – genannt)]. Im folgenden sollen zunächst die wesentlichsten Charakteristika der Makrosprache MACRO-PL1 erläutert werden.

4.1. Makrosprache MACRO-PL1

Die Makrosprache MACRO-PL1 gliedert sich in

(a) Vereinbarungen von Variablen; als Deklaration bzw. als sogenanntes Aktivieren;
(b) Anweisungen zur Textgenerierung mit Hilfe der Variablen.

Als Variablen können sowohl Zeichenkettenvariablen als auch numerische Variablen deklariert werden. Die Deklaration einer Zeichenkettenvariablen A lautet beispielsweise

%DECLARE A CHARACTER; (oder kurz: %DCL A CHAR;).

Die Länge der Zeichenkette aus A kann dabei im Verlauf der Makroabarbeitung unterschiedlich und im speziellen auch leer sein. Die Vereinbarung einer numerischen Variablen X lautet (bereits in der Kurzform)

%DCL X FIXED;

Der Wertebereich für X sind die ganzen Zahlen von −99999 bis +99999.
Die gezeigten Beispiele kennzeichnen bereits die Form der Anweisungen in MACRO-PL1, die stets mit dem Zeichen '%' beginnen und mit einem Semikolon enden.
Sollen mehrere Variablen zugleich vereinbart werden, so kann man zusammenfassend schreiben

%DCL(A,B,C)CHAR;

oder auch

```
%DCL(A,B)CHAR,X FIXED,Z CHAR;
```

Die zweite Form der Nutzbarmachung von Variablen für die Makroverarbeitung ist die Aktivierung, die beispielsweise für die Variable K die Form hat

```
%ACTIVATE K; (oder kurz: %ACT K;) .
```

Sie dient im allgemeinen der Wiederverwendung der Variablen K im Makroprogramm, nachdem diese zuvor inaktiv (%DEACTIVATE K; bzw. %DEACT K;) gesetzt wurde. Ihren Typ erhält die Variable K durch eine vorherige Deklaration obiger Art.
Über diese Variablen können sogenannte Ausdrücke formuliert werden; so zum Beispiel als arithmetischen Ausdruck

```
Y*(Z+3) .
```

Dabei sind als Operationszeichen '+', '−', '*' (für die Multiplikation) und '/' (für die Division) zugelassen. Die Klammerung erfolgt im mathematischen Sinne, jedoch mit ausschließlich runden Klammern.
Darüber hinaus besteht die Möglichkeit der Formulierung von logischen Ausdrücken, und zwar als Vergleiche, wie zum Beispiel

```
X<=Y, A¬=3   oder   C='NNN'
```

('<=' steht dabei für '≦' und '¬=' für '≠'), oder als mit dem logischen Und (als '&') oder dem logischen Oder ('|') verbundene Vergleiche, wie zum Beispiel

```
X<=Y & A¬=3 | C='NNN' .
```

Dabei ist jeweils die Präferenz vom Und gegenüber dem Oder zu beachten.
Zeichenketten werden in MACRO-PL1 in ihrer expliziten Darstellung durch Apostrophe begrenzt. Soll ein Apostroph in einer Zeichenkette enthalten sein, so ist er doppelt zu schreiben, also für A'C ist 'A''C' anzugeben.
Die dritte Art von Ausdrücken in MACRO-PL1 sind die Zeichenkettenausdrücke. Sie enthalten als Operationszeichen die Verkettung ('||'). Ein Zeichenkettenausdruck ist beispielsweise

```
A ||'WERT'||C .
```

Die Möglichkeit, den Variablen Werte zuzuweisen, ist durch die Ergibtanweisung gegeben. Die Anweisung

```
%A='WERT 1';
```

weist beispielsweise der Zeichenkettenvariablen A die Zeichenfolge 'WERT 1' zu. Ist auf der rechten Seite der Ergibtanweisung ein Ausdruck vorhanden, so wird zuerst dieser Ausdruck berechnet (interpretiert) und dann die Zuweisung zu der auf der linken Seite der Ergibtanweisung stehenden Variablen vorgenommen. So ergeben die Anweisungen für die numerischen Variablen R und S

```
%R=38;
%S=R/3;
%R=R-20000;
```

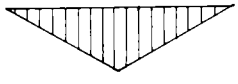
als Werte für R und S −19962 und 12.

Dabei können die Anweisungen auch hintereinandergeschrieben werden, also beispielsweise

```
%R=38;%S=R/3;%R=R-20000; .
```

Zu berücksichtigen ist jedoch das sogenannte Lochkartenformat (80 Zeichen), welches für MACRO-PL1 das zweite bis 72. Zeichen umfaßt (also 71 Zeichen pro Zeile).
Die Ersetzung des Wertes einer Variablen wird durch eine einfache Angabe dieser Variablen erreicht. Beachtet man auch die stets vor den Anweisungen notwendige Vereinbarung der Variablen, so lautet ein vollständiges Beispiel einer Makroabarbeitung

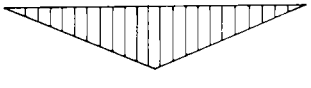
```
%DCL(A,C)CHAR;
%A='ZIEL';
%C='01';
%A=A||'WERT 1'||C;
A
```



```
ZIELWERT 101
```

Die Ersetzung wird in der Weise vorgenommen, daß jeweils rechts und links ein Leerzeichen eingesetzt wird, wie im folgenden Beispiel zu sehen ist.

```
%DCL B CHAR;
%B='01';
ZAHL-B UND WERT-B
```



```
ZAHL-␣01␣␣UND␣WERT-␣01
```

wobei '␣' das Leerzeichen kennzeichnet. Die Ersetzung einer Variablen kann also auch innerhalb eines Textes erfolgen. Voraussetzung dafür ist aber, daß diese Variable durch Zeichen ungleich A, B, C, ..., Z, #, ¢ und @ begrenzt ist.
Bei der Formulierung von Anweisungsfolgen in MACRO-PL1 kann es auftreten, daß die jeweiligen Zahlen in Zeichenketten bzw. die Zeichenketten in Zahlen umzuwandeln sind. Eine Zeichenkette in eine Zahl umzuwandeln ist nur dann möglich, wenn die Zeichenkette aus einer Ziffernfolge besteht, die im Innern keine Leerzeichen enthält. Die Anweisungsfolge

```
%DCL X FIXED; %X='23␣334';
```

würde also zu einem Fehler bei der Makroabarbeitung führen.
Die Umwandlung einer Zahl in eine Zeichenkette ist stets möglich. Dabei ist die erhaltene Zeichenkette genau acht Zeichen lang. So ergeben zum Beispiel

```
%DCL A CHAR; %A=46;
```

für A die Zeichenfolge '␣␣␣␣␣␣46' und

```
%DCL B CHAR; %B=99999;
```

die Zeichenfolge '␣␣␣99999', wobei '␣' wiederum für ein Leerzeichen steht.

Innerhalb eines Makroprogramms können in MACRO-PL1 Anweisungen auch markiert werden. Die Marken haben dieselbe syntaktische Form wie die Variablen, nämlich eine mit einem Alphazeichen beginnende Folge von Alphazeichen und Ziffern, wobei als Alphazeichen die Symbole A, B, C, ..., Z, #, ○ und @ zusammengefaßt sind.
Eine markierte Ergibtanweisung hat beispielsweise die Form

```
%M1:X=3;
```

mit der Marke M1. Die Markierung kann auch mehrfach erfolgen, wie zum Beispiel

```
%M1:M2:M3:A=C||B;
```

Ziel der Markierung ist die gezielte Unterbrechung der Anweisungsreihenfolge und Fortsetzung an einer anderen (der markierten) Stelle im Makroprogramm. Dies gewährleistet die sogenannte Sprunganweisung. Ein Sprung zur Marke M1 hat dabei die Form

```
%GOTO M1;   oder auch   %GO TO M1;   .
```

Für die Auswahl von Teilzeichenketten steht in MACRO-PL1 die Funktion SUBSTR (**Substr**ing) zur Verfügung. Sie hat den allgemeinen Aufruf

SUBSTR(*zeichenkettenvariable,von,länge*) ,

wobei für *zeichenkettenvariable* ein entsprechender Variablenname, für *von* eine Zahl, von der ab die Teilzeichenkette zu bilden ist, und *länge* eine Zahl, die die Länge der Teilzeichenkette angibt, anzugeben sind. Die SUBSTR-Funktion darf in Ergibtanweisungen nur auf der rechten Seite stehen. Ein Beispiel einer Teilzeichenkettenbildung ist

```
%DCL(A,B)CHAR; %A='BEISPIEL';
%B=SUBSTR(A,4,5);   ,
```

nach dessen Abarbeitung die Variable B den Zeichenkettenwert 'SPIEL' besitzt. Erfolgt die Teilzeichenkettenbildung bis zum Ende der Ausgangszeichenkette, so kann die *länge*-Angabe entfallen. Die Anweisung

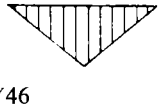
```
%B=SUBSTR(A,4);
```

liefert, auf das obige Beispiel bezogen, denselben Wert.
Eine weitere Möglichkeit der Makroprogrammverzweigung, und zwar in abgeschlossener Form, gibt die bedingte Anweisung in den beiden Formen

```
%IF A=1 %THEN C='EWERT';      bzw.
%IF X=Z %THEN K=R||'BEISPIEL'; %ELSE K='';
```

Dabei wird die Ergibtanweisung nach %THEN nur dann ausgeführt, wenn die angegebene Bedingung (hier 'A=1' bzw. 'X=Z') wahr ist. %ELSE steht für 'sonst'. Eine bedingte Textgenerierung hat dann in MACRO-PL1 beispielsweise die Form

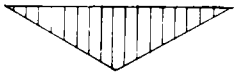
```
%DCL X FIXED, A CHAR;
%X=46;
%IF X>=10 %THEN %A='Y'||SUBSTR(X,8);
          %ELSE %A='Y'||SUBSTR(X,7);
A
```



```
Y46
```

Dieses Beispiel zeigt ebenfalls das Anketten einer Zahl, ohne einen Zwischenraum (also Leerzeichen) entstehen zu lassen. Es wird die generelle Länge einer Zahl als Zeichenkette, nämlich die Länge 8, berücksichtigt.
Ein weiteres Beispiel zeigt die mehrstufige Ersetzung von Makrovariablen, deren Zeichenkettenwerte wiederum Makrovariablen enthalten.

```
%DCL(A,B,D)CHAR;
%A='B||C';
%B='2';
%D='(A||E)';
X=A;
Y=A||B/D;
```



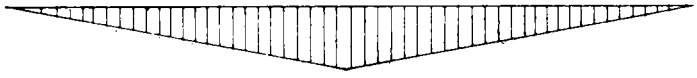
```
X= 2 ||C ;
Y= 2 ||C || 2 / ( 2 ||C ||E) ;
```

Hierin liegt aber auch ein gewisses Problem. Denn beispielsweise führt die Makroanweisungsfolge

```
%A='B||C';
%B=A;
```

mit den beiden Zeichenkettenvariablen A und B zu einer »unendlichen« Makroabarbeitung, also zum Fehler beim Ablauf auf dem Computer.
Das folgende Beispiel zeigt eine umfangreichere Ersetzung einer Anweisungsfolge in einer Programmiersprache (hier PL/1). Da dabei 'DCL' und 'CHAR' als Makrovariablen verwendet werden, ist für die Vereinbarung der Zeichenkettenvariablen die Langform von MACRO-PL1 zu verwenden.

```
%DECLARE(DCL,CHAR,FIXED,GET,EDIT,PUT,IF,THEN,
            ELSE,DO,END)CHARACTER;
%DCL='DEFINIERE';
%CHAR='ALS ZEICHENKETTE';
%FIXED='ALS ZAHL';
%IF='WENN'; %THEN='DANN'; %ELSE='SONST';
%GET='LIES'; %EDIT=''; %PUT='SCHREIBE';
%DO='BERECHNE'; %END='';
DCL A CHAR,X FIXED;
GET EDIT(X)(F(5));
IF X > 0 THEN DO;   A='POSITIVER WERT' ||X; END;
         ELSE DO;   X=2*X;
                    A='NEGATIVER VERDOPPELTER WERT'
                      ||X;END;
PUT EDIT(A)(A);
```



```
DEFINIERE  A  ALS ZEICHENKETTE ,X  ALS ZAHL ;
LIES  (X)(F(5));
WENN X > 0 DANN BERECHNE  ; A='POSITIVER WERT'||X;
           SONST BERECHNE  ; X=2*X;
                             A='NEGATIVER
                             VERDOPPELTER WERT'||X;
SCHREIBE  (A)(A);
```

Die Möglichkeit, eine Makroanweisungsfolge zusammenzufassen, ist durch die beiden Bezeichnungen 'DO' und 'END' gegeben. Eine solche Folge (Gruppe) lautet beispielsweise

```
%DO; %A='WERT 1'; %C='01'; %A=A||C; %END;
```

Diese Form muß zum Beispiel verwendet werden, wenn in einer bedingten Anweisung nach THEN oder ELSE mehr als eine Anweisung anzugeben sind. Das Präfix 'DO' dient aber auch noch der Kennzeichnung einer weiteren, für die Makrogenerierung bedeutungsvollen Anweisung, die zur zyklischen Texterzeugung angewandt wird. Ein Beispiel hierzu lautet

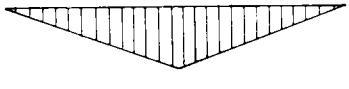
```
%DO I=1 TO N BY 3;
%X=A||I;
%END;
```

Die innere Anweisung ('%X=A||I;') wird dabei für die I-Werte von 1 bis (TO) N mit der Schrittweite (BY) 3 durchlaufen, also für $I=1,4,7,\ldots,\leq N$. Ohne die Angabe von 'BY' ist die Schrittweite stets 1. Es ist auch möglich, als Anfangswert eine Variable zu verwenden, also beispielsweise

```
%DO K=L TO M;
%Z=Z*K;
%END;
```

Die Anweisung '%Z=Z*K;' wird also hier für die Werte L,L+1,L+2,...,M der Größe K, die im allgemeinen als Laufvariable bezeichnet wird, durchlaufen.
Im Zyklus müssen aber nicht nur Makroanweisungen stehen, wie das folgende Beispiel zeigt.

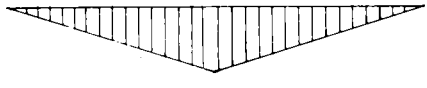
```
&DCL I FIXED;
%DO I=1 TO 6;
  Z(I)=Z(I) + X(I+1);
%END;
```



```
Z(bbbbbbbb1b)=Z(bbbbbbbb1b) + X(bbbbbbbb1b+1);
Z(      2 )=Z(      2 ) + X(      2 +1);
Z(      3 )=Z(      3 ) + X(      3 +1);
Z(      4 )=Z(      4 ) + X(      4 +1);
Z(      5 )=Z(      5 ) + X(      5 +1);
Z(      6 )=Z(      6 ) + X(      6 +1);
```

Ein weiteres Beispiel zeigt die zyklische Textgenerierung innerhalb einer Anweisungsfolge in der Basissprache (hier PL/1).

```
INIT: PROC (A,X);
      DCL A(5);
      %DCL(I,J)FIXED;
      %DO I=2 TO 10 BY 2;
      %J=I/2;
      A(J)=X**I;
      %END;
      END INIT;
```



```
INIT: PROC (A,X);
      DCL A(5);
      A(      1 )=X**      2 ;
      A(      2 )=X**      4 ;
      A(      3 )=X**      6 ;
      A(      4 )=X**      8 ;
      A(      5 )=X**     10 ;
```

Die Möglichkeit, in einem Makroprogramm einen Kommentar einzufügen, ist durch die Anweisung der Form

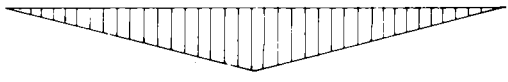
%/* *kommentarinhalt* */;

gegeben.

Eine Textgenerierung in MACRO-PL1 unter Anwendung der ACTIVATE- und DEACTIVATE-Anweisung hat beispielsweise die Form

```
      %DCL(A,B)CHAR,I FIXED;
      %I=0;
%M1:B='Z';
      %A='(C+D-B**2)';
      B=A*E/A;
      %IF I>=1 %THEN %GOTO M2;
      %DEACT A;
      %I=1;
      %GOTO M1;
```

```
%M2:IF I>=2 %THEN %GOTO ME;
    %I=I+1;
    %ACT A;
    %DEACT B;
    %GOTO M1;
    %ME:;
```



```
Z = (C+D- Z **2) *E/ (C+D- Z **2) ;
Z =A *E/A;
B= (C+D-B**2) *E/ (C+D-B**2) ;
```

Einfach ein Semikolon nach der Marke ME zu setzen, ist eine Anwendung der sogenannten Leeranweisung von MACRO-PL1. Sie lautet in ihrer einfachsten Form

```
%;  .
```

Eine weitere Möglichkeit der Zeichengenerierungsweise bietet die %INCLUDE-Anweisung. Sie bewirkt, daß an die Stelle im Makroprogramm, wo diese Anweisung geschrieben wurde, der jeweilige durch die %INCLUDE-Anweisung benannte Text eingefügt wird. Dabei werden stets ganze Zeilen übernommen. Die Textbenennung wird durch die Angabe des Bibliotheksnamens und des jeweiligen Bestandes in dieser Bibliothek vorgenommen. So führt beispielsweise die Anweisung

```
%INCLUDE BIBL1(MAKRO1);
```

zur Ersetzung des Textes vom Bestand 'MAKRO1' aus der Bibliothek 'BIBL1' an diese Stelle. Dabei ist es auch möglich, mehrere Texte als Bestände einzufügen, zum Beispiel durch

```
%INCLUDE BIBL1(MAKRO1),BIBL1(MAKRO2),
         BIBL2(TEXT1),BIBL1(MAKRO5);
```

Die Angabe des Bibliotheksnamens kann entfallen, wenn jener innerhalb der Abarbeitungsanweisungen für die Makrointerpretation (im Betriebssystem OS/ES die sogenannte SYSLIB-DD-Anweisung) definiert wurde.
Ist der ersetzte Text wiederum ein Makroprogramm, so erfolgt nach dem Einfügen dieses Textes die entsprechende Makroverarbeitung. Im folgenden Beispiel ist das mehrfache Einfügen eines Textes in der Weise gezeigt, daß der einzusetzende Text wiederum eine %INCLUDE-Anweisung enthält. Ein derartiger Text sei in der Bibliothek, die hier vorab definiert sei, als Bestand mit dem Namen 'DCL' und dem Inhalt

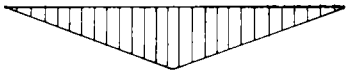
```
DCL 1 X,2 (K,L,M)CHAR,
    2 %INCLUDE LISTE;FIXED;
```

gegeben, wobei der Bestand LISTE beinhaltet

```
(A,B,C,D,E)  .
```

Eine konkrete Anwendung lautet dann

```
P:PROC(X);
%INCLUDE DCL;
GET EDIT %INCLUDE LISTE;(F(5));
Y=MAX %INCLUDE LISTE;;
PUT LIST(Y);
END P;
```



```
P:PROC(X);
DCL 1 X,2 (K,L,M)CHAR,
   2
    (A,B,C,D,E)
              FIXED;
   GET EDIT
    (A,B,C,D,E)
              (F(5));
   Y=MAX
    (A,B,C,D,E)
   ;
   PUT LIST(Y);
   END P;
```

Eine Makrodefinition ist in MACRO-PL1 durch die Definition und Anwendung der sogenannten Makroprozedur möglich. Dabei ist zunächst der Makroprozeduraufruf zu definieren und zwar in der Form

%DCL *prozname* ENTRY(*typ1,typ2,...,typ n*)
RETURNS(*typ r*);

Das Attribut ENTRY definiert *prozname* als Makroprozedur, deren Parameter vom Typ *typ1*, *typ2*, ..., *typ n* (als CHAR oder FIXED) sind. Durch RETURNS wird der Rückgabewert hinsichtlich seines Types (*typ r* als CHAR oder FIXED) festgelegt. Die Parameteranzahl darf nicht 15 übersteigen. Ebenso dürfen Makroprozeduren nicht verschachtelt definiert werden. Hinsichtlich der Makroerweiterung ist nur eine statische möglich. Als Makroanweisungen können die bisher genannten (mit Ausnahme von ACTIVATE, DEACTIVATE und INCLUDE) innerhalb des Makrokörpers verwendet werden. Dabei entfällt die Kennzeichnung durch '%'. Ein einfaches Beispiel einer Makroprozedur CONC lautet vollständig in MACRO-PL1

```
%DCL CONC ENTRY(CHAR,CHAR)RETURNS(CHAR);
%CONC:PROCEDURE(A,B)RETURNS(CHAR);
        DCL(A,B,C)CHAR;
        C=A||B;
        RETURN(C);
%END CONC;
```

Bei der Makrodefinition wird nach dem Namen CONC ein Doppelpunkt (als Markierung) und die Kennzeichnung PROCEDURE (in der Kurzform auch als PROC möglich) geschrieben. Danach folgt die Liste der formalen Parameter (hier A und B). Sie müssen im Prozedurkörper vereinbart werden und in ihrem Typ mit der Angabe des ENTRY-Attributes übereinstimmen. Soll ein ENTRY-Attribut für Parameterlisten verschiedenen Typs gelten, so ist für den Typ das Zeichen '*' zu schreiben, also zum Beispiel '...ENTRY(*,CHAR)...'. Damit wird nur festgelegt, daß der zweite Parameter vom Typ CHAR ist. Die Anweisung RETURN im Makrokörper kennzeichnet den Rückgabewert der Makroprozedur. Sie kann auch mehrmals auftreten, bedeutet aber stets das Verlassen der Makroprozedur.
Innerhalb der Makroprozedur können auch Variablen definiert werden, die nur innerhalb dieser Prozedur gelten (hier die Zeichenkettenvariable C). Andererseits kann sich in der Makroprozedur auch auf Variablen bezogen werden, die außerhalb jener definiert wurden. Die Funktion der Makroprozedur CONC besteht in der Verkettung der beiden Parameterwerte. Der Makroaufruf CONC(BEI,SPIEL) liefert die Zeichenkette 'BEISPIEL' als Rückgabewert, falls BEI und SPIEL nicht selbst Namen von Variablen des Makroprogramms sind. Zu beachten ist dabei, daß beispielsweise der Makroaufruf CONC('BEI','SPIEL') den Zeichenkettenwert '"BEI""SPIEL"' ergibt (der Apostroph im Innern der Zeichenkette wird wiederum doppelt angegeben). Ebenso werden auch Kommentare als Parameterwert mit in den Makrokörper übernommen. Eine Makroprozedur kann auch mehrfach benannt werden. Notiert man die ersten beiden Zeilen der Makroprozedur CONC in der FORM

```
%DCL(CONC,VK) ENTRY(CHAR,CHAR)RETURNS(CHAR);
%CONC:VK:PROC(A,B)RETURNS(CHAR); ,
```

so kann auch der Makroaufruf VK(BEI,SPIEL) neben den bereits oben angegebenen für diese Makroprozedur verwendet werden. Makroprozeduraufrufe können wie Makrovariablen im Makroprogramm verwendet werden. So ist beispielsweise

```
%IF CONC(A,B)='TAB003' %THEN %C=CONC(D,E);
```

mit den Zeichenkettenvariablen A, B, C, D und E eine gültige Anwendung von CONC.
Das folgende Beispiel einer weiteren Makroprozedur zeigt auch eine Makroabarbeitung.

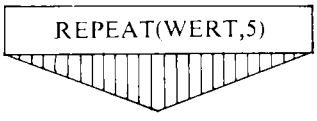
```
%DCL REPEAT ENTRY(CHAR,FIXED)RETURNS(CHAR);
%REPEAT:PROC(A,I)RETURNS(CHAR);
        DCL(A,B)CHAR,(I,J)FIXED;
        B=A;
        DO J=1 TO I;
        B=B||A;
        END;
        RETURN(B);
%END REPEAT;
```

WERTWERTWERTWERTWERTWERT

Das Makro REPEAT realisiert also die *n*-fache Verkettung des ersten Parameterwertes. Die Anzahl *n* ist dabei im zweiten Parameter anzugeben. Ist die Makrodefinition für REPEAT

im Bestand REP der Bibliothek BIBL1 gespeichert, so lautet die analog zu oben gegebene Makronutzung

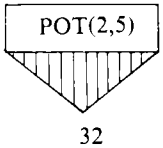
```
%INCLUDE BIBL1(REP);
```



WERTWERTWERTWERTWERTWERT

Eine weitere Makroprozedur, die zur Realisierung der Potenzfunktion innerhalb eines Makroprogrammes genutzt werden kann, hat beispielsweise die Form (einschließlich ihrer Nutzung)

```
%DCL POT ENTRY(FIXED,FIXED)RETURNS(CHAR);
%POT:PROC(I,J)RETURNS(CHAR);
      DCL(I,J,K,L)FIXED;
      L=I;
      DO K=2 TO J;
         L=L*I;
      END;
      RETURN(L);
%END POT;
```



32

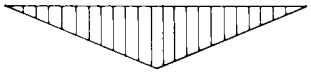
Wie bereits erwähnt, gilt auch in MACRO-PL1 die statische Makroerweiterung. Es ist somit auch möglich, das Makro REPEAT mit dem Aufruf REPEAT(X,POT(2,3)) zu nutzen.
Die folgende Makroprozedur vereinfacht das Anketten von Zahlen an Zeichenketten, ohne einen Zwischenraum, der durch die Umwandlung der maximal fünfstelligen Zahl in eine acht Zeichen lange Zeichenkette entstünde, zu lassen. Sie hat die Form

```
%DCL #Z ENTRY(FIXED)RETURNS(CHAR);
%#Z:PROC(I)RETURNS(CHAR);
      DCL(I,L)FIXED,A CHAR;
      A=I;
      DO L=7 BY -1 TO 3;
         IF SUBSTR(A,L,1)=' '
            THEN RETURN(SUBSTR(A,L+1,8-L));
      END;
      RETURN(SUBSTR(A,3,6));
%END #Z;
```

Diese Makrodefinition zeigt auch die Anwendung der Möglichkeiten von MACRO-PL1, bei einer Zyklusanweisung eine negative Schrittweite (hier −1) anzugeben und als Rück-

gabewert in der RETURN-Anweisung Ausdrücke zu verwenden. Unter der Voraussetzung, daß das Makro #Z im Bestand Z einer vorab definierten Bibliothek vorhanden ist, lautet eine Anwendung dieses Makros beispielsweise

```
%INCLUDE Z;
%DCL K FIXED; %K=217;
%DCL B CHAR; %B='WERT' ||#Z(K);
B
```



WERT217

In dieser und ähnlicher Weise ist es möglich, sich eine Anzahl von Hilfsfunktionen zu definieren, die eine universellere Nutzung von MACRO-PL1 gestattet.

Übungsaufgaben

4.1. Bestimmen Sie das Makroabarbeitungsergebnis des folgenden Makroprogrammes!

```
%DCL(A,B)CHAR,I FIXED;
%A='TEXT';
%B='';
%DO I=1 TO 5;
      %B=B||A||I;
%END;
B
```

4.2. Welchen Text ergibt die Abarbeitung der Makroanweisungsfolge

```
%DCL J FIXED;
%DO J=3 TO 10 BY 2;
Z(J)=A(J)*B(J);
%END;
```

4.3. Welchen Wert haben die Variablen R, S, T und U nach der Abarbeitung der folgenden MACRO-PL1-Anweisungen?

```
%DCL(R,S)FIXED,(T,U)CHAR;
%R=132;
%S=R/30;
%T=R;
%U=T||'33  ';
%R=U-16000;
```

4.4. Welchen Text liefert die Makroanweisungsfolge

```
%DCL(UND,ODER,WENN,SO)CHAR;
%UND='.AND.';
%ODER='.OR.';
%WENN='IF';
%SO='GOTO';
WENN (A UND B ODER C UND D) SO 13
```

4.5. Geben Sie das Interpretationsergebnis der folgenden MACRO-PL1-Anweisungen an!

```
%DCL(A,B,C)CHAR,I FIXED;
%A='DO X=1 TO N;';
%B='KLM'; %C='M1';
%DO I=1 TO 3;
      %C=C||SUBSTR(A,1,3)||SUBSTR(B,I,1)||SUBSTR(A,5);
%END;
C
S=FELD(K,L,M)+R;
END M1;
```

4.6. Welcher Text wird durch die Abarbeitung der Zeilen

```
P:PROC(Q,N);
  %INCLUDE QDEF;
  GET EDIT %INCLUDE L;(F(5));
  PUT LIST(MIN %INCLUDE L;);
  END P;
```

erzeugt, wenn der Bestand L den Inhalt

```
(X1,X2,X3,X4,X5,X6)
```

und der Bestand QDEF den Inhalt

```
DCL 1 Q(N),
      2 %INCLUDE L; FIXED,
      2 (X,Y,Z)CHAR;
```

besitzen?

4.7. Gesucht ist ein Makroprogramm, daß die Anweisungsfolge

```
A=SIN(X( 1 )**2)/SQRT(X( 1 )-1);
                .  .  .
A=SIN(X( 7 )**2)/SQRT(X( 7 )-1);
```

erzeugt.

4.8. Welche Parameterwerte werden dem Makro M durch den Aufruf

```
M('AB',C , 1,Y/*WERT*/,(,,C,D(4,1)))
```

übergeben?

4.9. Welches Ergebnis liefert die Makroprozedur

```
%DCLTTT ENTRY(CHAR)RETURNS(CHAR);
%TTT:PROC(A)RETURNS(CHAR);
      DCL A CHAR;
      RETURN(SUBSTR(A,5)||SUBSTR(A,1,4));
%ENDTTT;
```

mit dem Makroaufruf TTT(WORTSPIEL) ?

4.10. Welche Zeichenkette wird durch den Makroaufruf

```
REPEAT(ZAHL,POT(3,2))
```

generiert, wenn die Makros REPEAT und POT den im vorherigen Abschnitt definierten Inhalt haben?

4.11. Interpretieren Sie die Funktionsweise der folgenden Makroprozedur!

```
%DCL L ENTRY(CHAR)RETURNS(FIXED);
%L:PROC(A)RETURNS(FIXED);
    DCL(A,B,C)CHAR,I FIXED;
    B='*'; A=A||'*';
    DO I=1 TO 9999;
        C=SUBSTR(A,I);
        IF C=B THEN RETURN(I-1);
    END;
    RETURN(0);
%END L;
```

4.12. Schreiben Sie eine Makroprozedur TEST(A,B), die den Text

```
IF  A.NAME = B.NAME &
    A.NR   = B.NR   &
    A.BETRAG = B.BETRAG THEN
```

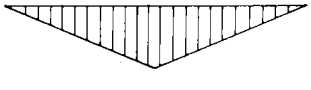
erzeugt!

4.2. Erweiterte PL/1-Makrosprache MACRO-PL1E

Mit der Verbesserung des PL/1-Compilers zum Optimierungs- und Testcompiler wurde auch eine modifizierte und erweiterte PL/1-Makrosprache, hier MACRO-PL1E genannt, implementiert. Die Erweiterungen dieser Makrosprache gegenüber MACRO-PL1 dienen vor allem einer effektiveren Textgenerierung hinsichtlich größerer Variabilität und schnellerer Makroabarbeitungszeiten. Im folgenden sollen diese Erweiterungen im einzelnen dargestellt werden.

In MACRO-PL1E werden bei der Textersetzung keine Leerzeichen rechts und links eingefügt. Damit hat das im vorherigen Abschnitt gegebene Makroabarbeitungsbeispiel jetzt die Form

```
%DCL B CHAR;
%B='01';
ZAHL-B UND WERT-B
```

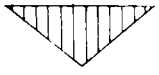


```
ZAHL-01 UND WERT-01
```

Weiterhin besteht in MACRO-PL1E die Möglichkeit, eine Variable beim Aktivieren mit dem Zusatz 'RESCAN' beziehungsweise 'NORESCAN' zu versehen. RESCAN bewirkt, daß bei der Wertersetzung der jeweiligen Variablen diese Ersetzung soweit wie möglich vorgenommen wird (also wie bisher). Hingegen wird bei NORESCAN nur ein einmaliges Ersetzen mit dem gerade vorhandenen Wert realisiert. Damit ist die im vorherigen Abschnitt genannte Fehlerursache für das sogenannte »unendliche« Ersetzen hier im wesentlichen

vermeidbar. Das dazu bereits gegebene Beispiel kann in MACRO-PL1E folgendermaßen formuliert werden

```
%DCL(A,B)CHAR; %ACT A NORESCAN;
%A='B+C';
%B=A;
A
```



```
B+C
```

Für eine effektivere Textgenerierung sind in MACRO-PL1E neben der SUBSTR-Funktion die folgenden weiteren Funktionen anwendbar:

LENGTH(*A*)

zur Bestimmung der Länge der Zeichenkette in *A*, wobei *A* auch ein Zeichenkettenausdruck sein kann;

INDEX(*A*,*B*)

zur Bestimmung der Position, ab der die Zeichenkette in *B* in der Zeichenkette in *A* vorkommt (der Funktionswert ist Null, wenn die Zeichenkette in *B* nicht in *A* enthalten ist); dabei können auch hier *A* und *B* Zeichenkettenausdrücke sein;

COUNTER

zur Generierung einer fünfstelligen Zahl (mit '00001' beginnend) als Zeichenkette, die sich bei jedem Aufruf von COUNTER um eins erhöht; beim 100000. Aufruf wird der Funktionswert '00000' erzeugt und eine Fehlermeldung ausgegeben;

COMPILETIME

generiert eine 18 Zeichen lange Zeichenkette, die das aktuelle Datum und die Uhrzeit zu Beginn der Makroabarbeitung enthält, und zwar in der Form *tt.nn.jj*␣␣*hh.mm.ss* mit *tt* für den Tag, *nn* für den Monat, *jj* für das Jahr, *hh* für die Stunden-, *mm* für die Minuten- und *ss* für die Sekundenangabe;

PARMSET(*P*)

zur Kennzeichnung, ob für den Parameter *P* einer Makroprozedur ein aktueller Wert vorhanden ist; der Funktionswert entspricht dem eines logischen Ausdruckes, also 'wahr' oder 'falsch';

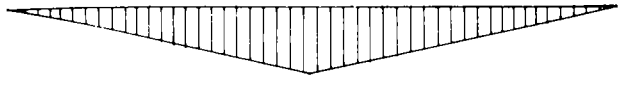
Bei der Nutzung der Hilfsfunktionen in MACRO-PL1E, also auch von SUBSTR, müssen diese zuvor aktiviert werden. Das ist beispielsweise für die LENGTH-Funktion in den beiden Formen

```
%ACT LENGTH;                    oder
%DCL LENGTH BUILTIN;
```

möglich.

