ANWENDER-DOKUMENTATION	Anleitung für den Programmierer	MOS
11/87	Makroprozessor m4	MUTOS 1700

Programmtechnische Beschreibung Teil 2

Anleitung für den Programmierer

Makroprozessor m4

AC A 7100/7150

VEB Robotron-Projekt Dresden

C 8014-0002-1 M 3030

Ausgabe: 11/87

Die Ausarbeitung dieser Dokumentation erfolgte durch ein Kollektiv des
VEB Robotron-Elektronik Dresden Stammbetrieb des VEB Kombinat Robotron.

Im Interesse einer ständigen Weiterentwicklung werden alle Leser gebeten, Hinweise zur Verbesserung der Dokumentation dem Herausgeber mitzuteilen.

Herausgeber:

VEB Robotron-Projekt Dresden
Leningrader Str. 9
Dresden 8010

Kurzreferat

"m4(1)" M4 ist ein Makroprozessor für MUTOS 1700, der zunächst als Präprozessor für Ratfor konzipiert wurde, wenn parameterlose Makros nicht wirkungsvoll genug sind. Er wurde auch als Präprozessor für so verschiedene Sprachen wie C und Cobol eingesetzt und ist auch für funktionelle Sprachen wie C, Fortran und PL/I geeignet, wenn die Makros in funktioneller Darstellung notiert werden.
In M4 sind zahlreiche Funktionen realisiert, die selbst umfangreichere Makroprozessoren selten bieten, einschließlich:

+ Makros mit Argumenten
+ Abfrage von Bedingungen
+ arithmetische Operationen
+ Zeichenkettenoperationen
+ File-Manipulation

Inhaltsverzeichnis

1. Einleitung

Ein Makroprozessor ist ein wertvolles Mittel, um die Verwendbarkeit und den Wert einer Programmiersprache zu erhöhen, sie lesbarer und anwenderfreundlicher zu machen bzw. sie auf spezielle Anwendungsgebiete zuzuschneiden. Der Befehl #define in C und das analoge define in Ratfor sind Beispiele für den Vorteil eines Makroprozessors - Ersetzen von Text durch anderen Text.

M4 ist ein geeigneter Präprozessor für Ratfor und C, kann aber auch in Zusammenhang mit Cobol verwendet werden. Neben dem einfachen Ersetzen einer Zeichenkette durch eine andere liefert M4 Makros mit Argumenten, Makroerweiterungen, Arithmetik, File-Manipulationen und einige spezielle Zeichenkettenfunktionen.

Die Grundoperation von M4 ist das Kopieren seines Eingabestromes auf seinen Ausgabestrom. Während der Eingabe wird jede Eingabezeichenfolge geprüft. Ist sie ein Makroname, so wird dieser Name durch den im Makro definierten Text ersetzt, und die resultierende Kette wird an die Eingabe zurückgegeben, wo sie erneut geprüft wird. Makros können mit Argumenten aufgerufen werden. In diesem Fall werden die Argumente an der richtigen Stelle im definierten Text ersetzt, bevor dieser erneut untersucht wird.

M4 liefert über 20 Standardfunktionen, die verschiedene nützliche Dinge leisten können. Darüberhinaus kann der Nutzer selbst neue Makros definieren. Standardfunktionen und Nutzermakros arbeiten in der gleichen Weise. Man beachte, daß einige Standardmakros Nebeneffekte bezüglich des Prozeßstatus haben.

2. Anwendung

Der Aufruf des Makroprozessors hat folgende Form:

```
m4 [files]
```

Die Argument-Files werden der Reihe nach abgearbeitet. Fehlt ein Argument oder ist stattdessen "-" angegeben, wird an dieser Stelle die Standardeingabe gelesen. Der bearbeitete Text wird auf die Standardausgabe geschrieben, kann aber zur Weiterverarbeitung auch mit

```
m4 [files] >ausgabefile
```

umgelenkt werden.

3. Definition von Makros

Die einfachste Standardfunktion von M4 ist define das zur Definition neuer Files benötigt wird. Die Eingabe

```
define(name,inhalt)
```

bewirkt, daß name definiert wird zu inhalt. Bei jedem weiteren Auftreten von name wird dieser durch inhalt ersetzt. name darf nur aus alphanumerischen Zeichen bestehen und muß mit einem Buchstaben beginnen (das Zeichen Unterstreichung (Underline) "_" zählt als Buchstabe). inhalt ist Text, der paarige Klammern enthalten und sich über mehrere Zeilen erstrecken kann.

