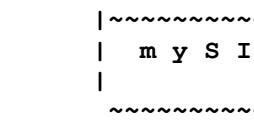
C o m p u t e r a l g e b r a s y s t e m



Handbuch und Systembeschreibung

Technische Universitaet Karl-Marx-Stadt

Sektion INFORMATIK

Wissenschaftsbereich Praktische Informatik

Forschungsgruppe FORMELMANIPULATION

Inhaltsverzeichnis

1. Ueberblick ueber myMATH und mySIMP

myMATH ist faehig, mathematische Formelausdruecke umzuformen. Dabei ist es im Unterschied zu "klassischen" Programmiersprachen, wie ALGOL, FORTRAN,PL/I, PASCAL, BASIC usw. nicht notwendig, dass die verwendeten Variablen zur Laufzeit einen Wert besitzen. Variable werden, wie in der Mathematik ueblich, als formale Rechengroessen verwendet, sie koennen als Bezeichner fuer einen Ausdruck fungieren, sie koennen aber auch fuer sich selbst stehen.

Der Nutzer von **myMATH** benoetigt zunaechst keinerlei Vorkenntnisse in der Programmierung, fuer ein erstes Vertrautmachen mit dem System genuegen folgende Konventionen:

1. Ein Stern, ***** , wird als Multiplikationsoperator verwendet.

2. Der nach oben gerichtete Pfeil, ^ , dient zur Kennzeichnung der Potenzbildung.

3. Das Ende jedes vom Nutzer eingegebenen Ausdrucks wird durch ein Semikolon angezeigt.

Moechte der Nutzer beispielsweise den mathematischen Ausdruck

$$2\,y\,(y^2 - z) + 2\,z\,(y + z)$$

vereinfachen, so gibt er

```
2 * Y * (Y^2 - Z) + 2 * Z * (Y + Z);
```

ein, und **myMATH** vereinfacht dies sofort zu dem aequivalenten Ausdruck

```
2*Y^3 + 2*Z^2 .
```

Andere algebraische Transformationen beinhalten beispielsweise das Zusammenfassen mehrerer rationaler Ausdruecke ueber einem Hauptnenner sowie das partielle Faktorisieren. Falls erforderlich, koennen die vereinfachten Ausdruecke fuer eine spaetere Weiterverwendung abgespeichert werden.

myMATH akzeptiert aber auch Gleichungen als Ausdruecke, in denen die beiden Seiten unabhaengig voneinander vereinfacht werden. Gleichungen koennen beispielsweise addiert oder multipliziert werden, um so schrittweise Loesungen herzuleiten. Es existiert andererseits aber auch eine Funktion **SOLVE**, mit der Gleichungen automatisch geloest werden koennen. Um zum Beispiel die Gleichung

$$x\,(3 + x^2) = 4\,x\,(1 + c^2) - x$$

nach x aufzuloesen, wird der folgende Ausdruck eingegeben:

```
SOLVE(X*(3+X^2) == 4*X*(1+C^2) - X, X);
```

In wenigen Sekunden liefert **myMATH** die Loesungsmenge

```
{X == -2*C,
```

```
X == 2*C,
X ==0}.
```

myMATH ermoeglicht auch die Verarbeitung von Feldern, Matrizen und Determinanten mit nichtnumerischen Elementen, es beherrscht viele Funktionen der Matrizenalgebra.

Um beispielsweise die Inverse der Matrix

```
[1   2 y z^2]
[y      2   ]
```

zu ermitteln, gibt der Nutzer den Ausdruck

```
{[1, 2*Y*Z^2],
 [Y,     2  ]} ^ -1;
```

ein und erhaelt von **myMATH** das Resultat

```
{[1+2*Y^2*Z^2/(2-2*Y^2*Z^2), -2*Y*Z^2/(2-2*Y^2*Z^2)],
 [   -Y/(2-2*Y^2*Z^2),       -1/(2-2*Y^2*Z^2)      ]}
```

myMATH kann auch Summationen in geschlossener Form ausfuehren, dies sogar fuer nichtnumerische Summationsgrenzen. Um zum Beispiel eine geschlossene Darstellung fuer

$$\sum_{j=0}^{n-1} (c^j + j^2)$$

zu erhalten, gibt der Nutzer

```
SIGMA (C^J +J^2, J 0, N-1);
```

ein, und in weniger als einer Minute liefert **myMATH** das Ergebnis

```
(-36-6*N+6*C*N-18*C*N^2+12*C*N^3+36*C^N+18*N^2-12*N^3) / (-36+36*C).
```

myMATH verfuegt ueber zahlreiche Faehigkeiten zur Transformation von logarithmischen, trigonometrischen und von Exponentialausdruecken. Diese koennen wahlweise automatisch oder nutzergesteuert wirksam werden. So kann **myMATH** etwa den Ausdruck

```
sin(2y) (4 cos^3x - cos(3x) + (cos(x+y+pi) - cos(x-y)) sin y
```

zu

```
4 sin y cos x cos y
```

vereinfachen, dies in weniger als einer halben Minute.

myMATH hat auch gewisse Faehigkeiten zur symbolischen (unbestimmten) Integration. Um beispielsweise das Integral

$$\int (c\ x^2 + \sin(x^2))\ dx$$

$$\int$$

zu bestimmen, gibt der Nutzer ein:

```
INT(C*X^2 + X*SIN(X^2), X);
```

In wenigen Sekunden liefert **myMATH** das Resultat

```
C*X^3/3 - COS(X^2)/2
```

Ferner kennt **myMATH** Differentiationsregeln und ist faehig, Taylorreihenentwicklungen zu erzeugen. So kann der Nutzer zur Ermittlung der Taylorreihe 5-ter Ordnung fuer

$$e^{\sin x}$$

an der Stelle x=0 eingeben

```
TAYLOR(#E^SIN(X), X, 0, 5);
```

und erhaelt in weniger als einer Minute das Resutat

```
1 + X + X^2 - X^4/8 - X^5/15.
```

myMATH kennt ausserdem die L'Hospital'sche Regel zur Bestimmung von Grenzwerten. Um beispielsweise

$$\lim_{x \to 0} \frac{a^x - a^{\sin x}}{x^3}$$

zu bestimmen, gibt der Nutzer

```
LIM((A^X - A^SIN(X)) / X^3, X, 0);
```

ein und erhaelt von **myMATH** das Resultat

```
LN(A) / 6 .
```

Es soll noch darauf hingewiesen werden, dass **myMATH** ueber eine Rationalarithmetik verfuegt, d.h. alle Zahlenrechnungen sind voellig frei von Rundungsfehlern. So reagiert das System beispielsweise auf die Eingabe von

```
1/2 + 1/3;
```

mit dem exakten Ergebnis **5/6.**

Die Groesse der Ausdruecke, die **myMATH** verarbeiten kann, haengt lediglich vom vefuegbaren Speicherbereich ab. Berechnungen mit hunderten von Termen, von denen jeder Koeffizienten mit mehreren Hundert Ziffern enthaelt, sind ohne Probleme durchfuehrbar.

Wie bereits angedeutet, hat **myMATH** sehr weitreichende mathematische Faehigkeiten. Es ist **modular** aufgebaut, so dass der Nutzer das System seinen Beduerfnissen und Kenntnissen entsprechend verwenden kann.

Die wenigen Beispiele haben bereits angedeutet, dass **myMATH** wie ein Taschenrechner anwendbar ist. Ausdruecke werden transformiert und vereinfacht, wenn sie eingegeben werden. Diese Art der Verwendung von **myMATH** im sog. **Calculator-Modus** erspart dem Nutzer das Schreiben langer Programme, er kann frei experimentieren, indem er Formeln eingibt und sofort das Resultat sieht. Fehler werden schnell entdeckt und koennen daher sofort beseitigt werden.

Fuer viele Zwecke ist der Calculator- Modus sehr guenstig anwendbar. **myMATH** ist andererseits aber in einer Programmiersprache mySIMP geschrieben, die gemeinsam mit **myMATH** zur Verfuegung gestellt wird, so dass der Nutzer die Faehigkeiten des Systems beliebig erweitern kann **(Programm-Modus).**

myMATH bietet dem Nutzer noch eine Anzahl von interaktiv nutzbaren Lektionen an, die das Programmieren in **mySIMP** erklaeren. Im Gegensatz zum Calculator-Modus erfordert das Programmieren in **mySIMP** aber gewisse Programmiererfahrengen vom Nutzer. Es sind dabei nicht vordergruendig Kenntnisse einer speziellen Programmiersprache erforderlich, sondern Grundprinzipien der Programmierungstechnik. Ein einfaches Beispiel soll einen ersten Endruck ueber die Art und Weise der Programmierung in **mySIMP** vermitteln:

Wir definieren in **mySIMP** eine Funktion zur Berechnung der n-ten Fibonacci-Zahl die bekanntlich durch folgende Relation definiert ist:

```
F(0) = 1,
F(1) = 1,
F(n) = F(n-1) + F(n-2)    fuer  n>1.
```

Die entsprechende Funktionsdefinition in **mySIMP** ist

```
FUNCTION F(N),
WHEN N=0 OR N=1, 1 EXIT,
F(N-1) + F(N-2),
ENDFUN;
```

Es wird hierbei eine grundsaetzliche Faehigkeit von **mySIMP** deutlich, die Moeglichkeit der Definition rekursiver Funktionen. Dies entspricht in vielen Faellen der urspruenglichen mathematischen Funktionsdefinition und ermoeglicht eine sehr kurze und elegante Notation der zu definierenden Funktionen. Diese Art der rekursiven Programmierung ist aber andererseits fuer viele Nutzer ungewohnt, rekursive Programme erfordern in manchen Faellen auch relativ viel Speicherplatz. **myMATH** bietet dem Nutzer deshalb auch programmiersprachliche Mittel an, mit denen er sequentielle Programme schreiben kann.

Funktionen wie das obige Beispiel koennen vom Nutzer unmittelbar am Terminal definiert werden (Calculator-Modus) sie sind unmittelbar nach ihrer Definition voll verfuegbar. Funktionsdefinitionen koennen aber auch mittels eines Textverarbeitungsprogrammes erzeugt und in ein vom Nutzer angelegtes File geschrieben werden. Diese File kann durch ein spezielles Kommando von **myMATH** bzw. **mySIMP** aus geladen werden, die in ihm enthaltene Funktionsdefinition ist dann sofort verfuegbar.

Neben der bereits angedeuteten Faehigkeit des rekursiven Programmierens ist die Verwendung dynamischer Datenstrukturen, wie Listen,

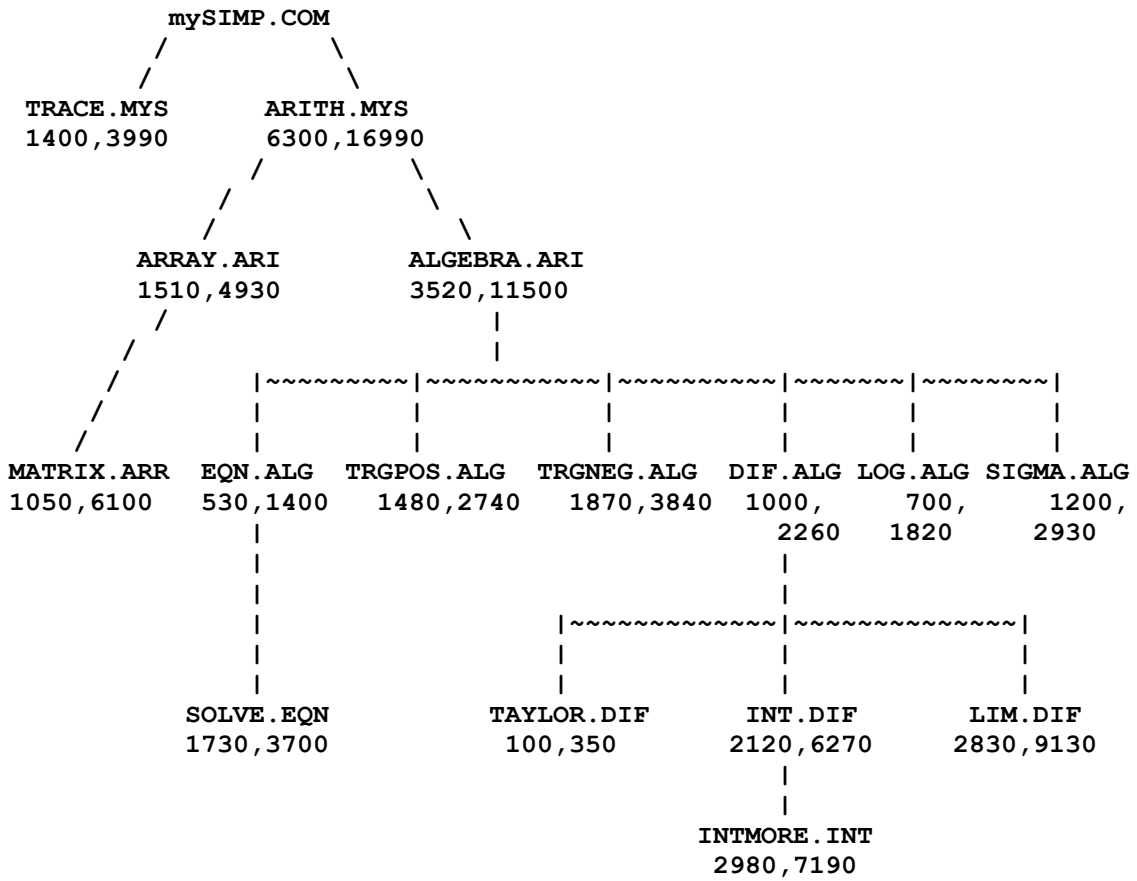
Baeume und dgl.fuer **mySIMP** charakteristisch. Die verwendeten Datenstrukturen sowie die verfuegbaren Verarbeitungsfunktionen entsprechen genau denen von **LISP**. Damit ist **mySIMP** durchaus als Programmierwerkzeug fuer Fragen der kuenstlichen Intelligenz verwendbar.

2. Struktur von myMATH (Filehierarchie)

myMATH bietet eine Vielzahl von Faehigkeiten zur Ausfuehrung symbolischer mathematische Berechnungen. Es wird als hierarchisches System von Quelltextfiles zur Verfuegung gestellt. Die in diesen Files enthaltenen Verarbeitungsfunktionen sind saemtlich in mySIMP geschrieben, sie haben einen modularen Aufbau und sind damit an verschiedene mathematische Aufgabenstellungen sowie Rechnerkonfigurationen anpassbar. Die Verwendung sowie die mit diesen Files verfuegbaren Faehigkeiten werden in diesem Handbuch ausfuehrlich erklaert.

Neben diesen Files mit Verarbeitungsfunktionen werden Files zur Verfuegung gestellt, die Lektionen enthalten, in denen der Umgang mit dem System und die wesentlichsten Faehigkeiten erklaert werden.

Folgendes Diagramm gibt einen Uberblick ueber die Struktur von myMATH und verdeutlicht die Abhaengigkeit der einzelnen Files:



Jedes myMATH-File in diesem Schema benoetigt zu seiner Arbeit alle zwischen ihm selbst und der Wurzel des Systems (also **mySIMP.COM**) liegenden Files. So benoetigt also z.B. **MATRIX.ARR** die Files **ARRAY.ARI, ARITH.MYS** sowie **mySIMP.COM.** Der Aufruf des Files **mySIMP.COM** laedt den Interpreter von mySIMP und stellt damit programmiersprachliche Werkzeuge zur Verfuegung.

myMATH-Systeme werden aufgebaut, indem zunaechst **mySIMP.COM** geladen

wird. Anschliessend werden aus der obigen Baumstruktur alle benoetigten Files gelesen. Dabei kann es durchaus guenstig sein, ausser den aus der Filehierarchie ersichtlichen unbedingt notwendigen Files noch weitere zu laden, dies haengt ab von den zu bearbeitenden Aufgaben und den dafuer erforderlichen bzw. wuenschenswerten mathematischen Faehigkeiten. So ist es beispielsweise ratsam, **LOG.ALG** zu laden, wenn **EQN.ALG** verwendet wird. Falls **INTMORE.INT** zur Ermittlung unbestimmter Integrale verwendet wird bzw. bei der Verwendung von **SIGMA.ALG** zur Bestimmung unendlicher Reihen ist es unbedingt notwendig, **LIM.DIF** zu laden.

Die erste Zahl unterhalb des jeweiligen Filenamens gibt die ungefaehre Anzahl der erforderlichen Bytes an, wenn das File in mySIMP gelesen wird. Diese Zahl kann benutzt werden, um zu bestimmen, ob das verfuegbare Computersystem ueber genuegend Speicher zur Erzeugung der gewuenschten myMATH-Variante verfuegt. Die zweite Zahl gibt die Grosse des Quelltextfiles in Bytes auf der Diskette an.

Fuer alle mit dem System gelieferten bzw. durch das SAVE-Kommando von mySIMP erzeugten Files gelten folgende Konventionen:

1. Files des Typs **COM** sind nichtdruckbare Kommandofiles in Maschinensprache, sie sind abarbeitbar als ein Kommando auf dem Niveau des Betriebssystems.

2. Files des Typs **SYS** sind nichtdruckbare Speicherabzuege, die vom Nutzer durch das mySIMP-Kommando **SAVE** generiert werden koennen.

3. Files, deren Name mit **CLESn** beginnt, sind interaktive Lektionen fuer den Calculator-Modus. Diese erfordern zu ihrer Abarbeitung, dass mindestens das File **mySIMP.COM** geladen ist.

4. Files, deren Name mit **PLESn** beginnt, sind interaktive Lektionen zur Erlauterung des sog. Programm-Modus.

5. Alle anderen Files des Systems sind mySIMP-Quelltextfiles. Derartige Quelltextfiles koennen durch einen beliebigen Texteditor erzeugt bzw. veraendert werden, sofern dieser keine Zeilennummern erzeugt.

3. Die Arbeit mit mySIMP und myMATH

3.1 Verwendung von SYS-Files

Um die Abarbeitung eines SYS-Files aus mySIMP heraus zu bewirken, kann der Nutzer auf dem Niveau des Betriebssystems ein Kommando der Form

drv1:mySIMP drv2:filename

eingeben. Hierbei ist drv1 das Laufwerk, auf dem sich das File **mySIMP.COM** befindet, **drv2:filename** repraesentieren das Laufwerk und den Namen des zu verwendenden SYS-Files.

Soll mySIMP allein aufgerufen werden, so genuegt das Kommando

drv1:mySIMP

Die Angabe des Laufwerkes kann entfallen, wenn die entsprechenden Files auf dem aktiven Laufwerk liegen, d.h. es genuegt der Aufruf von

mySIMP ALGEBRA

um beispielsweise **mySIMP** und das File **ALGEBRA.SYS** zu laden, sofern sich beide auf dem aktiven Laufwerk befinden.
mySIMP meldet sich dann mit der Ausschrift:

Computeralgebrasystem

m y M A T H

***** TUK-Version *****

Eine weitere Moeglichkeit des Ladens von SYS-Files besteht darin, nach dem Aufruf von mySIMP mittels

mySIMP

das gewuenschte SYS-File durch das Kommando

LOAD(filename);

zu laden.

3.2 Interaktive Nutzung von mySIMP

mySIMP zeigt dem Nutzer durch ein Fragezeichen seine Arbeitsbereitschaft an. Der Nutzer kann dann einen Ausdruck eingeben, der durch ein Semikolon und die Enter-Taste abgeschlossen wird. Er kann ausser dem Semikolon auch noch die Trennzeichen **"&"** und **"$"** verwenden. Zunaechst wird der eingegebene Ausdruck in eine interne Form ueberfuehrt. Anschliessend zeigt ein **"@:"** auf der naechsten Zeile den Wert des zuletzt bearbeiteten Ausdrucks an, dieses Resultat wird normalerweise in mathematischer Notation angezeigt. Hat der Nutzer das Symbol **"&"** als Trennzeichen verwendet,so wird die interne Darstellung des

Resultates angezeigt, bei Verwendung von **"$"** wird das Resultat hingegen nicht angezeigt. Der Nutzer kann sich in der naechsten Anweisung auf das unter **"@"** angezeigte Resultat erneut beziehen. Andererseits hat er die Moeglichkeit der Verwendung von Variablen und Ergibtanweisungen in der Form

var: ausdr;

Hierbei ist **var** eine frei waehlbare Variablenbezeichnung, **ausdr** ist ein beliebiger Ausdruck.

Zur Illustration soll nachfolgend ein kurzes Beispiel fuer einen einfachen Dialog in mySIMP angegeben werden.

```
? 2 + 3;

@: 5


? HANS = MARIE;

@: FALSE

? MEMBER (APFEL, '(BIRNE,APFEL,PFLAUME));

@: TRUE
```

Diese trivialen Beispiele verwenden den arithmetischen Operator fuer die Addition, den Vergleichsoperator sowie eine Funktion, mit der ermittelt werden kann, ob eine Menge ein bestimmtes Element enthaelt. In den Lektionen zur Erklaerung des Calculator-Modus werden viele der verfuegbaren Operatoren und Funktionen naeher erlaeutert.

Die Eingabe eines Trennzeichens nach einem Ausdruck bewirkt, dass mySIMP versucht, diesen Ausdruck auszuwerten. Falls der Ausdruck syntaktische Fehler enthaelt, wird eine Fehlermitteilung ausgegeben. Ist der Ausdruck syntaktisch korrekt, so wird er in jedem Falle ausgewertet. Dies kann dann unerwuenscht sein, wenn der Ausdruck semantisch falsch ist, der Nutzer sich also einfach "verschrieben" hat. In diesem Falle besteht die Moeglichkeit, die Auswertung zu unterbrechen, dies wird im naechsten Abschnitt beschrieben.

Falls ein syntaktisch richtiger, aber "unerwuenschter" (also semantisch falscher) Ausdruck noch nicht mit einem Trennzeichen abgeschlossen wurde, so besteht die einfachste Moeglichkeit darin, bewusst einen Syntaxfehler zu erzeugen und anschliessend ein Trennzeichen einzugeben, beispielsweise durch

(;

Eine spezielle Variable **NEWLINE** steuert die Anzahl der Leerzeilen zwischen der mySIMP-Eingabe und den Antwortzeilen. Standardmaessig hat **NEWLINE** den Wert 1, dies bedeutet eine Leerzeile nach jeder Eingabe und zwei Leerzeilen nach jeder Ausgabe des Systems. Um diesen Wert auf 0 zu veraendern, kann

NEWLINE:0;

eingegeben werden.

3.3 Unterbrechungsmoeglichkeiten

Eine laufende Auswertung eines Ausdrucks kann durch Druecken der **ESC**ape-Taste abgebrochen werden. Nach diesem Eingriff in die Auswertung erscheint folgendes Menue von Moeglichkeiten auf dem Bildschirm:

***** INTERRUPT: To Continue Type: RET;**
Executive: ESC,ALT; System: CTRL-C?

1. Die "normale" Reaktion ist das Druecken der ESCape-Taste, dies ermoeglicht einen neuen Abarbeitungszyklus, also die Fortsetzung des Dialogs, wobei der unterbrochene Ausdruck nicht mehr verfuegbar ist.

2. **RET**urn bewirkt die Forsetzung der unterbrochenen Auswertung.

3. **CRTL-C** bewirkt das Verlassen von mySIMP und den Uebergang zum Betriebssystem. Dies ist dann sinnvoll, wenn sich das System wegen Erschoepfung des Speichers oder aus anderen Gruenden "verklemmt" hat und auf Nutzereingaben nicht mehr sinnvoll reagiert. In einem solchen Fall empfiehlt es sich, eine neue myMATH-Sitzung zu beginnen.

3.4 Einlesen von mySIMP-Quellfiles

Falls kein SYS-File mit den gewuenschten mathematischen Faehigkeiten verfuegbar ist, so koennen die erforderlichen Quellfiles mit einem **R**ea**D** **S**elect-Kommando der Form

RDS (filename, filetyp, drv);

eingelesen werden, hierbei kennzeichnet drv das Laufwerk, von dem das entsprechende File einzulesen ist. Die Angabe von drv kann wiederum entfallen, wenn sich das einzulesende File auf dem aktiven Laufwerk befindet.

Hinweis:

Es ist notwendig zu bemerken, dass **filename** und **filetyp** im Gegensatz zur SCPX-Notation durch ein **Komma** getrennt werden! Ferner genuegt zur Kennzeichnung des gewuenschten Laufwerkes die Angabe eines einzelnen Buchstaben, der in SCPX normalerweise notwendige Doppelpunkt kann entfallen.