Die erste Form darf in Makroprozeduren nicht verwendet werden. Die Anwendung einiger oben angegebener Funktionen hinsichtlich ihrer Aktivierung in der jeweiligen Makroanweisung zeigt das folgende Beispiel.

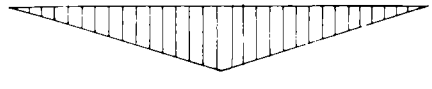
```
COUNTER=LENGTH(A)-INDEX(A,B);
%DCL A CHAR, K FIXED;
%A='X'||COUNTER;
%K=LENGTH(A)-INDEX(A,'0');
%DCL(COUNTER,LENGTH,INDEX)BUILTIN;
PUT LIST(K,COUNTER,LENGTH(A),INDEX(A,'2'),A);
```



```
COUNTER=LENGTH(A)-INDEX(A,B);
PUT LIST(      4,00002,        6,        0,X00001);
```

Das Beispiel zeigt die Notwendigkeit der Aktivierung der Funktionen für ihren expliziten Aufruf. Innerhalb von Makroanweisungen sind sie stets aktiv, also anwendbar.
So kann im folgenden Beispiel bei der Anwendung der COUNTER-Funktion dessen Aktivierung entfallen.

```
%DCL A CHAR,I FIXED;
%A='DCL(';
%DO I=1 TO 5;
      %A=A||'X'||COUNTER||',';
%END;
%A=A||'Y)FIXED;';
A
```



```
DCL(X00001,X00002,X00003,X00004,X00005,Y)FIXED;
```

Dieses Beispiel zeigt die Anwendung der COUNTER-Funktion zur eindeutigen Bezeichnungsgenerierung in der Programmiersprache PL/1. Bezüglich der Definition und Anwendung von Makroprozeduren gelten in MACRO-PL1E die folgenden Besonderheiten bzw. Erweiterungen

- die ENTRY-Deklaration kann entfallen; das heißt, daß die Parameterwerte beim Makroaufruf – falls erforderlich – automatisch umgewandelt werden (von FIXED in CHAR und umgekehrt);
- bei der Nutzung einer Makrodefinition in einem Makroprogramm ist diese stets zuvor mit der ACTIVATE-Anweisung zu aktivieren;
- für die erste Zeile einer Makrodefinition besteht die Möglichkeit der Angabe des sogenannten STATEMENT-Zusatzes; damit kann eine variablere Aufrufform für diese Makroprozedur benutzt werden; lautet beispielsweise die erste Zeile einer Makroprozedur P

```
%P:PROC(A,B,C,D)STATEMENT RETURNS(CHAR);
```

und sollen beim Aufruf die Parameterwertzuweisungen A=1, B=2, C=3 und D=4 erfolgen, so können die Aufrufformen

```
P(1,2,3,4);
P(1,2) C(3) D(4);
P(1,,,4) C(3) B(2);
P(1,,3,4,5,6,7) B(2);   oder
P C(3) B(2) D(4) A(1);
```

verwendet werden.
Eine konkrete Anwendung der STATEMENT-Form und die Berücksichtigung der weiteren oben genannten Charakteristika für Makroprozeduren in MACRO-PL1E zeigt das folgende Makroabarbeitungsbeispiel.

```
%QUOTIENT:PROC(ZAEHLER,NENNER,FAKTOR) STATEMENT
          RETURNS(CHAR);
          DCL(ZAEHLER,NENNER,FAKTOR)CHAR;
          RETURN('('||FAKTOR||') * ('||ZAEHLER
                 ||')/('||NENNER||')');
%END QUOTIENT;
%ACT QUOTIENT;
```

```
QUOTIENT ZAEHLER(X - 3) FAKTOR(LN(X)) NENNER(X * *2);
```

```
(LN(X)) * (X - 3)/(X * *2)
```

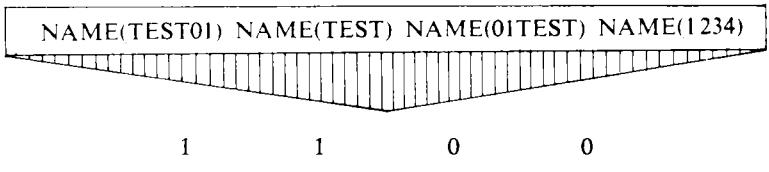
Für Zwecke der Werteprüfung wird insbesondere die INDEX-Funktion verwendet. Die folgenden Makroprozeduren prüfen beispielsweise, ob ein Zeichen ein sogenanntes Alphazeichen (A,B,C,...,Z, @,¢,#) – Makroprozedur ALFA – oder eine Ziffer – Makroprozedur ZAHL – darstellt.

```
%ALFA:PROC(X)RETURNS(FIXED);
      DCL(A,X)CHAR,INDEX BUILTIN;
      A='ABCDEFGHIJKLMNOPQRSTUVWXYZ @¢#';
      IF INDEX(A,X) ¬=0 THEN RETURN(1);
      RETURN(0);
%END ALFA;
%ZAHL:PROC(A)RETURNS(FIXED);
      DCL(A,Z)CHAR,INDEX BUILTIN;
      Z='0123456789';
      IF INDEX(Z,A) ¬=0 THEN RETURN(1);
      RETURN(0);
%END ZAHL;
```

Handelt es sich um ein Zeichen der geforderten Art, so wird jeweils die Ziffer 1 zurückgegeben, sonst eine Null. So ergeben beispielsweise die Aufrufe ALFA(X), ALFA(5), ZAHL(5) und ZAHL(X) die Werte 1, 0, 1 und 0.

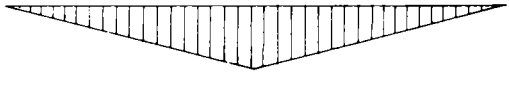
Diese beiden Makrodefinitionen können als Hilfsmakro für die Makrodefinition NAME Anwendung finden. NAME prüft, ob eine Zeichenkette ein zulässiger Name (als Alphazeichen gefolgt von Alphazeichen oder Ziffern) darstellt. Die Makrodefinition mit der entsprechenden statischen Makroerweiterung und ihrer Anwendung lautet

```
%INCLUDE ALFA,ZAHL; %ACT ALFA,ZAHL;
%NAME:PROC(A)RETURNS(FIXED);
        DCL(A,B)CHAR,I FIXED,(LENGTH,SUBSTR) BUILTIN;
        B=SUBSTR(A,1,1);
        IF ALFA(B)=0 THEN RETURN(0);
        DO I=2 TO LENGTH(A);
            B=SUBSTR(A,I,1);
            IF ALFA(B)=0 & ZAHL(B)=0 THEN RETURN(0);
        END;
%END NAME;
%ACT NAME;
```



Hierbei wurde vorausgesetzt, daß die Makrodefinition ALFA und ZAHL in einer implizit bereitgestellten Bibliothek mit den Bestandsnamen ALFA und ZAHL vorhanden sind.
Das folgende Beispiel zeigt eine einfache Anwendung der PARMSET-Funktion.

```
%COS:PROC(A)RETURNS(CHAR);
      DCL A CHAR;
      IF ¬PARMSET(A) THEN A='Y';
      RETURN('1-2*SIN('||A||'/2)**2');
%END COS;
FUER COS(X) GILT:
      COS(X)=%ACT COS; COS(X).
```

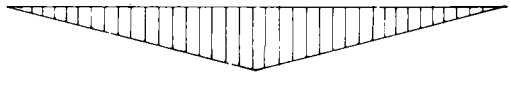


```
FUER COS(X) GILT:
      COS(X)=  1-2*SIN(X/2)**2.
```

Das Zeichen '¬' negiert die PARMSET-Auswertung, das heißt, wenn der Parameter A nicht belegt ist, wird das Zeichen 'Y' als Wert für diesen angenommen. So ergibt der Aufruf 'COS()' stets '1-2*SIN(Y/1)**2'. Eine derartige PARMSET-Anwendung wird auch als Standardwertbelegung bezeichnet.

Die bereits im vorherigen Abschnitt gegebene Makrodefinition #Z, die der Bereitstellung einer Zahl als Zeichenkette ohne Vorleerzeichen diente, kann hier in derselben Form verwendet werden. Es entfällt allerdings in MACRO-PL1E die ENTRY-Anweisung. Eine spezielle Anwendung dieser Funktion in einem Makroprogramm zeigt das folgende Makrogenerierungsbeispiel.

```
      X = 0;
      %DCL  @A CHAR, @I FIXED;
      %INCLUDE Z; %ACT #Z;
      % @A='I(1)';
M:    DO I(1)=1 TO N(1);
      %DO  @I=2 TO 6;
           DO I(#Z( @I))=1 TO N(#Z( @I)) ;
           % @A= @A||',I('||#Z( @I)'||')';
      %END;
              X( @A) = X( @A) * Y( @A);
      END M;
      PUT DATA(X);
```



```
      X = 0;
M:    DO I(1)=1 TO N(1);
         DO I(2)=1 TO N(2);
         DO I(3)=1 TO N(3);
         DO I(4)=1 TO N(4);
         DO I(5)=1 TO N(5);
         DO I(6)=1 TO N(6);
            X(I(1),I(2),I(3),I(4),I(5),I(6)) = X(I(1),I(2),I(3),I(4),I(5),I(6)) *
            Y(I(1),I(2),I(3),I(4),I(5),I(6));
         END M;
         PUT DATA(X);
```

Das Makro #Z habe hier den Bestandsnamen Z.
Als für verschiedene Anwendungen von MACRO-PL1E ungünstig erweist sich die Beschränkung der Anzahl der formalen Parameter in einer Makrodefinition. Diese Beschränkung kann umgangen werden, wenn man als Parameterwert eine geklammerte Liste zuläßt und diese mit Hilfe von Hilfsmakros entsprechend auswertet. Für ein Makro M hat der Aufruf beispielsweise die Form $M((P_1,P_2,P_3,\ldots,P_n))$. Es handelt sich also um *einen* Zeichenkettenwert, der in durch Kommata getrennte Werte durch Klammern begrenzt strukturiert ist. Die Hilfsmakros zur Auswertung dieser Liste haben dann die Funktionen

- Bestimmung der Anzahl der Werte in dieser Liste (als Makro #MNA),
- Bereitstellung der einzelnen Werte nacheinander (als Makro #ARG).

Konkrete Makrodefinitionen für #MNA und #ARG sind dann beispielsweise (s. a. [190])

```
%#ARG:PROC(X,I,J)RETURNS(CHAR);
        DCL(I,J,K,L)FIXED,X CHAR;SUBSTR BUILTIN;
        L=I; J=0;
        IF I>0 THEN DO;DO K=I TO 999;
                IF SUBSTR(X,K,1)=')' THEN DO; J=1;
                   RETURN(SUBSTR(X,I,K-I)); END;
                IF SUBSTR(X,K,1)=',' THEN DO; I=K+1;
                   RETURN(SUBSTR(X,L,K-L)); END;
                      END;
                   END;
              ELSE DO;DO K=1 TO 999;
                IF SUBSTR(X,K,1)=',' THEN DO; I=K+1;
                   RETURN(SUBSTR(X,2,K-2)); END;
                IF SUBSTR(X,K,1)=')' THEN DO; J=1;
                   RETURN(SUBSTR(X,2,K-2)); END;
                      END;
                   END;
        RETURN('');
%END #ARG;
```

und

```
%#MNA:PROC(P)RETURNS(FIXED);
        DCL(P,C)CHAR,(I,J,E)FIXED;
        I=0;
        DO J=1 TO 9999;
           C=#ARG(P,I,E);
           IF E=1 THEN IF I=0 THEN RETURN(0);
                                ELSE RETURN(J);
        END;
%END #MNA;
```

Durch die Wahl der jeweiligen Endwerte in den zyklischen Anweisungen sind hierbei die geklammerte Liste auf 9999 Zeichen und die einzelnen Werte (Argumente) auf jeweils 999 Zeichen begrenzt. Die Funktion #ARG setzt weiterhin voraus, daß der Parameterwert stets mit Klammern begrenzt ist und die einzelnen Werte nicht selbst Listen darstellen. Das Makro #MNA nutzt das Makro #ARG als Hilfsmakro. Während #MNA nur die Liste als Zeichenkette als einzigen Parameter besitzt und die Werteanzahl dieser Liste als Rückgabewert liefert, hat #ARG die drei Parameter X, I und J. X stellt die Zeichenkette (als Werteliste) dar, I beinhaltet die jeweilige Position in X nach Bereitstellung eines Listenwertes (I ist stets zu Beginn Null zu setzen), und J dient der Endeerkennung der Werteliste. Wurde der letzte Listenwert bereitgestellt, so hat J den Wert eins. Eine einfache Anwendung von #ARG deutet das folgende Beispiel an.

```
%P:PROC(L)RETURNS(CHAR);
   ...
   DCL A CHAR,(U,V,W)FIXED;
```

```
        U=0;
        DO V=1 TO 999;
            A=#ARG(L,U,W);
            ...
            IF W=1 THEN GOTO ME;
        END;
    ME: ...
    %ENDP;
```

Beim Aufruf P((X,WO3,VAR)) werden der Variablen A nacheinander die Werte 'X', 'WO3 und 'VAR' zugewiesen.
Weitere Anwendungen dieser Hilfsmakros sind in den folgenden Abschnitten gegeben.

Übungsaufgaben

4.13. Welchen Text liefert die Interpretation der MACRO-PL1E-Anweisungen

```
TRANS:PROC(X,Y);
        DCL X(5),Y(5);
        %DCL I FIXED;
        %DO I=1 TO 5;
        X(I) = Y(6-I);
        %END;
        END TRANS;        ?
```

4.14. Man gebe das Abarbeitungsergebnis des Makroprogramms

```
%DCL A CHAR,I FIXED;
%A='ABC';
SUBSTR(A,2,1)=
%ACT SUBSTR;
SUBSTR(A,2,1)||SUBSTR(A,3);
%DEACT SUBSTR;
%DO I=1 TO 3;
    %A='X'||SUBSTR(I,8);
A=SUBSTR(A,I,1);
%END;
```

an!

4.15. Definieren Sie eine Makroprozedur MULT(@1, @2, @3, @4, @5, @6), die den PL/1-Quelltext

```
DO I=1 TO @4;
DO J=1 TO @6;
@3(I,J)=0;
DO K=1 TO @5;
@3(I,J)= @3(I,J)+ @1(I,K)*@2(K,J);
END; END; END;
```

erzeugt. Als Standard, also bei fehlenden Parameterwerten, soll MULT(A,A,C,10,10,10) generiert werden.

4.16. Stellen Sie eine Makroprozedur VERIFY(X,Y) auf, die prüft, ob alle Zeichen der Zeichenkette X in der Zeichenkette Y enthalten sind. Das Resultat sei 1 beim Enthaltensein und 0 im anderen Falle!

4.17. Gesucht ist eine Makroprozedur FOR, die durch den Aufruf

```
FOR  I(A)  STEP(B)  UNTIL(C)  DO(D);
```

die Anweisungen

```
DO  I=A  TO  C  BY  B;
D
END;
```

generiert.

4.18. Definieren Sie eine Makroprozedur ARCCOS(X), die die Berechnung auf die in der Basissprache vorhandene Standardfunktion ASIN(X) für den arcsin x zurückführt!

4.19. Geben Sie eine Makrodefinition MIN in MACRO-PL1E an, die aus dem Aufruf

```
MIN  ARRAY(Y)  DIM(N)  VAR(A1,A2,A3,...,Am);
```

den Funktionsaufruf in der Basissprache PL/1

```
MIN(Y(1),Y(2),...,Y(N),A1,A2,A3,...,Am)
```

erzeugt!

4.3. Testung von PL/1-Makroprogrammen

Die interpretierende Arbeitsweise des PL/1-Makrointerpreters erfordert eine spezielle Vorgehensweise bei der Testung von PL/1-Makroprogrammen.
Die besonders schwerwiegenden Programmierfehler, die zu einer fehlerhaften bzw. nichtendenden Makroabarbeitung führen, sind dabei die folgenden:

(1) für eine Zeichenkettenvariable wird ein Zeichenkettenwert erzeugt, der wiederum den Namen der Zeichenkettenvariablen enthält; also zum Beispiel

```
%A='A(3)';
```

das Resultat ist eine unendliche Makrointerpretation;

(2) bei der Verwendung von explizit anzugebenden Zeichenketten wird die insgesamt paarweise Apostrophierung nicht beachtet; so zum Beispiel bei

```
%A='BEISPIEL;
%B='FEHLER';
%A=B||A;          ;
```

die Makrointerpretation beginnt mit der Zeichenkettenwertzuweisung für die Variable A mit dem Wert 'BEISPIEL; %B=' und ergibt somit eine Reihe von nicht beabsichtigten Fehlermeldungen;

(3) in einer zyklischen Anweisungskonstruktion ist das Zyklusende nicht ordnungsgemäß programmiert; so beispielsweise in

```
      %I=2;
%M:   I=I+2;
```

```
%A(I)=A(I)+S;
%IF I=11 %THEN%GOTO ME;
%GOTO M;
%ME:...
```

die sogenannte Abbruchsbedingung 'I=11' wird hierbei nicht erreicht und ein Makroabarbeitungsende nicht gewährleistet.

Die in (3) genannte Fehlerart ist besonders schwerwiegend, wenn die zyklisch abgearbeiteten Anweisungen selbst einen Fehler enthalten, der durch den PL/1-Makrointerpreter ausgewiesen wird. Dann wird auch die Liste der Fehlermeldungen »unendlich«.
Für die weitestgehende Vermeidung dieser und ähnlicher Fehler bei der PL/1-Makroprogrammierung gelten erfahrungsgemäß die nachfolgend beschriebenen Maßnahmen.

(a) Als Bezeichnung für Variablen (einschließlich für die Namen von Makroprozeduren) sollten Formen gewählt werden, die sich von den i.allg. verwendeten Symbolen und Bezeichnungen der Basissprache unterscheiden, wie zum Beispiel

○A01,○A02, ... oder @M1, @M2, ...

(b) Bei der Programmierung in MACRO-PL1E ist für die Variablen, soweit es das Generierungsziel zuläßt, die 'NORSCAN'-Kennzeichnung zu verwenden.

(c) Nach bestimmten Programmabschnitten, zum Beispiel nach jeder Makrodefinition innerhalb eines Makroprogramms, ist die Anweisung (nach [190])

```
%/*'*/;
```

zu setzen. Diese mit einem Kommentar versehene Leeranweisung bewirkt nach einem vergessenen Apostroph in den vorherigen Anweisungen die ordnungsgemäße Interpretation der folgenden Makroanweisungen.

(d) Für die Makrointerpretation in einem bestimmten Betriebssystem des jeweiligen Computers sollten Maßnahmen getroffen werden, die die Ausgabedruckliste des PL/1-Makrointerpreters beschränken (wie zum Beispiel durch den sogenannten OUTLIM-Parameter des Betriebssystems OS/ES).

(e) Innerhalb eines Makroprogramms sollten Testpunkte (als Testanweisungen) gesetzt werden, wie beispielsweise

```
. . .
%DCL TT CHAR;%TT='TP01'; TT
. . .
```

in Makroanweisungen oder

```
. . .
RETURN('TP01');
. . .
```

in einer Makrodefinition.

Die Fehlerursachen und Maßnahmen bei der PL/1-Makroprogrammierung zeigen, daß der Kontrolle des Makroprogramms als Quellprogramm besondere Bedeutung zukommt. Daher sind hierbei Prinzipien der Programmverifikation beziehungsweise der symbolischen Testung in seiner einfachsten Form als Testtabelle besonders geeignet.

Übungsaufgaben

4.20. Ergänzen Sie das in Aufgabe 4.1. gegebene Makroprogramm durch geeignete Testanweisungen!
4.21. Geben Sie für das in 4.1. definierte Makro REPEAT-Testanweisungen an!
4.22. Stellen Sie für das in Aufgabe 4.1. gegebene Makroprogramm eine Testtabelle auf!

4.4. Möglichkeiten der Generierung von PL/1-Makros

Die Generierung eines Quelltextes in einer bestimmten Basissprache wirft natürlich die Frage auf, ob diese Basissprache nicht selbst auch eine Makrosprache sein kann.
Für den PL/1-Makrointerpreter bedeutet das, zum Beispiel einen Text '%A=B||C;' zu generieren. Die Anweisungen lauteten hierfür

```
%DCL Z CHAR; %Z='%A=A||C;';
```

oder auch innerhalb einer Makroprozedur

```
RETURN('%A=B||C;');.
```

Das ist jedoch in MACRO-PL1 bzw. MACRO-PL1E nicht möglich. Die angegebenen Anweisungen führen zu Fehlermeldungen, da das Zeichen '%' nicht in einer Zeichenkette enthalten sein darf.
Die Generierung von PL/1-Makros kann also nicht unmittelbar durch die PL/1-Makrosprache selbst realisiert werden. Im folgenden seien drei Möglichkeiten, ein PL/1-Makroprogramm bzw. PL/1-Makros zu generieren, genannt, und zwar

(A) die Anwendung eines Generierungsprogramms in einer anderen Programmiersprache;
(B) die Transformation eines entsprechenden PL/1-Quelltextes;
(C) die Anwendung eines Generierungsprogramms auf der Basis einer vorherigen rechnergestützten Textanalyse.

Auf die Möglichkeit in (A) soll hier nicht weiter eingegangen werden. Die konkrete Realisierung kann sich der Leser selbst an einer ihm bekannten Programmiersprache, die allerdings Operationen zur Zeichenkettenverarbeitung enthalten muß, überlegen.
Die Möglichkeit (B) soll an einem einfachen Beispiel erläutert werden. Dazu soll der folgende PL/1-Programmausschnitt dienen.

```
DCL A CHAR(90)VAR,I FIXED;
A='';
DO I=1 TO 10;
A=A||I;
END;
```

Eine einfache Umformung dieses Quelltextes durch
(a) Weglassen der Zeichenkettendefinitionsangaben nach 'CHAR',
(b) Setzen des Zeichens '%' vor jede Anweisung
liefert schließlich

```
% DCL A CHAR      ,I FIXED;
% A='';
% DO I=1 TO 10;
% A=A||I;
% END;
```

und das ist ein PL/1-Makroprogramm. Natürlich liegen die PL/1-Programmausschnitte, die zu einem Makroprogramm gleicher Funktionsweise umgeformt werden sollen, nicht immer so »nahe« der PL/1-Makrosprache vor. Dennoch ist hier zumindest das Prinzip angedeutet, von PL/1-Programmen, deren Algorithmus auch innerhalb eines Makroprogramms verwendet werden soll, zu Makroprogrammen zu gelangen.
Für die Möglichkeit (C) sind in [176] Hilfsmittel beschrieben, die der Generierung einer Makroprozedur in MACRO-PL1 dienen. Diese Makroprozedur ist in ihrem Aufbau einfach gestaltet und im Sinne der Arbeitsweise des PL/1-Makrointerpreters laufzeiteffektiv.
Sie hat die allgemeine Form

```
%makroname:PROC(liste formaler parameter)RETURNS(CHAR);
                /*kommentar mit erstellungsdatumsangaben */
                DCL(variablenliste)CHAR;
H1= ... ;   }
 :          }   Zuweisung der Anfangswerte
Hn= ... ;   }   für die Hilfsvariablen
IF PAR1=wert1 THEN RETURN(H1|| ...);
 :
IF PAR k=wert k THEN RETURN(H1|| ...);
%END makroname;
```

Um ein derartiges Makro zu generieren, wird zunächst eine Ausgangsquelltextmenge analysiert. Dieser Vorgang wird ebenso wie die darauf folgende Makroerzeugung durch PL/1-Programme realisiert. Im folgenden soll zunächst diese Textanalyse näher betrachtet werden.
Dabei werden zwei oder mehrere Ausgangsquelltexte miteinander verglichen, und zwar durch Überdeckung. Ein einfaches Beispiel soll diese Textanalyse demonstrieren. Dazu seien die beiden PL/1-Programmstücke

```
MAX=A(1);
DO  I=2 TO N;
    IF MAX<A(I) THEN MAX=A(I);
END;
PUT DATA(MAX);
```

und

```
MIN=A(1);
DO  I=2 TO N;
    IF MIN>A(I) THEN MIN=A(I);
END;
PUT DATA(MIN);
```

gegeben.
Die genannte Textanalyse liefert dann das folgende Ergebnis:

```
K=      5          N=      2;
G( 1)= @ @ @       M @ @ @
S( 1, 1)= @ @ @ @ @AX @ @ @ @ @
S( 2, 1)= @ @ @ @ @IN @ @ @ @ @
G( 2)= @ @ @=A(1);                          DO I=2 TO N;
                                        IF M @ @ @
```

```
S( 1, 2)= @ @ @ @ @AX< @ @ @ @ @
S( 2, 2)= @ @ @ @ @IN> @ @ @ @ @
G( 3)= @ @ @A(I) THEN M @ @ @
S( 1, 3)= @ @ @ @ @AX @ @ @ @ @
S( 2, 3)= @ @ @ @ @IN @ @ @ @ @
G( 4)= @ @ @=A(I);                                    END;
                    PUT DATA(M @ @ @
S( 1, 4)= @ @ @ @ @AX @ @ @ @ @
S( 2, 4)= @ @ @ @ @IN @ @ @ @ @
G( 5)= @ @ @SUBSTR(G( 2),  5, 51) @ @ @
SE( 1)= @ @ @ @ @LEER @ @ @ @ @
SE( 2)= @ @ @ @ @LEER @ @ @ @ @
```

Dabei bedeuten die $G(i)$ die sogenannten gemeinsamen Teile beider Texte, $S(i,j)$ die separaten Teile (also die bei der Überdeckung unterschiedlichen Zeichenketten), $SE(j)$ die Textreste (falls die Texte unterschiedlich lang sind), N die Anzahl der verglichenen Texte und K die Anzahl der gemeinsamen Textstücke bei der Textanalyse. Die Zeichenfolgen '@ @ @' und '@ @ @ @ @' dienen nur der Begrenzung der jeweils dargestellten Textteile.
Das obige Resultat zeigt auch noch eine weitere Leistung des Textanalyseprogramms. Es handelt sich um die Generierung der Teilkettenbeziehung mit den entsprechenden Positions- und Längenangaben für die SUBSTR-Funktion, falls eine solche Beziehung zwischen den gemeinsamen Textteilen besteht.
Dieses Analyseergebnis geht nun unmittelbar in das folgende Makrogenerierungsprogramm ein. Außerdem können der Makroname, ein Kommentar und die separaten Textteile angegeben werden, die im zu generierenden Makro die Parameter darstellen sollen. Im ersten Fall seien dabei vorgegeben

MAKRO10	als Makroname,
BEISPIEL 21	als Kommentar und
'0102'	als Kennzeichnung dafür, daß der erste und zweite separate Textteil im erhaltenen Makro als Parameter realisiert sind.

Das generierte Makro lautet dann (einschließlich der ENTRY-Anweisung)

```
        %DCL MAKRO10 ENTRY(CHAR,CHAR)RETURNS(CHAR);
%MAKRO10:PROC(P1,P2)RETURNS(CHAR);
/*ERSTELLUNGSDATUM: 30.09.87
BEISPIEL 21 */
        DCL(P1,P2,H1,H2,H4,H7)CHAR;
        H1='      M'||P1;
        H2='=A(1);
DO I=2 TO N;                                        IF M'
||P2||'A(I) THEN M';
        H4='=A(I);                                    END;
                                          PUT DATA(M';
```

```
H7=SUBSTR(H2,  5, 51);
IF P1='AX' THEN RETURN(H1||H2||'AX'||H4||'AX'||H7||');
IF P1='IN' THEN RETURN(H1||H2||'IN'||H4||'IN'||H7||');
%END MAKRO10;
```

Dabei wird in Abhängigkeit vom ersten Parameterwert entweder der erste oder der zweite Ausgangstext durch das Makro erzeugt.
Die jeweiligen Makroaufrufe lauten dazu konkret

```
MAKRO10(AX,AX<)     beziehungsweise
MAKRO10(IN,IN>)  .
```

Durch diese Parametrisierung für die MAKRO10-Generierung können aber auch weitere von den Ausgangstexten unterschiedliche Texte generiert werden. So ergibt der Aufruf

```
MAKRO10(IN,IN>=)
```

den Quelltext

```
MIN=A(1);
DO I=2 TO N;
   IF MIN>=A(I) THEN MIN=A(I);
END;
PUT DATA(MIN);
```

Die Einfachheit dieses Beispiels einer Makrogenerierung gibt natürlich wenig Spielraum für die Verschiedenartigkeit der generierbaren Quelltexte aus diesem Makro.
Durch die Vorgaben 'MAKRO11' für den Makronamen, 'BEISPIEL 22' als Kommentar und keine Vorgabe für die Parametrisierung des zu generierenden Makros liefert das oben genannte Programm

```
%DCL MAKRO11 ENTRY(CHAR)RETURNS(CHAR);
%MAKRO11:PROC(P)RETURNS(CHAR);
/*ERSTELLUNGSDATUM: 30.09.87
BEISPIEL 22*/
        DCL(P,H1,H4,H6,H8,H11)CHAR;
        H1='      M';
        H4='=A(1)
DO I=2 TO N;                                    IF M';
        H6='A(I) THEN M';
        H8='=A(I);                                  END;
                                                PUT DATA(M';
        H11=SUBSTR(H4,  5, 51);
        IF P='VAR1' THEN RETURN(H1||'AX'||H4||'AX<'||H6||
                        'AX'||H8||'AX'||H11||');
        IF P='VAR2' THEN RETURN(H1||'IN'||H4||'IN>'||H6||
                        'IN'||H8||'IN'||H11||');
        %END MAKRO11;
```

Dieses Makro besitzt nur einen sogenannten Auswahlparameter, der mit dem Wert 'VAR1' den ersten und 'VAR2' den zweiten Ausgangsquelltext erzeugt. Weitere Quelltexte sind durch dieses Makro nicht generierbar.

Das in der bisher beschriebenen Weise generierte PL/1-Makro muß im konkreten Anwendungsfall natürlich noch nicht die endgültige Form sein, sondern kann gewissermaßen einen ersten Entwurf eines komplexeren oder leistungsfähigeren Makros darstellen.
Weiterhin ist es nicht erforderlich, daß die zur Textanalyse verwendeten Quelltexte nur einer Basissprache angehören.

4.5. PL/1-Makroprogrammierung für beliebige Basissprachen

Als teilweise integrierter Preprozessor dient der PL/1-Makrointerpreter – wie bereits oben genannt – konzeptionell der Erzeugung von PL/1-Quelltext. Durch die Parameterwahl für den PL/1-Compiler (dessen erste Phase ja der PL/1-Makrointerpreter darstellt) von 'M' (für Makrointerpretation), 'NSYN' (für die Unterdrückung der PL/1-Syntaxkontrolle) und 'MD' (für die Ausgabe des Makroabarbeitungsergebnisses auf eine spezielle Datei[1])) kann für den generierten Quelltext auch eine andere Basissprache als PL/1 gewählt werden.
Eine Einschränkung besteht lediglich darin, daß der PL/1-Makrointerpreter nur den sogenannten 60-Zeichen-Satz (von PL/1) verarbeitet. Es sind dies

- die Alphazeichen A,B,C,...,Z, @,¢,#,
- die Ziffern 0,1,2,...,9 und
- die Zeichen +,−,/,*,=,<,>,¬,%,?,(,),!,&,.,,,;,:,',",ß,_ .