Hier ein typisches Beispiel

```
define(N,100)
...
if (i > N)
```

definiert N zu 100, und benutzt diese "symbolische Konstante" in einem späteren if Befehl.

Die linke Klammer muß unmittelbar dem Wort define folgen, um zu kennzeichnen, daß dieses define Argumente hat. Wird ein Makro oder eine Standardfunktion nicht von einer öffnenden Klammer gefolgt, wird angenommen, es hätte keine Argumente. Das ist z.B. der Fall bei N.

Man sollte beachten, daß ein Makroname nur dann als solcher erkannt wird, wenn er von nichtalphanumerischen Zeichen eingeschlossen ist. So steht im folgenden Beispiel

```
define(N,100)
...
if (NNN > 100)
```

die Variable NNN in keinem Zusammenhang zu dem definierten Makro N, obwohl sie mehrere N enthält.

Es ist auch möglich, Makros indirekt zu definieren. Zum Beispiel definieren

```
define(N,100)
define(M,N)
```

sowohl M als auch N zu 100.

Dabei erfolgt die Ersetzung sofort vollständig. Das heißt, zunächst wird in der ersten Anweisung N durch 100 ersetzt, dann in der zweiten und dann M. Wird N an anderer Stelle neu definiert, hat das keinen Einfluß auf M. Wie kann nun also realisiert werden, daß sich bei einer Neudefinition von N auch M mit verändert?

In diesem Fall würde das Vertauschen der Makrodefinitionen genügen:

```
define(M,N)
define(N,100)
```

Jetzt wird M als N definiert und durch den aktuellen Wert von N ersetzt.

4. Apostrophieren

Das Problem läßt sich aber auch prinzipiell - von der Reihenfolge der Makrodefinitionen unabhängig - lösen durch das sogenannte Apostrophieren der Argumente. Dabei werden die zwischen den Zeichen Gravis (Grave accend) "`" und Apostrophe (Apostrophe) "'" eingeschlossenen Argumente nicht innerhalb ihres definierenden Textes, sondern erst beim Aufruf des Makros ersetzt.
Beispiel:

```
define(N,100)
define(M,`N')
...
if  M>200
```

Die Zeichen um N bewirken, daß M erst in dem Vergleich durch den aktuellen Wert von N ersetzt wird. Generell löst M4 erst dann innerhalb solcher Zeichen eingeschlossene Zeichenketten auf, wenn irgendetwas numerisch ausgewertet werden soll. Das trifft sogar außerhalb von Makros zu. Soll beispielsweise das Wort define unverändert in der Ausgabe erscheinen, so muß es bei der Eingabe durch Gravis "'" und Apostrophe "'" begrenzt werden, wie in

```
`define' = 1
```

Ein anderes Beispiel für die gleiche Sache, das noch überraschendere Effekte zeigt, ist die Redefinition von N:

```
define(N,100)
...
define(N,200)
```

Zunächst wird N ersetzt durch 100. Nun aber wird jedes weitere im Text vorkommende N durch diesen Wert ersetzt, also auch in der zweiten Definition und es entsteht:

```
define(100,200)
```

Diese Anweisung wird von M4 ignoriert, da 100 als Makroname unzulässig ist. Es ist offensichtlich, daß der gewünschte Effekt nicht eingetreten ist. Um N tatsächlich neu zu definieren, müssen folgende Anweisungen geschrieben werden:

```
define(N,100)
...
define(`N',200)
```

In M4 erweist es sich oft als zweckmäßig, das erste Argument eines Makros zwischen den Zeichen "`" und "'" einzuschließen.

Wenn die Zeichen "`" und "'" nicht genommen werden sollen, können neue mit dem Makro changequote definiert werden.

```
changequote([, ])
```

definiert die rechte und die linke eckige Klammer als Begrenzungszeichen
Der Ausgangszustand kann mit

```
changequote
```

ohne Argumente wiederhergestellt werden.