Werden vom Nutzer mittels eines Texteditors selbst erstellte mySIMP-Quellfiles eingelesen, so sollten diese zur Vermeidung von Fehlern mit den Anweisungen

STOP; RDS();

enden.

Da jedes Quellfile einen gewissen Speicherbereich benoetigt, sollte man sich vergewissern, ob der verfuegbare Speicherplatz zum Laden der gewuenschten Files ausreicht. Hierzu ist die Kenntnis folgender Konventionen wichtig:

1. Die erste Zahl unter jedem Filenamen in dem im Kapitel 2 angegebenen Diagramm gibt die ungefaehre Groesse des beim Laden in mySIMP notwendigen Speicherplatzes in Bytes an.

2. Die Anzahl der noch verfuegbaren Bytes kann durch das mySIMP-Kommando

RECLAIM ();

abgefragt werden.

3. mySIMP benoetigt einen Teil des verfuegbaren Speichers fuer seine Arbeit. Sollte dafuer nicht genuegend Platz bleiben, so verursacht dies entweder den Fehler "space exhaustet" oder fuehrt zu erheblichen Verzoegerungen bei der Arbeit des Systems. Die Greosse des von MYMSIMP geforderten Speichers haengt von den zu verarbeitenden Ausdruecken ab. Als empfohlenes Minimum sind 4k Bytes anzusehen, unter dieser Grenze ist eine sinnvolle Arbeit nicht moeglich.

Als Beispiel fuer die Verwendung von **RDS** in Verbindung mit **RECLAIM** wollen wir ein System erzeugen, das den Gleichungsvereinfacher, also das File **EQN.ALG**, einschliesslich der dafuer notwendigen uebrigen Files, enthaelt. Wir nehmen an, dass sich alle benoetigten Files auf einer Diskette befinden, die im Laufwerk **A** Liegt. Der folgende Beispieldialog zeigt eine Moeglichkeit zur Loesung dieses Problems:

```
A>mySIMP ARITH

mySIMP-85 2.02

? RECLAIM();
@: 30000

RDS (ALGEBRA,ARI);
@: ALGEBRA

? RECLAIM();
@: 26500

? RDS (EQN,ALG);
@: EQN
```

mySIMP gestattet es, mehrere Ausdruecke auf eine Zeile zu schreiben, beispielsweise

```
? RDS(ALGEBRA,ARI); RDS(EQN,ALG); RDS(SOLVE,EQN);
```

Der unmittelbar nach dem Aufruf von mySIMP noch verfuegbare Speicher haengt von der jeweiligen Hardwarekonfiguration und von den Besonderheiten des verwendeten Betriebssystems ab. Auch der von den Quelltextfiles benoetigte Speicher haengt von der konkreten Umgebung ab, in welche diese Files geladen werden. mySIMP nimmt von sich aus gewisse Optimierungen des Speicherplatzes vor.

Quellfiles enthalten Folgen von Anweisungen, die ebenso interaktiv vom Bildschirm aus eingegeben werden koennten. Beim Einlesen werden diese Anweisungen sofort in die interne Form ueberfuehrt und anschliessend ausgewertet. Falls eine solche Anweisung eine Funktionsdefinition darstellt, bewirkt der Auswertungsprozess eine Pseudocompilierungsphase, in der alle vorher compilierten Funktionen gesucht werden, die in der "neuen" Funktion benoetigt werden.

Es soll vermerkt werden, dass das Einlesen und gleichzeitige Interpretieren bzw. Compilieren von Quelltexten wesentlich langsamer ist als das Laden aequivalenter **SYS**-Files, die ja Speicherabzuege enthalten. Im naechsten Kapitel wird erlaeutert, wie SYS-Files erzeugt werden koennen.

4. Erzeugen von Speicherabzuegen (SYS-Files)

Das Kommando SAVE

Das mySIMP-Kommando **SAVE** rettet einen kompakten Speicherabzug der zum jeweiligen Zeitpunkt existierenden Umgebung. Diese Umgebung besteht aus den Werten, Funktionsdefinitionen und Eigenschaften jedes im System vorhandenen Bezeichners. Es ist zu beachten, dass jeder Aufruf von mySIMP und jedes Laden von SYS-Files mit dem Kommando **LOAD** einen neuen Anfangszustand erzeugt.

Aus folgenden Gruenden ist es oft guenstig, vollstaendige Umgebungen zu retten:

1. Zur Vereinfachung und Beschleunigung des Startens oft benoetigter Filekombinationen

2. Zur Sicherung der in einer Sitzung erhaltenen Resultate, Variablenbelegungen, Funktionsdefinitionen usw., die dann in einer spaeteren Sitzung unveraendert weiterverwendet werden koennen.

3. Zur Erzeugung einer bestimmten myMATH-Version.

Das mySIMP-Kommando **SAVE** hat die allgemeine Form

SAVE (filename,drv);

und rettet den augenblicklichen Systemzustand in ein File mit dem angegebenen Namen und dem Typ SYS auf das gewuenschte Laufwerk bzw. standardmaessig auf das gegenwaertig aktive Laufwerk. Die Groesse des entstehenden SYS-Files haengt vom jeweiligen Systemzustand ab. Als grober Richtwert kann gelten:

Es sei **i** der von **RECLAIM()** gelieferte Wert, wenn nur mySIMP aktiv ist, **s** sei der unmittelbar vor Ausfuehrung des **SAVE**-Kommandos von **RECLAIM()** gelieferte Wert. Dann ist die Groesse des erzeugten SYS-files in KBytes ungefaehr

6 + (i-s)/1000.

Hinweis:

Vor einem **SAVE**-Kommando sollte man sich vergewissern, dass sich eine Diskette mit genuegend freiem Speicher in einem Laufwerk befindet. Ferner kann das Schreiben auf eine ohne Kalt- bzw. Warmstart zwischenzeitlich neu eingelegte Diskette zu Problemen fuehren. Es ist daher ratsam, vor Aufruf von mySIMP eine Diskette einzulegen und einen Warmstart auszufuehren.

5. Abarbeitung der interaktiven Lektionen

Dieses Kapitel erlaeutert die Verwendung der Files, welche die interaktiven Lektionen zu mySIMP und myMATH enthalten. Diese bieten ein breites Spektrum an mathematischen Faehigkeiten, angefangen von der Arithmetik bis hin zur Integralrechnung, sowie die notwendigen programmierungstechnischen Hintergruende.

Die Folge der Lektionen fuer den **Calculator-Modus** , genannt **CLES1**,**CLES2** usw. erklaert, wie myMATH als arithmetischer oder symbolischer Rechner zu verwenden ist. Die Lektionen **PLES1**, **PLES2** usw. erklaeren, wie in mySIMP Programme zu schreiben sind.

Es gibt im Prinzip drei Moeglichkeiten, sich mit diesen Lektionen zu beschaeftigen:

1. Der beste Weg ist ihre interaktive Abarbeitung, wobei gleichzeitig die angebotenen Beispiele probiert werden koennen.

2. Ferner kann der Nutzer sich die gedruckte Aufzeichnung des gefuehrten Dialogs nachtraeglich ansehen.

3. Schliesslich besteht die Moeglichkeit, die Texte der die Lektionen enthaltenen Files auszudrucken bzw. sich am Bildschirm anzusehen. Bei dieser Variante sieht der Nutzer aber nur einen Teil des beabsichtigten Dialogs, er erhaelt keine Information ueber die Resultate der in den Lektionen enthaltenen Beispiele.

Um die Lektionen abarbeiten zu koennen, ist es notwendig mindestens das File **ARITH.myS** zu laden bzw. eine myMATH-Version, die dieses File enthaelt. Dann wird mit dem Kommando

```
RDS (CLES1, ARI, drv);
```

das File gelesen, welches die erste Lektion enthaelt, **drv** gibt hierbei das Laufwerk an, auf dem sich die Diskette mit den Lektionen befindet.
Diese Angabe kann entfallen, wenn dies das aktive Laufwerk ist. In der ersten Lektion wird erklaert, wie alle weiteren Lektionen aufzurufen sind.

Hinweis:

Die Lektionen laufen relativ schnell ueber den Bildschirm, vor allem deren erste Seiten. Die Anzeige der Lektionen kann durch Druecken der Tastenkombination **CTRL-S** gestoppt und durch eine beliebige andere Taste wieder in Gang gesetzt werden. Ferner ist zu bemerken, dass der Text dieser Lektionen gegenwaertig nur in englischer Sprache verfuegbar ist. Bei den Lektionen handelt es sich um mySIMP-Quelltextkommentare, in die abarbeitbare Beispiele eingebettet sind. Der Nutzer kann die ausgewerteten Beispiele studieren und aber auch eigene Beispiele versuchen. Moechte er sich den weiteren Text der Lektionen anzeigen lassen, so wird dies durch eine Anweisung der Form

```
RDS : T;
```

erreicht.

Verzeichnis der Lektionen:

CLES1.ARI
CLES2.ARI
CLES3.ARI
CLES4.ARI
CLES5.ARI

PLES1.TRA
PLES2.TRA
PLES3.TRA
PLES4.TRA
PLES5.TRA

Zur interaktiven Nutzung der Lektionen **PLESi** ist es notwendig, das File **TRACE.myS** zu laden.

6. Verwendung der myMATH-Quellfiles

In diesem Kapitel werden die mathematischen Faehigkeiten der verschiedenen Files von myMATH erlaeutert. Abschnitt 6.1 erklaert die Verwendung eines Trace-Files, mit dem sinnvollerweise mySIMP-Programme getestet werden koennen. In den restlichen Abschnitten werden spezielle mathematische Faehigkeiten der einzelnen Komponenten von myMATH erklaert.

6.1 TRACE.myS: Trace-Faehigkeiten

Das File **TRACE.myS** enthaelt ein Trace-Paket zur Unterstuetzung der Programmtestung. Diese Faehigkeiten sind im Calculator-Modus nicht erforderlich.

Erforderliche Files: **mySIMP.COM**

Verwendete Steuervariable:

1. **MATHTRACE** legt die Art der Notation (mathematische Notation oder Listenschreibweise) bei Verwendung der Trace- Faehigkeiten fest. Der Standardwert ist TRUE.

Anwendung:

```
TRACE(name1, name2, ...),
UNTRACE(name1,name2,...).
```

Beispiel:

```
? FUNCTION MEMB(EX1,EX2),                          %Definition%
     WHEN EMPTY(EX2), FALSE EXIT
     WHEN EX1 = FIRST(EX2), TRUE EXIT
     MEMB (EX1,REST(EX2))
  ENDFUN$

? TRACE (MEMB) $                                   %TRACE-Anweisung%

? MEMB ('HUND, '(KATZE, KUH, HUND, SCHWEIN))$    %Funktionsaufruf%

MEMB [HUND, (KATZE, KUH, SCHWEIN)]                 %Reaktion %
  MEMB [HUND, (KUH, HUND, SCHWEIN)]
    MEMB [HUND, (HUND, SCHWEIN)]
    MEMB = TRUE
  MEMB = TRUE
MEMB = TRUE

?  UNTRACE  (MEMB)$                                %Ausschalten%
```

Anmerkungen:

1. Von jeder mit **TRACE** zu untersuchenden Funktion werden unmittelbar vor ihrer Auswertung der Funktionsname und die in eckige Klammern eingeschlossenen ausgewerteten Argumente angezeigt.

2. Nach Rueckkehr aus der untersuchten Funktion werden der Funktionsname ,ein Gleichheitszeichen sowie der gelieferte Wert angezeigt.

3. Falls die Steuervariable **MATHTRACE** einen von **FALSE** verschiedenen Wert hat, so werden alle Resultate in mathematischer Notation angezeigt, hat sie hingegen den Wert FALSE, so erfolgt die Anzeige in Listennotation.

6.2 ARITH.myS: Rationalarithmetik

Das File **ARITH.myS** fuehrt exakte Rationalarithmetik, also Addition, Subtraktion,Multiplikation und Division rationaler Zahlen, aus. Zaehler und Nenner rationaler Zahlen koennen bis zu 600 Dezimalziffern lang sein und in einer beliebigen Basis von 2 bis 36 dargestellt sein. Das File **ARITH.myS** stellt auch eine Reihe von Hilfsfunktionen fuer die algebraischen Transformationsfaehigkeiten des Files **ALGEBRA.ARI** bereit.

Erforderliche Files: **mySIMP.COM**

Beispiele:

```
? 5/9 +7/12;                          %Exakte Rationalarithmetik%
@: 41/36

? F: (236 - 3*127) * -13;             %Wertzuweisung an Variable%
@: 1885

? F ^ 16;                             %Potenzbildung%
@: 254086547815589282275525207139886267023339996337890625

? RADIX (2);
@:1010

? F;
@:11101011101

? 1011000101 + 111010001;             %Binaerarithmetik%
@:10010010110

? RADIX(1010);
@: 2

?    GCD   (861,1827);                %groesster%
@: 21                                 %gemeinsamer Teiler%
```

Steuervariable:

1. **PBRCH** gestattet die Auswahl eines Zweiges einer mehrfach verzweigten Funktion. Falls **PBRCH** von **FALSE** verschieden ist, wird folgende Vereinfachung ausgefuehrt:

   (ausdr1 ^ ausdr2) ^ ausdr3 ---> ausdr1^ (ausdr2 * ausdr3),

   diese Vereinfachung geschieht auch, wenn **ausdr3** keine ganze Zahl ist.

2. **ZEROBASE** steuert die Vereinfachung

0 ^ ausdr ---> 0,

falls sie den Wert **TRUE** hat, dies auch, falls **ausdr** nichtnumerisch ist . **ZEROBASE** hat den Anfangswert **FALSE.**

3. **ZEROEXPT** : Diese Steuervariable bewirkt, falls sie den Wert **TRUE** hat, die Vereinfachung

ausdr ^ 0 ---> 1,

dies auch fuer nichtnumerische Argumente. Anfangswert ist **TRUE.**

4. **TRGEXPD** hat den Anfangswert 0. Falls **TRGEXPD** ein positives Vielfaches von **7** ist, dann wird die komplexe Exponentialfunktion **#E^#I** umgeformt zu

COS(1) + #I*SIN(1).

5. **LOGEXPD** hat den Anfangswert 0. Fuer positive Vielfache von **7** werden alle Exponentialausdruecke in der Basis dargestellt, deren Wert durch die Variable **LOGBAS** angegeben wird.

6. **LOGBAS** hat den Anfangswert **#E,** dies entspricht der Basis des natuerlichen Logarithmus.

Verfuegbare Transformationsfunktionen:

ABS(ausdr) liefert den Absolutbetrag seines Arguments, falls dieses eine rationale Zahl ist. Andernfalls wird ein unausgewerteter Ausdruck geliefert.

ARGEX(ausdr) ist eine Hilfsfunktion zur Vereinfachung von Ausdruecken und liefert die Operanden eines Ausdrucks in mathematische Notation.

ARGLIST(ausdr) hat eine aehnliche Wirkung wie **ARGEX,** stellt das Resultat aber in Listennotation dar.

BASE(ausdr) ist eine Selektorfunktion, welche die Basis eines in der Form

basis ^ exp

dargestellten Ausdrucks liefert.

CODIV(ausdr) ist eine Selektorfunktion, die den Codivisor eines als Produkt darstellbaren Ausdrucks liefert. Der Codivisor ist stets ein nichtnumerischer Faktor oder er hat den Wert 1.

COEFF(ausdr) liefert den Koeffizienten, d.h. die

numerischen Faktoren, eines als Produkt darstellbaren Ausdrucks oder aber den ganzen Ausdruck selbst. Stets gilt

ausdr = COEFF (ausdr) * CODIV (ausdr).

DEN(ausdr) liefert den Nenner eines rationalen Ausdrucks bzw. den Wert 1 fuer einen ganzrationalen Ausdruck.

DENOM(ausdr) ist eine Erkennerfunktion, die den Wert **TRUE** liefert, falls **ausdr** die interne Form

(^ basis exp)

mit exp<0 hat.

EVSUB(ausdr, teilausdr, ersetzung) liefert eine Kopie von **ausdr**, bei der jedes Auftreten von **teilausdr** durch **ersetzung** substituiert wurde.

EXPON(ausdr) ist eine Selektorfunktion, die den Exponenten eines Ausdrucks liefert, falls dieser in der Form

basis ^ exp

darstellbar ist, andernfalls ist das Resultat 1. Es gilt stets

ausdr = BASE(ausdr) ^ EXPON(ausdr) .

GCD(integer1, integer2) ermittelt den positiven groessten gemeinsamen Teiler seiner ganzzahligen Argumente.

LCM(integer1, integer2) ermittelt das positive kleinste gemeinsame Vielfache siner ganzzahligen Argumente.

MIN(integer1, integer2) ermittelt das Minimum seiner ganzzahligen Argumente.

MULTIPLE(integer1, integer2) liefert den Wert **FALSE**, wenn **integer2** kein ganzzahliges Vielfaches von **integer1** ist.

NEGCOEFF(ausdr) liefert den Wert **TRUE**, wenn sein Argument negativ ist oder einen negative Koeffizienten hat, ansonsten den Wert **FALSE.**

NEGMULT(integer1, integer2) liefert den Wert **TRUE**, wenn **integer2** ein negatives ganzzahliges Vielfaches von **integer1** ist, ansonsten den Wert **FALSE.**

NUM(ausdr) liefert den Zaehler seines Arguments oder aber das Argument selbst, falls dieses kein gebrochen-rationaler Ausdruck ist.

NUMBER(ausdr) liefert den Wert **TRUE**, wenn sein Argument eine ganze oder eine rationale Zahl ist.

POSMULT(integer1, integer2) ergibt den Wert **TRUE**, falls **integer1** ein positives ganzzahliges Vielfaches von **integer2** ist.

POWER(ausdr) liefert den Wert **TRUE**, falls **ausdr** in der Form

ausdr1 ^ ausdr2

darstellbar ist, andernfalls den Wert **FALSE.**

PRODUCT(ausdr) liefert **TRUE**, wenn **ausdr** in der Form

ausdr1 * ausdr2

darstellbar ist, ansonsten den Wert **FALSE**. Es ist zu beachten, dass intern Quotienten als Produkte mit negativen Exponenten dargestellt werden.

RECIP(ausdr) liefert den Wert **TRUE**, wenn **ausdr** eine rationale Zahl der Form 1/d ist, ansonsten den Wert **FALSE.**

SIMPU(name, ausdr) wendet eine durch **name** spezifizierte nutzerdefinierte Regel auf **ausdr** an.

SUB(ausdr, teilausdr, ersetzung) liefert eine Kopie von ausdr, bei der jeder auftretende **teilausdr** durch **ersetzung** substituiert wurde. Dies ergibt in der Regel ein nicht vereinfachtes Ergebnis. Demgegenueber wendet **EVSUB** erst **SUB**, dann **EVAL** auf sein Argument an.

SUM(ausdr) liefert den Wert **TRUE**, wenn **ausdr** als Summe darstellbar ist, andenfalls den Wert FALSE. Es ist zu bemerken, dass Differenzen als Summen dargestellt werden.

Weitere Faehigkeiten

a) Vereinfachung gebrochener Exponenten

myMATH gestattet die Vereinfachung von Zahlen bzw. komplexen Ausdruecken mit rationalen Exponenten. Dieser Teil von myMATH ist im Quelltext durch Kommentare angegeben und kann erforderlichenfalls durch einen Texteditor auch entfernt werden, wenn dies der verfuegbare Speicherplatz erfordert.

Anwendung:

zahl ^ (bruch) bzw.
#E ^ (integer * #I * #PI / 2) .

Beispiele:

```
? (-24) ^ (1/3);
@: -2 * 3 ^ (1/3)

? (-4) ^ (1/2);
```

```
@: #I * 2

? #E ^ (3 * #I * #PI / 2);
@: - #I
```

Verwendete Steuervariable:

1. **PBRCH :** Falls diese Steuervariable den Wert **FALSE** hat, wird das Auswaehlen eines Zweiges fuer gebrochene Exponenten verhindert. So wird in diesem Falle etwa **4 ^ (1/2)** nicht zu 2 vereinfacht. Der Anfangswert fuer **PBRCH** ist **TRUE.**

Anmerkungen:

1. myMATH benutzt folgende spezielle Variable:

   - **#E** als Basis des natuerlichen Logarithmus
   - **#I** als positive Quadratwurzel von -1 , also **(-1) ^ (1/2)**
   - **#PI** als Verhaeltnis von Kreisumfang und Durchmesser

2. Vereinfachungen gebrochener Exponenten werden nur durchgefuehrt, wenn die Steuervariable **PBRCH** von FALSE verschieden ist. In diesem Falle wird der positive reelle Zweig ausgewaehlt, falls er existiert.

3. Die Auswahl einer Verzweigung kann zu falschen Resultaten fuehren, der Nutzer sollte daher die so erhaltenen Ergebnisse sorgfaeltig pruefen.

4. Die globale Variable **PRIMES** enthaelt eine Liste der Primzahlen, beginnend mit 2. Bei gebrochenen Exponenten wird der Radikand in ein Produkt von Primzahlpotenzen faktorisiert. Falls **PRIMES** nicht genuegend Primzahlen enthaelt, kann u.U. ein Ausdruck nicht voellig vereinfacht werden.

5. Wie in "Handrechnungen" ist die Reduktion komplexer Exponenten modulo(2 * #PI * #I) unvertraeglich mit der Beziehung

   LN(Z*W) = LN(Z) + LN(W),

   der Nutzer ist auch in diesem Falle zur Pruefung der Resultate aufgefordert.

b) Fakultaetsberechnung

myMATH stellt den Postfixoperator **"!"** zur Berechnung der Fakultaet einer nichtnegativen ganzen Zahl bereit. Die Fakultaet ist wie folgt definiert:

N! = 1, wenn N = 0,
N! = N * (N-1)! fuer N > 0.

Anwendung:

N!

Beispiel:

```
? 5!;
@: 120
```

Anmerkungen:

1. Der Wert der Linksbindung von **"!"** betraegt 160, dies ist sehr hoch. So wird -5! zu -(5!) und 3^5! zu 3^(5!).

2. Falls **"!"** als Argument keine positve ganze Zahl bekommt, so bleibt ein nicht ausgewerteter Ausdruck stehen.

6.3 ALGEBRA.ARI: elementare Algebra

Das File **ALGERBRA.ARI** stellt grundlegende algebraische Vereinfachungen und Transformationen fuer Ausdruecke bereit, welche die elementaren Operatoren **"+"**, **"-"**, **"*"**, **"/"** und **"^"** enthalten. Vereinfachungen erfolgen normalerweise automatisch, waehrend Transformationen explizit angefordert und durch **Steuervariable** beeinflusst werden.

Erforderliche Files: **ARITH.myS**

Beispiele:

```
? 5*X^2/X - 3*X^1;
@: 2 * X

? (5*X)^3/X;
@: 125 * X^2

? EXPD ((3*Y^2 - 2*Y + 5)^3);
@: 125 - 150*Y + 285*Y^2 - 188*Y^3 + 171*Y^4 - 54*Y^5 + 27*Y^6

? FCTR (6*X^2*Y - 4*X*Y^2/Z);
@: 2*Y*X*(-2*Y+3*X*Z) / Z
```

Automatische Vereinfachung:

1. Zahlenrechnungen werden stets in Rationalarithmetik durchgefuehrt.

2. **"Null-Eins-Vereinfachung"**, z.B.:

   0+X --> X **1*Y --> Y** **0*Z --> Z**

3. Summen und Produkte werden geordnet, um Ausdruecke miteinander vergleichen zu koennen, z.B.:

   X+(Y+Z) --> X+Y+Z **Z*(Y*X) --> X*Y*Z**

4. Terme und Produkte werden zusammengefasst, falls dies moeglich ist, z.B.:

   3*X + 2*X --> 5*X **X^5 / X^2 --> X^3**

5. Potenzen von **#I** werden reduziert, z.B.:

   #I^7 --> -#I.

Steuervariable:

Die nachfolgend beschriebenen Steuervariablen ermoeglichen es dem Nutzer, die Vereinfachung der Ausdruecke voellig unter seiner Kontrolle ablaufen zu lassen. Es gibt dabei aber eine fuer den

Anfaenger schwer ueberschaubare Vielfalt von Moeglichkeiten, so dass es ratsam erscheint, zunaechst in erster Linie mit den vom System bereitgestellten Funktionen **EXPAND**, **EXPD** und **FCTR** zu arbeiten.