So erhält man beispielsweise durch das Makroprogramm[2])

```
%DCL #A CHAR,(#I,#J)FIXED,SUBSTR BUILTIN;
%DO #I=30 TO 50 BY 5;
     %DO #J=100 TO 500 BY 100;
          %=A='XH1,'||SUBSTR(#I,7)||'/XH2,'||
                SUBSTR(#J,6);
CLEAR   MF1
INITIAL #A
START   50
    %END;
%END;
```

nach der Makroabarbeitung die Anweisungsfolge in der Programmiersprache SIMDIS

```
CLEAR   MF1
INITIAL XH1,30/XH2,100
START   50
CLEAR   MF1
INITIAL XH1,35/XH2,200
START   50
  ...
CLEAR   MF1
INITIAL XH1,50/XH2,500
START   50
```

[1]) Sowohl für MACRO-PL1 als auch MACRO-PL1E heißt diese Datei im Betriebssystem OS/ES 'SYSPUNCH'.
[2]) Diese und die folgenden Beispiele werden ausschließlich in MACRO-PL1E gegeben.

Wie bei SIMDIS so sind auch die Anweisungen verschiedener anderer Sprachen formatiert, das heißt, sie sind zumeist in *einer* Zeile und dort nach bestimmten Positionierungsregeln anzugeben. Bei der Generierung mehrerer derartiger Zeilen durch MACRO-PL1E muß eine Anweisung jeweils mit so vielen Leerzeichen aufgefüllt werden, daß das entsprechende Anweisungsformat gewährleistet wird. Der Unterstützung dieser Generierungsweise dient das folgende Hilfsmakro #LIN.

```
%#LIN:PROC(ZK)RETURNS(CHAR);
      DCL(ZK,ZKR,LE)CHAR,(SUBSTR,LENGTH)BUILTIN;
      LE='                                ';
      LE=LE||LE||LE||'                    ';
      ZKR=ZK||SUBSTR(LE,1,71-LENGTH(ZK));
      RETURN(ZKR);
%END #LIN;
```

Mit Hilfe dieses Makros und der in 4.2. gegebenen Makros #ARG, #MNA und #Z kann beispielsweise das zweite in 2.1. gezeigte Einführungsbeispiel für die Basissprache COBOL wie folgt dargestellt werden.

```
%COB:PROC(POS)RETURNS(CHAR);
    DCL(NR,P,POS,TEXT)CHAR,(I,J,K,L)FIXED;
    TEXT='';
    J=#MNA(POS); K=0;
    DO I=1 TO J;
       P=#ARG(POS,K,L);  NR=#Z(I);
       TEXT=TEXT||#LIN('          03 COLUMN '||P||
              ' PIC 9(4) SOURCE ELEMENT ( '||NR||' ).');
    END;
    RETURN(TEXT);
%END COB;
```

Der Makroaufruf COB((10,15,20,25,30,35,40,45,50,55)) und die Bereitstellung und Aktivierung der genannten Hilfsmakros liefert schließlich den Ausgangstext des oben gezeigten Einführungsbeispiels.

Eine weitere Einschränkung für die Wahl der Basissprache bei der Nutzung von MACRO-PL1E besteht darin, daß der generierte Text stets das Zeilenformat (2,72) – bei einer Zeilenlänge von maximal 100 Zeichen – besitzt. Wird in einer Zeile längerer Text erzeugt, so werden die darüber hinaus liegenden Zeichen in die nächste Zeile übernommen. Das ist für die gebräuchlichsten Programmiersprachen keine Einschränkung bezüglich der Textlänge in einer Zeile, wohl aber bezüglich des Beginns der relevanten Zeichen in der zweiten Position. Für solche Programmiersprachen wie ASSEMBLER muß daher der Makrogenerierung noch ein weiterer Schritt – die Transformation des erzeugten Quelltextes in die erste Position – folgen, bevor der Quelltext in den Compiler eingegeben werden kann.

Das folgende Makrobeispiel zeigt die Generierung des Algorithmus zur Berechnung des Maximums dreier Zahlengrößen in verschiedenen Basissprachen. Die entsprechende Makro-

definition habe die Form

```
%MAX:PROC(S,A,B,C)RETURNS(CHAR);
      DCL(A,B,C,K,L,M,S)CHAR,SUBSTR BUILTIN;
      S=SUBSTR(S,1,2);
      IF S='FO' THEN DO; K='.GT.'; L='';
                         M='';     N='END IF';
                      END;
                ELSE DO; K='>'; M=';';
                         L=':'; N='';
      IF S='PL' THEN L='';
      IF S='AL' THEN N='FI';
      RETURN(#LIN('IF('||A||K||B||') THEN ')||
             #LIN('IF ('||A||K||C||') THEN ')||
             #LIN('MAX'||L||'='||A||M)||
             #LIN('ELSE')||#LIN('MAX'||L||'='||C||M)||
             #LIN('ELSE IF ('||B||K||C||') THEN ')||
             #LIN('MAX'||L||'='||B||M)||
             #LIN('ELSE')||#LIN('MAX'||L||'='||C||M)||
             #LIN(N));
%END MAX;
```

Die Wahl des Makronamens MAX und die Generierung der Bezeichnung MAX in der Basissprache erfordern als Aktivierungsanweisung für die Makroabarbeitung

```
%ACT MAX NORESCAN;
```

Die Bedeutung der einzelnen Parameter von MAX lautet

S als Basissprachkennzeichnung mit den Werten ADA, ALGOL 68, FORTRAN 77, PASCAL und PL/1, wobei im Makro nur die ersten beiden Zeichen ausgewertet werden;
A, B, C für die Bezeichnung der drei Variablen, aus denen das Maximum zu bestimmen ist.

So ergibt der Aufruf MAX(ADA,X,Y,Z) das Programmstück

```
IF  (X>Y) THEN
IF  (X>Z) THEN
MAX:=X;
ELSE
MAX:=Z;
ELSE IF (Y>Z) THEN
MAX:=Y;
ELSE
MAX:=Z;
```

Der Makroaufruf MAX(PL/1,W.A(1),W.A(2),S1.B(1)) liefert schließlich das Programmstück

```
IF (W.A(1)>W.A(2)) THEN
IF (W.A(1)>S1.B(1)) THEN
MAX=W.A(1);
ELSE
MAX=S1.B(1);
ELSE IF (W.A(2)>S1.B(1)) THEN
MAX=W.A(2);
ELSE
MAX=S1.B(1);
```

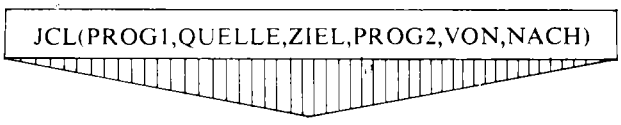
Als Basissprache soll im folgenden eine Kommandosprache gewählt werden, und zwar die Jobsteuersprache JCL (Job Control Language) des Betriebssystems OS/ES. Diese Sprache besitzt bereits selbst eine Reihe von Möglichkeiten, den Quelltext in dieser Sprache zu manipulieren, bevor er verarbeitet (hierbei interpretiert) wird. Um diese Manipulationsmöglichkeiten noch zu erweitern, kann ebenfalls MACRO-PL1E genutzt werden. Als Beispiel sei die folgende Makrodefinition gegeben:

```
%JCL:PROC(P1,D1,D2,P2,D3,D4)RETURNS(CHAR);
      DCL(P1,P2,D1,D2,D3,D4,TEXT)CHAR,PARMSET BUILTIN;
      TEXT=#LIN('// EXEC  PGM='||P1);
      TEXT=TEXT||#LIN('//EIN  DD  DSN='||D1||',DISP=SHR');
      TEXT=TEXT||#LIN('//AUS  DD  DSN=&BIB('||D2||
            '),DISP=SHR');
      IF PARMSET(P2) THEN DO;
         TEXT=TEXT||#LIN('//  EXEC  PGM='||P2);
         TEXT=TEXT||#LIN('//EIN  DD  DSN='||D3||
               ',DISP=SHR');
         TEXT=TEXT||#LIN('//AUS DD  DSN=&BIB('||D4||
               '),DISP=SHR');
                              END;
      RETURN(TEXT);
%END JCL;
```

Aufrufbeispiele für dieses Makro sind


```
//     EXEC  PGM=PROG1
//EIN    DD  DSN=EINDAT,DISP=SHR
//AUS    DD  DSN=&BIB(AUSDAT),DISP=SHR
```

beziehungsweise

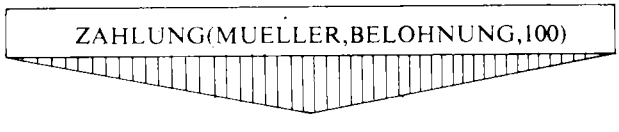



```
//      EXEC  PGM=PROG1
//EIN     DD  DSN=QUELLE,DISP=SHR
//AUS     DD  DSN=&BIB(ZIEL),DISP=SHR
//      EXEC  PGM=PROG2
//EIN     DD  DSN=VON,DISP=SHR
//AUS     DD  DSN=&BIB(NACH),DISP=SHR
```

Die weitere Nutzung dieser Generierungsergebnisse erfordert noch die oben genannte Transformation auf die erste Position in der jeweiligen Zeile.
Abschließend in diesem Abschnitt soll ein Beispiel für die Generierung verbalen Textes angegeben werden. Das Makro ZAHLUNG habe die Form

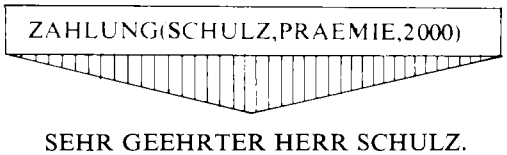
```
%ZAHLUNG:PROC(NAME,ART,ZAHL)RETURNS(CHAR);
            DCL(NAME,ART,ZAHL)CHAR;
            RETURN(#LIN('      SEHR GEEHRTER HERR'||
               NAME||'.')||#LIN('HIERMIT UEBERSENDEN'
               ||'WIR IHNEN EINE '||ART)||#LIN('IN'||
               'HOEHE VON '||ZAHL||' MARK.'));
%END ZAHLUNG;
```

Hierzu lauten zwei einfache Makroanwendungen




```
   SEHR GEEHRTER HERR MUELLER.
HIERMIT UEBERSENDEN WIR IHNEN EINE BELOHNUNG
IN HOEHE VON 100 MARK.
```

beziehungsweise




```
   SEHR GEEHRTER HERR SCHULZ.
HIERMIT UEBERSENDEN WIR IHNEN EINE PRAEMIE
IN HOEHE VON 2000 MARK.
```

Übungsaufgaben

4.23. Geben Sie das Interpretationsergebnis des oben gegebenen Makros MAX für die Makroaufrufe

```
MAX(ALGOL 68,WERT1,WERT2,WERT3)
MAX(FORTRAN 77,E,F,G)
```

an!

4.24. Gesucht ist eine PL/1-Makroprozedur ASM, die die ASSEMBLER-Anweisungen der Aufgabe 3.6 erzeugt.

4.25. Für das COBOL-Programmstück

```
COMPUTE  A1 = B1 + C1.
COMPUTE  A2 = B2 + C2.
. . .
COMPUTE  An = Bn + Cn.
```

ist eine Makroprozedur mit der Aufrufform

```
BER((A1,A2,...,An),(B1,B2,...,Bn),(C1,C2,...,Cn))
```

aufzustellen. Dabei sind die Hilfsmakros #ARG und #LIN zu verwenden!

4.26. Geben Sie eine Makroprozedur JOB mit der Aufrufform

```
JOB((P1,E1,A1,P2,E2,A2,...,Pn,En,An))
```

an, die Jobsteueranweisungen des OS/ES für die Abarbeitung der Programmfolge P1,P2,...,Pn mit den jeweiligen Ein- und Ausgabedateien E1,A1,E2,A2,...,En,An erzeugt und dabei das Makro JCL verwendet!

4.27. Schreiben Sie ein Makroprogramm in MACRO-PL1E, das eine durch Leerzeichen getrennte Buchstabenfolge in die entsprechende Zeichenfolge des Morsealphabetes überführt.

4.28. Zur Erzeugung des Textes

```
Werter Kollege  A  .
Am  B  findet um  C  Uhr  im Raum  D
unsere nächste Beratung statt. Wir erwarten
ihre Teilnahme.
                    Mit freundlichen Grüßen.
                                        E
```

formuliere man eine Makroprozedur EINLADUNG(A,B,C,D,E) !

4.6. Implementation von Fachsprachen mittels der PL/1-Makrotechnik

4.6.0. Einleitende Bemerkungen

Hinsichtlich der Fachsprachen, deren Implementationsmöglichkeiten hier gezeigt werden sollen, wird sich hierbei auf die algorithmenbeschreibenden beschränkt. Das bedeutet, daß die Anweisungen der hier berücksichtigten Fachsprachen sich stets auf Programme abbilden lassen.

4.6.1. Realisierungsmöglichkeiten durch die PL/1-Makroprogrammierung

Um Fachsprachen zu implementieren – sei es durch Übersetzung in Programme einer gebräuchlichen Programmiersprache oder durch unmittelbare Interpretation – ist es notwendig, einen Compiler beziehungsweise einen Interpreter zu schaffen, der im wesentlichen die in 1.3. gezeigten Funktionen realisiert.
In MACRO-PL1E ist dabei die notwendige Transformation syntaktischer Formen nur sehr begrenzt möglich, da die Makronamen bestimmten Restriktionen genügen müssen. Für die Gewährleistung einer ausreichenden Variabilität bei der Syntaxtransformation sind jedoch gerade nur die Makro (-Prozeduren) sinnvoll. Ihre einfachen Aufrufformen sind

MNAME bzw.

MNAME(P1,P2,...,Pn).

Die erste Form ermöglicht die Transformation einzelner Wörter, während durch die Anwendung der zweiten Aufrufform einfache Parametersprachen realisiert werden können. Die STATEMENT-Aufrufform in der allgemeinen Darstellung

MNAME PAR1(*wert1*) PAR2(*wert2*) ... PARn(*wert n*);

bei der die PARi-Angaben auch vertauscht werden können, gestattet eine höhere Variabilität bei der Sprachtransformation, ist in vielen Fällen aber auch noch nicht ausreichend. Eine nahezu beliebige (syntaktische) Transformation wird erst errreicht, wenn der gesamte zu transformierende Text den Zeichenkettenwert des Parameters *einer* Makroprozedur darstellt. Diese Realisierungsweise schafft auch günstige Voraussetzungen für die notwendige syntaktische Kontrolle der Anweisungen der Fachsprache.
Die bisher gezeigten Möglichkeiten der PL/1-Makroprogrammierung dienten ausschließlich der Codegenerierung. Für die in diesem Zusammenhang ebenfalls notwendige Syntaxkontrolle erweist sich die Einführung der in MACRO-PL1E nicht vorhandenen Datentypen einer Kellerspeichervariablen (kurz Keller genannt) und Feldern (speziell als Vektoren, Matrizen usw.) als sinnvoll. In ihrer konkreten Realisierung handelt es sich in MACRO-PL1E um Zeichenkettenvariablen, auf die die entsprechenden Lese- und Schreibfunktionen in Form von Hilfsmakros angewendet werden.
Für die Variable KELLER hat diese Lese- bzw. Schreibfunktion (hier PUT und STACK genannt) beispielsweise die Form

```
%STACK:PROC(WERT)RETURNS(CHAR);
        DCL WERT CHAR;
        KELLER=KELLER||'&'||WERT;
        RETURN('');
%END STACK;
```

beziehungsweise

```
%PUT:PROC RETURNS(CHAR);
    DCL WERT CHAR,(I,J)FIXED,(LENGTH,SUBSTR)BUILTIN;
    J=LENGTH(KELLER);
    DO I=J TO 1 BY -1;
        IF SUBSTR(KELLER,I,1)='&' THEN GOTO MA1;
    END;
```

```
        RETURN('');
MA1:    WERT=SUBSTR(KELLER,I+1,J-I);
        KELLER=SUBSTR(KELLER,1,I-1);
        RETURN(WERT);
%END PUT;
```

Die Variable KELLER ist dabei im Makroprogramm (außerhalb der beiden Makros) zu definieren. Die Schreibfunktion STACK setzt weiterhin voraus, daß die Variable KELLER zu Beginn leer ist (zum Beispiel durch die Anweisung %KELLER='';). Das Zeichen '&' dient als Trennzeichen zwischen den einzelnen Werten und darf daher selbst nicht Bestandteil des Zeichenkettenwertes sein.
Das folgende Beispiel zeigt eine einfache Anwendung dieser Hilfsmakros.

```
%INCLUDE STACK,PUT;%ACT STACK,PUT;
%DCL(FORMEL,KELLER)CHAR, I FIXED,SUBSTR BUILTIN;
%KELLER='';
%DO I=1 TO 4; STACK(+) STACK(-) %END;
%FORMEL='1/E=1';
%DO I=1 TO 8;
    %FORMEL=FORMEL||PUT||'1/'||SUBSTR(I,8)||'!';
%END;
FORMEL
```

```
1/E=1-1/1!+1/2!-1/3!+1/4!-1/5!+1/6!-1/7!+1/8!
```

Die Makros STACK und PUT seien als gleichnamige Bestände in einer implizit bereitgestellten Bibliothek vorausgesetzt. In der Anweisung '%DO I=1 TO 4;...' wird die Variable KELLER zunächst abwechselnd mit den Zeichenketten '+' und '-' gefüllt. Dann – in der Anweisung '%DO I=1 TO 8;...' – werden durch die PUT-Funktion diese Werte der Reihe nach (in umgekehrter Reihenfolge) wieder bereitgestellt.
Die Realisierung von Feldern kann in MACRO-PL1E in gleicher Weise erfolgen. Ein Feld (hier ARRAY genannt) ist wiederum eine Zeichenkettenvariable, und die erforderlichen Lese- und Schreibfunktionen stellen wiederum Makroprozeduren dar. Derartige Makros für die Lesefunktion (hier LIES genannt) und die Schreibfunktion sind beispielsweise

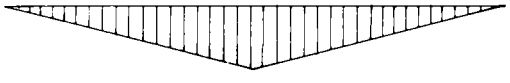
```
%LIES:PROC(I)RETURNS(CHAR);
        DCL I FIXED,SUBSTR BUILTIN;
        RETURN(SUBSTR(ARRAY,(I-5)*5+1,5));
%END LIES;
```

beziehungsweise

```
%SCHREIB:PROC(WERT,I)RETURNS(CHAR);
            DCL WERT CHAR,I FIXED,SUBSTR BUILTIN;
            ARRAY=SUBSTR(ARRAY,1,(I-1)*5)||WERT||
                    SUBSTR(ARRAY,(I-1)*5+1);
            RETURN('');
%END SCHREIB;
```

Die Variable ARRAY ist dabei außerhalb dieser Makros zu definieren. Die Elemente des Feldes ARRAY müssen – gemäß den oben gegebenen Makrodefinitionen LIES und SCHREIB – genau fünf Zeichen lange Zeichenketten sein. Da die Funktion SCHREIB nur eine Operation mit der globalen Variable ARRAY beinhaltet, ist ihr Rückgabewert die leere Zeichenkette.
Eine einfache Anwendung der beiden Hilfsmakros ist im folgenden gegeben.

```
%INCLUDE LIES,SCHREIB; %ACT LIES,SCHREIB;
%DCL ARRAY CHAR,COUNTER BUILTIN,I FIXED;
%ARRAY='';
%DO I=1 TO 9; SCHREIB(COUNTER,I) %END;
%DO I=9 TO 1 BY -1; LIES(I) %END;
```



```
00009 00008 00007 00006 00005 00004 00003 00002 00001
```

Hierbei wurde wiederum vorausgesetzt, daß die Makros LIES und SCHREIB als gleichnamige Bestände in einer implizit bereitgestellten Bibliothek vorhanden sind.

4.6.2. Ein Beispiel der Implementation einer einfachen Fachsprache

Um den Umfang dieses Beispiels zu begrenzen, sei als »Fach« die Implementation von matrizenverarbeitenden Anweisungen in der Basissprache PL/1 gewählt. Die Anweisungen der Fachsprache seien parametrisierte Funktionsnamen. Im folgenden sind die Form, der Inhalt und das Generierungsergebnis dieser Fachsprachenanweisungen dargestellt.

Fachsprachenanweisung	Generierungsergebnis	Bedeutung
MADEF(*a,n,m*)	DCL *a*(*n,m*)FIXED;	Definition einer Matrix *a* mit dem Format (*n,m*)
MALES(*a,n,m*)	GET EDIT(((*a*(I,J)DO J=1 TO *m*)DO I=1 TO *n*)) (*m*F(5));	Einlesen der Matrix *a* vom Format (*n,m*)
MADRU(*a,n,m*)	PUT EDIT(((*a*(I,J)DO J=1 TO *m*)DO I=1 TO *n*)) (*m*F(5));	Drucken einer Matrix *a* vom Format (*n,m*)
MABER(*a,n,m,b,l,c,op*)	für *op*='+' oder *op*='−': DO I=1 TO *n*, DO J=1 TO *m*; *c*(I,J)=*a*(I,J)*op* *b*(I,J); END; END;	Addition bzw. Subtraktion zweier Matrizen
	für *op*='*': DO I=1 TO *n*;DO J=1 TO *m*; *c*(I,J)=0;DO K=1 TO *l*; *c*(I,J)=*c*(I,J)+*a*(I,K)* *b*(K,J);END;END;END;	Multiplikation zweier Matrizen

Die Implementierung dieser Fachsprachenanweisungen in MACRO-PL1E lautet

```
%MADEF:PROC(M,Z,P)RETURNS(CHAR);
        DCL(M,Z,P)CHAR;
        /***   DEFINITION EINER MATRIX  M(Z,P)    **/
        RETURN('DCL '||M||'('||Z||', '||P||')FIXED;');
%END MADEF;
%MALES:PROC(M,Z,P)RETURNS(CHAR);
        DCL(M,Z,P)CHAR;
        /***    EINLESEN EINER MATRIX     ***/
        RETURN('GET EDIT((('||M||'(I,J)DO J=1 TO '||P||
                ')DO I=1 TO '||Z||')('||P||'F(5));');
%END MALES;
%MADRU:PROC(M,Z,P)RETURNS(CHAR);
        DCL(M,Z,P)CHAR;
        /***    DRUCKEN EINER MATRIX     ***/
        RETURN('PUT EDIT((('||M||'(I,J)DO J=1 TO '||P||
                ')DO I=1 TO '||Z||'))('||P||'F(5));');
%END MADRU;
%MABER:PROC(M1,Z1,P1,M2,P2,M3,OP)RETURNS(CHAR);
        DCL(M1,Z1,P1,M2,P2,M3,OP)CHAR;
        /*  MATRIXOPERATION:  M3(Z1,P2)=M1(Z1,P1)OP
        M2(P1,P2)*/
        IF OP='*' THEN RETURN('DO I=1 TO '||Z1||';DO J=1
                TO'||
                P1||';'||M3||'(I,J)=0; DO K=1 TO '||P2||
                ';'||M3||'(I,J)='||M3||'(I,J)+'||M1||
                '(I,K)*'||M2||'(K,J);END;END;END;');
        RETURN('DO I=1 TO '||Z1||'; DO J=1 TO '||P1||';'||M3||
            '(I,J)='||M1||'(I,J)'||OP||M2||'(I,J);END;END;');
%END MABER;
```

Ein vollständiges Programmstück zur Multiplikation zweier Matrizen mit den Formaten (5,10) und (10,6) in der Programmiersprache PL/1 zu implementieren, wird durch

```
MADEF(A,5,10)
MADEF(B,10,6)
MADEF(C,5,6)
MALES(A,5,10)
MALES(B,10,6)
MABER(A,5,10,B,6,C,*)
MADRU(C,5,6)
```

erreicht. Für die semantische Korrektheit (also zum Beispiel die Anweisung MABER(B,5,10, A,6,C,*) hier auszuschließen) hat der Nutzer dieser Fachsprache zu sorgen.

Hier soll im folgenden die Kontrolle der syntaktischen Korrektheit näher betrachtet werden. Die Anweisungen MADEF, MALES und MADRU haben die allgemeine syntaktische Form

anweisungsname (*buchstabe(n),ziffern,ziffern*) .

Die Klammern und Kommata ausgenommen, handelt es sich also um vier syntaktische Elemente. In diesem Sinne betrachtet, hat die Anweisung MABER acht syntaktische Elemente. Eine Makrodefinition SYNKON zur syntaktischen Kontrolle der Anweisungen des Fachsprachenbeispiels lautet dann

```
%SYNKON:PROC(ANW)RETURNS(CHAR);
        DCL(A,B,ANW)CHAR,(I,J,K)FIXED,(SUBSTR,LENGTH)
           BUILTIN;
        J=1;  K=0;  KELLER='';
        /*************************************/
        /**ZERLEGUNG DER ANWEISUNG IN DIE
            SYNTAKTISCHEN                    **/
        /** ELEMENTE NAME ( NAME, ZAHL, USW. UND */

        /**SPEICHERUNG IN DIE GLOBALE VARIABLE
            KELLER                           **/
        /*************************************/

        DO I=1 TO LENGTH(ANW);

        IF SUBSTR(ANW,I,1)='(' THEN DO;
          B=STACK(SUBSTR(ANW,1,I-1));K=K+1;J=I+1;END;
        IF SUBSTR(ANW,I,1)=',' THEN DO;
          B=STACK(SUBSTR(ANW,J,I-J));K=K+1;J=I+1;END;
        IF SUBSTR(ANW,I,1)=')' THEN DO;
           B=STACK(SUBSTR(ANW,J,I-J));K=K+1;GOTO
           KONT1;END;
     END;
     GOTO FEHLER;
KONT1:/*************************************/
     /**KONTROLLE DER ANZAHL DER SYNTAKTISCHEN
        ELEMENTE                             */
     /*************************************/
     IF K=8 | K=4 THEN; ELSE GOTO FEHLER;
     /*************************************/
     /** KONTROLLE DER SYNTAKTISCHEN ELEMENTE **/
     /*************************************/
     IF K=8 THEN DO; A=PUT;
          IF A='+' | A='-' | A='*' THEN;ELSE GOTO
             FEHLER;
          IF NAME(PUT)=1&ZIFFERN(PUT)=1&NAME(PUT)
              =1
             & ZIFFERN(PUT)=1 & ZIFFERN(PUT)=1 &
               NAME(PUT)=1 THEN; ELSE GOTO FEHLER;
          A=PUT; IF A='MABER' THEN GOTO KONT2;
             GOTO FEHLER;
                END;
```

```
        IF ZIFFERN(PUT)=1 & ZIFFERN(PUT)=1 & NAME(PUT)=1
              THEN; ELSE GOTO FEHLER;
        A=PUT;
        DO I=1 TO 3;
            IF A=LIES(I) THEN GOTO KONT2;
        END;
        GOTO FEHLER;
KONT2:  /**************************************/
        /**ANFUEGUNG DER KONTROLLIERTEN ANWEISUNG
            AN DIE                          **/
            GLOBALE VARIABLE MAPROG         **/
        /**************************************/
        MAPROG = MAPROG || ANW;
        RETURN('');
        /**************************************/
        /** FEHLERMELDUNG BEIM SYNTAXFEHLER  **/
        /**************************************/
FEHLER: RETURN('FEHLERHAFTE ANWEISUNG');
%END SYNKON;
```

Das Makro SYNKON nutzt bereits ebenfalls die in den vorherigen Abschnitten beschriebenen Makros NAME (mit den Hilfsmakros ALFA und ZAHL), LIES, PUT und STACK. Weiterhin wird ein Makro ZIFFERN, das die Kontrolle bezüglich einer Ziffernfolge realisiert und dabei das Makro ZAHL nutzt, verwendet.
Die Nutzung dieser Hilfsmakros erfordert bei der Anwendung von SYNKON, die Variablen KELLER und ARRAY zu deklarieren. Da aus dem Pseudofeld ARRAY ausschließlich gelesen wird, ist dieses zu Beginn mit den Anweisungsnamen der Fachsprache (MADEF, MALES und MADRU) zu belegen.
Für solche fehlerhaften Anweisungen der Fachsprache, wie

MAFEH(), MADEF(A,B,C) oder MALES(A,U,I,J,K),

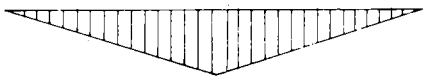
liefert SYNKON jeweils die Zeichenkette

'FEHLERHAFTE ANWEISUNG'.

Die obige Implementationsaufgabe einer Matrixmultiplikation lautet unter Einbeziehung der Syntaxkontrolle

```
%DCL(KELLER,ARRAY,MAPROG)CHAR;
%KELLER=''; %ARRAY='MADEFMALESMADRU'; %MAPROG='';
%INCLUDE SYNKON,NAME,ZIFFERN,PUT,STACK,LIES,MATRIX;
%ACT SYNKON,NAME,ZIFFERN,PUT,STACK,LIES;
SYNKON(MADEF(A,5,10))
SYNKON(MADEF(B,10,6))
SYNKON(MADEF(C,5,6))
SYNKON(MABER(A,5,10,B,6,C,*))
SYNKON(MADRU(C,5,6))
%ACT MADEF,MABER,MALES,MADRU;
```

```
%DEACT PUT;
MAPROG
```



```
DCL A(5,10)FIXED;DCL B(10,6)FIXED;DCL C(5,6)FIXED;GET EDIT
(((A(I,J)DO J=1 TO 10)DO I=1 TO 6))(10F(5));GET EDIT(((B(I,J)DO J=1
TO 6)DO I=1 TO 10))(6F(5));DO I=1 TO 5;DO J=1 TO 10;C(I,J)=0;DO
K=1 TO 6;C(I,J)=C(I,J)+A(I,K)*B(K,J);END;END;END;PUT EDIT
(((C(I,J)DO J=1 TO 6)DO I=1 TO 5))(6F(5));
```

Der Bestand MATRIX beinhalte hierbei die Makrodefinitionen für die Fachsprachenanweisungen. Die Aktivierung dieser Namen erst nach der Syntaxkontrolle ist erforderlich, da sonst diese Anweisungen bereits in SYNKON aufgelöst würden und eine eigentliche Syntaxkontrolle nicht mehr möglich wäre. Die Anweisung '%DEACT PUT;' ist notwendig, da PUT gleichzeitig im generierten Text enthalten ist.

Übungsaufgaben

4.29. Schreiben Sie für das Makro ZIFFERN eine Makrodefinition!

4.30. Geben Sie ein weiteres Anwendungsbeispiel zur Implementation eines PL/1-Programmstückes für die Addition zweier Matrizen mit dem Format (4,6) an, wobei die Anweisungen ebenfalls einer Syntaxkontrolle durch SYNKON zu unterziehen sind!

4.31. Für die Erzeugung der COBOL-Anweisungsfolge des zweiten Einführungsbeispiels in 2.1. ist ein Makroprogramm anzugeben, bei dem die Parameterangaben '(10,15,20,...,55)' in einem Feld (ARRAY) gespeichert sind und somit das Makro LIES angewendet werden kann.

4.32. Man schreibe eine Makroprozedur PRUEF(A), die zunächst die Elemente eines Ausdruckes A der syntaktischen Form

name op name

in die globale Variable KELLER speichert und dann die syntaktische Richtigkeit dieses Ausdruckes prüft. Für *op* seien die Zeichen '−' und '+' zugelassen. Man verwende dabei die Makros PUT und STACK. Der Rückgabewert von PRUEF sei 1 bei Korrektheit und sonst 0.

4.7. Weitere Anwendungsbeispiele der PL/1-Makrotechnik

In den vorangegangenen Abschnitten wurden bereits die Möglichkeiten einer nahezu beliebigen Zeichenkettengenerierung durch die PL/1-Makrotechnik einerseits und andererseits die Zusammenfassung von Makroaufrufen zu sogenannten Fachsprachen gezeigt. Die folgenden Beispiele sind spezieller Art und lassen sich nur im weiteren Sinne in diese Anwendungsfälle einordnen.

Als erstes Beispiel soll die Generierung eines Programms zur Prüfung und Manipulation von Daten betrachtet werden. Quelle für diese Generierung ist eine sogenannte Datenbeschreibungstabelle der Art

Datenname 1 / Prüfart 1 / D-Name 11/ Operation 1/D-Name 21
Datenname 2 / Prüfart 2 / D-Name 12/ Operation 2/D-Name 22
. . .
Datenname n / Prüfart n / D-Name 1n/ Operation n/D-Name 2n

Als 'Prüfart i' ($1 \leq i \leq n$) seien hierbei nur zugelassen '1' (für eine Kontrolle auf Numerischkeit) und '2' (für die Kontrolle bezüglich der Zeichen des Alphabetes A, B, C, ..., Z). Die Angaben ab 'D-Name 1i' ($1 \leq i \leq n$) sind wahlweise und bedeuten – falls sie für das jeweilige 'Datenelement i' angegeben sind –, daß sich der Wert von 'Datenelement i' aus Werten von 'D-Name 1i' und 'D-Name 2i', die durch 'Operation i' miteinander verknüpft sind, ergibt.
Ein Makro GEN zur Generierung eines Programms mit der oben beschriebenen Eigenschaft für die Basissprache PL/1 lautet dann beispielsweise

```
%GEN:PROC(DATAB,N)RETURNS(CHAR);
      DCL(DATAB,DAT,PRUEF,BER,PROG)CHAR,(I,N)FIXED;
      PROG='P:PROC OPTIONS(MAIN);DCL(□1,';
      DAT='□2';  PRUEF='';    BER='';
      DO I=1 TO N;
         IF  LIES(DATAB,I,2)='      1' THEN DO;
             PROG=PROG||','||LIES(DATAB,I,1);
             PRUEF=PRUEF ||'IF VERIFY('||LIES(DATAB,I,1)||\
                   ','"0123456789")=0 THEN GOTO FEHLER;';
                                              END;
         IF  LIES(DATAB,I,2)='      2' THEN DO;
             DAT=DAT||','||LIES(DATAB,I,1);
             PRUEF=PRUEF ||'IF VERIFY('||LIES(DATAB,I,1)||
                   ','"ABCDEFGHIJKLMNOPQRSTUVWXYZ")=
                   0 THEN '||
                   'GOTO FEHLER;';
                                              END;
         IF  LIES(DATAB,I,3)¬='' THEN
               BER=BER||LIES(DATAB,I,1)||'='||LIES(DATAB,I,3)
                   ||LIES(DATAB,I,4)||LIES(DATAB,I,5)||';';
   END;
   RETURN(PROG||')FIXED,('||DAT||')CHAR;DATUM=DATE;'||
          PRUEF||BER||'GOTO ENDE;FEHLER:PUT SKIP EDIT
          ("FEHLER")'||'(A);ENDE;END P;');
   %END GEN;
```

In diesem Makro dienen die Zeichenkettenvariablen

DAT der Generierung der Vereinbarungsliste des PL/1-Programms,

PRUEF der Generierung der Prüfalgorithmen und

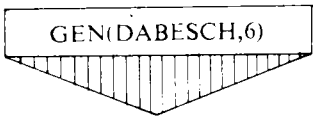
BER der Generierung eventuell auszuführender Berechnungen.