Es gibt zwei zusammengehörige Standardfunktionen, um die Definition von Makros zu realisieren. über define wurde schon gesprochen. Um eine define Anweisung aufzuheben, gibt es die Funktion undefine.

```
undefine(`N')
```

löscht also die Definition von N. Auch Standardfunktionen können auf diese Weise gelöscht werden, wobei beachtet werden muß, daß das nicht rückgängig zu machen ist, also beispielsweise

```
undefine(`define')
```

nicht unbedingt zu empfehlen ist.

Die Standardfunktion ifdef bietet die Möglichkeit, entsprechend vordefinierter Namen Textsegmente auszuwählen, also beispielsweise eine systemabhängige bedingte übersetzung zu realisieren.
Beispiel:

```
ifdef(`system1',`define(wordsize,16)')
ifdef(`system2',`define(wordsize,36)')
```

liefert eine maschinenabhängige Definition der Wortlänge. Nicht die Begrenzungszeichen vergessen!

ifdef hat im allgemeinen drei Argumente. Ist der Name in Argument1 nicht definiert, ist der Wert von ifdef gleich dem dritten Argument, wie in

```
ifdef(`mutos', in Mutos, nicht in Mutos)
```

5. Argumente

Wir haben nun die einfachste Form der Makrodefinition - das Ersetzen einer Zeichenkette durch eine andere (feste) Zeichenkette - abgehandelt. Vom Nutzer definierte Makros können aber auch Argumente haben, so daß verschiedene Makroaufrufe auch zu unterschiedlichen Resultaten führen können. Innerhalb des Ersetzungstextes eines Makros (also im 2. Argument von define) wird jedes auftretende $n durch das n.-te Argument im Makroaufruf ersetzt. So liefert der Makro bump, definiert als

```
define(bump, $1 = $1 + 1)
```

z.B. ein Makro, das die Erhöhung seines Argumentes bewirkt.

```
bump(x)
```

ist

```
x = x + 1
```

Ein Makro kann beliebig viele Argumente haben, aber nur die ersten neun werden verwendet, und zwar von $1 bis $9. (Der Makroname selbst ist $0, kann also nicht mit verwendet werden.) Fehlende Argumente werden durch leere Zeichenketten ersetzt. Damit können wir beispielsweise ein Makro cat definieren, das seine Argumente einfach aneinanderkettet:

```
define(cat,$1$2$3$4$5$6$7$8$9)
```

So ist also

```
cat(x,y,z)
```

äquivalent zu

```
xyz
```

$4 bis $9 sind Null, da keine entsprechenden Argumente angegeben sind.

Führende Leerzeichen, Tabulatoren und Neue Zeile "NL" werden ignoriert. Innerhalb von Argumenten werden sie übernommen. So definiert

```
define(a,   b   c)

a zu b   c .
```

Die Argumente werden durch Kommas getrennt. Enthalten jedoch eingeklammerte Ausdrücke Kommas, wird an dieser Stelle kein Argument abgeschlossen. So hat

```
define(a,(b,c))
```

nur zwei Argumente, nämlich a und (b,c). Einzelne Kommas können natürlich auch eingefügt werden, indem man sie apostrophiert.

6. Arithmetische Standardfunktionen

M4 bietet zwei arithmetische Standardfunktionen zum Rechnen mit (ausschließlich) ganzen Zahlen. Die einfachere ist das Inkrementieren incr wodurch das arithmetische Argument um 1 erhöht werden kann. Möchte man sich also eine Variable schaffen, die immer um 1 größer als der aktuelle Wert von N ist, schreibt man:

```
define(N,100)
define(N1,`incr(N)')
```

und N1 ist definiert als N + 1.

Die allgemeinere Standardfunktion zur Behandlung arithmetischer Ausdrücke ist eeevvvaaalll welches die folgenden Operatoren enthält (in der Reihenfolge ihrer Priorität):

+ und -	(Vorzeichen)
** oder ^	(Potenzierung)
* / %	(modulo Division)
+ -	(Vorzeichen)
== != < <= > >=	(Relationszeichen)
!	(nicht)
& oder &&	(logisches und)
| oder ||	(logisches oder)

Mit Klammern können Gruppen von Operationen zusammengefaßt werden. Sämtliche arithmetischen Ausdrücke von eval müssen numerischen Werten entsprechen. Der numerische Wert einer wahren Aussage (wie 1>0) ist 1. Die Genauigkeit der Operationen in eval beträgt 32 Bit.