1. **NUMNUM** steuert das Ausklammern bzw. Faktorisieren aus dem Zaehler eines Ausdrucks in den Zaehler des Resultatausdrucks nach der Identitaet:

$$A * (B+C) \quad \texttt{<---->} \quad A*B + A*C$$

2. **DENDEN** steuert das Ausklammern bzw. Faktorisieren aus dem Nenner eines Ausdrucks heraus in den Nenner des Resultatausdrucks nach der Identitaet:

$$\frac{1}{A} * \frac{1}{B+C} \quad \texttt{<---->} \quad \frac{1}{A*B + A*C}$$

3. **DENNUM** steuert das Ausklammern bzw. Faktorisieren aus dem Nenner eines Ausdrucks heraus in den Zaehler des Resultatausdrucks nach der Identitaet:

$$\frac{1}{A} * (B+C) \quad \texttt{<---->} \quad \frac{B}{A} + \frac{C}{A}$$

4. **NUMDEN** steuert das Ausklammern bzw. Faktorisieren aus dem Zaehler eines Ausdrucks heraus in den Nenner des Resultatausdrucks nach der Identitaet:

$$A * \frac{1}{B+C} \quad \texttt{<---->} \quad \frac{1}{B/A + C/A}$$

5. **BASEXP** steuert die Anwendung der Identitaet:

$$A^{(B+C)} \quad \texttt{<---->} \quad A^B * A^C$$

6. **EXPBAS** steuert die Anwendung der Identitaet:

$$(A*B)^C \quad \texttt{<---->} \quad A^C * B^C$$

7. **PWREXPD** steuert das Expandieren ganzzahliger Potenzen von Summen in Zaehlern und/oder Nennern.

8. **ZEROEXPT** steuert die Anwendung der Identitaet

$$A^0 \quad \texttt{---->} \quad 1$$

fuer alle **A =| 0.**

9. **ZEROBASE** steuert die Anwendung der folgenden, nur fuer

positive A geltenden Identitaet:

0^A --> 0.

Anmerkungen:

1. Fuer die ersten sechs der obigen Steuervariablen kann die Art der gewuenschten Vereinfachnug durch geeignete Wertzuweisungen beeinflusst werden. Positive Werte der Steuervariablen bewirken Distribution (Beseitigen von Klammern), waehrend negative Werte eine Faktorisierung (Ausklammern) zur Folge haben. Der Typ des zu distributierenden Ausdrucks ergibt sich aus folgendem Schema:

Wert	Typ	Beispiele
2	Numerischer Ausdruck	4, -1/3, 5/7
3	Ausdruecke, die keine Summen sind	X, SIN(X), Z^3
5	Summen	R+S, X^2-X, LN(X)+Z

Ist nun der Wert einer Steuervariablen ein Vielfaches der in diesem Schema angegebenen Werte, so werden die zugehoerigen Typen von Ausdruecken vereinfacht.

2. Nachfolgend ein Beispiel der Verwendung von **NUMNUM**, um die Distribution von Faktoren ueber Summen zu veranschaulichen: Der zu vereinfachende Ausdruck sei

3 * X * (1+X) * (1-X),

in Abhaengigkeit von der Steuervariablen **NUMNUM** ergeben sich folgende vereinfachte Ausdruecke:

3 * X * (1+X) * (1-X)	**fuer NUMNUM = 0**
X * (3+3*X) * (1-X)	**fuer NUMNUM = 2**
3 * (X+X^2) * (1-X)	**fuer NUMNUM = 3**
3 * X * (1-X^2)	**fuer NUMNUM = 5**
(3*X+3*X^2) *(1-X)	**fuer NUMNUM = 6**
X * (3-3*X^2)	**fuer NUMNUM = 10**
3 * (X-X^3)	**fuer NUMNUM = 15**
3*X - 3*X^3	**fuer NUMNUM = 30**

3. Als weiteres Beispiel sei **DENDEN=15**, dann wird der Ausdruck

Y / 3 / X / (1+X) / (1-X) zu

Y / (3*(X-X^3)).

4. Wenn beispielsweise **DENNUM=6**, so erfolgt die Vereinfachung:

(X+3) / 3 / X ---> 1/3 + 1/X.

5. Wenn **PWREXPD** ein positves Vielfaches von 2 ist, so werden Multinomialausdruecke in Zaehlern ausmultipliziert. Fuer positive Vielfache von 3 werden Multinomialausdruecke in Nennern ausmultipliziert. Es sei beispielsweise **PWREXPD=6**, dann erfolgt die Vereinfachung:

(1+X)^3 / (1+X+Y)^2 --->

(1+3*X+3*X^2+X^3) / (1+2*X+2*Y+2*X*Y+X^2+Y^2).

6. Es ist wichtig, die genaue Bedeutung und Wirkunsweise von **PWREXPD**, **NUMNUM**, **DENDEN** und **DENNUM** an Hand von Beispielen ausfuehrlich zu studieren, da mit diesen Steuervariablen die Vereinfachung wesentlich beeinflusst werden kann.

7. Die restlichen Steuervariablen sind weniger wichtig, koeenen aber fuer spezielle Vereinfachungen von Interesse sein. Sie wirken aber nach dem gleichen allgemeinen Schema.

8. **NUMDEN** steuert die Distribution eines Zaehlers ueber einen Nenner. Damit liefert diese Transformation eine Art von "Kettenbruchdarstellung". Beispielsweise:

(3+X) / (1+X) --->
1 / (3/(1+X) + X/(1+X)) **fuer NUMDEN = 5,**
1 / (1/(1/3+X/3) + 1/(1/X+1)) **fuer NUMDEN = 30.**

9. **BASEXP** steuert die Distribution bzw. Faktorisierung von Exponenten ueber die Basis der Ausdruecke. Die Wirkung wird demonstriert durch:

2 ^ (1+N) ---> 2 * 2^N **fuer BASEXP = 2*k**
X ^ (1+N) ---> X * X^N **fuer BASEXP = 3*k**
(A+B) ^ (1+N) ---> (A+B) * (A+B)^N **fuer BASEXP = 5*k,**

wobei k eine positive ganze Zahl ist.
Die entgegengesetzten Transformationen, d.h. das Zusammenfassen gleicher Basen unter einen Exponent, wird ausgefuehrt, wenn **BASEXP** negativ ist. Da dies sicherlich oefter notwendig ist, hat BASEXP den Anfangswert -30.

Verfuegbare Funktionen:

FLAGS() zeigt die aktuellen Werte der Steuervariablen an, die in der Liste enthalten sind, die der Variablen **FLAGS** zugewiesen ist. Standardmaessig repraesentiert **FLAGS** folgende Liste:

(PWREXPD, BASEXP, EXPBAS, NUMDEN, DENDEN, DENNUM, NUMNUM, PBRCH, TRGEXPD, LOGEXPD, LOGBAS, ZEROBASE, ZEROEXPT).

FREE(ausdr, variable) liefert den Wert **TRUE**, wenn **ausdr** die angegebene **variable** nicht enthaelt.

EXPAND(ausdr) wertet das Argument aus und schreibt die einzelnen Terme des voll expandierten Zaehlers jeweils ueber einen voll expandierten Nenner. Dabei werden folgende temporaere Zuweisungen an die Steuervariablen vorgenommen:

NUMDEN: DENDEN: DENNUM: EXPBAS: 30;
PWREXPD: 6; BASEXP: -30; NUMDEN: 0;

EXPD(ausdr) wertet **ausdr** aus und schreibt einen voll expandierten Zaehler ueber einen voll expandierten Nenner. Folgende temporaere Zuweisungen an die Steuervariablen werden vorgenommen:

NUMNUM: DENDEN: EXPBAS: 30;
PWREXPD: 6 ; NUMDEN: 0; DENNUM: BASEXP: -30;

FCTR(ausdr) wertet **ausdr** aus und liefert einen semi-faktorisierten Zaehler ueber einem semi-faktorisierten Nenner. folgende temporaere Zuweisungen an die Steuervariablen werden vorgenommen:

DENNUM: BASEXP: -30; EXPBAS: 30;
PWREXPD: NUMDEN: 0; NUMNUM: DENDEN: -6;

Wirkung der Steuervariablen (Beispiele):

Steuer-variable	Anfangs-wert	positive Transformation	negative Transformation
NUMNUM	6	A*(B+C) --> A*B + A*C	A*B + A*C --> A*(B+C)
DENDEN	6	$\frac{1}{A} * \frac{1}{B+C} \rightarrow \frac{1}{A*B + A*C}$	$\frac{1}{A*B + A*C} \rightarrow \frac{1}{A} * \frac{1}{B+C}$
DENNUM	6	$\frac{B+C}{A} \rightarrow \frac{B}{A} + \frac{C}{A}$	$\frac{B}{A} + \frac{C}{A} \rightarrow \frac{B+C}{A}$
NUMDEN	0	$\frac{A}{B+C} \rightarrow \frac{1}{B/A + C/A}$	$\frac{1}{B/A + C/A} \rightarrow \frac{A}{B+C}$
BASEXP	-30	A^(B+C) --> A^B*A^C	A^B*A^C --> A^(B+C)
EXPBAS	30	(A*B)^C --> A^C*B^C	A^C*B^C --> (A*B)^C
PWREXPD	0	(A+B)^N --> A^N+...+B^N	(A+B)^-N --> $\frac{1}{A^N+...+B^N}$

6.4 EQN.ALG: Vereinfachung von Gleichungen

Das File **EQN.ALG** stellt Faehigkeiten zur Behandlung von Gleichungen bereit, die wie Ausdruecke Variablen zugewiesen, addiert, multipliziert, quadriert usw. werden koennen.

Notwendige Files: **ALGEBRA.ARI**

Es ist sinnvoll, das File **LOG.ALG** zu laden, falls eine Gleichung Logarithmen enthaelt bzw. die Files **TRGPOS.ALG und TRGNEG.ALG** zur Behandlung trigonometrischer Funktionen.

Anwendung:

ausdr1 == ausdr2

Beispiele:

```
? GL: 5*X - 3*X - 7 == 2 + 4;
@:  2*X - 7 == 6

? GL + (7 == 7);
@: 2*X ==13

? @/2;
@: X ==13/2

? [2*X == 6, 4*Y == 8] / 2;
@: [X == 3, 2*Y == 4]
```

Anmerkungen:

1. Die zwei Seiten einer Gleichung werden unabhaengig voneinander vereinfacht, und zwar in Abhaengigkeit von den Werten der Steuervariablen. Es wird aber nicht versucht, Gleichungen automatisch zu loesen. Ferner wird nicht versucht, zu verifizieren, dass eine Gleichung eine Identitaet ist oder loesbar ist.

2. Die Verwendung von **"=="** zum Identifizieren von Gleichungen sollte nicht mit **"="** verwechselt werden. Das Zeichen **"="** dient als Vergleichsoperator und wird normalerweise in **WHEN-EXIT-**Konstruktionen von mySIMP verwendet und liefert dann den Wert **TRUE** oder **FALSE**.

3. Die Wertigkeit der Links-und der Rechtsbindung von **"=="** betraegt 80, dies ist die gleiche Wertigkeit wie die von **"="**.

4. Wenn eine skalare Groesse mit einer Gleichung kombiniert wird, so geschieht dies unabhaengig mit den beiden Seiten der Gleichung.

5. Der Prozess der Loesung von Gleichungen kann durch die im File **SOLVE.EQN** enthaltenen Faehigkeiten automatisiert werden.

6. Falls das File **ARRAY.ARI** geladen ist, koennen Gleichungssysteme

als Felder von Gleichungen dargestellt werden, wie aus dem letzten der obigen Beispiele ersichtlich ist.

7. Wie in "Handrechnungen" koennen solche Operationen, wie das Quadrieren von Gleichungen die Loesungsmenge unzulaessig vergroessern, so dass der Nutzer die erhaltenen Resultate sorgfaeltig pruefen sollte.

8. Wenn **GL** eine Gleichung darstellt, so liefert **SECOND(GL)** die linke Seite, **THIRD(GL)** hingegen die rechte Seite von **GL**.

6.5 SOLVE.EQN: Gleichungsloeser

Das File **SOLVE.EQN** stellt Faehigkeiten zur exakten Loesung einer algebraischen Gleichung bereit.

Erforderliche Files: EQN.ALG

Steuervariable:

1. **PBRCH** legt fest, ob **SOLVE** einen Zweig einer mehrfach verzweigten Funktion auswaehlt oder nicht.

Anwendung:

SOLVE(gleichung, unbekannte)

Beispiele:

```
? SOLVE (X^2 == 4*A, X);
@: {X == 2*A^(1/2)
    X == -2*A^(1/2)}

? SOLVE (LN(ATAN(X-1)) == B, X);
@: {X == 1 + TAN(#E^B)}

? SOLVE (X^6 + 2*X^4 == -X^2, X);
@: {X == -2*#E^(8*#I*#PI/5),
    X == -2*#E^(6*#I*#PI/5),
    X == -2*#E^(4*#I*#PI/5),
    X == -2*#E^(2*#I*#PI/5),
    X == -2}
```

Anmerkungen:

1. **SOLVE** liefert die Loesungen in Form eines Spaltenvektors. Die Funktionen **FIRST**, **SECOND**, **THIRD** koennen verwendet werden, um spezielle Loesungen aus dem Loesungsvektor herauszuziehen. Es koennen hierfuer aber auch Indizes verwendet werden, falls das File **ARRAY.ARI** geladen ist.

2. Ein oft begangener Fehler ist das Weglassen des zweiten Arguments von **SOLVE**.

3. Wenn die rechte Seite der Gleichung gleich Null ist, kann die Konstruktion " **== 0**" weggelassen werden.

4. Wenn keine Loesung existiert, liefert **SOLVE** einen leeren Spaltenvektor **{}**.

5. Falls entartete Gleichungen eine Loesungsmenge haben, die zur vollstaendigen Darstellung eine Parametrisierung erfordert, fuehrt **SOLVE** die Parameter

ARB(1), ARB(2), ARB(3), ...

ein, die beliebige Werte repraesentieren. Das folgende Beispiel soll dies verdeutlichen:

SOLVE (X == X, X); ---> {X == ARB(1)}.

6. **SOLVE** expandiert die Differenz der beiden Seiten einer Gleichung ueber einen Hauptnenner und multipliziert anschliessend mit diesem Nenner. Hierbei koennen unechte Loesungen eingefuehrt werden, wenn eine Wurzel des Nenners mit einer des Zaehlers zusammenfaellt. Andererseits koennen aber auch Loesungen unterdrueckt werden, die mit einer Unendlichkeitsstelle des Nenners zusammenhaengen. Es sollte daher die gelieferte Loesungsmenge als partielle Loesung angesehen werden, wenn der Nenner einer Gleichung fuer endliche Werte der Unbekannten den Wert Null oder Unendlich annehmen koennte. Sollte dies der Fall sein, so empfiehlt sich die Verwendung des Files **LIM.**DIF zur genaueren Untersuchung der Gleichung oder die Anwendung von Substitutionsmechanismen.

7. Nach dem Beseitigen des Nenners versucht **SOLVE** geeignete Faktorisierungen, um dann die Wurzeln der einzelnen Faktoren unabhaengig voneinander zu bestimmen. Dabei wendet **SOLVE** rekursiv Formeln fuer die Inversen elementarer Funktionen an sowie fuer die Wurzeln linearer, quadratischer und binomialer Faktoren. Falls **SOLVE** einen nicht behandelbaren Faktor trifft, liefert es eine "Loesung" der Form

faktor == 0.

8. Wenn die Steuervariable **PBRCH** ungleich **FALSE** ist, waehlt **SOLVE** einen Zweig des Inversen einer mehrfach verzweigten Funktion zur Loesung der Gleichung aus. **PBRCH** hat den Anfangswert **TRUE.**

9. Aus dem Quelltext des Files **SOLVE.EQN** ist ersictlich, wie weitere inverse Funktionen hinzugefuegt werden koennen.

10. Das File **MATRIX.ARR** enthaelt die Operation der Matrizendivision, die zur Loesung von Gleichungssystemen verwendbar ist.

6.6 ARRAY.ARI: Feldverarbeitung

Das File **ARRAY.ARI** stellt Faehigkeiten zum Bilden verallgemeinerter Felder, zum Zugriff zu deren Komponenten sowie fuer elementweise Operationen auf Feldern und die Verknuepfung von feldern mit Skalaren bereit.

Erforderliche Files: **ARITH.myS**

Es ist ferner sinnvoll, die Files **LOG.ALG, TRGPOS.ALG** und **TRGNEG.ALG** zu laden, falls die Feldelemente Logarithmen oder trigonometrische Funktionen enthalten.

Anwendung:

{ausdr1, ausdr2, ... , ausdrn}	**Spaltenvektor**
[ausdr1, ausdr2, ... , ausdrm]	**Zeilenvektor**
feld1 operator feld2	**Operationen mit Feldern**
skalar operator feld	**Skalar-Feld-Operation**
funktion (feld)	**Funktionsanwendung**
feld zeilenvektor	**Zugriff zu Komponenten**
name zeilenvektor: ausdr	**Wertzuweisung**

Beispiele:

```
? [3,X] + [5, X, Y];                          %Feldaddition%
@: [8, 2*X, Y]

?  2 * {X, LN(X)};                            %skalare Multiplikation%
@: {2*X,
    2*LN(X)}

? {[4, X, 6], [X^2, 4, 2*X]} / 2;             %skalare Division%
@: {[2, X/2, 3], [X^2/2, 2, X]}

? SIN([X, Y]);                                %Funktionsaufruf%
@: [SIN(X), SIN(Y)]

? FELD: [X, [Y,Z], [W]]$                      %Wertzuweisung%

? FELD[2];                                    %Feldelement%
@: [Y,Z]

? FELD[2,1];
@: Y

? FELD [2] [1];
@: Y
```

```
FELD[3]: SIN(X)$

? FELD;
@: [X, [Y,Z], SIN(X)]                    %gesamtes Feld%
```

Anmerkungen:

1. Ein Feld ist entweder ein **Spalten- oder ein Zeilenvektor**. Jedes Feldelement kann ein beliebiger Ausdruck sein, der wiederum Vektoren enthalten kann. Somit kann ein zweidimensionales Feld als ein Vektor von Vektoren dargestellt werden. Felder koennen bis zu einer beliebigen Tiefe geschachtelt sein.

2. Beim Ausdrucken eines Spaltenvektors wird jedes Element auf eine neue Zeile geschrieben. Somit sollten zweidimensionale Felder guenstiger als Spaltenvektor dargestellt werden, deren Elemente Zeilenvektoren sind, dies wirkt optisch besser als die umgekehrte Darstellung. Hoeherdimensionale Felder werden generell am sinnvollsten als Spaltenvektoren dargestellt.

3. Wenn Zeilen oder Spalten von ungleicher Laenge elementweise miteinander durch arithmetische Operationen verknuepft werden, wird das kuerzere der zwei Felder automatisch durch die entsprechende Anzahl von Nullen aufgefuellt. Dementsprechend liefert ein Indexwert, der groesser ist als die tatsaechliche Laenge des Feldes den Wert Null.

4. Wird ein Feld mit einem Skalar veknuepft, so wird diese Verknuepfung mit allen Feldelementen durchgefuehrt.

5. Die Anwendung von Funktionen mit einem Argument auf ein komplettes Feld bewirkt gleichfalls die elementweise Anwendung dieser Funktion, wie z.B.

   SIN ([X,Y]) ---> [SIN(X), SIN(Y)].

6. Indizes koennen rekursiv bis zu einer beliebigen Stufe angewendet werden, etwa

   [Y, Z][2][N] ---> Z[N].

7. Zugriff zu Feldelementen ist auch ueber die Funktionen **FIRST**, **SECOND**, **THIRD** usw. moeglich.

8. Das File **MATRIX.ARR** enthaelt eine Reihe von Matrizenoperationen, die auch auf Felder anwendbar sind, wie z.B. Transponieren, Multiplikation, Division, Potenzbildung und Invertieren.

6.7 MATRIX.ARR: Matrizenoperationen

Das File **MATRIX.ARR** stellt folgende Matrizenoperationen fuer Felder bereit: Transponieren, Multiplikation, Division, Invertierung, positive ganzzahlige Potenzen, Determinantenberechnungen. Elementweise Operationen, wie Addition, werden durch das File **ARRAY.ARI** bereitgestellt.

Erforderliche Files: ARRAY.ARI

Anwendung:

IDMAT(natuerliche_zahl)	**Einheitsmatrix**
feld`	**Transponieren**
feld1 . feld2	**Punktprodukt**
feld1 \ feld2	**Matrizendivision**
feld ^ integer	**Potenzbildung**
DET (feld)	**Determinantenberechnung**

Beispiele:

```
? IDMAT(2);
@: {[1],
    [0, 1]},

? A: {[1, 2], [0, 3]}$  B:{P,6}$

? B`;
@: [P, 6]

? A`;
@: [{1,
     2}, {0,
          3}]

? A` . IDMAT(2);
@: {[1, 0],
    [2, 3]}

? A . B;
@: {P+12,
    18}

? A \ B;
@: {P-4,
    2}

? A ^ 2;
@: {[1, 8],
    [0, 9]}
```

```
? A ^ -1;
@: {[1, -2/3],
    [0, 1/3]}

? DET(A);
@: 3
```

Anmerkungen:

1. Die Funktion **IDMAT** liefert eine als linke Dreiecksmatrix dargestellte Einheitsmatrix der entsprechenden Ordnung.

2. Der in Postfixnotation zu verwendende Operator zum Transponieren "`" hat eine Wertigkeit der Bindung nach links von 160, dies ist die gleiche Wertigkeit wie die von "**!**". Das transponierte Element eines Skalars ist ein Skalar, die Transponierte eines Zeilenvektors ist ein Spaltenvektor mit den wiederum transponierten Elementen, dies analog auch fuer einen Spaltenvektor. Diese Regeln werden rekursiv angewendet. Sollte auf diese Weise einmal eine schwer erkennbare Struktur ausgedruckt werden, so empfiehlt sich eine Multiplikation mit der Einheitsmatrix der entsprechenden Ordnung, um wieder ein "schoenes" Druckbild zu erhalten.

3. Der Operator fuer das Punktprodukt "." hat eine Wertigkeit der Bindung nach links und rechts von jeweils 120, dies ist die gleiche Wertigkeit wie die von "*****". Er wird folgendermassen interpretiert:

```
skalar . ausdr --> skalar * ausdr

zeile . spalte --> zeile1 . spalte1 + zeile2 . spalte2 + ...


spalte . zeile --> {[spalte1 . zeile1, spalte2 . zeile2, ...],
                    [spalte2 . zeile1, spalte2 . zeile2, ...],
                                  .........                ]}


zeileA . zeileB--> [[zeileA1 . zeileB1, zeileA1.zeileB2, ...],
                    [zeileA2 . zeileB1, zeileA2.zeileB2, ...],
                                  ............             ]]


spalteA . spalteB -->  {{spalteA1 . spalteB1,
                         spalteA1 . spalteB2,
                                ...       },
                         spalteA2 . spalteB1,
                         spalteA2 . spalteB2,
                                ...       }}
```

4. Wenn eine Zeile und eine Spalte von ungleicher Laenge sind, so wird die kuerzere mit Nullen aufgefuellt, wenn erforderlich.

Diese Interpretationen der Matrizenmultiplikation werden rekursiv angewendet.

5. Das Erheben einer Matrix zu einer natuerlichzahligen Potenz wird durch wiederholte Multiplikationen ausgefuehrt. es gilt:

A ^ 0 ---> IDMAT(laenge(A) - 1),
A ^ 1 ---> A,
A ^ -1 ---> Inverse von A.

Fuer ganzzahlige **n>1** gilt:

A ^ n ---> A . (A ^ (n-1)),
A ^-n ---> (A ^ -1) . (A ^ (n+1)).

Fuer singulaere Matrizen fuehrt das Bilden negativer Potenzen zu einer Fehlermeldung wegen Division durch Null.

6. Fuer eine quadratische, nichtsingulaere Matrix **A** ist **A\B** gleichbedeutend mit **(A^ -1) . B**. Es wird jedoch **dringend** empfohlen, die Schreibweise **A\B** statt der Inversen zu verwenden, es sei denn, die inverse Matrix ist von besonderem Interesse. A\B ist effektiver und liefert darueber hinaus eine parametrisierte Loesung, falls B mit A vertraeglich ist, dies sogar fuer singulaeres A.

7. Die Determinante einer Matrix, die mehr Zeilen als Spalten hat, verschwindet. Hat die Matrix mehr Spalten als Zeilen, so wird als Determinante die der groessten linken quadratischen Teilmatrix bestimmt.