Die Zeichenketten '□1' und '□2' sollen dabei nur einen ordnungsgemäßen Listenanfang für das Generierungsergebnis gewährleisten.
Das Makro GEN verwendet das Hilfsmakro LIES für die Bereitstellung der Elemente der Pseudomatrix DATAB. Diese Pseudomatrix hat genau fünf Spalten und eine Elementlänge von acht Zeichen. Eine Zeilenbeschränkung besteht nicht.

Als Parameterwerte sind bei der Nutzung von GEN der Pseudomatrixname und die Zeilenanzahl dieser Matrix anzugeben. Für das folgende Beispiel einer GEN-Nutzung habe diese Matrix den Namen DABESCH und den Inhalt

NAME	2		
GEBDATUM	1		
GEHALT	1	GESAMT	- ABZUEGE
GESAMT	1		
ABZUEGE	1		
ALTER	1	DATUM	- GEBDATUM

Das Anwendungsbeispiel, bei dem die Bereitstellung und Aktivierung des Makros GEN und des Hilfsmakros LIES vorausgesetzt wurden, lautet dann




```
P:PROC OPTIONS(MAIN);DCL(○1,GEBDATUM, GEHALT , GESAMT,
ABZUEGE, ALTER )FIXED,(○2,      NAME)CHAR;DATUM=DATE;
IF VERIFY    NAME, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')=0
THEN GOTO FEHLER;IF VERIFY(GEBDATUM,'0123456789')=0 THEN
GOTO FEHLER;IF VERIFY( GEHALT, '0123456789')=0 THEN GOTO
FEHLER;IF VERIFY(  GESAMT,'0123456789')=0 THEN GOTO FEH
LER;IF VERIFY( ABZUEGE,'0123456789')=0 THEN GOTO FEHLER;
IF VERIFY(  ALTER ,'0123456789')=0 THEN GOTO FEHLER; GEHALT
= GESAMT      -ABZUEGE; ALTER = DATUM     -GEBDATUM;
GOTO ENDE;FEHLER :PUT SKIP EDIT('FEHLER')(A);ENDE :END P;
```

Die Pseudomatrix DABESCH ist zuvor als Zeichenkette zu vereinbaren und mit dem oben angegebenen Wert zu belegen.
Dem Leser sei es überlassen, die Problemstellung zu erweitern und leistungsfähigere Basissprachprogramme zu generieren.
Das zweite Beispiel zur Anwendung der PL/1-Makroprogrammierung dient dem verdichteten Speichern von Text schlechthin und besitzt vor allem für die Archivierung von Daten eine besondere Bedeutung. Die hier beschriebene Art und Weise der Textverdichtung geht auf eine Idee von STORER [124] zurück.
Es sei dazu ein Text in Form einer Zeichenkette der Art

$$z_1 z_2 z_3 \cdots z_n$$

betrachtet. Sind in dieser Zeichenkette beispielsweise die Teilzeichenketten $z_1 z_2 \cdots z_{10}$ und $z_{13} z_{14} \cdots z_{22}$ identisch, so kann abkürzend geschrieben werden

$$z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10} z_{11} z_{12}\ @(1,10) z_{23} z_{24} \cdots z_n .$$

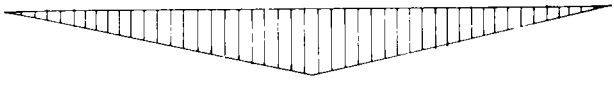
@(1,10) bedeutet, daß an dieser Stelle zehn Zeichen ab Position eins dieser Zeichenkette einzusetzen sind. ' @(1,10)' hat dabei selbst sieben Zeichen und ergibt – mit dem notwendigen

Leerzeichen vor dem Funktionnamen für die Erkennung der @-Funktion bei der PL/1-Makroprogrammierung – eine Einsparung um zwei Zeichen. Eine Makrodefinition in MACRO-PL1E zur Erzeugung des verdichteten Textes lautet dann beispielsweise

```
%DICHT:PROC(TEXT,N)RETURNS(CHAR);
        DCL TEXT CHAR,(I,J,N)FIXED,
        (LENGTH,SUBSTR)BUILTIN;
        I=0;
   MA1:I=I+1;
        IF I>=LENGTH(TEXT)-N THEN GOTO MA3;
        J=I;
   MA2:J=J+1;
        IF J>=LENGTH(TEXT)-N+1 THEN GOTO MA1;
        IF SUBSTR(TEXT,I,N)=SUBSTR(TEXT,J,N) THEN
          TEXT=SUBSTR(TEXT,1,J-1)||' @('||#Z(I)||
                ','||#Z(N)||')'||SUBSTR(TEXT,J+N);
        GOTO MA2;
   MA3:RETURN(TEXT);
%END DICHT;
```

Der Parameter N ermöglicht es, die Länge der Teilzeichenkette, die auf Gleichheit innerhalb des Textes geprüft werden soll, selbst zu bestimmen. Das Makro DICHT nutzt als Hilfsmakro die in den vorherigen Abschnitten definierte Funktion #Z.
Eine Anwendung von DICHT ist im folgenden Makroabarbeitungsbeispiel gegeben.

```
%INCLUDE Z,DICHT; %ACT #Z,DICHT;
%DCL TEXT CHAR;
%TEXT='FISCHERS FRITZ FISCHT FRISCHE FISCHE';
%TEXT=DICHT(TEXT,6);
%TEXT=DICHT(TEXT,5);
TEXT
```



```
FISCHER FRITZ (1,5)T FR (2,5) (15,6)E
```

Dieses Beispiel zeigt nochmals, daß

- eine wirkungsvolle Textverdichtung nur dann erreicht wird, wenn der zu ersetzende Text für eine @-Angabe wesentlich länger als diese Angabe ist;
- die für die Erkennung eines Makronamens in einer Zeichenkette notwendige Verbindung mit einem Nichtalphazeichen (also beispielsweise auch einem Leerzeichen) bei der PL/1-Makroprogrammierung für die korrekte Anwendung der Verdichtung gewisse Konventionen erfordert.

Das dritte und letzte Beispiel beinhaltet die Transformation von drei parametrisierten Anweisungen in sogenannte Struktogramme. Die drei Anweisungen lauten in Anlehnung an die Strukturierte Programmierung SEQUENZ, SELEKTION und ITERATION. Ihre

Parameterwerte seien selbst Anweisungen in irgendeiner Programmiersprache. Die Anweisungen sollen dabei konkret folgende Transformationen realisieren:

SEQUENZ(a_1 @a_2 @a_3 @... @a_n) generiert

a_1

a_2

a_n

SELEKTION(b_1⨀a_1 @b_2⨀a_2 @... @b_m⨀a_m) generiert

b_1	b_2	...	b_m
a_1	a_2	...	a_m

ITERATION(b @a) generiert

b

a

Dabei kennzeichnen a und a_i ($1 \leq i \leq n$ bzw. $1 \leq i \leq m$) jeweils die Anweisungen in der Programmiersprache und b, b_j ($1 \leq j \leq n$ bzw. $1 \leq j \leq m$) die Bedingungen, bei deren Gültigkeit die jeweiligen Anweisungen ausgeführt werden sollen. Die Zeichen '@' und '⨀' sind hierbei Trennzeichen.

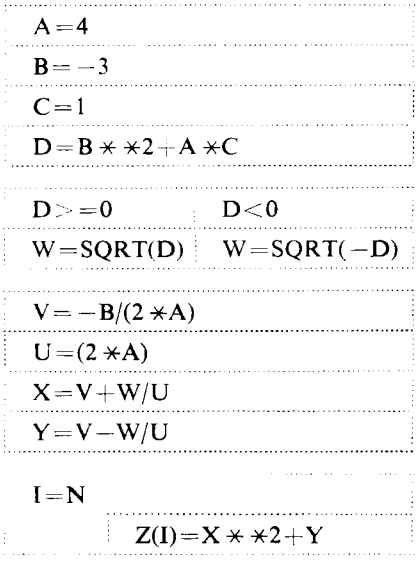
Eine mögliche Implementation der Funktion ITERATION lautet in MACRO-PL1E beispielsweise

```
%ITERATION:PROC(A)RETURNS(CHAR);
           DCL(STRICH,LEER,A,B)CHAR,I FIXED,
           (LENGTH,SUBSTR)BUILTIN;
           STRICH='                    ';
           LEER ='|                   |';
           DO I=1 TO 999;
              IF SUBSTR(A,I,1)=' @' THEN GOTO MA1;
           END;
     MA1: B=SUBSTR(A,1,I-1); A=SUBSTR(A,I+1);
          RETURN(#LIN(STRICH)||#LIN('|'||B||SUBSTR
                 (LEER, LENGTH(B)+3))||#LIN('|      '||
                 SUBSTR(STRICH,7))||#LIN('|      |'||A||
                 SUBSTR(LEER,LENGTH(A)+8))||#LIN('|
                 |'||SUBSTR(LEER,8))||#LIN(STRICH));
%END ITERATION;
```

In analoger Weise können auch SEQUENZ und SELEKTION in MACRO-PL1E implementiert werden. Die Anweisungsfolge

```
SEQUENZ(A=4 @B=-3 @C=1 @D=B**2+A*C)
SELEKTION(D>=0⨀W=SQRT(D) @D<0⨀W=SQRT(-D))
SEQUENZ(V=-B/(2*A) @U=(2*A) @X=V+W/U @Y=V-W/U)
ITERATION(I=N @Z(I)=X**2+Y)
```

ergibt dann das Struktogramm



Diese Makros gestatten allerdings noch nicht die beim Aufstellen von Struktogrammen im allgemeinen mögliche Verschachtelung (zum Beispiel als Selektion von Sequenzen u.a.m.), zeigen aber eine prinzipielle Realisierungsweise.

Übungsaufgaben

4.33. Geben Sie eine Makrodefinition für die im obigen Abschnitt verwendete LIES-Funktion an!

4.34. Formulieren Sie ein Makro *@(n,m)*, das eine Teilzeichenkette von der global zu definierenden Zeichenkettenvariablen TEXT ab Position *n* mit der Länge *m* bildet!

4.35. Schreiben Sie eine Makrodefinition, die die Transformation der SEQUENZ-Anweisung zur Struktogrammgenerierung realisiert!

5. Zusammenfassung und Ausblick

In den vorangegangenen Abschnitten wurden die wesentlichen Grundbegriffe und Anwendungsmöglichkeiten der Makroprogrammierung aufgezeigt. Dabei ist zu erkennen, daß die Makroprogrammierung vor allem der Erweiterung bestimmter Basissprachen dient.
Bei der Konzipierung neuer Programmiersprachen werden im allgemeinen stets Makroprogrammierungsprinzipien einbezogen; ob das das GENERIC-Element der Systemprogrammiersprache ADA oder die REPLACE-Anweisung im neuen Standard-COBOL sind. Dennoch ergeben sich im Laufe der Zeit immer wieder Anwendungen von Makroprozessoren auch für diese neueren Sprachen als Basissprache.
Die Entwicklung dieser Makroprozessoren vollzieht sich dabei in drei Richtungen. Zum einen ist ersichtlich, daß insbesondere die allgemeinen Makroprozessoren Eigenschaften von Programm- beziehungsweise Textgeneratoren überhaupt besitzen und der ursprüngliche Gedanke der Zusammenfassung bestimmter Quelltexte in einer Basissprache weitestgehend verallgemeinert ist.
Die zweite Richtung zeigt durch die Entwicklung der eingebetteten Makroprozessoren [wie beispielsweise SUPERMAC ([28])] eine Orientierung auf das ursprüngliche Prinzip der Makrotechnik. Dabei wird eine minimale Differenz zwischen der Makrosprache und der entsprechenden Basissprache angestrebt.
Die dritte Richtung geht schließlich vom Maschinencodemakro aus und findet insbesondere bei Personalcomputern breite Anwendung. Es handelt sich dabei um die Zusammenfassung von Anweisungen des jeweiligen Betriebs- oder Softwaresystems als sogenannte »keystrokes« und Zuordnung zu einer Taste des Keyboards. Die Betätigung dieser Taste löst dann im Sinne einer Funktionstaste die Abarbeitung der vordefinierten Kommandofolge aus.
Diese und andere Entwicklungen sind eingebettet in den weltweiten Prozeß der Effektivierung der Programmierung und immer breiteren Anwendung der Computer im Lebensprozeß der Menschen.
Dem Leser sei es überlassen, sich auf der Basis der im vorliegenden Buch gegebenen Informationen und Anregungen mit diesen Fragen weiter zu beschäftigen und auseinanderzusetzen. Sollten einige Anregungen auch für die eigene Programmiertätigkeit entstanden sein, so ist der Zweck dieses Buches bereits erfüllt.

6. Hinweise zur Lösung der Übungsaufgaben

1.1. Durch die generative Grammatik G_5 wird die sogenannte DIKE-Sprache (alle möglichen Klammerungsformen algebraischer Ausdrücke) erzeugt.

1.2. Die Regelmenge P_6 hat den Inhalt

$\{S \to ASB,\ S \to SB,\ S \to B,\ A \to a,\ B \to b\}$

1.3. Die gegebene unvollständige Ableitung zur Indexgrammatik G_4 liefert letztlich das Wort *aaaabbbbcccc*.

1.4. Die erzeugte Sprache lautet $L(G_7)=\{a^n b^{2n} | n \geq 1\}$.

1.5. Der in Bild 3 gegebene Programmablaufplan entspricht nicht der Strukturierten Programmierung, da hier durch die Verbindung von der Anweisung A_2 zu A_4 die Forderung nach *einem* Alternativausgang verletzt wird.

1.6. Während bei der Programmverifikation im allgemeinen nach jeder Anweisung das, laut vorgegebenem Algorithmus, geforderte Ergebnis anzugeben ist, erfolgt der Vergleich mit dem beabsichtigten Resultat bei der symbolischen Testung erst nach der Durchführung dieser Testform für das gesamte Programm.

1.7. Parametersprachen lassen im allgemeinen (bei Ausschluß verschachtelter Anweisungsformen) nur eine sequentielle Programmstruktur zu.

1.8. Fachsprachen gehören zu den höheren Programmiersprachen (bzw. problemorientierten Programmiersprachen). (*Aber*: Für den Systemprogrammierer können auch maschinenorientierte Sprachen die Fachsprache sein!)

1.9. Die realisierte Übersetzung durch c_1 und c_2 lautet zusammengefaßt

$$\underset{L,L_2,L'}{c}$$

1.10. Eine Komposition für die Generatoren g_1, g_2 und g_3 hat beispielsweise die Form

$$\underset{L_1,L_2,L}{g_1} \ * \ \underset{L_2,L_3,L}{g_2} \ * \ \underset{L_3,L_4,L}{g_3} \ .$$

1.11. Anwendungen von Prozessoren auf einen Generator g sind:

$$\underset{L_1,L_2,L}{c} \ (\ \underset{L_3,L_4,L_1}{g} \ ) = \underset{L_3,L_4,L_2}{g}$$

als Übersetzung des in L_1 geschriebenen Generators g in einen Generator, der in L_2 geschrieben ist;

$$\underset{L_1,L_2,L}{i} \ (\ \underset{L_3,L_2,L_1}{g} \ ) = w$$

als Interpretation (Abarbeitung) des Generators g mit dem Ergebnis $w \in L_2$;

$$\underset{L_1,L_2,L}{g} \ (\ \underset{L_3,L_4,L_1}{g} \ ) = \underset{L_3,L_4,L_2}{g}$$

als Generierung des in L_1 geschriebenen Generators g in einen in L_2 geschriebenen durch Codegenerierung (als funktionell reduzierte Form zum Compiler). ■

1.12. Eine derartige Komposition verschiedener Prozessoren lautet beispielsweise

$$\begin{array}{ccccccc} i_1 & * & c & * & v & * & i_2 \\ L_A,L_Q,L_1 & & L_Q,L_O,L_2 & & L_O,L_C,L_3 & & L_C,L_R,L_4 \end{array}$$

mit L_Q als Quellprogrammsprache, L_O als Objektsprache, L_C als Computersprache und L_R als Sprache, in der das Resultat dargestellt ist.

2.1. Die generierte Makrosprache ist $L(G_{M_4})=\{ba^nc|n\geqq 0\}$.

2.2. Eine einfache Makrogrammatik G_{M_5} lautet beispielsweise $G_{M_5}=(\{a,b\},\{F(p),S\},\{p\},S,P_{M_5})$ mit

$P_{M_5} = \{S \rightarrow F(b),\ F(p) \rightarrow ap,\ F(p) \rightarrow F(p)b\}$.

2.3. Eine mögliche Form einer erweiterten Makrogrammatik für die zu erzeugende Sprache ist

$G_{M_6} = (\{$03 COLUMN,PIC 9(4) SOURCE ELEMENT (,)., *position, index*$\}$, $\{F(p_1,p_2),S\}$, $S,P_{M_6})$ mit

$P_{M_6} = \{S \rightarrow F($*position, index*$)$,

$F(p_1,p_2) \rightarrow$

$p_2 \rightarrow$ 03 COLUMN *position*

PIC 9(4) SOURCE ELEMENT (*index*).$\}$

2.4. Im Beispiel 16 wird ein Makroaufruf mit einem verteilten Makronamen definiert.

2.5. Makroerweiterungen definieren die Parameteraufrufformen

- Namensaufruf mit einem Makronamen,
- Wertaufruf mit einem Makroaufruf als Wert.

2.6. Beispiel (a) statische Makroerweiterung;
Beispiel (b) dynamische Makroerweiterung.

2.7.

Beispiel	Variablen	Operationen	Funktionen
BEISPIEL 13	(v2)	(o1), (o2)	(p1), (p2)
BEISPIEL 14	(v1), (v2), (v4)	(o1), (o2), (o4)	(p1)
BEISPIEL 17	(v2), (v3)	(o3)	(f4), (p1)

2.8. BEISPIEL 12: *n*:*n*-, Text-, internes Makro
BEISPIEL 15: 1:*n*-, Text-, internes Makro
BEISPIEL 22: *n*:*n*-, Maschinencode-, internes Makro

2.9. Die Bereitstellung des gegebenen Bestandes aus der Makrobibliothek liefert schließlich:
Anweisungen in der Basissprache
Makrodefinitionen
Anweisungen in der Basissprache
Makrodefinitionen

2.10.

#LV	#LW	generierter Text	#LW < #LV
3	1	X = X + A	ja
	2	X = X + A	
		X = X + B	ja
	3	X = X + A	
		X = X + B	
		X = X + C(1,2)	nein

2.11. BEISPIEL 11: spezieller Makroprozessor als Preprozessor (teilweise integriert)
BEISPIEL 12: spezieller Makroprozessor als Preprozessor
BEISPIEL 24: vollständig integrierter Makroprozessor
BEISPIEL 26: spezieller Preprozessor

2.12. Prozessorkompositionsbeispiele für

- MAX: MAX $\ast$ c
  MAX-Sprache,T,L T,L_Z,L'
  mit T für die Sprache des generierten Textes
- MP/1:
  MP/1 $\ast$ $c_{FORTRAN}$
  MP/1-Sprache,FORTRAN,L FORTRAN,ASSEMBLER,L'
- MISEPP:
  MISEPP $\ast$ MACRO-OS
  MISEPP-Sprache,MACRO-OS,L MACRO-OS,ASSEMBLER,L'

2.13. Ein Makroprozessor vom Preprozessortyp kann alle genannten Makroarten – Syntax- und Maschinencodemakros ausgenommen – verarbeiten.

2.14. BEISPIEL 1: Sprachtransformation
BEISPIEL 10: Erweiterung der Basissprache
BEISPIEL 13: Erweiterung der Basissprache
BEISPIEL 18: Implementation einer Parametersprache
BEISPIEL 25: Sprachtransformation.

2.15. Eine weitere Sprachtransformation beschreibt Beispiel 28.

3.1. bis 3.6. siehe Anhang 1!

3.7. Es werden folgende Zeichenketten gebildet
a) 'TECHNIK', 'SPIEL' und 'AS';
b) 'MAKROAUFRUF' und 'TEXTTEIL'.

3.8. siehe Anhang 1!

3.9. a) Stellungstyp,
b) gemischter Typ,
c) Kennworttyp.

3.10. und 3.11. siehe Anhang 1!

4.1. bis 4.7. siehe Anhang 1!

4.8. Dem Makro M werden die Parameterwerte
'"AB"', 'C', '1', 'Y/$\ast$WERT$\ast$/' und
'(,,C,D(4,1))'
übergeben.

4.9. und 4.10. siehe Anhang 1!

4.11. Das Makro L liefert die Länge der über dem Parameter bereitgestellten Zeichenkette.

4.12. bis 4.19. siehe Anhang 1!

4.20. Das Makroprogramm – einschließlich Maßnahmen zur Testunterstützung – lautet vollständig:

```
%DCL(A,B)CHAR,I FIXED;
%A='TEXT';          A
%B='';              B
%DO I=1 TO 5;
      %B=B||A||I;     'I=' I  'B=' B
%END;
B .
```

4.21. Im Makro REPEAT sind für den ersten Test nach der Anweisung 'B=A;' zunächst 'RETURN(B)' und für den zweiten Test zur Kontrolle des korrekten Zyklusanfangs nach der DO-Anweisung die Anweisung 'RETURN(J||B)' einzufügen.

4.22. Die Testtabelle für das in Aufgabe 4.1. gegebene Makroprogramm lautet

I	A	B			
–	TEXT	leer			
1		TEXT	1		
2		TEXT	1TEXT	2	
3		TEXT	1TFXT	2TEXT	3
4		TEXT	1TEXT	2TEXT	3
		TEXT	4		
5		TEXT	1TEXT	2TEXT	3
		TEXT	4TEXT	5	

wobei der letzte Wert für B auch das Generierungsergebnis des Makroprogramms darstellt.

4.23. bis 4.35.: siehe Anhang 1!

Anhang

1. Protokolle zu Beispielen der Makroabarbeitung

1.1. Beispiele zu MACRO-80

Die folgenden Beispiele stellen Lösungen für die Übungsaufgaben 3.1. bis 3.3. dar. Die im Ergebnis der Makroabarbeitung generierten Zeilen sind hierbei durch das Zeichen '+' gekennzeichnet. Die Angabe von Fehlern ('error(s)') ist auf den fehlenden Kontext in Form von MACRO-80-Assembleranweisungen zurückzuführen.

```
          MACRO-80 3.43   28-Jul-86        PAGE    1

   0000                           00100   X        SET     0
                                  00300            IFT     X+1
                                  00400            IRP     Y,<B,C,D,E,F,G,H>
                                  00500            LD      Y,A
                                  00600            ENDM
U 0000'   00 07                +                   LD      B,A
U 0002'   01 07                +                   LD      C,A
U 0004'   02 07                +                   LD      D,A
U 0006'   03 07                +                   LD      E,A
U 0008'   00 07                +                   LD      F,A
U 000A'   00 07                +                   LD      G,A
U 000C'   04 07                +                   LD      H,A
                                  00700            ELSE
                                  00800            IRPC    Z,<ACDEFGH>
                                  00900            LD      Z,B
                                  01000            ENDM
                                  01100            ENDIF



          MACRO-80 3.43   28-Jul-86        PAGE    S

Macros:

Symbols:
0000U   F               0000U   G               0000U   LD
0000    X

7 Fatal error(s)
```

```
          MACRO-80 3.43   28-Jul-86        PAGE    1


                                    00100   JGE     MACRO   P2
                                    00200           JRZ     P2
                                    00300           JP      P2
                                    00400           ENDM
                                    00500   JLT     MACRO   P3
                                    00600           LOCAL   L2
                                    00700           JRZ     L2
                                    00800           JMP     P3
                                    00900   L2:
                                    01000           ENDM
                                    01100   JLE     MACRO   P4
                                    01200           JRZ     P4
                                    01300           JMP     P4
                                    01400           ENDM
  0000'   FE 03                    01500           CPI     3
                                    01600           JGE     M0
U 0002'   00                +                       JRZ     M0
U 0003'   F2 0000             +                       JP      M0
  0006'   3F 04                    01700           MVI     A,4
  0008'   FE 00                    01800           CPI     B
                                    01900           JLT     M1
U 000A'   0E'                 +                       JRZ     ..0000
U 000B'   C3 0000             +                       JMP     M1
  000E'                       +             ..0000:
  000E'   00                        02000           NOP
  000F'   FE 02                     02100           CPI     2
                                    02200           JLE     M2
U 0011'   00                  +                       JRZ     M2
U 0012'   03 0000             +                       JMP     M2



          MACRO-80 3.43   28-Jul-86        PAGE    S

Macros:
JGE             JLE             JLT

Symbols:
000E'   ..0000         0000U   JRZ             0000U   M0
0000U   M1              0000U   M2

6 Fatal error(s)
```

```
                MACRO-80 3.43   28-Jul-86        PAGE    1

                                        00100   AUFG3   MACRO   X
                                        00200           IRPC    Y,<X>
                                        00300           INC     Y
                                        00400           ENDM
                                        00500           ENDM
                                        00600           AUFG3   ABCD
U 0000'   07                +                         INC     A
U 0001'   00                +                         INC     B
U 0002'   01                +                         INC     C
U 0003'   02                +                         INC     D


                MACRO-80 3.43   28-Jul-86        PAGE    S

Macros:
AUFG3

Symbols:
0000U   INC

4 Fatal error(s)
```

1.2. Beispiele zu MACRO-SM

Die hier gegebenen Beispiele sind Lösungen zu den Übungsaufgaben 3.4. bis 3.6. Die generierten Zeilen als Ergebnis der Makroabarbeitung sind dabei durch sechsstellige Zahlen gekennzeichnet.

```
.MAIN.  MACRO M1110  28-JUL-86 07:24  PAGE 1


     1          000005                    A=5
     2          000003                    B=3
     3                                              .IF     GT,A-B
     4          000002                              .REPT   A-B
     5                                              CLR     (R2)+
     6                                              .ENDM
       000000   005022                              CLR     (R2)+
       000002   005022                              CLR     (R2)+
     7                                              .IFF
     8                                              .IRPC   Q1,WERT
     9                                              MOVB    #'Q1,(R1)+
    10                                              .ENDM
    11                                              .ENDC
    12          000001                              .END

.MAIN.  MACRO M1110  28-JUL-86 07.21  PAGE 1-1
SYMBOL TABLE

A      = 000005           B      = 000003

. ABS.  000000      000
        000004      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  60 WORDS  ( 1 PAGES)
DYNAMIC MEMORY:  3170 WORDS  ( 12 PAGES)
ELAPSED TIME:  00:00:32
TTT,LP:/LI:ME=TTT02
```

```
.MAIN.  MACRO M1110  28-JUL-86 07:30  PAGE 1


     1                                          .MACRO  ZYKL   P1,P2,?P3,?P4
     2                                          MOV     C,#P1
     3                                          MOV     HL,#P2-P1
     4                                  P4:     CMP     #A,HL
     5                                          BEQ     P3
     6                                          INC     HL
     7                                          DEC     C
     8                                          BNE     P4
     9                                  P3:
    10                                          .ENDM
    11 000000                           FELD:   .BLKW   20.
    12 000050                           B:      .BLKW   20.
    13 000120                           BFELD:  .BLKW   4
    14 000130                           C:      .BLKW   2
    15 000134                           HL:     .BLKW   4
    16 000144                           A:      .BLKW   2
    17 000150                                   ZYKL    FELD,B
       000150  016727  177754  000000          MOV     C,#FELD
       000156  016727  177752  000120          MOV     HL,#BFELD
       000164  022767  000144  177742  658:     CMP     #A,HL
       000172  001405                           BEQ     648
       000174  005267  177734                   INC     HL
       000200  005367  177724                   DEC     C
       000204  001367                           BNE     658
       000206                             648:
    18         000001                          .END

.MAIN.  MACRO M1110  28-JUL-86 07:30  PAGE 1-1
SYMBOL TABLE

A       000144R         BFELD   000120R         C       000130R
FELD    000000R         HL      000134R         B       000050R

. ABS.  000000      000
        000206      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  183 WORDS  ( 1 PAGES)
DYNAMIC MEMORY:  3170 WORDS  ( 12 PAGES)
ELAPSED TIME:  00:00:52
TTT,LP:/LI:ME=TTT04
```

```
.MAIN.  MACRO M1110  28-JUL-86 07:21  PAGE 1


     1                                                  .MACRO  AUFG3   P1
     2                                                  .IRPC   Q1,<P1>
     3                                                  ADD     R'Q1,<R5>+
     4                                                  .ENDM
     5                                                  .ENDM
     6 000000                                           AUFG3   1234
                                                        .IRPC   Q1,<1234>
                                                        ADD     R'Q1,<R5>+
                                                        .ENDM
       000000  060125                                   ADD     R1,<R5>+
       000002  060225                                   ADD     R2,<R5>+
       000004  060325                                   ADD     R3,<R5>+
       000006  060425                                   ADD     R4,<R5>+
     7         000001                                   .END

.MAIN.  MACRO M1110  28-JUL-86 07:24  PAGE 1-1
SYMBOL TABLE


. ABS.  000000      000
        000010      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  118 WORDS  ( 1 PAGES)
DYNAMIC MEMORY:  3170 WORDS  ( 12 PAGES)
ELAPSED TIME:  00:00:33
TTT,LP:/LI:ME=TTT03
```

1.3. Beispiele zu MACRO-OS

Die folgenden Beispiele sind Lösungen zu den Aufgaben 3.8., 3.10. und 3.11. Die durch die Makroabarbeitung erzeugten Zeilen sind hier mit dem Zeichen '+' nach der laufenden Anweisungsnummer gekennzeichnet.

```
  LOC   OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                       1           LCLA  &A
                                       2           LCLC  &B(2)
                                       3 &A        SETA  5
                                       4 &B(1)     SETC  '8'
                                       5 &B(2)     SETC  '00'
                                       6 .ZYKL     AIF   (&A EQ 1).ZENDE
                                       7 &A        SETA  &A-1
                                       8           L     &B(1),0(&A)
000000 5884 0000        00000          9+          L     8,0(4)
                                      10           ST    &B(1),&A.&B(2)
000004 5080 0190        00190         11+          ST    8,400
                                      12           AGO   .ZYKL
                                      13 .ZYKL     AIF   (&A EQ 1).ZENDE
                                      14 &A        SETA  &A-1
                                      15           L     &B(1),0(&A)
000008 5883 0000        00000         16+          L     8,0(3)
                                      17           ST    &B(1),&A.&B(2)
00000C 5080 012C        0012C         18+          ST    8,300
                                      19           AGO   .ZYKL
                                      20 .ZYKL     AIF   (&A EQ 1).ZENDE
                                      21 &A        SETA  &A-1
                                      22           L     &B(1),0(&A)
000010 5882 0000        00000         23+          L     8,0(2)
                                      24           ST    &B(1),&A.&B(2)
000014 5080 00C8        000C8         25+          ST    8,200
                                      26           AGO   .ZYKL
                                      27 .ZYKL     AIF   (&A EQ 1).ZENDE
                                      28 &A        SETA  &A-1
                                      29           L     &B(1),0(&A)
000018 5881 0000        00000         30+          L     8,0(1)
                                      31           ST    &B(1),&A.&B(2)
00001C 5080 0064        00064         32+          ST    8,100
                                      33           AGO   .ZYKL
                                      34 .ZYKL     AIF   (&A EQ 1).ZENDE
                                      35 .ZENDE    ANOP
                                      36           END
```