Als ein kleines Beispiel für die Anwendung von eval wollen wir ein Makro M definieren, das für den aktuellen Wert von N den Wert 2**N+1 liefert:

```
define(N,3)
define(M,`eval(2**N+1)')
```

Es ist prinzipiell ratsam, den definierenden Text eines Makros zu apostrophieren, sofern er nicht ganz simpel ist. So wird mit einiger Sicherheit das gewünschte Resultat erzielt.

7. File-Manipulation

Man kann an beliebiger Stelle der Eingabe den Inhalt eines Text-Files mit der Standardfunktion include einfügen:

```
include(filename)
```

fügt den Inhalt des Files filename anstelle der include-Anweisung ein. Diese Files beinhalten häufig Definitionen. Der Wert von include (das ist sein ersetzender Text) ist der Inhalt des Files. Dieser kann untergliedert sein in Definitionen und anderes.

Es ist ein Fehler, wenn das in einem include angegebene File nicht gefunden wird. Um einen Programmabbruch an dieser Stelle zu vermeiden, gibt es die Funktion sinclude ("stilles include"), welche keine Fehlermeldung bringt, wenn das File nicht gefunden wird.

Es ist möglich, mit der Funktion divert die Ausgaben von M4 während der Abarbeitung in temporäre Files auszulagern und die gesammelten Informationen auf ein Kommando hin auszugeben. M4 ermöglicht 9 verschiedene solche Auslagerungen von 1 bis 9. Schreibt man also

```
divert(n)
```

werden alle nachfolgenden Ausgaben an das Ende des temporären Files, das durch n bezeichnet wird, geschrieben. Die Ausgabe auf dieses File wird durch ein neues Kommando divert gestoppt, insbesondere wird durch divert oder divert(0) der normale Ausgabeprozeß wieder hergestellt.

Ausgelagerter Text wird normalerweise am Ende des Prozesses in der Reihenfolge der Nummern der Files ausgegeben. Es ist aber auch möglich, ihn jederzeit mit der Funktion undivert zurückzuholen, um ihn im aktuellen Ausgabestrom einzufügen. So gibt

```
undivert
```

alles der Reihe nach zurück, wobei undivert mit Argumentent die ausgewählten Auslagerungen in der angegebenen Reihenfolge ausgibt. Bei der Rückgabe ausgelagerter Informationen wird der Inhalt gelöscht, ebenso beim Auslagern auf Files, die nicht zwischen 0 und 9 liegen.

Der Wert von undivert ist nicht der ausgelagerte Inhalt. Darüberhinaus wird zurückgegebener Text nicht noch einmal auf eventuell enthaltene Makronamen hin überprüft.

Die Standardfunktion

divnum

liefert die Anzahl der im Moment genutzten Auslagerungen. Während der normalen Abarbeitung ist ihr Wert Null.

8. Systemkommandos

Jedes Kommando kann innerhalb der aktuellen Arbeit aktiviert werden, indem es an der gewünschten Stelle mit der Standardfunktion syscmd gerufen wird. So wird z.B. mit

```
syscmd(date)
```

das Kommando date aktiviert. Normalerweise wird syscmd benutzt, um Files für ein späteres include zu erzeugen.

Um die Herstellung verschiedener File-Namen zu erleichtern, gibt es die Standardfunktion maketemp deren Wirkungsweise mit der des Systemkommandos mktemp identisch ist: eine Folge von XXXXX im Argument wird ersetzt durch die Prozeß-ID des aktuellen Prozesses.

9. Bedingte Anweisungen

Es gibt eine Standardfunktion mit dem Namen ifelse, welche es ermöglicht, Bedingungen zu testen und je nach Resultat bestimmte Aktionen auszuführen. In der einfachsten Form vergleicht

```
ifelse(a,b,c,d)
```

die zwei Zeichenketten a und b. Sind sie identisch, liefert ifelse die Zeichenkette c, andernfalls die Zeichenkette d. So sind wir beispielsweise in der Lage, ein Makro compare zu erzeugen, das zwei Zeichenketten vergleicht und je nachdem "gleich" oder "ungleich" liefert:

```
define(compare,`ifelse($1,$2,gleich,ungleich)')
```

Man beachte das Apostrophieren, das ein zu zeitiges Ersetzen von ifelse verhindern.