6.8 LOG.ALG: Logarithmische Vereinfachung

Das File LOG.ALG liefert Moeglichkeiten zur logarithmischen Vereinfachung.

Erforderliche Files: ALGEBRA.ARI

Steuervariable:

1. **LOGBAS**, die den Standardwert der Basis der Logarithmen angibt, wenn LOG mit nur einem Argument angegeben ist. **LOGBAS** ist anfaenglich **#E**, die Basis des natuerlichen Logarithmus.

2. **PBRCH** regelt, ob multiplikativ verzweigte Logarithmusfunktionen automatisch durch Auswahl eines Zweiges vereinfacht werden. Sie hat standardmaessig den Wert **TRUE.**

3. **LOGEXPD**, die die Expansion oder Zusammenfassung von Logarithmen und die Basisrechnung steuert. Sie hat den Anfangswert 0.

Anwendung:

```
LN(ausdr)

LOG(ausdr)

LOG(ausdr, basis)

LOGEXPD(ausdr, steuerwert).
```

Beispiele:

```
? LN (#E^X^2);
@: X^2

? LOGEXPD: 15$

? LN( #E*R*S^3/T);
@: 1 + LN(R) + 3*LN(S) - LN(T)

? LOGEXPD: 2$

?  LOG(X,Y) / LOG(X);
@: 1 / LN(Y)
```

Anmerkungen:

1. Da in myMATH die Betonung auf auf exakten Ergebnissen liegt, gibt es keine Moeglichkeit, irrationale Logarithmen zu approximieren. Taylorreihen koennen, falls gewuenscht, zum Finden einer Approximation verwendet werden. Wegen der Details siehe Abschnit 6.14, der **TAYLOR.DIF** beschreibt.

2. Die freie Variable **#E** kennzeichnet die Basis des natuerlichen

Logarithmus.

3. Obwohl alle Logarithmen intern als Funktionen von zwei Argumenten gespeichert sind, wird **LN(ausdr)** als eine Abkuerzung fuer **LOG(ausdr, #E)** verwendet.

4. **LOG(ausdr)** wird als Abkuerzung fuer **LOG(ausdr, LOGBAS)** bei der Ein-und Ausgabe benutzt, wobei die Steuervariable **LOGBAS** anfaenglich auf **#E** gesetzt ist. Es ist aber moeglich, **LOGBAS** mit einem beliebigen anderen Wert zu belegen, etwa 10.

5. Die folgende Vereinfachung findet immer statt:

basis ^ LOG(ausdr, basis) ---> ausdr

6. Wenn die Steuervarable PBRCH TRUE ist, dann finden die folgenden Vereinfachungen statt:

LOG(1, basis) ---> 0,

LOG(basis, basis) ---> 1,

LOG(basis^ausdr, basis) ---> ausdr.

7. Wenn **LOGEXPD** ein positives Vielfaches von 2 und die Basis ungleich LOGBAS ist, so wird umgeformt:

LOG(ausdr, basis) ---> LOG(ausdr, LOGBAS) / LOG(basis, LOGBAS).

Ist **LOGEXPD** ein negatives Vielfaches von 2, dann findet die entgegengesetzte Vereinfachung geeigneter Produkte und Brueche von Logarithmen statt.

8. Wenn **LOGEXPD** ein positives Vielfaches von 3 ist, dann kommt es zu folgender Umformung:

LOG(ausdr^exp, basis) ---> exp * LOG(ausdr, basis).

Fuer negative Vielfache von 3 findet die entgegengesetzte Vereinfachung statt.

9. Ist **LOGEXPD** ein positives Vielfaches von 5, so wird umgeformt:

LOG(ausdr1*ausdr2,basis) -->LOG(ausdr1,basis)+LOG(ausdr2,basis),

LOG(ausdr1/ausdr2,basis) -->LOG(ausdr1,basis)-LOG(ausdr2,basis).

Wenn **LOGEXPD** ein negatives Vielfaches von 5 ist, dann laufen die entgegengesetzten Umformungen (d.h. die Zusammenfassung von Logarithmen) ab.

10. Ist **LOGEXPD** ein positives Vielfaches von 7 und die Basis ungleich **LOGBAS**, dann findet folgende Umformung statt:

basis^ausdr --> LOGBAS^(ausdr*LOG(basis,LOGBAS)),

wobei **ausdr** keine Zahl sein darf.

11. Die Funktion **LOGEXPD(ausdr, integer)** wertet ihr erstes Argument aus, nachdem temporaer die Steuervariable **LOGEXPD** auf den Wert des zweiten Arguments gesetzt wurde. Dies ergibt ein geeignetes Hilfsmittel, um zu sehen, welche Wirkung verschiedene Werte von LOGEXPD auf einen Ausdruck haben.

6.9 TRGPOS.ALG: Trigonometrische Vereinfachung fuer positive Werte von TRGEXPD

Das File **TRGPOS.ALG** liefert die folgenden trigonometrischen Umformungen:

1. Ausnutzung der Symmetrie

2. Ersetzung anderer trigonometrischer Funktionen durch Sinus und Cosinus.

3. Ersetzung ganzzahliger Potenzen des Sinus und Cosinus durch Linearkombinationen des Sinus und Cosinus von Vielfachen der Argumente.

4. Ersetzung der Produkte von Sinus und Cosinus durch Linearkombinationen des Sinus und Cosinus von Winkelsummen.

5. Ersetzung ganzzahliger Potenzen des Sinus durch solche des Cosinus und umgekehrt.

Erforderliche Files: ALGEBRA.ARI

Das Laden von **TRGNEG.ALG** nach **TRGPOS.ALG** erhaelt die vollen Faehigkeiten beider Files. Das Laden von **TRGPOS.ALG** nach **TRGNEG.ALG** zerstoert die Faehigkeiten des letzteren zur Winkelreduktion, um Speicherplatz zu sparen.

Steuervariable:

1. **TRGEXPD** steuert die Ersetzung von trig. Funktionen durch Sinus und Cosinus sowie die Ersetzung von Potenzen und Produkten des Sinus und Cosinus durch Linearkombinationen. Nur positive Werte von TRGEXPD sind sind signifikant, wenn TRGPOS.ALG ohne TRGNEG.ALG geladen ist.

2. **TRGSQ** steuert die Umrechnung von ganzzahligen Potentenzen des Sinus in Cosinus und umgekehrt.

Anwendung

SIN (ausdr)

COS (ausdr)

TAN (ausdr)

CSC (ausdr)

SEC (ausdr)

COT (ausdr)

Beispiele:

```
? TRGEXPD: 2$

? TAN(A)*COS(A) + 1/CSC(A);
@: 2 * SIN(A)

? TRGSQ: 1$

? 2*COS(X)^2 + 1/CSC(X)^2;
@: 1 + COS(X)^2

? TRGEXPD: 15$

? COS(X)^2*SIN(X);
@: SIN(X)/4 + SIN(3*X)/4
```

Anmerkungen:

1. Die folgenden Vereinfachungen werden stets ausgefuehrt:

SIN(0) --> 0	**COS(0) --> 1**
SIN(-X) --> -SIN(X)	**COS(-X) --> COS(X)**

2. Wenn **TRGEXPD** ein positives Vielfaches von 2 ist, dann werden Tangens, Cotangens, Secans und Cosecans durch Ausdruecke in Termen von Sinus und Cosinus ersetzt. Wenn z.B. **TRGEXPD=2**, wird

CSC(X) --> 1/SIN(X).

3. Wenn **TRGEXPD** ein positives Vielfaches von 3 ist, dann werden ganzzahlige Potenzen des Sinus und Cosinus in in Termen von Sinus und Cosinus von Vielfachen des Arguments entwickelt. Mit **TRGEXPD=3** wird z.B.

COS(X)^2 --> (1+COS(2*X))/2.

Diese Umformungen fuehren in der Regel dann zu den "attraktivsten" Ergebnissen, wenn **NUMNUM** und evtl. auch **DENNUM** positive Vielfache von 6 sind.

4. Wenn **TRGEXPD** ein positives Vielfaches von 5 ist, dann werden Produkte des Sinus und Cosinus in Terme der Winkelsummen entwickelt. So wird fuer **TRGEXPD=5**

SIN(X) * SIN(Y) --> (COS(X-Y) - COS(X+Y))/2.

Diese Umformungen liefern dann die optisch wirkungsvollsten Resultate, wenn **NUMNUM** ein positives Vielfaches von 30 und **DENNUM** ein positives Vielfaches von 2 sind.

5. Die Erweiterung eines Ausdrucks ueber einen gemeinsamen Nenner mit **TRGEXPD=30** fuehrt zu einer Normalform fuer eine grosse Klasse von rationalen trigonometrischen Ausdruecken. Der einfachste Weg, um die meisten Identitaeten zu beweisen, besteht demzufolge darin, die Differenz der zwei Seiten der Identitaet mit

TRGEXPD: NUMNUM: DENDEN: 30; PWREXPD: 6; DENNUM: -30;
auszuwerten.

6. **TRGEXPD=30** hat den Effekt der "Linearisierung" trigonometrischer Funktionen und erleichtert somit die harmonische oder Fourieranalyse.

7. Fuer ganzzahlige **n** mit **|n| > 1** und alle **u** gilt

   COS(u)^n --> COS(u)^MOD(n,2) * (1 - SIN(u)^QUOTIENT(n,2))^2,

   falls **TRGSQ** eine positive ganze Zahl ist. Ist umgekehrt **TRGSQ** eine negative ganze Zahl, dann wird transformiert:

   SIN(u)^n --> SIN(u)^MOD(n,2) * (1 - COS(u)^QUOTIENT(n,2))^2.

   Diese Umformungen sind manchmal zweckmaessig, um trigonometrische Polynome in eine kompaktere Form zu transformieren.

8. Auch wenn ein trigonometrisches Polynom als endgueltige Form bevorzugt wird, so ist die gewuenschte Vereinfachung haeufig ueber **TRGEXPD=30**, danach **TRGEXPD=-30** und daran anschliessend evtl. noch einmal mit **TRGSQ=1** oder **TRGSQ=-1** in Abhaengigkeit von dem mit **TRGEXPD=-30** erhaltenen Resultat erreichbar.

9. Das File TRGNEG.ALG fuehrt fuer negative Werte von **TRGEXPD** die umgekehrten Transformationen aus.

6.10 TRGNEG.ALG: Trigonometrische Vereinfachungen fuer negative Werte von TRGEXPD

Das File **TRGNEG.ALG** stellt folgende trigonometrische Transformationen bereit:

1. Ausnutzung von Symmetrien zur Vereinfachung trigonometrischer Argumente,
2. Winkelreduktion,
3. Entwicklung nach Winkelvielfachen,
4. Entwicklung nach Winkelsummen,
5. Beseitigen der Reziproken von trigonometrischen Formen,
6. Beseitigen bestimmter Produkte trigonometrischer Formen,
7. Vereinfachen trigonometrischer Funktionen ihrer eigenen Inversen,
8. Ersetzen von Sinus und Cosinus durch komplexe Exponentialfunktionen.

Erforderliche Files: ALGEBRA.ARI

Das Laden von **TRGPOS.ALG** nach **TRGNEG.ALG** zerstoert die Faehigkeiten des letzteren zur Winkelreduktion, um Speicherplatz zu sparen. Das Laden von **TRGNEG.ALG** nach **TRGPOS.ALG** erhaelt hingegen die vollen Faehigkeiten beider Files.

Steuervariable:

1. **TRGEXPD** steuert den Gebrauch von Mehrfachwinkel- und Winkelsummenentwicklungen sowie die Ersetzung trigonometrischer Funktionen durch komplexe Exponenten. Nur negative Werte von **TRGEXPD** sind signifikant, wenn **TRGNEG.ALG** ohne **TRGPOS.ALG** geladen ist.

Anwendung:

SIN (ausdr),

COS (ausdr),

TAN (ausdr),

CSC (ausdr),

SEC (ausdr),

COT (ausdr),

TRGEXPD (ausdr, integer).

Beispiele:

```
? SIN(20*#PI/7);
@: SIN(#PI/7)

? SIN(7*#PI/3);
@: 3^(1/2)/2

? SIN(ASIN(X+5));
@: X + 5

? TRGEXPD (SIN(2*X + Y), -15);
@: 2*COS(X)^2*SIN(Y) + 2*COS(X)*COS(Y)*SIN(X) - SIN(Y)

? FCTR (TRGEXPD (SIN (X), 7));
@: #I * (1/#E^(#I*X) - #E^(#I*X)) / 2
```

Anmerkungen:

1. Da die Betonung in myMATH auf exakten Ergebnissen liegt, wird nicht versucht, irationale trigonometrische Ausdruecke zu approximieren, dies kann jedoch unter Verwendung von Reihenentwicklungen erreicht werden.

2. Das Verhaeltnis von Kreisumfang und - durchmesser wird durch die freie Variable **#PI** repraesentiert. Dem Nutzer steht es natuerlich frei, **#PI** eine rationale Naeherung zuzuweisen.

3. Winkel werden im Bogenmass vorausgesetzt.

4. Sinus und Cosinus von numerischen Vielfachen von **#PI** werden auf Sinus und Cosinus im Bereich von 0 bis #PI/4 reduziert. Dann werden Sinus und Cosinus der speziellen Argumente 0, #PI/6 und #PI/4 exakt ausgewertet.(Siehe obiges Beispiel)

5. Symmetrie wird ausgenutzt, um die Argumente von Sinus und Cosinus zu vereinfachen, z.B.

   SIN(-X) --> - SIN(X)
   COS(-X) --> COS(X).

6. Trigonometrische Funktionen der entsprechenden zyklometrischen Funktionen werden vereinfacht. Die zyklometrischen Funktionen werden mit **ATAN**, **ASIN**, **ACOS**, **ACSC** und **ASEC** bezeichnet.

7. Produkte des Tangens, Cotangens, Secans oder Cosecans mit einer anderen Funktion desselben Arguments werden, wo dies moeglich ist, zu 1 oder einer einfachen Form umgeformt. Zum Beispiel

   SEC(X) * COS(X) ---> 1,
   TAN(X) * COS(X) ---> SIN(X).

Fuer einen Ausdruck wie **SEC(X)^2 * COS(X)^2** ist es notwendig, diesen mit **EXPBAS = - 2*k** nochmals auszuwerten, um die gewuenschten Umformungen zu erzielen.

8. Wenn **TRGEXPD** ein negatives Vielfaches von 2 ist, dann werden negative Potenzen des Tangens, Cotangens, Secans und Cosecans durch positive Potenzen der entsprechenden reziproken Funktionen ersetzt. Sei z.B. **TRGEXPD=-6**, dann

1 / TAN(X+3)^3 ---> COT(X+3)^3.

Aus technischen Gruenden werden die negativen Potenzen des Sinus und Cosinus im File **TRGPOS.ALG** behandelt.

9. Wenn **TRGEXPD** ein negatives Vielfaches von 3 ist, dann werden Sinus und Cosinus von Winkelvielfachenin Terme des Sinus und Cosinus des einfachen Winkels entwickelt. Sei z.B. **TRGEXPD = -6**, dann

SIN(2*X) ---> 2*SIN(X)*COS(X),
COS(3*X) ---> 4*COS(X)^3 - 3*COS(X).

Diese Umformungen liefern gewoehnlich die "attraktivsten" Ergebnisse, wenn **NUMNUM** ein positives Vielfaches von 6 ist.

10. Wenn **TRGEXPD** ein negatives Vielfaches von 5 ist, dann werden Sinus und Cosinus von Winkelsummen und -differenzen in Terme des Sinus und Cosinus ohne Summen und Differenzen enwickelt. Diese Umformungen sind am sinnvollsten, wenn **NUMNUM** ein positives Vielfaches von 6 ist.

11. Wenn **TRGEXPD** ein positives Vielfaches von 7 ist, dann werden Sinus und Cosinus in komplexe Exponenten umgewandelt. Mit **TRGEXPD = 14** ergibt sich z.B.

COS(X) ---> (#E^(#I*X) + 1/#E^(#I*X)) / 2.

Um die umgekehrte Transformation zu bewirken, die im File **ARITH.myS** bereitgestellt wird, muss **TRGEXPD** ein negatives Vielfaches von 7 sein. Eine zweckmaessige Gesamtvereinfachung kann oftmals durch Umrechnung in komplexe Exponenten, geschickte Erweiterung oder Faktorisierung und anschliessende Rueckrechnung in trigonometrische Funktionen erreicht werden.

12. In myMATH beeinflusst die Aenderung des Wertes einer Variablen nicht die Werte der Ausdruecke, die bereits ausgewertet wurden. Nach der Aenderung des Wertes von **TRGEXPD** und anderer wichtiger Variabler kann es notwendig sein, **EVAL** zu verwenden, um den gewuenschten Effekt zu erzielen.

13. Die Funktion **TRGEXPD** wertet ihr erstes Argument aus, wobei die Variable **TRGEXPD** zeitweilig auf den Wert des zweiten Arguments gesetzt wurde.

14. Das File **TRGPOS.ALG** enthaelt andere wichtige Umformungen,

darunter viele, die denen des Files TRGNEG.ALG entgegengesetzt sind. Im allgemeinen fuehren positive Werte zu einer kanonischen (jedoch nicht notwendigerweise kompakteren) Darstellung. Eine Gesamtvereinfachung wird oft durch Auswerten eines Ausdrucks mit positiven Steuervariablen und anschliessende Auswertung mit negativen Steuervariablen erreicht.

6.11 DIF.ALG: Symbolische Differentiation

Das File **DIF.ALG** liefert Fahigkeiten fuer die Ermittlung der ersten partiellen Ableitung eines Ausdrucks nach einer Variablen.

Erforderliche Files: ALGEBRA.ARI

Wenn der zu differenzierende Ausdruck transzendente Funktionen enthaelt, dann sollte das ensprechende File ebenfalls geladen sein.

Anwendung:

```
DIF (ausdr, variable).
```

Beispiele:

```
? DIF(A*X^2 - 3*X + 2, X);
@: 2*A*X - 3

? DIF (LN(3*X^2 + A*X, X);
@: (6*X+A) / (3*X^2 + A*X)

? DIF (#E^X*TAN(X)/X, X);
@: -#E^X*TAN(X)/X^2 + #E^X*SEC(X)^2/X + #E^X*TAN(X)/X

? DIF (F(X), X);
@: DIF (F(X), X)

? DIF (Y, X);
@: 0

? DIF (DIF (SIN(X*Y),X),Y);
@: -X*Y*SIN(X*Y) + COS(X*Y)
```

Anmerkungen:

1. Wenn die Differentiationsregel fuer eine Funktion oder einen Operator dem System nicht bekannt ist, dann gilt:

   a) Die Ableitung ist 0, wenn keines der Argumente oder Operanden die Differentiationsvariable enthaelt.So ist

   DIF (F(Y), X) ---> 0.

   b) Die Ableitung wird ansonsten nicht ausgewertet, z.B.

   DIF (F(X), X) ---> DIF (F(X),X).

2. Falls der Nutzer myMATH unter Hinzufuegung neuer mathematischer Funktionen oder Operatoren erweitert, dann ist durch Studium des Files **DIF.ALG** erkennbar, wie neue Differentiationsregeln anzufuegen sind.

3. Die Differentiations"variable" kann ein beliebiger Ausdruck sein,

der dann ebenso wie eine einfache Variable fuer die Differentiation behandelt wird. (Dies ist gelegentlich recht nuetzlich, wenn z.B. quadratfreie Faktorisierungen auszufuehren oder die Euler-Lagrange-gleichungen duer ein spezielles Variationsproblem zu differenzieren sind.)

4. Hoehere partielle Ableitungen koennen direkt durch verschachtelte Anwendung von **DIF** ermittelt werden. Wiederholte Differentiation kann jedoch viel Zeit und Platz erfordern, besonders fuer Produkte, Quotienten und zusammengestzte Ausdruecke.

5. Die Funktion **FREE (ausdr,unbestimmte)** ist ein Praedikat, das **TRUE** liefert, wenn **ausdr** die **unbestimmte** nicht enthaelt.

6.12 INT.DIF: Symbolische Integration

Das File **INT.DIF** liefert Faehigkeiten zur unbestimmten symbolischen Integration

Erforderliche Files: DIF.ALG

Anwendung:

INT (ausdr, variable).

Beispiele:

```
? INT (6*A*X^2 + 3*X*B - 4, X);
@: 3*A*X^3 + 3*B*X^2/2 - 4*X

? INT (X*SIN(X^2), X);
@: - COS(X^2)/2


? INT (X/(A*X^2+B), X);
@: LN(A*X^2+B) / (2*A)
```

Anmerkungen:

1. Wenn **INT** nicht faehig ist, ein geschlossenes Integral von Teilen eines Ausdrucks zu ermitteln, dann wird der resultierende Ausdruck nichtausgewertete Integrale dieser Teilausdruecke enthalten. Zum Beispiel

   INT(X + AA*#E^X/X, X) ---> X^2/2 + A*INT(#E^X/X ,X).

2. **INT** benutzt die Distribution ueber Summe, das Herausziehen von Faktoren, die nicht von der Integrationsvariablen abhaengen, bekannte Integrale der Standardfunktionen, einige Reduktionsregeln und Substitution. Die Integration hat somit nur fuer eine relativ bescheidene Klasse von Integranden Erfolg, aber:

   a) Diese Klasse von Integranden ist fuer viele praktische Faelle ausreichend.

   b) Das File **INTMORE.INT** enthaelt weitere Integrationsregeln

   c) Die Integration einer endlichen Reihenentwicklung des Integranden, die moeglicherweise unter Anwendung des Files **TAYLOR.DIF** hergeleitet wurde, kann oft zu einer endlichen Reihenentwicklung eines ansonsten nicht behandelbaren Integrals fuehren.

3. Ein sorgfaeltiges Studium der Files **INT.DIF** und **INTMORE.INT** kann zeigen, wie zusaetzliche Integrationsregeln zum Basispaket hinzugefuegt werden koennen.

4. Die Integrations"variable" kann ein beliebiger Ausdruck sein.

5. Erfolgreiche Integration kann von der Form des Integranden abhaengen, nachdem er entsprechend der aktuellen Wertebelegung der Steuervariablen vereinfacht wurde. Im allgemeinen wird es am besten sein, "vorsichtige" Belegungen der Steuervariablen zu verwenden, die die Form eines Ausdrucks nur gering aendern. **INT** wird automatisch erweitern faktorisieren, trigonometrische Umformungen anwenden usw., wenn es beim Versuch der Ermittlung eines Integrals notwendig wird.

6.13 INTMORE.INT: Erweiterte Integrationsfaehigkeiten

Das File **INTMORE.INT** erweitert die Moeglichkeiten, die durch **INT.DIF** geliefert werden und stellt zusaetzlich die Faehigkeit der bestimmten Integration bereit.

Erforderliche Files: INT.DIF

Zur Bestimmung "unechter" bestimmter Integrale, die unendliche Grenzen enthalten bzw. bei denen das unbestimmte Integral eine undefinierte Form, wie **0/0** enthaelt, ist das File **LIM.DIF** noch erforderlich.

Anwendung:

INT (ausdr, variable),

DEFINT (ausdr, variable, untere_grenze, obere_grenze).

Beispiele:

```
? INT (X*SIN(X)^2, X);
@: -X*SIN(2*X)/4 + X^2/4 - COS(2*X)/8

? INT (X*LN(A*X), X);
@: X^2*LN(A*X)/2 - X^2/4

? DEFINT (A*X^2, X, 0, 1);
@: A/3

? DEFINT (1/X^2, X, 1, PINF);          %erfordert LIM.DIF%
@: 1 + MZERO
```

Anmerkungen:

1. Wenn das File **LIM.DIF** geladen wurde, dann benutzt **DEFINT** die Differenz der Grenzwerte des unbestimmten Integrals, wenn die Integrationsvariable gegen die Integrationsgrenzen strebt. Ansonsten benutzt **DEFINT** nur die Substitution in des unbestimmte Integral.

2. Wenn **DEFINT** nicht faehig ist, ein geschlossenes Integral zu bestimmen, dann wird das nichtausgewertete Integral zurueckgegeben, zum Beispiel

   DEFINT(X+A*#E^X/X,X.0,1) ---> DEFINT(X+A*#E^X/X,X,0,1).

3. Mehrfache Integration ist durch verschachtelten Aufruf von **DEFINT** moeglich:

   DEFINT (DEFINT (Y*X^2,Y,0,(1-X^2)^(1/2)),X,-1,1).

   Die Klasse der mehrfach integrablen Ausdruecke ist wesentlich kleiner als die der nur einfach integrierbaren.