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      1          MACRO
                                      2          DEF   &A,&B,&C
                                      3          LCLA  &K
                                      4 &A       DC    X'00
                                      5 &K       SETA  &B
                                      6 .ANF     AIF   (&K EQ 0).END
                                      7 &C.&K    DS    CL&K
                                      8 &K       SETA  &K-1
                                      9          AGO   .ANF
                                     10 .END     ANOP
                                     11          MEND
                                     12          DEF   A,5,B
000000 00                            13+A        DC    X'00
000001                               14+B5       DS    CL5
000006                               15+B4       DS    CL4
00000A                               16+B3       DS    CL3
00000D                               17+B2       DS    CL2
00000F                               18+B1       DS    CL1
                                     19          END
```

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      1          MACRO
                                      2          AUFG5 &I,&P
                                      3          LCLA  &X
                                      4 &X       SETA  0
                                      5 .MA      AIF   (&X EQ &I).ME
                                      6          L     &X,&SYSLIST(2,2*&X+1)
                                      7          ST    &X,&SYSLIST(2,2*&X+2)
                                      8 &X       SETA  &X+1
                                      9          AGO   .MA
                                     10 .ME      ANOP
                                     11          MEND
                                     12    AUFG5 4,(24,44,108,32,16,8,156,52)
000000 5800 0018     00018           13+         L     0,24
000004 5000 002C     0002C           14+         ST    0,44
000008 5810 006C     0006C           15+         L     1,108
00000C 5010 0020     00020           16+         ST    1,32
000010 5820 0010     00010           17+         L     2,16
000014 5020 0008     00008           18+         ST    2,8
000018 5830 009C     0009C           19+         L     3,156
00001C 5030 0034     00034           20+         ST    3,52
                                     21          END
```

1.4. Beispiele zu MACRO-PL1

Die hier gegebenen Beispiele sind Lösungen der Übungsaufgaben 4.1. bis 4.7., 4.9., 4.10. und 4.12. Die Aufgabenstellung bzw. die Lösung als Makroprogramm sind hierbei vom Ma-

```
COMPILE-TIME MACRO PROCESSOR

MACRO SOURCE2 LISTING

  1          /*******************************************/
  2          /*  LOESUNGEN DER UEBUNGSAUFGABEN          */
  3          /*******************************************/
  4          /*** AUFGABE 4.1 ***************************/
  5          %DCL(A,B)CHAR,I FIXED;
  6          %A='TEXT';
  7          %B='';
  8          %DO I=1 TO 5;
  9              %B=B||A||I;
 10          %END;
 11          B
 12          /*******************************************/
 13          /*** AUFGABE 4.2 ***************************/
 14          %DEACT A,B;
 15          %DCL J FIXED;
 16          %DO J=3 TO 10 BY 2;
 17          Z(J)=A(J)*B(J);
 18          %END;
 19          /*******************************************/
 20          /*** AUFGABE 4.3 ***************************/
 21          %DCL(R,S)FIXED,(T,U)CHAR;
 22          %R=132;
 23          %S=R/30;
 24          %T=R;
 25          %U=T||'33  ';
 26          %R=U-16000;
 27          R,S,T,U
 28          /*******************************************/
 29          /*** AUFGABE 4.4. **************************/
 30          %DCL(UND,ODER,WENN,SO)CHAR;
 31          %UND='.AND.';
 32          %ODER='.OR.';
 33          %WENN='IF';
 34          %SO='GOTO';
 35          WENN (A UND B ODER C UND D) SO 13
 36          /*******************************************/
 37          /*** AUFGABE 4.5 ***************************/
 38          %ACT A,B; %DEACT R,S; %DCL C CHAR;
 39          %A='DO X=1 TO N';
 40          %B='KLM';  %C='M1:';
 41          %DO I=1 TO 3;
 42             %C=C||SUBSTR(A,1,3)||SUBSTR(B,I,1)||SUBSTR(A,5);
 43          %END;
 44          C
 45          S=FELD(K,L,M)+R;
 46          END M1;
 47          /*******************************************/
 48          /*** AUFGABE 4.6 ***************************/
 49          P:PROC(Q,N);
 50            %INCLUDE QDEF;
 51            GET EDIT %INCLUDE  L;(F(5));
 52            PUT LIST(MIN %INCLUDE L;);
 53            END P;
 54          /*******************************************/
```

krointerpretationsergebnis getrennt. Die durch Sterne markierten Zeilen sind Kommentare und kennzeichnen die jeweilige Aufgabenstellung und das entsprechende Ergebnis in gleicher Weise.

```
MACRO SOURCE2 LISTING

 55          /*** AUFGABE 4.7 ****************************/
 56          %DEACT A;
 57          %DO I=1 TO 7;
 58              %C=SUBSTR(I,7);
 59              A=SIN(X(C)**2)/SQRT(X(C)-1);
 60          %END;
 61          /*****************************************/
 62          /*** AUFGABE 4.9 ****************************/
 63          %DCL TTT ENTRY(CHAR)RETURNS(CHAR);
 64          %TTT:PROC(A)RETURNS(CHAR);
 65               DCL A CHAR;
 66               RETURN(SUBSTR(A,5)||SUBSTR(A,1,4));
 67          %END TTT;
 68          TTT(WORTSPIEL)
 69          /*****************************************/
 70          /*** AUFGABE 4.10 ***************************/
 71          %INCLUDE REPOT;
 72          REPEAT(ZAHL,POT(3,2))
 73          /*****************************************/
 74          /*** AUFGABE 4.12 ***************************/
 75          %DEACT A,B;
 76          %DCL TEST ENTRY(CHAR,CHAR)RETURNS(CHAR);
 77           %TEST:PROC(A,B)RETURNS(CHAR);
 78                 DCL(A,B)CHAR;
 79                 RETURN('      IF '||A||'NAME='||B||'.NAME  &
 80                                          '||A||'.NR='||B||'.NR &
 81                                                   '||A||'.BETRAG='
 82                              ||B||'.BETRAG THEN');
 83          %END TEST;
 84           TEST(X,Y)
 85          /*****************************************/
 86          /*****************************************/


INCLUDED TEXT FOLLOWS FROM DD.MEMBER = SYSLIB  .QDEF

 87          DCL 1 Q(N);
 88              2 %INCLUDE L;FIXED,
 89              2 (X,Y,Z)CHAR;

INCLUDED TEXT FOLLOWS FROM DD.MEMBER = SYSLIB  .L

 90                  (X1,X2,X3,X4,X5,X6)

INCLUDED TEXT FOLLOWS FROM DD.MEMBER = SYSLIB  .L

 91                  (X1,X2,X3,X4,X5,X6)

INCLUDED TEXT FOLLOWS FROM DD.MEMBER = SYSLIB  .L

 92                  (X1,X2,X3,X4,X5,X6)
```

```
MACRO SOURCE2 LISTING

INCLUDED TEXT FOLLOWS FROM DD.MEMBER = SYSLIB   .REPOT

 93          %DCL REPEAT ENTRY(CHAR,CHAR)RETURNS(CHAR);
 94          %REPEAT:PROC(A,I)RETURNS(CHAR);
 95                  DCL(A,B)CHAR,(I,J)FIXED;
 96                  B=A;
 97                  DO J=1 TO I;
 98                     B=B||A;
 99                  END;
100                  RETURN(B);
101          %END REPEAT;
102          %DCL POT ENTRY(FIXED,FIXED)RETURNS(FIXED);
103          %POT:PROC(I,J)RETURNS(CHAR);
104               DCL(I,J,K,L)FIXED;
105               L=I;
106               DO K=2 TO J;
107                  L=L*I;
108               END;
109               RETURN(L);
110          %END POT;
```

```
                MACRO PHASE OUTPUT
                GENERATED SOURCE STATEMENTS.
/*********************************************/
/*  LOESUNGEN DER UEBUNGSAUFGABEN            */
/*********************************************/
/*** AUFGABE 4.1 ****************************/
 TEXT        1TEXT        2TEXT        3TEXT        4TEXT        5
/*********************************************/
/*** AUFGABE 4.2 ****************************/
Z(         3 )=A(         3 )*B(         3 );
Z(         5 )=A(         5 )*B(         5 );
Z(         7 )=A(         7 )*B(         7 );
Z(         9 )=A(         9 )*B(         9 );
/*********************************************/
/*** AUFGABE 4.3 ****************************/
    -2767 ,         4 ,       132 ,       13233
/*********************************************/
/*** AUFGABE 4.4 ****************************/
 IF  (A  .AND.  B  .OR.  C  .AND.  D)  GOTO  13
/*********************************************/
/*** AUFGABE 4.5 ****************************/
 M1:DO K=1 TO N;DO L=1 TO N;DO M=1 TO N;
S=FELD(K,L,M)+R;
END M1;
/*********************************************/
/*** AUFGABE 4.6 ****************************/
P:PROC(Q,N);
 DCL 1 Q(N),
     2
       (X1,X2,X3,X4,X5,X6)
                          FIXED,
     2 (X,Y,Z)CHAR;
 GET EDIT
      (X1,X2,X3,X4,X5,X6)
                         (F(5));
 PUT LIST(MIN
      (X1,X2,X3,X4,X5,X6)
                        );
 END P;
/*********************************************/
/*** AUFGABE 4.7 ****************************/
    A=SIN(X(  1 )**2)/SQRT(X(  1 )-1);
    A=SIN(X(  2 )**2)/SQRT(X(  2 )-1);
    A=SIN(X(  3 )**2)/SQRT(X(  3 )-1);
    A=SIN(X(  4 )**2)/SQRT(X(  4 )-1);
    A=SIN(X(  5 )**2)/SQRT(X(  5 )-1);
    A=SIN(X(  6 )**2)/SQRT(X(  6 )-1);
    A=SIN(X(  7 )**2)/SQRT(X(  7 )-1);
/*********************************************/
/*** AUFGABE 4.9 ****************************/
SPIELWORT
/*********************************************/
/*** AUFGABE 4.10 ***************************/
ZAHLZAHLZAHLZAHLZAHLZAHLZAHLZAHLZAHL
/*********************************************/
/*** AUFGABE 4.12 ***************************/
        IF XNAME=Y.NAME &
           X.NR=Y.NR &
         X.BETRAG=Y.BETRAG THEN
/*********************************************/
/*********************************************/
```

1.5. Beispiele zu MACRO-PL1E

Die folgenden Beispiele sind Lösungen zu den Übungsaufgaben 4.13. bis 4.19., 4.23. bis 4.35. Die Eingabe ist hierbei durch 'MACROINTERPRETER INPUT' und das Interpretationsergebnis durch 'COMPILER SOURCE LISTING' gekennzeichnet.

```
MACROINTERPRETER INPUT

LINE NO.

  1          /*******************************************/
  2          /***   LOESUNGEN ZU DEN UEBUNGSAUFGABEN   ***/
  3          /*******************************************/
  4          /*** AUFGABE 4.13 ***************************/
  5          TRANS:PROC(X,Y);
  6                DCL X(5),Y(5);
  7                %DCL I FIXED;
  8                %DO I=1 TO 5;
  9                X(I) = Y(6-I);
 10                %END;
 11                END TRANS;
 12          /*******************************************/
 13          /*** AUFGABE 4.14 ***************************/
 14          %DCL A CHAR;
 15          %A='ABC';
 16          SUBSTR(A,2,1)=
 17          %ACT SUBSTR;
 18          SUBSTR(A,2,1)||SUBSTR(A,3);
 19          %DEACT SUBSTR;
 20          %DO I=1 TO 3;
 21             %A='X'||SUBSTR(I,8);
 22          A=SUBSTR(A,I,1);
 23          %END;
 24          /*******************************************/
 25          /*** AUFGABE 4.15 ***************************/
 26          %DEACT A,I;
 27          %MULT:PROC(@1,@2,@3,@4,@5,@6)RETURNS(CHAR);
 28                DCL(@1,@2,@3,@4,@5,@6)CHAR;
 29                IF ^PARMSET(@1) THEN @1='A';
 30                IF ^PARMSET(@2) THEN @2='A';
 31                IF ^PARMSET(@3) THEN @3='C';
 32                IF ^PARMSET(@4) THEN @4='10';
 33                IF ^PARMSET(@5) THEN @5='10';
 34                IF ^PARMSET(@6) THEN @6='10';
 35                RETURN('       DO I=1 TO '||@4||';
 36                                  DO J=1 TO '||@6||';
 37                                                '||@3||'(I,J)=0;
 38                                        DO K=1 TO '||@5||';
 39                                        '||@3||'(I,J)='||@3||'(I,J)+'||
 40                                    @1||'(I,K)*'||@2||'(K,J);
 41                                             END;END;END;');
 42          %END MULT;
 43          %ACT MULT;
 44          MULT()
 45          /*******************************************/
 46          /*** AUFGABE 4.16 ***************************/
 47          %INCLUDE VERIFY; %ACT VERIFY;
 48          %DCL(U,V)CHAR;
 49          %U='TESTBEISPIEL';
 50          %V='LEISE';
 51          VERIFY(V,U)
 52          %V='XYZ';
 53          VERIFY(V,U)
 54          /*******************************************/
 55          /*** AUFGABE 4.17 ***************************/
 56          %INCLUDE FOR; %ACT FOR;
 57          FOR I(S) STEP(-1) UNTIL(1) DO(X(i)=Y(S));
```

```
 58          /*****************************************/
 59          /*** AUFGABE 4.18 ************************/
 60          %ARCCOS:PROC(X)RETURNS(CHAR);
 61                  DCL X CHAR;
 62                  RETURN('ASIN(SQRT(1-'||X||'**2))');
 63          %END ARCCOS;
 64          %ACT ARCCOS;
 65             G=7*ARCCOS(T)-1 T;
 66          /*****************************************/
 67          /*** AUFGABE 4.19 ************************/
 68          %INCLUDE Z; %ACT #Z;
 69          %MIN:PROC(ARRAY,DIM,VAR)STATEMENT RETURNS(CHAR);
 70               DCL(ARRAY,VAR,TEXT)CHAR,(DIM,I)FIXED;
 71               TEXT='MIN(';
 72               DO I=1 TO DIM;
 73                  TEXT=TEXT||ARRAY||'('||#Z(I)||'),';
 74               END;
 75               TEXT=TEXT||VAR||')';
 76               RETURN(TEXT);
 77          %END MIN;
 78          %ACT MIN NORESCAN;
 79          MIN ARRAY(ST) DIM(7) VAR(A1,B3,WERT);
 80          /*****************************************/

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .VERIFY

 81          %VERIFY:PROC(X,Y)RETURNS(FIXED);
 82                 DCL(X,Y)CHAR,I FIXED,(INDEX,LENGTH,SUBSTR)BUILTIN;
 83                 DO I=1 TO LENGTH(X);
 84                    IF INDEX(Y,SUBSTR(X,I,1))=0 THEN RETURN(0);
 85                 END;
 86                 RETURN(1);
 87          %END VERIFY;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .FOR

 88          %FOR:PROC(I,STEP,UNTIL,DO)STATEMENT RETURNS(CHAR);
 89                DCL(I,STEP,UNTIL,DO)CHAR;
 90                RETURN('DO  I='||||' TO '||UNTIL||' BY '||STEP||' ;'
 91                       ||DO||';END;');
 92          %END FOR;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .Z

 93          %#Z:PROC(I)RETURNS(CHAR);
 94               DCL(I,L)FIXED,A CHAR;
 95               A=I;
 96               DO L=7 BY -1 TO 3;
 97                 IF SUBSTR(A,L,1)=' ' THEN RETURN(SUBSTR(A,L+1,8-L));
 98               END;
 99               RETURN(SUBSTR(A,3,6));
100          %END #Z;
```

COMPILER SOURCE LISTING

```
        /*******************************************/
        /***  LOESUNGEN ZU DEN UEBUNGSAUFGABEN    ***/
        /*******************************************/
        /*** AUFGABE 4.13 ***************************/
        TRANS:PROC(X,Y);
              DCL X(5),Y(5);
              X(        1)= Y(6-        1);
              X(        2)= Y(6-        2);
              X(        3)= Y(6-        3);
              X(        4)= Y(6-        4);
              X(        5)= Y(6-        5);
              END TRANS;
        /*******************************************/
        /*** AUFGABE 4.14 ***************************/
        SUBSTR(ABC,2,1)=
        B||C;
        X1=SUBSTR(X1,        1,1);
        X2=SUBSTR(X2,        2,1);
        X3=SUBSTR(X3,        3,1);
        /*******************************************/
        /*** AUFGABE 4.15 ***************************/
                  DO I=1 TO 10;
               DO J=1 TO 10;
                     C(I,J)=0;
        DO K=1 TO 10;
C(I,J)=C(I,J)+A(I,K)*A(K,J);
                END;END;END;
        /*******************************************/
        /*** AUFGABE 4.16 ***************************/
              1
              0
        /*******************************************/
        /*** AUFGABE 4.17 ***************************/
        DO  I=S TO 1 BY -1 ;X(I)=Y(S) ;END;
        /*******************************************/
        /*** AUFGABE 4.18 ***************************/
            G=7*ASIN(SQRT(1-T**2))-1/T;
        /*******************************************/
        /*** AUFGABE 4.19 ***************************/
        MIN(ST(1),ST(2),ST(3),ST(4),ST(5),ST(6),ST(7),A1,B3,WERT)
        /*******************************************/
```

```
MACRONINTERPRETER INPUT

LINE NO.

  1          /*****************************************/
  2          /* LOESUNGEN DER UEBUNGSAUFGABEN          ***/
  3          /*****************************************/
  4          /*** AUFGABE 4.23 ***************************/
  5          %INCLUDE MAX; %ACT MAX NORESCAN;
  6          %INCLUDE LIN; %ACT #LIN;
  7          MAX(ALGOL 68,WERT1,WERT2,WERT3)
  8          MAX(FORTRAN 77,E,F,G)
  9          /*****************************************/
 10          /*** AUFGABE 4.24 ***************************/
 11          %INCLUDE Z; %ACT #Z;
 12          %ASM:PROC(I)RETURNS(CHAR);
 13               DCL(I,J)FIXED,TEXT CHAR;
 14               TEXT='';
 15               DO J=1 TO I;
 16                 TEXT=TEXT||#LIN('  ADD  R'||#Z(J)||',(R5)+');
 17               END;
 18               RETURN(TEXT);
 19          %END ASM;
 20          %ACT ASM;
 21          ASM(4)
 22          /*****************************************/
 23          /*** AUFGABE 4.25 ***************************/
 24          %INCLUDE ARG; %ACT #ARG;
 25          %AUFG425:PROC(X,Y,Z)RETURNS(CHAR);
 26                   DCL(X,Y,Z,A,B,C,TEXT)CHAR,(I1,I2,I3,J)FIXED;
 27                   I1=0; I2=0; I3=0; TEXT='';
 28                   M1:A=#ARG(X,I1,J);
 29                      B=#ARG(Y,I2,J);
 30                      C=#ARG(Z,I3,J);
 31                      TEXT=TEXT||#LIN(' COMPUTE  '||A||' = '||B||
 32                           ' + '||C||'.');
 33                      IF J=0 THEN GOTO M1;
 34                   RETURN(TEXT);
 35          %END AUFG425;
 36          %ACT AUFG425;
 37          AUFG425((A,B,C,D),(W,X,Y,Z),(K1,K2,K3,K4))
 38          /*****************************************/
 39          /*** AUFGABE 4.26 ***************************/
 40          %INCLUDE JCL; %ACT JCL;
 41          %JOB:PROC(LISTE)RETURN(CHAR);
 42               DCL(LISTE,P,E,A,TEXT)CHAR,(I,J)FIXED;
 43               TEXT=''; I=0;
 44               M:P=#ARG(LISTE,I,J);
 45                 E=#ARG(LISTE,I,J);
 46                 A=#ARG(LISTE,I,J);
 47                 TEXT=TEXT||JCL(P,E,A);
 48               IF J=0 THEN GOTO M;
 49               RETURN(TEXT);
 50          %END JOB;
 51          %ACT JOB;
 52          JOB((P1,E1,A1,PROG,EIN,AUS,BER,VON,NACH))
```

```
 53          /*****************************************/
 54          /*** AUFGABE 4.27 ************************/
 55          %INCLUDE MORSE;
 56          B E I S P I E L     E I N E R
 57          T E X T U M S E T Z U N G
 58          /*****************************************/
 59          /*** AUFGABE 4.28 ************************/
 60          %EINLADUNG:PROC(A,B,C,D,E)RETURNS(CHAR);
 61          DCL(A,B,C,D,E)CHAR;
 62          RETURN(#LIN('WERTER KOLLEGE '||A)||
 63                 #LIN('AM '||B||' FINDET UM '||C||' UHR IM RAUM '||D)||
 64                 #LIN('UNSERE NAECHSTE BERATUNG STATT. WIR ERWARTEN')||
 65                 #LIN('IHRE TEILNAHME.')||
 66                 #LIN('      MIT FREUNDLICHEN GRUESSEN!')||
 67                 #LIN('            '||E));
 68          %END EINLADUNG;
 69          %ACT EINLADUNG;
 70          EINLADUNG(MUELLER,2.2.87,19.00,204,DER VORSTAND)
 71          /*****************************************/
 72          /*****************************************/

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .MAX

 73          %MAX:PROC(S,A,B,C)RETURNS(CHAR);
 74               DCL(A,B,C,K,L,M,N,S)CHAR,SUBSTR BUILTIN;
 75               S=SUBSTR(S,1,2);
 76               IF S='FO' THEN K='.GT.'; ELSE K='>';
 77               IF S='FO' | S='PL' THEN L='';ELSE L=':';
 78               IF S='FO' THEN M=''; ELSE M=';';
 79               IF S='FO' THEN N='END IF'; ELSE IF S='AL' THEN N='FI';
 80                                                        ELSE N='';
 81               RETURN(#LIN('IF ('||A||K||B||') THEN ')||
 82                      #LIN('IF ('||A||K||C||') THEN ')||
 83                      #LIN('MAX'||L||'='||A||M)||
 84                      #LIN('ELSE')||#LIN('MAX'||L||'='||C||M)||
 85                      #LIN('ELSE IF ('||B||K||C||') THEN ')||
 86                      #LIN('MAX'||L||'='||B||M)||
 87                      #LIN('ELSE')||#LIN('MAX '||L||'='||C||M)||
 88                      #LIN(N));
 89          %END MAX;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .LIN

 90          %#LIN:PROC(ZK)RETURNS(CHAR);
 91                DCL(ZK,ZKR)CHAR,LENGTH BUILTIN;
 92                DCL LE CHAR;
 93                LE='                    ';
 94                LE=LE||LE||LE||'            ';
 95                ZKR=ZK||SUBSTR(LE,1,71-LENGTH(ZK));
 96                RETURN(ZKR);
 97          %END #LIN;
```

```
TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .Z

 98          %#Z:PROC(I)RETURNS(CHAR);
 99              DCL(I,L)FIXED,A CHAR;
100              A=I;
101              DO L=7 BY -1 TO 3;
102                IF SUBSTR(A,L,1)=' ' THEN RETURN(SUBSTR(A,L+1,8-L));
103              END;
104              RETURN(SUBSTR(A,3,6));
105          %END #Z;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .ARG

106          %#ARG:PROC(X,I,J)RETURNS(CHAR);
107                DCL(I,J,K,L)FIXED,X CHAR;
108                L=I; J=0;
109                IF I>0 THEN DO;DO K=I TO 999;
110                                  IF SUBSTR(X,K,1)=')' THEN DO;
111                                     J=1;RETURN(SUBSTR(X,I,K-I)); END;
112                                  IF SUBSTR(X,K,1)=',' THEN DO;
113                                     I=K+1;RETURN(SUBSTR(X,L,K-L));END;
114                               END;
115                           END;
116                      ELSE DO;DO K=1 TO 999;
117                                  IF SUBSTR(X,K,1)=',' THEN DO;
118                                     I=K+1;RETURN(SUBSTR(X,2,K-2));END;
119                                  IF SUBSTR(X,K,1)=')' THEN DO;
120                                     J=1;RETURN(SUBSTR(X,2,K-2));END;
121                               END;
122                          END;
123                   RETURN('');
124          %END #ARG;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .JCL

125          %JCL:PROC(P1,D1,D2,P2,D3,D4)RETURNS(CHAR);
126               DCL(P1,P2,D1,D2,D3,D4,TEXT)CHAR,PARMSET BUILTIN;
127               TEXT=#LIN('//   EXEC  PGM='||P1);
128               TEXT=TEXT||#LIN('//EIN  DD  DSN='||D1||',DISP=SHR');
129               TEXT=TEXT||#LIN('//AUS  DD  DSN=&BIB('||D2||
130                    '),DISP=SHR');
131               IF PARMSET(P2) THEN DO;
132                 TEXT=TEXT||#LIN('//   EXEC  PGM='||P2);
133                 TEXT=TEXT||#LIN('//EIN  DD  DSN='||D3||
134                      ',DISP=SHR');
135                 TEXT=TEXT||#LIN('//AUS  DD  DSN=&BIB('||D4||
136                      '),DISP=SHR');
137                                   END;
138               RETURN(TEXT);
139          %END JCL;



TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .MORSE

140          %DCL(A,B,C,D,E,F,G,H,I,J,K,L,M)CHAR;
141          %DCL(N,O,P,Q,R,S,T,U,V,W,X,Y,Z)CHAR;
142          %A='.-';  %B='-...'; %C='-.-.'; %D='-..';  %E='.';
143          %F='..-.';%G='--.';  %H='....'; %I='..';   %J='.---';
144          %K='-.-'; %L='.-..'; %M='--';   %N='-.';   %O='---';
145          %P='.--.';%Q='--.-'; %R='.-.';  %S='...';  %T='-';
146          %U='..-'; %V='...-'; %W='.--';  %X='-..-'; %Y='-.--';
147          %Z='--..';
```

COMPILER SOURCE LISTING

```
/*******************************************/
/* LOESUNGEN DER UEBUNGSAUFGABEN           ***/
/*******************************************/
/*** AUFGABE 4.23 ***************************/
IF (WERT1>WERT2) THEN
IF (WERT1>WERT3) THEN
MAX:=WERT;
ELSE
MAX:=WERT3;
ELSE IF (WERT2>WERT3) THEN
MAX:=WERT2;
ELSE
MAX :=WERT3;
FI

IF (E.GT.F) THEN
IF (E.GT.G) THEN
MAX=E
ELSE
MAX=G
ELSE IF (F.GT.G) THEN
MAX=F
ELSE
MAX =G
END IF

/*******************************************/
/*** AUFGABE 4.24 ***************************/
  ADD  R1,(R5)+
  ADD  R2,(R5)+
  ADD  R3,(R5)+
  ADD  R4,(R5)+

/*******************************************/
/*** AUFGABE 4.25 ***************************/
 COMPUTE  A = W + K1.
 COMPUTE  B = X + K2.
 COMPUTE  C = Y + K3.
 COMPUTE  D = Z + K4.

/*******************************************/
/*** AUFGABE 4.26 ***************************/
//   EXEC  PGM=P1
//EIN  DD  DSN=E1,DISP=SHR
//AUS  DD  DSN=&BIB(A1),DISP=SHR
//   EXEC  PGM=PROG
//EIN  DD  DSN=EIN,DISP=SHR
//AUS  DD  DSN=&BIB(AUS),DISP=SHR
//   EXEC  PGM=BER
//EIN  DD  DSN=VON,DISP=SHR
//AUS  DD  DSN&BIB(NACH),DISP=SHR

/*******************************************/
/*** AUFGABE 4.27 ***************************/
-... . .. ... .--. .. . .-.. .. -. . .-.
- . -..- - ..- -- ... . - --.. ..- -. --.
```

COMPILER SOURCE LISTING

```
/*******************************************/
/*** AUFGABE 4.28 ***************************/
WERTER KOLLEGE MUELLER
AM 2.2.87 FINDET UM 19.00 UHR IM RAUM 204
UNSERE NAECHSTE BERATUNG STATT. WIR ERWARTEN
IHRE TEILNAHME.
     MIT FREUNDLICHEN GRUESSEN!
          DER VORSTAND

/*******************************************/
/*******************************************/
```

```
MACROINTERPRETER INPUT

LINE NO.

  1          /*******************************************/
  2          /* LOESUNGEN DER UEBUNGSAUFGABEN           ***/
  3          /*******************************************/
  4          /*** AUFGABE 4.29 **************************/
  5          %ZIFFERN:PROC(A)RETURNS(CHAR);
  6                   DCL(A,B,C)CHAR,(LENGTH,SUBSTR,INDEX)BUILTIN,I FIXED;
  7                   C='0123456789';
  8                   DO I=1 TO LENGTH(A);
  9                      IF INDEX(C,SUBSTR(A,I,1))=0 THEN RETURN(0);
 10                   END;
 11                   RETURN(1);
 12          %END ZIFFERN;
 13          %ACT ZIFFERN;
 14          ZIFFERN(1234) ZIFFERN(12X4)
 15          /*******************************************/
 16          /*** AUFGABE 4.30 **************************/
 17          %DCL(ARRAY,KELLER,MAPROG)CHAR;
 18          %MAPROG=''; %KELLER=''; %ARRAY='MADEFMALESMADRU';
 19          %INCLUDE SYNKON; %ACT SYNKON;
 20          SYNKON(MADEF(X,4,6))
 21          SYNKON(MADEF(Y,4,6))
 22          SYNKON(MADEF(Z,4,6))
 23          SYNKON(MALES(X,4,6))
 24          SYNKON(MALES(Y,4,6))
 25          SYNKON(MABER(X,4,6,Y,6,Z,+))
 26          SYNKON(MADRU(Z,4,6))
 27          %INCLUDE MATRIX;
 28          %ACT MADEF,MALES,MABER,MADRU;
 29          %DEACT PUT;
 30      MAPROG
 31          /*******************************************/
 32          /*** AUFGABE 4.31 **************************/
 33          %DCL TEXT CHAR,I FIXED;
 34          %ARRAY='   10   15   20   25   30   35   40   45   50   55';
 35          %TEXT='';
 36          %INCLUDE LIN,Z; %ACT #LIN,#Z;
 37          %DO I=1 TO 10;
 38              %TEXT=TEXT||#LIN('   02 COLUMN '||SUBSTR(LIES(I),4)||
 39                    ' PIC 9(4) SOURCE ELEMENT ( '||#Z(I)||' ).');
 40          %END;
 41          TEXT
 42          /*******************************************/
 43          /*** AUFGABE 4.32 **************************/
 44          %KELLER='';
 45          %PRUEF:PROC(A)RETURNS(CHAR);
 46                 DCL(A,X,Y)CHAR,(INDEX,SUBSTR)BUILTIN;
 47                 X=INDEX(A,'+'); IF X^=0 THEN GOTO M1;
 48                 X=INDEX(A,'-'); IF X^=0 THEN GOTO M1;
 49                 RETURN(0);
 50            M1:  Y=STACK(SUBSTR(A,1,X-1));
 51                 Y=STACK(SUBSTR(A,X+1));
 52                 IF NAME(PUT) & NAME(PUT) THEN RETURN(1);
 53                 RETURN(0);
 54          %END PRUEF;
```

```
 55          %DCL TERM CHAR; %TERM='X1-WERT2';
 56          %ACT PRUEF;
 57          PRUEF(TERM)
 58          %TERM='03/X';
 59          PRUEF(TERM)
 60          /*******************************************/

TEXT INCLUDED FROM DD.MEMBER = SYSLIB   .SYNKON

 61          %INCLUDE STACK,PUT,NAME,LIES1;
 62          %ACT STACK,PUT,NAME,LIES;
 63    %SYNKON:PROC(ANW)RETURNS(CHAR);
 64          DCL(A,B,ANW)CHAR,(SUBSTR,LENGTH)BUILTIN,(I,J,K)FIXED;
 65          J=1; K=0; KELLER='';
 66          /*******************************************/
 67          /***  ZERLEGUNG DER ANWEISUNG IN DIE SYNTAKTI- */
 68          /***  SCHEN ELEMENTE   NAME ( NAME , ZAHL  USW.*/
 69          /***  UND SPEICHERUNG IN DIE GLOBALE VARIABLE  */
 70          /***  KELLER                                   */
 71          /*******************************************/
 72          DO I=1 TO LENGTH(ANW);
 73            IF SUBSTR(ANW,I,1)='(' THEN DO;B=STACK(SUBSTR(ANW,1,I-1));
 74                                           K=K+1;    J=I+1;
 75                                        END;
 76            IF SUBSTR(ANW,I,1)=',' THEN DO;B=STACK(SUBSTR(ANW,J,I-J));
 77                                           K=K+1;    J=I+1;
 78                                        END;
 79            IF SUBSTR(ANW,I,1)=')' THEN DO;B=STACK(SUBSTR(ANW,J,I-J));
 80                                           K=K+1;    GOTO KONT1;
 81                                        END;
 82          END;
 83          GOTO FEHLER;
 84       KONT1:/*******************************************/
 85            /*** KONTROLLE DER ANZAHL DER SYNTAKTISCHEN  */
 86            /*** ELEMENTE (IN K ENTHALTEN)               */
 87            /*******************************************/
 88          IF K=8 | K=4 THEN;ELSE GOTO FEHLER;
 89            /*******************************************/
 90            /*** KONTROLLE DER SYNTAKTISCHEN ELEMENTE    */
 91            /*******************************************/
 92          IF K=8 THEN DO;A=PUT;
 93                       IF A='+' | A='-' | A='*' THEN;
 94                            ELSE GOTO FEHLER;
 95                       IF NAME(PUT)=1 & ZIFFERN(PUT)=1 & NAME(PUT)=1
 96                         & ZIFFERN(PUT)=1 & ZIFFERN(PUT)=1 &
 97                         NAME(PUT)=1 THEN; ELSE GOTO FEHLER;
 98                       A=PUT; IF A='MABER' THEN GOTO KONT2;
 99                       GOTO FEHLER;
100                     END;
101          IF ZIFFERN(PUT)=1 & ZIFFERN(PUT)=1 & NAME(PUT)=1
102                THEN; ELSE GOTO FEHLER;
103          A=PUT;
104          DO I=1 TO 3;
105            IF A=LIES(I) THEN GOTO KONT2;
106          END;
107          GOTO FEHLER;
```