Fehlt das vierte Argument, wird es als leer vorausgesetzt.

ifelse kann eine Vielzahl von Argumenten haben, so daß eine eingeschränkte Form einer Mehrwegeentscheidung möglich wird. Wird beispielsweise

```
ifelse(a,b,c,d,e,f,g)
```

eingegeben, so werden zunächst a und b verglichen und das Resultat ist bei übereinstimmung c. Andernfalls wenn d gleich e ist, ist das Ergebnis f, ansonsten g. Wird das letzte Argument weggelassen, ist das Ergebnis Null, so auch bei

```
ifelse(a,b,c).
```

Das Ergebnis ist c, wenn a und b übereinstimmen und sonst Null.

10. Zeichenkettenmanipulation

Die Standardfunktion len liefert die Länge der Zeichenkette, die ihr Argument bildet. So ist

```
len(abcdef)
```

gleich 6 und

```
len((a,b))
```

gleich 5.

Die Standardfunktion substr kann benutzt werden, um aus einer Zeichenkette eine Teilzeichenkette auszuwählen.

```
substr(s,i,n)
```

liefert eine Teilzeichenkette von s, die bei Position i beginnt (der Anfang ist bei 0) und n Zeichen lang ist. Wird n weggelassen, wird der Rest der Zeichenkette geliefert. So ergibt

```
substr(`now is the time',1)
```

die Teilzeichenkette "ow is the time".

Die Standardfunktion index liefert die Position einer Teilzeichenkette.

```
index(s1,s2)
```

liefert den Index (die Position) in s1 bei der die Teilzeichenkette s2 beginnt oder -1, wenn s2 nicht in s1 enhalten ist. Wie in substr, ist der Anfang der Zeichenketten bei Position 0.

Die Standardfunktion translit führt eine Zeichenumwandlung durch.

```
translit(s,f,t)
```

modifiziert die Kette s derart, daß jedes Zeichen aus f in das korrespondierende Zeichen in t umgewandelt wird.

Beispiel: Ersetzen der Vokale durch die entsprechenden Ziffern

```
translit(s,aeiou,12345)
```

Mit translit können beispielsweise Groß- und Kleinbuchstaben ineinander umgewandelt werden. Wenn t kürzer als f ist, werden die Zeichen, denen eine Entsprechung in t fehlt, gelöscht, im Grenzfall werden bei fehlendem t alle Zeichen aus f in s gelöscht. So löscht

```
translit(s,aeiou)
```

sämtliche Vokale in s.

Es gibt auch eine Standardfunktion dddnnnlll die alle nachfolgenden Zeichen einschließlich "NL" löscht. Auf diese Weise können z.B. störende Leerzeilen entfernt werden, die nach dem Ersetzen von M4-Anweisungen übrigbleiben. Schreibt man beispielsweise

```
define(N,100)
define(M,200)
define(L,300)
```

ist "NL" am Ende jeder Zeile kein Teil der Definition, wird aber mit in die Ausgabe kopiert, wo es nicht hingehört. Schreibt man nun dnl in jede dieser Zeile, wird das verhindert.

Ein anderer, vielleicht etwas eleganterer Weg zum gleichen Ziel ist der folgende:

```
divert(-1)
    define(...)
    ...
    divert
```

11. Ausgabe

Die Standardfunktion errprint schreibt ihre Argumente auf das Standard-Error-File. So kann man beispielsweise schreiben:

```
errprint(`fatal error')
```

Die Standardfunktion dumpdef ist eine Funktion, die bei der Fehlersuche hilft. Sie gibt die aktuell definierten Makronamen und den definierenden Text der Nutzermakros aus. Ohne Argumente gibt dumpdef alle Definitionen aus, ansonsten nur die, deren Namen als Argumente angegeben sind. Nicht das Apostrophieren bei den Makronamen vergessen!

12. Zusammenstellung der Standardfunktionen

```
changequote(L, R)
define(name, ersetzung)
divert(nummer)
divnum
dnl
dumpdef(`name', `name', ...)
errprint(s, s, ...)
eval(numerischer ausdruck)
ifdef(`name',`name',...)
ifelse(a, b, c, d)
include(file)
incr(variable)
index(s1, s2)
len(string)
maketemp(...XXXXX...)
sinclude(file)
substr(string, position, anzahl)
syscmd(s)
translit(str, von, zu)
undefine(`name')
undivert(nummer,nummer,...)
```