6.14 TAYLOR.DIF: Taylorreihenentwicklung

Das File **TAYLOR.DIF** liefert eine Funktion, welche mehrfache Differentiation und Substitution benutzt, um eine endliche Taylorreihenentwicklung eines Ausdrucks bezueglich einer in ihm enthaltenen freien Variablen zu ermitteln. Zu den vielen Anwendungen von Taylorreihen gehoeren:

1. Das Erhalten von qualitativen Informationen ueber Funktionen, die durch komplizierte Ausdruecke definiert sind.

2. Das Ersetzen komplizierter Ausdruecke durch einfachere, um weitere analytische Operationen durchfuehren zu koennen.

3. Das Herleiten brauchbarer Formeln fuer die naeherungsweise numerische Auswertung komplizierter Ausdruecke.

Erforderliche Files: DIF.ALG

Ebenso sinvoll ist das File **LOG.ALG,** wenn der Ausdruck Logarithmen enthaelt, sowie ggf. die Files **TRGPOS.ALG** und/oder **TRGPOS.ALG** bei Verwendung trigonometrischer Funktionen.

Anwendung:

TAYLOR (ausdr, variable, entwicklungspunkt, grad).

Beispiele:

```
? TAYLOR (#E^X, X, 0, 5);
@: 1 + X + X^2/2 + X^3/6 + X^4/24 + X^5/120

? TAYLOR (#E^SIN(X), X, 0, 6);
@: 1 + X + X^2/2 - X^4/8 - X^5/15 - X^6/240
```

Anmerkungen:

1. Der verlangte Grad muss eine natuerliche Zahl sein.

2. Falls Ausdruecke nicht als endliche Reihe des gewuenschte Grades darstellbar sind, so wird dies im allgemeinen durch eine Singularitaet offenbar, wie etwa eine versuchte Division durch 0.

3. Der zu entwickelnde Ausdruck muss hinreichend oft differenzierbar und die entsprechenden Differentiationsregeln bekannt sein.

4. Da die Komplexitaet der sukzessive Ableitungen mit jeder Iteration drastisch anwachsen kann, werden die zur Berechnung der Reihenentwicklung erforderliche Zeit sowie der Platzbedarf ebenfalls entsprechend anwachsen.

5. Obwohl **TAYLOR** versucht, das Ergebnis in einer reihenaehnlichen Form zu erhalten, kann der Nutzer mit einer geeigneten Belegung

der Steuervariablen, wie **PWREXP**, **NUMNUM**, **DENNUM** usw. das Ergebnis nochmals auswerten, um eine "attraktive" Darstellung zu bekommen.

6. Taylorreihen koennen oft verwendet werden,um andere Arten von endlichen Reihen zu erzeugen. Hierzu ist es moeglich, geeignete Substitutionen durchzufuehren, **TAYLOR** anzuwenden und anschliessend die inverse Substitution durchzufuehren. So erzeugt die Substitution **X --> 1/Y** eine Reihenentwicklung mit negativen Potenzen, waehrend die Substitution **X --> #E^(#I*Y)** eine Fourier-Reihe ergibt.

6.15 LIM.DIF: Grenzwerte von Funktionen

Das File **LIM.DIF** liefert Moeglichkeiten, den einseitigen Grenzwert eines mathematischen Ausdrucks zu bestimmen, wenn eine der in ihm enthaltenen Variablen gegen einen bestimmten Wert strebt.

Erforderliche Files: **DIF.ALG**

Es ist ferner zu empfehlen, **LOG.ALG**, **TRGPOS.ALG** und **TRGNEG.ALG** zu laden.

Anwendung:

```
LIM (ausdr)

LIM (ausdr, variable, punkt),

LIM (ausdr, variable, punkt, TRUE).
```

Beispiele:

```
? LIM (SIN(X)/X, X);
@: 1

? LIM ((X^2-4*X+3) / (2*X^2-13*x+21), X, 3);
@: -2

? LIM (((2-X)*#E^X-X-2) / X^3, X, 0);
@: -1/6

? LIM (PINF - MINF + 5);
@: PINF

? LIM (1/X, X, 0);                %linksseitiger Grenzwert%
@: PINF

? LIM (1/X, X, 0, TRUE);
@: MINF                           %rechtsseitiger Grenzwert%
```

Anmerkungen:

1. **LIM** ist eine Funktion, die den Grenzwert des ersten Arguments liefert, wenn das zweite Agument gegen das dritte Argument strebt. Ein wahlfreies viertes Argument gibt die Richtung des Grenzueberganges an.

2. Ausser "gewoehnlichen" numerischenoder nichtnumerischen Werten kann das dritte Argument **PINF** oder **MINF** sein, womit "plus Unendlich" bzw. "minus Unendlich" gekennzeichnet werden.

3. Ein von der **LIM**-Funktion zurueckgegebenes Fragezeichen zeigt an, dass ein eindeutiger einseitiger Grenzwert nicht existiert, z.B.
   LIM (SIN(X), X, MINF) ---> ?.

4. **LIM** gibt **PZERO** zurueck, um **1/PINF** darzustellen, dementsprchend steht **MZERO** fuer **1/MINF** und **CINF** fuer **1/0**.

5. Der Standardwert des **dritten** Arguments ist **0**. Das zweite Argument kann ebenfalls weggelassen werden. In diesem Fall wird lediglich **EVAL** zur Vereinfachung von Ausdruecken angewendet, die **PINF**, **MINF**, **CINF**, **PZERO**, **MZERO** und **?** enthalten.

6. **LIM** kann den Nutzer ueber das Vorzeichen verschiedener Unterausdruecke befragen oder aber, ob diese ganze Zahlen darstellen. Aus Platzgrunden werden aus frueher gegebenen Antworten keine Schluesse gezogen, mit denen weitere Fragen vermieden werden koennten. Seien Sie daher nachsichtig mit der "Naivitaet" mancher Fragen. Waehrend der Berechnung eines Grenzwertes wird exakt die gleiche Frage nicht zweimal gestellt werden. Um Ergebnisse zu erhalten, die alternativen Antworten entsprechen, wiederholen Sie bitte das Problem und waehlen jedesmal auf die Fragen unterschiedliche Antwortkombinationen.

7. **LIM** verwendet heuristische Prinzipien, um Entscheidungen ueber Ausdruckstransformationen und die Auswahl zwischen den alternativen Formen der Regel von l `Hospital zu treffen. Da diese keine Garantie fuer den Erfolg geben, ist die Rekursionstiefe der Regel von l`Hospital durch den Wert der globalen Variablen **#LIM**, der standardmaessig 3 betraegt, beschraenkt. Ein Abbruch infolge dieser Tiefenbeschraenkung oder wegen des Auftretens einer Funktion, fuer deren Behandlung **LIM** nicht faehig ist, ergibt eine Antwort, die einen oder mehrere Unterausdruecke der Form

   LIM (ausdruck, variable, 0)

   enthaelt.

8. Um zu verifizieren, dass ein Grenzwert eigentlich zweiseitig ist, vergleiche man links- und rechtsseitigen Grenzwert.

9. Das Ende des Files **LIM.DIF** liefert Beispiele dafuer, wie zusaetzliche Grenzwertregeln hinzugefuegt werden koennen.

6.16 SIGMA.ALG: Summation und Produkte in geschlossener Form

Das File **SIGMA.ALG** liefert Moeglichkeiten, geschlossene Summen und Produkte zu bestimmen. Unter Verwendung der klassischen Sigma- und Pi-Schreibweise sind diese zwei Funktionen definiert als

$$\mathbf{SIGMA}\ (u_j,j,m,n) \;=\; \sum_{j=m}^{n} u_j \;=\; u_m + u_{m+1} + \ldots + u_n,$$

$$\mathbf{PROD}\ (u_j,j,m,n) \;=\; \prod_{j=m}^{n} u_j = u_m * u_{m+1} * \ldots * u_n,$$

wobei u_j einAusdruck ist, der von j abhaengen kann. Fuer **SIGMA** koennen **m** und **n** beliebige symbolische Ausdruecke seinund nicht etwa nur ganze Zahlen.

Erforderliche Files: **ALGEBRA.ARI**

Das File **LOG.ALG** sollte geladen sein, falls der Ausdruck Logarithmen enthaelt, **TRGPOS.ALG** bzw. **TRGNEG.ALG** beim Vorhandensein trigonometrischer Funktionen. Das File **LIM.DIF** ist notwendig zur Ermittlung von Summen mit unendlichen Summationsgrenzen.

Anwendung:

SIGMA (ausdr, J, M, N)	**Summation von J = M bis N**
PROD (ausdr, J, M, N)	**Produkt von J = M bis N.**

Beispiele:

```
? SIGMA (A*J, J, 1, N);
@: A*N*(N+1)/2

? SIGMA (1/K - 1/(K+1), K, 1, N);
@: N / (1+N)

? SIGMA (2^-J, J, 0, PINF);        %erfordert LIM.DIF%
@: 2 + MZERO

? PROD (LN(J), J, 2, 4);
@: LN(2) * LN(3) * LN(4)
```

Anmerkungen:

1. Iteriert wird, wenn beide Grenzen numerisch sind. Andernfalls ist das Problem analog zu dem der symbolischen Integration.

2. Die implementierten Methoden schliessen die Pruefung auf verschachtelte Summen, die Distribution von Summen ueber die Terme eines Summanden, das Herausziehen von Faktoren, die nicht

vom Index abhaengen, sowie die Anwendung bekannter Regeln fuer Summen von Exponenten, Potenzen usw. ein.

3. Wenn eine untere Grenze die obere uebertrifft, dann ist die Summe **0** oder das Produkt **1**.

4. Aus **SIGMA.ALG** ist ersichtlich, wie weitere Summations- oder Produktregeln hinzugefuegt werden koennen.

5. Um Summen mit unendlichen Grenzen auszuwerten, sollte das File **LIM.DIF** geladen werden.

7. Grundelemente von mySIMP

Dieses Kapitel gibt zusaetzliche Einblicke in die Programmiersprache mySIMP. Er ist als ergaenzendes Material zu den in den Files **PLESn** enthaltenen Informationen gedacht.

7.1 Datenstrukturen

mySIMP-Daten werden gebildet aus **Namen**, **Zahlen** und **Knoten**. Jeder Typ ist identifzierbar und besteht aus einer festen Anzahl von Pointerzellen, die Speicheradressen enthalten. Die Zellen koennen entweder auf andere Objekte verweisen oder auf spezielle Objekte ausserhalb des Pointer-Raumes von Objekten. Alle drei Typen haben jedoch eine **FIRST**-Zelle und eine **REST**-Zelle, diese **FIRST/REST**-Zellenpaare koennen nur auf Objekte des sog. Pointer-Raumes verweisen. Dies erspart die Notwendigkeit von Typpruefungen zur Laufzeit fuer Selektorfunktionen, die mit diesen Pointern arbeiten.

7.1.1 Namen

Wert	Eigenschaft	Funktion	Printname

Interne Darstellung eines Namens

Ein **Name** ist ein identifizierbares Datenobjekt, das aus vier Pointerzellen besteht. Namen werden eindeutig gespeichert, dies in dem Sinne, dass keine zwei Namen im System identische Printnamen haben. Nachfolgend nun die Erlaeuterung der vier Pointerzellen:

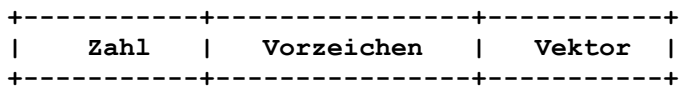
1. Die **FIRST** - oder **Wert** - Zelle enthaelt einen Zeiger auf den gegenwaertigen Wert des Namens, der von der Auswertungsfunktion benutzt wird. Der **Wert** eines Namens wird zunaechst mit dem Namen selbst initialisiert. Dies wird als **Auto-Quoting** bezeichnet. Der Zeiger wird durch die Zuweisungsfunktionen oder durch Funktionen modifiziert, die den Namen als formalen Parameter in ihrer Definition verwenden.

2. Die **REST**- oder **Eigenschaftslisten**-Zelle enthaelt einen Verweis auf die Eigenschaftsliste (**property list**) des Namens. Elemente dieser Liste sind **Indikatoren**, die mit entsprechenden **Eigenschaften** behaftet sind. Eigenschaftslisten sind anfaenglich leer.

3. Die **Funktions**zelle enthaelt einen Pointer auf die mit dem Namen verbundene Funktionsdefinition, falls eine solche existiert. Zum Inhalt dieser Zelle kann nur durch Funktionsanwendungen zugegriffen werden, ebenso kann dieser Inhalt in der Regel nicht veraendert werden, es sei denn durch Anwendung der Funktion selbst. Solange ein Name nicht als Funktion definiert ist, ist der Inhalt der Funktionszelle undefiniert.
4. Die **Printname** - Zelle enthaelt einen Pointer auf den String

von ASCII-Zeichen, die benutzt werden, um den Namen zu drucken. Dieser Printname kann beliebig lang sein. Zugriff auf diese Zelle beschraenkt sich auf Input/Output und subatomare Elemente. **Printnamen** werden geschaffen, wenn ein Name das erste Mal verwendet wird, sie koennen nicht mehr modifiziert werden.

7.1.2 Zahlen



Interne Darstellung einer Zahl

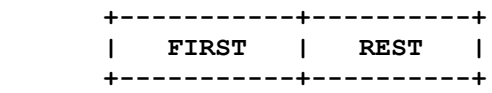
Eine **Zahl** ist ein Datenobjekt, das aus drei Pointerzellen besteht. Zahlen werden nicht eindeutig gespeichert, so dass Duplikate von Zahlen im Speicher existieren koennen. Die Bedeutung der einzelnen Zellen ist folgende:

1. Die **FIRST**-Zelle enthaelt einen Verweis auf die Zahl selbst.

2. Die **REST**- oder **Vorzeichen** - Zelle wird mit **FALSE** initialisiert, wenn die Zahl nichtnegativ ist, andernfalls verweist sie auf **TRUE**. Der Wert dieser Zelle wird bei der Bildung einer Zahl festgelegt.

3. Die **Vektor**zelle enthaelt einen Verweis auf den binaeren Zahlenvektor, der den **numerischer Wert** der Zahl festlegt. Er besteht aus einem vorzeichenbehafteten Vektor von maximal 254 Bytes. Damit ist die Groeese der Zahlen auf **256^254** beschraenkt, was **ueber 600 Dezimalziffern** entspricht.

7.1.3 Knoten



Interne Darstellung eines Knotens

Binaere Baeume sind die hauptsaechliche Datenstruktur von mySIMP. Intern werden diese als ein Netz von Zellenpaaren, **Knoten** genannt, dargestellt. Jeder Knoten besteht aus einer **FIRST**-Zelle und einer **REST**-Zelle.

Knoten werden wegen der von **PRINT** erzeugten und von **READLIST** akzeptierten externen Darstellung oft auch als **gepunktete Paare** bezeichnet:
Die Notation **(X.Y)** stellt einen Knoten dar, dessen **FIRST**-Zelle auf **X** und dessen **REST**-Zelle auf **Y** verweist.

Obwohl die Punktnotation allgemeiner ist, kann man sich die Daten oftmals bequemer als eine lineare Liste vorstellen. Zu diesem Zweck werden **Listen** rekursiv wie folgt definiert:

1. Die **leere Liste** wird durch den Namen **FALSE** bezeichnet.

2. Wenn **X** ein beliebiges **Objekt** ist und **Y** eine **Liste**, dann ist **(X.Y)** eine **Liste**.

Eine Liste von Objekten wird durch die Funktion **PRINT** als eine Folge ihrer Elemente gedruckt, die durch Komma getrennt werden und in Klammern eingeschlossen sind. Die Funktion **READLIST** erkennt diese Notation bei der Eingabe.

Wenn beispielsweise **Y** die Liste

(Y1, Y2, ... , Yn)

ist, dann wird das gepunktete Paar **(X.Y)** ausgedruckt als

(X, Y1, Y2, ... , Yn)

Umgekehrt wird eine Eingabe der Form **(X, Y1, Y2, ... , Yn)** als **(X.Y)** von **READLIST** erkannt.

7.2 Speicherverwaltung

mySIMP fuehrt eine dynamische Speicherverwaltung durch.

Aufteilung des Datenraumes

Waehrend der Initialisierungsphase von mySIMP wird zunnaechst die Groesse des fuer den Interpreter notwendigen Read/Write- Speichers berechnet. Der Speicher wird dann in vier disjunkte Datenraeume nach dem folgenden Schema aufgeteilt:

Anteil	Datenraum	Inhalt
4:32	**Atomraum**	**Pointerzellen fuer Namen und Zahlen**
3:32	**Vektorraum**	**Pnamen-Strings und Zahlenvektoren**
23:32	**Pointerraum**	**Knoten und D-Code**
2:32	**Stack-Raum**	**Kontrollstack**

Garbage Collection

Neue Datenstrukturen werden waehrend der Laufzeit eines mySIMP-Programmes gebildet. Strukturen, auf die kein Verweis mehr existiert, werden automatisch entfernt. Wenn alle verfuegbaren Ressourcen erschoepft sind, wird automatisch Garbage Collection durchgefuehrt. Wenn dabei fuer einen bestimmten Speicherbereich kein neuer Platz zur Verfuegung gestellt werden kann, so wird eine Neuverteilung des gesamten freien Speichers auf die vier Teilbereiche vorgenommen.

7.3 Fehlerbehandlung

Fehler werden in mySIMP in der Regel durch Nachrichten angezeigt, nach denen die Abarbeitung wie vorher fortgesetzt wird. Der Nutzer kann selbst geeignete Reaktionen einleiten. Der einzige Fehler, der nicht in dieser Weise behandelt wird, ist das Erschoepfen des verfuegbaren Speichers. Hier hat der Nutzer die Moeglichkeit, entweder zum mySIMP-Driver zurueckzukehren oder direkt in das Betriebssystem zu gehen.

7.3.1 Speicherueberlauf

Normalerweise stellen das automatische Garbage-Collection und die dynamische Neuaufteilung der Datenraeume genuegend Speicher zur Verfuegung, um den Anforderungen der Nutzerprogramme zu entsprechen.
Sollte dennoch der verfuegbare Speicher zur Darstellung der Datenobjekte nicht ausreichen, so wird die Fehlernachricht

ALL Spaces Exhaustet
Executive: ESC, ALT; System: Ctrl-C?

ausgegeben. Der Nutzer kann dann eine der angegebenen Moeglichkeiten durch die entsprechende Taste auswaehlen. Die **EXECUTIVE**-Moeglichkeit uebergibt die Steuerung zurueck an den mySIMP-Driver, ohne Funktionsdefinitionen, Eigenschaften, Werte oder Namen zu aendern. Die **SYSTEM**-Moeglichkeit beendet mySIMP und gibt die Steuerung zurueck an das Betriebssystem.

7.3.2 Diskettenfehler

Diskettenfehler koennen entstehen, falls nicht genuegend Platz auf einer Diskette ist bzw. wenn versucht wird, nach einem end-of-file zu lesen. Die entsprechenden Fehlernachrichten sind

End-Of-File Read
No Disk Space

7.3.3 Undefinierte numerische Operationen

Wenn das zweite Argument einer der Funktionen **QUOTIENT**, **MOD**, **DIVIDE** den Wert **0** hat, so wird der Zero-Divide-fehler verursacht, es erscheint die Nachricht

ZERO Divide Error

7.3.4 Syntaxfehler bei der Eingabe

Die Funktion **PARSE** erkennt Syntaxfehler bei der Eingabe und gibt folgende Fehlernachrichten aus:

***** SYNTAX ERROR: expression USED AS NAME**
***** SYNTAX ERROR: expression USED AS PREFIX OPERATOR**

***** SYNTAX ERROR: expression USED AS INFIX OPERATOR**
***** SYNTAX ERROR: delimiter NOT FOUND**

Hierbei ist **delimiter** ein fehlender rechter Begrenzer, wie **rechte Klammer**, **ENDFUN**, **ENDSUB**, **EXIT**, **ENDLOOP**, **ENDBLOCK**. In jedem Fall wird der Rest der Eingabe vom Auftreten des Fehlers bis zum naechsten Terminator, wie **";"**, **"$"**, **"&"** mit angezeigt, um das Auffinden des Fehlers zu erleichtern. Beispiele fuer diese vier Typen von Fehlern sind:

```
5 (X);

X Y;

X*/Y;

WHEN ATOM(X, EXIT
```

8. Funktionen und Operatoren von mySIMP

8.1 Selektorfunktionen

Selektorfunktionen werden verwendet, um einen gewuenschten Teilbaum eines eines binaeren Baumes zu bestimmen. Damit ist es moeglich, Informationen aus den elementaren Datenstrukturen von mySIMP zu gewinnen. **FIRST** und **REST** sind die zwei fundamentalen Selektoren. Mehrfache Anwendung dieser zwei Funktionen ermoeglicht das vollstaendige Durchqueren eines beliebigen Baumes. Beide Funktionen sind aus Gruenden der Effektivitaet in Maschinensprache definiert.

```
FUNCTION FIRST (X) ,
     Inhalt der FIRST-Zelle von X,
ENDFUN;
```

Die korrekte Interpretation von FIRST(X) haengt davon ab, ob **X** ein **Atom** ist bzw. ob es andernfalls als **Liste** oder als ein **binaerer Baum** betrachtet wird: Wenn **X** ein **Atom** ist , dann liefert FIRST(X) den **aktuellen Wert** von X.Ist X eine **Liste**, dann liefert FIRST(X) das **erste Element** dieser Liste. Wird X als **binaerer Baum** aufgefasst, so liefert FIRST(X) den **linken Teilbaum** von X.

```
FUNCTION REST (X) ,
     Inhalt der REST-Zelle von X,
ENDFUN;
```

Wie bei FIRST haengt die Interpretation von REST(X) von X ab. Ist X ein **Atom**, dann liefert REST(X) die **Property-Liste** von X. Ist X eine **Liste**, dann liefert REST(X) die um FIRST(X) verminderte Liste. Ist schliesslich X ein **binaerer Baum**, so liefert REST(X) den **rechten Teilbaum** von X.

```
FUNCTION SECOND (X),
     FIRST (REST (X)),
ENDFUN;
```

SECOND(X) liefert das **zweite Element** der Liste X.

```
FUNCTION THIRD (X),
     FIRST (REST (REST (X))),
ENDFUN;
```

THIRD(X) liefert das **dritte Element** der Liste X.

```
FUNCTION RREST (X),
     REST (REST (X)),
ENDFUN;
```

RREST(X) liefert eine um die ersten beiden Elemente von X

verminderte Liste.

```
FUNCTION RRREST (X),
     REST (REST (REST (X))),
ENDFUN;
```

RRREST(X) liefert eine um die ersten drei Elemente von X verminderte Liste.

8.2 Konstruktorfunktionen

Konstruktorfunktionen werden zur dynamischen Erzeugung von Datenstrukturen verwendet. In mySIMP sind solche Strukturen **Listen**, **binaere Baeume** oder **gerichtete Graphen**. Der allgemeine Konstruktor **ADJOIN** bildet einen neuen Knoten. Der dafuer noetige Speicherplatz wird vom Pointerraum genommen. Wenn durch vorhergehende ADJOIN-Operationen der Speicher erschoepft ist, so wird automatisch Garbage Collection durchgefuehrt.

```
FUNCTION ADJOIN (X, Y),
     Ein neuer Knoten mit der FIRST-Zelle X und der REST-Zelle Y,
ENDFUN;
```

Die korrekte Interpretation von ADJOIN haengt davon ab, wie die Datenstrukturen aufgefasst werden. Ist **Y** eine **Liste**, dann liefert ADJOIN(X,Y) eine **Liste**, andernfalls wird ein **Baum** erzeugt, der den linken Unterbaum X und den rechten Unterbaum Y hat. ADJOIN(X,Y) aendert die Strukturen X und Y **nicht**.

```
SUBROUTINE LIST (X1, X2, ..., Xn),
     WHEN n = 0, FALSE EXIT,
     ADJOIN (EVAL (X1), LIST (X2, X3, ..., Xn)),
ENDSUB;
```

LIST akzeptiert eine beliebige Anzahl von Argumenten und liefert eine dynamisch erzeugte **Liste** mit den **ausgewerteten** Elementen. Im Gegensatz dazu werden X1 bis Xn in einer Konstruktion der Form

```
'(X1,X2, ..., Xn)
```

nicht ausgewertet.

```
FUNCTION REVERSE (X, Y),
     WHEN ATOM (X), Y EXIT,
     REVERSE (REST (X), ADJOIN (FIRST (X), Y)),
ENDFUN;
```

REVERSE wird normalerweise mit nur **einem Argument** verwendet, in diesem Falle liefert es eine Liste mit den gleichen Elementen, aber

in umgekehrter Reihenfolge. Ist **Y** ebenfalls eine **Liste**, dann liefert REVERSE(X,Y) eine Liste, die aus den Elementen von REVERSE(X), gefolgt von Y, besteht.