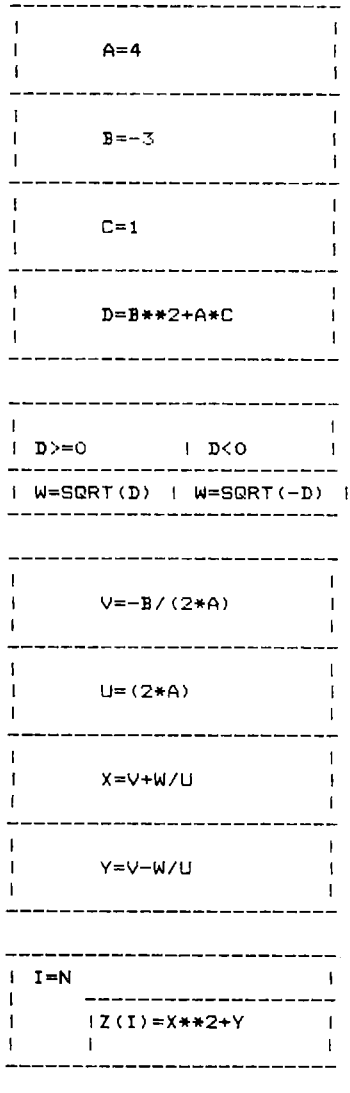
```
108       KONT2:/*******************************************/
109             /*** ANFUEGUNG DER KONTROLLIERTEN ANWEISUNG  */
110             /*** AN DIE GLOBALE VARIABLE  MAPROG         */
111             /*******************************************/
112           MAPROG = MAPROG || ANW;
113           RETURN('');
114             /*******************************************/
115             /*** FEHLERMELDUNG                           */
116             /*******************************************/
117       FEHLER:RETURN('FEHLERHAFTE ANWEISUNG');
118           %END SYNKON;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .STACK

119           %STACK:PROC(WERT)RETURNS(CHAR);
120                  DCL WERT CHAR;
121                  KELLER=KELLER||'&'||WERT;
122                  RETURN('');
123           %END STACK;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .PUT

124           %PUT:PROC RETURNS(CHAR);
125                DCL WERT CHAR,(I,J)FIXED,LENGTH BUILTIN;
126                DCL SUBSTR BUILTIN;
127                J=LENGTH(KELLER);
128                DO I=J TO 1 BY -1;
129                  IF SUBSTR(KELLER,I,1)='&' THEN GOTO MA1;
130                END;
131                RETURN('');
132                MA1:WERT=SUBSTR(KELLER,I+1,J-I);
133                KELLER=SUBSTR(KELLER,1,I-1);
134                RETURN(WERT);
135           %END PUT;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .NAME

136           %ALFA:PROC(X)RETURNS(FIXED);
137                 DCL(A,X)CHAR,INDEX BUILTIN;
138                 A='ABCDEFGHIJKLMNOPQRSTUVWXYZ@$#';
139                 IF INDEX(A,X)^=0 THEN RETURN(1);
140                 RETURN(0);
141           %END ALFA;
142           %ZAHL:PROC(A)RETURN(FIXED);
143                 DCL(A,Z)CHAR,INDEX BUILTIN;
144                 Z='0123456789';
145                 IF INDEX(Z,A)^=0 THEN RETURN(1);
146                 RETURN(0);
147           %END ZAHL;
148           %NAME:PROC(A)RETURNS(FIXED);
149                 DCL(A,B)CHAR,I FIXED,(LENTGH,SUBSTR)BUILTIN;
150                 B=SUBSTR(A,1,1);
151                 IF ALFA(B)=0 THEN RETURN(0);
152                 DO I=2 TO LENGTH(A);
153                    B=SUBSTR(A,I,1);
154                    IF ALFA(B)=0 & ZAHL(B)=0 THEN RETURN(0);
155                 END;
156                 RETURN(1);
157           %END NAME;
158           %ACT ALFA,ZAHL;
```

```
COMPILER SOURCE LISTING

            /*****************************************/
            /* LOESUNGEN DER UEBUNGSAUFGABEN          ***/
            /*****************************************/
            /*** AUFGABE 4.29 ***************************/
                  1         0
            /*****************************************/
            /*** AUFGABE 4.30 ***************************/




      DCL X(4,6)FIXED;DCL Y(4,6)FIXED;DCL Z(4,6)FIXED;GET EDIT(((X(I,J)
DO J=1 TO 6)DO I=1 TO 4))(6F(5));GET EDIT(((Y(I,J)DO J=1 TO 6)DO I=1 TO
 4))(6F(5));DO I=1 TO 4;DO J=1 TO 6;Z(I,J)=X(I,J)+Y(I,J);END;END;PUT
EDIT(((Z(I,J)DO J=1 TO 6)DO I=1 TO 4))(6F(5));
            /*****************************************/
            /*** AUFGABE 4.31 ***************************/
               02 COLUMN 10 PIC 9(4) SOURCE ELEMENT ( 1 ).
               02 COLUMN 15 PIC 9(4) SOURCE ELEMENT ( 2 ).
               02 COLUMN 20 PIC 9(4) SOURCE ELEMENT ( 3 ).
               02 COLUMN 25 PIC 9(4) SOURCE ELEMENT ( 4 ).
               02 COLUMN 30 PIC 9(4) SOURCE ELEMENT ( 5 ).
               02 COLUMN 35 PIC 9(4) SOURCE ELEMENT ( 6 ).
               02 COLUMN 40 PIC 9(4) SOURCE ELEMENT ( 7 ).
               02 COLUMN 45 PIC 9(4) SOURCE ELEMENT ( 8 ).
               02 COLUMN 50 PIC 9(4) SOURCE ELEMENT ( 9 ).
               02 COLUMN 55 PIC 9(4) SOURCE ELEMENT ( 10 ).

            /*****************************************/
            /*** AUFGABE 4.32 ***************************/
                     1
                     0
            /*****************************************/
```

```
MACROINTERPRETER INPUT

LINE NO.

  1          /*****************************************/
  2          /*  LOESUNGEN DER UEBUNGSAUFGABEN          ***/
  3          /*****************************************/
  4          /*** AUFGABE 4.33 ***************************/
  5          %LIES:PROC(ARRAY,I,J)RETURNS(CHAR);
  6                DCL(I,J)FIXED,ARRAY CHAR,SUBSTR BUILTIN;
  7                RETURN(SUBSTR(ARRAY,(I-1)*40+(J-1)*8+1,8));
  8          %END LIES;
  9          %ACT LIES;
 10          %INCLUDE DABESCH;
 11          LIES(DABESCH,2,1)  LIES(DABESCH,6,2)   LIES(DABESCH,3,5)
 12          /*****************************************/
 13          /*** AUFGABE 4.34 ***************************/
 14          %@:PROC(I,J)RETURNS(CHAR);
 15             DCL(I,J)FIXED;
 16             RETURN(SUBSTR(TEXT,I,J));
 17          %END @;
 18          %DCL TEXT CHAR;
 19          %ACT @;
 20          %TEXT='WERT KANN SEIN: ZAHL-@(1,4) UND ZEICHEN-@(17,4) ';
 21          TEXT
 22          /*****************************************/
 23          /*** AUFGABE 4.35 ***************************/
 24          %INCLUDE ITR,SEL,SEQ,LIN;
 25          %ACT ITERATION,SELEKTION,SEQUENZ,#LIN;
 26          SEQUENZ(A=4@B=-3@C=1@D=B**2+A*C)
 27          SELEKTION(D>=0ØW=SQRT(D)@D<0ØW=SQRT(-D))
 28          SEQUENZ(V=-B/(2*A)@U=(2*A)@X=V+W/U@Y=V-W/U)
 29          ITERATION(I=N@Z(I)=X**2+Y)
 30

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .DABESCH

 31          %DCL DABESCH CHAR;
 32          %DABESCH='     NAME        2                          '||
 33                   'GEBDATUM         1  GESAMT        - ABZUEGE'||
 34                   '  GEHALT         1                          '||
 35                   '  GESAMT         1                          '||
 36                   ' ABZUEGE         1                          '||
 37                   '    ALTER        1   DATUM        -GEBDATUM';

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .ITR

 38          %ITERATION:PROC(A)RETURNS(CHAR);
 39              DCL(STRICH,LEER,A,B)CHAR,I FIXED,(LENGTH,SUBSTR)BUILTIN;
 40              STRICH='-------------------------';
 41              LEER  ='|                        |';
 42              DO I=1 TO 999;
 43                 IF SUBSTR(A,I,1)='@' THEN GOTO M1;
 44              END;
 45            M1:B=SUBSTR(A,1,I-1); A=SUBSTR(A,I+1);
 46              RETURN(#LIN(STRICH)||#LIN('|  '||B||SUBSTR(LEER,
 47                     LENGTH(B)+3))||#LIN('|      '||SUBSTR(STRICH,7))||
 48                     #LIN('|      |'||A||SUBSTR(LEER,LENGTH(A)+8))||
 49                     #LIN('|      |'||SUBSTR(LEER,8))||#LIN(STRICH));
 50          %END ITERATION;
```

```
TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .SEL

 51          %SELEKTION:PROC(A)RETURNS(CHAR);
 52                 DCL(STRICH,LEER,A,B,C,D)CHAR,(I,J,K,L,M)FIXED;
 53                    (LENGTH,SUBSTR)BUILTIN;
 54                 STRICH='--------------------------';
 55                 LEER  ='|                         |';
 56                 I=0;   L=LENGTH(A);
 57                 DO J=1 TO L;
 58                      IF SUBSTR(A,J,1)='¤' THEN I=I+1;
 59                 END;
 60                 M=25/I;
 61                 B=#LIN(STRICH)||#LIN(LEER);
 62                 J=1;  I=0;  C='|';  D='|';
 63               MA1:I=I+1;
 64                 IF I>=L THEN GOTO MA2;
 65                 IF SUBSTR(A,I,1)='¤' THEN DO;C=C||' '||SUBSTR(A,J,I-J)
 66                           ||SUBSTR(LEER,27-M+I-J);   K=I+1;
 67                           GOTO MA1;
 68                                          END;
 69                 IF SUBSTR(A,I,1)='@' THEN DO;D=D||' '||SUBSTR(A,K,I-K)
 70                           ||SUBSTR(LEER,27-M+I-K);   J=I+1;
 71                                          END;
 72                 GOTO MA1;
 73               MA2:D=D||' '||SUBSTR(A,K,L-K+1)||SUBSTR(LEER,
 74                           27-M+I-K);
 75                 B=B||#LIN(C)||#LIN(STRICH)||#LIN(D)||#LIN(STRICH);
 76                 RETURN(B);
 77          %END SELEKTION;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .SEQ

 78          %SEQUENZ:PROC(A)RETURNS(CHAR);
 79                 DCL(STRICH,LEER,A,B)CHAR,(I,J,K)FIXED,LENGTH BUILTIN;
 80                 DCL SUBSTR BUILTIN;
 81                 STRICH='--------------------------';
 82                 LEER  ='|                         |';
 83                 I=0;   J=1;   K=LENGTH(A);
 84                 B=#LIN(STRICH)||#LIN(LEER);
 85              MA1:I=I+1;
 86                 IF I=K THEN GOTO MA2;
 87                 IF SUBSTR(A,I,1)='@' THEN DO;B=B||#LIN('|      '||
 88                           SUBSTR(A,J,I-J)||SUBSTR(LEER,7+I-J)||
 89                           #LIN(LEER)||#LIN(STRICH)||#LIN(LEER);
 90                           J=I+1;
 91                                          END;
 92                 GOTO MA1;
 93              MA2:B=B||#LIN('|      '||SUBSTR(A,J,K-J+1)||SUBSTR
 94                           (LEER,8+K-J))||#LIN(LEER)||#LIN(STRICH);
 95                 RETURN(B);
 96          %END SEQUENZ;

TEXT INCLUDED FROM DD.MEMBER = SYSLIB  .LIN

 97          %#LIN:PROC(ZK)RETURNS(CHAR);
 98                DCL(ZK,ZKR)CHAR,LENGTH BUILTIN;
 99                DCL LE CHAR;
100                LE='                          ';
101                LE=LE||LE||LE||'              ';
102                ZKR=ZK||SUBSTR(LE,1,71-LENGTH(ZK));
103                RETURN(ZKR);
104          %END #LIN;


NO PREPROCESSOR DIAGNOSTIC MESSAGES PRODUCED
```

```
COMPILER SOURCE LISTING

/*******************************************/
/*  LOESUNGEN DER UEBUNGSAUFGABEN          ***/
/*******************************************/
/*** AUFGABE 4.33 ***************************/
GEBDATUM          1     ABZUEGE
/*******************************************/
/*** AUFGABE 4.34 ***************************/
WERT KANN SEIN: ZAHL-WERT UND ZEICHEN-ZAHL
/*******************************************/
/*** AUFGABE 4.35 ***************************/
```



2. Alphabetische Übersicht der bekanntesten Makrosprachen

Makrosprache	Makroprozessor	Basissprache	Computer	Autor(en)	Institution	Jahr
ACSL-Makros	ACSL-Interpreter	FORTRAN	IBM u.a.	GAUTHIER	Mitchell & Gauthier Association Incorporation, Massachusetts, USA	1979
ALEC	ALEC-Prozessor	ASSEMBLER	ICL 1906A	NAPPER, FISHER	Universität von Manchester, Großbritannien	1974
ALGORITHM 622	FORTRAN-Routinen	FORTRAN	PDP u.a.	RICE, RIBBENS, WARD	Purdue Universität, West Lafayette, USA	1984
APAREL	APAREL-Prozessor	PL/1	IBM 360	BALZER, FARBER	Rand Corporation, Santa Monica, Kalifornien, USA	1969
AS6500	MACRO-11	ASSEMBLER	PDP-11/34	TRAVENIER, NOTREDAME	Laboratory voor Electronica en Meetechnick, Ghent, Belgien	1980
BESM6-Makrosprache	BESM6-Compiler	wählbar	BESM6	RAUPACH, TARTZ	BESM6-Rechenzentrum des IFE, Berlin, DDR	1972
BIDABS-M	OS-Makroassembler	ASSEMBLER	ESER	Autorenkollektiv	Rechenzentrum der Deutschen Staatsbibliothek, Berlin, DDR	1982
BLISS-11-Makrosprache	BLISS-11-Compiler	Compiler	PDP-11	RAMMELT	Technische Universität Dresden, Hersteller: Universität Pittsburg, USA	1975
CLEF	CLEF-Prozessor	COBOL	IBM u.a.	TRIANCE, LAYZELL	Universität von Manchester, Großbritannien	1985
CMS	PL/1-Makrointerpreter	COBOL	ESER	DUMKE	Technische Hochschule Magdeburg, DDR	1984
COBRA	COBRA-Makroprozessor	COBOL	CDC u.a.	HAMILTON, FINLAYSON	Cobra Systems & Programming Ltd., Camberley, Großbritannien	1973

C-Preprozessor-sprache	C-Preprozessor	C	PDP-11	KERNIGHAN, RITCHIE	Bell Laboratories, New Jersey, USA	1981
DLIMP	ML/I	ASSEMBLER	PDP-7, IBM 360 u.a.	BROWN	University of Kent at Canterbury, Großbritannien	1969
Experimentelle Simulationssprache	Syntaxmakro-translator	Simulations-sprache	CDC u.a.	LEAVENWORTH	International Business Machines Corporation, New York, USA	1966
FLAN	MAKFOR	FORTRAN	BESM6, ESER	ZAUPJE, MENZ	Rechenzentrum der Moskauer Universität, UdSSR	1974
FORMAL	PL/1-Programm	FORTRAN	FACOM 230-75	NAGATA	Hokkaido Universität, Japan	1979
FORTRAN, algorithmic	SAP-Assembler	ASSEMBLER	IBM 704	MCILROY	Bell Telephone Laboratories, New Jersey, USA	1959
GPM-Sprache	GPM	wählbar	TITAN	STRACHEY	University Mathematical Laboratory, Cambridge, Großbritannien	1965
GPSG-Sprache	GPSG	wählbar	ICL	HAAKE	Siemens-Rechenzentrum, BRD	1978
GRASPL	MAKROPL	PL/1	ESER	NIKITIN, SKRIGAN	AdW der Belorussischen SSR, Minsk, UdSSR	1979
IMP-Sprache	IMP	ASSEMBLER	IBM	LAMPSON	Cambridge, Massa-chusetts, USA	1965
JOBOL-Makro-sprache	JOBOL-Pre-prozessor	Jobsteuer-sprache	IBM 360	MOORE	Londoner Universität, Großbritannien	1977
KATE-Sprache	KATE	Kommandosprache	CDC 6600	TESKEY	University of Kent at Canterbury, Groß-britannien	1977
LIMP-Sprache	LIMP	wählbar	IBM 7040, ATLAS 2	WAITE	University of Colorado, USA	1966
LORD-Makro-sprache	LORD-Compiler	wählbar	ESER	SEBERJAKOV	Rechenzentrum der AdW der UdSSR, Kiew	1975
MACRO-ALGOL	erweiterter ALGOL-Compiler	ALGOL	IBM	LEROY	New York, USA	1966

Makrosprache	Makroprozessor	Basissprache	Computer	Autor(en)	Institution	Jahr
MACROBOL	MACRO-11	ASSEMBLER	PDP-11	HAMANN	Gesellschaft für Prozeß-steuerungs- und Infor-mationssysteme, Berlin (West)	1978
MACROPEARL	MACRO-11	ASSEMBLER	PDP-11	HAMANN	s.o., Berlin (West)	1978
Macro Set	Makroassembler der IBM 360	ASSEMBLER	IBM 360	GRIFFITHS, PELTIER	Institute de Mathe-matique Appliquees, Grenoble, Frankreich	1969
Macro Set	Makroassembler der IBM 7090	ASSEMBLER (IBM 7040)	IBM 7090	DELLERT	Research Analysis Corporation, Virginia, USA	1965
Macro Set	MACRO-FAP	ASSEMBLER	IBM 7094	FLETCHER	University of California, USA	1965
MACRO-SM-Sprache	MACRO-SM	ASSEMBLER	SM3, SM4	HORN, KOFER	Ingenieurhochschule Dresden, DDR	1979
MACRO SPITBOL	erweiterter Makroassembler	wählbar	PDP-11	DEWAR, MCCANN	New York, USA Leeds, Großbritannien	1977
MACRO-1520-Sprache	MACRO-1520	ASSEMBLER	K1520	HORN, UTKE	Ingenieurhochschule Dresden, DDR	1981
MACTRAN	MACTRAN-Prozessor	FORTRAN	IBM360/75	PESCHKE	Konstanz, BRD	1971
Makroassembler	Makroassembler	ASSEMBLER	K1510	HORN, DIENHOLD	Ingenieurhochschule Dresden, DDR	1979
Makroassembler	Makroassembler DOS	ASSEMBLER	ESER	Autorenkollektiv	VEB Kombinat Robotron, Dresden, DDR	1975
Makroassembler	Makroassembler OS	ASSEMBLER	ESER	Autorenkollektiv	VEB Kombinat Robotron, Dresden, DDR	1975
MAKROBOL	MAKROBOL-Prozessor	COBOL	ESER	BABENKO, SINJAKOVSKAJA	Institut für Kybernetik der AdW der UdSSR, Kiew	1976
Makros zur Nor-mierten Pro-grammierung	PL/1-Makro-interpreter	PL/1	ESER	GOLLA	VEB Datenverarbei-tungszentrum Suhl, DDR	1979

Makros zur SIM-DIS-Generierung	PL/1-Makro-interpreter	SIMDIS	ESER	DUMKE, LORENZ, STUCHLIK	Technische Hochschule Magdeburg, DDR	1978
Makros zur Struktogrammgenerierung	BESM6-Makro-prozessor	FORTRAN	BESM6	ZARODIN u.a.	Moskauer Universität, UdSSR	1982
Makros zur Strukturierten Programmierung	PL/1-Makro-interpreter	PL/1	ESER	HOHNDORF	VEB Kombinat Robotron, Dresden, DDR	1977
MALCROL 68	MACRO-11	ASSEMBLER	PDP-11	HAMANN	Gesellschaft für Prozeßsteuerungs- und Informationssysteme, Berlin (West)	1978
MALIS	PL/1-Makro-interpreter	wählbar	ESER	DUMKE	Technische Hochschule Magdeburg, DDR	1980
MAP	MAP-Prozessor	PASCAL	CDC u.a.	COMER	Purdue Universität, West Lafayette, USA	1979
MASPAL-11	MACRO-11	ASSEMBLER	PDP-11	SCHOLTES-PASSAN	Zürich, Schweiz	1974
MASSA	Makroassembler	ASSEMBLER	A5601.20	VETTEROZZI	VEB Kombinat Robotron, Erfurt, DDR	1984
MAX-Sprache	MAX	wählbar	ICL1900	NUDDS	Computing Laboratory of Bradford, USA	1977
MCOBOL	erweiterter COBOL-Compiler	COBOL	CDC 7600	TRIANCE, YOW	University of Manchester, Großbritannien	1980
MESS	ML/1	Compiler	PDP-11	FORGACS, BOS	Informatica of Science University of Nijinegen, Niederlande	1978
METACOBOL	METACOBOL-Generator	COBOL	IBM 360	Autorenkollektiv	Applied Data Research (ADR), Princeton, USA	1976
METAMACR	Konzept	wählbar	–	KAUFMAN	Moskauer Universität, UdSSR	1977
METAPLAN-Makrosprache	METAPLAN-Prozessor	ASSEMBLER	Spectra 70	FERGUSON	Programmatics Incorporation, Los Angeles, USA	1966
MICMAC-Sprache	MICMAC	ASSEMBLER	CONTROL DATA 3600	LETTELIER, WEBER	Societé d'Informatique Appliquee, Paris, Frankreich	1965
MIPS-Sprache	MIPS	ASSEMBLER	CDC u.a.	LEININGER, WERNER	Sperry Corporation, USA	1984

Makrosprache	Makroprozessor	Basissprache	Computer	Autor(en)	Institution	Jahr
MISEPP	MISEPP-Prozessor	Makroassembler	HITACM-170	HAYASHI	Kobe Universität, Japan	1983
ML1/E-Sprache	ML1/E	ASSEMBLER	IBM 360	GOEKE, KRÄMER	Technische Hochschule Aachen, BRD	1985
ML/I	ML/I-Prozessor	wählbar	PDP, ICL, ATLAS 2	BROWN	University of Kent at Canterbury, Großbritannien	1966
MOL	SIMCMP	wählbar	ZAM-41, ODRA-1204	JAROSINSKA	WAT, Warschau, VR Polen	1972
MP/2-Sprache	MP/2	wählbar	ICL1900	MANDIL	University of Belfast, Nordirland	1972
MPL/1	PL/1-Makro-interpreter	PL/1	ESER	GORBATENKO	Moskauer Universität, UdSSR	1978
MXEC-Sprache	MXEC	Kommando-sprache	DEC TOPS20	ASH	Bolt, Barn. Incorporation, Cambridge, USA	1981
ONYX	PL/1-Makro-interpreter	PL/1	ESER	KRAUSE	Martin-Luther-Universität Halle, DDR	1982
PL/1-MAKRO	PL/1-Makro-interpreter	PL/1	ESER	KRAUSE	Martin-Luther-Universität, Halle, DDR	1978
PM-Sprache	PM	FORTRAN	Mikrocomputer	SASSA	Tokyo Institute of Technology, Tokio, Japan	1979
POLYP	POLYP-Prozessor	ASSEMBLER	IBM 370	MUSSTOPF	IBM-Rechenzentrum, BRD	1972
POP-2-Sprache	POP-2	wählbar	ICL	MIDDLETON	University of Swansea, Großbritannien	1976
PROCEDURE MACRO	modifizierter GPP	GPP-ASSEMBLER	IBM2250, GPP360D	JONES	Systems International Limited, Keyworth, Großbritannien	1971
Psil-Sprache	Psil	ASSEMBLER	CDC	CAYROL	Université Paul Sabatier, Toulouse, Frankreich	1985
SAL	ML/I	Compiler	PDP-11	TANENBAUM	Vrije Universitaet Amsterdam, Niederlande	1976
SEL	SEL-Prozessor	ASSEMBLER	Hewlett Packard	MOLNAR	Institution di elaborazione della informazione, Pisa, Italien	1971

SIL	SIL-Prozessor	ASSEMBLER	IBM 360, CDC 6000	GRISWOLD	University of Arizona, USA	1972
SIMCMP-Sprache	SIMCMP	ASSEMBLER	IBM 7044, CDC 6400	ORGASS, WAITE	University of Colorado, USA	1969
SIMDIS-2-Makrosprache	SIMDIS-2-Compiler	SIMDIS	ESER	Autorenkollektiv	VEB Kombinat Robotron, Dresden, DDR	1976
SIXTRAN	SIXTRAN-Prozessor	FORTRAN	IBM 360/70	BURKHARDT	Universität Stuttgart, BRD	1975
SKODA	Makroassembler	ASSEMBLER	ESER	PFEIFFER	Humboldt-Universität Berlin, DDR	1982
SL/1	SL/1-Prozessor	ASSEMBLER	ISKRA 2000	EXEL, POPOVIČ	Universität Lubljana, Jugoslawien	1982
SMACRO-Sprache	Konzept	wählbar	–	CHEATHAM	IBM New York, USA	1966
SMAL	SMAL-Prozessor	ASSEMBLER	Mikrocomputer	JONES	University of Iowa, USA	1982
SMAPS	Makroassembler OS	ASSEMBLER	ESER	STRELLER	(Hersteller: AdW der VR Polen)	1976
SMOK	SMOK-Prozessor	PL/1-Makrosprache	ESER	KARELJOV, POLJAKOV	Moskauer Universität, UdSSR	1983
SNAP-Sprache	SNAP	Compiler	Atlas	NAPPER	University of Manchester, Großbritannien	1967
SP/1-Sprache	SP/1	wählbar	ICL 1907	MACLOED	Queens University, Kingston, USA	1969
STAGE2-Sprache	STAGE2	wählbar	CDC6400	WAITE	University of Colorado, USA	1970
STEP-Sprache	STEP	wählbar	IBM u.a.	SIMPSON	Stanford Linear Accelerator Center, Kalifornien, USA	1979
STRMACS	Makroassembler	ASSEMBLER	IBM 360	WRANDLE	Groldard Space Flight Center, Maryland, USA	1974
Structured Assembly Language	Makroassembler	ASSEMBLER	Mikrocomputer	REUSS	Cambridge, Massachusetts, USA	1982
STRUKTUR-MAC	MACRO-SM	ASSEMBLER	SM4	LANGENBERGER	LFA Berlin, DDR	1984
SUPERMAC-BCPL	SUPERMAC	BCPL	PDP-11	BROWN	University of Kent at Canterbury, Großbritannien	1980

Makrosprache	Makroprozessor	Basissprache	Computer	Autor(en)	Institution	Jahr
SUPERMAC-PASCAL	SUPERMAC	PASCAL	PDP-11	BROWN, OGDEN	University of Kent at Canterbury, Großbritannien	1981
SUPERMAC-PL1	SUPERMAC	wählbar	ESER	DUMKE	Technische Hochschule Magdeburg, DDR	1986
SYGMA-Sprache	SYGMA	ASSEMBLER	ICL	ERSHOV, RAR	International Business Machine Corporation, New York, USA	1966
Syntactic macros	Preprozessor	Compiler	PDP	CAMPBELL	University of St. Andrew, Großbritannien	1978
TILT	STAGE2	Kommandosprache	PDP-11	ELLIS	Washington University, St. Louis, USA	1978
TOSI	SDMP	wählbar	A-9132, A-7447	ABRAMSON	University of Vancouver, Kanada	1977
TRAC-Sprache	TRAC	wählbar	PDP-5, SDS930	MOOERS	Cambridge, Massachusetts, USA	1966
VMSIMAC-Sprache	VMSIMAC	SIMDIS	ESER	DILS	AdW der DDR, Berlin, DDR	1978
WISP-Sprache	WISP	wählbar	ICL	WILKES	University Mathematical Laboratory, Cambridge, Großbritannien	1964
XPOP	MACRO-FAP	ASSEMBLER	IBM7000	HALPERN	International Business Machines Corporation, San Jose, USA	1964
ZEUSS-SM	MACRO-SM	ASSEMBLER	TPA-Computer	WINDE	AdW der DDR, Berlin, DDR	1976
803-Sprache	803-Makrogenerator	ASSEMBLER	ELLIOT 803	ELCOCK, WILSON	University of Aberdeen, Großbritannien	1965
8088-Assembly Macros	8088-Assembler	ASSEMBLER	8088-Mikrocomputer	BURK	USA	1984

3. Verzeichnis einiger Begriffe zur Makrotechnik in deutsch, englisch, russisch und französisch

deutsch	englisch	russisch	französisch
allgemeiner Makroprozessor	general-purpose macro processor	макропроцессор общего назначения	macroprocesseur général
Basissprache	basic language	базовый язык	langage fondamental
Bedingungsmakro	condition macro, logic macro	логический макрос	macro conditional
Deklarationsmakro	declarativ macro	определительный макрос	macro déclaratif
geschlossenes Makro	closed macro	замкнутый макрос	macro fermé
Hilfsmakro	service macro, low-level macro	вспомогательный макрос	macro service
Makro	macro	макрос, макро	macro
Makroabarbeitung	macro action	макрообработка	macro-action
Makroanweisung	macro instruction, macro statement	макроинструкция	macro-instruction
Makroassembler	macro assembler	макроассемблер	macro-assembleur
Makroaufruf	macro call	макровызов	macro-appel
Makroausdruck	macro term	макрообозначение	macro-expression
Makrobibliothek	macro library	макробиблиотека	macro-bibliothèque
Makrodefinition	macro definition	макроопределение	macro-définition
Makroerweiterung	macro expansion	макрорасширение	macro-expansion
Makrogenerierung	macro generation	макродействие	macro-génération
Makrogrammatik	macro grammar	макрограмматика	macro-grammaire
Makrokommando	macro command	макрокоманда	macro-commandé
Makrokörper	macro body	макротело	corps de macro
Makromittel (-möglichkeit)	macro facility	макросредства	macro-possibilité
Makroparameter	macro parameter	макропараметер	macro-paramètre
Makroprogramm	macro program	макропрограмм	macroprogramme
Makroprogrammierung	macro programming	макропрограммирование	macro-programmation
Makroprozedur	macro procedure	макропроцедура	macro-procédure
Makroprozessor	macro processor	макропроцессор	macro-processeur
Makrosprache	macro language	макроязык	macro-langage
Makrosystem	macrosystem	система макрокоманд	macrosystème
Makrovariable	macro variable	макропеременные	macro-variable
Maschinencodemakro	computational macro	вычислительный макрос	macro comptual
offenes Makro	open macro	открытый макрос	macro ouvert
Spezialmakro	special-purpose macro	специальный макрос	macro spéciale

deutsch	englisch	russisch	französisch
spezieller Makroprozessor	special-purpose macro processor	спесиализированный макрогенератор	macro-processeur spéciale
Strukturmakro	structure macro	структурный макрос	macro structuric
Syntaxmakro	syntactic macro, informal macro	синтактический макрос	macro syntactic
Systemmakro	systemmacro	системный макрос	macro de système
Testmakro	debugging macro	отладочный макрос	macro de teste
Textmakro	text macro	текстовный макрос	macro textual
Verbindungsmakro	join macro	связывающий макрос	macro intégratif
verschachtelte Makrodefinition	nested macro definition, imbedded macro	вложенные определения макроса	macro-definition mode interfolie

4. Realisierung des SUPERMAC-Prinzips für die Programmiersprache PL/1

In 2.5.4. sind bereits implementierte SUPERMAC-Beispiele als Makroprozessoren mit einer speziellen Einbettungsform für die Programmiersprachen BCPL und PASCAL angegeben. Hier soll das Prinzip, welches von P. J. BROWN aufgestellt, erfolgreich realisiert und in [26], [28], [33] und [204] erläutert ist, an Hand einer Implementierung für die Programmiersprache PL/1 näher beschrieben werden.
Der allgemeine Ablauf ist in Bild 11 dargestellt. Der Makrointerpretation und Realisierung dienen zwei PL/1-Prozeduren, und zwar

@MACRO (*name,panz,pname*)	für die Zuordnung des Prozedurnamens *pname*, der den Namen der PL/1-Prozedur angibt, die bei Auftreten eines Makroaufrufs mit dem Namen *name* und der maximalen Parameteranzahl *panz* anzuwenden ist;
@MSCAN(*ein,aus*)	für die Realisierung der Makroabarbeitung, wobei *ein* die Eingabedatei und *aus* die Ausgabedatei für den generierten Text kennzeichnen.