FUNCTION OBLIST (),
Eine Liste der aktuellen elementaren und nutzerdefinierten Namen,
ENDFUN;

OBLIST() liefert die Objektliste, die aus allen gegenwaertig im System vorhandenen Namen besteht, diese sind in der Reihenfolge ihrer Definition geordnet.

8.3 Modifikatorfunktionen

Modifikatorfunktionen aendern Pointer in existierenden Datenstrukturen. Sie koennen sehr effektiv zur Aenderung existierender Datenstrukturen verwendet werden und vermeiden das aufwendige ADJOIN. Der unerfahrene Nutzer sollte diese Funktionen vermeiden, denn:

1. Der Versuch des Druckens einer mit diesen Funktionen erzeugten Ringliste fuehrt zu einem endlosen Drucken.

2. Da Datenstrukturen in myMSIMP oft gemeinsame Teilstrukturen haben, wird sich die Aenderung derartiger Teilstrukturen auf alle sie enthaltenden uebergeordneten Strukturen auswirken. Dies kann unvorhersehbare und nicht gewollte Effekte bewirken und wirkt sich vor allem auf das Testen von Programmen unguenstig aus.
   Es gibt in den meisten mySIMP-Programmen keine zwingende Notwendigkeit der Verwendung von Modifikatorfunktionen.

FUNCTION REPLACEF (X, Y),
FIRST-Zelle von X: Y,
ENDFUN;

Falls **X** eine **Liste** ist, ersetzt diese Funktion das erste Element von X durch Y. Ist X ein **gepunktetes Paar**, wird das linke Element von X durch Y ersetzt. Ist schliesslich X ein **Atom**, so bekommt X den Wert Y.

FUNCTION REPLACER (X, Y),
REST-Zelle von X: Y,
ENDFUN;

Wenn X eine **Liste** ist, so ersetzt diese Funktion den Rest der Liste durch Y. ist X ein **gepunktetes Paar**, so wird das rechte Element von X durch Y ersetzt, ist X ein **Atom**, so wird die Property-Liste von X

durch Y ersetzt.

```
FUNCTION CONCATEN (X, Y),
       WHEN ATOM (X), Y EXIT,
     WHEN ATOM (REST (X)), REPLACER  (X, Y) EXIT,
     CONCATEN (REST (X), Y),
     X,
ENDFUN;
```

Diese Funktion verkettet die Liste X mit der Liste Y ohne Verwendung von ADJOIN, indem der letzte Pointer von X so modifiziert wird, dass er auf Y verweist. **CONCATEN(X,X)** erzeugt eine **Ringliste.**

8.4 Erkennerfunktionen

Erkennerfunktionen werden verwendet, um Datenstrukturen zu identifizieren. Sie akzeptieren grundsaetzlich nur ein Argument und liefern als Resultat entweder **TRUE** oder **FALSE.**

```
FUNCTION NAME (X),
     WHEN X ist ein Name EXIT,
ENDFUN;

FUNCTION INTEGER (X),
     WHEN X ist eine ganze Zahl EXIT,
ENDFUN;


FUNCTION ATOM (X),
     NAME (X) OR INTEGER (X),
ENDFUN;

FUNCTION EMPTY (X),
     X = FALSE,
ENDFUN;
```

EMPTY erkennt die **leere Liste.** Diese Funktion hat die gleiche Definition wie der logische Operator **NOT.** Da das Atom FALSE zur Kennzeichnung der leeren Liste verwendet wird, kann auch die Funktion ATOM zum Erkennen der leeren Liste verwendet werden.

```
FUNCTION POSITIVE (X),
     X > 0,
ENDFUN;
```

POSITIVE erkennt positive ganze Zahlen.

```
FUNCTION NEGATIVE (X),
     X < 0,
ENDFUN;
```

NEGATIVE erkennt negative ganze Zahlen.

```
FUNCTION ZERO (X),
     X = 0,
ENDFUN;
```

ZERO erkennt die ganze Zahl 0.

```
FUNCTION EVEN (X),
     ZERO (MOD (X, 2)),
ENDFUN;
```

EVEN prueft, ob das Argument geradzahlig ist.

8.5 Vergleichsfunktionen und -operatoren

Vergleichsfunktionen bzw. -operatoren werden verwendet, um Datenstrukturen miteinander zu vergleichen. Sie akzeptieren grundsaetzlich zwei Argumente und liefern als Resultat entweder TRUE oder FALSE.

```
PROPERTY RBP, EQ, 80;

PROPERTY LBP, EQ, 80;

FUNCTION EQ (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y), ZERO (X-Y) EXIT,
     WHEN X und Y verweisen auf das gleiche Objekt EXIT,
     FALSE,
ENDFUN;
```

Normalerweise wird der Infixoperator EQ verwendet, um Atome, also Zahlen und Namen, auf Identitaet zu pruefen. Es kann aber durchaus auch sinnvoll sein, nichtatomare Strukturen miteinander zu vergleichen. Fuer Objekte, die keine Zahlen sind, liefert EQ den Wert TRUE genau dann, wenn ihre Argumente identisch sind, also auf das gleiche Datenobjekt oder die gleiche Speicheradresse verweisen. Wie in Kapitel 7 beschrieben, werden Namen in mySIMP eindeutig abgespeichert. Damit liefert EQ eine Moeglichkeit, die Gleichheit von Namen zu pruefen. die Wirkung von PROPERTY wird in 8.8 erklaert.

```
PROPERTY RPB, =, 80;

PROPERTY LBP, =, 80;


FUNCTION = (X, Y),
     WHEN ATOM (X), EQ (X, Y) EXIT,
     WHEN ATOM (Y), FALSE EXIT,
     WHEN FIRST (X) = FIRST (Y),  REST (X) = REST (Y)  EXIT,
ENDFUN;
```

Der Infixoperator **"="** liefert TRUE genau dann, wenn seine Argumente gleich sind. Fuer atomare Argumente ist dies gleichbedeutend mit der Wirkung von EQ. Nichtatomare Argumente werden hingegen nur dahingehend peprueft, ob sie zueinander isomorph sind. Dies bedeutet, dass sie beim Ausdrucken das gleiche Druckbild erzeugen. Da EQ wesentlich schneller ist, sollte es verwendet werden, wenn bekannt ist, dass mindestens eines der Argumente atomar ist.

```
FUNCTION ORDERP (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y),
          LESSP (X, Y) EXIT,
     WHEN die Adresse von X ist kleiner als die von Y,
          TRUE EXIT,
     FALSE,
ENDFUN;
```

ORDERP liefert eine generische Ordnung fuer dem System bekannte Namen, die auf der Reihenfolge ihres Einfuehrens beruht. Falls also der Name X vor dem Namen Y eingefuehrt wurde (dies bedeutet, X liegt rechts von Y in OBLIST()), so liefert ORDERP (X,Y) den Wert TRUE, ansonsten den Wert FALSE. ORDERP hat keinen Sinn fuer Argumente, die keine Namen sind.

```
FUNCTION ORDERED (X, Y),
     WHEN ATOM (Y), ORDERP (X, Y) EXIT,
     WHEN ATOM (X), TRUE EXIT,
     WHEN ORDERED (FIRST (X), FIRST(Y)), TRUE EXIT,
     WHEN FIRST (X) = FIRST (Y),
          ORDERED (REST(X), REST(Y)) EXIT,
ENDFUN;
```

ORDERED liefert eine effektive kanonische Ordnung fuer Ausdruecke, welche fuer viele Zwecke verwendet werden kann. Beispielsweise benutzt myMATH die Funktion ORDERED, um Terme von Summen bzw. Faktoren in Produkten zu sortieren.

```
FUNCTION MEMBER (X, Y),
     WHEN ATOM (Y), FALSE EXIT,
     WHEN X = FIRST (Y) EXIT,
     MEMBER (X, REST (Y)),
ENDFUN;
```

MEMBER (X, Y) liefert den Wert TRUE, wenn X im Sinne des Operators **"="** mit irgendeinem Element der Liste Y uebereinstimmt.

```
FUNCTION GREATER (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y),
          X > Y EXIT,
     FALSE,
ENDFUN;
```

```
FUNCTION LESSER (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y),
          X < Y EXIT,
     FALSE,
ENDFUN;
```

Diese Funktionen vergleichen ganze Zahlen. Man beachte, dass sie auch dann den Wert FALSE liefern, wenn eines der Argumente keine ganze Zahl ist.

```
PROPERTY RBP, >, 80;

PROPERTY LBP, >,80;

FUNCTION > (X, Y),
     GREATER (X, Y),
ENDFUN;


PROPERTY RBP, <, 80;

PROPERTY LBP, <, 80;

FUNCTION < (X, Y),
     LESSER (X, Y),
ENDFUN;
```

8.6 Logische Operatoren

Logische Operatoren gestatten das Arbeiten mit Wahrheitswerten. Dabei ist zu beachten, dass in mySIMP **jeder von FALSE verschiedene Wert als TRUE interpretiert wird.**

```
PROPERTY RBP, NOT, 70;

FUNCTION NOT (X),
     X EQ FALSE,
ENDFUN;


PROPERTY RBP, AND, 60;

PROPERTY LBP, AND, 60;



SUBROUTINE AND (X1,X2, ... , Xn),
     WHEN n = 0, TRUE EXIT,
     WHEN NOT EVAL (X1), FALSE EXIT,
     AND (X2, X3, ... , Xn),
ENDSUB;
```

Der n-stellige Infixoperator **AND** liefert genau dann den Wert TRUE, wenn sich **jedes** seiner Argumente zu einem von FALSE verschiedenen Wert auswerten laesst. AND ist eine Subroutine, die Argumente werden sequentiell ausgewertet, bis entweder ein Wert FALSE entsteht oder alle Argumente von FALSE verschieden sind. Damit werden unnoetige Auswertungen der gesamten Argumentliste vermieden.

```
PROPERTY RBP, OR, 50;

PROPERTY LBP, OR, 50;

SUBROUTINE OR (X1, X2, ... , Xn),
     WHEN n = 0 FALSE EXIT,
     WHEN EVAL (X1), TRUE EXIT,
     OR (X2, X3, ... , Xn),
ENDSUB;
```

Der n-stellige Infixoperator **OR** liefert den Wert TRUE, wenn sich **mindestens eines** seiner Argumente zu einem von FALSE verschiedenen Wert auswerten laesst. Die Auswertung der Argumentliste wird durchgefuehrt, bis der erste von FALSE verschiedene Wert gefunden wurde, die restlichen Argumente bleiben dann unausgewertet.

8.7 Zuweisungen

Auf Grund der Moeglichkeiten der Verwendung rekursiver und funktionaler Programmkonstrukte ist es prinzipiell moeglich, voellig ohne Zuweisungen zu arbeiten. myMATH gestattet es dennoch, Zuweisungen vorzunehmen, man sollte aber wissen, dass diese Nebeneffekte verursachen koennen, die das Testen von Programmen sehr schwierig gestalten. Ferner ist zu beachten, dass das Zeichen **"="** als **Vergleichsoperator** verwendet wird. In mySIMP wird der Doppelpunkt, **":"** als **Zuweisungsoperator** verwendet, ferner ist noch die Zuweisungsfunktion **ASSIGN** verfuegbar.

```
FUNCTION ASSIGN (X, Y),
     REPLACEF (X, Y),
     Y,
ENDFUN;
```

Diese Funktion ersetzt die FIRST-Zelle des ersten Arguments durch das zweite Argument und liefert als Resultat das zweite Argument. Man beachte, dass die Funktion ASSIGN auch dann definiert ist, wenn das erste Argument kein Name ist, in diesem Falle ist ihre Anwendung aber sinnlos.

```
PROPERTY RBP, :, 20;

PROPERTY LBP, :, 180;

SUBROUTINE : (X, Y),
     ASSIGN (X, EVAL (Y)),
ENDSUB;
```

Ein Ausdruck der Form **X : Y** bewirkt, dass FIRST('X) auf Y verweist und Y als Wert des Ausdrucks zurueckgegeben wird. Normalerweise ist X ein Name, so dass FIRST('X) die Wertzelle des Namens X ist, dies entspricht der Ergibtanweisung in den meisten traditionellen Programmiersprachen. X kann aber auch nichtatomar sein, dies bewirkt Seiteneffekte, aehnlich wie bei REPLACEF. In der Praxis wird **":"** oefter verwendet als die Funktion ASSIGN. Folgendes Beispiel soll den Unterschied verdeutlichen: Der Wert von **DOG** sei **FIDO**, dann aendert

DOG: '(A DOBERMAN PINSCHER)

den Wert von DOG in (A DOBERMAN PINSCHER). Im Unterschied dazu wuerde

ASSIGN (DOG, '(A DOBERMAN PINSCHER))

den Wert von **FIDO** in **(A DOBERMAN PINSCHER)** aendern, waehrend der Wert von **DOG** unveraendert **FIDO** ist.

```
SUBROUTINE POP (X),
     POP1  (X, EVAL (X)),
ENDSUB;
```

POP verwendet die **Hilfsfunktion POP1:**

```
FUNCTION POP1 (X, Y),
     ASSIGN(X, REST (Y)),
     FIRST (Y),
ENDFUN;
```

Falls Y der Name einer Liste ist, dann liefert POP(X) das **FIRST-**Element von X und setzt X selbst auf REST(X). Dies ist die bekannte Pop-Operation in Stacks.

```
SUBROUTINE PUSH (X, Y),
     ASSIGN (Y, ADJOIN  (EVAL (X), EVAL (Y))),
ENDSUB;
```

Wenn Y der Name einer Liste ist und Y ein Ausdruck, so fuegt PUSH(X,Y) X zur Liste Y hinzu, Y verweist anschliessend auf diese verlaengerte Liste. Dies ist die bekannte Push-Operation fuer Stacks.

8.8 Property-Funktionen

Property-Funktionen bieten eine Moeglichkeit zur **Verknuepfung von Eigenschaften mit Namen**. Die Property-Liste eines Namens wird benutzt, um diese Eigenschaften zusammen mit zugeordneten Indikatoren abzuspeichern. Die mit einem Indikator assoziierte Eigenschaft kann durch die Funktion **GET** abgefragt und weiter verwendet werden. Diese Eigenschaften gestatten die Kostruktion von flexiblen und effektiven Datenbasen. So werden in myMATH beispielsweise Property-Listen zur Erhoehung der

Geschwindigkeit und der Modularitaet der einzelnen Bausteine verwendet.

```
FUNCTION ASSOC (X, Y),
     WHEN ATOM (X), Y EXIT,
     WHEN ATOM (FIRST (Y)), ASSOC (X, REST (Y)) EXIT,
     WHEN FIRST (FIRST (Y)) = X, FIRST (Y) EXIT,
     ASSOC (X, REST (Y)),
ENDFUN;
```

ASSOC(X,Y) fuehrt in der Assoziationsliste Y eine lineare Suche aus. Gesucht wird dabei ein nichtatomares Element, dessen FIRST-Komponente mit X uebereinstimmt, wobei die Uebereinstimmung im Sinne des Operators **"="** zu verstehen ist. Falls ein derartiges Element existiert, so ist es das Resultat, andernfalls wird FALSE geliefert.

```
FUNCTION PUT (X, Y, Z),
          WHEN EMPTY (GET (X, Y)),
               REPLACER (X, ADJOIN (ADJOIN (Y, Z), REST (X))),
               Z EXIT,
          REPLACER (ASSOC (Y, REST (X)), Z),
          Z,
ENDFUN;
```

PUT(X,Y,Z) setzt die Eigenschaft Z unter dem Indikator Y auf die Eigenschaftsliste des Namens X.Alle unter dem gleichen Indikator frueher abgespeicherten Eigenschaften werden dabei zerstoert.

```
SUBROUTINE PUTPROP (X, Y, Z),
     WHEN GET (X, Y) = Z OR GETD (GET (X, Y)) = Z, X EXIT,
     BLOCK
          WHEN NOT GET (X, Y) EXIT,
          WHEN SCAN EQ '$ EXIT,
          PRINT "*** REDEFINED:",
     ENDBLOCK,
     WHEN FIRST (Z) = 'FUNCTION,
          PUTD (PUT (X, Y, COMPRESS (LIST (X, Y))), Z),
          X EXIT,
     PUT (X, Y, Z),
     X,
ENDSUB;
```

Falls Z sich noch nicht unter dem Indikator Y auf der Eigenschaftsliste des Namens X befindet, so setzt PUTPROP(X,Y,Z) entweder Z oder die komprimierten Namen X und Y auf die Eigenschaftsliste von X unter dem Indikator Y. Ist Z ein Funktionskoerper, wie er in 8.14 beschrieben wird, so wird er benutzt, um die Funktion mit dem Namen COMPRESS(X Y) zu definieren. Das Umwandeln von Z in eine compilierte Funktion reduziert den Speicherbedarf auf 1/3 des urspruenglichen Wertes und beschleunigt die Abarbeitung um 20%. PUTPROP gibt eine Warnung aus, falls der Eigenschaftswert umdefiniert wurde.

```
FUNCTION PUTPROPER (EX1, EX2),
     WHEN NAME (EX1) AND NAME (EX2),
          LOOP
               WHEN NOT SCAN EQ COMMA EXIT,
               SCAN(),
          ENDLOOP,
          LIST (PUTPROP, EX1, EX2, PARSE (SCAN, 0)) EXIT,
     SYNTAX (),
ENDFUN;


PROPERTY PREFIX, PROPERTY,
     PUTPROPER (READLIST  (SCAN), READLIST (SCAN));
```

Der **Operator PROPERTY** bietet die Moeglichkeit, Eigenschaftswerte auf einen Namen zu setzen. Das folgende mySIMP-Kommando bewirkt, dass die Funktion PUTPROP die Eigenschaft Z unter dem Indikator Y auf die Property-Liste von X setzt,wobei Z auch eine Funktionsdefinition sein kann:

```
PROPERTY X, Y, Z;
```

Wenn entweder X oder Y ein Name ist, so bewirkt dies einen Syntax-fehler. Wenn es einen frueheren Wert in der Eigenschaftsliste von X unter dem Indekator Y gibt, wird eine Warnung ausgegeben. Die drei Operanden eines PROPERTY-Kommandos werden automatisch quotiert.

```
FUNCTION GET (X, Y),
     X: ASSOC (Y, REST (X)),
     WHEN ATOM (X), FALSE EXIT,
     REST (X),
ENDFUN;
```

GET(X,Y) liefert die mit dem Namen X unter dem Indikator Y verbundenen Eigenschaftswerte, falls solche existieren bzw. den Wert FALSE.

8.9 Definition von Funktionen und Subroutinen

Die nachfolgend erlaeuterten Kommandos und Funktionen zur Funktionsdefinition sind das einzige Mittel, um zur **Funktionszelle** eines Namens zuzugreifen. Jede Funktion wird nach ihrer Definition pseudocompiliert und in einer sehr kompakten Form, dem sog. **D-Code** oder **destilled Code,** abgespeichert. Dies bewirkt eine 3-fache Erhoehung der Codeeffektivitaet und eine 20%- ige Erhoehung der Interpretationsgeschwindigkeit gegenueber der Funktionsdarstellung in Form verketteter Listen. Der umgekehrte Prozess der Decompilierung einer Funktionsdefinition in die Listendarstellung wird durch Aufruf der Funktion **GETD** bewirkt.

```
FUNCTION GETD (X),
     WHEN NOT NAME (X), FALSE EXIT,
     WHEN X ist keine Funktion oder Subroutine,  FALSE EXIT,
     WHEN SUBR (X) OR FSUBR (X),
          Adresse der Maschinencode-Routine  EXIT,
     zum D-Code der Funktion X aequivalente Liste,
ENDFUN;
```

GETD wird benutzt, um die Definition einer mySIMP-Funktion oder Subroutine fuer eine Weiterverarbeitung "zurueckzugewinnen". Ist diese Funktion oder Subroutine in Maschinensprache definiert, dann wird die physische Speicheradresse geliefert. Ansonsten ist das Resultat das Listenaequivalent des D-Codes.

```
FUNCTION PUTD (X, Y),
     WHEN NOT NAME (X), FALSE EXIT,
     WHEN INTEGER (Y),
          Funktionszelle von X: Y,
          Y EXIT,
     Funktionszelle von X: D-Code -Aequivalent von Y,
     Y,
ENDFUN;
```

Ist Y eine ganze Zahl, so belegt PUTD(X,Y) die Funktionszelle von X mit der durch die ganze Zahl Y angegebenen Speicheradresse, diese Rechnung wird **modulo 64K** durchgefuehrt. Andernfalls wird die Definitonszelle mit dem D-Code-Aequivalent von Y belegt.

```
FUNCTION MOVD (X, Y),
     WHEN NOT NAME (X) OR NOT NAME (Y), FALSE EXIT,
     Funktionszelle von Y : Funktionszelle von X,
     GETD (Y),
ENDFUN;
```

Diese Funktion bewirkt, dass die Funktionszelle von Y auf die gleiche Speicherstelle verweist wie die von X. MOVD ist guenstiger in Bezug auf Speicherbedarf und Zeitverhalten als GETD oder PUTD.

8.9.1 Definition von Funktionen

Zur Funktionsdefinition wird die Funktion **FUNCTION** verwendet, sie hat folgende allgemeine Form:

```
FUNCTION name parameter,
     task1,
     task2,
      ...
     taskn,
ENDFUN;
```

Der Funktionsname kann weggelassen werden, wenn keine Notwendigkeit besteht, sich spaeter auf ihn zu beziehen. Dies ist z.B. der Fall, wenn eine nichtrekursive Funktion in einer property-Liste mit der Anweisung **PROPERTY** gespeichert wird. Sind **parameter** angegeben, so kann die Funktion mit einer beliebigen Anzahl von Parametern aufgerufen werden. Diese werden der Funktion als eine einzige Liste von Parametern uebergeben. Werden beim Aufruf der Funktion mehr Parameter verwendet als bei der Funktionsdefinition, so werden die "ueberschuessigen" Parameter mit **FALSE** initialisiert und koennen somit als lokale Variable fungieren.

Die Auswertung der **tasks** innerhalb der Funktion erfolgt sukzessive, bis kein task mehr vorhanden ist oder ein von FALSE verschiedenes Praedikat ausgewertet wurde. Der Funktionswert ist der Wert des zuletzt ausgewerteten task.

Als interne Konstanten in Funktionsdefinitionen koennen keine gepunkteten Paare erwendet werden.

8.9.2 Definitionen von Subroutinen

Zur Definition von Subroutinen wird die **SUBROUTINE**-Anweisung verwendet. Diese hat folgende allgemeine Form:

```
SUBROUTINE name parameter,
     task1,
     task2,
      ...
     taskn,
ENDSUB;
```

Die Argumente des Aufrufs einer Subroutine werden vor ihrer Uebergabe **nicht ausgewertet**. Ansonsten velaeuft die Abarbeitung einer Subroutine wie die einer Funktion.

8.10 Subatomare Funktionen

Diese werden so bezeichnet, weil sie einen Zugriff zu den Zeichen des Printnamens eines Namens oder zum den Elementen des Zahlenvektors einer Zahl ermoeglichen. Dies macht es moeglich, zeitweilig den Printnamen eines Atoms zu "entpacken", mit der resultierenden Liste von Zeichen zu arbeiten und diese Liste schliesslich wieder zu verdichten, um einen neueu Namen zu erzeugen.

```
FUNCTION COMPRESS (X),
     WHEN ATOM (X), " " EXIT,
     WHEN NAME(FIRST(X)),
          Verbinde den Printnamen von FIRST(X) mit dem Anfang von
          COMPRESS (REST(X)),
          Gib den resultierenden Namen zurueck EXIT,
     WHEN INTEGER (FIRST (X)),
          Verbinde  den  Printnamen  der  Zahl  FIRST(X)  mit
          COMPRESS(REST(X)),
          Gib den resultierenden Namen zurueck EXIT,
     COMPRESS (REST (X)),
ENDFUN;
```

COMPRESS liefert einen Namen, dessen Printname eine gepackte Version der Printnamen der atomaren Elemente der Liste X ist. Die aktuelle RADIX-Basis wird dabei verwendet, um den String der Printnamen von ganzen Zahlen zu ermitteln. COMPRESS liefert stets einen Namen.

```
FUNCTION EXPLODE (X),
     WHEN NAME (X),
          Gib eine Liste der einzelnen Zeichen des Printnaens von
          X in der entsprechenden Reihenfolge zurueck EXIT,
     WHEN INTEGER (X),
          Gib  eine  Liste  der Ziffern von  X  zurueck EXIT,
     FALSE,
ENDFUN;
```

EXPLODE liefert eine Liste von Namen, deren einstellige Printnamen mit der Folge der Zeichen in der gedruckten Darstellung des Atoms X uebereinstimmen. Falls X nichtatomar ist , wird FALSE geliefert.

```
FUNCTION LENGTH (X),
     WHEN NAME (X),
          Anzahl der Zeichen im Printnamen von X EXIT,
     WHEN INTEGER (X),
          Anzahl der zum Drucken xon X notwendigen Ziffern  EXIT,
     WHEN ATOM (REST (X)), 1 EXIT,
     1 + LENGTH (REST (X)),
ENDFUN;
```

LENGTH vereinigt in sich die Wirkung von **drei** Funktionen:

1. Wenn X ein Name ist, so wird die Anzahl der zum Druck von X notwendigen Zeichen geliefert. Dabei wird der Wert der Steuervariablen PRINT zur Bestimmung dieser Anzahl verendet. Die Wirkung

der Variablen PRINT wird in 8.13 beschrieben.

2. Wenn X eine ganze Zahl ist, so wird die Anzahl der zum Druck von X erforderlichen Zeichen geliefert. Dabei werden die aktuelle Basis, das evtl. vorhandene Vorzeichen "-" und/oder fuehrende Ziffern "0" in diese Berechnung mit einbezogen.

3. Ist X nichtatomar, dann wird die Anzahl der Knoten dieser Liste geliefert.

8.11 Arithmetische Operatoren und Funktionen

Die arithmetischen Operatoren und Funktionen realisieren exakte integer-Arithmetik fuer Zahlen bis zu einer Groesse von **2^2032-1**, dies entspricht mehr als 600 Dezimalziffern. Sind die Operanden keine ganzen Zahlen, so wird der Wert FALSE geliefert, dies auch im Falle eines Ueberlaufs. Division durch 0 verursacht die Fehlermeldung:

ZERO Divide Error

und liefert den Wert FALSE.

mySIMP realisiert keine Rationalarithmetik, dies wird durch das File ARITH.myS vorgenommen. mySIMP stellt die Grundfunktionen **MINUS, PLUS, DIFFERENCE, TIMES und QUOTIENT**F fuer die Integer-Arithmetik zur Verfuegung.

```
FUNCTION MINUS (X),
     WHEN INTEGER (X), -X EXIT,
ENDFUN;


FUNCTION PLUS (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y), X + Y EXIT,
ENDFUN;


FUNCTION DIFFERENCE (X, Y),
     WHEN INTEGER(X) AND INTEGER (Y) , X - Y EXIT,
ENDFUN;


FUNCTION TIMES (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y), X * Y EXIT,
ENDFUN;


FUNCTION QUOTIENT (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y),
          WHEN Y = 0, Zero-Divide Error EXIT,
          WHEN POSITIVE (Y),  floor (X/Y) EXIT,
          ceiling ( X/Y) EXIT,
ENDFUN;
```

QUOTIENT liefert einen mit folgender Regel konsistenten Quotienten:

```
X = Y*QUOTIENT(X,Y) + MOD(X,Y).
```

```
FUNCTION MOD(X, Y),
     X - (Y * QUOTIENT(X,Y)),
ENDFUN;
```

```
FUNCTION DIVIDE (X, Y),
     WHEN INTEGER (X) AND INTEGER (Y),
          ADJOIN (QUOTIENT (X, Y), MOD (X, Y)) EXIT,
ENDFUN;
```

DIVIDE liefert ein gepunktetes Paar, das aus dem ganzzahligen Quotient und dem Rest besteht. Man beachte, dass gepunktete Paare nur dann richtig ausgedruckt werden, wenn der Begrenzer **"&"** oder die Funktion **PRINT** verwendet werden.

```
PROPERTY PBP, +, 100;

PROPERTY LBP, +, 100;

PROPERTY PREFIX, +, PARSE (SCAN, 130);

FUNCTION + (X, Y),
     PLUS (X, Y),
ENDFUN;
```

Der Parser ignoriert das einstellige **"+"**.

```
PROPERTY RBP, -, 100;

PROPERTY LBP, -, 100;

PROPERTY REFIX, -, LIST ('-,PARSE(SCAN,130));

FUNCTION - (X, Y),
     WHEN EMPTY (Y),
          MINUS (X) EXIT,
     DIFFERENCE (X, Y),
ENDFUN;
```

"-" wird ein- und zweistellig verwendet.

```
PROPERTY RBP, *, 120;

PROPERTY LBP, *, 120;
FUNCTION * (X, Y),
     TIMES (X, Y),
ENDFUN;
```

```
PROPERTY RBP, /, 120;

PROPERTY LBP, /, 120;

FUNCTION / (X, Y),
     QUOTIENT (X, Y),
ENDFUN;
```

8.12 READER-Funktionen, Analysefunktionen

8.12.1 Reader-Funktionen

Readerfunktionen dienen der zeichenweisen Eingabe, sie lesen Zeichen vom aktuellen Eingabemedium. Dies kann entweder die Konsole oder ein auf Diskette befindliches Textfile sein. Die Steuerung wird durch die Funktion **RDS** in Verbindung mit der Steuervariable **RDS** realisiert.

```
FUNCTION RDS (X, Y, Z),
     WHEN EMPTY (X),
          RDS : FALSE EXIT,
     WHEN NAME (X) AND NAME (Y),
          WHEN EMPTY (Z),
               WHEN ein File X.Y auf dem aktiven Laufwerk
                    existiert,
                    eroeffne das Eingabefile X.Y,
                    RDS : X EXIT,
               RDS : FALSE EXIT,
          WHEN NAME (Z),
               WHEN ein File X.Y auf dem Laufwerk Z existiert,
                    eroeffne das Eingabefile X.Y auf dem Laufwerk
                    Z,
                    RDS : FALSE EXIT,
               RDS : FALSE EXIT,
          RDS : FALSE EXIT,
     RDS : FALSE EXIT,
ENDFUN;
```

Die Funktion RDS (**R**ea**D** **S**elect) wird verwendet, um ein Eingabequellfile auszuwaehlen. Wenn das gewaehlte File gefunden wurde, wird es fuer die Eingabe eroeffnet, und die Variable RDS erhaelt als Wert den Namen des Files. Damit wird diese File das neue aktive Eingabequellfile. Wird RDS ohne Argumente bzw. mit falschen Argumenten aufgerufen aufgerufen oder wenn das File nicht gefunden wurde, so bekommt die Variable RDS den Wert FALSE, damit wird die Konsole aktuelles Eingabemedium.

```
FUNCTION READCHAR (),
     Lies ein Zeichen vom aktuellen Eingabemedium,
     SCAN : das entsprechende mySIMP-Atom,
ENDFUN;
```

Diese Funktion liest und liefert einzelne Zeichen vom aktuellen Eingabemedium. Dies erfordert, dass alle Erkennerfunktionen vom Anwendungsprogramm realisiert werden. Als zusaetzliche Faehigkeit wird das gelieferte Atom noch dem **mySIMP-Namen SCAN** zugewiesen.

```
FUNCTION SCAN (),
     Lies ein Token vom aktuellen Eingabemedium,
     SCAN : entsprechender Name oder Zahl,
ENDFUN;
```

Diese Funktion liest Token vom aktuellen Eingabemedium. Ein Token ist ein String von Zeichen, der entweder durch einen Begrenzer oder durch ein Unterbrechungszeichen abgeschlossen wird. Integers werden durch ein nicht in der aktuellen Basis enthaltenes Zeichen abgeschlossen.

Begrenzer dienen nur zum Abschluss von Tokens, sie werden durch SCAN() nicht als Atome zurueckgegeben. Folgende Zeichen dienen als **Begrenzer** fuer SCAN:

space, carriage return, line-feed, tab (CTRL-I)

Unterbrechungszeichen sind:

! $ & ') (* + , - . / @ :

; < = > ?] \ [^ _ ` { | }

Sie werden von SCAN() als Namen zurueckgegeben, die nur aus einem Zeichen bestehen. Spezielle Zeichen, wie Kommentarzeichen, Begrenzer und Unterbrechungszeichen koennen als Namen oder teile von Namen verwendet werden, wenn die entsprechenden Strings in doppelte Quotierungszeichen " eingeschlossen werden.
Kommentare koennen an einer beliebigen Stelle eines Eingabefiles enthalten sein, sie werden durch **Prozentzeichen %** begrenzt.

8.12.2 Steuervariable

Der Reader verwendet folgende Steuervariable mit den angegebenen Anfangswerten:

```
RDS : FALSE;
READ : 'READ;
READCHAR : 'READCHAR;
ECHO :FALSE;
```

Die Steuerung kann an die Konsole zureckgegeben werden, ohne das aktuelle Eingabefile zu schliessen, indem der Steuervariablen RDS der Wert FALSE innerhalb dieses Files zugewiesen wird. Eine spaetere Zuweisung eines von FALSE verschiedenen Wertes an RDS von der Konsole aus gibt die Steuerung an die Unterbrechungsstelle im Eingabefile zurueck. Diese Technik wird beispielsweise verwendet, um die Abarbeitung der interaktiven Lektionen zu steuern. Der DRIVER von mySIMP setzt beim Start und bei Fehlern die Variable RDS auf den Anfangswert FALSE.

Falls READ von FALSE verschieden ist, sind auch Kleinbuchstaben verwendbar, diese werden von den Grossbuchstaben streng unterschieden. Die einzige Ausnahme hiervon ist die, dass bei Verwendung der Funktionen WRS und RDS Kleinbuchstaben in Filenamen automatisch in Grossbuchstaben transformiert werden. Hat RDS den Wert FALSE, so werden alle Kleinbuchstaben bei der Eingabe in Grossbuchstaben transformiert. Bereits im System vorhandene Kleinbuchstaben bleiben aber erhalten.

Der Standardwert von READCHAR ist **'READCHAR**, die Eingabe von der Konsole erfolgt dann im Zeilenedit-Modus.

Wenn ECHO bei der Eingabe von Diskette einen von FALSE verschiedenen Wert hat, dann werden alle Eingabezeichen auf die Ausgabe gelegt, normalerweise die Konsole.

8.12.3 Analysefunktionen

Wichtigste Analysefunktion ist **PARSE (ausdr, rbp)**, sie wird verwendet, um einen myMATH-Ausdruck zu lesen und in Listennotation zu transformieren. Dabei werden bei Operatoren die Wertigkeit der Links- bzw. Rechtsbindung (**LBP - Left Binding Power** und **RBP - Right Binding Power**) und die **INFIX**- bzw.**PREFIX**- Eigenschaft beruecksichtigt. Folgen mehrere Operatoren gleicher Prioritaet aufeinander, so erfolgt die indung von links nach rechts.

PARSE benutzt folgende **Properties**:

1. **INFIX** ist ein Name, auf dessen Property-Liste Ausdruecke gespeichert sind, die genau festlegen, wie die Infixoperatoren durch PARSE auszuwerten sind.

2. **PREFIX** ist ein Name, auf dessen Property-Liste Auswertungsregeln fuer Praefixoperatoren gespeichert sind.

3. **LBP** ist ein Name, auf dessen Property-Liste die Wertigkeiten der Linksbindung von Infix- und Postfixoperatoren gespeichert

sind.

4. **RBP** enthaelt auf seiner Property-Liste die Wertigkeiten der Rechtsbindung von Infix- und Praefixoperatoren.

Folgende **globale Konstanten** werden von PARSE verwendet:

```
RPAR : '")";
LPAR : '"(";
COMMA : '",";
DELIMITER : '(EXIT, ENDLOOP, ENDBLOCK, ENDFUN, ENDSUB, ")", ",");
```

Diese Konstanten koennen anstelle von Klammern bzw. Kommas verwendet werden sie sind frei von jeglichen Parse-Eigenschaften. DELIMITER enthaelt die vom Parser akzeptierten Begrenzer.

```
FUNCTION TERMINATOR (),
     SCAN = '; OR SCAN = '$ OR SCAN = '&,
ENDFUN;

FUNCTION DELIMITER (),
     TERMINATOR () OR MEMBER (SCAN, DELIMITER),
ENDFUN;
```

DELIMITER ist ein Praedikat, das den Wert FALSE liefert, wenn der aktuelle Wert von SCAN weder ein Begrenzer ist noch in der Liste DELIMITER enthalten ist.

```
FUNCTION MATCH (DELIM),
     % benutzt die Variable SCAN, die von SCAN () gesetzt wird%
     WHEN SCAN = DELIM, SCAN (), FALSE EXIT,
     WHEN SCAN = COMMA, SCAN (), MATCH (DELIM) EXIT,
     WHEN DELIMITER (), SYNTAX (DELIM, "NOT FOUND") EXIT,
     ADJOIN (PARSE (SCAN, 0), MATCH (DELIM)),
ENDFUN;
```

MATCH(DELIM) ist eine Funktion zum Analysieren von Ausdruecken, die durch Kommas getrennt und durch den Wert ihrer Argumente abgeschlossen werden. MATCH liefert eine Liste der resultierenden Darstellungen dieser Ausdruecke.

```
PROPERTY PREFIX, WHEN,
     MATCH ('EXIT);

PROPERTY PREFIX, BLOCK,
     MATCH ('ENDBLOCK);

PROPERTY PREFIX, LOOP,
     ADJOIN ('LOOP, MATCH ('ENDLOOP));
```

Obige Anweisungen definieren die Parse-Eigenschaften fuer die **WHEN - EXIT**, **BLOCK - ENDBLOCK** und die **LOOP - ENDLOOP** - Steuerkonstrukte.

Sie rufen MATCH() auf, um den entsprechenden Begrenzer zu finden.

```
PROPERTY INFIX, "(", COND (
     WHEN NAME (EX1),
          ADJOIN (EX1, MATCH(RPAR)) EXIT,
     WHEN SYNTAX () EXIT;

FUNCTION MATCHNOP (X, DELIM),
     WHEN SCAN EQ DELIM, SCAN(), EXIT,
     SYNTAX (DELIM, "NOT FOUND"),
ENDFUN;

PROPERTY PREFIX, "(",
     MATCHNOP (PARSE (EX2, 0), RPAR);

FUNCTION READLIST (X),
     WHEN X EQ LPAR,
          LOOP
               WHEN NOT SCAN () EQ COMMA EXIT,
          ENDLOOP,
          WHEN SCAN EQ RPAR, SCAN (), FALSE EXIT,
          ADJOIN (READLIST (SCAN), READREST (SCAN)) EXIT,
     WHEN X EQ RPAR, SYNTAX () EXIT,
     SCAN (),
     WHEN X EQ COMMA, READLIST (SCAN) EXIT,
     X,
ENDFUN;
FUNCTION READREST (X),
     WHEN X EQ RPAR, SCAN (), FALSE EXIT,
     WHEN X EQ .,
          X : READLIST (SCAN ()),
          WHEN SCAN EQ RPAR, SCAN (), X EXIT,
          SYNTAX () EXIT,
     ADJOIN (READLIST (X), READREST (SCAN)),
ENDFUN;

PROPERTY PREFIX, ', LIST (', READLIST (SCAN));
```

Hiermit werden die Parse-Eigenschaften des Quotierungsoperators angegeben. Dieser Operator wird in folgenden zwei Faellen verwendet:

a) Als Funktion unterdrueckt er die Auswertung seiner Argumente.
b) Als Operator bewirkt er, dass sein Operand in Listennotation eingelesen wird.

```
FUNCTION SYNTAX (X),
     WHEN ERR EXIT,
     ERR : TRUE,
     NEWLINE (),
     PRINT (" *** SYNTAX ERROR:  "),
     drucke jedes Element in X,
     NEWLINE (),
     Lies  und  zeige die Zeichen an bis ein  Begrenzer  gefunden
     wird,
```

```
        RDS : FALSE,
    ENDFUN;
```

SYNTAX ist eine Funktion, die eine beliebige Anzahl von Argumenten akzeptiert. Wenn die globale Variable **ERR** den Wert FALSE hat, dann wird die Nachricht **"*** SYNTAX ERROR: "** , gefolgt von den Argumenten von SYNTAX ausgedruckt. Der Rest des Ausdrucks wird gelesen bis zum naechsten Begrenzer. Schliesslich wird die Variable RDS auf FALSE gesetzt und damit die Steuerung an die Konsole zurueckgegeben.

Tabelle der Operatorbindungen

Operator	Operation	LBP	RBP
(	Gruppierung	200	0
:	Zuweisung	180	20
!	Fakultaet	160	0
^	Exponentiation	140	139
*	Multiplikation	120	120
/	Division	120	120
+	Addition	100	100
-	Subtraktion	100	100
==	Gleichung	80	80
=	Gleichheit	80	80
<	kleiner als	80	80
>	groesser als	80	80
NOT	Negation	70	70
AND	Konjunktion	60	60
OR	Disjunktion	50	50

Falls **"+"** und **"-"** als Praefixoperatoren verwendet werden, so betraegt die Wertigkeit der Rechtsbindung 130.

8.13 Druckfunktionen und Steuervariable

8.13.1 Druckfunktionen

Die Druckfunktionen von mySIMP lenken die die Ausgabe auf das aktuelle Ausgabemedium. Dies kann entweder die Konsole oder ein Diskettenfile sein und wird durch die Variable und die Funktion WRS festgelegt.

```
FUNCTION WRS (X, Y, Z),
     WHEN NOT EMPTY (WRS),
          Schreibe den letzten Record von WRS aus und schliesse
          das File,
          WRS : FALSE,
          WRS (X, Y, Z) EXIT,
     WHEN EMPTY (X),
          WRS : FALSE EXIT,
     WHEN NAME (X) AND NAME (Y),
          WHEN EMPTY (Z),
               Falls auf dem aktiven Laufwerk ein File X.Y
               existiert, so entferne das ggf. existierende File
               X.BAK, benenne X.Y um in X.BAK und erzeuge einen
               neuen Verzeichniseintrag fuer X.Y,
               WRS : X EXIT,
          WHEN NAME (Z),
               Falls auf dem Laufwerk Z ein File X.Y existiert,
               so entferne das ggf. vorhandene File X.BAK,
               benenne das File X.Y um in X.BAK und erzeuge einen
               neuen Verzeichniseintrag fuer X.Y,
               WRS : X EXIT,
          WRS : FALSE EXIT,
     WRS : FALSE,
ENDFUN;
```

Die uebliche Anwendung von WRS hat folgende Form:

```
WRS:FALSE;
ECHO:TRUE;
WRS(filename, filetyp, drv);
aktion1;
  ...
aktionn;
WRS(FALSE);
```

Die Resultate der Wirkung von $aktion_1$ bis $aktion_n$ werden auf ein Diskettenfile **filename.filetyp** auf dem Laufwerk**drv** geschrieben.

Da hierbei auch das Prompterzeichen und weitere Informationen mit auf dieses File geschrieben werden, entstehen beim erneuten Eilesen dieses Files mittels **RDS(filename,filetyp,drv);** in der Regel Syntaxfehler. Es ist daher ratsam, diese stoerenden Zeichen aus dem erzeugten File zu entfernen.

```
FUNCTION PRINT (X),
     WHEN NAME (X),
          Gib den Printnamen von X zurueck EXIT,
     WHEN INTEGER (X),
          Gib die Ziffern von X in der aktuellen Basis zurueck
```

```
EXIT,
          PRINT (LPAR),
          PRINLIST (X), X,
     ENDFUN;
```

PRINT verwendet folgende Hilfsfunktion:

```
          FUNCTION PRINLIST (X),
               PRINT (FIRST (X)),
               WHEN EMPTY (REST (X)), PRINT (RPAR) EXIT,
               SPACES (1),
               WHEN ATOM (REST (X)),
                    PRINT (","),
                    PRINT (REST (X)),
                    PRINT (RPAR)EXIT,
               PRINLIST (RES (X)),
          ENDFUN;
```

```
     FUNCTION PRINTLINE (X),
          PRINT (X),
          NEWLINE (),
          X,
     ENDFUN;
```

```
     FUNCTION LINELENGTH (X),
          WHEN X > 11 AND X < 256,
               Setze die maximale Zeilenlaenge auf X,
               Gib die fruehere Zeilenlaenge zurueck EXIT,
          Gib die aktuelle Zeilenlaenge zurueck,
     ENDFUN;
```

Der Anfangswert der Zeilenlaenge betraegt 72.

```
     FUNCTION RADIX (X),
          WHEN X > 1 AND X < 37,
               Setze die Basis auf X,
               Gib die alte Basis zurueck EXIT,
          Gib die aktuelle Basis zureuck,
     ENDFUN;
```

```
     FUNCTION NEWLINE (X),
          WHEN ZERO (X) , FALSE EXIT,
          Gib  ein  carriage  return und line feed  auf  das  aktuelle
          Ausgabemedium,
          WHEN POSITIVE (X) AND X < 256,
               NEWLINE (X-1) EXIT,
     ENDFUN;
```

Falls X nichtnegativ ist, so gibt NEWLINE (X) X neue Zeilen aus,

andernfalls eine neue Zeile.

```
FUNCTION SPACES (X),
     WHEN X > 0 AND X < 256,
          PRINT (" "),
          SPACES (X - 1) EXIT,
     Gib die aktuelle Kursorposition zurueck,
ENDFUN;
```

Die Funktion **PRTMATH (EX1, RBP, LBP, PRTSPACE)** druckt EX1 in mathematischer Notation. Die "normale" Anwendung hat die Form

PRTMATH (expr, 0, 0, TRUE).

8.13.2 Steuervariable

Der Printer verwendet die folgenden Steuervariablen mit den entsprechenden Anfangswerten:

```
WRS : FALSE;
PRINTLINE : 'PRINTLINE;
PRINT : 'PRINT;
ECHO : FALSE;
```

Wird **WRS** auf **FALSE** gesetzt, so kann die Ausgabe von einem Diskettenfile auf die Konsole umgelenkt werden, ohne das File zu schliessen. Eine spaetere Zuweisung eines von FALSE verschiedenen Wertes an WRS lenkt die Ausgabe dann wieder auf das Diskettenfile um.

PRINTLINE steuert die Umwandlung von Grossbuchstaben in Kleinbuchstaben bei der Ausgabe. Normalerweise ist PRINTLINE von FALSE verschieden, dann werden alle Buchstaben entsprechend ihrer internen Darstellung ausgegeben. Hat PRINTLINE den Wert FALSE, so werden Grossbuchstaben in Kleinbuchstaben umgewandelt. Dies hat aber keinen Einfluss auf die interne Abspeicherung der Printnamen.

Wenn die **Variable PRINT** den Wert **FALSE** hat, werden Namen, die Trenn- oder Unterbrechungszeichen enthalten, unter Verwendung doppelter Quotierungszeichen ausgegeben. PRINT ist normalerweise von FALSE verschieden.

Die Wirkung von ECHO wurde in 8.12.2 erlaeutert.

8.14 Driver- und Auswertungsfunktionen

8.14.1 Driverfunktion

Die Steuerung des Abarbeitungszyklus wird von der Funktion **DRIVER** realisiert. Nach einigen notwendigen Initialisierungen tritt DRIVER in eine Schleife mit den folgenden Aufgaben ein:

Nach dem Druck des Prompterzeichens von mySIMP wird die Funktion PARSE aufgerufen, um vom aktuellen Eingabemedium zu lesen und die Eingabe in die interne Form zu transformieren. Dann wird die Funktion **EVAL** aufgerufen, um den Ausdruck auszuwerten. Schliesslich wird **PRTMATH** aufgerufen, hierbei wird der resultierende Wert in mathematischer Notation auf das aktuelle Ausgabemedium ausgegeben.

```
FUNCTION DRIVER (EX1,EX2)
     RDS : FALSE,
     WRS :FALSE,
     NEWLINE (2),
     LOOP
          ERR: FALSE,
          BLOCK
               WHEN ECHO (),
                    PRINT ("? "),
                    WHEN NOT RDS AND BELL,
                         PRINT ("^G") EXIT EXIT,
          ENDBLOCK,
          EX1 : FALSE,
          EX1 : PARSE (SCAN(), 0),
          EX2 : SCAN,
          BLOCK
               WHEN ECHO (), NEWLINE (NEWLINE) EXIT,
          ENDBLOCK,
          BLOCK
               WHEN ERR OR NOT TERMINATOR (),
                    SYNTAX (),
                    NEWLINE () EXIT,
               WHEN EX2 = '$,
                    @ : EVAL (EX1),
                    WHEN ECHO (), NEWLINE () EXIT,
               PRINT ("@:"),
               @: EVAL( EX1),
               SPACES (1),
               BLOCK
                    WHEN EX2 = ';,
                         PRTMATH (@, 0, 0, TRUE) EXIT,
                    PRINT (@),
               ENDBLOCK,
               NEWLINE (2),
               NEWLINE (NEWLINE),
          ENDBLOCK,
     ENDLOOP,
ENDFUN;


BELL : TRUE;
FUNCTION ECHO(),
```

```
        NOT RDS OR ECHO,
    ENDFUN;
```

8.14.2 Auswertungsfunktionen

Das erste Element der internen Darstellung einer Funktionsdefinition legt den Typ dieser Funktion fest. Es ist entweder FUNCTION oder SUBROUTINE. FUNCTION zeigt an, dass es sich um eine "call-by-value-Funktion" (CBV) handelt. Beim Aufruf einer CBV-Funktion werden zuerst die Argumente ausgewertet und anschliessend an die Funktion uebergeben. Eine als SUBROUTINE definierte Funktion ist vom Typ call by name (CBN). Derartige Funktionen erhalten ihre Argumente beim Aufruf in unausgewerteter Form.