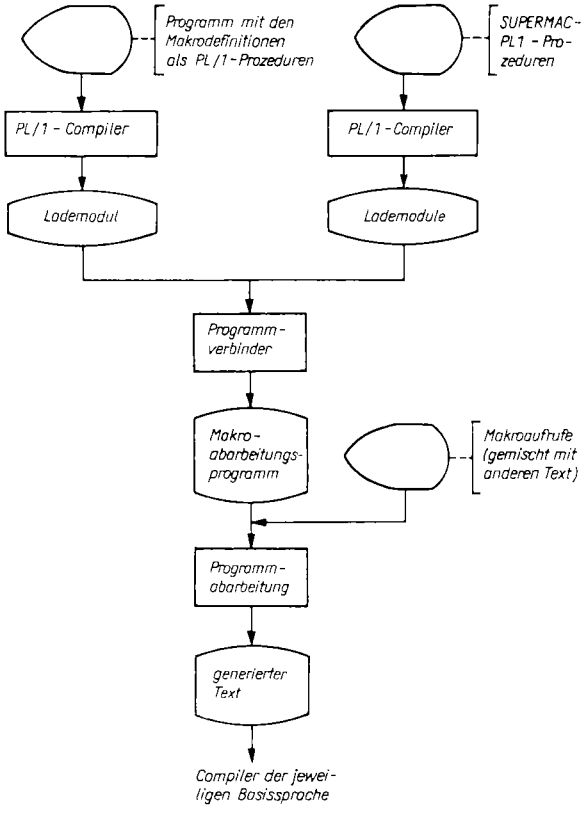


Bild 11. Allgemeiner Ablauf der Makroabarbeitung nach dem SUPERMAC-Prinzip für PL/1

Für das Interpretationsergebnis aus @MACRO und die nachfolgende Makrorealisierung wird eine Tabelle @MTAB (als externe Struktur) mit folgenden Spaltenangaben definiert:

@MNAME$_i$ – für die Einspeicherung des i-ten Makronamens durch @MACRO,

@FPARANZ$_i$ – für die Einspeicherung der Anzahl formaler Parameter der i-ten Makrodefinition durch @MACRO,

@APARANZ$_i$ – für die Einspeicherung der Anzahl aktueller Parameterwerte des Makroaufrufs zur i-ten Makrodefinition, die während der Makroabarbeitung durch @MSCAN bestimmt wird,

@PNAME$_i$ – für die Einspeicherung des Namens der anzuwendenden Prozedur für das i-te Makro durch @MACRO,

@ARGUM$_{i,j}$ – für die Einspeicherung des j-ten aktuellen Parameterwertes des Aufrufs des i-ten Makros während der Makroabarbeitung durch @MSCAN.

Als Funktionen (externe PL/1-Prozeduren), die in den Prozeduren für die unmittelbare Textgenerierung verwendet werden können, seien hier gegeben

@ARG(j) – zur Bereitstellung des j-ten aktuellen Parameterwertes des jeweils abgearbeiteten Makros,

@MNA – zur Bereitstellung der Anzahl aktueller Parameterwerte des jeweiligen Makroaufrufs,

@MSTR(*kette*) – für die Ausgabe des (generierten) Textes *kette* auf die durch den *aus*-Parameter von @MSCAN gekennzeichnete Datei,

@LIN(*kette*) – für das Auffüllen von *kette* auf 72 Zeichen,

@Z(*zahl*) – für die Transformation von *zahl* in eine Zeichenkette ohne vorangestellte Leerzeichen.

Für die im folgenden beschriebene Implementation dieser Funktionen (einschließlich @MACRO und @MSCAN) seien Einschränkungen in der Art gegeben, daß

(a) maximal 10 Makrodefinitionen für eine Makroabarbeitung zugelassen sind;

(b) die formale Parameteranzahl eines Makros 20 nicht übersteigt;

(c) die Länge eines aktuellen Parameterwertes 20 Zeichen nicht überschreitet;

(d) sich die Ein- und Ausgabedatei ergibt durch *ein* und *aus* (als @MSCAN-Parameter) als Index einer vorgegebenen Dateinamenmenge, die hierbei laute

ein	Dateiname
1	SYSIN
2	EDAT
3	EIN

aus	Dateiname
1	SYSPRINT
2	ADAT
3	AUS

Verwendet man für den Index als Kennzeichnung der jeweiligen Makrodefinition bei ihrer Interpretation (also bei @MACRO) die globale Variable @AZ und bei der Abarbeitung (also in @MSCAN) die globale Variable @IND, so haben die SUPERMAC-PL1-Prozeduren den folgenden Aufbau.

für @MACRO:

```
@MACRO:PROC(NAME,PNAME);
        DCL NAME CHAR(8)VAR,(PANZ,I,J)FIXED(2),PNAME
            ENTRY, (INDEX,SUBSTR)BUILTIN;
        externaldefinition
                              @MNAME( @AZ)=NAME;
                              @FPARANZ( @AZ)=PANZ;
                              @PNAME( @AZ)=PNAME;
                              @AZ= @AZ+1;
        RETURN;
    END @MACRO;
```

für @MSCAN:

```
@MSCAN:PROC(EIN,AUS);     externaldefinition
        DCL E RECORD,A RECORD OUTPUT,ESATZ CHAR(80),
        (INDEX,LENGTH,SUBSTR)BUILTIN,(I,J,K,L,M,N,EIN,AUS)
        FIXED(2),DE(3)CHAR(8)VAR  INIT('SYSIN','EDAT','EIN'),
        DA(3)  CHAR(8)VAR  INIT('SYSPRINT','ADAT','AUS');
        OPEN FILE(E)TITLE(DE(EIN));
        OPEN FILE(A)TITLE(DA(AUS));
        ON ENDFILE(E) GOTO ME;
        DO I= @AZ TO 10;
            @MNAME(I)=' @ @ @';
        END;
    M1:READ FILE(E)INTO(ESATZ);
        @IND=SEARCH;
        IF  @IND¬=0 THEN DO;  @APARANZ( @IND)=
                                @FPARANZ( @IND);
                                DO J=1 TO  @FPARANZ
                                ( @IND);
                                @ARGUM( @IND,J)=APW;
                                IF  @ APARANZ( @IND)<
                                @FPARANZ( @IND)
                                    THEN GOTO MC;
                                END;
                            MC: CALL  @PNAME( @IND);
                        ELSE WRITE FILE(A)FROM(ESATZ);
      GOTO M1; ME:;
```

```
SEARCH:PROC RETURNS(FIXED(2));
        /*** BESTIMMUNG DES MAKRODEFINITIONS-
        INDEXES                                          ***/
        I=1;

        DO WHILE( @MNAME(I) ¬ =' @ @ @');
             IF INDEX(ESATZ, @MNAME(I))¬=0
                THEN RETURN(I);
             I=I+1;

        END;
        RETURN(0);
        END SEARCH;

   APW:PROC RETURNS(CHAR(20)VAR);
        /*** ERMITTLUNG DES AKTUELLEN
        PARAMETERWERTES                                  ***/
        IF J=1 THEN DO;K=INDEX(ESATZ, @MNAME( @IND));
                       K=INDEX(SUBSTR(ESATZ,K),'(')+K;
                  END;

        M=INDEX(SUBSTR(ESATZ,K),',');
        IF M=0 THEN DO; @APARANZ( @IND)=J
                        N=INDEX(SUBSTR(ESATZ,K),')');
                        RETURN(SUBSTR(ESATZ,K,N-1));
                    END;

        N=K;  K=K+M;
        RETURN(SUBSTR(ESATZ,N,M-1));
        END APW;
   END @MSCAN;
```

für @ARG:

```
@ARG:PROC(INDEX)RETURNS(CHAR(20)VAR);
        DCL INDEX FIXED(2);
        externaldefinition
        RETURN( @ARGUM( @IND,INDEX));
        END @ARG;
```

für @MNA:

```
@MNA:PROC RETURNS(FIXED(2));
        externaldefinition
        RETURN( @APARANZ( @IND));
        END @MNA;
```

für @MSTR:

```
@MSTR:PROC(KETTE);
      DCL KETTE CHAR(80)VAR,ASATZ CHAR(80);
      externaldefinition
      ASATZ=KETTE;
      WRITE FILE(A)FROM(ASATZ);
      RETURN;
      END @MSTR;
```

für @LIN:

```
@LIN:PROC(Z)RETURNS(CHAR(72));
      DCL ZK CHAR(72)VAR,LENGTH BUILTIN,LE CHAR(72)
      INIT(' '); externaldefinition
      RETURN(ZK||SUBSTR(LE,1,72-LENGTH(ZK)));
      END @LIN;
```

und für @Z:

```
@Z:PROC(ZAHL)RETURNS(CHAR(8)VAR);
      externaldefinition
      DCL ZAHL FIXED,KETTE CHAR(8)VAR,SUBSTR
      BUILTIN;
      KETTE=ZAHL;
      DO I=7 TO 3 BY -1;
          IF SUBSTR(KETTE,I,1)=' ' THEN
                RETURN(SUBSTR(KETTE,I+1,8-I));
      END;
      RETURN(SUBSTR(KETTE,3,6));
      END @Z;
```

wobei für *externaldefinition* jeweils einzusetzen ist

```
DCL 1 @MTAB(10)EXT,2 @MNAME CHAR(8)VAR,2 @FPARANZ
    FIXED(2), 2 @APARANZ FIXED(2), 2 @PNAME ENTRY,2
    @ARGUM(20) CHAR(20)VAR,( @AZ, @IND)EXT FIXED(2);
```

Nachdem diese externen Prozeduren jeweils separat mit dem PL/1-Optimierungscompiler übersetzt wurden und der erhaltene Lademodul dieselbe Bezeichnung wie die jeweilige Prozedur erhalten hat, kann eine Anwendung der folgenden Form realisiert werden. Der Makroaufruf lautet hierbei

```
COB(3,5,6,11,15,24,29,30,38,45,54,66,71,99,101,108).
```

Das Ergebnis kann wie folgt kurz charakterisiert werden:

(1) SUPERMAC-PL1 hat den Vorteil, daß die Prozeduren für die Textgenerierung alle Möglichkeiten von PL/1 verwenden können. So ist beispielsweise für die Zeichenkettenverarbeitung die Anwendung der PL/1-Funktionen REPEAT und VERIFY unmittelbar möglich und muß nicht erst (siehe Abschnitt 4.) implementiert werden.

(2) Natürlich erscheint es wenig sinnvoll, für eine Programmiersprache wie PL/1 einen Makroprozessor zu implementieren, der eigentlich als MACRO-PL1E bereits existiert. Bedenkt man aber, daß die Makroabarbeitung des Makros COB (siehe 4.5.) auf einer ESER-Anlage immerhin 13,8 CPU-Sekunden benötigte, während SUPERMAC-PL1 (s.o.) für die Realisierung nur 0,8 CPU-Sekunden erforderte, so erkennt man doch den praktischen Wert dieser (als Demonstrationsbeispiel zu verstehenden) SUPERMAC-PL1-Implementation.

(3) Die effektive Anwendung von SUPERMAC-PL1 erfordert natürlich gute PL/1-Kenntnisse.

Die Darstellung des SUPERMAC-Prinzips am Beispiel einer PL/1-Implementation kann auch Ausgangspunkt für die Anwendung auf weitere höhere Programmiersprachen sein.

```
COMPILER SOURCE LISTING                          MTEST:PROC OPTIONS(MAIN);
STMT NO. LEV NT     MTEST:PROC OPTIONS(MAIN);
       1       0          /*********************************************/
                          /*   TEST DES SUPERMAC-PRINZIPS IN PL/1      **/
                          /*********************************************/
                          /*   GLOBALE DEFINITIONEN, DIE IN JEDEM MAKRO-*/
                          /*   DEFINITIONSPROGRAMM ANZUGEBEN SIND       */
       2   1   0          DCL 1 @MATB(10)EXT,2 @MNAME CHAR(8)VAR,2 @FPARANZ
                             FIXED(2),2 @APARANZ FIXED(2),2 @PNAME ENTRY,
                             2 @ARGUM(20)CHAR(20)VAR,(@AZ,@IND)EXT FIXED
                             (2)INIT(1);
       3   1   0          DCL @MACRO ENTRY(CHAR(70)VAR,ENTRY);
       4   1   0          DCL @MSCAN ENTRY(FIXED(2),FIXED(2));
       5   1   0          DCL @ARG ENTRY(FIXED(2))RETURNS(CHAR(20)VAR);
       6   1   0          DCL @MSTR ENTRY(CHAR(80)VAR);
       7   1   0          DCL @Z ENTRY(FIXED)RETURNS(CHAR(8)VAR);
       8   1   0          DCL @LIN ENTRY(CHAR(71)VAR)RETURNS(CHAR(72));
       9   1   0          DCL @MNA ENTRY RETURNS(FIXED(2));
                          /*********************************************/
                          /*   AUFRUF DER INTERPRETATION DER MAKROAUF- */
                          /*   RUFDEFINITION                           */
                          /*********************************************/
      10   1   0          CALL MACRO('COB',16,COB);
                          /*********************************************/
                          /*   MAKRODEFINITION ALS PL/1-PROZEDUR       */
                          /*********************************************/
      11   1   0    COB:  PROC;
      12   2   0          DCL P CHAR(4)VAR,NR CHAR(8)VAR,(I,J)FIXED;
      13   2   0          TEXT='';
      14   2   0          J=@MNA;
      15   2   0          DO I=1 TO J;
      16   2   1           P=@ARG(I);   NR=@Z(I);
      18   2   1           CALL @MSTR(@LIN('      02 COLUMN '||P||' PIC 9(4)
                                  ||' SOURCE ELEMENT ( '||NR||' ).'));
      19   2   1          END;
      20   2   0          END COB;
                          /*********************************************/
                          /*   AUFRUF DER MAKROABARBEITUNG             */
                          /*********************************************/
      21   1   0          CALL @MSCAN(2,1);
      22   1   0          END MTEST;


       02 COLUMN 3 PIC 9(4) SOURCE ELEMENT ( 1 ).
       02 COLUMN 5 PIC 9(4) SOURCE ELEMENT ( 2 ).
       02 COLUMN 6 PIC 9(4) SOURCE ELEMENT ( 3 ).
       02 COLUMN 11 PIC 9(4) SOURCE ELEMENT ( 4 ).
       02 COLUMN 15 PIC 9(4) SOURCE ELEMENT ( 5 ).
       02 COLUMN 24 PIC 9(4) SOURCE ELEMENT ( 6 ).
       02 COLUMN 29 PIC 9(4) SOURCE ELEMENT ( 7 ).
       02 COLUMN 30 PIC 9(4) SOURCE ELEMENT ( 8 ).
       02 COLUMN 38 PIC 9(4) SOURCE ELEMENT ( 9 ).
       02 COLUMN 45 PIC 9(4) SOURCE ELEMENT ( 10 ).
       02 COLUMN 54 PIC 9(4) SOURCE ELEMENT ( 11 ).
       02 COLUMN 66 PIC 9(4) SOURCE ELEMENT ( 12 ).
       02 COLUMN 71 PIC 9(4) SOURCE ELEMENT ( 13 ).
       02 COLUMN 99 PIC 9(4) SOURCE ELEMENT ( 14 ).
       02 COLUMN 101 PIC 9(4) SOURCE ELEMENT ( 15 ).
       02 COLUMN 108 PIC 9(4) SOURCE ELEMENT ( 16 ).

* * * * *     END OF DATA SET     * * * * *
```

Literatur- und Quellenverzeichnis

Zu Abschnitt 1.

[1] AHO, A. V.; ULLMAN, J. D.: The Theory of Parsing, Translation and Compiling. – New Jersey: Prentice-Hall Inc., 1972

[2] BACHMANN, P.: Grundlagen der Compilertechnik. – Leipzig: Teubner Verlag, 1975

[3] GEWALD, K.; HAAKE, G.; PFADLER, W.: Software Engineering: Grundlagen und Technik rationeller Programmentwicklung. – München; Wien: Oldenbourg Verlag, 1977

[4] GLUSCHKOW, V. M.; [u.a.]: Algebra Sprachen Programmierung. – Berlin: Akademie-Verlag, 1980

[5] HESSE, W.: Methoden und Werkzeuge zur Software-Entwicklung: Einordnung und Überblick. – In: Informatik-Fachberichte 43. – Berlin (West); Heidelberg: Springer-Verlag, 1981

[6] KLINT, P.: Interpretation Techniques. – In: Software – Practice and Experience. – London [u.a.], 11 (1981) 9 – S. 963–973

Zu Abschnitt 2.

[7] ABRAMSON, H.; RUSHWORTH, T.; VENEMA, T.: TOSI: A Tree Oriented String Interpreter for the Design and Implementation of Semantics. – In: Software – Practice and Experience. – London [u.a.], 7 (1977) 6 – S. 667–670

[8] ARDEN, B. W.; GALLER, B. A.; GRAHAM, R. M.: The MAD Definition Facility. – In: Comm. of the ACM. – Baltimore, 12 (1969) 8 – S. 432–439

[9] ARUNZANJAN, O. B. [u.a.]: MAKFOR: jazyk opisanija makromodulej generatora programm (MAKFOR: Sprache zur Implementation von Makromodulen für einen Programmgenerator). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1974) 22 – S. 16–26

[10] ARUNZANJAN, O. B. [u.a.]: Pakety makromodulej čislennogo analiza (Ein Paket von Makromodulen zur numerischen Analyse). – In: Vyčislitel'nye metody i programmirovanie. – Moskau (1974) 22 – S. 48–60

[11] ARUNZANJAN, O. B. [u.a.]: Translator s MAKFOR na FLAN (Ein Übersetzer auf der Basis des FLAN-gestützten MAKFOR). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1974) 22 – S. 41–47

[12] ASH, W. L.: MXEC: Parallel Processing with an Advanced Macro Facility. – In: Comm. of the ACM. – Baltimore, 24 (1981) 8 – S. 502–509

[13] Autorenkollektiv: Cobol Manager Macro Writing Feature. – In: Introductory Guide, Cobra System & Programming Ltd. – Surrey, 1983

[14] Autorenkollektiv: Popis makrojazyka a makrokompilatoru IMP (Eine Makrosprachanwendung zum Macrocompiler IMP). – Kuřim, 1972

[15] Autorenkollektiv: PS SIMDIS-2: Sprachbeschreibung. – Dresden: VEB Kombinat Robotron ZFT, 1981

[16] BABENKO, L. P.; ZIMJAKOVSKAJA, V. V.: MAKROBOL – generator KOBOL-Programm (MAKROBOL – ein COBOL-Programmgenerator). – In: Kibernetika. – Kiew, (1976) 2 – S. 22–27

[17] BALZER, R. M.; FARBER, D. J.: APAREL – A Parse-Request Language. – In: Comm. of the ACM. – Baltimore, 12 (1969) 11 – S. 624–631

[18] BANAHAN, M.; RUTTER, A.: UNIX: Lernen, verstehen, anwenden. – München; Wien: Carl Hanser Verlag, 1984
[19] BECKER, D.; LÄUFER, J.: Makros für die Normierte Programmierung in Assembler und PL/1. – In: Rechentechnik Datenverarbeitung. – Berlin, 17 (1980) 3. Beiheft – S. 40–44
[20] BENNETT, R. K.; NEUMAN, H. D.: Extension of Existing Compilers by the Sophisticated Use of Macros. – In: Comm. of the ACM. – Baltimore, 7 (1964) 9 – S. 541–542
[21] BORMANN, J.; LOETZSCH, J.: Realisierung von Fachsprachen durch Programmtransformation. – In: WBZ für MKR der TU Dresden. – Dresden, (1974) 7 – S. 15–44
[22] BROWN, P. J.: A practical usage of outer and inner syntax. – In: The Computer Journal. – London, 16 (1973) 2 – S. 122–123
[23] BROWN, P. J.: Levels of Languages for Portable Software. – In: Comm. of the ACM. – Baltimore, 15 (1972) 12 – S. 1059–1065
[24] BROWN, P. J.: Macro Processors. – In: Software Portability. – Cambridge: University Press, 1977 – S. 89–98
[25] BROWN, P. J.: Macro Processors and Techniques for Portable Software. – London: John Wiley & Sons, 1974 (Moskau: Verlag Mir, 1977)
[26] BROWN, P. J.: Macros without tears. – In: Software – Practice and Experience. – London [u.a.], 9 (1979) 6 – S. 433–437
[27] BROWN, P. J.: SCAN: A simple conversational programming language for text analysis. – In: Computers and Humanities. – London, 6 (1972) 4 – S. 223–227
[28] BROWN, P. J.: SUPERMAC: A macro facility that can be added to existing compilers. – In: Software – Practice and Experience. – London [u.a.], 10 (1980) 6 – S. 431–434
[29] BROWN, P. J.: Survey of Macro Processors. – Oxford [u.a.]: Pergamon Press, 1971 (Moskau: Verlag Statistika, 1975)
[30] BROWN, P. J.: The ML/I Macro Processor. – In: Comm. of the ACM. – Baltimore, 10 (1967) 10 – S. 618–623
[31] BROWN, P. J.: UNRAVEL: a programming language to put intelligence into dumps. – In: The Computer Journal. – London, 16 (1973) 1 – S. 10–12
[32] BROWN, P. J.: Using a Macro Processor to Aid Software Implementation. – In: The Computer Journal. – London, 12 (1969) 4 – S. 327–331
[33] BROWN, P. J.; OGDEN, J. A.: The SUPERMAC macro processor in Pascal. – In: Software – Practice and Experience. – London [u.a.], 13 (1983) 4, – S. 295–304
[34] BURKHARDT, W. H.: SIXTRAN: Ein erweiterungsfähiges Programmiersystem mit strukturierter FORTRAN-Sprache. – In: Elektronische Rechenanlagen und Computer Praxis. – München, 17 (1975) 5 – S. 236–240
[35] CAMPBELL, W. R.: A Compiler-Definition Facility Based on the Syntactic Macro. – In: The Computer Journal. – London, 21 (1978) 1, – S. 35–41
[36] CAMPBELL-KELLY, M.: An introduction into macros. – New York: Macdonald London and American Elsevier Inc., 1973 (Moskau: Verlag Mir, 1978)
[37] CAYROL, M.: Psil: manipulation d'objects infinis pour l'intelligence artificielle. – In: Technique et Science Informatiques. – Paris, 4 (1985) 4 – S. 373–381
[38] CHEATHAM, T. E.: The Introduction of Definitional Facilities into Higher Level Programming Languages. – In: AFIPS Conference Proceedings. – New York, 29 (1966) – S. 623–637
[39] COMER, D.: MAP: A Pascal Macro Preprocessor for Large Program Development. – In: Software – Practice and Experience. – London [u.a.], 9 (1979) – S. 203–209
[40] COWELL, W. R.: The Toolpack/IST Programming Environment. – In: IEEE 83-CH1919-0, Conference of Software Engineering. – USA, 1983 – S. 326–333
[41] CURLEY, R. F.: Getting started with MACRO-11. – In: The DEC Professional. – New York, 3 (1984) 2 – S. 134–144
[42] DEWAR, R. B.; MCCANN, A. P.: MACRO SPITBOL – A SNOBOL4 Compiler. – In: Software – Practice and Experience. – London [u.a.], 7 (1977) 1 – S. 95–113
[43] DILS, G.; DÖRNBERG, V.: Ein erweitertes Makrokonzept für PS SIMDIS. – In: Rechentechnik Datenverarbeitung. – Berlin, 15 (1978) 3. Beiheft – S. 29–32

[44] DJORDJEVIČ, J.; BARBACI, M.; HOSLER, B.: A PMS Level Notation for the Description and Simulation of Digital Systems. – In: The Computer Journal. – London, 28 (1985) 4 – S. 357–365
[45] DONOVAN, J.: System Programming. – New York: McGraw-Hill Book Company, 1972
[46] DUMKE, R.: Beiträge zur Rationalisierung der Datenverarbeitungsprojektierung für das Leitungs- und Informationssystem des Hochschulwesens. – Magdeburg, Dissertation A, 1980
[47] DUMKE, R.: Bibliographie zur Makrotechnik. – Technische Hochschule »Otto von Guericke« Magdeburg, 1978
[48] DUMKE, R.; LORENZ, P.; STUCHLIK, F.: Modifikation und Generierung von Simulationsprogrammen. – In: Rechentechnik Datenverarbeitung. – Berlin, 15 (1978) 3. Beiheft – S. 24–29
[49] DUSKE, J.; PARCHMANN, R.; SEDELLO, M.; SPECHT, J.: IO-Macrolanguages and Attributed Translation. – In: Information and Control. – New York; London, 35 (1977) 2 – S. 87–105
[50] EINERT, E.: Die Programmiersprache C. – In: Rechentechnik Datenverarbeitung. – Berlin, 21 (1984) 9 – S. 10–13
[51] ELLIS, R. A.: On the interactive use of a macroprocessor to generate operating system batch streams. – In: IEEE Transactions on Software Engineering. – New York, 4 (1978) 2 – S. 146–148
[52] ENGELFRIET, J.: Macro grammars, Lindemayer systems and other copying devices. – In: Automata, Languages and Programming (Lecture Notes on Computer Science 52). – Berlin (West) [u.a.]: Springer-Verlag, 1977
[53] ENGELFRIET, J.: Simple Program Schemes and Formal Languages. – In: Lecture Notes on Computer Science 20. – Berlin (West) [u.a.]: Springer-Verlag, 1974
[54] ENGELFRIET, J.; SCHMIDT, E. M.; LEEUWEN, J. V.: Stack machines and classes of nonnesting macro languages. – In: Journal of the ACM. – London, 27 (1980) 1 – S. 96–117
[55] EVANOV, V.: Generation of program packages with dynamic structure based on a macroprocessor. – In: Avtomatika. – Moskau, (1982) 4 – S. 80–87
[56] EVANS, M.: Software Engineering for the COBOL Environment. – In: Comm. of the ACM. – Baltimore, 25 (1982) 12 – S. 874–882
[57] EXEL, M.; POPOVIČ, B. T.; PROJATELJ, F.: SL1 Language: A Specification and Design Tool for Switching Systems Software Development. – In: IEEE Transactions Commun. – New York, 30 (1982) 6 – S. 1356–1362
[58] FEELEY, P. M.: Messages Stored in Macro. – In: IBM Technical Disclosure Bulletin. – Armonk, 27 (1984) 2 – S. 1321–1323
[59] FITZWATER, D. R.: A storage allocation and references structure. – In: Comm. of the ACM. – Baltimore, 7 (1964) 9 – S. 542–549
[60] FLETCHER, J. G.: A Program to Solve the Pentomino Problem by the Recursive Use of Macros. – In: Comm. of the ACM. – Baltimore, 8 (1965) 10 – S. 621–623
[61] FLORES, I.: Computer Software. – New Jersey: Prentice-Hall Inc., 1965
[62] FORGACS, T.; BOS, J. VAN DEN: MESS: A Macro Language for Structured Systems Programming. – In: Angewandte Informatik. – Braunschweig, 20 (1978) 1 – S. 25–32
[63] FURMAN, M. E.: Makrooperacii struktorirovanija programm v OS ES (Makros für die Programmstrukturierung im OS/ES). – In: Programmirovanie. – Moskau, 8 (1982) 4 – S. 25–31
[64] GARWICK, J. V.: GPL: a truly general purpose language. – In: Comm. of the ACM. – Baltimore, 11 (1968) 9 – S. 634–638
[65] GAUTHIER, J. S.: Macros in Simulation. – Concord: Mitchell & Gauthier Assoc. Inc., 1979
[66] GOEKE, L.; KRÄMER, R.: Einsatzmöglichkeiten des Makroprozessors ML1/E zur Generierung von Sprachinterpretern. – In: Angewandte Informatik. – Braunschweig, 27 (1985) 1 – S. 29–34
[67] GRIFFITHS, M.; PELTIER, M.: A macro-generable language for the 360 computers. – In: The Computer Bulletin. – London, 13 (1969) 11 – S. 389–390
[68] GRISWOLD, R. E.: The Macro Implementation of SNOBOL4. – In: Brown, P. J.: Software Portability. – Cambridge: University Press, 1977 – S. 196–210
[69] GROSCH, J.: Portierung eines Codegenerators mit der Makrosprache STAGE2. – In: Schneider: Portable Software. – Stuttgart, 1980 – S. 163–174
[70] HAAKE, G.: Software Engineering: Ein Satz von Werkzeugen zur Unterstützung der ingenieurmäßigen Entwicklung von Software. – In: Data report. – Miesbach, 13 (1978) 6 – S. 17–22

[71] HALPERN, M. I.: Toward a general processor for programming languages. – In: Comm. of the ACM. – Baltimore, 11 (1968) 1 – S. 15–25
[72] HAMANN, K.: MALCROL 68: A Macro Language for Structured Systems Programming. – In: Angewandte Informatik. – Braunschweig, 20 (1978) 12 – S. 524–525
[73] HANISCH, C.: Makros oder Unterprogramme in der Anwendungsprogrammierung?. – In: Rechentechnik Datenverarbeitung. – Berlin, 14 (1977) 9 – S. 22–24
[74] HAYASHI, T.: A Program Structuring Preprocessor for a Macro Assembly Language. – In: Software – Practice and Experience. – London [u.a.], 13 (1983) 6 – S. 487–494
[75] HIGMAN, B.: Programmiersprachen: Eine vergleichende Studie. – Leipzig: Teubner Verlag, 1971
[76] HILDEBRAND, H.; REIMANN, W.: Programmdokumentation SPAS (Simulations-Programmsystem für analoge Systeme). – Berlin, 1980
[77] IRONS, E. T.: Experience with an extensible language. – In: Comm. of the ACM. – Baltimore, 13 (1970) 1 – S. 31–40
[78] JAROSINSKA, F.: Makrogenerativnij jazyk MOL: primenenie i realisacija (Die Makrosprache MOL: Entwurf und Implementation). – In: WBZ für MKR der TU Dresden. – Dresden, (1974) 8 – S. 142–151
[79] JONES, D. W.: Machine Independent SMAL: A Symbolic Macro Assembly Language. – In: Technical Report 82–03 of the University of Iowa. – Iowa, 1982
[80] JONES, D. W.: Improved Interpretation of UNIX-like File Names Embedded in Data. – In: Comm. of the ACM. – Baltimore, 27 (1984) 7 – S. 782–784
[81] JONES, W. R.: Macro Facility for Interactive Display. – In: Software – Practice and Experience. – London [u.a.], 1 (1971) 2 – S. 159–166
[82] KARELEV, S. V.; POLJAKOV, A. K.: SMOK: univerzal'nyj makrogenerator sintaksičeskogo tipa dlja razrabotki jazykovych konvertov (SMOK: ein syntaxgesteuerter universaler Makrogenerator zur Sprachkonvertierung). – In: Programmirovanie. – Moskau, 9 (1983) 3 – S. 25–32
[83] KAUFMAN, V. S.: On the macro extension of programming languages. – In: Lecture Notes on Computer Science 47. – Berlin (West): Springer-Verlag, 1977 – S. 301–313
[84] KERNIGHAN, B. W.; RITCHIE, D. M.: Programmieren in C. – München; Wien: Carl Hanser Verlag, 1983
[85] KOMARNICKI, O.: Makrotechnik und Makros: ein Mittel zur effizienten Programmerstellung. – In: Heilmann: 2. Jahrbuch der EDV-Akademie. – Wiesbaden, 1973 – S. 232–250
[86] KUČEVSKI, J. V.: Makroprocessor generatora programm (Ein Makroprozessor zur Programmgenerierung). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1974) 22 – S. 48–60
[87] LANGENBERGER; Ansorge: Strukturierte Programmierung für SKR. – In: Rechentechnik Datenverarbeitung. – Berlin, 21 (1984) 3 – S. 38
[88] LAYZELL, P. J.: The History of Macro Processors in Programming Language Extensibility. – In: The Computer Journal. – London, 28 (1985) 1 – S. 29–33
[89] LEAVENWORTH, B. M.: Syntax macros and extended translation. – In: Comm. of the ACM. – Baltimore, 9 (1966) 11 – S. 790–793
[90] LEVIN, D. J.: Jazyk svenchvysokogo urovnja SETL i ego realizacija (dlja EVM BESM-6) (Die high-level Programmiersprache SETL und ihre Implementation auf der BESM-6). – Novosibirsk: Verlag Nauka, 1983
[91] MACLOED, I. A.: MP/1: A FORTRAN Macroprocessor. – In: The Computer Journal. – London, 14 (1971) 3 – S. 229–231
[92] MACLOED, I. A.: SP/1: A FORTRAN Integrated String Processor. – In: The Computer Journal. – London, 13 (1970) 3 – S. 225–260
[93] MANDIL, S. H.: A general-purpose »problem-to-program« translator. – In: The Computer Bulletin. – London, 16 (1972) 10 – S. 492–496
[94] MCILROY, M. D.: Macro Instruction Extension of Compiler Languages. – In: Comm. of the ACM. – Baltimore, 3 (1960) 4 – S. 214–220
[95] MIDDLETON, A. G.: On the use of macros for iteration. – In: The Computer Journal. – London, 19 (1976) 2 – S. 170–172
[96] MOLNAR, G.: SEL: A self-extensible programming language. – In: The Computer Journal. – London, 14 (1971) 3 – S. 238–242

[97] MOOERS, C. N.: TRAC: A procedure-describing language for the reactive typewriter. – In: Comm. of the ACM. – Baltimore, 9 (1966) 3 – S. 215–219
[98] MOORE, L.: Design for a transportable job organisation language. – In: The Computer Journal. – London, 22 (1979) 4 – S. 296–302
[99] MÜLLER, B.; TSCHIRSCH, G.: Macroprocessing auf BESM6 und ESER für höhere Programmiersprachen. – Berlin: ZFR-Information der AdW der DDR, 1981
[100] MUSSTOPF, G.: Das Programmiersystem POLYP. – In: Angewandte Informatik. – Braunschweig, 14 (1972) 10 – S. 441–448
[101] MUSSTOPF, G.: Sistema programmirovanie POLYP (Das Programmiersystem POLYP). – In: Programmirovanie. – Moskau, 5 (1979) 1 – S. 43–54
[102] NAGATA, H.: FORMAL: A Language with a Macro-oriented Extension Facility. – In: Computer Languages. – London, 5 (1980) – S. 65–76
[103] NAPPER, R. B.: Some proposals for SNAP: A language with formal macro facilities. – In: The Computer Journal. – London, 10 (1967) 3 – S. 231–243
[104] NAPPER, R. B.; FISCHER, R. N.: ALEC: A User Extensible Scientific Programming Language. – In: The Computer Journal. – London, 19 (1976) 1 – S. 25–31
[105] NAPPER, R. B.; FISCHER, R. N.: RCC: A user-extensible system implemented language. – In: The Computer Journal. – London, 23 (1980) 3 – S. 212–222
[106] NIBLETT, B.: Macro Search Techniques in the Integration of Legal Language. – In: Jurimetrics Journal. – Spring, 16 (1976) – S. 201–205
[107] NUDDS, D.: The design of the MAX macroprocessor. – In: The Computer Journal. – London, 20 (1977) 1 – S. 30–36
[108] ORGASS, R. J.; WAITE, W. M.: A base for a mobil programming system. – In: Comm. of the ACM. – Baltimore, 12 (1969) 9 – S. 507–510
[109] PESCHKE, J. VON: Erweiterung von Programmiersprachen durch ihre Benutzer. – In: Angewandte Informatik. – Braunschweig, 13 (1971) 7 – S. 331–340
[110] RAMMELT, K.: Die Sprache BLISS. – In: WBZ für MKR der TU Dresden. – Dresden, (1975) 12 – S. 26–47
[111] RAUPACH; TARTZ: Beschreibung der Makrotechnik für die BESM6. – In: Software-Information des BESM6-Rechenzentrums des IFE. – Berlin, (1972) 17
[112] REITBAUER, E.; ÖSTERLE, H.; GRIESE, J.: Ein System zur Unterstützung von interaktiven Programmen. – In: Angewandte Informatik. – Braunschweig, 15 (1973) 4 – S. 157–173
[113] REUSS, J. L.: Macro Implementation of a Structured Assembly Language. – In: IEEE Transactions on Software Engineering. – New York, 8 (1982) 1 – S. 30–46
[114] REVESZ, G.: A Note on Macro Generation. – In: Software – Practice and Experience. – London [u.a.], 15 (1985) 5 – S. 423–426
[115] RICE, J. R.; RIBBENS, C.; WARD, W. A.: Algorithm 622: A Simple Macroprocessor. – In: ACM Transactions on Mathematical Software. – New York, 10 (1984) 4 – S. 410–416
[116] ROLKE, C.: DATE.TEC: A little TECO macro. – In: The DEC Professional. – New York, 2 (1983) 6 – S. 45
[117] SAINI, J. S.; ZALUSKA, E. J.: Design tool for digital macro systems. – In: Software and Micros. – London, 3 (1984) 5–6 – S. 101–105
[118] SASSA, M.: A Pattern Matching Macro Processor. – In: Software – Practice and Experience. – London [u.a.], 9 (1979) – S. 439–456
[119] SASSA, M.: Macro Languages. – In: Information Processing Society. – Tokio, 22 (1981) 6 – S. 505–509
[120] SCHREITER, D.: Zu einigen Problemen der Entwicklung von Programmiersprachen. – In: Rechentechnik Datenverarbeitung. – Berlin, 11 (1974) 1. Beiheft – S. 2–6
[121] SEBERJAKOV, V. A.: Makrogeneracija v sisteme LORD (Makrogenerierung im Programmiersystem LORD). – In: Programmirovanie. – Moskau, 4 (1978) 1 – S. 28–30
[122] SILANDES, J. L.: ABSTRAC: communicaciones convencionales hasta X.25. – In: Proceso de datos. – Madrid, (1983) 133 – S. 34–46
[123] SIMPSON, J. W.: The STEP Processor. – Stanford: Report of the Stanford Linear Accelerator Center, 1977

[124] STORER, J. A.; SZYMANSKI, T. G.: Data compression via textual substitution. – In: Journal of the ACM. – London, 29 (1982) 4 – S. 928–951
[125] STRACHEY, C.: A general purpose macrogenerator. – In: The Computer Journal. – London, 8 (1965) 3 – S. 225–241
[126] TANENBAUM, A. S.: A general-purpose macro processor as a poor man's compiler-compiler. – In: IEEE Transactions on Software Engineering. – New York, 2 (1976) 2 – S. 121–125
[127] TANENBAUM, A. S.; STEVEREN, H. [u.a.]: A practical tool Kit for making portable compilers.– In: Comm. of the ACM. – Baltimore, 26 (1983) 9 – S. 654–660
[128] TANIKAWA, K.; TERAKADO, J.; ISODA, K.: The Design of I/O Macro for FORTRAN-like Formatting. – Ibaraki: Report of the Ibaraki University, 1978
[129] TESKEY, N.; KATE: A macro processor for extending command languages. – In: The Computer Journal. – London, 20 (1977) 2 – S. 187–189
[130] TRIANCE, J. M.: A Macro Facility for COBOL. – Venice: Proceedings of European Cooperation in Informatics, Oktober 1978
[131] TRIANCE, J. M.; LAYZELL, P. J.: A Language Enhancement Facility for COBOL: its Design and Implementation. – In: The Computer Journal. – London, 28 (1985) 2 – S. 128–133
[132] TRIANCE, J. M.; LAYZELL, P. J.: Macro Processors for Enhancing High-Level Languages: some Design Principles. – In: The Computer Journal. – London, 28 (1985) 1 – S. 34–43
[133] TRIANCE, J. M.; YOW, J. F.: MCOBOL: A Prototype Macro Facility for COBOL. – In: Comm. of the ACM. – Baltimore, 23 (1980) 8 – S. 432–439
[134] VOEVODIN, V. V.; GAJZARJAN, S. S.; KABANOV, M. I.: Avtomatizirovanija generasija programm (Automatisierte Programmgenerierung). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1974) 22
[135] VOJUZ, V. I.: Obespečenie univerzalizma v makrogeneratora obščego naznačenija ES EVM (Gewährleistung der Universalität bei allgemeinen Makrogeneratoren). – In: Programmirovanie. – Moskau, 5 (1979) 4 – S. 69–72
[136] VOJUZ, V. I.: Universal'nyj makrogenerator v DOS ES (Allgemeine Makrogeneratoren im DOS/ES). – In: Programmirovanie. – Moskau, 8 (1982) 6 – S. 85–89
[137] WAITE, W. M.: A language-independent macro processor. – In: Comm. of the ACM. – Baltimore, 10 (1967) 7 – S. 433–440
[138] WAITE, W. M.: The mobile programming system: STAGE2. – In: Comm. of the ACM. – Baltimore, 13 (1970) 7 – S. 415–421
[139] WINKELMANN, K.: Notes on a Large Analysis System for Heavy Ion Physics Experiments. – In: Angewandte Informatik. – Braunschweig, 25 (1983) 10 – S. 440–446
[140] ZARODIN, G. A.; KLOKOV, K.; SIDEL'NIKOV, V. I.: O dvumernom makrorasširenii jazykov vysokogo urovnja (Zu Makroabarbeitungsformen in höheren Programmiersprachen). – In: Programmirovanie. – Moskau, 8 (1982) 4 – S. 32–35
[141] ZAUPE, G.; MENZ, H.: Makroprocessor generator fortrannych programm dlja ES EVM (Ein Makroprozessor für die Generierung von FORTRAN-Programmen der ESER-Rechner). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1976) 25 – S. 172–180
[142] ZURJAEVA, I. V.: K voprosu ob otladke makromodulej (Zu Problemen der Makromodultestung). – In: Vyčislitel'nye metody i programmirovanie. – Moskau, (1976) 25 – S. 157–171

Zu Abschnitt 3.

[143] APTE, P. R.: Macros allow more conditional jumps. – In: Electronical Engineering. – London, 55 (1983) 682 – S. 36–37
[144] Autorenkollektiv: Assembler-Makrosystem BIDABS-M. – Berlin: Deutsche Staatsbibliothek, Rechenzentrum, 1982
[145] Autorenkollektiv: Assembler: Sprachbeschreibung. – Dresden: VEB Kombinat Robotron, 1975
[146] Autorenkollektiv: Die Makrosprache DPL/M. – Köln: MDS-Recorder 1, 1976
[147] Autorenkollektiv: Makros der Datenverwaltung: OS/ES. – Dresden: VEB Kombinat Robotron, 1975

[148] BLUDAU, I.; RIEDEL, R.: Ein System von Makros zur Formulargestaltung. – In: Rechentechnik Datenverarbeitung. – Berlin, 11 (1974) 1 – S. 32–36
[149] BOLE, L.: Metody postroenija kompilatorov (Methoden des Entwurfs von Compilern). – In: Genuys: Jazyk programmirovanie. – Moskau: Verlag Mir, 1972
[150] BURK, R.: 8088 Assembly Macros. – In: Programming Journal. – New York, 2 (1984) 4 – S. 21–23
[151] CALINGEART, I. P.: Assemblers, Compilers and Program Translation. – New York: Computer Science Press, 1979
[152] CASTANON, H. F.: Desarrollo de un sistema de macroinstrucciones en DOS (Entwicklung eines Makroanweisungssystems in DOS). – In: Control, Cibernetica y Automatizacion. – Havanna, 17 (1983) 3 – S. 3–7
[153] DELLERT, G. T.: Use of macros in translation of symbolic assembly language of one computer to another. – In: Comm. of the ACM. – Baltimore, 8 (1965) 12 – S. 742–748
[154] DIMES, F. N.: Using the AIM-65 as a Macro-Assembler. – In: Proceedings of the Rochester FORTH Applications Conference. – Rochester, (1983) – S. 117–123
[155] FURGESON, D. E.: Evolution of the Meta-Assembly Program. – In: Comm. of the ACM. – Baltimore, 8 (1965) 10 – S. 621–623
[156] GLINSKI, K.: Macroassembler for 8080 Microcomputers. – In: Pr. Przem. Institut Elektronika. – Warschau, 23 (1982) 82 – S. 48–51
[157] GUROVA, L. I.: Osnovy programmirovanii (Grundlagen der Programmierung). – Moskau: Verlag Finansi i statistika, 1983
[158] HAUPT, D.; KRETSCHMAR, R.: Makroprogrammierung DOS/ES. – Leipzig: Teubner Verlag, 1975
[159] HESSELBARTH: Mikro/Makrodekoder für CABS/MABS 1520. – In: Rechentechnik Datenverarbeitung. – Berlin, 20 (1983) 12 – S. 3
[160] HORN, T.; KOFER, C.: Handbuch der SKR-Programmierung. – In: Informationsverarbeitung im Hoch- und Fachschulwesen. – Berlin, 1 (1979) 1, Teile 1 und 2
[161] HORN, T.; UTKE, M.: Sprachbeschreibung der Makroassemblersprache MACRO 1520 für den Mikrorechner K 1520. – In: Informationsverarbeitung im Hoch- und Fachschulwesen. – Berlin, 2 (1981) 3
[162] LETTELIER, G.; WEBER, J.: Point de vue sur la programmation. – In: Revue de Française de Traitement de l'Information. – Paris, 8 (1965) 2 – S. 121–133
[163] MUČNIK, M. M.: Ob ispol'zovanii makrogeneratora obščego naznačenija dlja realizacii jazykov programmirovanija SVCM i mikroprocessorov (Die Anwendung allgemeiner Makrogeneratoren für die Implementation von Programmiersprachen für Mini- und Mikrorechner). – In: Programmirovanie. – Moskau, 5 (1979) 3 – S. 86–89
[164] PARK, T. C.; KORPMAN, R. A.: TMML: an assembler-level programming aid. – In: SIGSMALL Newsl. – New York, 8 (1982) 1 – S. 30–46
[165] PFEIFFER, P.: Makrosystem SKODA. – In: EDV-Aspekte. – Berlin, 1 (1982) 3 – S. 34–36
[166] SOBOTTA, H.: Makroassemblersprache. – Dresden: Ingenieurhochschule, 1978
[167] STRELLER, D.: SMAPS: Ein Makrosystem zur problembezogenen Assemblerprogrammierung in OS/ES. – Karl-Marx-Stadt: 3. Symposium zu Grundlagen und Anwendung der Informationsverarbeitung der Technischen Hochschule Karl-Marx-Stadt, 1982 – S. 67–77
[168] TAVENIER, K. R.; NOTREDAM, P. H.: Macro-based cross assembler. – In: IEEE Transactions on Software Engineering. – New York, 6 (1980) 4 – S. 334–340
[169] VOSTRIKOVA, Z. P.: Programmirovanie na jazyk assemblera ES EVM (Programmierung in der ESER-Assemblersprache). – Moskau: Verlag Nauka, 1985
[170] WAKERLY, J. F.: Microcomputer architecture and programming. – London: John Wiley & Sons, 1981 (Moskau: Verlag Mir, 1984)
[171] WINDE, M.: Strukturierte Programmierung mit dem Makroassembler MACRO-SM. – Karl-Marx-Stadt: 3. Symposium zu Grundlagen und Anwendung der Informationsverarbeitung der Technischen Hochschule Karl-Marx-Stadt, 1982 – S. 46–59
[172] WRANDLE BARTH, C.: STRCMACS: An extensive set of macros for structured programming in OS/360 assembly language. – Maryland: Goddard Space Flight Center, 1974

Zu Abschnitt 4.

[173] Autorenkollektiv: PL/1: Sprachbeschreibung OS/ES. – Dresden: VEB Kombinat Robotron, 1981
[174] Conrow, K.; Smith, R. G.: NEATER2: A PL/I source statement reformatter. – In: Comm. of the ACM. – Baltimore, 13 (1970) 11 – S. 669–675
[175] Dumke, R.: Aufbau einer Programmbank mit MALIS. – In: EDV-Aspekte. – Berlin, 1 (1982) 3 – S. 49–51
[176] Dumke, R.: DORIS: Dialogorientiertes Quelltextredigierungs- und Makroimplementierungssystem. – Technische Hochschule »Otto von Guericke« Magdeburg, Oktober 1979
[177] Dumke, R.: CMS (Cobol Macro Set): Anwendungsbeschreibung. – Technische Hochschule »Otto von Guericke« Magdeburg, 1984
[178] Dumke, R.: Effektive Nutzungsformen der PL/1-Makrotechnik bei der Realisierung moderner Programmierungsmethoden. – In: WBZ für MKR der TU Dresden. – Dresden (1984) 68 – S. 18–21
[179] Dumke, R.: MALIS (Makrosprache für die LIS-Datenverarbeitungsprojektierung): Sprachbeschreibung. – Technische Hochschule »Otto von Guericke« Magdeburg, 1980
[180] Dumke, R.: Über eine Methode zur Implementation einer Projektierungssprache. – In: WBZ für MKR der TU Dresden. – Dresden, (1979) 40 – S. 23–25
[181] Frolof, G.; Oljumin, V.: Praktičeskij kurs programmirovanija na jazyk PL-I (Praktischer Kurs zur PL/1-Programmierung). – Moskau: Verlag Nauka, 1983
[182] Golla, E.: Normierte Programmierung in PL/1/OS. – In: Rechentechnik Datenverarbeitung. – Berlin, 16 (1979) 4 – S. 29–30
[183] Gorbatenko, D. D.: O makrorealizacii sistem programmnogo modelirovanija (Zur Makrorealisierung eines programmgestützten Modellierungssystems). – In: Programmirovanie. – Moskau, 4 (1978) 9 – S. 59–63
[184] Gricaj, V. P.: O realizacii instrumentarija strukturnogo programmirovanii mul'tiprocessist (Zur Implementation von Hilfsmitteln für die strukturierte Programmierung im Multiprozessorbetrieb). – In: Kibernetika. – Kiew, 18 (1983) 3 – S. 118–123
[185] Grund, F.; Issel, W.: PL/I – Programmierung. – Berlin: Verlag der Wissenschaften, 1975
[186] Hohndorf, G.: Makrosystem zur Unterstützung der Strukturierten Programmierung in PL/1. – In: Rechentechnik Datenverarbeitung. – Berlin, 14 (1977) 9 – S. 12–13
[187] Hübel, S.: Ein Generator für Normierte PL/1-Programme (DOS/ES). – In: Rechentechnik Datenverarbeitung. – Berlin, 16 (1979) 3 – S. 28–30
[188] Hughes, I. K.; Michton, J. I.: A Structured Approach to Programming. – New Jersey: Prentice-Hall Inc., 1977 (Moskau: Verlag Mir, 1980)
[189] Krause, W.: Makrogenerator ONYX. – Martin-Luther-Universität Halle, Rechenzentrum, März 1982
[190] Krause, W.: PL/1-MAKRO: Konzeption zur Arbeit mit dem PL/1-Makrointerpreter. – Martin-Luther-Universität Halle, Rechenzentrum, Juni 1978
[191] Krüger, I.: PL/1-Makrointerpreter für den R 21. – In: Rechentechnik Datenverarbeitung. – Berlin, 15 (1978) 3 – S. 15–16
[192] Nikitin, A. S.; Skrigan, N. J.: Ierarchičeskie grafovie struktury i programmnye sredstva ich obrabotki (Hierarchisch strukturierte Graphen und Mittel zu ihrer programmgestützten Verarbeitung). – In: Kibernetika. – Kiew, 17 (1982) 2 – S. 31–34
[193] Semakii, I. S.: Makroprocessor na osnove preprocessora PL/1 (Ein Makroprozessor auf der Basis des PL/1-Vorübersetzers). – In: Programmirovanie. – Moskau, 10 (1984) 5 – S. 64–67

Weitere Literatur zur Makrotechnik und ihrer Anwendung

[194] ADR: Macro facility reference manual. – New Jersey: Report SM2G-01-00, 1977
[195] ADR: MetaCobol Concepts and Facilities. – New Jersey: ADR-Report, 1976
[196] Autorenkollektiv: Programmierhandbuch SYPS 4000. – Dresden: VEB Kombinat Robotron, 1974

[197] BANDY, T.: Spawing BASIC-PLUS-2 Tasks under RSX-11M. – In: The DEC Professional. – New York, 3 (1984) 1 – S. 68
[198] BARNETT, M.: Macro-directive approach to high speed computing. – Cambridge: Massechusetts Institute of Technology, 1959
[199] BELL, J. R.: Transformations: the extension facility PROTEUS. – In: Proceedings of the SIGPLAN Extensible Language Symposium, 1969
[200] BIRKNER, J.: Macro's for Programmable Logic. – In: Electronical Conventions. – New York, (1982) – S. 1–4, 21–22
[201] BISCHOFF, C.: Textverarbeitung mit TEPROS. – In: Rechentechnik Datenverarbeitung. – Berlin, 21 (1984) 7 – S. 22–23
[202] BOWLDEN, H. J.: Symbolic Macro Extension to ALGOL. – New York: Scientific Paper 65-104-COMP-P2, 1965
[203] BREITENECKER, F.: The Concept of Supermacros in Today's and Future Simulation Languages. – In: Mathematical and Computational Simulation. – Amsterdam, 25 (1983) 3 – S. 279–289
[204] BROWN, P. J.: Embedded Macro Processors. – In: The Computer Journal. – London, 27 (1984) 4 – S. 348–353
[205] BROWN, P. J.: Extending high-level languages by macros: a practical case evaluation. – In: Proceedings of Software 72 Conference. – London, (1972) – S. 95–99
[206] BROWN, P. J.: Macro Processors and their use in implementing software. – Cambridge: Ph. D. Thesis, University, 1968
[207] BROWN, P. J.: ML/I user's manual. – Cambridge: Mathematical Laboratory of the University, 1966
[208] BROWN, P. J.: The use of ML/I in implementing a machine-independent language in order to bootstrap itself from machine to machine. – Cambridge: Technical Memorandum Nr. 68/1 of the University, 1968
[209] CHARUSHISA, I.: Entwurf von Makrosprachen über Display zur Programmierung in der Sprache C (jap.). – In: BIT.-Tokio,14 (1982) 9 – S. 1106–1111
[210] COLE, A. J.: Macroprocessors. – Cambridge: University Press, 1976
[211] COLLIS, B.: Macro Speed 8080: Z80 Multiplication. – In: EDN. – New York, 28 (1983) 24 – S. 225
[212] CORNELL, D.: FORTH to PC-DOS Interface. – In: Dr. Dobb's Journal. – New York, 9 (1984) 1 – S. 44–59
[213] DAVIES, K. E.; PARADINE, C.: Extension to the macro language. – Hewsley: IBM-Report, 1969
[214] DICKMANN, B. N.: ETC: An extensible macro-based compiler. – In: Proceedings of the AFIPS. – New York, 38 (1971) 8
[215] EASTWOOD, D. E.; MCILROY, M. D.: Macro compiler modification of SAP. – New Jersey: Report of the Bell Telephone Laboratories Computation Center, 1959
[216] ELCOCK, E. W.; WILSON, D. M.: 803 Macrogenerator. – University of Aberdeen, November 1965
[217] EMMER, M. B.: Implementing SNOBOL4 for the 8086 Microcomputer Family. – In: SIGSMALL/PC Notes. – New York, 10 (1984) 4 – S. 12–20
[218] ERSHOV, A. P.; RAR, A. F.: SIGMA: A symbolic generator and macro-assembler. – In: Proceedings of the IFIP Working Conference on Symbol Manipulation Languages. – Pisa, 1968
[219] FISCHER, M. J.: Grammars with macro-like productions. – Cambridge: Ph. D. Thesis, Havard University, 1968
[220] FRANK, M.; LORENZ, P.: Simulation diskreter Prozesse. – Leipzig: Fachbuchverlag, 1979
[221] FREEMAN, D. N.: Macro Language Design for System/360. – In: IBM Systems Journal. – New York, 5 (1966) 2 – S. 63–77
[222] FUKAZAWA, Y.: Abstraction mechanisms supported by a macroprocessor. – In: Journal of Information Processing. – Tokio, 6 (1983) 2 – S. 59–65
[223] GAGI, B.; LADET, P.: Grafcat Synthesis Computer System and Procedure Description in FMS. – In: Proceedings of the IFIP Working Conference. – Amsterdam, (1984) – S. 205–213

[224] GALLER, B. A.; PERLIS, A. J.: A proposal for definitions in Algol. – In: Comm. of the ACM. – Baltimore, 10 (1967) 4 – S. 204–219
[225] GINTIS, H.: The portable PIDGIN-Z80 macro-assembly implementation. – In: Dr. Dobb's Journal. – New York, 7 (1982) 10 – S. 25–34
[226] GOVE, J. R.: FORTRAN: The forgotten language. – In: The DEC Professional. – New York, 3 (1984) 1 – S. 8–22
[227] GRANT, C. A.: Syntax translation with context macro or macros without arguments. – In: SIGPLAN Notices. – New York, 6 (1971) 12 – S. 48–50
[228] GRECO, F. D.: Using the C preprocessor. – In: Programming Journal. – New York, 3 (1985) 1 – S. 12–13
[229] GREENWALD, I. D.: Handling a technique for macro-instructions. – In: Comm. of the ACM. – Baltimore, 2 (1959) 11 – S. 21–22
[230] GREENWOOD, S. R.: MACRO: A programming language. – In: SIGPLAN Notices. – New York, 14 (1979) 12 – S. 80–91
[231] GRUND, F.: Prinzipien des Betriebssystems OS/ES. – Berlin: Verlag der Wissenschaften, 1981
[232] HAGERMAN, A.: Macro and the classroom. – In: The DEC Professional. – New York, 2 (1983) 3 – S. 127, 132
[233] HALPERN, M. I.: A manual of the XPOP programming system. – San Jose, International Business Machines Corporation, 1967
[234] HAMILTON, J. G. [u.a.]: Computer aided program production. – In: Proceedings of the Datafair Conference. – Nottingham, (1973) – S. 191–196
[235] HEPPKER, D.: DMSL: A descriptive macro simulation language. – In: Ann. Assoc. International Calculations. – New York, 17 (1975) 4
[236] HOPEWELL, B.: Extension to autocode using ML/I. – Cambridge: Dipl. Diss., University, 1967
[237] KEESE, W. M.: A note on automatic generation of documentation by macro-assemblers. – Washington: Memorandum TM-64-1031-1, 1964
[238] KENT, W.: Assembler-language macroprogramming: a tutorial oriented toward the IBM 360. – In: Computing Surveys. – London [u.a.], 1 (1969) 4 – S. 183–196
[239] KINSELLA, A.: Fitting files on floppies. – In: The DEC Professional. – New York, 3 (1984) 1 – S. 149–151
[240] KORF, R. E.: Macro-operators: a weak method for learning. – In: Artificielle Intelligence. – Amsterdam, 26 (1985) 1 – S. 35–77
[241] LAMPSON, B. W.: Interactive machine programming. – In: Proceedings of the AFIPS Fall Joint Computer Conference. – New York, 27 (1965)
[242] LAYZELL, P. J.; VAN DER LINDEN, P.: Implementing the Proposed 1981 Cobol Standard by a Cobol Language Enhancement Feature. – Manchester: Report 249, Computing Department, 1980
[243] LEDGARD, H.: ADA An Introduction – ADA Reference Manual. – Berlin (West): Springer-Verlag, 1980
[244] LEININGER, B. S.; WERNER, R. A.; TRITES, H. T.: On Performance Simulation at the Macro Instruction Level. – New York: IEEE 84-CH2029-7, 1984 – S. 676–678
[245] LEROY, H.: A proposal for macro-facilities in Algol. – In: Algol Bulletin. – New York, (1966) 22
[246] LINOFF, J.; NEIMAN, D.: A macro sub-program for determining time elapsed between two dates. – In: The DEC Professional. – New York, 2 (1983) 6 – S. 16, 18, 20, 22
[247] MAGNUSON, R. A.: Macro programming. – Bethesda: RCA Data System Center, Januar 1964
[248] MANDIL, S. H.: Macro processing for the man-machine interface. – Peterle: IBM Scientific Center, 1971
[249] MASTERSON, K. S.: Compilation for two computers using NELIAC. – In: Comm. of the ACM. – Baltimore, 3 (1960) 11 – S. 607–611
[250] MAURER, W. D.: The compiling macro assembler. – In: AFIPS Conference Proceedings. – New York, 34 (1969) – S. 89–93
[251] MCKINNON WOOD, T. R.: A multi-access implementation of an interpretativ text processing language. – In: Information Proceedings. – Amsterdam, 1 (1968) – S. 373–377

[252] METZGER, J. J.; DNIESTROWSKI, A.: PLATINE: A software engineering environment. – New York: Report IEEE CH1919-0/83, 1983 – S. 193–199
[253] METZNER, J. R.: A graded bibliography on macro and extensible languages. – In: SIGPLAN Notices. – New York, 14 (1979) 1 – S. 57–68
[254] MEUSE, S.: Demystifying the Macro. – In: Call-A.P.P.L.E. – New York, 7 (1984) 11 – S. 10–21
[255] MIYAMOTO, I.; YEH, R. T.: Some ideas on software productivity measures. – In: Proceedings of the International Conference on System Science. – Hawaii, 1 (1982) – S. 906–916
[256] MOOERS, C. N.: How some fundamental problems are treated in the design of the TRAC language. – In: Bobrow: Symbol Manipulation Languages and Techniques. – Amsterdam, 1968 – S. 178–190
[257] MOOERS, C. N.; DEUTSCH, L. P.: TRAC: A text handling language. – In: Proceedings of the 20th ACM national Conference. – New York, (1965) – S. 229–246
[258] MOORE, A.: DUMMY.MAC: or the use of macros within MACRO11. – In: VAX/RSTS Professional. – New York, 6 (1984) 6 – S. 40–52
[259] MORGAN, H.: UTS-I: A macro system for trafic network simulation. – In: AFIPS Conference Proceedings. – New Jersey, 36 (1969) – S. 217–222
[260] PELTIER, M.: The Macro System. – Grenoble, Institute de Mathematiques Appliquees, 1969
[261] POPOV, G.: Ispol'zovanie na makroprocessory za avtomatizacija na programmironeto (Anwendung eines Makroprozessors zur automatisierten Programmerstellung). – In: Avtomat. na proizvod. i upravlenie. – Sofia, (1979) 10 – S. 15–18
[262] RIVIERE, J.: Exercise d'assembleur et de macro-assembleur. – Montrouge: Verlag Dunod, 1982
[263] ROVNER, R. B.; ORWET, T. S.: Implementation of an interactive man-machine interface utilizing a programmable controller based macroprocessor. – In: IEEE Transactions Ind. Electron. – New York, 30 (1982) 4 – S. 323–325
[264] SAMSON, J.: Make good use of the magical macro. – In: Datalink. – London, 27 (1984) 2 – S. 12–13
[265] SCHOLTES-PASSAN, M. D.: MASPAL-11: Macros for the Advancement of the European Docusmeeting. – Zürich, Juli 1974
[266] SOBEL, M. E.: Techniques for overlay prevention. – In: IBM Technical Disclosure Bulletin. – New York, 26 (1984) 8 – S. 4164–4165
[267] SOREFF, J.: Macro Expansion in FORTH. – In: FORTH Dimension. – New York, 5 (1984) 5 – S. 9–10
[268] STANDISH, T. A.: PPL: An extensible language that failed. – In: SIGPLAN Notices. – New York, 6 (1971) 12 – S. 144–145
[269] SVEJGAARD, B.: IMP: A programming language with floating definitions. – Kopenhagener Universität, 1971
[270] TAYLOR, D.: Macro Generation in FORTH-83. – In: FORTH Dimension. – New York, 7 (1985) 1 – S. 27–28
[271] TOBORON, D. I.; AHENGUGOI, A. V.: Makrositema ES-77 (Das Makrosystem ES-77). – Kišinev, 1979
[272] TRIANCE, J. M.; LAYZELL, D. J.: CLEF: a COBOL language enhancement facility. – Manchester: Report Nr. 273, Universität, 1982
[273] VAN DER MAY, J. E.; VARNEY, R. C.; PATCHEN, R. E.: SYMPLE: A general syntax directed macro processor. – In: AFIPS Conference Proceedings. – New Jersey, 35 (1969) – S. 157–168
[274] VETTORAZZI, D.: Effektivere Programmentwicklung für Mikrorechnersystem K 1520. – In: Rechentechnik Datenverarbeitung. – Berlin, 21 (1984) 2 – S. 17–19
[275] WAITE, W. M.: Building a mobile programming system. – In: The Computer Journal. – London, 13 (1970) 1 – S. 28–31
[276] WANG, M. S.; COURTNEY, J. F.: Formulating a conceptual model for developing decision support system generators. – In: Proceedings of the International Conference on System Science. – Hawaii, 1 (1982) – S. 812–819
[277] WILKES, M. V.: An experiment with a self-compiling compiler for a simple list processing language. – In: Annual Review in Automatic Programming. – New York, 4 (1964) – S. 1–48
[278] WILLIAMS, R.: A concise notation for the TRAC scanning algorithm. – In: SIGPLAN Notices. – New York, 3 (1968) 7 – S. 31–32

Sachwortverzeichnis

Ein eigenständiges Buch zur Makroprogrammierung?

Diese Frage wird den Lesern wohl als erste einfallen. Der Autor hat sich dafür entschieden, weil

- heute eine zusammenfassende Darstellung zur Programmierung (ihren Grundlagen, den gebräuchlichsten Programmiersprachen und den Programmiertechniken) kaum noch möglich ist;
- die Anzahl der relativ weit verbreiteten Makrosprachen in der Welt die Fünfzig schon weit überschritten hat und für die gebräuchlichsten Programmiersprachen – wie BASIC, FORTRAN, COBOL, PL/1, PASCAL u.v.a.m. – genutzt wird;
- die Notwendigkeit einer rationellen Programmentwicklung für die rasch ansteigende Computerzahl im praktischen Einsatz auch die Anwendung der speziellen Vorteile der Makroprogrammierung erfordert.

Der besondere Charakter der **Makroprogrammierung** als Programmiertechnik (der Makrotechnik) besteht in der **Zusammenfassung von Programmanweisungen** einer Programmiersprache nach solchen Aspekten wie

- der Schaffung neuer, verallgemeinerter Anweisungen über eine vorgegebene Programmiersprache,
- der Nutzungsmöglichkeit dieser Anweisungszusammenfassungen durch weitere Programmierer und der damit möglichen effektiven Ausnutzung von Programmiererfahrung.

Im vorliegenden Buch sind nach einer theoretischen Einführung zur Makroprogrammierung eine Reihe von Makrobeispielen zu den verschiedensten Makrosprachen angegeben, die – unabhängig von der Verfügbarkeit der jeweiligen Makrosprache – die grundlegenden Techniken und die breiten Anwendungsmöglichkeiten darstellen. Detaillierte Beschreibungen von Assemblermakrosprachen für Mikro-, Mini- und ESER-Computer sowie zur Makroprogrammierung für die universelle Programmiersprache PL/1 dienen der praktischen Umsetzung der allgemeinen Beschreibung der Makroprogrammierung und ihren Anwendungsmöglichkeiten

μακρό макро マクロ macro ماكرو

ISBN 3-343-00228-3