Das zweite Element der Funktionsdefinition ist entweder ein Name oder eine Liste von Namen, sie definieren die formalen Parameter. Wenn ein von FALSE verschiedener formaler Parameter verwendet wird, so wird die Funktion als no-spread-Funktion aufgefasst. Eine derartige Funktion bekommt ihre Argumente in Form einer an diesen Namen gebundenen Liste. Sie kann folglich beliebig viele Argumente haben.

Ist das zweite Element der Funktionsdefinition hingegen eine Liste von Atomen, so werden die Argumente beim Aufruf an diese einzelnen formalen Parameter gebunden (spread-Funktion).

Die restlichen Elemente der Funktionsdefinition bilden den Funktionskoerper. Dieser stellt eine Liste von Task's dar, die beim Aufruf der Funktion sukzessive abgearbeitet werden. Der Wert des letzten abgearbeiteten Task ist der Wert der Funktion. Die Art der Abarbeitung der Task's haengt von ihrer Struktur ab:

1. Ist der Task ein Atom, so ist sein Wert der Wert dieses Atoms.

2. Ist das FIRST-Element des Task's ein Atom, so wird es als Name einer Funktion aufgefasst, die auf den REST des Task anzuwenden ist.

3. Ist FIRST (FIRST (task)) ein Atom, so wird das erste Element des Task als ein Praedikat betrachtet, das entsprechend 2. ausgewertet wird. Ist der Wert dieses Praedikates FALSE, so hat der Task auch den Wert FALSE. Ist der Wert des Praedikates von FALSE verschieden, so wird der urspruengliche Funktionskoerper uebergangen, und die Auswertung verlaeuft unter Verwendung des RESTes des Task's als neuer Funktionskoerper.

4. In allen anderen Faellen wird der Task rekursiv als ein Funktionskoerper ausgewertet, ehe mit der Auswertung des Top-Level-Funktionskoerpers fortgesetzt wird.

Dieses Auswertungsschema ist von vornherein fuer rekursive Programmstrukturen ausgelegt. Iterative Faehigkeiten koennen aber recht einfach hinzugefuegt werden. So wird ein in LOOP-ENDLOOP eingeschlossener Funktionskoerper wie oben beschrieben ausgewertet, mit dem Unterschied, dass die Auswertung nach dem letzten Task erneut mit dem Anfang des Funktionskoerpers begonnen wird. Dies wird fortgesetzt, bis ein von FALSE verschiedenes Praedikat auftritt.

```
SUBROUTINE ' (X),
     X,
ENDSUB;
```

Der Quotierungsoperator ' unterdrueckt die Auswertung seines Arguments. Er liefert auch einen Mechanismus, um als Listen bzw. gepunktete Paare dargestellte Daten einzulesen.

```
FUNCTION EVAL (X),
     WHEN ATOM (X), FIRST (X) EXIT,
     WHEN NAME (FIRST (X)),
          WHEN UNDEFINED (FIRST (X)),
               WHEN FIRST (X) EQ EVAL (FIRST (X)),
                    EVLIS (X) EXIT,
               EVAL (ADJOIN (EVAL (FIRST (X)), REST (X))) EXIT,
          WHEN CBVP (GETD (FIRST (X))),
               APPLY (FIRST (X), EVLIS (REST (X))) EXIT
          WHEN CBNP (GETD (FIRST (X))),
               APPLY (FIRST (X), REST (X)) EXIT,
          EVLIS (X) EXIT,
     WHEN CBVP (FIRST (X)),
          APPLY (FIRST (X), EVLIS (REST (X))) EXIT,
     WHEN CBNP (FIRST (X)),
          APPLY (FIRST (X), REST (X)) EXIT,
     EVLIS (X),
ENDFUN;
```

EVAL wertet sein Argument unter Verwendung folgender Hilfsfunktionen aus:

```
FUNCTION EVLIS (X),
     WHEN ATOM (X), FALSE EXIT,
     ADJOPIN (EVAL (FIRST (X)), EVLIS (REST (X))),
ENDFUN;


FUNCTION UNDEFINED (X),
     EMPTY (GETD (X)),
ENDFUN;

FUNCTION CBVP (X),
     MACHINEFUNCTION (X) OR DCODEFUNCTION (X),
ENDFUN;

FUNCTION CBNP (X),
     MACHINESUBROUTINE (X) OR DCODESUBROUTINE (X),
ENDFUN;



FUNCTION MACHINEFUNCTION (X),
     Liefere TRUE, falls X auf eine Maschinencodefunktion vom
```

Typ

```
          call-by-value veweist, ansonsten den Wert FALSE,
     ENDFUN;


     FUNCTION MACHINESUBROUTINE (X),
          Liefere TRUE, falls X auf eine Maschinencodefunktion vom
Typ
          call-byname verweist, ansonsten den Wert FALSE,
     ENDFUN;


     FUNCTION DCODEFUNCTION (X),
          FIRST (X) EQ 'EXPR,
     ENDFUN;
     FUNCTION DCODESUBROUTINE (X),
          FIRST (X) EQ 'FEXPR,
     ENDFUN;


FUNCTION APPLY (X< Y),
     WHEN NAME (X),
          WHEN UNDEFINED (X),
               WHEN X EQ EVAL (X), FALSE EXIT,
               APPLY (EVAL (X), Y) EXIT,
          WHEN MACHINEFUNCTION (GETD (X)),
               WHEN ATOM (Y),
                    X (FALSE, FALSE, FALSE) EXIT,
               WHEN ATOM (REST (Y)),
                    X (FIRST (Y), FALSE, FALSE) EXIT,
               WHEN ATOM (RREST (Y)),
                    X (FIRST (X), SECOND (X), FALSE) EXIT,
               X (FIRST (Y), SECOND (Y), THIRD (Y)) EXIT,
          WHEN MACHINESUBROUTINE (GETD (X)),
               X (Y) EXIT,
          WHEN DCODEFUNCTION(GETD(X)) OR DCODESUBROUTINE(GETD(X)),
               BIND (SECOND (GETD (X)), Y),
               Y : EVALBODY (FALSE, RREST (GETD (X))),
               Y EXIT,
          FALSE EXIT,
     WHEN DCODEFUNCTION (X) OR DCODESUBROUTINE (X),
          BIND (SECOND (X), Y),
          Y : EVALBODY (FALSE, RREST (X)),
          UNBIND (SECOND (X)),
          Y EXIT,
     FALSE,
ENDFUN;
```

APPLY (X, Y) wendet die Funktion X auf die Argumentliste Y an. Ist X eine Maschinenroutine, so wird die Steuerung an diese uebergeben. Ist X eine im D-Code definierte Funktion, so werden die formalen Argumente temporaer an die aktuellen Argumente gebunden, der Funktionskoerper wird ausgewertet, anschliessend die urspruenglichen Werte der formalen Argumente wieder rueckgespeichert und der Wert des Funktionskoerpers zurueckgegeben. APPLY benutzt folgende Hilfsfunktionen:

```
FUNCTION EVALBODY (X, Y),
     WHEN ATOM (Y), X EXIT,
     WHEN ATOM (FIRST (Y)) OR ATOM (FIRST (FIRST (Y))),
          EVALBODY (EVAL (FIRST (Y)), REST (Y)) EXIT,
     WHEN ATOM (FIRST (FIRST (FIRST (Y)))),
          X : EVAL (FIRST (FIRST (Y))),
          WHEN NOT X,
               EVALBODY (FALSE, RST (Y)) EXIT,
          EVALBODY (X, REST (FIRST (Y))) EXIT,
     EVALBODY (EVALBODY (X, FIRST (Y)), REST (Y)),
ENDFUN;
```

```
FUNCTION BIND (X, Y),
     WHEN ATOM (Y),
          WHEN ATOM (X), FALSE EXIT,
          PUSH (EVAL (FIRST (X)), ARGSTACK),
          ASSIGN (FIRST (X), FALSE),
          BIND (REST (X), Y) EXIT,
     WHEN ATOM (X), FALSE EXIT,
     PUSH (EVAL (FIRST (X)), ARGSTACK),
     ASSIGN (FIRST (X), FIRST (Y)),
     BIND (REST (X), REST (Y)),
ENDFUN;
```

```
SUBROUTINE LOOP (X1, X2, ... , Xn),
     EVALLOOP (LIST (X1, X2, ... , Xn), LIST (X1, X2, ... ,Xn)),
ENDSUB;
```

LOOP verwendet die Hilfsfunktion EVALLOOP:

```
FUNCTION EVALLOOP (X, Y, Z),
     WHEN ATOM (Y),
          EVALLOOP (X, Y) EXIT,
     WHEN ATOM (FIRST (Y)) OR ATOM (FIRST (FIRST (Y))),
          EVAL (FIRST (Y)),
          EVALLOOP (X, REST (Y)) EXIT,
     WHEN ATOM (FIRST (FIRST (FIRST (Y)))),
          Z : EVAL (FIRST (FIRST (Y))),
          WHEN NOT Z,
               EVALLOOP (X, REST (FIRST (Y))) EXIT,
          EVALBODY (Z, REST (FIRST (Y))) EXIT,
     EVALBODY (FALSE< FIRST (Y)),
     EVALLOOP (X, REST (Y)),
ENDFUN;
```

LOOP wertet sein Argument in der Weise aus, wie auch Funktionskoerper ausgewertet werden. Falls alle Argumente ausgewertet wurden, ohne einen bedingten Ausgang zu erreichen, beginnt die Auswertung wieder von vorn. Der Wert einer LOOP-Konstruktion ist der des letzten ausgewerteten Task.

```
SUBROUTINE COND (X1, X2, ... , Xn),
     EVALCOND (LIST (X1, X2, ... ,Xn)),
ENDSUB;
```

COND verwendet die Hilfsfunktion EVALCOND:

```
FUNCTION EVALCOND (X, Y),
     WHEN ATOM (X), FALSE EXIT,
     Y : EVAL (FIRST (FIRST (X))),
     WHEN NOT Y,
          EVALCOND (REST (X)) EXIT,
     EVALBODY (Y, REST (FIRST (X))),
ENDFUN;
```

8.15 Steuerkonstrukte

Steuerkonstrukte werden in Funktionskoerpern verwendet. Es gibt zwar nur eine geringe Anzahl von diesen, sie sind aber hinreichend ausdrucksstark und flexibel, um einen "natuerlichen" Programmierstil zu realisieren.

8.15.1 Alternativen (WHEN - EXIT)

Die **WHEN - EXIT**- Konstruktion wird zur Auswertung von Praedikaten benutzt und hat die allgemeine Form

```
WHEN praedikat,
     body1 EXIT,
body2
```

Falls das Praedikat zu FALSE ausgewertet wir, so erfolgt Fortsetzung mit body2, andernfalls wird body1 ausgewertet, das Resultat dieser Auswertung ist das Resultat des gesamten Konstrukts. Die Steuerung wird anschliessend an das uebergeordnete Sprachkonstrukt zurueckgegeben. WHEN-EXIT ist nur sinnvoll anwendbar in **FUNCTION**- bzw. **SUBROUTINE** - Definitionen sowie in **LOOP - ENDLOOP**.

8.15.2 Schleifen (LOOP - ENDLOOP)

Die Struktur **LOOP - ENDLOOP** kann in Funktionsdefinitionen zur Bildung iterativer Schleifen verwendet werden. Sie hat die Form

```
LOOP
     task1,
     task2,
     ...
     taskn,
ENDLOOP,
```

Jeder Task wird solange ausgewertet, bis entweder ein von FALSE verschiedenes Praedikat in einer WHEN - EXIT - Konstruktion erreicht wird oder alle Task's ausgewertet wurden. Im ersten Fall ist der Wert

der WHEN - EXIT -Konstruktion auch der Wert von LOOP. Im letzteren Fall beginnt die Auswertung wieder von vorn. LOOP- Konstrukte koennen auch ausserhalb von Funktionsdefinitionen verwendet werden.

LOOP - Konstrukte koennen als Ausdruecke verwendet werden. Sie definieren ein neues Sprachniveau. Der Nutzer muss selbst dafuer sorgen, dass die LOOP - Schleife wieder verlassen wird, da andernfalls eine endlose Schleife entsteht.

LOOP - Konstrukte gestatten auf Grund ihrer Vielseitigkeit die Konstruktion von Abweisschleifen (**while**), von Nichtabweisschleifen (**repeat**) und von Laufanweisungen.

Laufanweisungen der allgemeinen Form

for laufvariable:=anfangswert **step** schrittweite
to endwert **do** task

sind darstellbar als

```
laufvariable:anfangswert;
LOOP WHEN laufvariable > endwert EXIT,
     task,
     laufvariable:laufvariable + schrittweite,
ENDLOOP;
```

Abweisschleifen der Form

while praedikat **do** task

haben dementsprechend die Struktur

```
LOOP WHEN NOT (praedikat) EXIT,
     task,
ENDLOOP;
```

Schliesslich sind **Nichtabweisschleifen** der Form

repeat task1, ... , taskn **until** praedikat

nach dem Schema

```
LOOP
     task1,
     ...
     taskn,
     WHEN praedikat EXIT,
ENDLOOP
```

angebbar.

Es sei darauf hingewiesen, dass hier nur allgemeine Hinweise gegeben werden koennen. Die konkrete Struktur haengt von der zu loesenden Problemstellung ab.

8.15.3 Fallanweisungen (BLOCK - ENDBLOCK)

Die **BLOCK - ENDBLOCK**- Konstruktion liefert eine verallgemeinerte Fallanweisung, sie hat die Form

```
BLOCK
     WHEN ... EXIT,
     ...,
ENDBLOCK,
```

Wie ersichtlich, muss der erste Task in einem Block eine WHEN-EXIT-Konstruktion sein. Die Auswertung der task's in einem Block erfolgt sequentiell. Durch WHEN-EXIT kann die Fortsetzung mit der unmittelbar auf ENDBLOCK folgenden Anweisung erreicht werden. Der Wert des Blocks ist der des letzten ausgewerteten Ausdrucks.

8.16 Speicherverwaltungsfunktionen

```
FUNCTION RECLAIM(),
     Sammle alle Knoten, auf die kein Verweis existiert,
     Gib die Groesse des Freispeichers in Bytes zurueck,
ENDFUN;
```

RECLAIM() bewirkt Garbage Collection. es liefert die Groesse des noch verfuegbaren Speichers in Bytes. Fuer jede Pointerzelle werden 2 Bytes benoetigt. Somit benoetigen Knoten 4 Bytes, Zahlen 6 Bytes, Bezeichner 8 Bytes. Zusaetzlich erfordern Bezeichner und Zahlen Speicher fuer die entsprechenden Printnamen und Zahlenvektoren.

```
FUNCTION SYSTEM(),
     Kompaktiere alle Datenstrukturen,
     Kehre zum Betriebssystem zurueck,
ENDFUN;
```

8.1' Environment-Funktionen

Diese Funktionen werden verwendet, um mySIMP-Umgebungen zu retten bzw. zu laden (SYS-Files). Das Laden von SYS-Files erfogt wesentlich schneller als das Einlesen von Quellfiles.

```
FUNCTION SAVE (X,  Y),
     WHEN NOT EMPTY (WRS),
          Schreibe  das letzte Record von WRS und  schliesse  das
          File,
          WRS : FALSE,
          SAVE (X, Y) EXIT,
     WHEN NAME (X) AND NAME (Y),
          WHEN EMPTY (Y),
               Verdichte alle aktuellen Datenstrukturen,
               Rette einen Speicherabzug der aktuellen Umgebung
                    als  File mit dem Namen X.SYS auf dem aktiven
                    Laufwerk,
               TRUE EXIT,
          Verdichte alle aktuellen Datenstrukturen,
```

```
          Rette  einen Speicherabzug der aktuellen  Umgebung  als
               File mit dem Namen X.SYS auf dem Laufwerk Y,
          TRUE EXIT,
     FALSE,
ENDFUN;


FUNCTION LOAD (X, Y),
     WHEN NAME (X) AND NAME (Y),
          WHEN EMPTY (Y),
               Lade X.SYS vom aktuellen Laufwerk,
               RDS : FALSE,
               Kehre direkt zur DRIVER-Schleife zurueck EXIT,
          Lade X.SYS vom Laufwerk Y,
          RDS : FALSE,
          Kehre direkt zur DRIVER-Schleife zurueck EXIT,
     FALSE,
ENDFUN;
```

9. Verzeichnis der Funktionen und Variablen

Nachfolgend werden die wichtigsten Funktionen, Variablen und Konstanten angegeben, die in MYSIMP und MYMATH verwendet werden. Zu jedem Namen wird angegeben, in welchem File er erklaert wird. Nach Funktionsnamen wird in Klammern die Anzahl der normalerweise verwendeten Argumente angegeben.

=	*	Vergleicher	MYSIMP.COM	8-8
>	*	Vergleicher	MYSIMP.COM	8-10
<	*	Vergleicher	MYSIMP.COM	8-10
'	*	Auswerter	MYSIMP.COM	8-35
:	*	Zuweisung	MYSIMP.COM	8-12
+	*	Addition	MYSIMP.COM	8-21
-	*	Subtraktion	MYSIMP.COM	8-21
*	*	Multiplikation	MYSIMP.COM	8-21
/		Division	MYSIMP.COM	8-22
^		Exponentiation	ARITH.MYS	6-3
!		Fakultaet	ARITH.MYS	6-7
==		Gleichung	EQN.ALG	6-15
\		Matrixdivision	MATRIX.ARR	6-21
`		transponieren	MATRIXARR	6-21
?		Nichtexistenz	LIM.DIF	6-43
#E		Konstante	ARITH.MYS	6-7
#I		Konstante	ARITH.MYS	6-7
#PI		Konstante	ARITH.MYS	6-7

Anmerkung:

Die mit einem **"*"** gekennzeichneten Funktionen von **MYSIMP.COM** sind als Maschinenfunktionen implementiert.
Ferner dienen die zur Beschreibung einiger Funktionen verwendeten Hilfsfunktionen nur zur einfacheren Notation, sie sind nicht aufrufbar.

Hilfsfunktionen:

BIND, CBNP, CBVP, DCODEFUNCTION, DCODESUBROUTINE, EVALBODY, EVALCOND, EVALLOOP, EVLIS, MACHINEFUNCTION, MACHINESUBROUTINE, UNBIND, UNDEFINED

Anhang A: Verzeichnis der nutzungfaehigen Lademoduln

Zur Vereinfachung der Arbeit mit myMATH werden 4 abarbeitbare Versionen als SYS - Files vertriben. Der in diesen bereitgestellte Funktionsumfang duerfte fuer viele Anwendungfaelle, speziell aber fuer das Kennenlernen von myMATH ausreichend sein. Dem Nutzer bleibt bei Verwendung dieser vorgefertigten Systemversionen das zeitaufwendige Erstellen eigener Varianten erspart.

Es ist beabsichtigt, weitere vorgefertigte Systemversionen zur Verfuegung zu stellen. Die gegenwaertig verfuegbaren Versionen tragen die Bezeichnungen

MOD1.SYS
MOD2.SYS
MOD3.SYS
MOD4.SYS

Durch **LOAD(MODx);** werden diese Versionen geladen.

Funktionsumfang:

Die Lademoduln MOD1, MOD2, MOD3, MOD4 enthalten jeweils eine Auswahl der im Kapitel 2 angegebenen Struktur von Quelltextfiles mit den entsprechenden Faehigkeiten (siehe auch Kapitel 6). Im Einzelnen enthalten sie folgende **Komponenten:**

MOD1: **mySIMP.COM**
ARITH.myS
ALGEBRA.ARI

MOD2: **mySIMP.COM**
ARITH.myS
ALGEBRA.ARI
ARRAY.ARI
MATRIX.ARR
EQN.ALG
SOLVE.EQN
TRGPOS.ALG
TRGNEG.ALG
DIF.ALG
LOG.ALG
SIGMA.ALG

MOD3: **mySIMP.COM**
ARITH.myS
ALGEBRA.ARI
EQN.ALG
SOLVE.EQN
TRGPOS.ALG
TRGNEG.ALG
DIF.ALG
INT.DIF
INTMORE.INT
TAYLOR.DIF
LIM.DIF
LOG.ALG
SIGMA.ALG

MOD4: **mySIMP.COM**

ARITH.myS
ALGEBRA.ARI
TRGPOS.ALG
TRGNEG.ALG
DIF.ALG
INT.DIF
INTMORE.INT
TAYLOR.DIF
LIM.DIF
LOG.ALG

In MOD4 hat die Steuervariable **READ** standardmaessig den Wert **FALSE,** so dass bei der Eingabe automatisch alle Kleinbuchstaben in Grossbuchstaben umgewandelt werden.

Ferner wurden zwei weitere Steuervariable **TRINT** und **TRARG** eingefuehrt, die standarmaessig den Wert FALSE haben. Wird **TRINT** ein von FALSE verschiedener Wert zugewiesen, so werden die Namen aller bei aufgerufenen Integratorfunktionen aufgelistet. Ist **TRARG** von FALSE verschieden, so werden ausserdem noch die Argumente dieser Funktionen angegeben.

Anhang B: Hilfsfunktionen

Zur Unterstuetzung der Faehigkeiten von mySIMP werden in einem zusaetlichen File **HILF.myS** weitere Funktionen zur Arbeit mit mySIMP-Daten zur Verfuegung gestellt. Es handelt sich um folgende Funktionen:

VARS(ausdruck) liefert eine Liste der in ausdruck enthaltenen Variablen in der Reihenfolge ihrer internen Ordnung.
Wird VARS auf **OBLIST()** angewendet, so sind damit die gegenwaertig aktiven Variablen anzeigbar.

FKTLIST (ausdruck) liefert eine Liste aller in ausdruck enthaltenen Bezeichner mit funktionaler Bedeutung.

FKTDEF(ausdruck) liefert die Listendarstellung des D-Codes aller in ausdruck enthaltenen Funktionen.

FKTDEF1(ausdruck) hat die gleiche Wirkung wie FKTDEF, liefert aber eine leichter lesbare Darstellung des D-Codes.

DEFAN(funktionsname) liefert eine leicht lesbare Darstellung des D-codes von funktionsname.

ATOMS(ausdruck) liefert eine Liste aller in ausdruck enthaltenen Atome, unabhaengig von ihrer Verschachtelungstiefe.

REMOVE(ausdruck, arg) entfernt ausdruck aus arg.

VOR(element, liste) liefert eine Liste der vor element in liste angeordneten Elemente.

NACH(element, liste) liefert eine Liste der nach element in liste angeordneten Elemente.

VOREIN(arg, element, liste) traegt arg vor element in liste ein.

NACHEIN(arg, element, liste) traegt arg nach element in liste ein.

LAST(liste) liefert das letzte Element von liste.

APPEND(liste1, liste2) vereinigt liste1 und liste2.

ERSETZE(arg1, arg2, liste) ersetzt arg1 durch arg2 in liste, ohne die Argumente zu zerstoeren.

LISTP(arg) liefert den Wert TRUE, wenn arg eine Liste ist (also auch die leere Liste).

TLIST(arg) liefert den Wert TRUE, wenn arg eine "echte", d.h. nichtleere Liste ist.

NTH(n, liste) liefert das n-te Element von liste auf del obersten Niveau.